Finding the Median (Obliviously) with Bounded Space

Paul Beame*, Vincent Liew**, and Mihai Pătraşcu***

Abstract. We prove that any oblivious algorithm using space S to find the median of a list of n integers from $\{1, \ldots, 2n\}$ requires time $\Omega(n \log \log_S n)$. This bound also applies to the problem of determining whether the median is odd or even. It is nearly optimal since Chan, following Munro and Raman, has shown that there is a (randomized) selection algorithm using only s registers, each of which can store an input value or $O(\log n)$-bit counter, that makes only $O(\log \log_s n)$ passes over the input. The bound also implies a size lower bound for read-once branching programs computing the low order bit of the median and implies the analog of $\mathsf{P} \neq \mathsf{NP} \cap \mathsf{coNP}$ for length $o(n \log \log n)$ oblivious branching programs.

1 Introduction

The problem of selection or, more specifically, finding the median of a list of values is one of the most basic computational problems. Indeed, the classic deterministic linear-time median-finding algorithm of [9], as well as the more practical expected linear-time randomized algorithm QuickSelect are among the most widely taught algorithms.

Though these algorithms are asymptotically optimal with respect to time, they require substantial manipulation and re-ordering of the input during their execution. Hence, they require the ability to write into a linear number of memory cells. (These algorithms can be implemented with only $O(1)$ memory locations in addition to the input if they are allowed to overwrite the input memory.) In many situations, however, the input is stored separately and cannot be overwritten unless it is brought into working memory. The number of bits S of working memory that an algorithm with read-only input uses is its *space*. This naturally leads to the question of the tradeoffs between the time T and space S required to find the median, or for selection more generally.

Munro and Paterson [18] gave multipass algorithms that yield deterministic time-space tradeoff upper bounds for selection for small space algorithms and showed that the number of passes p must be $\Omega(\log_s n)$ where $S = s \log_2 n$. Building on this work, Frederickson [14] extended the range of space bounds to nearly

* University of Washington. Research supported by NSF grants CCF-1217099 and CCF-0916400

** University of Washington. Research supported by NSF grant CCF-1217099

*** Much of this work was done with Mihai in 2009 and 2010 when the lower bounds for oblivious algorithms were obtained. This paper is dedicated to his memory.

linear space, deriving a multipass algorithm achieving a time-space tradeoff of the form $T = O(n \log^* n + n \log_s n)$. In the case of randomly ordered inputs, Munro and Raman [19] showed that on average an even better upper bound of $p = O(\log \log_s n)$ passes and hence $T = O(n \log \log_s n)$ is possible. Chakrabarti, Jayram, and Pătraşcu [12] showed that this is asymptotically optimal for multipass computations on randomly ordered input streams. Their analysis also applied to algorithms that perform arbitrary operations during their execution.

Chan [13] showed how to extend the ideas of Munro and Raman [19] to yield a randomized median-finding algorithm achieving the same time-space tradeoff upper bound as in the average case that they analyze. The resulting algorithm, like all of those discussed so far, only accesses its input using comparisons. Chan coupled this algorithm with a corresponding time-space tradeoff lower bound of $T = \Omega(n \log \log_S n)$ for randomized comparison branching programs, which implies the same lower bound for the randomized comparison RAM model. This is the first lower bound for selection allowing more than multipass access to the input; the input access can be input-dependent but the algorithm must base all its decisions on the input order. Though a small gap remains because $S \neq s$, the main question left open by [13] is that of finding time-space tradeoff lower bounds for median-finding algorithms that are not restricted to the use of comparisons.

Comparison-based versus general algorithms Though comparison-based algorithms for selection may be natural, when the input consists of an array of $O(\log n)$-bit integers, as one often assumes, there are natural alternatives to comparisons such as hashing that might potentially yield more efficient algorithms. Though comparison-based algorithms match the known time-space tradeoff lower bounds in efficiency for sorting when time T is $\Omega(n \log n)$ [10,4,21], they are powerless in the regime when T is $o(n \log n)$. Moreover, if one considers the closely related problem of element distinctness, determining whether or not the input has duplicates, the known time-space tradeoff lower bound of $T = \Omega(n^{2-o(1)}/S)$ for (randomized) comparison branching programs [22] can be beaten for S up to $n^{1-o(1)}$ by an algorithm using hashing [5] that achieves $T = \tilde{O}(n^{3/2}/S^{1/2})$[1]. Therefore, the restriction to comparison-based algorithms can be a significant limitation on efficiency.

Our results We prove a tight $T = \Omega(n \log \log_S n)$ lower bound for median-finding using arbitrary oblivious algorithms. Oblivious algorithms are those that can access the data in any order, not just in a fixed number of sweeps across the input, but that order cannot be data dependent. Our lower bound applies even for the decision problem of computing MedianBit, the low order bit of the median, when the input consists of n integers chosen from $\{1, \ldots, 2n\}$. This bound substantially generalizes the lower bound of [12] for multipass median-finding algorithms. Though our lower bound does not apply when there is input-dependent access to the input, it allows one to hash the input data values into working storage, and to organize and manipulate working storage in arbitrary ways.

[1] We use $\tilde{O}$ and $\tilde{\Omega}$ notations to hide logarithmic factors.

The median can be computed by a simple nondeterministic oblivious read-once branching program of polynomial size that guesses and verifies which input integer is the median. When expressed in terms of size for time-bounded oblivious branching programs our lower bound therefore shows that for every time bound T that is $o(n \log\log n)$, MedianBit and its complement have nondeterministic oblivious branching programs of polynomial size but MedianBit requires super-polynomial size deterministic oblivious branching programs, hence separating the analogs of P from $\mathsf{NP} \cap \mathsf{coNP}$.

We derive our lower bound using a reduction from a new communication complexity lower bound for two players to find the low order bit of median of their joint set of input integers in a bounded number of rounds. The use of communication complexity lower bounds in the "best partition" model to derive lower bounds for oblivious algorithms is not new, but the necessity of bounded rounds is. We derive our bound via a round-preserving reduction from oblivious computation to best-partition communication complexity [20,2]. This reduction is asymptotically less efficient than the reductions of [3,11] but the latter do not preserve the number of rounds, which is essential here since there is a very efficient $O(\log n)$-bit communication protocol using an unbounded number of rounds [17]. Moreover, the loss in efficiency does not prevent us from achieving asymptotically optimal lower bounds.

We further show that the fact that the median function is symmetric in its inputs implies that our oblivious branching program lower bound also applies to the case of non-oblivious read-once branching programs. Ideally, we would like to extend our non-oblivious results to larger time bounds. However, we show that extending our lower bound even to read-twice branching programs in the non-oblivious case would require fundamentally new lower bound techniques. The hardness of the median problem is essentially that of a decision problem: Though the median problem has $\Theta(\log n)$ bits of output, the high order bits of the median are very easy to compute; it is really the low order bit, MedianBit, that is the hardest to produce and encapsulates all of the difficulty of the problem. Moreover, all current methods for time-space tradeoff lower bounds for decision problems on general branching programs, and indeed for read-k branching programs for $k > 1$, also apply to nondeterministic algorithms computing either the function or its complement and hence cannot apply to the median because it is easy for such algorithms.

2 Preliminaries

Let D and R be finite sets. We first define branching programs that compute functions $f : D^n \to R$: A *D-way branching program* is a connected directed acyclic multigraph with special nodes: the *source node* and possibly many *sink nodes*, a sequence of n input values and one output. Each non-sink node is labeled with an input index and every edge is labeled with a symbol from D, which corresponds to the value of the input indexed at the originating node; there is precisely one out-edge from each non-sink node labeled by each element of D. We assume that each sink node is labeled by an element of R. The time

T required by a branching program is the length of the longest path from the source to a sink and the space S is $\log_2$ of the number of nodes in the branching program. A branching program is *leveled* iff all the paths from the source to any given node in the program are of the same length; a branching program can be leveled by adding at most $\log_2 T$ to its space.

A branching program B computes a function $f_B : D^n \to R$ by starting at the source and then proceeding along the nodes of the graph by querying the input locations associated with each node and following the corresponding edges until it reaches a sink node; the label of the sink node is the output of the function.

A branching program is *oblivious* iff on every path from the source node to a sink node, the sequence of input indices is precisely the same. It is (syntactic) *read-k* iff no input index appears more than k times on any path from the source to a sink.

Branching programs can easily simulate any sequential model of computation using the same time and space bounds. In particular branching programs using time T and space S can simulate random-access machine (RAM) algorithms using time T measured in the number of input locations queried and space S measured in the number of bits of read/write storage required. The same applies to the simulation of randomized RAM algorithms by randomized branching programs.

We also find it useful to discuss nondeterministic branching programs for (non-Boolean) functions, which simulate nondeterministic RAM algorithms for function computation. These have the property that multiple outedges from a single node can have the same label and outedges for some labels may not be present. Every input must have at least one path that leads to a sink and all paths followed by an input vector that lead to a sink must lead to the same one, whose label is the output value of the program. This is different from the usual version for decision problems in which one only considers accepting paths and infers the output value for those that are not accepting. When we consider Boolean functions we will typically assume the usual version based on accepting paths only.

We consider bounded-round versions of deterministic and randomized two-party communication complexity in which two players Alice and Bob receive $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ and cooperate to compute a function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$. A round in a protocol is a maximal segment of communication in which the player who speaks does not change. For a distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$, we say that a 2-party deterministic communication protocol computes f with error at most $\varepsilon < 1/2$ under $\mathcal{D}$ iff the probability over $\mathcal{D}$ that the output of the protocol on input $(x, y) \sim \mathcal{D}$ is equal to $f(x, y)$ is at least $1 - \varepsilon$. As usual, via Yao's lemma, for any such distribution $\mathcal{D}$, the minimum number of bits communicated by any deterministic protocol that computes f with error at most ε is a lower bound on the number of bits communicated by any (public coin) randomized protocol that computes f with error at most ε.

We say that a 2-party deterministic communication protocol has parameters $[P, \varepsilon; m_1, m_2, \dots]$ for f over a distribution $\mathcal{D}$ if:

– the first player to speak is $P \in \{A, B\}$;

– it has error $\varepsilon < \frac{1}{2}$ under input distribution $\mathcal{D}$;
– the players alternate turns, sending messages of $m_1, m_2, \ldots$ bits, respectively.

For probability distributions P and Q on a domain U, the statistical distance between P and Q, is $||P - Q|| = \max_{A \subseteq U} |P(A) - Q(A)|$, which is 1/2 of the L_1 distance between P and Q. Let log denote $\log_2$ unless otherwise specified. i Let $H(X)$ be the binary entropy of random variable X, $H(X|Y) = \mathbb{E}_{y \sim Y} H(X|Y{=}y)$, and let $I(X;Y|Z)$ be the mutual information between random variables X and Y conditioned on random variable Z. We have $I(X;Y|Z) \leq H(X|Z) \leq H(X)$.

3 Round Elimination

Let $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ and consider a distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$. We define the 2-player communication problem $f^{[k]}$ as follows: Alice receives $x \in \mathcal{X}^k$, while Bob receives $y \in \mathcal{Y}^k$ and $j \in [k]$; together they want to find $f(x_j, y_j)$. Also, given $\mathcal{D}$ we define an input distribution $\mathcal{D}^{[k]}$ for $f^{[k]}$ by choosing each (x_i, y_i) pair independently from $\mathcal{D}$, and independently choosing j uniformly from $[k]$.

The following lemma is a variant of standard techniques and was suggested to us by Anup Rao; its proof is in the full paper.

Lemma 1. *Assume that there exists a 2-party deterministic protocol for $f^{[k]}$ with parameters $[A, \varepsilon; m_1, m_2, m_3, \ldots]$ over $\mathcal{D}^{[k]}$ where $m_1 = \delta^2 k/(8 \ln 2)$. Then there exists a 2-party deterministic protocol for f with parameters $[B, \varepsilon + \delta; m_2, m_3, \ldots]$ over $\mathcal{D}$.*

The intuition for this lemma is that, since $f^{[k]}$ has k independent copies of the function f and Alice's first message has length at most m_1 which is only a small fraction of k, there must be some copy of f on which B learns very little information. This is so much less than one bit that B could forego this information in computing f and still only lose δ in his probability of correctness. The quadratic difference between the number of bits of information per copy, $\delta^2/(8 \ln 2)$, and the probability difference, δ, comes from Pinsker's inequality which relates information and statistical distance.

4 The Bounded-Round Communication Complexity of (the Least-Significant Bit of) the Median

We consider the complexity of the following communication game. Given a set A of n elements from $[2n]$ partitioned equally between Alice and Bob, determine the least significant bit of the median of A. (Since n must be even in order for A to be partitioned evenly, we take the median to be $n/2$-th largest element of A.) We consider the number of rounds of communication required when the length of each message is at most m for any $m \geq \log n$.

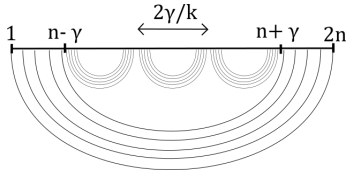

Fig. 1. Recursive construction of the pairing for the hard instances.

A Hard Distribution on Median Instances For our hard instances we first define a pairing of the elements of $[2n]$ that depends on the value of m. The set A will include precisely one element from each pair. For the input to the communication problem, we randomly partition the pairs equally between the two players which will therefore also automatically equally partition the set A. We then show how to randomly choose one element from each pair to include in A.

In the construction, we define the pairing of $[2n]$ recursively; the parameters of each recursive pairing will depend on the initial value n_0 of n. Let $k = k(m, n_0) = m \log^2 n_0$. If $\sqrt{n} < k \log^3 n_0$ then the elements of $[1, 2n]$ are simply paired consecutively. If $\sqrt{n} \geq k \log^3 n_0$ then the pairing of $[2n]$ consists of a "core" of $\gamma = \sqrt{n}/\log^2 n_0$ pairs, plus $n-\gamma$ "shell" pairs on $[1, n-\gamma] \cup [n+1+\gamma, 2n]$. In the shell, i and $2n+1-i$ are paired. The core pairs are obtained by embedding k recursive instances (using the same values of m and n_0) of $n' = \frac{\gamma}{k}$ pairs each on consecutive sets of $\frac{2\gamma}{k}$ elements, and placing them back-to-back in the value range $[n - \gamma + 1, n + \gamma]$, see Figure 1. The size of the problem at each level of recursion decreases from n to $n' = \gamma/k = \sqrt{n}/(m \log^4 n_0)$. In determining the median, the only relevant information about the shell elements is how many are below n; let this number be $\frac{n}{2} - x$. If $x \in [1, \gamma]$, the median of the entire array A will be the x-th order statistic of the core.

If furthermore, $x = \frac{\gamma}{k}(j - \frac{1}{2})$ for an integer j, the median of A will be exactly the median of the j-th embedded subproblem. In our distribution of hard instances, we will ensure that x has this nice form.

Formally, the distribution $\mathcal{D}^n_{m,n_0}$ of the hard instances A of size n on $[2n]$ is the following. Generate k recursive instances on $\mathcal{D}^{\gamma/k}_{m,n_0}$ and place shifted versions of them back-to-back inside the core. Choose $j \in [k]$ uniformly at random. Choose $\frac{n}{2} - \frac{\gamma}{k}(j - \frac{1}{2})$ uniformly random shell elements in $[1, n - \gamma]$ to include in A; for every $i \in [1, n-\gamma] \setminus A$, we have $2n + 1 - i \in A$. This will ensure that the median of A is precisely the median of the j-th recursive instance inside the core.

Initially we have $n = n_0$ and the recursion only continues when $\gamma = \sqrt{n}/\log^2 n_0 \geq k \log n_0$, so in the base case we have at least $\log n_0$ elements. In this case, the i-th element is chosen randomly and uniformly from the paired elements $2i - 1$ and $2i$ and so the least significant bit of the median is uniformly chosen in $\{0, 1\}$.

The size of the problem after t levels of recursion remains at least $n_0^{1/2^t}/(m \log^4 n_0)^{2-1/2^{t-1}}$ and our definition gives at least t levels provided that this size $n_0^{1/2^t}/(m \log^4 n_0)^{2-1/2^{t-1}} \geq \log n_0$; i.e., $n_0 \geq m^{2^{t+1}-2} \log^{9 \cdot 2^t - 2} n_0$. We

will show that after one message for each level of recursion, the answer is still not determined.

The general idea of the lower bound is that each round of communication, which consists of at most m bits and is much smaller than the branching factor k, will give almost no information about a typical recursive subproblem in the core.

We use the round elimination lemma to make this precise, and with it derive the following theorem:

Theorem 1. *If, for A chosen according to $\mathcal{D}^n_{m,n}$ and partitioned randomly, Alice and Bob determine the least significant bit of the median of A with bounded error $\varepsilon < 1/2$ using t messages of at most $m \geq \log n$ bits each, then $m^{2^{t+1}-2} > n/\log^{9\cdot 2^t-2} n$, which implies that $t \geq \log\log_m n - c$ for some constant c.*

The Partition Between the Players To ensure that neither player has enough information to skip a level of the recursion, we insist that the shell for each subproblem be nicely partitioned between the two players. For any given shell there is a set of $n' > m^2/2 \geq 0.5\log^2 n_0$ shell pairs. Since a player receives a random $1/2$ of all pairs, by Hoeffding's inequality, with probability $2^{-\Omega(n')}$, which is $n_0^{-\Omega(\log n_0)}$, at least $\frac{n'}{3}$ pairs go to each player. We can use this to say that with high probability at least $1/3$ of all shell elements at a level go to each player at every level of the recursion: This follows easily because over all levels of the recursive pairing, there are only a total of $o(\sqrt{n_0})$ different shells associated with subproblems and each one fails only with probability $n_0^{-\Omega(\log n_0)}$.

From now on, fix a partition satisfying the above requirement at all recursion nodes. We will prove a lower bound for any partition satisfying this property. Since we are discarding $o(1)$ of possible partitions, the error of the protocol may increase by $o(1)$, which is negligible.

The Induction Our proof of Theorem 1 will work by induction, using the following message elimination lemma:

Lemma 2. *Assume that there is a protocol for the median on instances of size n, with error ε on $\mathcal{D}^n_{m,n_0}$ for $\sqrt{n} \geq k\log n_0 = m\log^3 n_0$, using t messages of size at most m starting with Alice. Then, there is a protocol for a subproblem of size γ/k, with error $\varepsilon + O(\frac{1}{\log n_0})$ on $\mathcal{D}^{\gamma/k}_{m,n_0}$, using $t-1$ messages of size at most m starting with Bob.*

We use Lemma 2 to prove Theorem 1 by inductively eliminating all messages. Let $n_0 = n$. At each application we remove one message to get an error increase of $O(\frac{1}{\log n_0})$. If the number of rounds is less than the number of levels of recursion, i.e., $m^{2^{t+1}-2} \leq n/\log^{9\cdot 2^t-2} n$, then the MedianBit value of the subproblem will still be a uniformly random bit on the remaining input, but the protocol will have no communication and the error will have increased to at most $\epsilon + O(\frac{t}{\log n}) < 1/2$ since t is $O(\log\log_m n)$, which is a contradiction.

To prove Lemma 2 we want to apply Lemma 1 using the k subproblems in the core, but the assumption of Lemma 1 requires that (1) Alice does not know anything about which subproblem $j \in [k]$ is chosen by Bob, and (2) that subproblem j is chosen uniformly at random. The choice of subproblem j is determined by the shell elements at this level.

Denote Alice's shell elements by x^s, and Bob's shell elements by y^s. Let Alice's part of the core subproblems be $x_1, \ldots, x_k$, and Bob's part be $y_1, \ldots, y_k$. Note that the choice of the relevant subproblem j is some function of (x^s, y^s), and the median of the whole array is the median of $x_j \cup y_j$.

The proof of Lemma 2 proceeds in two stages:

Fixing x^s. We first fix the value of x^s so that the choice of subproblem does not depend on Alice's input and, moreover, so that the probabilities for different values of j over Bob's input y^s will not be very different from each other because they are still near the middle binomial coefficients.

By the niceness of the partition of the pairs, we know that the number of Alice's shell pairs is $|x^s| \in \left[\frac{1}{3}(n-\gamma), \frac{2}{3}(n-\gamma)\right]$. Let a be the number of elements in x^s that are below n. We want to fix x^s such that the error does not increase too much, and $|a - \frac{|x^s|}{2}| \leq \sqrt{n} \cdot \log n_0$:

No matter which value of $j \in [k]$ is chosen in the input distribution, the shell elements chosen to be below n consist of a random subset of $x^s \cup y^s$ of a fixed size that is between $n/2-\gamma$ and $n/2+\gamma$; i.e., of fractional size p_j between $\frac{1}{2} - \frac{\gamma}{n}$ and $\frac{1}{2} + \frac{\gamma}{n}$. By Hoeffding's inequality, the probability that the actual number a of these elements that land in x^s deviates from $|x^s|/2$ by more than $(t+\frac{\gamma}{n})|x^s|$ is at most $2e^{-2t^2|x^s|}$. Since $(n-\gamma)/3 \leq |x^s| \leq 2(n-\gamma)/3$, the probability that this deviates from $|x^s|/2$ by more than $\sqrt{n}\log n_0$ is at most $n_0^{-O(\log n_0)}$. We discard all values of x^s that lead to a outside this range. Now fix x^s to be the value that minimizes the conditional error.

Making j uniform. Once x^s is fixed, j is a function only of y^s. Thus, we are close to the setup of Lemma 1: Alice receives $x_1, \ldots, x_k$, Bob receives $y_1, \ldots, y_k$ and $j \in [k]$, and they want to compute a function $f(x_j, y_j)$. The only problem is that the lemma requires a uniform distribution of j, whereas our distribution is no longer uniform (having fixed x^s). However, we will argue that it is not far from uniform.

For each fixed $j_0 \in [k]$, if a shell elements from Alice's part are below n, then Bob must have $\frac{n}{2} - a - \frac{\gamma}{k}(j_0 - \frac{1}{2})$ shell elements below n. Therefore, $\Pr[j = j_0]$ is proportional to $\binom{|y^s|}{\frac{n}{2} - a - \frac{\gamma}{k}(j_0 - \frac{1}{2})}$. More precisely $\Pr[j = j_0]$ is this binomial coefficient divided by the sum of the coefficients for all j_0. Thus, to understand how close j is to uniform, we must understand the the dependence of these binomial coefficients on j_0.

Let $\Delta = a - |x^s|/2$. This satisfies $|\Delta| \leq \sqrt{n}\log n_0$. Since $|y^s| = n - |x^s| \geq \frac{n-\gamma}{3} > n/4$ we have $\binom{|y^s|}{\frac{n}{2} - a - \frac{\gamma}{k}(j_0 - \frac{1}{2})} = \binom{|y^s|}{|y^s|/2 - \Delta - \delta_{j_0}}$ where $0 < \delta_{j_0} < \gamma$. Assume wlog that $\Delta \geq 0$. The ratio between different binomial coefficients is at most the

ratio

$$\binom{n/4}{n/8-\Delta}/\binom{n/4}{n/8-\Delta-\gamma} = \frac{(n/8+\Delta+\gamma)\cdots(n/8+\Delta+1)}{(n/8-\Delta)\cdots(n/8-\Delta-\gamma+1)} \leq \left(1+\frac{10(2\Delta+\gamma)}{n}\right)^{\gamma}$$

which is $1+O(\frac{\Delta\gamma}{n}) = 1+O(\frac{1}{\log n_0})$ given the values of Δ and γ.

Therefore we have shown that the statistical distance between the induced distribution on j and the uniform distribution is $O(\frac{1}{\log n_0})$. We can thus consider the following alternative distribution for the problem: pick j uniformly at random, and manufacture y^s conditioned on this j. The error on the new distribution increases by at most $O(\frac{1}{\log n_0})$. Now we can apply Lemma 1. As $k \geq m \log^2 n_0$, the round elimination will increase the error by $O(\frac{1}{\log n_0})$.

5 Oblivious Branching Programs and the Median

The following result is essentially due to Okol'nishnikova [20], who used it with slightly different parameters for read-k branching programs, and was independently derived by Ajtai [2] in the context of general branching programs.

Proposition 1. *Let s be a sequence of of kn elements from $[n]$. If s is divided into $r = 4k^2$ segments $s_1, \ldots, s_r$, each of length $n/(4k)$, then there is an assignment of $2k$ segments s_j to a set L_A and all remaining segments s_j to L_B so that the number n_A (n_B) of elements of $[n]$ whose only appearances are in segments in L_A (respectively, L_B) satisfy $n_A \geq n/(2\binom{4k^2}{2k})$ and $n_B \geq n/2$.*

Proof. There is a subset V of at least $n/2$ elements of $[n]$ that occur at most $2k$ times in s and hence appear in at most $2k$ segments of s. Choose the $2k$ sets s_j to include in L_A uniformly at random. For a given $i \in V$, i will contribute to n_A if and only if all of the the at most $2k$ segments that contain its occurrences are chosen for L_A. This occurs with probability at least $1/\binom{r}{2k}$; hence the expected number of elements in V that only occur in segments of L_A is at least $|V|/\binom{r}{2k}$. Therefore we can select a fixed assignment that contains has at least this number. Since the total length of segments in L_A is at most $2kn/(4k) \leq n/2$, at least $n/2$ elements of $[n]$ only occur in segments in L_B.

Lemma 3. *Suppose that there is a $2n$-way oblivious branching program of size 2^S running in time $T = kn$ that computes* MedianBit *for n distinct inputs from $[2n]$. Then there is deterministic 2-party communication protocol using at most $4k$ messages of S bits each plus a final 1-bit message to compute* MedianBit *for $N = \lceil n/\binom{4k^2}{2k} \rceil$ distinct inputs from $[2N]$ that are divided evenly between the two players.*

Proof. Let s be the length T sequence of indices of inputs queried by the oblivious branching program. Let $k = T/n$, $r = 4k^2$, and $N = \lceil n/\binom{r}{2k} \rceil$. Fix the assignment of segments to L_A and L_B given by Proposition 1. Arbitrarily select a subset I_A of $N/2$ of the n_A indices that only appear in L_A and give those

inputs to player A. Similarly, select a subset I_B of $N/2$ of the n_B indices that only appear in L_B and give those inputs to player B. Let Q be the remaining set of $n - N$ input indices.

Fix any input assignment to the indices in Q that assigns $(n-N)/2$ distinct values from $[n-N]$ to half the elements of Q and the same number of distinct values from $[n+N+1, 2n]$ to the other half of the elements of Q. After fixing this partial assignment we restrict the remaining inputs to have values in the segment $[n-N+1, n+N]$ of length $2N$.

The communication protocol is derived as follows: Alice (resp. Bob) interprets her $N/2$ inputs from $[2N]$ as assignments from $[2n]$ to the elements of I_A (resp. I_B) by adding $n-N$ to each value. Alice will simulate the branching program executing the segments in L_A and Bob will simulate the branching program executing the segments in L_B. A player will continue the simulation until the next segment is held by the other player, at which point that player communicates the name of the node in the branching program reached at the end of its layer. Since L_A has only $2k$ segments, there are at most $4k$ alternations between players as well as the final output bit which gives the total communication. By construction, the median of the whole problem is the median of the N elements and the final answer for MedianBit on $[2N]$ is computed by XOR-ing the result with the low order bit of $n-N$.

Theorem 2. *Any oblivious branching program computing* MedianBit *for n inputs from $[2n]$ in time $T \le kn$ requires size at least $2^{\tilde{\Omega}(n^{1/2^{4k+2}})}$; in particular, if it uses space S, any oblivious branching program requires time $T \ge 0.25n \log\log_S n - c\, n$ for some constant c.*

Proof. Since $T/n \le k$, applying Lemma 3 we derive a 2-party communication protocol sending $t = 4k+1$ messages of at most $S \ge \log n$ bits each to compute MedianBit on $N \ge n/\binom{4k^2}{2k} \ge n/(2ek)^{2k}$ inputs from $[2N]$. By Theorem 1, $S > N^{1/(2^{t+1}-2)}/\log^{(9\cdot 2^t-2)/(2^{t+1}-2)} N > N^{1/(2^{4k+2}-2)}/\log^{71/15} N$ since $t \ge 4$ and hence $S \ge n^{1/(2^{4k+2}-2)}/\log^5 n$. The size of the branching program is 2^S where S is its space. Moreover, taking logarithms base S and then base 2 we have $4k \ge \log\log_S n - c'$ for some constant c'.

Analog of P ≠ NP ∩ coNP for time-bounded oblivious BPs

Corollary 1. *Any oblivious branching program of length $T \le kn$ computing the low order bit of the median requires size at least $exp(\tilde{\Omega}(n^{1/2^{4k+2}}))$; in particular, this size is super-polynomial when T is $o(n \log\log n)$.*

On the other hand, the median can be computed by a nondeterministic oblivious read-once branching program using only $O(\log n)$ space.

Lemma 4. *There is a nondeterministic oblivious read-once branching program of size $O(n^4)$ that computes the median on n integers from $[2n]$.*

Proof. The branching program guesses the value of the median in $[2n]$ and keeps track of the number of elements that it has seen both less than the median and equal to the median in order to check that the value is correct.

In particular, in contrast to Corollary 1, Lemma 4 implies that MEDIAN-BIT can be computed in polynomial size by length n nondeterministic and co-nondeterministic oblivious branching programs, hence we have shown the analog of $\mathsf{P} \neq \mathsf{NP} \cap \mathsf{coNP}$ for oblivious branching programs of length $o(n \log\log n)$.

6 Beyond Oblivious Branching Programs

We first observe that our lower bounds for the median problem extend to the case of read-once branching programs by using the fact that such programs for the median can also be assumed to be oblivious without loss of generality. (Oblivious read-once branching programs are also known as *ordered binary decision diagrams (OBDDs).*)

Lemma 5. *If $f : D^n \to R$ is a symmetric function of its inputs then for every read-once branching B computing f there is an oblivious read-once branching program, of precisely the same size as B, that computes f.*

Proof. With each node v in a read-once branching program, we can associate a set $I_v \subseteq [n]$ of input indices that are read along paths from the source node to v. We make B into an oblivious branching program by replacing the index at node v by $|I_v| + 1$. This yields an oblivious read-once branching program (not necessarily leveled) that reads its inputs in the order $x_1, x_2, \ldots, x_n$ along every path (possibly skipping over some inputs on the path). Since f is a symmetric function, a path of length $t \leq n$ in B queries t different input locations and the value of the function on the partial inputs is the same because the function is symmetric and the values in those t input locations are the same.

Corollary 2. *For any $\varepsilon < 1/2$, any read-once branching program computing* MEDIANBIT *for n integers from $[2n]$ requires size $2^{n^{\Omega(1)}}$.*

In particular this means that MEDIANBIT is another example, after those in [16], of a problem showing the analogue of $\mathsf{P} \neq \mathsf{NP} \cap \mathsf{coNP}$ for read-once branching programs. However, proving the analogous property even for read-twice branching programs remains open and will require a fundamentally new technique for deriving branching program lower bounds.

The approach in all lower bounds for general branching programs (or even for read-k branching programs) computing decision problems [11,20,7,2,1,6,8] applies equally well to nondeterministic computation. (For example, the fact that the technique also works for nondeterministic computation is made explicit in [11].) Though this technique has been used to separate nondeterministic from deterministic computation [2] computing a Boolean function f, it is achieved by proving a nondeterministic lower bound for computing $\overline{f}$. Since the nondeterministic oblivious read-once branching program computing the median has $T = n$ and $S = O(\log n)$, the core of the median's hardness, MEDIANBIT, and its complement do not have non-trivial lower bounds; hence current time-space tradeoff lower bound techniques are powerless for computing the median.

We conjecture that the lower bound $T = \Omega(n \log\log_S n)$ also holds for finding the median using general non-oblivious algorithms.

References

1. M. Ajtai. A non-linear time lower bound for boolean branching programs. In *Proceedings 40th IEEE FOCS conference*, pages 60–70, New York, NY, 1999.
2. M. Ajtai. Determinism versus non-determinism for linear time RAMs with memory restrictions. *Journal of Computer and System Sciences*, 65(1):2–37, August 2002.
3. N. Alon and W. Maass. Meanders and their applications in lower bounds arguments. *Journal of Computer and System Sciences*, 37:118–129, 1988.
4. P. Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM Journal on Computing*, 20(2):270–277, 1991.
5. P. Beame, R. Clifford, and W. Machmouchi. Element distinctness, frequency moments, and sliding windows. In *Proceedings 54th IEEE FOCS conference*, pages 290–299, Berkeley, CA, 2013.
6. P. Beame, M. Saks, X. Sun, and E. Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *J. ACM*, 50(2):154–195, 2003.
7. P. Beame, T. S. Jayram, and M. Saks. Time-space tradeoffs for branching programs. *Journal of Computer and System Sciences*, 63(4):542–572, 2001.
8. P. Beame and E. Vee. Time-space tradeoffs, multiparty communication complexity, and nearest-neighbor problems. In *Proceedings 34th ACM STOC conference*, pages 688–697, Montreal, Quebec, Canada, 2002.
9. M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1972.
10. A. Borodin and S. A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM Journal on Computing*, 11(2):287–297, May 1982.
11. A. Borodin, A. A. Razborov, and R. Smolensky. On lower bounds for read-k times branching programs. *Computational Complexity*, 3:1–18, October 1993.
12. A. Chakrabarti, T. S. Jayram, and M. Patrascu. Tight lower bounds for selection in randomly ordered streams. In *Proceedings 19th ACM-SIAM SODA conference*, pages 720–729, San Francisco, CA, 2008.
13. T. M. Chan. Comparison-based time-space lower bounds for selection. *ACM Transactions on Algorithms*, 6(2):26:1–16, 2010.
14. G. N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *Journal of Computer and System Sciences*, 34(1):19–26, 1987.
15. T. Holenstein. Parallel repetition: Simplification and the no-signaling case. *Theory of Computing*, 5(1):141–172, 2009.
16. S. Jukna, A. A. Razborov, P. Savický, and I. Wegener. On P versus NP∩ co-NP for decision trees and read-once branching programs. *Computational Complexity*, 8(4):357–370, 1999.
17. E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, Cambridge, England ; New York, 1997.
18. J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.
19. J. I. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *Theoretical Computer Science*, 165(2):311–323, 1996.
20. E. Okol'nishnikova. On lower bounds for branching programs. *Siberian Advances in Mathematics*, 3(1):152–166, 1993.
21. J. Pagter and T. Rauhe. Optimal time-space trade-offs for sorting. In *Proceedings 39th IEEE FOCS conference*, pages 264–268, Palo Alto, CA, 1998.
22. A. C.-C. Yao. Near-optimal time-space tradeoff for element distinctness. *SIAM Journal on Computing*, 23(5):966–975, 1994.