Tight Space Complexity of the Coin Problem

Mark Braverman* Princeton University
Sumegha Garg† Harvard University
Or Zamir‡ Institute for Advanced Study

Abstract

In the coin problem we are asked to distinguish, with probability at least 2/3, between n *i.i.d.* coins which are heads with probability $\frac{1}{2}+\beta$ from ones which are heads with probability $\frac{1}{2}-\beta$. We are interested in the space complexity of the coin problem, corresponding to the width of a read-once branching program solving the problem.

The coin problem becomes more difficult as β becomes smaller. Statistically, it can be solved whenever $\beta = \Omega(n^{-1/2})$, using counting. It has been previously shown that for $\beta = O(n^{-1/2})$, counting is essentially optimal (equivalently, width $poly(n)$ is necessary [Braverman-Garg-Woodruff *FOCS*'20]). On the other hand, the coin problem only requires $O(\log n)$ width for $\beta > n^{-c}$ for any constant $c > \log_2(\sqrt{5}-1) \approx 0.306$ (following low-width simulation of AND-OR tree of [Valiant *Journal of Algorithms*'84]).

In this paper, we close the gap between the bounds, showing a tight threshold between the values of $\beta = n^{-c}$ where $O(\log n)$ width suffices and the regime where $poly(n)$ width is needed, with a transition at $c = 1/3$. This gives a complete characterization (up to constant factors) of the memory complexity of solving the coin problem, for all values of bias β.

We introduce new techniques in both bounds. For the upper bound, we give a construction based on recursive majority that does not require a memory stack of size $\log n$ bits. For the lower bound, we introduce new combinatorial techniques for analyzing progression of the success probabilities in read-once branching programs.

1 Introduction

The coin problem. Given n independent coins that all have bias of β either towards 1 or -1, the coin problem asks to determine the direction of the bias. The problem becomes harder with decreasing β. As we shall discuss later, it is closely connected to the majority problem. It appears naturally in the context of streaming lower bounds [BGW20], pseudo-randomness [RSV13, BRRY14] and circuit complexity [Aar10, LSS+19]. In the context of read-once branching programs, it was first formally considered in [BV10].

Statistically, the optimal strategy to distinguish would be to output the majority of the n coins. Majority attains a constant advantage when $\beta > n^{-0.5}$, and a vanishing failure probability whenever $\beta \gg n^{-0.5}$. However, majority is not computable by many natural models of computation, such as bounded-width read-once branching programs, $\mathbf{AC}^0$ circuits [Ajt83, FSS84] and $\mathbf{AC}^0[\oplus]$ circuits [Raz87, Smo87]. In this paper, we focus on solving the coin problem using read-once branching programs (ROBPs). The main complexity measure of such programs is width — corresponding to the number of different states a program can be in after reading t bits of the input. Informally, a width-w ROBP corresponds to a streaming algorithm with $\log w$ bits of memory.

Prior results. Motivated by derandomization, [BV10, Ste13, CGR14] study the smallest bias β such that width-w length-n ROBPs can distinguish between the two cases. The works [BV10, Ste13] show that a

*Email: mbraverm@cs.princeton.edu. Research supported in part by the NSF Alan T. Waterman Award, Grant No. 1933331, a Packard Fellowship in Science and Engineering, and the Simons Collaboration on Algorithms and Geometry.

†Email: sumegha.garg@gmail.com. Research supported by Michael O. Rabin Postdoctoral Fellowship.

‡Email: orzamir@ias.edu. Research supported in part by NSF grant No. CCF-1900460.

width-w length-n ROBP cannot solve the coin problem for $\beta < c/(2\log n)^{w-2}$ (for a small enough constant $c > 0$), which gives a width-$\Omega\left(\frac{\log n}{\log\log n}\right)$ lower bound for solving the coin problem for any $\beta = n^{-\Theta(1)}$. Works of [MNSW98, KNW10] imply hardness of computing majority using $n^{\Omega(1)}$-width ROBPs, which are correct on every input with high probability, but such results don't imply hardness of solving the coin problem. Recent work of [BGW20] shows that any ROBP which, on n *i.i.d.* uniform input bits, outputs majority with high advantage (over the uniform distribution), requires $n^{\Omega(1)}$ width. This result implies that solving the coin problem (up to statistical distance between the two distributions) for $\beta < cn^{-\frac{1}{2}}$ (for a small enough constant $c > 0$) requires $n^{\Omega(1)}$ width, improving on the previously known $\Omega\left(\frac{\log n}{\log\log n}\right)$ width lower bound for smaller values of β (see Appendix B for the details).

Next, focusing on upper bounds, [BV10], using the circuit construction for approximating majority by [Ama09, ABO84], shows that for constant width ROBPs, the lower bound by [Ste13] is tight. That is, [BV10] constructs a width-w (for constant w) ROBP that solves the coin problem for $\beta = O((\log n)^{2-w})$. A recursive OR-AND $((x_1 \wedge x_2) \vee (x_3 \wedge x_4))$ tree of depth $\log_4 n$ can detect a bias of $\Omega(n^{-\log_2(\sqrt{5}-1)}) \approx \Omega(n^{-0.306})$ (more on this in Section 2.1). In other words, if the input is *i.i.d.* $\mathrm{Ber}(1/2+\delta)$, then the output of a recursive OR-AND tree is $\sim \mathrm{Ber}(1/2 + n^{0.306} \cdot \delta)$. Valiant [Val84] used such a recursive OR-AND tree to compute majority. In fact, [Bop85, DZ97] showed that this is the best bias-amplification a read-once boolean formula can achieve. A read-once boolean formula of depth d can be executed by a ROBP of width equal to $(d+1)$ (instead of the brute force 2^d-width recursive implementation). Thus, [Val84] implies a length-n, $O(\log n)$-width, ROBP that distinguishes the coin problem for bias greater than $n^{-0.306}$ — we can already detect a very small bias with memory much better than counting. However, it turns out that $n^{-0.306}$ is the smallest bias read-once boolean formulas can detect [Bop85, DZ97]. A natural question is whether this limitation extends to $O(\log n)$-width ROBP.

Previous and new results for various β's are summarized in the table below. Prior to our work there was an exponential gap between the upper bound $n^{O(1)}$ and the lower bound $\Omega(\log n)$ on the width of a ROBP for solving the coin problem for values of β between $n^{-0.306}$ and $n^{-0.5}$. It was strongly suspected that the threshold at which the required width becomes polynomial is at one of the endpoints of this interval: either the upper bound one gets from OR-AND trees is tight, and improving on it requires polynomial width — or, $\beta \sim n^{-0.5}$ is a singularly difficult case, and bias $n^{-0.5+\varepsilon}$ can be detected using width $O(\log n)$. It turns out that neither is the case. Surprisingly, the transition from logarithmic to polynomial width happens around $\beta = n^{-1/3}$:

Range of β	Upper bound	lower bound
$\beta > (\log n)^{-c}$	$w = O(1)$ [BV10]	$w = \Omega(1)$ (trivial)
$n^{-0.306} \ll \beta \ll \frac{1}{\log n}$	$w = O(\log(1/\beta))$ (Similar to [BV10, Val84])	$w = \Omega\left(\frac{\log(1/\beta)}{\log\log n}\right)$ [BV10, Ste13]
$\beta \sim n^{-0.5}$	$w = n^{O(1)}$ (counting)	$w = n^{\Omega(1)}$ [BGW20]
$n^{-1/3+\varepsilon} < \beta < n^{-0.306}$	$\mathbf{w = O(\log n)}$ **[New]**	$w = \Omega\left(\frac{\log n}{\log\log n}\right)$ [BV10, Ste13]
$\beta < n^{-1/3-\varepsilon}$	$w = n^{O(1)}$ (counting)	$\mathbf{w = n^{\Omega(1)}}$ **[New]**

An equivalent[1] formulation of bounded-width ROBPs is the model of computation in which the input is given bit-by-bit and we are limited to store only M bits of memory after processing each bit, yet, we are given an access to a clock that is incremented with every given input bit. In [HC70] it is shown that a m-state automaton[2] (time-invariant function) cannot distinguish coins with bias $\beta < \frac{c}{m}$ (for a small enough constant $c > 0$). Hence, without clock access only $\beta = \Omega(1)$ can be distinguished with constant-sized memory $M = O(1)$. Nevertheless, the tribes function $\mathrm{Tribes}\,(x_{i,j}) := \vee_{i=1}^{s} \wedge_{j=1}^{w} x_{i,j}$ can be easily computed with $M = O(1)$ ([BV10]) for any w, s, when $(x_{i,j})$ are the $n = s \cdot w$ input bits (where $x_{i,j}$ is the $((i-1)w + j)$th input bit). For $w \approx \log n$, $s \approx \frac{n}{\log n}$ the value of the Tribes function can distinguish coins with bias $\beta = \frac{1}{\log n - \log\log n + O(1)}$. Hence, with clock-access $M = O(1)$ memory suffices for $\beta \gg \frac{1}{\log n}$. This

[1] up to taking a logarithm, i.e., $w = 2^M$.

[2] compare m with 2^M.

illustrates the importance of using the clock and like [BV10], in constructing our upper bound, we crucially use the time-step of a read-once branching program.

New upper bound. As noted before, any depth-d read-once formula for the coin problem can be translated into a width-$O(d)$ ROBP (and thus into a streaming with clock automaton with $\sim \log d$ bits of memory). Using the boolean AND-OR tree [Val84], one gets that $O(\log n)$-width suffices when $\beta > n^{-\log_2(\sqrt{5}-1)} \approx n^{-0.306}$, but this cannot be improved for read-once boolean formulas. We show that the $O(\log n)$-width upper bound can be extended to $\beta > n^{-1/3+\varepsilon}$ for any $\varepsilon > 0$. This is interesting for two reasons: (1) Previous upper bounds for the coin problem relied on the connection between read-once formula depth and ROBP upper bounds. We cannot rely on this connection due to known limitations on read-once boolean formulas. Instead, we expand the type of recursive programs that can be executed with very limited memory. This shows a gap between the strength of read-once $\mathbf{NC}^1$ formulas and $O(\log n)$-width read-once branching programs. (2) The upper bound is tight: the memory complexity jumps exponentially once β drops below $n^{-1/3}$.

New lower bound. We show that whenever $\beta < n^{-1/3-\varepsilon}$, the memory M needed to solve the coin problem increases from $O(\log \log n)$ to $\Omega(\log n)$ — this is matched by a trivial upper bound attained by counting the number of heads. Previously, such lower bound was only known for $\beta \sim n^{-0.5}$ — proved in [BGW20] using information complexity techniques[3]. The fact that $\Theta(\log n)$ memory is required to detect bias as high as $n^{-1/3}$ is surprising, because the information complexity of detecting such bias[4] (e.g. as measured by the amount of information about the coin tosses that the automaton needs to store) is $o(\log n)$. Indeed, our proof introduces combinatorial techniques to obtain the lower bound. These combinatorial techniques are likely to be useful in other ROBP contexts where information-theoretic techniques cannot work.

Main theorems.

Theorem 1.1 (Informal statement of Theorem 4.12)**.** *For all constant $\epsilon > 0$, there exists a length-n width-$O(\log n)$ ROBP that solves the coin problem for bias $\beta \geq n^{-\frac{1}{3}+\epsilon}$.*

Theorem 1.2 (Corollary of 5.16)**.** *For all constant $\epsilon > 0$, any length-n ROBP that solves the coin problem for bias $\beta \leq n^{-\frac{1}{3}-\epsilon}$, requires $n^{\Omega(\epsilon)}$ width.*

Other Related Work The coin problem has been studied in various computational models, such as $\mathbf{AC}^0$ circuits [SV10, Aar10, CGR14], $\mathbf{AC}^0[\oplus]$ [LSS+19], product tests [LV18] and regular ROBPs [BRRY14, RSV13]. Lower bounds on coin problem have applications in the area of pseudorandomness and circuit lower bounds. [BRRY14] used the fact that regular width-w length-n ROBPs cannot distinguish coins with bias $\beta < c/w$ (for small enough constant $c > 0$) to improve the state-of-art construction of pseudorandom generators for regular ROBPs. [LSS+19] used the coin problem for constructing functions that proved a fixed-depth size-hierarchy theorem for uniform $\mathbf{AC}^0[\oplus]$.

2 Proof Overview

In this section, we give a proof outline for our tight bounds for solving the coin problem on low-width read-once branching programs (see Section 3 for the formal definition). Before we dive into the bounds for ROBPs, we prove tight bounds for a model of computation we call **Skip-Forward Read-Once Program** (SF-ROP). A size-ℓ SF-ROP P on n-bit input is defined as a sequence of ℓ states $S_1, S_2, \ldots, S_\ell$ (S_1 represents the start state and $S_{\ell-1} = t_0$ and $S_\ell = t_1$ represent the two terminal states). Each non-terminal state s

[3]Note that $n^{-0.5}$ is the smallest value of β for which the bias is statistically detectable from n samples.

[4]Consider the following streaming algorithm: given a sequence of coin tosses $x_1, x_2, \ldots, x_n$, keep track of $\sum y_i$, where $y_i = x_i$ with probability $\frac{1}{2} + O\left(\frac{1}{n^{1/6}}\right)$. This algorithm detects bias of $n^{-1/3}$ but has low information complexity according to the definition in [BGW20].

has an input bit $x_{i(s)}$ associated with it, which it reads to jump to the next state. Let $P(s,0)$ (and $P(s,1)$) represent the states that P transitions to, from state s, when $x_{i(s)}$ is 0 (and 1). We require that these states are ahead of s in the sequence and that the input bits associated with $P(s,0)$ and $P(s,1)$ have strictly higher indexes than $i(s)$. We also require that each execution on input x reaches one of the terminal states. Each input bit of x is read once as the input indexes that the program reads keep increasing at every state transition. SF-ROP is called "skip forward" as the program skips some states to transition to the next state in sequence.

In principle, SF-ROPs are more powerful than ROBPs but in our context, they turn out to be the "right" abstraction. We prove that to solve the coin problem with bias β (coins with bias towards 1 ends at t_1 with probability $> 2/3$, and ones with bias towards 0 end at t_0 with probability $> 2/3$), the size complexity of a SF-ROP is $\Theta(\beta^{-3})$ (both upper and lower bound). In particular, this readily implies our lower bound when $\beta < n^{-1/3}$ as a length-w width-n ROBP can be seen as a size-$(w \cdot n)$ SF-ROP.

In the opposite direction, we use the upper bound on SF-ROP solving the coin problem to get a "special" constant-sized ROBP, which amplifies the bias and can be recursed on, without blowing up the width exponentially.

2.1 Overview of the Upper Bound

We show that one can detect a very small bias using a read-once branching program with memory requirements that are exponentially smaller than what is needed to count and take the majority (which is the optimal strategy). Intuitively, the layers of the program — the "time dimension" — ends up substituting for memory. As an instructive example, let us first consider the alternating two-input OR-AND tree of depth $d = \log_4 n$ ([Val84] used such a tree to compute majority). The function $F(x_1, x_2, x_3, x_4) := (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$ maps *i.i.d.* inputs $\mathrm{Ber}(q)$ into $\mathrm{Ber}(f(q))$. The function f has one fixed point at $p = \frac{\sqrt{5}-1}{2}$, where $f(p) = p$. If the inputs are distributed according to $\mathrm{Ber}(p)$, then the output is distributed according to $\mathrm{Ber}(p)$ (fixed point can be shifted to $1/2$ by randomly adding 1s to the input stream). If the inputs are distributed according to $\mathrm{Ber}(p+\varepsilon)$, then the output is distributed according to $\mathrm{Ber}(p + (\sqrt{5}-1)^2\varepsilon + O(\varepsilon^2))$. Thus β will get amplified (approximately) to

$$(\sqrt{5}-1)^{2d} \cdot \beta = (\sqrt{5}-1)^{\log_2 n} \cdot \beta = n^{\log_2(\sqrt{5}-1)} \cdot \beta,$$

which is constant as long as $\beta = \Omega(n^{-\log_2(\sqrt{5}-1)}) \approx \Omega(n^{-0.306})$. As noted in previous papers (for example, [CSV15]), a depth d read-once boolean formula on n bits can be executed by a length-n width-$(d+1)$ read-once branching program, which gives a $O(\log n)$-width ROBP to solve the coin problem for bias greater than $\Omega(n^{-0.306})$. [Bop85, DZ97] showed that this is the best a read-once boolean formula can detect. We improve the bias detected by read-once branching programs to $\sim n^{-1/3}$ (using a construction based on the majority function) without going through read-once boolean formulas, but before that let's look at low-width simulation of d-depth read-once boolean formulas for intuition.

ROBP for read-once boolean formulas. Consider a read-once boolean formula F (with AND/OR gates at non-leaf nodes) on variables $x_1, \ldots, x_n$, where the variables appear in F in this order. To convert the computation into a branching program format, it is instructive to consider the following question: suppose we are given $x_{i+1}, \ldots, x_n$; what information about $x_1, \ldots, x_i$ do we need to compute $F(x_1, \ldots, x_n)$?

We can view the evaluation of F as a recursive execution, where if $F = \vee_{j=1}^{k} F_j$, the subformulas F_j are executed in order, and when F_j is being executed, the stack contains the value of $F_1 \vee \ldots \vee F_{j-1}$. Therefore, to complete the computation of F given $x_{i+1}, \ldots, x_n$, all we need to know is the state of the stack of the computation of F at time i — that is, after observing $x_1, \ldots, x_i$.

Each layer of the recursive evaluation of a formula with AND and OR gates is evaluating a single AND/OR function, and thus requires 1 bit of state. Therefore, to evaluate a depth d formula, the stack appears to require d bits of storage, corresponding to 2^d different possible states and giving a width-2^d ROBP to evaluate a read-once boolean formula. However, one can do better by noting that if $F = F_1 \vee F_2$, and F_1 has evaluated to 1 (that is, the first bit of the stack is 1), then the output of the executing of F_2 will not affect the output

of the calculation (which will be 1). This observation holds in general: for each layer of the recursion, one value on the stack means that we care about the output of the child being computed, and the other value (0 for the AND gate and 1 for an OR gate) means that we do not care. Whenever we don't care, the execution is unaffected if we modify (or forget) the portions of the stack corresponding to the evaluation of that child.

To save space, let us replace values on the stack that cannot affect evaluation of the function with 0's. This means that for each $i \in [n]$, there are only $d+1$ possible states of the stack (represented by a position $t \in [0, d]$), such that the stack till t stores values which may affect the evaluation of F (for every layer of recursion, it's type (AND or OR) fixes this value), followed by 0's. This gives a length-n width-$(d+1)$ read-once branching program evaluating read-once formulas.

Thus, for an AND/OR formula, recursion doesn't blow up the width exponentially. What helps an AND/OR gate is the fact that when we learn the value of a AND/OR gate G, we "suspend" or "fast forward" the computation on the remaining inputs of G, since these cannot affect the ultimate outcome of the computation. Hence, we open the recursion on a child for only a singe value of the stack till now. Unfortunately, this property is unique to the AND/OR gates. Given an AND(G_1, G_2, G_3) gate (with G_1, G_2, G_3 appearing in this order), saying "we care about the value of G_2" implies that G_1 evaluated to 1. On the other hand, for a majority gate $MAJ(G_1, G_2, G_3)$, the fact that G_2 is influential does not determine the value of G_1, and thus the recursive execution will open G_2 for multiple values of the stack, increasing the number of states to $2^{\Omega(d)}$.

Our construction. The bias amplifying gadget in the read-once formula was an OR-AND gate which, as discussed above, could be recursed on without blowing up the width exponentially. Our bias amplifying read-once branching program (ROBP) is based on the majority function and has the property that saying "we care about the value of the $(i+1)$th input bit" fixes the state of the program in the ith layer. Such a property is true for ROBPs with **effective width** 1. A ROBP has an **effective width** of w' if every layer has at most w' non-dormant nodes, which are nodes that transition to different states, in the next layer, depending on whether the next bit is 0 or 1. In other words, a node in ith layer is **dormant** if it doesn't look at the $(i+1)$th input bit while transitioning to the next layer. ROBPs with **effective width** 1 can be recursed on without blowing the width exponentially. This is because, we open the recursion on a child only for a single state in every layer and hence the width increases linearly. Thus, if we show a length-c_1 width-c_2 ROBP (with **effective width** 1) that amplifies the bias β to $a \cdot \beta$ (where c_1 and c_2 are constants), then we can recurse up to $\log_{c_1} n$ levels to get a $O(\log n)$-width ROBP, which detects a bias of $n^{-\log_{c_1} a}$.

We first construct a length-m width-$\sqrt{m}$ ROBP B_1 that amplifies the bias of each input bit (β) to $\sim \sqrt{m}\beta$ (outcome has bias of $\sqrt{m}\beta$). B_1 keeps track of the count, stops whenever the count reaches $\pm\sqrt{m}$ and outputs the sign of the count. B_1 can be viewed as an approximation to the majority function[5]. As every ROBP corresponds to a SF-ROP when the states are sequenced in increasing order of their layers, B_1 gives a size-$(m^{3/2})$ SF-ROP P on m-bit input that amplifies the bias of the input bits (β) to $\sim \sqrt{m}\beta$.

Next, we just look at the skip-forward read-once program P and convert it into a ROBP B with **effective width** equal to 1. Let $S_1, S_2, \ldots, S_{m^{3/2}}$ be the states in the sequence of the program P. The nodes in the ith layer of B are indexed by $\{S_j\}_{j \in [m^{3/2}]}$, where the node corresponding to S_i is 'Active' and the nodes corresponding to states $S_j | j \neq i$ are of the FF $[S_j]$ type (short for 'Fast-Forward' to S_j). In other words, ith layer of B corresponds to the state S_i and $(i+1)$th input bit is read or cared about iff B reaches state S_i – 'Active' node in the ith layer. To execute P, B reads the next input bit whenever it reaches an 'Active' state (say in the ith layer) and uses 'Fast-Forward' nodes to transition to $P(S_i, x_{i+1})$ (FF $[S_j]$ skips the input bits till it reaches the jth layer).

Thus, B is a $m^{3/2}$-length $m^{3/2}$-width ROBP with **effective width** 1, which imitates the execution of P using a $m^{3/2}$-length input instead of a m-length input[6]. The question is – how does the output distribution of B compare to that of SF-ROP P when the input bits are *i.i.d.* Ber(p) for both programs. We observe

[5]Truncating the count at m_1 would have amplified the bias β to $\min\{m_1, \sqrt{m}\} \cdot \beta$; thus, $m_1 \approx \sqrt{m}$ gives the optimal construction.

[6]A more careful conversion will give $m^{3/2}$-length $O(m^{1/2})$-width ROBP with **effective width** 1, however this does not affect the resulting performance (which is a function of length), and the width-$m^{3/2}$ reduction is more straightforward to describe.

that the output distribution is the same, as for P – the bit read at a state s is independently drawn from $\mathrm{Ber}(p)$ conditioned on reaching s and the transition probabilities will remain unchanged when each state is given it's unique bit as in B. Therefore, B is a $m^{3/2}$-length $m^{3/2}$-width ROBP with **effective width** 1, which amplifies the input bias β to output bias of $\sim \sqrt{m}\beta$. Recursion up to $\log_{m^{3/2}} n$ levels gives us a $O(\log n)$-width ROBP, which detects a bias of $n^{-\log_{m^{3/2}} \sqrt{m}} \approx n^{-1/3}$.

2.2 Overview of the Lower Bound

Attaching posterior probabilities to states and transitions. The key observation that makes our lower bound possible is that at any point of the execution of the SF-ROP the 'knowledge' of the program about the answer (the coin's bias) can be summarized using a single number: the current estimate $p(S) \in [0,1]$ on the probability that the coin is biased towards 1, assuming the initial bias has been chosen to be $+\beta$ or $-\beta$ with an equal probability. Thus, in the beginning of the program's execution $p(S_1) = 1/2$, while the success requirement implies that $p(t_1) > 2/3$ and $p(t_0) < 1/3$.

Probabilities $p(s)$ can be attached to transitions between states and not just to states themselves. For a transition $s \to s_0$ the corresponding $p(s \to s_0)$ is the probability that the coin is biased towards 1 conditioned on the program finding itself in the $s \to s_0$ transition.

How posterior probabilities evolve throughout the program. Each time a state s queries a bit $x_{i(s)}$, the result is either 0, in which case it transitions to $s_0 := P(s,0)$, or 1, in which case it transitions to $s_1 := P(s,1)$. It is important to note that there is a simple connection between $p(s)$ and $p(s \to s_0)$ and $p(s \to s_1)$, given by Bayes rule. For example,

$$p(s \to s_1) = \Pr[\text{Bias='+'}|\text{reach } s \to s_1] = \frac{\Pr[\text{reach } s \to s_1|\text{Bias='+'}] \cdot 0.5}{\Pr[\text{reach } s \to s_1]} = \frac{\Pr[\text{reach } s|\text{Bias='+'}] \cdot 0.5 \cdot (0.5+\beta)}{\Pr[\text{reach } s] \cdot (p(s) \cdot (0.5+\beta) + (1-p(s)) \cdot (0.5-\beta))} = \frac{p(s) \cdot (0.5+\beta)}{0.5 + 2 \cdot p(s) \cdot \beta - \beta}.$$

The only fact we need to know about the above equation is that $|p(s) - p(s \to s_1)| < O(\beta)$: reading one input bit cannot change our belief about the answer by too much.

When a number of transitions $S_1 \to S$, ..., $S_k \to S$ merge into a single state S, their conditional probabilities of positive bias also merge in a predictable way into the weighted average of the conditional probabilities:

$$p(S) = \frac{\Pr[\text{reach } S_1 \to S] \cdot p(S_1 \to S) + \ldots + \Pr[\text{reach } S_k \to S] \cdot p(S_k \to S)}{\Pr[\text{reach } S_1 \to S] + \ldots + \Pr[\text{reach } S_k \to S]} \tag{1}$$

Split-and-merge game. Sweep the SF-ROP from first to last state. During step t consider all the transitions of the form $S_i \to S_j$ that cross the $(t-1)-t$ boundary between $\{S_1, \ldots, S_{t-1}\}$ and $\{S_t, \ldots, S_\ell\}$. That is, such that $i < t \le j$. For each such transition $S_i \to S_j$, let $w(S_i \to S_j)$ be the probability that the program reaches that transition.

We can map the transitions through the $(t-1)-t$ boundary to distributions X_t of masses on the interval $[0,1]$, where for each transition $S_i \to S_j$ with $i < t \le j$ we put mass $w(S_i \to S_j)$ at location $p(S_i \to S_j)$. Note that $w(S_i \to S_j)$ of transitions that cross the $(t-1)-t$ boundary adds up to 1, because every program execution crosses that boundary exactly once. Therefore, each X_t is a distribution. To streamline the argument, we assume that the terminal states are repeated twice among the last four states: $S_{\ell-3} = t_0$, $S_{\ell-2} = t_1$, and there is a probability 1 transition from $S_0 \to S_1$, $S_{\ell-3} \to S_{\ell-1}$ and $S_{\ell-2} \to S_\ell$ (S_0 is an additional start state).

Initially, X_1 is an atomic mass at $1/2$. At the end of the execution, by the success condition, X_ℓ will be a point mass outside the $(1/3, 2/3)$ interval. For each t, the transition from X_t to X_{t+1} involves transitions entering state S_t being replaced with transitions leaving state S_t. These can be broken down to two steps: (1) **Merge:** masses corresponding to transitions entering S_t are merged into their center of mass $p(S_t)$

according to equation (1); (2) **Split:** the mass at $p(S_t)$ is split into two masses corresponding to the value of $x_{i(S_t)}$. The two resulting masses are in the interval $(p(S_t)-\beta, p(S_t)+\beta))$, and their center of mass is still $p(S_t)$. Intuitively, as our goal is to push as much mass away from 1/2 as possible, split steps are working in our favor, while merge steps work against us.

In our lower bound proof, we show that at least $\Omega(\beta^{-3})$ split moves are needed to move the mass away from the middle. The overarching technique is related to the lower bound proof using pebbles in [BRRY14]. Interestingly, in our case here we do not believe there is a direct analytical argument lower bounding the number of split steps. For example, a second-moment argument only gives an $\Omega(\beta^{-2})$ lower bound (roughly corresponding to bias $\beta = n^{-0.5}$). The second moment of X_0 is 0, and the second moment of X_ℓ is constant. Since each split step is confined into an interval of length 2β, splitting a mass m increases second moment by $O(m \cdot \beta^2)$, giving a β^{-2} lower bound. The extra β^{-1} factor requires an additional argument — showing that the mass is spread across the interval, and thus a typical split step will only involve $\sim \beta$ mass, leading to only a β^3 increase in the second moment. We do not know if a direct analytic proof (involving an energy function) exists — we give a combinatorial argument.

The discrete split-only game, and a combinatorial lower bound. After some transformations, the split-and-merge game described in the previous paragraph can be transformed into the following *discrete-state, split-only* game. At step t, Y^t is a distribution on states $\{-k, -k+1, \ldots, 0, 1, \ldots, k\}$, where $k = \Theta(\beta^{-1})$. A split step on location i, takes the mass Y_i^t and splits it evenly between locations $i+1$ and $i-1$:

$$Y_i^{t+1} = 0;\ Y_{i-1}^{t+1} = Y_{i-1}^t + Y_i^t/2;\ Y_{i+1}^{t+1} = Y_{i+1}^t + Y_i^t/2.$$

Initially, $Y_0^1 = 1$ — all the mass is at location 0. Our goal is to move at least half the mass outside of the $[-k/2, k/2]$ interval. We would like to accomplish this goal using as few split steps as possible. It is not hard to see that this can be done using $O(k^3)$ split steps. We give a combinatorial argument showing that this is in fact tight. We sketch it here.

Imagine that the masses are made of play dough. Embed $n = \delta \cdot k$ (for a small constant δ) grains of sparkling sand into the play dough, and observe them throughout a typical trajectory of the split game. Let random variable Q_i denote the number of times particle i experiences a split operation, and let Q_{ij} be the number of times particles i and j experience a split operation together. By an inclusion-exclusion argument we have

$$\text{Total number of splits} \geq \sum_{i=1..n} Q_i - \sum_{i,j=1..n} Q_{ij}. \tag{2}$$

Each individual grain of sand follows an unbiased random walk. Since it leaves the $[-k/2, k/2]$ interval with probability greater than half[7], it must be the case that $\mathbb{E}[Q_i] = \Omega(k^2)$. A slightly more involved calculation shows that two particles, each following a random walk, meet at most $O(k)$ times in expectation: $\mathbb{E}[Q_{ij}] = O(k)$. Therefore, using (2), we get:

$$\mathbb{E}[\text{Total number of splits}] \geq \Omega(n \cdot k^2) - O(n^2 k) = k^3 \cdot (\Omega(\delta) - O(\delta^2)) = \Omega(k^3),$$

for an appropriately chosen small constant $\delta > 0$.

2.3 Outline of the paper

In Section 3, we establish certain notations, formally define read-once branching programs (ROBP), skip-forward read-once programs (SF-ROP) and related notions needed for our construction. In Section 4, we first establish the bias-amplifying properties of a majority-like gadget of constant size and construct a read-once branching program with **effective width** 1, which executes the gadget by encoding states in a time-step (or location in the topological order). Then, we define taking tensor of two branching programs (for recursive execution) and prove the upper bound for coin problem with bias greater than $n^{-1/3+\Omega(1)}$. In Section 5, we prove the lower bound for SF-ROPs solving the coin problem, which instantly gives a lower bound on the width of ROBPs for bias less than $n^{-1/3-\Omega(1)}$.

[7] Assuming each split divides the sand grains randomly between left and right.

3 Preliminaries

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. Given $x \in \{-1, 1\}^n$, we represent the projection of x on coordinates from a to b as $x[a : b]$, that is, $x[a : b] = (x_a, x_{a+1}, \ldots, x_b)$. We will interchangeably represent the set of bits by $\{0, 1\}$ or $\{-1, 1\}$ depending on the context. Let $\mathrm{Ber}(p)$ ($p \in [0, 1]$) represent the Bernoulli distribution, that puts weight p on 1 and $1-p$ on -1 (or 0). Given a distribution $D : \mathcal{X} \to [0, 1]$, let $x \sim D$ represent that x is drawn from D. Given a k-length sequence $x = (x_1, \ldots, x_k) \in \{-1, 1\}^k$, let $\mathrm{flip}(x)$ represent the sequence $(-x_1, \ldots, -x_k)$. Let $\mathbb{R}$ represent the set of real numbers. The following inequalities would be useful in the paper:

1. $\forall x \in \mathbb{R}, 1 - x \leq e^{-x}$.
2. $\forall x \in [0, 1]$, $e^{-x} \leq 1 - \frac{x}{2}$.

The following Stirling's approximation [Rob55] for factorials would be useful in the paper:

$$\sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n} \leq n! \leq e n^{n+\frac{1}{2}} e^{-n}$$

Definition 3.1. *A read-once branching program*[8] *(ROBP) of length-n and width-w is an acyclic directed graph whose nodes, called states, are partitioned into n layers, where each layer consists of at most w nodes/states. There is an additional "start" state at the start of the program. The last layer consists of 2 states called "accept" (or 1) and "reject" (or -1). From every state but for the latter two, there are two outgoing edges, labeled by -1 and 1, to the states in the following layer. On input $x \in \{-1, 1\}^n$, the computation reads the bits of x (ith bit at the $(i-1)$th layer) and traverses the branching program, starting from the "start" state, by following the edges according to the labels given by the bits of x. The string x is accepted by the program if the computation ends in the "accept" state.*

In the paper, we will also consider ROBPs that output a state, instead of "accept" or "reject". Given a length-n and width-w ROBP B, let $\mathcal{L}_i^B$ represent the set of nodes/states in the ith layer ($i \in \{0, 1, \ldots, n\}$), where $\mathcal{L}_0^B$ just contains the "start" state, that is, s_B. We represent the state transition from $(i-1)$th layer to ith layer using the function $B_i : \mathcal{L}_{i-1}^B \times \{-1, 1\} \to \mathcal{L}_i^B$ ($i \in [n]$), that is, the edge labeled $b \in \{-1, 1\}$, from a state s in $(i-1)$th layer, goes to the state $B_i(s, b)$ in the ith layer. Given an n-bit input $x \in \{-1, 1\}^n$, $B(x) \in \mathcal{L}_n^B$ represents the output of B on x, that is, the state reached in after traversing B along edges with labels corresponding to bits of x. $B(x)$ is in $\{-1, 1\}$, for ROBPs with last layer consisting of only "accept" and "reject".

The goal of the coin problem is to distinguish between n independent coins *i.i.d* from $\mathrm{Ber}\left(\frac{1}{2} - \delta\right)$ and n independent coins *i.i.d.* from $\mathrm{Ber}\left(\frac{1}{2} + \delta\right)$.

Definition 3.2. *We say a length-n and width-w ROBP B solves the coin problem for parameter/bias δ with an advantage p if*

$$\Pr_{\forall i \in [n], x_i \sim Ber\left(\frac{1}{2}+\delta\right)}[B(x) = 1] - \Pr_{\forall i \in [n], x_i \sim Ber\left(\frac{1}{2}-\delta\right)}[B(x) = 1] \geq p \tag{3}$$

$$\Pr_{\forall i \in [n], x_i \sim Ber\left(\frac{1}{2}-\delta\right)}[B(x) = -1] - \Pr_{\forall i \in [n], x_i \sim Ber\left(\frac{1}{2}+\delta\right)}[B(x) = -1] \geq p \tag{4}$$

Next, we define certain operations on the ROBPs that would make the exposition of the $O(\log n)$-width ROBP for solving the coin problem easier. Given a read-once branching program B outputting a state and a mapping $O_B : \mathcal{L}_n^B \to \{-1, 1\}$, we convert B into a ROBP that outputs -1 or 1 using the following merge operation.

Definition 3.3. *Let B be a length-n, width-w ROBP and $O_B : \mathcal{L}_n^B \to \{-1, 1\}$ be a mapping that assigns each output state of B a -1 or 1. Merge(B, O_B) outputs a length-n, width-w ROBP read-once branching program B' such that $\forall i \in [n-1]$, $B'_i = B_i$ and $B'_n = O_B(B_n)$. Hence, $\forall x \in \{-1, 1\}^n, B'(x) = O_B(B(x))$.*

[8] In this paper, we consider branching programs where each node in the ith layer reads the $(i+1)$th bit. This model has also been known as ordered and oblivious read-once branching program.

Let B be a length-n, width-w ROBP outputting 1 or -1, and B' be a length-n', width-w' ROBP outputting 1 or -1. In the paper, we would be interested in constructing a length-$n \cdot n'$ ROBP $B'' = B \otimes B'$ such that on a $(n \cdot n')$-bit input $x \in \{1, -1\}^{n \cdot n'}$, $B''(x) = B(B'(x[1 : n']), B'(x[n' + 1 : 2n']), \ldots, B'(x[((n - 1) \cdot n') + 1 : n \cdot n']))$. To compute the width of such a B, we would be interested in the following definitions. We will formally define $B \otimes B'$ in Section 4.3.

Definition 3.4. *Let B be a length-n, width-w ROBP outputting 1 or -1. A node s in the ith layer of B ($i \in \{0, 1, \ldots, n-1\}$) is called **dormant** if both the edges from s go to the same node in the next layer, that is,*

$$B_{i+1}(s, 1) = B_{i+1}(s, -1).$$

Let $\mathcal{D}_i^B = \{s \in \mathcal{L}_i^B \mid s \text{ is } \textbf{dormant}\}$ be the set of dormant nodes in the ith layer of B.

Definition 3.5. *Let B be a length-n, width-w ROBP outputting 1 or -1. Let $w' \geq 1$ be the smallest $w'' \geq 1$ such that for every layer of B, at most w'' ($w'' \leq w$) nodes are **not dormant**. Then, w' is called the **effective width** of B. Thus,*

$$\forall i \in \{0, 1, \ldots, n-1\},\ |\mathcal{L}_i^B \setminus \mathcal{D}_i^B| \leq w'.$$

3.1 Introducing Skip-Forward Read-Once Program

In this section, we formally define Skip-Forward Read-Once Program. The following exposition gives a teaser for the tight bounds for ROBPs solving the coin problem.

Definition 3.6. *We define a size-ℓ* **Skip-Forward Read-Once Program (SF-ROP)** *P on input $x_1, \ldots, x_n$ as follows. It has ℓ states, which are indexed by $\{1, \ldots, \ell-1, \ell\}$ such as $S_1, \ldots, S_\ell$ (to specify a topological order), with two special terminal states $S_{\ell-1} = t_0, S_\ell = t_1$ and a starting state S_1. Each non-terminal state $s \in \{S_j\}_{j \in [\ell-2]}$ has associated $i(s) \in [n]$ – the index of the bit that it is reading, $P(s, 0), P(s, 1)$ – the states, that are ahead of s in the topological order, and to which it transitions when $x_{i(s)} = 0$ and $x_{i(s)} = 1$, respectively.*

In addition, we require that whenever $P(s, 0)$ and $P(s, 1)$ are non-terminal, $i(P(s, 0)), i(P(s, 0)) > i(s)$. In other words, the indexes (input as well as state) at which the program is looking are strictly increasing throughout the execution. We also require that the program always reaches one of the two terminal states.

We say that the program P distinguishes distributions μ_0 and μ_1 with advantage δ if

$$\left| \Pr_{x \sim \mu_0}[P \text{ reaches } t_1] - \Pr_{x \sim \mu_1}[P \text{ reaches } t_1] \right| > \delta$$

t_0 corresponds to an output of 0 (or -1 depending on context) and t_1 corresponds to outputting 1. For $x \in \{0, 1\}^n$, $P(x)$ denotes the output of P when executed on x. We say that a SF-ROP P solves the coin problem for bias β if

$$\Pr_{\forall i \in [n], x_i \sim \text{Ber}(1/2+\beta)}[P(x) = 1] \geq 2/3 \quad \text{and} \quad \Pr_{\forall i \in [n], x_i \sim \text{Ber}(1/2-\beta)}[P(x) = 0] \geq 2/3.$$

SF-ROP is a more general model of computation than a ROBP as a length-n width-w ROBP can be seen as a SF-ROP of size at most $(w \cdot n + 1)$ on an n-bit input[9]. Thus, a lower bound of s on the size of SF-ROP would translate to a lower bound of $(s - 1)/n$ on the width of a ROBP.

Definition 3.7. *For every ℓ-sized SF-ROP P using n input bits, we define flattened version of P as Flat[P], which is an ℓ-sized SF-ROP using $\ell - 2$ input bits, such that $\forall j \in [\ell - 2], b \in \{0, 1\}$, Flat[$P$]$(S_j, b) = P(S_j, b)$ and $i(S_j) = j$. In other words, under Flat[P], each state has a unique input bit associated with it, matching the index of the state under the topological ordering.*

[9] order the states the ROBP in increasing order of the corresponding layers, and this satisfies the requirements of a SF-ROP.

We observe that for the purpose of solving the coin problem where each input bit is *i.i.d.*, the state-input bit associations under a SF-ROP doesn't affect the probability of reaching $t_1/\ t_0$. This allows us to convert a SF-ROP into a ROBP with **effective width** 1 (more on this in Section 4.2), which performs well under recursion. Formally,

Claim 3.8. *Let P_1 and P_2 be two ℓ-sized SF-ROPs having identical state transition functions, that is, $\forall s \in \{S_j\}_{j\in[\ell-2]}, b \in \{0,1\}$, $P_1(s,b) = P_2(s,b)$, but different association functions $i_1(s) \in [n_1]$ and $i_2(s) \in [n_2]$ respectively. Then, $\forall\ 0 \le p \le 1$,*

$$\Pr_{\forall i\in[n_1], x_i\sim Ber(p)}[P_1(x)=1] = \Pr_{\forall i\in[n_2], x_i\sim Ber(p)}[P_2(x)=1]$$

and similarly for output 0*.*

Proof. We prove the claim for P and Flat$[P]$, and that would prove the claim for P_1 and P_2, as the flattened version of P_1 and P_2 are equivalent. As P is read-once (reads every input at most once in order), and all input bits are *i.i.d.* Ber(p), conditioned on reaching any non-terminal state s, $x_{i(s)}$ is drawn from Ber(p) even conditioned on reaching s. Then, if we transition according to a new *i.i.d.* Ber(p) bit on state s, the probability of reaching the next state remains the same and it doesn't affect the subsequent transitions which read other independent bits. Thus, we can give each state their unique bit, as in Flat$[P]$, without changing the distribution of the output when bits are drawn *i.i.d.* Ber(p). □

In Section 5 (Lemma 5.16), we prove that any SF-ROP that solves the coin problem for bias β with success probability 2/3 requires $\Omega(\beta^{-3})$ size. In Section 4.2 (Figure 2, Remark 4.6), we construct a SF-ROP of size $O(\beta^{-3})$, thus showing tight bounds for SF-ROPs solving the coin problem. The lower bound on the size of a SF-ROP solving the coin problem requires work and the lower bound for ROBPs then follows easily. Whereas, the upper bound for SF-ROP solving the coin problem follows comparatively easily but the upper bound for ROBPs, using the corresponding flattened SF-ROP, requires work.

4 Coin Problem Upper Bound

In this section, we construct a $O(\log n)$-width ROBP that solves the coin problem for bias greater $n^{-1/3+\epsilon}$, with success probability at least 0.98. In Section 4.1, we describe a random process that amplifies the bias of a random bit (almost) as well as majority but in square root number of states. In Section 4.2, we construct a length-$m^{3/2}$, width-$m^{3/2}$ ROBP with **effective width** 1 that amplifies the bias β to $\approx m^{1/2}\cdot\beta$. In Section 4.3, we recurse to amplify the bias and length exponentially, while increasing the width only linearly.

4.1 Bias-Amplifying Random Process

Consider the following k (k is odd) step random process (imitating majority) on states $\{-k', -k'+1, \ldots, 0, \ldots, k'-1, k'\}$ associated with parameter p (also described in Figure 1). We start the random process at state $s_0 = 0$. Let s_i be the random variable denoting the state reached after i steps.

Random Process $RP(k, k', p)$

1. (Start) At the start of the random process, state is 0, that is, $s_0 = 0$.
2. (Walk) For all $i \in [k]$, at the ith step, sample an input bit $b_i \sim$ Ber(p). Update the state as follows:

$$s_{i+1} = \begin{cases} s_i & \text{if } s_i \in \{k', -k'\} \\ s_i + b_i & \text{otherwise.} \end{cases}$$

3. (End) Let y be a random variable that takes value -1 if $s_k < 0$ and 1 if $s_k > 0$.

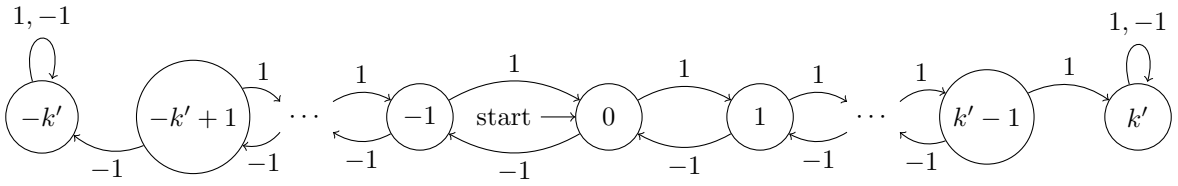

Figure 1: State transitions for the Random Process $RP(k, k', p)$. The process starts at 0 and takes k steps according to the above automaton where at each step, input bit is 1 with probability p.

Abusing the notation, given a k-bit sequence $x = (x_1, \ldots, x_k) \in \{-1, 1\}^k$, we define $s_i(x)$ to be the state reached at the ith step after following Update 2 on the input sequence x (that is, instead of sampling the bit b_i, we use $b_i = x_i$). Note that, as k is odd, $s_k \neq 0$ and, we let $y(x) = 1$ if $s_k(x) > 0$ and -1 if $s_k(x) < 0$.

We prove the following claim comparing the probabilities that the random process $RP(k, k', p)$ end at j or $-j$ ($j \in [k']$), which would be useful in analyzing the probability that the output (y) is 1.

Claim 4.1. *Let $k \geq k' > 0$. Then, under the random process $RP(k, k', p)$ ($p \in (0, 1)$), for all $j \in [k']$,*

$$\frac{\Pr[s_k = j]}{\Pr[s_k = -j]} = \left(\frac{p}{1-p}\right)^j.$$

The proof follows easily using symmetry and we give the complete proof in Appendix A.1. The claim allows to show the following proposition for odd k (which is formally proved in Appendix A.1).

Proposition 4.2 (Bias Doesn't Decrease)**.** *For odd $k \geq k' > 0$ and $0 < \delta \leq \frac{1}{2}$, under the random process $RP\left(k, k', \frac{1}{2} + \delta\right)$, we have that*

$$\Pr[y = 1] = \Pr[s_k > 0] \geq \frac{1}{2} + \delta.$$

And under the random process $RP\left(k, k', \frac{1}{2} - \delta\right)$, we have that $\Pr[y = -1] = \Pr[s_k < 0] \geq \frac{1}{2} + \delta$, implying

$$\Pr[y = 1] = \Pr[s_k > 0] \leq \frac{1}{2} - \delta.$$

Next, we look into the probability that $s_k > 0$ under the random process $RP(k, k', p)$ for certain values of k, k' and p. We assume that k is odd and $k \geq c_{amp}$ where $c_{amp} > 0$ is a large enough constant to be determined later. We take $k' = \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor$ and assume $p = \frac{1}{2} + \delta$ (where $0 < \delta \leq \frac{1}{\sqrt{k}}$). We prove that $\Pr[y = 1] = \Pr[s_k > 0] > \frac{1}{2} + \frac{1}{20}\sqrt{k} \cdot \delta$ for such values of k, k' and p.

Lemma 4.3 (*RP* Amplifies Bias)**.** *For odd $k \geq c_{amp}$ (where $c_{amp} > 0$ is a large enough constant) and $0 < \delta \leq \frac{1}{\sqrt{k}}$, under the random process $RP\left(k, \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor, \frac{1}{2} + \delta\right)$, we have that*

$$\Pr[y = 1] = \Pr[s_k > 0] > \frac{1}{2} + \frac{1}{20}\sqrt{k} \cdot \delta. \tag{5}$$

And under the random process $RP(k, \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor, \frac{1}{2} - \delta)$, we have that $\Pr[y = -1] = \Pr[s_k < 0] > \frac{1}{2} + \frac{1}{20}\sqrt{k} \cdot \delta$, implying

$$\Pr[y = 1] = \Pr[s_k > 0] < \frac{1}{2} - \frac{1}{20}\sqrt{k} \cdot \delta. \tag{6}$$

Short Proof. Refer to Appendix A.1 for the complete proof. For odd k, we first show that, under the random process $RP(k, k', \frac{1}{2} + \delta)$,

$$\Pr[s_k > 0] > \frac{1}{2} + \frac{\Pr[s_k = k']}{2}\left(1 - e^{-2\delta k'}\right) \tag{7}$$

This lower bound follows from the disparity (Claim 4.1) in probability of ending at k' and $-k'$ under $RP(k, k', \frac{1}{2}+\delta)$. Using the fact that $\forall x \in [0,1], e^{-x} \leq 1 - \frac{x}{2}$ and $2\delta k' \leq 1$ for $k' = \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor$ and $\delta \leq \frac{1}{\sqrt{k}}$, we get that

$$\Pr[s_k > 0] \geq \frac{1}{2} + \frac{\Pr[s_k = k']}{2}\left(1 - (1 - \delta k')\right) = \frac{1}{2} + \Pr[s_k = k'] \cdot \frac{k'}{2} \cdot \delta$$

Using Sterling's approximation and facts about Binomial distribution, we prove that $\Pr[s_k = k'] \geq \frac{1}{4}$ for $k' = \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor$ and $\delta \geq 0$. Thus, continuing the calculations above, we get the following bound on $\Pr[s_k > 0]$, assuming that $k \geq c_{amp} \geq 100$:

$$\Pr[s_k > 0] > \frac{1}{2} + \frac{1}{8} \cdot \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor \cdot \delta \geq \frac{1}{2} + \frac{1}{20}\sqrt{k} \cdot \delta \tag{8}$$

Thus, $\Pr[y = -1] = \Pr[s_k < 0] < \frac{1}{2} - \frac{1}{20}\sqrt{k}\delta$. By symmetry, under the random process $RP\left(k, \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor, \frac{1}{2} - \delta\right)$, we get that $\Pr[s_k > 0] < \frac{1}{2} - \frac{1}{20}\sqrt{k}\delta$ (for $0 < \delta \leq \frac{1}{\sqrt{k}}$ and $k \geq ck$). □

Starting with bits that are biased towards 1 with probability $\frac{1}{2}+\delta$, the random process $RP(k, \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor, \frac{1}{2}+\delta)$ "outputs" a random variable y that is biased towards 1 with probability at least $\frac{1}{2} + \frac{1}{20}\sqrt{k} \cdot \delta$ (for $0 < \delta < \frac{1}{\sqrt{k}}$), which is strictly greater than $\frac{1}{2} + \delta$ for $k > 400$. Hence, we view the random process as a bias-amplification gadget. This random process approximates majority and gets optimal amplification up to constant factors. We use $k' = \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor$ to amplify the bias till $\frac{1}{\sqrt{k}}$ and then use the majority function and the following claim for end cases.

Claim 4.4 (Majority is a Distinguisher for Biased Coins)**.** *Let $X_1, ..., X_m$ be i.i.d. $\sim$ Ber(p) random variables. Then for $p \geq \frac{1}{2} + \frac{1}{r}$ and $m \geq 20r^2$,*

$$\Pr\left[\sum_{i=1}^{m} X_i > 0\right] \geq 0.99$$

And for $p \leq \frac{1}{2} - \frac{1}{r}$ and $m \geq 20r^2$,

$$\Pr\left[\sum_{i=1}^{m} X_i < 0\right] \geq 0.99$$

Claim follows easily using Chernoff bound [Che52] and see Appendix A.1 for details. Consider the random process $RP(m, m, p)$. Then, $\Pr[s_m > 0]$ is exactly equal to the probability that $\Pr\left[\sum_{i=1}^{m} X_i > 0\right]$, where $X_1, ..., X_m$ are *i.i.d.* $\sim \text{Ber}(p)$ random variables. This leads to the following proposition.

Proposition 4.5. *Let $0 < \delta \leq \frac{1}{2}$ and $k \geq \frac{20}{\delta^2}$. Under the random process $RP\left(k, k, \frac{1}{2} + \delta\right)$, we have that*

$$\Pr[y = 1] = \Pr[s_k > 0] \geq 0.99.$$

And under the random process $RP\left(k, k, \frac{1}{2} - \delta\right)$, we have that

$$\Pr[y = 1] = \Pr[s_k > 0] \leq 0.01.$$

4.2 Bias-Amplifying Read-Once Branching Program

Next, we construct a read-once branching program that imitates the random process $RP(k, k', p)$ and amplify the bias of a coin as RP does.

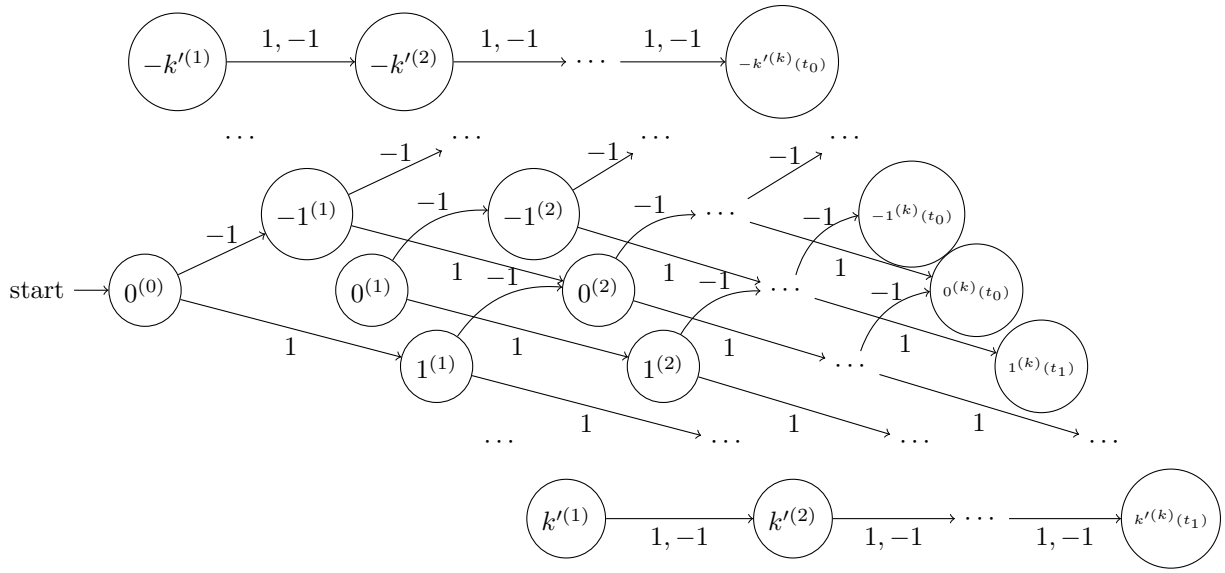

Figure 2: SF-ROP imitating the Random Process $RP(k, k', p)$ when input bits are *i.i.d.* Ber(p). The states of P are ordered as $0^{(0)}, -k'^{(1)}, \ldots, 0^{(1)}, \ldots, k'^{(1)}, \ldots, -k'^{(k-1)}, \ldots, 0^{(k-1)}, \ldots, k'^{(k-1)}$, t_0, t_1. $(i+1)$th input bit is associated with the state $[s]^{(i)}$ $(s \in \{-k', \ldots, k'\})$.

Bias-amplifying width-based ROBP Our first attempt at the read-once branching program $B^{(1)}$ just stores the state of the random process at each step, and outputs 1 if the end state is greater than 0 and -1 otherwise. Let B' be a length-k width-$(2k'+1)$ read-once branching program that stores the state after i steps of the random process $RP(k, k', p)$ in the ith layer. Formally, $\forall i \in [n], s \in \{-k', \ldots, 0, \ldots, k'\}, b \in \{-1, 1\}$,

$$B'_i(s, b) = \begin{cases} s & \text{if } s \in \{k', -k'\} \\ s + b & \text{otherwise,} \end{cases}$$

where $s_{B'} = 0$. We use the merge operation to define the $B^{(1)}$ (Definition 3.3). Let

$$O_{B'}(s) = \begin{cases} 1 & \text{if } s > 0 \\ -1 & \text{otherwise} \end{cases}.$$

$B^{(1)} = \text{Merge}(B', O_{B'})$. It is easy to see that $B^{(1)}(x) = y(x)$, that is, 1 if $s_k(x) > 0$ and -1 if $s_k(x) < 0$ (for odd k). Lemma 4.3 applies to the output of $B^{(1)}$ when input is *i.i.d.* Ber$(1/2 \pm \delta)$.

The **effective width** of $B^{(1)}$ is $2k' - 1$ (states k' and $-k'$ are **dormant** in every layer). Thus, the width blows up exponentially when we recurse on $B^{(1)}$ to amplify the bias to the maximum. As mentioned in Section 3.1, we use Skip-Forward Read-Once Programs as an intermediary model to construct a ROBP $B^{(2)}$ (which is equivalent to $B^{(1)}$ for the purpose of solving the coin problem) with **effective width** 1.

Bias-amplifying SF-ROP (Figure 2) Let P be a size-ℓ SF-ROP on k-bit input representing $B^{(1)}$, where $\ell < k(2k' + 1) + 1$. P orders the states in increasing order of the layers, that is, in the order $0^{(0)}, -k'^{(1)}, \ldots, 0^{(1)}, \ldots, k'^{(1)}, \ldots, t_0, t_1$ (all edges thus go from left to right). Here, the states $\{-k'^{(i)}, \ldots, 0^{(i)}, \ldots, k'^{(i)}\}$ represent the ith layer of the branching program $B^{(1)}$ and reads x_{i+1}, and the accept state of $B^{(1)}$ is replaced by t_1 and reject state by t_0. P imitates the Random Process $RP(k, k', p)$ when input is *i.i.d.* Ber(p).

Remark 4.6 (Upper bound for SF-ROP solving the coin problem)**.** *Let $k = \Theta(k'^2)$ such that, under $RP(k, k', \frac{1}{2} + \delta)$,*

$$\Pr[s_k \in \{-k', k']\} \geq 0.99.$$

Such a k exists for every k' using anti-concentration of Binomial distribution. Then, for every $\delta > 0$, there exists a large enough $k' = \Theta(1/\delta)$, such that

$$\Pr[s_k > 0] > \Pr[s_k = k'] > 2/3$$

under $RP(k, k', 1/2 + \delta)$ (using Claim 4.1). This gives a $O(\delta^{-3})$-sized SF-ROP that solves the coin problem with probability $2/3$.

Bias-amplifying time-based ROBP Next, we use Flat$[P]$ (Definition 3.7), which is an $\ell-$*sized* SF-ROP on $\ell - 2$ input bits ($\ell \leq k(2k' + 1) + 1$), to construct a length-$(\ell - 2)$, width-$(\ell - 1)$ ROBP $B^{(2)}$ with **effective width** equal to 1 satisfying the bias-amplification properties of $RP(k, k', p)$. The reduction is simple – let $S_1, S_2, \ldots, S_\ell$ be the states of Flat$[P]$. Layer i of $B^{(2)}$ ($0 \leq i \leq \ell - 2$) corresponds to the state S_{i+1} of Flat$[P]$ (except for last layer, whose reject state corresponds to $S_{\ell-1} = t_0$ and accept state corresponds to $S_\ell = t_1$). The nodes in $B^{(2)}$ (except for in the start and the last layer) are labelled by FF$[s]$, where $s \in \{S_j\}_{2 \leq j \leq \ell}$. The start vertex of $B^{(2)}$ is labelled by FF$[S_1]$. FF$[s]$ is short for 'Fast-Forward to s' and it signifies a command to ignore the input bits till you reach the layer or time-step corresponding to state s. And, then $B^{(2)}$ transitions according Flat$[P]$ to the next state. For the ease of formally defining the transition functions of $B^{(2)}$, we also label the accept vertex as FF$[S_\ell]$ and reject vertex as FF$[S_{\ell-1}]$. The transition function from layer $(i-1)$ to i ($i \in [\ell - 2]$) is as follows ($s \in \{S_j\}_{2 \leq j \leq \ell}, b \in \{-1, 1\}$):

$$B_i^{(2)}(\text{FF}[s], b) = \begin{cases} \text{FF}[\text{Flat}[P](s, b)] & \text{if } s = S_i \\ \text{FF}[s] & \text{otherwise} \end{cases}$$

Recall that Flat$[P](s, b)$ represents the state, ahead of s in the topological order, to which it transitions when the associated input-bit is b. On every $(\ell - 2)$-bit input x, $B^{(2)}$ is able to execute Flat$[P]$ on x in read-once fashion as the state indexes keep increasing during the execution. $B^{(2)}$ uses the jth input bit iff it reaches the state S_j (under Flat$[P]$, $i(S_j) = j$) and it reaches an accept/reject state as a SF-ROP always reaches a terminal state. Therefore, $\forall x, B^{(2)}(x) = P(x)$. For all $i \in \{0, \ldots, \ell - 3\}$, all nodes except FF$[S_{i+1}]$ are **dormant** in the ith layer of $B^{(2)}$ and thus the **effective width** of $B^{(2)}$ is 1. Using Claim 3.8 and equivalence between $B^{(1)}$ and P, we get that the output of $B^{(2)}$ imitates the end state of the Random Process $RP(k, k', p)$ and thus, $B^{(2)}$ satisfies the bias-amplification properties of RP. Lemma 4.3, Proposition 4.2 and Proposition 4.5 implies the following lemma.

Lemma 4.7. *The read-once branching program $B^{(2)}$ with length and width of at most $k(2k' + 1)$, as defined above, satisfies the following properties:*

1. *(Imitating Bias-Amplification Gadget) For odd $k \geq c_{amp}$ (where $c_{amp} > 0$ is a large enough constant), $k' = \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor$ and $0 < \delta \leq \frac{1}{\sqrt{k}}$,*

$$\Pr_{\forall i,\, x_i \sim Ber\left(\frac{1}{2}+\delta\right)}[B^{(2)}(x) = 1] > \frac{1}{2} + \frac{1}{20}\sqrt{k}\delta,$$

and

$$\Pr_{\forall i,\, x_i \sim Ber\left(\frac{1}{2}-\delta\right)}[B^{(2)}(x) = 1] < \frac{1}{2} - \frac{1}{20}\sqrt{k}\delta.$$

2. *(Bias Does't Decrease) For odd $k \geq k' > 0$ and $0 < \delta \leq \frac{1}{2}$,*

$$\Pr_{\forall i,\, x_i \sim Ber\left(\frac{1}{2}+\delta\right)}[B^{(2)}(x) = 1] \geq \frac{1}{2} + \delta,$$

and

$$\Pr_{\forall i,\, x_i \sim Ber\left(\frac{1}{2}-\delta\right)}[B^{(2)}(x) = 1] \leq \frac{1}{2} - \delta.$$

3. *(Imitating Majority) For $k' = k > 0$ and $\sqrt{\frac{20}{k}} \leq \delta \leq \frac{1}{2}$,*

$$\Pr_{\forall i,\, x_i \sim Ber\left(\frac{1}{2}+\delta\right)}[B^{(2)}(x) = 1] \geq 0.99,$$

and

$$\Pr_{\forall i,\, x_i \sim Ber\left(\frac{1}{2}-\delta\right)}[B^{(2)}(x) = 1] \leq 0.01.$$

4. *The **effective width** of $B^{(2)}$ is 1.*

4.3 $O(\log n)$-width ROBP Solving the Coin Problem

Bias-Amplification Recursion Before we describe the ROBP to solve the coin problem for bias greater than $n^{-1/3+\epsilon}$, we analyze the recursion that represents the tensorization of the bias-amplification process. The following lemma is more sophisticated in calculations but basically explains the following idea: if the bias gets amplified by $k^{1/2}$ at every recursion level (for a constant k), length gets multiplied by $k^{3/2}$ and width increases by $poly(k)$, then we get a n-length $O(\log n)$-width ROBP that amplifies bias $\approx n^{-1/3}$ to a constant.

Lemma 4.8. *Let $\delta > 0$ and $k > 400$. Let $r = k \cdot (2\left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor + 1)$. We consider the following recursion:*

1. *Initialization: $\delta_0 = \min\left(\delta, \frac{1}{\sqrt{k}}\right)$, $n_0 = 1$ and $w_0 = 0$.*
2. *Recursion: $\delta_i = \min\left(\frac{1}{20}\sqrt{k} \cdot \delta_{i-1}, \frac{1}{\sqrt{k}}\right)$, $n_i \leq n_{i-1} \cdot r$ and $w_i \leq w_{i-1} + r$.*

Then, for all $\epsilon \in (0, 1/3)$, $k > \max(400, c_\epsilon)$ (where $c_\epsilon > 0$ is a large enough constant depending on ϵ), $n \geq c_{k,\epsilon}$ (where $c_{k,\epsilon} > 0$ is a large enough constant depending on k and ϵ), $\delta \geq n^{-\frac{1}{3}+\epsilon}$, for $j = \left\lfloor \frac{\log n - \log(20k(40k+1))}{1.5\log k + 1} \right\rfloor$, we have that

$$\delta_j \geq \frac{1}{\sqrt{k}}, \tag{9}$$

$$n_j \leq \frac{n}{(20k(40k+1))} \tag{10}$$

and

$$w_j = j \cdot r \leq k(\sqrt{k}+1) \cdot \log n \tag{11}$$

The proof follows from straightforward calculations and is deferred to Appendix A.2.

Tensorization of the ROBPs To use the bias-amplification recursion as analyzed, we define the tensor of two ROBPs as follows:

Definition 4.9. *Let A be a length-n_a width-w_a ROBP with **effective width** w'_a (Definition 3.5) and B be a length-n_b width-w_b ROBP. Then, $A \otimes B$ is a length-$n_a \cdot n_b$ ROBP defined as follows:*

1. *Description of the layers: Nodes of $A \otimes B$ are cartesian products of nodes in A and that in B (in the corresponding layers) except for when they are dormant. More formally,*
   - $\forall i \in [n_a]$, $\mathcal{L}^{A\otimes B}_{(i-1)\cdot n_b} = \{(s, s_B) \mid s \in \mathcal{L}^A_{i-1}\}$. *Recall that s_B represents the starting node of B.*
   - $\forall i \in [n_a], j \in [n_b - 1]$,
   $$\mathcal{L}^{A\otimes B}_{(i-1)\cdot n_b+j} = \{(s, s_B) \mid s \in \mathcal{D}^A_{i-1}\} \cup \{(s, s') \mid s \in \mathcal{L}^A_{i-1} \setminus \mathcal{D}^A_{i-1} \text{ and } s' \in \mathcal{L}^B_j\}.$$
   - $\mathcal{L}^{A\otimes B}_{n_a\cdot n_b} = \{1, -1\}$.
2. *Description of the transition functions: On layers that do not correspond to the last layer of B, $A \otimes B$ transitions according to B and then uses the output of B to transition according to A. More formally,*
   - $\forall i \in [n_a - 1]$, *and* $\forall s \in \mathcal{L}^A_{i-1} \setminus \mathcal{D}^A_{i-1}, s' \in \mathcal{L}^B_{n_b-1}$,
   $$(A \otimes B)_{i\cdot n_b}((s, s'), b) = (A_i(s, B_{n_b}(s', b)), s_B),$$
   and $\forall s \in \mathcal{D}^A_{i-1}$ *($A_i(s, 1) = A_i(s, -1)$),*
   $$(A \otimes B)_{i\cdot n_b}((s, s_B), b) = (A_i(s, 1), s_B).$$
   These are the transitions according to A using the output of B.
   - $\forall i \in [n_a], j \in [n_b - 1]$, $\forall s \in \mathcal{L}^A_{i-1} \setminus \mathcal{D}^A_{i-1}, s' \in \mathcal{L}^B_{j-1}$,
   $$(A \otimes B)_{(i-1)\cdot n_b+j}((s, s'), b) = (s, B_j(s', b)),$$
   and $\forall s \in \mathcal{D}^A_{i-1}$,
   $$(A \otimes B)_{(i-1)\cdot n_b+j}((s, s_B), b) = (s, s_B).$$
   These are the transitions according to B per node of A.
   - $\forall s \in \mathcal{L}^A_{n_a-1} \setminus \mathcal{D}^A_{n_a-1}, s' \in \mathcal{L}^B_{n_b-1}$,
   $$(A \otimes B)_{n_a\cdot n_b}((s, s'), b) = A_{n_a}(s, B_{n_b}(s', b)),$$
   and $\forall s \in \mathcal{D}^A_{n_a-1}$ *($A_{n_a}(s, 1) = A_{n_a}(s, -1)$),*
   $$(A \otimes B)_{n_a\cdot n_b}((s, s_B), b) = A_{n_a}(s, 1).$$
   This defines the transitions to the output layer of $A \times B$.

Under $A \otimes B$, A uses the output of B to make the transition at every node rather than using the next input bit. Thus, each transition of A is replaced by B that reads the next n_b bits and its output is then used to transition according to A. Each node of A is thus expanded into a length-n_b width-w_b ROBP. However, we can do better at dormant nodes of A. As, for dormant nodes, the output of B doesn't affect the node in the next layer that A transitions to, we can replace each dormant node of A by a length-n_b width-1 ROBP always outputting 1 instead of B, thus saving on the width of $A \otimes B$.

Remark 4.10. *The $(n_a \cdot n_b)$-length ROBP $A \otimes B$ as in Definition 4.9 has width at most $(w'_a \cdot (w_b - 1) + w_a)$.*

Proof. $A \otimes B$ is a width-$(w'_a \cdot (w_b - 1) + w_a)$ ROBP as all the layers have at most that many nodes and this can be shown formally as follows. $\forall i \in [n_a], |\mathcal{L}^{A\otimes B}_{(i-1)\cdot n_b}| = |\mathcal{L}^A_{i-1}| \leq w_a$ and $\forall i \in [n_a], j \in [n_b - 1]$,

$$\begin{aligned}
\mathcal{L}^{A\otimes B}_{(i-1)\cdot n_b + j} &= |\mathcal{D}^A_{i-1}| + |\mathcal{L}^A_{i-1} \setminus \mathcal{D}^A_{i-1}| \cdot |\mathcal{L}^B_j| \\
&= |\mathcal{L}^A_{i-1}| - |\mathcal{L}^A_{i-1} \setminus \mathcal{D}^A_{i-1}| + |\mathcal{L}^A_{i-1} \setminus \mathcal{D}^A_{i-1}| \cdot |\mathcal{L}^B_j| \\
&= |\mathcal{L}^A_{i-1}| + |\mathcal{L}^A_{i-1} \setminus \mathcal{D}^A_{i-1}| \cdot (|\mathcal{L}^B_j| - 1) \\
&\leq w_a + w'_a \cdot (w_b - 1). \quad \square \qquad \text{(\textbf{effective width} of } A \text{ is } w'_a\text{)}
\end{aligned}$$

Remark 4.11. *The output of $A \otimes B$ (Definition 4.9) on input $x \in \{1,-1\}^{n_a \cdot n_b}$ is given by*

$$A \otimes B(x) = A\left(B(x[1:n_b]), B(x[n_b+1:2n_b]), \ldots, B(x[(n_a-1)\cdot n_b + 1 : n_a \cdot n_b])\right).$$

Proof. We focus on the $(i \cdot n_b)$th ($i \in [n_a]$) layers of $A \otimes B$. Let C be a length-n width-w ROBP. Let $0 \leq i_1 < i_2 \leq n$ and $C_{[i_1,i_2]} : \mathcal{L}^C_{i_1} \times \{1,-1\}^{(i_2-i_1)} \to \mathcal{L}^C_{i_2}$ represent the state transition from the i_1th layer of C to i_2th layer of C, that is, the path labelled by $y \in \{1,-1\}^{(i_2-i_1)}$ from a state t in the i_1th layer goes to the state $C_{[i_1,i_2]}(t,y)$ in the i_2th layer. By Definition 4.9, $\mathcal{L}^{A\otimes B}_{(i-1)\cdot n_b} = \{(s, s_B) \mid s \in \mathcal{L}^A_{i-1}\}$. We prove that (for $y \in \{1,-1\}^{n_b}$),

$$\forall i \in [n_a - 1], (A \otimes B)_{[(i-1)\cdot n_b, i\cdot n_b]}((s, s_B), y) = (A_i(s, B(y)), s_B)$$

and

$$(A \otimes B)_{[(n_a-1)\cdot n_b, n_a\cdot n_b]}((s, s_B), y) = A_{n_a}(s, B(y)),$$

implying that $A \times B(y_1, y_2, \ldots, y_{n_a}) = A(B(y_1), B(y_2), \ldots, B(y_{n_a}))$, which would prove the remark. Now, we look at the function $(A \otimes B)_{[(i-1)\cdot n_b, i\cdot n_b]}$ for a fixed $i \in [n_a - 1]$ (proof for $(A \otimes B)_{[(n_a-1)\cdot n_b, n_a\cdot n_b]}$ follows similarly). Let $s \in \mathcal{L}^A_{i-1} \setminus \mathcal{D}^A_{i-1}$, then by Definition 4.9 (Point 2), we have that (for $y \in \{1,-1\}^{n_b}$),

$$(A \otimes B)_{[(i-1)\cdot n_b, i\cdot n_b - 1]}((s, s_B), y[1:n_b-1]) = (s, B_{[0,n_b-1]}(s_B, y[1:n_b-1])),$$

and as $\forall s' \in \mathcal{L}^B_{n_b-1}, b \in \{-1,1\}$, $(A \otimes B)_{i\cdot n_b}((s,s'),b) = (A_i(s, B_{n_b}(s', b)), s_B)$, we get that

$$\begin{aligned}
(A \otimes B)_{[(i-1)\cdot n_b, i\cdot n_b]}((s, s_B), y) &= (A_i(s, B_{n_b}(B_{[0,n_b-1]}(s_B, y[1:n_b-1]), y_{n_b})), s_B) \\
&= (A_i(s, B(y)), s_B). \qquad (B_{n_b}(B_{[0,n_b-1]}(s_B, y[1:n_b-1]), y_{n_b}) = B(y))
\end{aligned}$$

Next, we prove the same statement for $s \in \mathcal{D}^A_{i-1}$. By Definition 4.9 (Point 2), we have that (for $y \in \{1,-1\}^{n_b}$)

$$(A \otimes B)_{[(i-1)\cdot n_b, i\cdot n_b - 1]}((s, s_B), y[1:n_b-1]) = (s, s_B),$$

and as $(A \otimes B)_{i\cdot n_b}((s, s_B), b) = (A_i(s,1), s_B)$, we get that

$$\begin{aligned}
(A \otimes B)_{[(i-1)\cdot n_b, i\cdot n_b]}((s, s_B), y) &= (A_i(s,1), s_B) \\
&= (A_i(s, B(y)), s_B). \qquad (A_i(s,1) = A_i(s,-1) \text{ for } s \in \mathcal{D}^A_{i-1})
\end{aligned}$$

This completes the proof. $\square$

Main Theorem Now that we have all the ingredients, we are ready to prove the main theorem.

Theorem 4.12. *For all constant $0 < \epsilon < \frac{1}{3}$ and $n \geq C_\epsilon$ (where $C_\epsilon > 0$ is a large enough constant), there exists a length-n width-$O_\epsilon(\log n)$ (constants in the Big Oh notation depend on ϵ) ROBP B that solves the coin problem for parameter $\delta \geq n^{-\frac{1}{3}+\epsilon}$ with an advantage 0.98 (according to Definition 3.2).*

The constant 0.98 is arbitrary and we can update the construction to achieve an advantage of at least $1 - \gamma$ for any constant $\gamma > 0$, while losing only constant factors in the width.

Proof. We use Lemma 4.7 and 4.8 to prove the theorem. For the given constant ϵ, we take k to be the smallest odd integer strictly greater than $\max(c_{amp}, 400, c_\epsilon)$, where $c_{amp} > 0$ and $c_\epsilon > 0$ are large enough constants determined in Lemmas 4.3 and 4.8 respectively. We assume $C_\epsilon \geq c_{k,\epsilon}$ where $c_{k,\epsilon} > 0$ is the constant determined in Lemma 4.8. Given n, we take $j = \left\lfloor \frac{\log n - \log(20k(40k+1))}{1.5 \log k + 1} \right\rfloor$ as in Lemma 4.8.

We define B^{top} to be the branching program $B^{(2)}$ as defined in Section 4.1 (Lemma 4.7) imitating the $20k$-step random process RP with $k' = 20k$ (B^{top} has length and width of at most $(20k(40k+1))$). We define B^{amp} to be the branching program $B^{(2)}$ (with length and width at most $k\left(2\left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor + 1\right)$) as defined in Section 4.1 imitating the k-step random process RP with $k' = \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor$. We define the branching program B using a sequence of branching programs through tensorization as follows:

1. Let $B^{[1]} = B^{\text{amp}}$.
2. $\forall i \in \{2, 3, \ldots, j\}$, let $B^{[i]} = B^{\text{amp}} \otimes B^{[i-1]}$.
3. $B = B^{\text{top}} \otimes B^{[j]}$.

Analyzing the width of B Let w_i denote the width of the branching program $B^{[i]}$. Then, w_1 is equal to the width of the branching program B^{amp}, that is, $w_1 \leq k\left(2\left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor + 1\right)$. As the **effective width** of B^{amp} is 1 (Lemma 4.7), using Remark 4.10, $\forall i \in \{2, 3, \ldots, j\}$, $w_i \leq 1 \cdot (w_{i-1} - 1) + r \leq w_{i-1} + r$ (where $r = k \cdot (2\left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor + 1)$). Thus, using the recursion in Lemma 4.8, by Equation (11), we have that $w_j \leq k(\sqrt{k}+1)\log n$. Let w_{top} be the width of the branching program B^{top}, that is, $w_{\text{top}} = (20k(40k+1))$. As $B = B^{\text{top}} \otimes B^{[j]}$ and **effective width** of B^{top} is 1, width of B is at most $w_j - 1 + (20k(40k+1)) \leq k(\sqrt{k}+1)\log n + poly(k) = O_\epsilon(\log n)$, where O_ϵ hides constants that depends on ϵ.

Analyzing the length of B We will prove that B as defined above has length n', where $n' \leq n$, but we would assume the length of B to be n as an n'-length ROBP can be trivially extended to a n-length ($\geq n'$) ROBP by adding $n - n'$ identity layers. Such an operation preserves the properties of B to solve the coin problem. Let n_i denote the length of the branching program $B^{[i]}$. Then, n_1 is equal to the length of the branching program B^{amp}, that is, $n_1 \leq k \cdot \left(2\left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor + 1\right)$. By Definition 4.9, $\forall i \in \{2, 3, \ldots, j\}$, $n_i \leq n_{i-1} \cdot k\left(2\left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor + 1\right)$. Thus, using the recursion in Lemma 4.8, by Equation (10), we have that

$$n_j \leq \frac{n}{(20k(40k+1))}.$$

Let n_{top} be the length of the branching program B^{top}, that is, $n_{\text{top}} = (20k(40k+1))$. As $B = B^{\text{top}} \otimes B^{[j]}$, the length of B (by Definition 4.9) is given by $n_j \cdot n_{\text{top}} \leq \frac{n}{(20k(40k+1))} \cdot (20k(40k+1)) = n$.

Analyzing the output of B on *i.i.d.* biased bits To prove that B solves the coin problem for parameter δ (where $\delta \geq n^{-\frac{1}{3}+\epsilon}$) with an advantage 0.98 (by Definition 3.2), it suffices to show the following:

$$\Pr_{\forall l \in [n], x_l \sim \text{Ber}\left(\frac{1}{2}+\delta\right)}[B(x) = 1] \geq 0.99 \quad \text{and} \quad \Pr_{\forall l \in [n], x_l \sim \text{Ber}\left(\frac{1}{2}-\delta\right)}[B(x) = -1] \geq 0.99.$$

For all $i \in [j]$, let $\delta_0 = \min\left(\delta, \frac{1}{\sqrt{k}}\right)$ and $\delta_i = \min\left(\frac{1}{20}\sqrt{k} \cdot \delta_{i-1}, \frac{1}{\sqrt{k}}\right)$. By Lemma 4.8 , as $\delta \geq n^{-\frac{1}{3}+\epsilon}$, $\delta_j \geq \frac{1}{\sqrt{k}}$. Before proving the statement for B, we prove using induction that $\forall i \in [j]$,

$$\Pr_{\forall l \in [n_i], x_l \sim \text{Ber}\left(\frac{1}{2}+\delta\right)}[B^{[i]}(x) = 1] \geq \frac{1}{2} + \delta_i \quad \text{and} \quad \Pr_{\forall l \in [n_i], x_l \sim \text{Ber}\left(\frac{1}{2}-\delta\right)}[B^{[i]}(x) = -1] \geq \frac{1}{2} + \delta_i.$$

For $i = 1$, $B^{[1]} = B^{\text{amp}}$ and by Lemma 4.7 Property 1 (k satisfies the conditions), for $0 < \delta \leq \frac{1}{\sqrt{k}}$, $\Pr_{\forall l \in [n_1], x_l \sim \text{Ber}\left(\frac{1}{2}+\delta\right)}[B^{[1]}(x) = 1] \geq \frac{1}{2} + \frac{1}{20}\sqrt{k} \cdot \delta$ and $\Pr_{\forall l \in [n_1], x_l \sim \text{Ber}\left(\frac{1}{2}-\delta\right)}[B^{[1]}(x) = -1] \geq \frac{1}{2} + \frac{1}{20}\sqrt{k} \cdot \delta$. For $\delta > \frac{1}{\sqrt{k}}$, we use Property 2 of Lemma 4.7, which says that $\Pr_{\forall l \in [n_1], x_l \sim \text{Ber}\left(\frac{1}{2}+\delta\right)}[B^{[1]}(x) = 1] \geq \frac{1}{2} + \delta$ and $\Pr_{\forall l \in [n_1], x_l \sim \text{Ber}\left(\frac{1}{2}-\delta\right)}[B^{[1]}(x) = -1] \geq \frac{1}{2} + \delta$. Therefore, we get that

$$\Pr_{\forall l \in [n_1], x_l \sim \text{Ber}\left(\frac{1}{2}+\delta\right)}[B^{[1]}(x) = 1] \geq \frac{1}{2} + \min\left(\frac{1}{20}\sqrt{k} \cdot \delta, \frac{1}{\sqrt{k}}\right) = \frac{1}{2} + \delta_1$$

$$\text{and} \quad \Pr_{\forall l \in [n_1], x_l \sim \text{Ber}\left(\frac{1}{2}-\delta\right)}[B^{[1]}(x) = -1] \geq \frac{1}{2} + \min\left(\frac{1}{20}\sqrt{k} \cdot \delta, \frac{1}{\sqrt{k}}\right) = \frac{1}{2} + \delta_1.$$

This is because, for $\delta \leq \frac{1}{\sqrt{k}}$, we use the first expression in the min and for $\delta \geq \frac{1}{\sqrt{k}}$, we use the second expression in the min. Next, we assume that

$$\Pr_{\forall l \in [n_i], x_l \sim \text{Ber}\left(\frac{1}{2}+\delta\right)}[B^{[i]}(x) = 1] \geq \frac{1}{2} + \delta_i \quad \text{and} \quad \Pr_{\forall l \in [n_i], x_l \sim \text{Ber}\left(\frac{1}{2}-\delta\right)}[B^{[i]}(x) = -1] \geq \frac{1}{2} + \delta_i,$$

and prove the statement for $i + 1$.

Let $\Pr_{\forall l \in [n_i], x_l \sim \text{Ber}\left(\frac{1}{2}+\delta\right)}[B^{[i]}(x) = 1] = \frac{1}{2} + \delta_i'$. By definition, $B^{[i+1]} = B^{\text{amp}} \otimes B^{[i]}$. Thus, by Remark 4.11,

$$B^{[i+1]}(x) = B^{\text{amp}}(B^{[i]}(x[1 : n_i]), \ldots, B^{[i]}(x[n_{i+1} - n_i + 1 : n_{i+1}])).$$

Therefore,

$$\begin{aligned}
&\Pr_{\forall l \in [n_{i+1}], x_l \sim \text{Ber}\left(\frac{1}{2}+\delta\right)}[B^{[i+1]}(x) = 1] \\
&= \Pr_{\forall l \in [n_{i+1}], x_l \sim \text{Ber}\left(\frac{1}{2}+\delta\right)}[B^{\text{amp}}(B^{[i]}(x[1 : n_i]), \ldots, B^{[i]}(x[n_{i+1} - n_i + 1 : n_{i+1}])) = 1] \\
&= \Pr_{\forall l' \in [n_1], y_{l'} \sim \text{Ber}\left(\frac{1}{2}+\delta_i'\right)}[B^{\text{amp}}(y) = 1] \\
&\qquad\qquad (\forall l \in [n_1], B^{[i]}(x[(l-1) \cdot n_i + 1 : l \cdot n_i]) \text{ are independent random variables}) \\
&\geq \frac{1}{2} + \min\left(\frac{1}{20}\sqrt{k} \cdot \delta_i', \frac{1}{\sqrt{k}}\right) \qquad \text{(using the base case as } B^{[1]} = B^{\text{amp}}) \\
&\geq \frac{1}{2} + \min\left(\frac{1}{20}\sqrt{k} \cdot \delta_i, \frac{1}{\sqrt{k}}\right) \qquad (\delta_i' \geq \delta_i) \\
&= \frac{1}{2} + \delta_{i+1}.
\end{aligned}$$

Similarly, we prove that $\Pr_{\forall l \in [n_{i+1}], x_l \sim \text{Ber}\left(\frac{1}{2}-\delta\right)}[B^{[i+1]}(x) = -1] \geq \frac{1}{2} + \delta_{i+1}$. As $\delta_j \geq \frac{1}{\sqrt{k}}$, we get that

$$\Pr_{\forall l \in [n_j], x_l \sim \text{Ber}\left(\frac{1}{2}+\delta\right)}[B^{[j]}(x) = 1] \geq \frac{1}{2} + \frac{1}{\sqrt{k}} \quad \text{and} \quad \Pr_{\forall l \in [n_j], x_l \sim \text{Ber}\left(\frac{1}{2}-\delta\right)}[B^{[j]}(x) = -1] \geq \frac{1}{2} + \frac{1}{\sqrt{k}}.$$

As B^{top} is a $(20k(40k+1))$-length $(20k(40k+1))$-width branching program (as in Lemma 4.7) imitating the $20k$-step random process RP with $k' = 20k$, by Lemma 4.7 Property 3, we get that for $\delta' \geq \frac{1}{\sqrt{k}}$,

$$\Pr_{\forall l \in [n_{\text{top}}], x_l \sim \text{Ber}\left(\frac{1}{2}+\delta'\right)}[B^{\text{top}}(x) = 1] \geq 0.99 \quad \text{and} \quad \Pr_{\forall l \in [n_{\text{top}}], x_l \sim \text{Ber}\left(\frac{1}{2}-\delta'\right)}[B^{\text{top}}(x) = -1] \geq 0.99.$$

We are now ready to prove the statement for $B = B^{\text{top}} \otimes B^{[j]}$.

$$
\begin{aligned}
&\Pr_{\forall l\in[n],x_l\sim\mathrm{Ber}\left(\frac{1}{2}+\delta\right)}[B(x)=1]\\
&=\Pr_{\forall l\in[n],x_l\sim\mathrm{Ber}\left(\frac{1}{2}+\delta\right)}[B^{\mathrm{top}}(B^{[j]}(x[1:n_j]),\ldots,B^{[j]}(x[(n_{\mathrm{top}}-1)\cdot n_j+1:n_{\mathrm{top}}\cdot n_j]))=1] && \text{(Remark 4.11)}\\
&=\Pr_{\forall l'\in[n_{\mathrm{top}}],y_{l'}\sim\mathrm{Ber}\left(\frac{1}{2}+\delta'_j\right)}[B^{\mathrm{top}}(y)=1]\\
&\geq 0.99. && (\delta'_j\geq\delta_j\geq\tfrac{1}{\sqrt{k}})
\end{aligned}
$$

Similarly, we can prove that $\Pr_{\forall l\in[n],x_l\sim\mathrm{Ber}\left(\frac{1}{2}-\delta\right)}[B(x)=-1]\geq 0.99$. □

5 Coin Problem Lower Bound

We prove our coin problem lower bound by defining a discrete *Mass Split-and-Merge Game* in Section 5.1, showing that the merge moves cannot help the game in Section 5.2, proving a lower bound on just the *Mass Split Game* in Section 5.3, and finally reducing ROBPs (actually SF-ROPs) to this game in Section 5.4.

5.1 Mass split-and-merge game and mass split game

Definition 5.1. *The MSMG is defined as follows. For $i\in\{-k,\ldots,k\}$, $t=0,\ldots,T$, we have $x_i^t\geq 0$. We have $x_0^0=1$ and $x_i^0=0$ for $i\neq 0$. For each t we have $\sum_i x_i^t=1$. x^{t+1} is obtained from x^t using one of two moves:*

- **Split move.** *For $i\in\{-k+1,\ldots,k-1\}$ and $y\in[0,x_i^t]$ we split a y portion of the mass at location i:*

$$
x_j^{t+1}=\begin{cases} x_j^t-y & \text{if } j=i\\ x_j^t+y/2 & \text{if } j=i-1,i+1\\ x_j^t & \text{otherwise}\end{cases} \tag{12}
$$

- **Merge move.** *Let $y_i\in[0,x_i^t]$ be any values such that the weighted average*

$$
\left(\sum_i i\cdot y_i\right)/\left(\sum_i y_i\right)=:\ell\in\{-k,\ldots,k\}
$$

is an integer. Set

$$
x_j^{t+1}=\begin{cases} x_j^t-y_j+\sum_i y_i & \text{if } j=\ell\\ x_j^t-y_j & \text{otherwise}\end{cases} \tag{13}
$$

Remark: The split and merge moves can be defined analogously for any non-negative $\{x_i^t\}_{i\in\{-k,\ldots,k\}}$ and have the property that $\sum_i x_i^{t+1}=\sum_i x_i^t$.

The *Mass Split Game* (MSG) is defined exactly as the MSMG game, except that only split moves are allowed (i.e., merge moves do not exist). We'll need a theorem of the following form:

Theorem 5.2. *Let x_i^t be an execution of a MSMG. Suppose that*

$$
\sum_i |i|\cdot x_i^T>\delta\cdot k.
$$

Then there have been $>c\cdot\delta^3\cdot k^3$ split moves, where $c>0$ is a universal constant.

5.2 Reducing MSMG to MSG

We now show that every MSMG strategy can be converted to a MSG strategy achieving the same expected absolute value ($\sum_i |i| \cdot x_i^T$) using at most the same number of moves. Hence, merge moves are unnecessary in the MSMG.

Lemma 5.3. *Let x_i^0 be an arbitrary state of the MSMG. Let M be a merge move turning x_i^0 to x_i^1, and let S be split move subsequently turning x_i^1 to x_i^2. Then, there exists a split move S' and a sequence of merge moves $M'_1, \ldots, M'_k$ (for some $k \leq 3$) such that starting with x_i^0 and performing the sequence of moves $S', M'_1, \ldots, M'_k$ we reach the same end state x_i^2.*

Proof. Denote by ℓ_1 the index to which the mass in the merge move M is merged into. Denote by ℓ_2 the index from which the mass in the split move S is split from.

If $\ell_1 \neq \ell_2$ we show that M and S are interchangeable and thus we are done with $k = 1,\ S' = S,\ M_1 = M$. First, $x^1_{\ell_2} \leq x^0_{\ell_2}$ and thus S is feasible starting with x_i^0. Denote by y_i^M the portion of mass moved from index i in M. Denote by y^S the portion of mass split from ℓ_2 in S. As S is feasible from x_i^1, it must hold that $y^S \leq x^1_{\ell_2} = x^0_{\ell_2} - y^M_{\ell_2}$. Denote by $\tilde{x}_i^1$ the state we reach starting from x_i^0 and applying S. For every $i \neq \ell_2$ we have $\tilde{x}_i^1 \geq x_i^0 \geq y_i^M$, and we also have $\tilde{x}^1_{\ell_2} = x^0_{\ell_2} - y^S \geq y^M_{\ell_2}$. In particular, M is feasible from $\tilde{x}_i^1$.

Otherwise, $\ell_1 = \ell_2$. Denote by $\ell := \ell_1 = \ell_2$ and y^S, y_i^M as before. Denote by $y^M = \sum_i y_i^M$ the total mass merged in M. We may assume that $y^M_{\ell_1} = 0$ as we can simply not merge the mass from the index to itself without changing the outcome of the merge move. We again consider two sub-cases.

If $y^M < y^S$, then we begin with split move S' that splits $\left(y^S - y^M\right)$ portion of mass from index ℓ. This step is feasible as $0 \leq x^2_\ell = x^0_\ell + y^M - y^S$ and hence $\left(y^S - y^M\right) \leq x^0_\ell$. We next show that the merge move M can be split into two merge moves M'_1, M'_2 such that both are feasible and push $\frac{1}{2}y^M$ amount of mass to $\ell-1, \ell+1$ respectively. Hence, performing S', M'_1, M'_2 gets from x_i^0 to x_i^2 as well. We need to show that there exist $0 \leq y'_i \leq y_i^M$ such that $\sum_i y'_i = \frac{1}{2}y^M$ and $\left(\sum_i i y'_i\right) / \left(\sum_i y'_i\right) = \ell + 1$. If so, then we can define M'_1 by y'_i and M'_2 by $y_i^M - y'_i$. By applying a linear transformation to all indices i (and to ℓ) we may assume without loss of generality that $\ell = 0,\ \sum_i y_i^M = 1$ and we need to find $0 \leq y'_i \leq y_i^M$ such that $\sum_i y'_i = \frac{1}{2},\ \left(\sum_i i y'_i\right) / \left(\frac{1}{2}\right) = 1$. We may think of y_i^M as a distribution over indices i. Let $\text{Median}\left(y_i^M\right)$ be the median of this distribution. Since $y_0^M = 0$ and the median must be in the support of the distribution, we have $\left|\text{Median}\left(y_i^M\right)\right| \geq 1$. We may also assume without loss of generality that $\text{Median}\left(y_i^M\right) \geq 1$ (otherwise, replace the roles of i and $-i$ and then of y'_i and $y_i^M - y'_i$). We find suitable values for y'_i using the following continuous process. Begin with the feasible values $y'_i = \frac{1}{2}y_i^M$. For these initial values we clearly have $0 \leq y'_i \leq y_i^M,\ \sum_i y'_i = \frac{1}{2},\ \sum_i i y'_i = \frac{1}{2}\ell = 0$. We now continuously move mass from the lowest index j_1 such that $y'_{j_1} > 0$ to the highest index j_2 such that $y'_{j_2} < y^M_{j_2}$ (i.e., we move mass from the left-most non-empty bucket to the right-most non-full bucket in the distribution). Throughout the process, it is maintained that $0 \leq y'_i \leq y_i^M,\ \sum_i y'_i = \frac{1}{2}$. As we always move mass from lower to higher indices ($j_1 < j_2$) the quantity $\sum_i i y'_i$ increases throughout the process. If we would carry on with the process until no longer possible, we would end up with the mass fully being in indices larger or equal to $\text{Median}\left(y_i^M\right)$ and in particular at this point we have $\sum_i i y'_i \geq \frac{1}{2}\text{Median}\left(y_i^M\right) \geq \frac{1}{2}$. In particular, by continuity, there is some point throughout the process in which we have exactly $\sum_i i y'_i = \frac{1}{2}$, as we wanted.

We are left to deal with the case that $y^M \geq y^S$. In that case we actually do not need any split move (as $x^2_\ell \geq x^0_\ell$). Instead, we remove the split move (or replace it with a split move that splits no mass) and partition the merge move M into three merge moves M'_1, M'_2, M'_3. We let M'_1 be simply a scaling down of M to a mass of $0 \leq y^M - y^S \leq y^M$. The rest y^S of the portion of mass to be merged is partitioned to two merges M'_2, M'_3 each merging $\frac{1}{2}y^S$ portion of mass to $\ell - 1, \ell + 1$ respectively, in the same manner as in the previous case. □

Theorem 5.4. *Given an execution of MSMG of length T ending with expected absolute value v, there exists an execution of MSG of length $T' \leq T$ ending with expected absolute value $v' \geq v$.*

Proof. Consider the execution of the MSMG. As long as there is a merge move followed by a split move, we apply the transformation of Lemma 5.3. This does not change the rest of the game execution and in

particular not the final value. This process must end in finite time as each application of Lemma 5.3 does not change the number of split moves yet strictly reduces the lexicographical order of the set of indices of split moves. After such transitions are no longer possible, the execution must consist of only split moves followed by only merge moves. As merge moves only reduces the expected absolute value, the prefix of the executions that contains only the split moves satisfies the Theorem's statement. □

5.3 Lower bound for MSG

Due to Section 5.2 to prove Theorem 5.2 it is enough to prove the following.

Theorem 5.5. *Let x_i^t be an execution of a MSG. Suppose that*

$$\sum_i |i| \cdot x_i^T > \delta \cdot k.$$

Then there have been $> c \cdot \delta^3 \cdot k^3$ split moves, where $c > 0$ is a universal constant.

In fact, we first prove a similar result for a seemingly weaker variant of value. Consider the following trimmed formulation of the *Mass Splitting Game* (MSG). We have $2k + 1$ variables $x_{-k}, x_{-(k-1)}, \ldots, x_{-1}, x_0, x_1, \ldots, x_{k-1}, x_k$. Initially, the value of x_0 is 1 and the value of x_i for every other $-k \le i \le k, i \ne 0$ is 0. The game also has a *value* v which is initially 0. In each *step*, the player picks a variable index $-k \le i \le k$ and an amount $0 \le m \le x_i$ of *mass* that is currently in the variable x_i. The mass m is removed from x_i and is split evenly between its predecessor and successor. That is, after the step the new values of x_{i-1}, x_i, x_{i+1} are $x_{i+1} + \frac{m}{2}, x_i - m, x_i + \frac{m}{2}$. We abuse notation by setting $x_{-(k+1)} = x_{k+1} = v$. That is, if the player chooses $i = -k$ or $i = k$ then half of the mass m is added to the *value* of the game. The objective of the player is to reach a constant *value* v in the lowest number of steps. In this section we prove the following.

Theorem 5.6. *If a MSG strategy using T steps gets $v \ge \alpha$, then $T \ge \frac{\sqrt{\pi/2} \cdot \alpha^{3/2}}{32 \ln^{3/2}(8/\alpha)} k^3$.*

We first show that Theorem 5.5 follows from Theorem 5.6.

Proof of Theorem 5.5. Assume that $\sum_i |i| \cdot x_i^T > \delta \cdot k$. For every $j \ge 1$, denote by

$$p_j := \sum_{i \text{ s.t. } |i| \ge j} x_i^T$$

the total mass that ended out of $[-(j-1), (j-1)]$. We notice that

$$\sum_{i=-\infty}^{\infty} |i| \cdot x_i^T = \sum_{j=1}^{\infty} p_j.$$

By Theorem 5.6 there exists some constant c' such that if a MSG strategy[10] using T steps gets $\ge \alpha$ of mass out of $[-j, j]$ then $T \ge c' \cdot \alpha^2 j^3$. Denote by $c = \frac{1}{27} c'$. If there exists j such that $c' \cdot p_j^2 \cdot j^3 \ge c \cdot \delta^3 \cdot k^3$ then we are done by the last statement. Otherwise, for every j we have $p_j^2 < \frac{c}{c'} \cdot \frac{\delta^3 \cdot k^3}{j^3}$ and hence $p_j < \sqrt{\frac{c}{c'}} \cdot \frac{\delta^{3/2} \cdot k^{3/2}}{j^{3/2}}$. Therefore,

$$\delta \cdot k < \sum_{j=1}^{\infty} p_j \le \sum_{j=1}^{(c/c')^{1/3} \cdot \delta \cdot k} 1 \; + \sum_{j=(c/c')^{1/3} \cdot \delta \cdot k + 1}^{\infty} \sqrt{\frac{c}{c'}} \cdot \frac{\delta^{3/2} \cdot k^{3/2}}{j^{3/2}}$$

[10] we restrict the strategy to $\{-j+1, \ldots, 0, \ldots, j-1\}$ and add to the *value* whenever $j-1$ or $-j+1$ is split; ignoring any future split on the mass that has already been added to the *value*. This may affect the splits in $\{-j+1, \ldots, 0, \ldots, j-1\}$ and we perform these splits to the best extent possible, while ignoring the mass that had already been added to *value*.

$$\leq \left(\frac{c}{c'}\right)^{1/3} \cdot \delta \cdot k \; + \; \sqrt{\frac{c}{c'}} \cdot \delta^{3/2} \cdot k^{3/2} \cdot \sum_{j=(c/c')^{1/3} \cdot \delta \cdot k+1}^{\infty} \frac{1}{j^{3/2}}$$

$$\leq \left(\frac{c}{c'}\right)^{1/3} \cdot \delta \cdot k \; + \; \sqrt{\frac{c}{c'}} \cdot \delta^{3/2} \cdot k^{3/2} \cdot \frac{2}{\sqrt{(c/c')^{1/3} \cdot \delta \cdot k}}$$

$$= 3 \cdot \left(\frac{c}{c'}\right)^{1/3} \cdot \delta \cdot k = \delta \cdot k,$$

which is a contradiction. □

We begin by showing that in an optimal strategy, the player always chooses to split *all* of the mass in its chosen variable.

Lemma 5.7. *Let $S =< (i_1, m_1), (i_2, m_2), \ldots, (i_T, m_T) >$ be a MSG strategy of length $T = |S|$ getting value $v(S)$. There exists a strategy S' of length $T' = |S'| \leq T$ getting value $v(S') \geq v(S)$ in which every step splits the full mass, that is, for each j before the j-th step we have $m_j = x_j$.*

Proof. We prove the claim by induction on r, the number of steps j in which $m_j < x_j$. If $r = 0$ there is nothing to prove. Otherwise, consider the last step j in which $m_j < x_j$. We observe that for every $\ell > j$, the step $s_\ell = (i_\ell)$ that fully splits the mass of x_{i_ℓ} is a linear map M_{s_j} on $(x_{-k}, \ldots, x_k, v)$. In particular, as a composition of linear maps, the function $M := M_{s_T} \cdot \ldots \cdot M_{s_{j+1}}$ that maps the values of $(x_{-k}, \ldots, x_k, v)$ after the j-th step to those in the end of the game is linear, and so is its restriction M' to the value of v.

Let $(x_{-k}, \ldots, x_k, v)$ be the values of the variables and the game before the j-th step. After the step, these values are

$$(x_{-k}, \ldots, x_k, v) + (\ldots, 0, \frac{m_j}{2}, -m_j, \frac{m_j}{2}, 0, \ldots)$$

where in the added vector the non-zero coordinates are $x_{i_j-1}, x_{i_j}, x_{i_j+1}$. In particular, the value of the game in the end of it is

$$\begin{aligned} &M'\left((x_{-k}, \ldots, x_k, v) + (\ldots, 0, \frac{m_j}{2}, -m_j, \frac{m_j}{2}, 0, \ldots)\right) \\ =&M'(x_{-k}, \ldots, x_k, v) + m_j \cdot M'\left(\ldots, 0, \frac{1}{2}, -1, \frac{1}{2}, 0, \ldots\right). \end{aligned}$$

This is a linear function of m_j and thus in the interval $m_j \in [0, x_j]$ it must attain its maximum in either of the endpoints 0 or x_j. Thus, we can either remove the j-th step or increase m_j to x_j without lowering the value of the strategy. This reduces the number of such steps by one and thus proves the induction step. □

Using Lemma 5.7 we assume from now on that in each step the player just picks a variable index i and fully splits its mass. We now introduce a randomized variant of MSG, denoted by *rMSG*. In this variant, we initially place n *particles* on the 0-th variable. In each step, the player chooses an index i, and then each particle that currently resides in i randomly moves to either $i-1$ or $i+1$ uniformly. We again equate $-(k+1)$ and $k+1$ with v, and now consider the value of the game to be the fraction of particles that reached v. We may translate strategies from MSG to rMSG (and vice versa) in a natural way.

Observation 5.8. *Let S be a strategy of MSG with value $v(S)$. If we apply S to rMSG then the probability that a specific particle ends up in v is exactly $v(S)$.*

Intuitively, each particle behaves as a simple random walk and thus would need to move $\Omega(k^2)$ times in order to be likely to reach v. Furthermore, after $\Omega(k^2)$ moves a particle resides in a roughly uniformly random index and thus only a $\frac{1}{\Omega(k)}$ fraction of the particles would be in each index at every time. In particular, we would need $\Omega(k^3)$ steps to make each particle move $\Omega(k^2)$ times. The exact argument follows.

We fix a MSG strategy S and apply it to rMSG with n particles named $P_1, \ldots, P_n$. For each particle P_i denote by T_i the random variable counting the number of steps in which P_i was moved. For each pair of particles P_i, P_j denote by $T_{i,j}$ the random variable counting the number of steps in which *both* P_i and P_j were moved.

Lemma 5.9. *We have* $T \geq \sum_i T_i - \sum_{i,j} T_{i,j}$.

Proof. If r particles are moved in a step, then the change to $\sum_i T_i - \sum_{i,j} T_{i,j}$ in this step is $r - \binom{r}{2} \leq 1$. □

Lemma 5.10. *We have* $E[T_{1,2}] \leq \frac{2\sqrt{2}}{\sqrt{\pi}} \cdot \sqrt{E[T_1]}$.

Proof. Denote by $\tilde{T}_{1,2}$ the number of steps in which either P_1 or P_2 (or both) were moved. Denote these steps by $j_1, j_2, \ldots, j_{\tilde{T}_{1,2}}$. Denote the locations of P_1, P_2 after the j_ℓ-th step by $P_1^{(\ell)}, P_2^{(\ell)}$ respectively. Let $D_{1,2}^{(\ell)} := |P_1^{(\ell)} - P_2^{(\ell)}|$ be the distance between them. We note that the sequence $D_{1,2}$ behave almost as a uniform single dimensional random walk: If $D_{1,2}^{(\ell)} > 0$, then $D_{1,2}^{(\ell+1)}$ distributes uniformly between $D_{1,2}^{(\ell)} - 1$ and $D_{1,2}^{(\ell)} + 1$. If $D_{1,2}^{(\ell)} = 0$, then $D_{1,2}^{(\ell+1)}$ distributes uniformly between 0 and 2. In Appendix C.1 we show by standard techniques that the expected number of times such a random walk of length r visits the origin is at most $\frac{2}{\sqrt{\pi}}\sqrt{r}$. Thus,

$$E[T_{1,2}] = E[E[T_{1,2}|\tilde{T}_{1,2}]] \leq E\left[\frac{2}{\sqrt{\pi}}\sqrt{\tilde{T}_{1,2}}\right] \leq \frac{2}{\sqrt{\pi}}\sqrt{E[\tilde{T}_{1,2}]} \leq \frac{2}{\sqrt{\pi}}\sqrt{2E[T_1]},$$

where the second inequality follows from Jensen's Inequality and the last one is a union bound. □

Lemma 5.11. *We have* $T \geq \frac{\sqrt{\pi}}{4\sqrt{2}} \left(E[T_1]\right)^{\frac{3}{2}}$.

Proof. By Lemma 5.9 we have that always $T \geq \sum_i T_i - \sum_{i,j} T_{i,j}$. Therefore,

$$T \geq E[\sum_i T_i - \sum_{i,j} T_{i,j}] = \sum_i E[T_i] - \sum_{i,j} E[T_{i,j}] = nE[T_1] - \binom{n}{2} E[T_{1,2}].$$

Then, by Lemma 5.10 we have that

$$nE[T_1] - \binom{n}{2} E[T_{1,2}] \geq nE[T_1] - \binom{n}{2} \cdot \frac{2\sqrt{2}}{\sqrt{\pi}} \cdot \sqrt{E[T_1]}.$$

By choosing $n = \frac{\sqrt{E[T_1]}}{\frac{2\sqrt{2}}{\sqrt{\pi}}}$, the above expression is at least as large as

$$\frac{\sqrt{E[T_1]}}{\frac{2\sqrt{2}}{\sqrt{\pi}}} E[T_1] - \frac{1}{2}\left(\frac{\sqrt{E[T_1]}}{\frac{2\sqrt{2}}{\sqrt{\pi}}}\right)^2 \cdot \frac{2\sqrt{2}}{\sqrt{\pi}} \cdot \sqrt{E[T_1]} = \frac{\sqrt{\pi}}{4\sqrt{2}} \left(E[T_1]\right)^{\frac{3}{2}}.$$

□

Lemma 5.12. *If* $v(S) \geq \alpha$ *then* $E[T_1] \geq \frac{\alpha}{4\ln(8/\alpha)} k^2$.

Proof. Assume the contradiction $E[T_1] < \frac{\alpha}{4\ln(8/\alpha)} k^2$. By Observation 5.8, $v(S)$ is the probability that the particle P_1 ends up in v. By Markov's inequality, $Pr(T_1 > \frac{1}{2\ln(8/\alpha)} k^2) < \frac{\alpha}{2}$. Thus, the probability that P_1 reaches v conditioned on $T_1 \leq \frac{1}{2\ln(8/\alpha)} k^2$ is at least $v(s) - \frac{\alpha}{2} \geq \frac{\alpha}{2}$.

Let $j_1, \ldots, j_{T_1}$ be the steps in which P_1 was moved and $P_1^{(1)}, \ldots, P_1^{(T_1)}$ its location after each of these steps. The sequence of locations is a uniform one-dimensional random walk. We finish by noting that the probability of a simple random walk of length $\frac{1}{2\ln(8/\alpha)} k^2$ to reach $-(k+1)$ or $k+1$ is lower than $\frac{\alpha}{2}$, which is a contradiction. We include a proof of this fact in Appendix C.2. □

We finally conclude the proof of Theorem 5.6.

Proof. By the assumption that $v \geq \alpha$ and Lemma 5.12 we have $E[T_1] \geq \frac{\alpha}{4\ln(8/\alpha)}k^2$. From Lemma 5.11 we thus have

$$T \geq \frac{\sqrt{\pi}}{4\sqrt{2}}\left(E[T_1]\right)^{\frac{3}{2}} \geq \frac{\sqrt{\pi}}{4\sqrt{2}}\left(\frac{\alpha}{4\ln(8/\alpha)}k^2\right)^{\frac{3}{2}} = \frac{\sqrt{\pi/2}\cdot\alpha^{3/2}}{32\ln^{3/2}(8/\alpha)}k^3.$$

□

5.4 Reducing ROBP to MSMG

Recall the definition of a Skip-Forward Read-Once Program (SF-ROP) from Section 3.1. We observed that a standard ROBP of width w can be cast as a SF-ROP of size $(w \cdot n + 1)$. Therefore, a lower bound s on the size of SF-ROP would translate into a lower bound of s/n on w for ROBP.

From now on we fix a parameter $\varepsilon > 0$, fix $B \in_U \{-1, 1\}$ be the random variable representing the sign of the bias of the coin. Let b be the value taken by B. The input x conditioned on $B = b$ is distributed as $X|_{B=b} \sim \mathrm{Ber}(1/2 + b \cdot \varepsilon)^n$ — n i.i.d. copies of $(b \cdot \varepsilon)$-biased coin tosses.

Let P be a SF-ROP program on input X as above (whose goal is to guess the value of B). For each state $s \in P$, define the probability estimate of the value of B as

$$\beta(s) := \Pr[B = 1 \mid P(x) \text{ reaches state } s].$$

Denote also

$$p(s) := \Pr[P(x) \text{ reaches state } s].$$

Note that $\beta(S_1) = 1/2$ (where S_1 is the start state), and the following claim holds:

Claim 5.13. *The advantage of P at determining the value of B is given by*

$$\mathsf{ADV}(P) = \mathbb{E}_{\textit{Execution of the program reaches } t}\left[2 \cdot |\beta(t) - 1/2|\right] \tag{14}$$

It will also be convenient to denote the bias of an edge in the SF-ROP: for an edge $s \to s'$ between two states, denote by

$$\beta(s \to s') := \Pr[B = 1 \mid P(x) \text{ reaches state } s \text{ and then takes the edge } s \to s'],$$

and

$$p(s \to s') := \Pr[P(x) \text{ reaches state } s \text{ and then takes the edge } s \to s'].$$

Note that the following formulas hold by Bayes rule ($s_0 = P(s, 0)$ and $s_1 = P(s, 1)$):

$$\beta(s \to s_0) = \frac{\beta(s)\cdot(1-2\varepsilon)}{1+2\varepsilon-4\varepsilon\beta(s)} \quad \text{and} \quad \beta(s \to s_1) = \frac{\beta(s)\cdot(1+2\varepsilon)}{1-2\varepsilon+4\varepsilon\beta(s)} \tag{15}$$

If we assume that $\varepsilon < 0.1$, then we have

$$\beta(s \to s_0),\ \beta(s \to s_1) \in (\beta(s) - 5\varepsilon, \beta(s) + 5\varepsilon) \tag{16}$$

To complete our proof we will need some results about manipulating states in the mass split-and-merge game. A configuration $z : \{-k, \ldots, k\} \to \mathbb{R}^+$ is an allocation of mass (which does not necessarily adds up to 1). We define a partial order relationship on configurations:

Definition 5.14. *We say that $z \prec y$ if z can be obtained from y by a finite sequence of merge moves.*

For a real number $a \in (-k, k)$ define the configuration

$$c(a)_i := \begin{cases} 1 - \{a\} & \text{if } i = \lfloor a \rfloor \\ \{a\} & \text{if } i = \lfloor a \rfloor + 1 \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

Here, $\{a\}$ represents the fractional part of a, that is $\{a\} = a - \lfloor a \rfloor$.

Claim 5.15.

1. *If $z \prec y$ then their mass and centers of mass match:*
$$\sum_i z_i = \sum_i y_i, \quad \sum_i i \cdot z_i = \sum_i i \cdot y_i;$$
2. *For any configuration x scalar $a \geq 0$, $z \prec y$ implies $x + z \prec x + y$ and $a \cdot z \prec a \cdot y$;*
3. *for each z there exists a unique reduced configuration $r(z)$ such that $r(z)$ is supported on either 1 or 2 adjacent values and such that $r(z) \prec z$;*
4. *for any $a, b \in (-k, k)$ and $\lambda \in [0, 1]$,*
$$c(\lambda \cdot a + (1-\lambda) \cdot b) \prec \lambda \cdot c(a) + (1-\lambda) \cdot c(b);$$
5. *Suppose $-k < x - 1 < y \leq x \leq z < x + 1 < k$ are numbers such that $x = \lambda \cdot y + (1-\lambda) \cdot z$. Then*
$$\lambda \cdot c(y) + (1-\lambda) \cdot c(z) \prec \frac{1}{2} \cdot c(x-1) + \frac{1}{2} \cdot c(x+1).$$

Proof. Each merge move (13) preserves the total mass of the configuration, and its center of mass. To see that the center of mass is preserved, observe that after the merge step given by (13),
$$\sum_j j \cdot (x_j^t - x_j^{t+1}) = \sum_j j \cdot y_j - \ell \cdot \sum_i = \sum_j j \cdot y_j - \sum_i i \cdot y_i = 0.$$
By induction on the merge steps leading from y to z, this proves the **first** statement.

The **second** statement follows by observing that the same sequence of merges that leads from y to z also leads from $x + y$ to $x + z$. If we multiply each weight in the sequence of merges leading from y to z by a, we will get a sequence of merges leading from $a \cdot y$ to $a \cdot z$.

For any configuration y let max_y and min_y be the largest and smallest indexes i such that $y_i > 0$. If $max_y - min_y \geq 2$, we can perform a merge move that will reduce either y_{max_y} or y_{min_y} to 0, thus reducing the value of $max_y - min_y$ by at least 1. Starting with a configuration z, continue this process until we reach a configuration $r(z)$ with $max_{r(z)} - min_{r(z)} \leq 1$. This establishes the existence part of the **third** statement. Uniqueness follows from the first statement above, since the values of $\sum_i r_i$, $\sum_i i \cdot r_i$ and the fact that $max_r - min_r \leq 1$ uniquely determine r.

Note that the first and third properties so far imply that whenever $\sum_i y_i = \sum_i z_i$ and $\sum_i i \cdot y_i = \sum_i i \cdot z_i$, we have
$$r(y) = r(z).$$
Also, observe that $r(c(a)) = c(a)$, since $c(a)$ is already in a reduced form. Therefore
$$r(x) = \left(\sum_i x_i \right) \cdot c \left(\sum_i i \cdot x_i \right). \tag{18}$$
Plugging $\lambda \cdot c(a) + (1-\lambda) \cdot c(b)$ into (18), we get the **fourth** statement:
$$\lambda \cdot c(a) + (1-\lambda) \cdot c(b) \succ r(\lambda \cdot c(a) + (1-\lambda) \cdot c(b)) = 1 \cdot c(\lambda \cdot a + (1-\lambda) \cdot b) = c(\lambda \cdot a + (1-\lambda) \cdot b).$$

To establish the **fifth** statement, we repeatedly apply the fourth one, writing $y = (x-y) \cdot (x-1) + (y - x + 1) \cdot x$ and $z = (z-x) \cdot (x+1) + (x+1-z) \cdot x$, $x = \frac{1}{2} \cdot (x-1) + \frac{1}{2} \cdot (x+1)$:
$$\lambda \cdot c(y) + (1-\lambda) \cdot c(z) \prec \lambda \cdot (x-y) \cdot c(x-1) + \lambda \cdot (y-x+1) \cdot c(x) + (1-\lambda) \cdot (x+1-z) \cdot c(x) + (1-\lambda) \cdot (z-x) \cdot c(x+1)$$

$$\prec \left[\lambda\cdot(x-y)+\frac{1}{2}\cdot\lambda\cdot(y-x+1)+\frac{1}{2}\cdot(1-\lambda)\cdot(x+1-z)\right]\cdot c(x-1)+$$
$$\left[\frac{1}{2}\cdot\lambda\cdot(y-x+1)+\frac{1}{2}\cdot(1-\lambda)\cdot(x+1-z)+(1-\lambda)\cdot(z-x)\right]\cdot c(x+1)=$$
$$\left[\frac{x}{2}-\frac{\lambda\cdot y}{2}-\frac{(1-\lambda)\cdot z}{2}+\frac{1}{2}\right]\cdot c(x-1)+\left[-\frac{x}{2}+\frac{\lambda\cdot y}{2}+\frac{(1-\lambda)\cdot z}{2}+\frac{1}{2}\right]\cdot c(x+1)$$
$$=\frac{1}{2}\cdot c(x-1)+\frac{1}{2}\cdot c(x+1).$$

□

Our main lemma is as follows.

Lemma 5.16. *Suppose there is a size-ℓ SF-ROP that gains advantage δ on the coin problem with bias ε. Let $k := 1 + \lfloor 1/(20\varepsilon)\rfloor$. Then there is a valid MSMG on $\{-k,\dots,k\}$ that uses at most 2ℓ split moves and achieves*

$$\sum_i |i|\cdot x_i^T > \delta\cdot k/2.$$

Together with Theorem 5.2, Lemma 5.16 implies an $\Omega(\varepsilon^{-3})$ lower bound on the size of an SF-ROP solving the ε-biased coin problem. Therefore, we get that a ROBP solving the ε-biased coin problem must have width

$$w = \Omega(\varepsilon^{-3}/n).$$

In particular, if $\varepsilon = n^{-1/3-\eta}$ for a constant η, we get a lower bound of $\Omega(n^{3\eta})$ on the width, and therefore an $\Omega(\log n)$ lower bound on the memory required.

Proof of Lemma 5.16. First, we map biases to real numbers in $[-k+1, k-1]$. For a $\beta\in[0,1]$ let

$$f(\beta) := (\beta - 1/2)/(10\varepsilon).$$

Let $S_1,\dots,S_\ell$ be the states of the SF-ROP in the order that they appear. To streamline the argument, we assume that the terminal states are repeated twice among the last four states: $S_{\ell-3}=t_0$, $S_{\ell-2}=t_1$, and there is a probability 1 transition from $S_0\to S_1$, $S_{\ell-3}\to S_{\ell-1}$ and $S_{\ell-2}\to S_\ell$ (S_0 is an additional start state).

For any $t=0,\dots,\ell-2$, define X^t as follows:

$$X^t := \sum_{i\le t, j>t} p(S_i\to S_j)\cdot c(f(\beta(S_i\to S_j))). \tag{19}$$

The only transition out of S_0 is to S_1, thus X^0 has all the mass on $X^0_0=1$. Further, for all t, $\sum_i X^t_i = 1$, and at $t=\ell-2$ we have

$$X^{\ell-2} = p(t_0)\cdot c(f(\beta(t_0))) + p(t_1)\cdot c(f(\beta(t_1))),$$

therefore

$$\sum_i |i|\cdot X_i^{\ell-2} \ge p(t_0)\cdot|f(\beta(t_0))| + p(t_1)\cdot|f(\beta(t_1))| = \frac{\mathsf{ADV}(P)}{20\varepsilon} > \delta\cdot k/2.$$

It remains to see that $X^{\ell-2}$ is obtained from X^0 using at most 2ℓ split moves. To this end, we will see that for any t, X^t can be obtained from X^{t-1} using 2 split moves (and some merge moves).

The move from X^{t-1} to X^t involves moving the mass

$$M^- = \sum_{i<t} p(S_i\to S_t)\cdot c(f(\beta(S_i\to S_t)))$$

to

$$M^+ = \sum_{j<t} p(S_t \to S_j) \cdot c(f(\beta(S_t \to S_j))).$$

Denote $p := p(S_t)$ and

$$x := p(S_t)^{-1} \cdot \sum_{i<t} p(S_i \to S_t) \cdot f(\beta(S_i \to S_t))$$

By Claim 5.15 we have $p \cdot c(x) \prec M^-$. Therefore, it suffices to show that M^+ can be obtained from $p \cdot c(x)$ using two splits. Let S^0 be the state to which S_t goes when it reads 0 answer ($S^0 = P(S_t, 0)$) and S^1 be the state to which S_t goes when it reads 1 answer ($S^1 = P(S_t, 1)$). Let

$$\lambda := p(S_t \to S^0)/p(S_t).$$

We have

$$M^+ = p \cdot \left[\lambda \cdot c(f(\beta(S_t \to S^0))) + (1-\lambda) \cdot c(f(\beta(S_t \to S^1)))\right].$$

We have $x - 1 < f(\beta(S_t \to S^0)) \le x \le f(\beta(S_t \to S^1)) < x + 1$. Therefore, by Claim 5.15, we have

$$M^+ \prec \frac{1}{2} \cdot p \cdot c(x-1) + \frac{1}{2} \cdot p \cdot c(x+1).$$

To complete the proof, observe that $\frac{1}{2} \cdot p \cdot c(x-1) + \frac{1}{2} \cdot p \cdot c(x+1)$ is obtained from $p \cdot c(x)$ using one or two split operations (on the points constituting $c(x)$). □

References

[Aar10] Scott Aaronson. BQP and the polynomial hierarchy. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 141–150, 2010.

[ABO84] Miklós Ajtai and Michael Ben-Or. A theorem on probabilistic constant depth computations. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 471–474, 1984.

[Ajt83] Miklós Ajtai. $\sum_1^1$-formulae on finite structures. *Annals of pure and applied logic*, 24(1):1–48, 1983.

[Ama09] Kazuyuki Amano. Bounds on the size of small depth circuits for approximating majority. In *International Colloquium on Automata, Languages, and Programming*, pages 59–70. Springer, 2009.

[BGW20] Mark Braverman, Sumegha Garg, and David P Woodruff. The coin problem with applications to data streams. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 318–329. IEEE, 2020.

[Bop85] Ravi B Boppana. Amplification of probabilistic boolean formulas. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 20–29. IEEE, 1985.

[BRRY14] Mark Braverman, Anup Rao, Ran Raz, and Amir Yehudayoff. Pseudorandom generators for regular branching programs. *SIAM Journal on Computing*, 43(3):973–986, 2014.

[BV10] Joshua Brody and Elad Verbin. The coin problem and pseudorandomness for branching programs. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 30–39. IEEE, 2010.

[CGR14] Gil Cohen, Anat Ganor, and Ran Raz. Two sides of the coin problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[Che52] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952.

[CSV15] Sitan Chen, Thomas Steinke, and Salil Vadhan. Pseudorandomness for read-once, constant-depth circuits. *arXiv preprint arXiv:1504.04675*, 2015.

[DZ97] Moshe Dubiner and Uri Zwick. Amplification by read-once formulas. *SIAM Journal on Computing*, 26(1):15–38, 1997.

[FSS84] Merrick Furst, James B Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical systems theory*, 17(1):13–27, 1984.

[HC70] Martin E Hellman and Thomas M Cover. Learning with finite memory. *The Annals of Mathematical Statistics*, pages 765–782, 1970.

[KNW10] Daniel M Kane, Jelani Nelson, and David P Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1161–1178. SIAM, 2010.

[LSS+19] Nutan Limaye, Karteek Sreenivasaiah, Srikanth Srinivasan, Utkarsh Tripathi, and S Venkitesh. A fixed-depth size-hierarchy theorem for AC0[$\oplus$] via the coin problem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 442–453, 2019.

[LV18] Chin Ho Lee and Emanuele Viola. The coin problem for product tests. *ACM Transactions on Computation Theory (TOCT)*, 10(3):1–10, 2018.

[MNSW98] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.

[Raz87] Alexander A Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[Rob55] Herbert Robbins. A remark on Stirling's formula. *The American mathematical monthly*, 62(1):26–29, 1955.

[RSV13] Omer Reingold, Thomas Steinke, and Salil Vadhan. Pseudorandomness for regular branching programs via fourier analysis. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 655–670. Springer, 2013.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 77–82, 1987.

[Ste13] John Steinberger. The distinguishability of product distributions by read-once branching programs. In *2013 IEEE Conference on Computational Complexity*, pages 248–254. IEEE, 2013.

[SV10] Ronen Shaltiel and Emanuele Viola. Hardness amplification proofs require majority. *SIAM Journal on Computing*, 39(7):3122–3154, 2010.

[Val84] Leslie G. Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5(3):363–366, 1984.

A Proofs from Section 4

A.1 Facts about Bias-Amplifying Random Process $RP(k, k', p)$

Proof of Claim 4.1. Let S_j ($j \in \{-k', \ldots, k'\}$) be the set of all k-length sequences $x = (x_1, \ldots, x_k) \in \{-1, 1\}^k$ such that $s_k(x) = j$. By symmetry,

$$\forall x, x \in S_j \iff \text{flip}(x) \in S_{-j}.$$

Recall that $\text{flip}(x_1, x_2, \ldots, x_k) = (-x_1, -x_2, \ldots, -x_k)$. Let $|x|$ denote the number of 1s in the sequence x. Fix $j \in [k' - 1]$. We have that $\forall x \in S_j$, $\sum_{i=1}^{k} x_i = 2|x| - k = j$ (either the process hits k' or $-k'$, or ends up at the sum of the bits). Let $q(x)$ be the probability that the input to the random process $RP(k, k', p)$ is x, and as x_is are *i.i.d.* Ber(p),

$$q(x) = p^{|x|}(1-p)^{k-|x|}.$$

Thus,

$$\frac{q(x)}{q(\text{flip}(x))} = \frac{p^{|x|}(1-p)^{k-|x|}}{p^{k-|x|}(1-p)^{|x|}} = \left(\frac{p}{1-p}\right)^j. \qquad (2|x| - k = j)$$

Therefore,

$$\frac{\Pr[s_k = j]}{\Pr[s_k = -j]} = \frac{\sum_{x \in S_j} q(x)}{\sum_{x \in S_{-j}} q(x)} = \frac{\sum_{x \in S_j} q(x)}{\sum_{x \in S_j} q(\text{flip}(x))} = \left(\frac{p}{1-p}\right)^j.$$

Next, we look at $\frac{\Pr[s_k = k']}{\Pr[s_k = -k']}$. Note that $\forall x \in S_{k'}$, there exists step $i_x \in [k]$ such that $s_{i_x}(x) = k'$ ($\forall i > i_x,\ s_i(x) = k'$) and $\forall i < i_x,\ s_i(x) < k'$. Let $\text{flip}_j(x)$ represent the sequence where only the first j bits in the sequence are flipped, that is, $\text{flip}_j(x) = (-x_1, \ldots, -x_j, x_{j+1}, \ldots, x_k)$. It is easy to see that $\forall x \in S_{k'}, \text{flip}_{i_x}(x) \in S_{-k'}$. As $\forall x \in S_{k'}$, $\sum_{i=1}^{i_x} x_i = k'$,

$$\frac{q(x)}{q(\text{flip}_{i_x}(x))} = \frac{\Pr_{\forall i, x'_i \sim \text{Ber}(p)}[x' = x]}{\Pr_{\forall i, x'_i \sim \text{Ber}(p)}[x' = \text{flip}_{i_x}(x)]} = \frac{p^{k'}}{(1-p)^{k'}},$$

and as flip_{i_x} defines a bijection from sequences in $S_{k'}$ to sequences in $S_{-k'}$, we get

$$\Pr[s_k = k'] = \frac{p^{k'}}{(1-p)^{k'}} \Pr[s_k = -k']. \ \square$$

Proof of 4.2. We assume $0 \le \delta < \frac{1}{2}$ as the proposition is trivial to prove for $\delta = \frac{1}{2}$. Note that as k is odd, $\Pr[s_k > 0] + \Pr[s_k < 0] = 1$. Under the random process $RP\left(k, k', \frac{1}{2} + \delta\right)$,

$$\begin{aligned}
\Pr[s_k > 0] - \Pr[s_k < 0] &= \sum_{j=1}^{k'} \Pr[s_k = j] - \Pr[s_k = -j] \\
&= \sum_{j=1}^{k'} (\Pr[s_k = j] + \Pr[s_k = -j]) \cdot \left(\frac{\frac{\Pr[s_k = j]}{\Pr[s_k = -j]} - 1}{\frac{\Pr[s_k = j]}{\Pr[s_k = -j]} + 1} \right) \\
&= \sum_{j=1}^{k'} (\Pr[s_k = j] + \Pr[s_k = -j]) \cdot \left(\frac{\left(\frac{\frac{1}{2}+\delta}{\frac{1}{2}-\delta}\right)^j - 1}{\left(\frac{\frac{1}{2}+\delta}{\frac{1}{2}-\delta}\right)^j + 1} \right) \qquad \text{(Claim 4.1)}
\end{aligned}$$

$$\geq \sum_{j=1}^{k'} (\Pr[s_k = j] + \Pr[s_k = -j]) \cdot \left(\frac{\left(\frac{\frac{1}{2}+\delta}{\frac{1}{2}-\delta}\right) - 1}{\left(\frac{\frac{1}{2}+\delta}{\frac{1}{2}-\delta}\right) + 1} \right)$$

$$(j, k' \geq 1, \left(\frac{\frac{1}{2}+\delta}{\frac{1}{2}-\delta}\right)^j \text{ increases with } j, \text{ and } \frac{t-1}{t+1} = 1 - \frac{2}{t+1} \text{ increases with } t \text{ for } t > 0)$$

$$= 2\delta$$

As $\Pr[s_k > 0] + \Pr[s_k < 0] = 1$, we get that $\Pr[s_k > 0] \geq \frac{1}{2} + \delta$ and $\Pr[s_k < 0] \leq \frac{1}{2} - \delta$. By symmetry, under the random process $RP\left(k, k', \frac{1}{2} - \delta\right)$, we get that $\Pr[s_k > 0] \leq \frac{1}{2} - \delta$. □

Proof of Lemma 4.3. As for odd k, $\Pr[s_k > 0] + \Pr[s_k < 0] = 1$, $\Pr[s_k > 0]$ under the random process $RP(k, k', \frac{1}{2} + \delta)$ can be rewritten as follows:

$$\Pr[s_k > 0] = \frac{1}{2}\left(1 + \Pr[s_k > 0] - \Pr[s_k < 0]\right)$$

$$= \frac{1}{2}\left(1 + \sum_{j=1}^{k'} \Pr[s_k = j] - \Pr[s_k = -j]\right) \tag{20}$$

Claim 4.1 states that $\forall j \in [k']$, $\frac{\Pr[s_k = j]}{\Pr[s_k = -j]} = \left(\frac{p}{1-p}\right)^j$ under the random process $RP(k, k', p)$. Thus, for $p = \frac{1}{2} + \delta$, $\forall j \in [k' - 1]$, $\Pr[s_k = j] > \Pr[s_k = -j]$. Coming back to Equation (20), we get (for $0 < \delta \leq \frac{1}{\sqrt{k}}$) the following:

$$\Pr[s_k > 0] > \frac{1}{2}\left(1 + \Pr[s_k = k'] - \Pr[s_k = -k']\right)$$

$$= \frac{1}{2} + \frac{\Pr[s_k = k']}{2}\left(1 - \frac{\Pr[s_k = -k']}{\Pr[s_k = k']}\right)$$

$$= \frac{1}{2} + \frac{\Pr[s_k = k']}{2}\left(1 - \frac{(1/2 - \delta)^{k'}}{(1/2 + \delta)^{k'}}\right) \qquad \text{(Claim 4.1)}$$

$$\geq \frac{1}{2} + \frac{\Pr[s_k = k']}{2}\left(1 - (1 - 2\delta)^{k'}\right)$$

$$\geq \frac{1}{2} + \frac{\Pr[s_k = k']}{2}\left(1 - e^{-2\delta k'}\right) \qquad (\forall x, 1 - x \leq e^{-x})$$

$$\geq \frac{1}{2} + \frac{\Pr[s_k = k']}{2}\left(1 - (1 - \delta k')\right) \qquad (\forall x \in [0,1], e^{-x} \leq 1 - \tfrac{x}{2} \text{ and } 2\delta k' \leq 2\left\lfloor \tfrac{1}{2}\sqrt{k} \right\rfloor \cdot \tfrac{1}{\sqrt{k}} \leq 1)$$

$$= \frac{1}{2} + \Pr[s_k = k'] \cdot \frac{k'}{2} \cdot \delta$$

We prove that $\Pr[s_k = k'] \geq \frac{1}{4}$ for $k' = \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor$ and any $p > \frac{1}{2}$. Thus, continuing the calculations above, we get the following bound on $\Pr[s_k > 0]$, assuming that $k \geq c_{amp} \geq 100$:

$$\Pr[s_k > 0] > \frac{1}{2} + \frac{1}{8} \cdot \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor \cdot \delta \geq \frac{1}{2} + \frac{1}{20}\sqrt{k} \cdot \delta \tag{21}$$

Next, we lower bound $\Pr[s_k = k']$ for $k' = \left\lfloor \frac{1}{2}\sqrt{k} \right\rfloor$ as follows:

$$\Pr[s_k = k'] \geq \Pr_{\forall i \in [k], x_i \sim \mathrm{Ber}(p)} \left[\sum_i x_i \geq k' \right]$$

$$\geq \Pr_{\forall i \in [k], x_i \sim \mathrm{Ber}(1/2)} \left[\sum_i x_i \geq k' \right] \qquad (k' > 0,\ p > 1/2)$$

$$
\begin{aligned}
&= \frac{1}{2}\left(1 - \sum_{j\in[k'] \text{ and } j \text{ is odd}} \Pr_{\forall i\in[k], x_i\sim \mathrm{Ber}(1/2)}\left[\sum_i x_i = j\right]\right) && (k \text{ is odd})\\
&= \frac{1}{2}\left(1 - \frac{1}{2^k}\sum_{j\in[k'] \text{ and } j \text{ is odd}} \binom{k}{\frac{k+j}{2}}\right)\\
&\geq \frac{1}{2}\left(1 - \frac{1}{2^k}\cdot k' \cdot \binom{k}{\frac{k+1}{2}}\right)\\
&\geq \frac{1}{2}\left(1 - k'\cdot\frac{1}{\sqrt{k}}\right) && \text{(see calculations below)}\\
&\geq \frac{1}{4} && (k' = \left\lfloor \tfrac{1}{2}\sqrt{k}\right\rfloor \leq \tfrac{1}{2}\sqrt{k})
\end{aligned}
$$

Using Striling's approximation, we get the second last inequality (for $k \geq c_{amp} \geq 4$) as follows:

$$
\begin{aligned}
\frac{1}{2^k}\binom{k}{\frac{k+1}{2}} &= \frac{1}{2^k}\frac{k!}{\frac{k+1}{2}!\frac{k-1}{2}!}\\
&\leq \frac{1}{2^k}\frac{ek^{k+\frac{1}{2}}}{2\pi\left(\frac{k+1}{2}\right)^{\frac{k}{2}+1}\left(\frac{k-1}{2}\right)^{\frac{k}{2}}}\\
&= \frac{e}{\pi}\cdot\frac{k^k}{(k^2-1)^{k/2}}\cdot\frac{\sqrt{k}}{k+1}\\
&\leq \frac{e}{\pi}\cdot\frac{1}{\left(1-\frac{1}{k^2}\right)^{k/2}}\cdot\frac{1}{\sqrt{k}}\\
&\leq \frac{e}{\pi}\cdot\frac{1}{\left(1-\frac{1}{2k}\right)}\cdot\frac{1}{\sqrt{k}} && (\left(1-\tfrac{1}{k^2}\right)^{k/2} \geq 1-\tfrac{k}{2k^2})\\
&\leq \frac{1}{\sqrt{k}} && (\text{as } k\geq 4 \text{ and } \tfrac{8e}{7\pi}\leq 1)
\end{aligned}
$$

Thus, $\Pr[s_k < 0] < \frac{1}{2} - \frac{1}{20}\sqrt{k}\delta$. By symmetry, under the random process $RP\left(k, \left\lfloor\frac{1}{2}\sqrt{k}\right\rfloor, \frac{1}{2}-\delta\right)$, we get that $\Pr[s_k > 0] < \frac{1}{2} - \frac{1}{20}\sqrt{k}\delta$ (for $0 < \delta \leq \frac{1}{\sqrt{k}}$ and $k \geq ck$). □

Proof of Claim 4.4. Claim follows easily using Chernoff bound[Che52]. $Y_i = \frac{X_i+1}{2}$ are independent random variables, each taking value 1 with probability p and 0 with probability $1-p$. For $p \geq \frac{1}{2}+\frac{1}{r}$,

$$
\begin{aligned}
\Pr\left[\sum_{i=1}^m X_i < 0\right] &= \Pr\left[\sum_{i=1}^m Y_i < \frac{m}{2}\right]\\
&\leq \Pr\left[\sum_{i=1}^m Y_i < \left(1-\frac{1}{r}\right)\cdot\mathbb{E}\sum_{i=1}^m Y_i\right] && (\mathbb{E}\textstyle\sum_{i=1}^m Y_i \geq \frac{m}{2}+\frac{m}{r}, \text{ and } \frac{m}{2r} \geq \frac{m}{r^2} \text{ as } r\geq 2)\\
&\leq e^{-\frac{1}{2r^2}\cdot\mathbb{E}\sum_{i=1}^m Y_i} && \text{(Chernoff bound)}\\
&\leq e^{-\frac{1}{2r^2}\cdot\frac{m}{2}} && (\mathbb{E}\textstyle\sum_{i=1}^m Y_i \geq \frac{m}{2})\\
&\leq e^{-\frac{1}{2r^2}\cdot\frac{20r^2}{2}}\\
&< 0.01
\end{aligned}
$$

When $p \leq \frac{1}{2}-\frac{1}{r}$, we take $Y_i = \frac{1-X_i}{2}$ and then use the same inequality as above.

$$\Pr\left[\sum_{i=1}^m X_i > 0\right] = \Pr\left[\sum_{i=1}^m Y_i < \frac{m}{2}\right] < 0.01. \ \square$$

A.2 Bias-Amplification Recursion

Proof of Lemma 4.8. Using the recursion, we get that $n_i \le r^i$ (where $r = k\left(2\left\lfloor \frac{1}{2}\sqrt{k}\right\rfloor + 1\right)$), $w_i \le i \cdot r$ and $\delta_i = \min\left(\left(\frac{1}{20}\sqrt{k}\right)^i \cdot \delta, \frac{1}{\sqrt{k}}\right)$.

We prove the bound on δ_i using induction. For $i = 0$ (base case), $\delta_0 = \min\left(\delta, \frac{1}{\sqrt{k}}\right)$. Next, assuming that $\delta_{i-1} = \min\left(\left(\frac{1}{20}\sqrt{k}\right)^{i-1} \delta, \frac{1}{\sqrt{k}}\right)$ (induction case), we prove the following.

$$\begin{aligned}
\delta_i &= \min\left(\frac{1}{20}\sqrt{k} \cdot \delta_{i-1}, \frac{1}{\sqrt{k}}\right) \\
&= \min\left(\frac{1}{20}\sqrt{k} \cdot \min\left(\left(\frac{1}{20}\sqrt{k}\right)^{i-1} \delta, \frac{1}{\sqrt{k}}\right), \frac{1}{\sqrt{k}}\right) && \text{(induction case)} \\
&= \min\left(\frac{1}{20}\sqrt{k} \cdot \left(\frac{1}{20}\sqrt{k}\right)^{i-1} \cdot \delta, \frac{1}{20}\sqrt{k} \cdot \frac{1}{\sqrt{k}}, \frac{1}{\sqrt{k}}\right) \\
&= \min\left(\left(\frac{1}{20}\sqrt{k}\right)^{i} \cdot \delta, \frac{1}{\sqrt{k}}\right) && (\sqrt{k} \ge 20)
\end{aligned}$$

Equation (11) easily follows from the fact that $r = 2\left\lfloor \frac{1}{2}\sqrt{k}\right\rfloor \le \sqrt{k}$ and $j = \left\lfloor \frac{\log n - \log(20k(40k+1))}{1.5\log k + 1}\right\rfloor \le \log n$. To derive Equation (10), we note that $2\left\lfloor \frac{1}{2}\sqrt{k}\right\rfloor + 1 \le \sqrt{k} + 1 \le 2\sqrt{k}$ and thus,

$$\begin{aligned}
n_j &\le r^{\left\lfloor \frac{\log n - \log(20k(40k+1))}{1.5\log k+1}\right\rfloor} \\
&\le (k \cdot 2\sqrt{k})^{\frac{\log n - \log(20k(40k+1))}{1.5\log k+1}} \\
&\le \frac{n}{(20k(40k+1))}.
\end{aligned}$$

Next, we show that for all $\delta \ge n^{-\frac{1}{3}+\epsilon}$ and large enough k and n, $\left(\frac{1}{20}\sqrt{k}\right)^j \delta \ge 1$, implying that $\delta_i \ge \frac{1}{\sqrt{k}}$. We assume that $k > \max(400, c_\epsilon)$ (where $c_\epsilon > 0$ is a large enough constant depending on ϵ such that $\log k \ge \frac{2}{\epsilon}\log 20$). Next, we assume that $n \ge c_{k,\epsilon}$ where $c_{k,\epsilon} > 0$ is a large enough constant depending on k, ϵ, such that $\epsilon \log n \ge \log(20k(40k+1)) + 1.5\log k + 1$. Under these assumptions, it is straightforward to prove the bound on $\left(\frac{1}{20}\sqrt{k}\right)^j \delta$.

$$\begin{aligned}
\left(\frac{1}{20}\sqrt{k}\right)^j &= \left(\frac{1}{20}\sqrt{k}\right)^{\left\lfloor \frac{\log n - \log(20k(40k+1))}{1.5\log k+1}\right\rfloor} \\
&\ge \left(\frac{1}{20}\sqrt{k}\right)^{\frac{\log n - \log(20k(40k+1))}{1.5\log k+1} - 1} \\
&\ge \left(\frac{1}{20}\sqrt{k}\right)^{\frac{(1-\epsilon)\log n}{1.5\log k+1}} && (\epsilon \log n \ge \log(20k(40k+1)) + 1.5\log k + 1) \\
&\ge \left(\frac{1}{20}\sqrt{k}\right)^{\frac{(1-\epsilon)\log n}{\frac{1.5\log k}{(1-\epsilon/3)}}} && (1.5\log k + 1 \le \tfrac{1.5\log k}{(1-\epsilon/3)} \text{ as } \log k \ge 2/\epsilon) \\
&\ge \left(\frac{1}{20}\sqrt{k}\right)^{\frac{(1-4\epsilon/3)\log n}{1.5\log k}}
\end{aligned}$$

$$
\begin{aligned}
&\geq \frac{2^{\frac{\log k}{2} \cdot \frac{(1-4\epsilon/3)\log n}{1.5 \log k}}}{20^{\frac{\log n}{1.5 \log k}}} \\
&\geq \frac{n^{\frac{1}{3}-\frac{4\epsilon}{9}}}{n^{\frac{\log 20}{1.5 \log k}}} \\
&\geq \frac{n^{\frac{1}{3}-\frac{4\epsilon}{9}}}{n^{\epsilon/3}} && (\tfrac{\log 20}{1.5 \log k} \leq \tfrac{\epsilon}{3}) \\
&\geq n^{\frac{1}{3}-\epsilon}
\end{aligned}
$$

Thus, for all $\delta \geq n^{-\frac{1}{3}+\epsilon}$, $\left(\frac{1}{20}\sqrt{k}\right)^j \delta \geq 1$. This completes the proof. □

B $n^{\Omega(1)}$ Width Lower Bound for Coin Problem with Bias $O(n^{-\frac{1}{2}})$

In this section, we use the result from [BGW20] to prove $n^{\Omega(1)}$-width lower bounds for the coin problem. [BGW20] (Theorem 1) states that

Theorem B.1. *Let $X_1, \ldots, X_n$ be a stream of uniform i.i.d. $\{-1, 1\}$ bits. Let A be a n-length read-once branching program which reads $X_1, \ldots, X_n$ in order and outputs the majority bit with probability at least 0.999. Then A uses $n^{\Omega(1)}$ width.*

Theorem 1 of [BGW20] states the result as a lower bound on the memory used by streaming algorithms, but it actually proves a width lower bound for ROBPs and translate it into memory bounds for one-pass streaming algorithms. Let $c > 0$ be a small enough constant and we consider the coin problem for the parameter $\delta = cn^{-\frac{1}{2}}$. Let D_δ be the probability distribution on $\{-1, 1\}^n$ where each coordinate is *i.i.d.* $\mathrm{Ber}(\frac{1}{2} + \delta)$, and $D_{-\delta}$ be the probability distribution on $\{-1, 1\}^n$ where each coordinate is *i.i.d.* $\mathrm{Ber}(\frac{1}{2} - \delta)$. Given two distributions $P.Q : \mathcal{X} \to [0, 1]$, let $|P - Q|_1$ denote the statistical distance between P and Q, that is, $|P - Q|_1 = \sum_{x \in \mathcal{X}} |P(x) - Q(x)|$. Let B be a length-n width-w ROBP as

$$
\Pr_{x \sim D_\delta}[B(x) = 1] - \Pr_{x \sim D_{-\delta}}[B(x) = 1] \leq \frac{|P - Q|_1}{2},
$$

the an advantage of B in solving the coin problem for parameter $\delta = cn^{-\frac{1}{2}}$ can at most be $\frac{|D_\delta - D_{-\delta}|_1}{2}$. Using Theorem B.1, we prove that if B solves the coin problem for parameter $\delta = cn^{-\frac{1}{2}}$ with an advantage at least $\frac{|D_\delta - D_{-\delta}|_1}{2} - 10^{-15}$, then $w = n^{\Omega(1)}$.

Lemma B.2. *Let B be a length-n width-w ROBP. There exists a small enough constant $c > 0$ such that if B solves the coin problem for parameter $\delta = cn^{-\frac{1}{2}}$ with an advantage of at least $\frac{|D_\delta - D_{-\delta}|_1}{2} - 10^{-15}$, then B has at least $n^{\Omega(1)}$ width.*

Note that $|D_\delta - D_{-\delta}|_1 = \Omega(1)$ for $\delta = \Omega(n^{-1/2})$, so the coin problem is statistically interesting for $\delta = cn^{-\frac{1}{2}}$ for any constant $c > 0$

Proof. By Definition 3.2,

$$
\begin{aligned}
\Pr_{x \sim D_\delta}[B(x) = 1] - \Pr_{x \sim D_{-\delta}}[B(x) = 1] &\geq \frac{|D_\delta - D_{-\delta}|_1}{2} - 10^{-15} \\
\text{and} \quad \Pr_{x \sim D_{-\delta}}[B(x) = -1] - \Pr_{x \sim D_\delta}[B(x) = -1] &\geq \frac{|D_\delta - D_{-\delta}|_1}{2} - 10^{-15}.
\end{aligned}
$$

For $x \in \{-1, 1\}^n$, let $|x|$ equal to the number if 1s in x.

$$
\Pr_{x \sim D_{-\delta}}[B(x) = -1] - \Pr_{x \sim D_\delta}[B(x) = -1]
$$

$$
\begin{aligned}
&= \sum_{x\in\{1,-1\}^n} (D_{-\delta(x)} - D_\delta(x)) \cdot 1_{B(x)=-1} \\
&= \sum_{x\in\{1,-1\}^n, |x|<\frac{n}{2}} (D_{-\delta}(x) - D_\delta(x)) \cdot 1_{B(x)=-1} + \sum_{x\in\{1,-1\}^n, |x|\geq\frac{n}{2}} (D_{-\delta}(x) - D_\delta(x)) \cdot 1_{B(x)=-1} \\
&= \sum_{x\in\{1,-1\}^n} (D_{-\delta}(x) - D_\delta(x)) \cdot 1_{B(x)=-1 \text{ and } |x|<\frac{n}{2}} - \sum_{x\in\{1,-1\}^n, |x|\geq\frac{n}{2}} (D_\delta(x) - D_{-\delta}(x)) \cdot 1_{B(x)=-1} \\
&\leq \frac{|D_\delta - D_{-\delta}|_1}{2} - \sum_{x\in\{1,-1\}^n, |x|\geq\frac{n}{2}} (D_\delta(x) - D_{-\delta}(x)) \cdot 1_{B(x)=-1}.
\end{aligned}
$$

As $\Pr_{x\sim D_\delta}[B(x)=1] - \Pr_{x\sim D_{-\delta}}[B(x)=1] \geq \frac{|D_\delta - D_{-\delta}|_1}{2} - 10^{-15}$, we get that

$$
\sum_{x\in\{1,-1\}^n, |x|\geq\frac{n}{2}} (D_\delta(x) - D_{-\delta}(x)) \cdot 1_{B(x)=-1} \leq 10^{-15}. \tag{22}
$$

We prove that there exists a small enough constant $c > 0$ such that for $\delta = cn^{-\frac{1}{2}}$, Equation (22) implies that $\frac{1}{2^n}\sum_{x\in\{1,-1\}^n, |x|\geq\frac{n}{2}} 1_{B(x)=-1} \leq 10^{-4}$. Similarly, we prove that $\frac{1}{2^n}\sum_{x\in\{1,-1\}^n, |x|<\frac{n}{2}} 1_{B(x)=1} \leq 10^{-4}$. Hence, $\Pr_{x\sim D_0}[B(x) = \text{majority}(x)] \geq 1 - 2\cdot 10^{-4}$ and Theorem B.1 implies that B has at least $n^{\Omega(1)}$ width. To prove by contradiction, we assume that

$$
\frac{1}{2^n} \sum_{x\in\{1,-1\}^n, |x|\geq\frac{n}{2}} 1_{B(x)=-1} > 10^{-4},
$$

and calculate the minimum value that $\sum_{x\in\{1,-1\}^n, |x|\geq\frac{n}{2}} (D_\delta(x) - D_{-\delta}(x)) \cdot 1_{B(x)=-1}$ can take.

$$
D_\delta(x) - D_{-\delta}(x) = \left(\frac{1}{2}+\delta\right)^{|x|}\left(\frac{1}{2}-\delta\right)^{n-|x|} - \left(\frac{1}{2}-\delta\right)^{|x|}\left(\frac{1}{2}+\delta\right)^{n-|x|},
$$

and as it increases with increasing $|x|$, the minimum is obtained when $B(x) = -1$ for smaller values $|x|$. Let y be the smallest $0 \leq y' \leq \frac{n}{2}$ such that

$$
\frac{1}{2^n} \sum_{x\in\{1,-1\}^n, y'+\frac{n}{2}\geq|x|\geq\frac{n}{2}} 1 = \frac{1}{2^n} \sum_{|x|=\frac{n}{2}}^{y'+\frac{n}{2}} \binom{n}{|x|} > 10^{-4}.
$$

As using Stirling's Approximation (similarly to as used in the proof of Lemma 4.3), for large enough n,

$$
\frac{\binom{n}{|x|}}{2^n} \leq \frac{\binom{n}{\lfloor n/2\rfloor}}{2^n} \leq \frac{1}{\sqrt{n}},
$$

we get that $y \geq 10^{-4}\sqrt{n}$. Let $z = 10^{-4}\sqrt{n} \leq y$. Therefore, minimum of $\sum_{x\in\{1,-1\}^n, |x|\geq\frac{n}{2}} (D_\delta(x) - D_{-\delta}(x)) \cdot 1_{B(x)=-1}$, conditioned on $\frac{1}{2^n}\sum_{x\in\{1,-1\}^n, |x|\geq\frac{n}{2}} 1_{B(x)=-1} > 10^{-4}$, is obtained when $B(x) = -1$ for $\frac{n}{2} \leq |x| \leq \frac{n}{2} + y$. Assuming n is large enough,

$$
\begin{aligned}
&\sum_{x\in\{1,-1\}^n, \frac{n}{2}+y\geq|x|\geq\frac{n}{2}} (D_\delta(x) - D_{-\delta}(x)) \\
&\geq \sum_{x\in\{1,-1\}^n, \frac{n}{2}+z\geq|x|\geq\frac{n}{2}+z/2} (D_\delta(x) - D_{-\delta}(x)) \\
&= \frac{1}{2^n} \sum_{|x|=\frac{n}{2}+z/2}^{\frac{n}{2}+z} \binom{n}{|x|} \left((1+2\delta)^{|x|}(1-2\delta)^{n-|x|} - (1-2\delta)^{|x|}(1+2\delta)^{n-|x|}\right)
\end{aligned}
$$

$$\geq \left(\frac{1}{2^n}\sum_{|x|=\frac{n}{2}+z/2}^{\frac{n}{2}+z}\binom{n}{|x|}\right)\left((1+2\delta)^{n/2+z/2}(1-2\delta)^{n/2-z/2}-(1-2\delta)^{n/2+z/2}(1+2\delta)^{n/2-z/2}\right)$$

Let $a = 10^{-4}$ and thus, $z = a\sqrt{n}$. Using Stirling's approximation,

$$\begin{aligned}
\frac{1}{2^n}\binom{n}{\frac{n}{2}+z} &\geq \frac{2\sqrt{2\pi}n^{n+\frac{1}{2}}}{e^2(n+2z)^{\frac{n}{2}+z+\frac{1}{2}}(n-2z)^{\frac{n}{2}-z+\frac{1}{2}}} \\
&= \frac{2\sqrt{\pi}}{e^2}\frac{n^{n+\frac{1}{2}}}{(n^2-4z^2)^{\frac{n}{2}}(n-2z)\left(\frac{n+2z}{n-2z}\right)^z} \\
&= \frac{2\sqrt{\pi}}{e^2}\frac{\sqrt{n}}{n-2a\sqrt{n}}\frac{\left(\frac{1-2\frac{a}{\sqrt{n}}}{1+2\frac{a}{\sqrt{n}}}\right)^{a\sqrt{n}}}{\left(1-4\frac{a^2}{n}\right)^{\frac{n}{2}}} \\
&\geq \frac{2\sqrt{\pi}}{e^2}\frac{1}{\sqrt{n}}\left(\frac{1-2\frac{a}{\sqrt{n}}}{1+2\frac{a}{\sqrt{n}}}\right)^{a\sqrt{n}} && (n \text{ is large enough}) \\
&\geq \frac{2\sqrt{\pi}}{e^2}\frac{1}{\sqrt{n}}\left(e^{-10\frac{a}{\sqrt{n}}}\right)^{a\sqrt{n}} && (\tfrac{1-x}{1+x} \geq e^{-5x} \text{ for } 0 \leq x \leq 0.5) \\
&\geq \frac{\sqrt{\pi}}{e^2\sqrt{n}} && (\left(e^{-10\frac{a}{\sqrt{n}}}\right)^{a\sqrt{n}} \geq 0.5 \text{ for } a < 10^{-2}) \\
&\geq \frac{0.2}{\sqrt{n}}
\end{aligned}$$

Therefore,

$$\begin{aligned}
\sum_{x\in\{1,-1\}^n,\frac{n}{2}+y\geq|x|\geq\frac{n}{2}}(D_\delta(x)-D_{-\delta}(x)) &\geq \frac{z}{2}\cdot\frac{0.2}{\sqrt{n}}\cdot\left((1+2\delta)^{n/2+z/2}(1-2\delta)^{n/2-z/2}-1\right) \\
&= \frac{a}{10}\cdot\left((1+2\delta)^{n/2+z/2}(1-2\delta)^{n/2-z/2}-1\right)
\end{aligned}$$

For $-\frac{1}{3} \leq x \leq \frac{1}{3}$, we show that $1 + x \geq e^{x-x^2}$. $\frac{\partial(1+x-e^{x-x^2})}{\partial x} = 1-(1-2x)e^{x-x^2}$. For $\frac{1}{3} \geq x \geq 0$, we have that $1 + 2x \geq e^{1.5x} \geq e^{x+x^2}$, and thus,

1. For $\frac{1}{3} \geq x \geq 0$, $1 \geq (1+2x)(1-2x) \geq (1-2x)e^{1.5x} \geq (1-2x)e^{x-x^2}$, implying that $1 + x - e^{x-x^2}$ is increasing in $x \in (0, 1/3]$.
2. For $-\frac{1}{3} \leq x < 0$, $1 - 2x \geq e^{-x+x^2} \implies (1-2x)e^{x-x^2} \geq 1$, implying that $1 + x - e^{x-x^2}$ is decreasing in $x \in [-1/3, 0)$.

Therefore, 0 is the minimum for the function $1 + x - e^{x-x^2}$ in $[-1/3, 1/3]$, implying $1 + x \geq e^{x-x^2}$ in $[-1/3, 1/3]$. Thus for large enough n,

$$\begin{aligned}
(1+2\delta)^{n/2+z/2}(1-2\delta)^{n/2-z/2} &\geq e^{(2\delta-4\delta^2)\cdot(n/2+z/2)+(-2\delta-4\delta^2)\cdot(n/2-z/2)} \\
&= e^{2\delta z-4\delta^2 n} \\
&= e^{2ca-4c^2} && (\text{for } \delta = cn^{-\frac{1}{2}} \text{ and } z = a\sqrt{n}) \\
&\geq e^{ca} && (\text{for } c \leq \tfrac{a}{4}) \\
&\geq 1 + ca.
\end{aligned}$$

Taking $c = \frac{a}{10}$, we get that (recall $a = 10^{-4}$)

$$\sum_{x \in \{1,-1\}^n, \frac{n}{2}+y \geq |x| \geq \frac{n}{2}} (D_\delta(x) - D_{-\delta}(x)) \geq \frac{a}{10} \cdot \left(\frac{a^2}{10}\right) \geq 10^{-14}$$

This contradicts Equation (22) and we prove the lemma. □

C Facts about simple random walks

Lemma C.1. *Let $t_1, t_2, \ldots, t_r$ be a random walk where if $t_i > 0$ then t_{i+1} distributes uniformly between $t_i - 1$ and $t_i + 1$ and if $t_i = 0$ then t_{i+1} distributes uniformly between 0 and 2. The expected number of indices j such that $t_j = 0$ is at most $\frac{2}{\sqrt{\pi}}\sqrt{r}$.*

Proof. Consider a standard uniform one-dimensional random walk $t'_1, t'_2, \ldots, t'_{2r}$ of length $2r$. If $|t'_i| > 0$ then $|t'_{i+1}|$ is distributed uniformly between $|t'_i| - 1$ and $|t'_i| + 1$. On the other hand, if $|t'_i| = 0$ then $t'_i = 0$ and thus t'_{i+1} distributes uniformly between -1 and 1 and t'_{i+2} distributes among $-2, 0, 2$ with respective probabilities $\frac{1}{4}, \frac{1}{2}, \frac{1}{4}$. In particular, $|t'_{i+2}|$ distributes uniformly between 0 and 2. We conclude that the expected number of times t_i equals zero is at most the expected number of times t'_i equals zero. For completeness, we also include a proof for a well-known bound on the number of times a uniform one-dimensional random walk visits the origin. If $t'_0 = 0$ then $t'_i = 0$ can hold only for even indices $i = 2j$. The probability that $t_{2j} = 0$ is

$$\frac{\binom{2j}{j}}{2^{2j}} \leq \frac{1}{\sqrt{\pi j}},$$

where the inequality follows from a standard upper bound on the central binomial coefficient. By linearity of expectation, the expected number of indices in which $t'_{2j} = 0$ is at most

$$\sum_{j=1}^{r} \frac{1}{\sqrt{\pi j}} \leq \frac{1}{\sqrt{\pi}} \left(1 + \int_1^r \frac{dx}{\sqrt{x}}\right) = \frac{1}{\sqrt{\pi}} \left(1 + 2\sqrt{r} - 2\sqrt{1}\right) \leq \frac{2}{\sqrt{\pi}}\sqrt{r}.$$

□

Lemma C.2. *The probability of a simple random walk of length T to pass through $-(k+1)$ or $k+1$ is lower than $4\exp\left(-\frac{k^2}{2T}\right)$.*

Proof. Denote the random walk by $s_0 = 0, s_1, \ldots, s_T$. Denote by $m_\ell := \max\{s_0, \ldots, s_\ell\}$ the maximal value reached in the first ℓ steps. It is known that for any $r \geq 1$,

$$Pr(m_T \geq r) = Pr(s_T = r) + 2Pr(s_T > r) \leq 2Pr(s_T \geq r).$$

By symmetry, the probability that the walk ever passed $-(k+1)$ or $k+1$ is thus at most $4Pr(s_T > k)$. Let R be the number of $+1$ steps taken, this is the sum of T i.i.d uniform Bernoulli variables. By Hoeffding's inequality,

$$Pr\left(s_T > k\right) = P\left(R > \frac{1}{2}\left(T + k\right)\right) = Pr\left(s_T > \left(\frac{1}{2} + \frac{k}{2T}\right)T\right) < \exp\left(-2\left(\frac{k}{2T}\right)^2 T\right).$$

□