Time-Space Lower Bounds for Simulating Proof Systems with Quantum and Randomized Verifiers

Abhijit S. Mudigonda
Portland, OR, USA
https://abhijit-mudigonda.github.io/math/
abhijitm@mit.edu

R. Ryan Williams
EECS and CSAIL, MIT, Cambridge, MA, USA
https://people.csail.mit.edu/rrw/
rrw@mit.edu

Abstract

A line of work initiated by Fortnow in 1997 has proven model-independent time-space lower bounds for the SAT problem and related problems within the polynomial-time hierarchy. For example, for the SAT problem, the state-of-the-art is that the problem cannot be solved by random-access machines in n^c time and $n^{o(1)}$ space simultaneously for $c < 2\cos(\frac{\pi}{7}) \approx 1.801$.

We extend this lower bound approach to the quantum and randomized domains. Combining Grover's algorithm with components from SAT time-space lower bounds, we show that there are problems verifiable in $O(n)$ time with quantum Merlin-Arthur protocols that cannot be solved in n^c time and $n^{o(1)}$ space simultaneously for $c < \frac{3+\sqrt{3}}{2} \approx 2.366$, a super-quadratic time lower bound. This result and the prior work on SAT can both be viewed as consequences of a more general formula for time lower bounds against small-space algorithms, whose asymptotics we study in full.

We also show lower bounds against randomized algorithms: there are problems verifiable in $O(n)$ time with (classical) Merlin-Arthur protocols that cannot be solved in n^c randomized time and $O(\log n)$ space simultaneously for $c < 1.465$, improving a result of Diehl. For quantum Merlin-Arthur protocols, the lower bound in this setting can be improved to $c < 1.5$.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases Time-space tradeoffs, lower bounds, QMA

Digital Object Identifier 10.4230/LIPIcs.ITCS.2021.50

Related Version A full version of the paper is available at https://arxiv.org/abs/2012.00330.

Funding *R. Ryan Williams*: Supported by NSF CCF-1909429 and CCF-1741615.

Acknowledgements We thank the anonymous reviewers for helpful comments. The first author thanks Ryan Williams for his support and patience throughout this research. The first author also thanks Aram Harrow, Peter Shor, Saeed Mehraban, Ashwin Sah, and Lisa Yang for contributing office space and helpful conversations and Lijie Chen and Shyan Akmal for reading and editing a draft of this manuscript. Lastly, the first author apologizes to the Theory Group at MIT CSAIL eating so many of the chocolate-covered pretzels.

1 Introduction

A flagship problem in computational complexity is to prove lower bounds for the SAT problem. While it is conjectured that no polynomial-time algorithms exist for SAT (in other words, P $\neq$ NP), not much progress has been made in that direction. Furthermore, several significant barriers towards such a separation are known [3, 22, 1]. Therefore, approaches have centered around proving weaker lower bounds on SAT first.

12th Innovations in Theoretical Computer Science Conference (ITCS 2021).
Editor: James R. Lee; Article No. 50; pp. 50:1–50:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A natural preliminary step in showing that no polynomial-time algorithm can decide SAT is showing that no algorithms of logarithmic space can decide SAT, or in other words, showing that $\mathsf{L} \neq \mathsf{NP}$. Unlike the P vs. NP problem, the aforementioned complexity barriers (arguably) do not apply as readily to L vs. NP, and concrete progress has been made.[1]

Following a line of work [11], R. Williams [25] proved that SAT (equivalently[2] $\mathsf{NTIME}[n]$, nondeterministic linear time) cannot be decided by algorithms (even with constant-time random access to their input and storage) using both $n^{o(1)}$ space and n^c time, for $c < 2\cos(\frac{\pi}{7}) \approx 1.802$. If one could show the same lower bound for arbitrarily large constant c, the separation $\mathsf{L} \neq \mathsf{NP}$ would follow immediately. In the following we use $\mathsf{TS}[n^c]$ to denote the class of languages decidable by $n^{o(1)}$-space algorithms using n^c time.

All the aforementioned work builds on the *alternation-trading proofs* approach [27]. This approach combines two elements: a *speedup rule* that reduces the runtime of an algorithm by "adding a quantifier" ($\exists$ or $\forall$) to an alternating algorithm, and a *slowdown rule* that uses a complexity theoretic assumption (for example, $\mathsf{SAT} \in \mathsf{TS}[n^c]$) to "remove a quantifier" and slightly increase the runtime of the resulting algorithm. Both rules yield inclusions of complexity classes. Our ultimate goal is to contradict a time hierarchy theorem (e.g., proving n^{100} time computations can be simulated in n^{99} time) by applying these rules in a nice order, and with appropriately chosen parameters.

One may hope that the constant c from [25] can be made arbitrarily large, and eventually show that $\mathsf{L} \neq \mathsf{NP}$. Unfortunately, in [7], R. Williams and S. Buss showed that no alternation-trading proof based purely on the speedup and slowdown rules from that line of work could improve on the exponent of [25].

Nevertheless, there is hope that alternation-trading proofs might yield stronger lower bounds for problems harder than SAT. For example, R. Williams [27] showed that the $\Sigma_2\mathsf{P}$-complete problem $\Sigma_2\mathsf{SAT}$ is not in $\mathsf{TS}[n^c]$, for $c < 2.903$. In this paper, we make further progress in this direction. In particular, we focus on the quantum and randomized analogues of $\mathsf{NTIME}[n]$, $\mathsf{QCMATIME}[n]$ and $\mathsf{MATIME}[n]$, obtaining stronger lower bounds against both classes.[3] We believe our lower bound for $\mathsf{QCMATIME}[n]$ (Main Theorem 2) to be particularly interesting because it yields a *nontrivial* separation between a quantum complexity class and a classical complexity class *without the need for oracles.*[4] While there are several results [6, 21, 24] demonstrating the power of quantum computation against very restricted low-depth classical circuit models (NC^0, AC^0, $\mathsf{AC}^0[2]$) which also imply strong oracle separation results, our result appears to be the first non-trivial lower bound for a quantum class against the much more general random-access machine model (with simultaneous time and space constraints).

1.1 Our Results

1.1.1 Generic slowdown rules and a lower bound for $\mathsf{QCMATIME}[n]$

For showing stronger lower bounds on $\mathsf{QCMATIME}[n]$, our key observation is that the stronger assumption $\mathsf{QCMATIME}[n] \subseteq \mathsf{TS}[n^c]$ (compared to $\mathsf{NTIME}[n] \subseteq \mathsf{TS}[n^c]$) can be applied to construct a stronger conditional slowdown rule. Formally, we generalize the

[1] For example, there exist oracles relative to which the lower bounds in the following paragraph are false.

[2] At least, up to polylogarithmic factors.

[3] Recall that QCMA (quantum classical Merlin-Arthur) is essentially NP with a quantum verifier and MA (Merlin-Arthur) is essentially NP with a randomized verifier.

[4] By "nontrivial", we mean a separation that does not immediately follow from known classical results. For example, $\mathsf{QCMATIME}[n] \not\subseteq \mathsf{TS}[n^{1.8}]$ follows immediately from the classical lower bound $\mathsf{NTIME}[n] \not\subseteq \mathsf{TS}[n^{1.8}]$, but our result does not.

previous framework of alternation-trading proofs by introducing the notion of a *generic slowdown rule* (defined formally in Definition 11), which are slowdown rules parameterized by a constant $\alpha \in (0, 1]$ controlling the runtime cost associated with removing a quantifier. Smaller values of α correspond to stronger slowdown rules. We prove the following theorem showing how generic slowdown rules imply time-space lower bounds.

▶ **Main Theorem 1.** *Fix $\alpha \in (0, 1]$ and let $\mathcal{C}$ be a complexity class. Let r_1 be the largest root of the polynomial $P_\alpha(x) := \alpha^2 x^3 - \alpha x^2 - 2\alpha x + 1$. If $\mathcal{C} \subseteq \mathsf{TS}[n^c]$ implies a generic slowdown rule with parameters α and c, then $\mathcal{C} \not\subseteq \mathsf{TS}[n^c]$ for $c < r_1$.*

The assumption $\mathsf{NTIME}[n] \subseteq \mathsf{TS}[n^c]$ implies a slowdown rule with $\alpha = 1$. The previous $\mathsf{NTIME}[n] \not\subseteq \mathsf{TS}[n^c]$ for $c < 2\cos(\frac{\pi}{7})$ lower bound [25] becomes an immediate corollary of Main Theorem 1 if we take $\mathcal{C} = \mathsf{NTIME}[n]$. The stronger slowdown rule that we obtain from the stronger assumption $\mathsf{QCMATIME}[n] \subseteq \mathsf{TS}[n^c]$ has $\alpha = \frac{2}{3}$, which allows us to derive lower bounds for larger values of c. In particular, we obtain the following lower bound for Quantum (Classical) Merlin-Arthur linear time.

▶ **Main Theorem 2.** $\mathsf{QCMATIME}[n] \not\subseteq \mathsf{TS}[n^c]$ *for $c < \frac{3+\sqrt{3}}{2} \approx 2.366$.*

The main advantage of the generic slowdown approach and Main Theorem 1 is that improvements in slowdown rules translate immediately into stronger bounds against TS. Figure 1 shows how the lower bound exponent we obtain changes as α does. As expected, the lower bound exponent goes to infinity as α approaches zero, but we see that even modest improvements in α yield substantially stronger bounds. We discuss potential applications further in Section 1.3. In the full version of the paper, we show that this dependence between the generic slowdown parameter and lower bound exponent is "optimal" for the tools we use, extending the optimality theorem of Buss and Williams [7] to the general case while also providing a shorter proof of their optimality theorem.

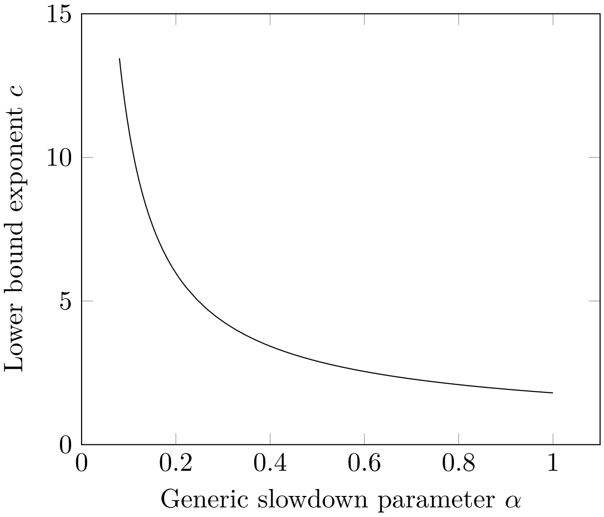

Figure 1 The lower bound exponent as a function of the generic slowdown parameter α.

1.1.2 Lower bounds against randomized logspace

We also prove lower bounds against randomized logspace with two-sided error. While the techniques used in this setting are similar, the results do not follow from Main Theorem 1, so we state them separately. When discussing the randomized setting, we will slightly abuse notation by referring to the class of languages decidable by a logspace machine in n^c time as $\mathsf{BPTS}[n^c]$.[5] We prove lower bounds for both linear-time Merlin-Arthur protocols and linear-time Classical-Merlin Quantum-Arthur protocols.

▶ **Main Theorem 3.** *Let $r_1 \approx 1.465$ be the largest root of the polynomial $x^3 - x^2 - 1$. Then,* $\mathsf{MATIME}[n] \not\subseteq \mathsf{BPTS}[n^c]$ *for $c < r_1$. Furthermore,* $\mathsf{QCMATIME}[n] \not\subseteq \mathsf{BPTS}[n^c]$ *for $c < 1.5$.*

Prior to this work, the state-of-the-art, due to [10, 8], was that $\mathsf{MATIME}[n] \not\subseteq \mathsf{BPTS}[n^c]$ for $c < \sqrt{2} \approx 1.414$. Observe that $\mathsf{MATIME}[t] \subseteq \mathsf{PPTIME}[t^2]$ because we can amplify Arthur's completeness and soundness to $(1 - 2^{-100t}, 2^{-100t})$ while increasing the runtime of the verifier by a factor of $O(t)$, and we can union bound over Merlin's strings.[6] A similar argument, coupled with the quasilinear-time simulation of bounded-error quantum computation with unbounded-error random computation of [23], shows that $\mathsf{QCMATIME}[t] \subseteq \mathsf{PPTIME}[t^2]$. Therefore, pushing either of the lower bound exponents in Main Theorem 3 to beyond 2 would yield superlinear bounds for decision versions of counting-type problems (e.g. MAJ-SAT) against randomized logspace. (Note it is known that #SAT requires $\tilde{O}(n^2)$ time for randomized logspace [17].) While these results are admittedly incremental improvements, they use some different ideas compared to previous works, and may be amenable to further improvement (see Section 1.3 for more details).

The lower bounds of Main Theorem 2 and Main Theorem 3 actually hold for complexity classes that are presumably even "smaller" than $\mathsf{QCMATIME}[n]$ and $\mathsf{MATIME}[n]$; we describe these further in Section 2.

1.2 Techniques

1.2.1 Alternation-trading proofs

Many time-space lower bounds for SAT and related problems are proved via *alternation-trading proofs*, which give a chain of inclusions of complexity classes. We will formally define alternation-trading proofs in Section 2; for now, let us give a cursory explanation. An *alternation-trading proof* consists of a sequence of containments of *alternating complexity classes.* An alternating complexity class can be thought of as a "fine-grained" version of $\Sigma_k\mathsf{P}$ or $\Pi_k\mathsf{P}$: it is a complexity class parameterized by $(k+1)$ positive constants bounding the length of the output of each quantifier and the verifier runtime. For example,

$$(\exists n^2)(\forall n^2)\mathsf{TS}[n^5]$$

is an alternating complexity class. This notation refers to the class of languages decided by a Σ_2 machine where, on inputs of length n, the two quantifiers each quantify over $\tilde{O}(n^2)$ bit strings and the *verifier runtime* is $\tilde{O}(n^5)$.

In an alternation-trading proof, there are two main ways of passing from one alternating complexity class to the next. The first is a **speedup rule**, which adds a quantifier to the class, and decreases the verifier runtime. For example, a speedup rule might yield an inclusion of the form

$$\ldots \mathsf{TS}[n^d] \subseteq \ldots (Qn^x)(\neg Qx \log n)\mathsf{TS}[n^{d-x}] \tag{1}$$

[5] We call this a "slight abuse" because we defined TS to be a class of languages decided using $n^{o(1)}$ space, whereas BPTS is defined with respect to $O(\log n)$ space.

[6] By a union bound, there is a gap between the case where all 2^t Merlin strings have a 2^{-100t} chance of accepting, and the case where a single Merlin string accepts with probability at least $1 - 2^{-100t}$.

for some constant $0 < x < d$ and quantifier $Q \in \{\exists, \forall\}$, where $\neg Q$ denotes the opposite quantifier and the $\ldots$ refer to other quantifiers. Two important points to note are that (a) the speedup rule is generally an unconditionally true inclusion and (b) the second quantifier has only $O(\log n)$ bits.

The second major component of alternation-trading proofs is a **slowdown rule**, which removes a quantifier and increases the verifier runtime. We will use slowdown rules that hold conditioned on complexity-theoretic assumptions (that we will later contradict). For example, the slowdown rule used to prove lower bounds on $\mathsf{NTIME}[n]$ can be informally stated as follows: assuming $\mathsf{NTIME}[n] \subseteq \mathsf{TS}[n^c]$ for some $c > 0$,

$$\ldots(Qn^a)(\neg Qn^b)\mathsf{TS}[n^d] \subseteq \ldots(Qn^a)\mathsf{TS}[n^{c\cdot\max(a,b,d)}]. \tag{2}$$

where again $Q \in \{\exists, \forall\}$ and $\neg Q$ denotes the opposite quantifier. This rule follows from an application of padding/translation.

While the speedup and slowdown rules are themselves simple, the existing lower bounds on $\mathsf{NTIME}[n]$ arise by applying these rules in a long intricate sequence, with appropriately chosen parameters for the speedup rule applications. Ultimately, we aim to use slowdown and speedup rules to exhibit a sequence of inclusions that shows (for example) that $\mathsf{NTIME}[n^d] \subseteq \mathsf{NTIME}[n^{d'}]$ for $d' < d$, contradicting the nondeterministic time hierarchy theorem and demonstrating that our initial assumption must have been false.

All time-space lower bounds for SAT against random-access models of computation, including the state-of-the-art bound [25], use an alternation-trading proof. This particular proof will be a starting point for this work.

1.2.2 Generic slowdown rules

We start by introducing the notion of a generic slowdown rule. Generic slowdown rules are parameterized by a constant α such that $0 < \alpha \leq 1$, along with a constant $c \geq 1$ that generally comes with an assumption being made. Informally, generic slowdown rules allow us to – under an appropriate assumption – prove conditional inclusions of the form

$$\ldots(Qn^a)(\neg Qn^b)\mathsf{TS}[n^d] \subseteq \ldots(Qn^a)\mathsf{TS}[n^{\alpha c\cdot\max(a,b,d)}].$$

Observe that when $\alpha = 1$ we recover (2), but when $\alpha < 1$ we obtain stronger inclusions. In Main Theorem 1, we use generic slowdowns in the alternation-trading proof from [25] and characterize the lower bound we obtain as a function of the parameter α in our generic slowdown rule. While the core idea of this proof is essentially the same as the presentation of the proof in [27] (and the result can be thought of as "putting α everywhere"), our proof technique is somewhat different. In the full version of the paper, this different approach yields a shorter proof of optimality than the one presented by Buss and Williams [7].

1.2.3 A generic slowdown rule from Grover search

In order to apply Main Theorem 1 to $\mathcal{C} = \mathsf{QCMATIME}[n]$ and obtain a better lower bound for QCMA, we show that the assumption $\mathsf{QCMATIME}[n] \subseteq \mathsf{TS}[n^c]$ implies a generic slowdown rule for $\alpha = \frac{2}{3}$. Recall that Grover's algorithm lets us search a space of size N with only $O(\sqrt{N})$ quantum queries. We obtain our stronger slowdown rule by showing that Grover's algorithm can be used to more efficiently remove the $(x \log n)$-bit quantifiers that arise after applications of the speedup rule, such as (1). In the NTIME vs TS setting, there are two ways to remove an $(x \log n)$-bit quantifier. First, we could remove it with an $O(n^x)$

multiplicative blowup, by having the verifier exhaustively search through all strings of length $x \log n$. However, naively running n^x trials of an n^{d-x} computation would take n^d time, and our simulation would end up no faster than it was initially. Second, we may try to use a slowdown rule as in (2), but this incurs a runtime cost that depends on c. This option becomes expensive as we try to increase c and prove stronger lower bounds. Our key insight is that if our verifier is allowed to be quantum, Grover's algorithm can be applied to perform this quantifier elimination in only $O(n^{x/2})$ extra time overhead, independent of c. Then, by applying the assumption $\mathsf{QCMATIME}[n] \subseteq \mathsf{TS}[n^c]$, we can remove this quantum verifier along with the quantifier $(\exists n^x)$ and ultimately demonstrate the inclusions

$$\begin{aligned}
&\ldots (Qn^a)(\neg Qn^b)\mathsf{TS}[n^d] && \\
\subseteq &\ldots (Qn^a)(\neg Qn^b)(Qn^x)(\neg Qx \log n)\mathsf{TS}[n^{d-x}] && \text{Speedup Rule} \\
\subseteq &\ldots (Qn^a)(\neg Qn^b)(Qn^x)\mathsf{BQTIME}[n^{d-\frac{x}{2}}] && \text{Grover's Algorithm} \\
\subseteq &\ldots (Qn^a)(\neg Qn^b)\mathsf{TS}[n^{c\cdot\max(b,x,d-\frac{x}{2})}]. && \text{Assumption on } \mathsf{QCMATIME}[n]
\end{aligned}$$

Letting $x := \frac{2d}{3}$, we obtain a generic slowdown rule with $\alpha = \frac{2}{3}$.

1.2.4 Lower bounds against randomized logspace

Some obstacles arise when trying to prove Main Theorem 3, which shows lower bounds against BPTS. The main problem is that the usual speedup rules for deterministic computation do not tell us how to use quantifiers to speedup randomized small-space computations. Fortunately, this particular issue was resolved by Diehl and Van Melkebeek [10], who gave a speedup rule for small-space randomized machines by coupling Nisan's space-bounded derandomization [20] with the Sipser-Gács-Lautemann theorem [16]. This speedup rule, while somewhat less efficient than the speedup rule for deterministic machines, is still enough to obtain interesting superlinear time lower bounds. Applying this speedup rule, similar arguments as used in Main Theorem 1 yield the desired lower bounds.

1.3 Future Work

As mentioned earlier, the main advantage of the generic slowdown framework and Main Theorem 1 is that improvements in slowdown rules translate immediately into stronger bounds against TS. To this end, we highlight two particularly interesting directions.

1. Is it possible to prove a "quantum speedup rule", whereby the runtime of quantum computations could be reduced by adding quantifiers (possibly over quantum states)? Presently, we are forced to use a slowdown rule to remove a quantum verifier from an alternating complexity class as soon as it is added. Having a quantum speedup rule would enable us to work with the quantum verifier before removing it, drastically widening the scope for potential alternation-trading proofs. It's not hard to show that even certain weak forms of a quantum speedup rule would improve the generic slowdown parameter α we can obtain in the QCMA vs. TS setting. Speedup rules also have applications outside complexity theory. For example, versions of speedup rules for low-space computation appear in the construction of delegation schemes in cryptography [4, 13, 14], and quantum speedup rules could play a part in extending such work to the quantum domain.

2. Can we use the ideas of this paper to improve existing lower bounds on counting-type (#P related) problems, such as #SAT and MAJ-SAT? For example, could our super-quadratic time lower bound for $\mathsf{QCMATIME}[n]$ be somehow applied to obtain super-quadratic lower bounds for #SAT as well? Because

   $$\mathsf{MATIME}[t] \subseteq \mathsf{QCMATIME}[t] \subseteq \mathsf{PPTIME}[t^{2+o(1)}]$$

   as discussed in Section 1.1.2, lower bounds on $\mathsf{QCMATIME}[n]$ and $\mathsf{MATIME}[n]$ do translate to some lower bounds for counting problems against small-space. However, the known reductions from classes like MA and QCMA to counting problems incur a quadratic blowup. Furthermore, there is evidence that a quadratic blowup is necessary for black-box techniques [8, 9]. As such, it appears we must either improve the lower bound exponent, or prove that we can bypass the quadratic blowup outside of black-box settings.

1.4 Organization

Section 2 covers relevant background, especially on alternation-trading proofs. In Section 3, we study alternation-trading proofs with generic slowdowns and prove Main Theorem 1. In Section 4, we use Grover's algorithm to prove Lemma 30, allowing us to obtain a generic slowdown with $\alpha = \frac{2}{3}$ and prove Main Theorem 2. In Section 5, we prove Main Theorem 3.

2 Preliminaries

We assume familiarity with classical complexity and quantum computing on the levels of [2] and [19], respectively.

2.1 Alternation-Trading Proofs

We start by defining various time-space complexity classes particular to this work. For a simple example of an alternation-trading proof, see Section 2.1.1 of the full version.

▶ **Definition 1.** $\mathsf{TS}[t(n)]$ *is the class of languages computable by a deterministic random-access machine using space* $n^{o(1)}$ *and time* $\tilde{O}(t(n))$ *on an* n*-bit input.* $\mathsf{BPTS}[t(n)]$ *is the class of languages computable by a two-sided error randomized random-access machine using space* $O(\log n)$ *and time* $\tilde{O}(t(n))$ *on an* n*-bit input.*

Note that a *randomized* random-access machine with random access to its input has only read-once access to its randomness.

▶ **Definition 2.** *For positive constants* $\{a_i\}_{i\geq 1}$ *and* $\{b_i\}_{i\geq 1}$ *and quantifiers* $Q_i \in \{\exists, \forall\}$*, the* ***alternating complexity class*** $(Q_1 n^{a_1})^{b_1}(Q_2 n^{a_2})^{b_2} \ldots (Q_k n^{a_k})^{b_k}\mathsf{TS}[n^d]$ *is the set of languages decidable by a machine operating in the following fashion on an* n*-bit input. Computation occurs in* $k+1$ *stages. In the* i^{th} *stage, for* $1 \leq i \leq k$*, the machine obtains from* Q_i *a string of length* $\tilde{O}(n^{a_i})$*. It then uses an* $n^{o(1)}$*-space machine and* $\tilde{O}(n^{b_i})$ *time to compute* $\tilde{O}(n^{b_i})$ *bits that are passed on to the next stage, taking as input the* $\tilde{O}(n^{a_i})$ *bit string from the quantifier and the* $\tilde{O}(n^{b_{i-1}})$*-bit input from the previous stage of computation. The input to the first stage* ($i = 1$) *is the* n*-bit input string itself. The verifier at the end receives an* $\tilde{O}(n^{b_k})$*-bit input and uses a* $n^{o(1)}$*-space machine and* $\tilde{O}(n^d)$ *time to compute the final answer. The criteria for acceptance and rejection are analogous to those for* $\Sigma_k\mathsf{P}$ *and* $\Pi_k\mathsf{P}$.

Note that our notation obscures $n^{o(1)}$ factors everywhere, although we may occasionally write out small factors for clarity. We index our b_i differently from the notation of [27], as our b_i is their b_{i+1}. This difference will be immaterial.

▶ **Definition 3.** *Given an alternating complexity class $(Q_1 n^{a_1})^{b_1} \dots (Q_k n^{a_k})^{b_k} \mathsf{TS}[n^d]$, we will refer to n^d as the* ***verifier runtime****.*

▶ **Lemma 4** (Speedup Lemma [18][15])**.** *For every $0 < x < d$,*

$$\mathsf{TS}[n^d] \subseteq (Qn^x)^{\max(x,1)}(\neg Qx \log n)^1 \mathsf{TS}[n^{d-x}].$$

Because our notation obscures $n^{o(1)}$ factors, we may write this as

$$\mathsf{TS}[n^d] \subseteq (Qn^x)^{\max(x,1)}(\neg Qn^0)^1 \mathsf{TS}[n^{d-x}]. \tag{3}$$

Proof. We will prove the lemma when $Q = \exists$; the other case follows because $\mathsf{TS}[n^d]$ is closed under complementation. The idea is that we can break up the transcript of a deterministic computation of length n^d into n^x pieces each of length n^{d-x}. Let M be an n^d time machine using $n^{o(1)}$ space. On an n-bit input, our Σ_2 machine will:

1. **Existentially** guess $n^x - 1$ intermediate machine configurations $X_1, \dots, X_{n^x-1}$ of M, each of size $n^{o(1)}$. These are passed, along with the input, to the next stage. This corresponds to the $(Qn^x)^{\max(x,1)}$ part of the class in (3).
2. **Universally** quantify over all intermediate configurations, picking one. There are n^x pieces so our quantifier only needs $O(\log n^x) \leq \tilde{O}(n^0)$ bits. If the quantifier picks the i^{th} configuration, then we pass the state pair (X_{i-1}, X_i) (along with the input) on to the next stage. We take X_0 to be the initial configuration of M, and X_n to be the (WLOG unique) accepting machine configuration. This corresponds to the $(\neg Qn^0)^1$ part in (3).
3. Given input x and a pair of configurations (X, Y) of M, the verifier simulates M starting at X for n^{d-x} steps, accepting if the configuration at the end is Y and rejecting otherwise. This corresponds to the $\mathsf{TS}[n^{d-x}]$ part of (3).

This completes the proof. ◀

As an extension, we may derive the speedup rule that we will use throughout this paper.

▶ **Corollary 5** ("Usual" Speedup Rule, [27])**.** *For every $0 < x < d$,*

$$\begin{aligned}&(Q_1 n^{a_1})^{b_1} \dots (Q_k n^{a_k})^{b_k} \mathsf{TS}[n^d] \\ &\qquad \subseteq (Q_1 n^{a_1})^{b_1} \dots (Q_k n^{\max(a_k,x)})^{\max(b_k,x)} (Q_{k+1} x \log n)^{b_k} \mathsf{TS}[n^{d-x}].\end{aligned}$$

Proof. Observe that we may merge together two quantifiers of the same type. Thus, taking $Q = Q_k$ in Lemma 4, we find that

$$\begin{aligned}&(Q_1 n^{a_1})^{b_1} \dots (Q_k n^{a_k})^{b_k} \mathsf{TS}[n^d] \\ &\qquad \subseteq (Q_1 n^{a_1})^{b_1} \dots (Q_k n^{a_k})^{b_k} (Q_k n^x)^{\max(b_k,x)} (Q_{k+1} x \log n)^{b_k} \mathsf{TS}[n^{d-x}] \\ &\qquad \subseteq (Q_1 n^{a_1})^{b_1} \dots (Q_k n^{\max(a_k,x)})^{\max(b_k,x)} (Q_{k+1} x \log n)^{b_k} \mathsf{TS}[n^{d-x}]\end{aligned}$$

where the second containment follows from Lemma 4. ◀

One might wonder whether we can do any better by also considering the containment arising from taking $Q = \neg Q_k$ in Lemma 4. It turns out that any alternation-trading proof using this latter rule can be obtained with Corollary 5, and therefore we may safely ignore this option. This is Lemma 3.2 of [27].

▶ **Definition 6.** *We refer to the value x in an application of the speedup rule as the* ***speedup parameter*** *for that application.*

In Section 5, we will work with alternating complexity classes with randomized space-bounded verifiers, rather than deterministic ones. We will use a speedup rule due to Diehl and van Melkebeek [10].

▶ **Theorem 7** ([10]). $\mathsf{BPTS}[n^d] \subseteq (\forall n^0)^1(\exists n^x)^{\max(x,1)}(\forall x \log n)\mathsf{TS}[n^{d-x}]$.

A sketch of the proof is in the full version. Note again that our notation allows us to hide $n^{o(1)}$ factors. We chose not to obscure the last $x \log n$ bits, as they will be extremely relevant later when we use Grover's algorithm to remove $O(\log n)$-bit quantifiers.

As before, we may express Theorem 7 as a rule that can be applied to alternating complexity classes.

▶ **Corollary 8** (The "Randomized" Speedup Rule). *For every* $0 < x < d$*,*

$$\begin{aligned}&(Q_1 n^{a_1})^{b_1} \ldots (Q_k n^{a_k})^{b_k}\mathsf{BPTS}[n^d]\\ &\quad \subseteq (Q_1 n^{a_1})^{b_1} \ldots (Q_k n^{a_k})^{b_k}(Q_{k+1} n^x)^{\max(b_k,x)}(Q_{k+2} x \log n)^{b_k}\mathsf{TS}[n^{d-x}].\end{aligned}$$

Note that unlike Corollary 5, which adds two quantifiers to speed up a deterministic computation, Corollary 8 adds three[7].

We've already introduced the usual slowdown lemma, which we restate for convenience.

▶ **Lemma 9** (Slowdown Lemma [11]). *Assume that* $\mathsf{NTIME}[n] \subseteq \mathsf{TS}[n^c]$ *for some* $c > 1$*. Then for all* $d \geq 1$*,* $\mathsf{NTIME}[n^d] \cup \mathsf{coNTIME}[n^d] \subseteq \mathsf{TS}[n^{cd}]$*.*

The slowdown rule follows from a padding argument and the observation that $\mathsf{TS}[n^c]$ is closed under complementation.

▶ **Corollary 10** ("Usual" Slowdown Rule [27]). *Assuming* $\mathsf{NTIME}[n] \subseteq \mathsf{TS}[n^c]$*, we have*

$$\begin{aligned}&(Q_1 n^{a_1})^{b_1} \ldots (Q_k n^{a_k})^{b_k}\mathsf{TS}[n^d]\\ &\quad \subseteq (Q_1 n^{a_1})^{b_1} \ldots (Q_{k-1} n^{a_{k-1}})^{b_{k-1}}\mathsf{TS}[n^{c\cdot\max(d,b_k,a_k,b_{k-1})}].\end{aligned}$$

Note that the exponent b_{k-1} is present in the maximum, because our assumption is $\mathsf{NTIME}[n] \subseteq \mathsf{TS}[n^c]$ rather than $\mathsf{NTIME}[n^\delta] \subseteq \mathsf{TS}[n^{c\delta}]$ for all $\delta > 0$.

▶ **Definition 11.** *Let* $c, \alpha \in \mathbb{R}$ *such that* $0 < \alpha \leq 1 < c$*. A* ***generic slowdown rule with parameters*** $\boldsymbol{\alpha}$ ***and*** $\boldsymbol{c}$ *shows that*

$$(Q_1 n^{a_1})^{b_1} \ldots (Q_k n^{a_k})^{b_k}\mathsf{TS}[n^d] \subseteq (Q_1 n^{a_1})^{b_1} \ldots (Q_{k-1} n^{a_{k-1}})^{b_{k-1}}\mathsf{TS}[n^{c\cdot\max(\alpha d,b_k,a_k,b_{k-1})}].$$

Intuitively, having a generic slowdown rule with parameters c and α means that we can turn classes like

$$\exists\forall\ldots\forall\exists\mathsf{TIME}[n^d]$$

into

$$\exists\forall\ldots\forall\mathsf{TIME}[n^{\alpha c d}].$$

We are now ready to define alternation-trading proofs.

[7] In both cases, the first quantifier is absorbed into the previous quantifier if one exists, in which case the number of "new" quantifiers is one and two respectively.

▶ **Definition 12** ([27]). *An **alternation-trading proof** is a list of alternating complexity classes, where each subsequent class on the list is contained in the previous class. Each class is derived from the previous by applying one of the following rules.*

1. *If the class is* $\mathsf{TS}[n^d]$ *(i.e., the verifier is deterministic and there are no quantifiers), we may apply Lemma 4:*

$$\mathsf{TS}[n^d] \subseteq (\exists n^x)^{\max(x,1)}(\forall x \log n)^1 \mathsf{TS}[n^{d-x}]$$

for some $x \in (0, d)$*.*

2. *If the class has at least one quantifier and the verifier is deterministic (i.e., the class ends with* $\mathsf{TS}[n^d]$*), we may apply Corollary 5:*

$$\begin{aligned}&(Q_1 n^{a_1})^{b_1} \ldots (Q_k n^{a_k})^{b_k} \mathsf{TS}[n^d]\\ &\quad\subseteq (Q_1 n^{a_1})^{b_1} \ldots (Q_k n^{\max(a_k,x)})^{\max(b_k,x)} (Q_{k+1} x \log n)^{b_k} \mathsf{TS}[n^{d-x}]\end{aligned}$$

for some $x \in (0, d)$*.*

3. *If the class is* $\mathsf{BPTS}[n^d]$ *(i.e., the verifier is randomized and there are no quantifiers), we may apply Theorem 7:*

$$\mathsf{BPTS}[n^d] \subseteq (\exists n^0)^1 (\forall n^x)^{\max(x,1)} (\exists x \log n) \mathsf{TS}[n^{d-x}]$$

for some $x \in (0, d)$*.*

4. *If the class has at least one quantifier and the verifier is randomized (i.e., the class ends with* $\mathsf{BPTS}[n^d]$*), we may apply Corollary 8:*

$$\begin{aligned}&(Q_1 n^{a_1})^{b_1} \ldots (Q_k n^{a_k})^{b_k} \mathsf{BPTS}[n^d]\\ &\quad\subseteq (Q_1 n^{a_1})^{b_1} \ldots (Q_k n^{a_k})^{b_k} (Q_{k+1} n^x)^{\max(b_k,x)} (Q_{k+2} x \log n)^{b_k} \mathsf{TS}[n^{d-x}]\end{aligned}$$

for some $x \in (0, d)$*.*

5. *If the class has at least one quantifier, and a generic slowdown rule with parameters* α *and* c *hold for the class, we may apply it:*

$$\ldots (Q_k n^{a_k})^{b_k} (\mathsf{BP})\mathsf{TS}[n^d] \subseteq \ldots (Q_{k-1} n^{a_{k-1}})^{b_{k-1}} (\mathsf{BP})\mathsf{TS}[n^{c \cdot \max(\alpha d, b_k, a_k, b_{k-1})}].$$

*We say that an alternation-trading proof **shows a contradiction for c** if it contains an application of a speedup rule and the proof shows either* $\mathsf{TS}[n^d] \subseteq \mathsf{TS}[n^{d'}]$ *for* $d' \leq d$ *or* $\mathsf{BPTS}[n^d] \subseteq \mathsf{BPTS}[n^{d'}]$ *for* $d' \leq d$*.*

Note that rules 3 and 4 only apply when proving lower bounds against BPTS.

The containment $\mathsf{TS}[n^d] \subseteq \mathsf{TS}[n^{d'}]$ for $d \geq d'$ does not automatically yield a contradiction[8]. Fortunately however, we are still able to derive contradictions from this.

▶ **Theorem 13** (Lemma 3.1 of [27]). *If, under the assumption* $\mathsf{NTIME}[n] \subseteq \mathsf{TS}[n^c]$*, there is an alternation-trading proof with at least two inclusions showing that* $\mathsf{TS}[n^d] \subseteq \mathsf{TS}[n^{d'}]$ *for* $d' \leq d$*, then the assumption must have been false and* $\mathsf{NTIME}[n] \not\subseteq \mathsf{TS}[n^c]$

[8] To apply the naive approach, we need a single machine in $\mathsf{TS}[n^d]$ that can simulate everything in $\mathsf{TS}[n^{d'}]$. However, any fixed machine in $\mathsf{TS}[n^d]$ cannot simulate things use more space than it does. If our simulating machine in $\mathsf{TS}[n^d]$ uses space $f(n) = n^{o(1)}$ then there is always a machine in $\mathsf{TS}[n^{d'}]$ that uses more space while still being $n^{o(1)}$ and our simulating machine cannot simulate this one.

In Section 5, we will show that similar statements hold for BPTS in the contexts in which we need them to hold, thus allowing us to derive contradictions from $\mathsf{BPTS}[n^d] \subseteq \mathsf{BPTS}[n^{d'}]$ for $d' \leq d$.

▶ **Definition 14** ([27]). *An alternating complexity class* $(Q_1 n^{a_1})^{b_1} \dots (Q_k n^{a_k})^{b_k} (\mathsf{BP})\mathsf{TS}[n^d]$ *is* ***orderly*** *if for all* $i \in [k]$, $a_i \leq b_i$.

▶ **Lemma 15** ([27]). *Any alternation-trading proof using rules from Definition 12 is orderly.*

As noted by Buss and Williams [7], Lemma 15 implies that, when describing an alternation-trading proof consisting of speedups and slowdowns, it is sufficient to only specify the $\{b_i\}$ and disregard the $\{a_i\}$. We will be somewhat more careful when we apply Grover's algorithm in the quantum setting, but when we can we will simplify our notation by writing only a single exponent inside the parenthesis. Thus, we may abuse notation to write alternating complexity classes in the form $(Q_1 n^{a_1}) \dots (Q_k n^{a_k})(\mathsf{BP})\mathsf{TS}[n^d]$, where the a_i are then understood to be the maxima of the corresponding pairs of exponents in the full notation.

▶ **Definition 16** ([27]). *A* ***proof annotation*** *is a way of specifying a sequence of applications of speedup and slowdown rules. We write* **1** *to denote a speedup rule and* **0** *to denote a (possibly generic) slowdown rule. When appropriate, we put a subscript under a* **1** *to denote the speedup parameter used for that speedup rule application.*

In this paper, we will work with alternation-trading proofs which apply only one slowdown rule (many times). For such proofs, the sequence of speedups and slowdowns fully determines whether the verifier is deterministic or randomized at a given line of the proof. This means that, when specifying a proof annotation, we do not need to specify which speedup rule we are applying between Corollary 5 and Corollary 8. When the verifier is randomized we must apply Corollary 8, and when the verifier is deterministic we should always apply Corollary 5 as it is strictly more efficient than Corollary 8.

2.2 Computational Model

All functions used to bound runtime or space are assumed to be constructible in the given resources. Our model for classical computation is the space-bounded random-access machine, unless specified otherwise. Our proofs are robust to all notions of random access we know.

Our model for quantum computation will be that of Van Melkebeek and Watson [23], but our results hold for any reasonable quantum model capable of *obliviously* applying unitaries from a fixed universal set and with quantum random-access to the input.[9] Recall that if $x \in \{0,1\}^n$ is an input, an algorithm is said to have quantum random-access to x if it can perform the transformation

$$\sum_{i\in[n]} \alpha_i \left|i\right\rangle \left|b\right\rangle \mapsto \sum_{i\in[n]} \alpha_i \left|i\right\rangle \left|b \oplus x_i\right\rangle,$$

where i is an index and x_i denotes the bit at the i^{th} position of x. The model of [23] is capable of simulating all the usual models of quantum computing, and deals carefully with issues like intermediate measurements and numerical precision.

[9] Here, "obliviously" means that the unitaries applied depend only on the length of the input.

2.3 Some Atypical Complexity Classes

We stated our lower bounds in Section 1 in terms of QCMA and MA. However, our results actually hold for slightly smaller classes, which we describe below.

▶ **Definition 17.** *The complexity class* $\exists \cdot \mathsf{BQP}$ *is the set of languages* L *for which there exists a* BQP *verifier* $\mathcal{A}$ *such that*

- $x \in L \implies (\exists w) \Pr[\mathcal{A}(x,w) = 1] \geq \frac{2}{3}$
- $x \notin L \implies (\forall w) \Pr[\mathcal{A}(x,w) = 1] \leq \frac{1}{3}$.

We will write $\exists \cdot \mathsf{BQP}_{s,c}$ *to denote* $\exists \cdot \mathsf{BQP}$ *where Arthur has completeness* c *and soundness* s. *We will write* $\exists \cdot \mathsf{BQTIME}[t(n)]$ *to denote* $\exists \cdot \mathsf{BQP}$ *where the length of Merlin's proof and the runtime of the verifier are both* $O(t(n))$.

We may define $\exists \cdot \mathsf{BPP}$ and $\exists \cdot \mathsf{BPTIME}$ analogously.

▶ Remark 18. Note that, while $\exists \cdot \mathsf{BQP} \subseteq \mathsf{QCMA} \subseteq \mathsf{QMA}$ (respectively, $\exists \cdot \mathsf{BPP} \subseteq \mathsf{MA}$), it is not clear if $\exists \cdot \mathsf{BQP} = \mathsf{QCMA}$ ($\exists \cdot \mathsf{BPP} = \mathsf{MA}$) due to differences in the promise conditions. In $\exists \cdot \mathsf{BQP}$, we require that the verifier $\mathcal{A}(x,w)$ lie in BQP, meaning that it satisfies the promise on every input pair (x,w). On the other hand, in the "yes" case of QCMA (when a string x is in the language), we require only that there exists a polynomial-sized witness y making the verifier $\mathcal{A}(x,w)$ accept with probability exceeding $\frac{2}{3}$. This does not preclude the existence of a witness string w' such that $\frac{1}{3} < \Pr[\mathcal{A}(x,w) = 1] < \frac{2}{3}$.

3 Lower Bounds With Generic Slowdowns

We will start by introducing a method to reduce the verifier runtime of a class without increasing any of the quantifier exponents, assuming that the verifier runtime isn't too large. This was the main feature in the alternation-trading proofs of [26] that allowed improvement beyond the results of Lipton-Fortnow-Van Melkebeek-Viglas [11].

▶ **Lemma 19** (Generalizes [26]). *Let* $0 < \alpha \leq 1$ *be a real number. Let* c *be a positive real such that* $c < \frac{1+\alpha}{\alpha}$. *Given any class*

$$\ldots(Q_k n^{a_k})\mathsf{TS}[n^{a_{k+1}}]$$

where $ca_k \leq a_{k+1} < \frac{\alpha c}{\alpha c - 1} a_k$, *there is a nonnegative integer* $N := N(a_k)$ *such that the annotation* $\mathbf{(10)^N0}$ *with the appropriate speedup parameters proves that*

$$\ldots(Q_k n^{a_k})\mathsf{TS}[n^{a_{k+1}}] \subseteq \ldots(Q_k n^{a_k})\mathsf{TS}[n^{ca_k}] \subseteq \ldots\mathsf{TS}[n^{c^2 a_k}].$$

We prove this lemma in the full version. The use of a_k as the speedup parameter in Lemma 19 may seem arbitrary, but it will turn out that this is in fact the best speedup parameter in this setting. Based on this lemma, we can define a new rule that we may use in alternation-trading proofs.

▶ **Definition 20.** *Consider an alternating complexity class* $(Q_1 n^{a_1})\ldots(Q_k n^{a_k})\mathsf{TS}[n^d]$. *Given* $0 < \alpha \leq 1$ *and* $c > 0$ *satisfying* $c < \frac{1+\alpha}{\alpha}$, *we define the* ***wiggle rule*** *with parameters* α, c *to be the following:*

- *If* $d < \frac{\alpha c}{\alpha c - 1} a_k$, *apply* $\mathbf{(1_{a_k}0)^t}$ *for* $t := \left\lceil \frac{a_{t+1}}{a_t} \right\rceil$.
- *Otherwise, do nothing.*

We call an application of the wiggle rule ***proper*** *if we are in the first case and* ***improper*** *otherwise.* **In a proof annotation, when α and c are fixed, we will denote an application of the wiggle rule by 2.**

It is worth remembering the ratio

$$\frac{\alpha c}{\alpha c - 1}$$

as it will show up frequently in the remainder of this paper. Following the notation of Definition 20, if the value of d (the verifier runtime exponent) for an alternating complexity class is at most $\frac{\alpha c}{\alpha c-1}$ times a_k (the exponent in the final quantifier), we may reduce the verifier runtime to its smallest possible value ca_k with an application of Definition 20 and Lemma 19, as the following corollary shows.

▶ **Corollary 21** (Corollary of Lemma 19). *Consider a class*

$$(Q_1 n^{a_1}) \dots (Q_k n^{a_k})\mathsf{TS}[n^d].$$

Given $0 < \alpha \le 1$ and $c > 0$ satisfying $c < \frac{1+\alpha}{\alpha}$, applying the wiggle rule (Definition 20) yields the class:

- $(Q_1 n^{a_1}) \dots (Q_k n^{a_k})\mathsf{TS}[n^{ca_k}]$, *if $d < \frac{\alpha c}{\alpha c-1} a_k$.*
- $(Q_1 n^{a_1}) \dots (Q_k n^{a_k})\mathsf{TS}[n^d]$ *otherwise.*

We are now in a position to prove Main Theorem 1, which we restate for convenience.

▶ **Main Theorem 1.** *Fix $\alpha \in (0,1]$ and let $\mathcal{C}$ be a complexity class. Let r_1 be the largest root of the polynomial $P_\alpha(x) := \alpha^2 x^3 - \alpha x^2 - 2\alpha x + 1$. If $\mathcal{C} \subseteq \mathsf{TS}[n^c]$ implies a generic slowdown rule with parameters α and c, then $\mathcal{C} \not\subseteq \mathsf{TS}[n^c]$ for $c < r_1$.*

We will use the following lemma in the proof of Main Theorem 1; its proof can be found in the full version.

▶ **Lemma 22.** *If r_1 and r_2 are the two largest roots of $P_\alpha(x) := \alpha^2 x^3 - \alpha x^2 - 2\alpha x + 1$ then*

$$r_2(\alpha) < \frac{1+\sqrt{1+4\alpha}}{2\alpha} < r_1(\alpha) < \frac{1+\alpha}{\alpha}$$

for all $\alpha > 0$.

Proof of Main Theorem 1. The lower bound will follow from applying the annotation $\boldsymbol{1^k 0(20)^k}$ as $k \to \infty$. (Recall that $\mathbf{2}$ denotes an application of the wiggle rule of Definition 20.) We will choose speedup parameters $\{x_i\}$ so that the following sequence of inclusions is valid and every application of the wiggle rule is proper.

$$\begin{aligned}
\mathsf{TS}[n^d] &\subseteq (\exists n^{x_1})(\forall n^{x_2}) \dots (Q_k n^{x_k})(Q_{k+1} n^{x_k})\mathsf{TS}[n^{(d-x_1-x_2-\dots-x_k)}] && \mathbf{1^k}0(20)^k \\
&\subseteq (\exists n^{x_1})(\forall n^{x_2}) \dots (Q_k n^{x_k})\mathsf{TS}[n^{\alpha c(d-x_1-x_2-\dots-x_k)}] && 1^k\mathbf{0}(20)^k \\
&\subseteq (\exists n^{x_1})(\forall n^{x_2}) \dots (Q_k n^{x_k})\mathsf{TS}[n^{cx_k}] && 1^k0\mathbf{2}0(20)^{k-1} \\
&\subseteq (\exists n^{x_1})(\forall n^{x_2}) \dots (Q_{k-1} n^{x_{k-1}})\mathsf{TS}[n^{\alpha c^2 x_k}] && 1^k02\mathbf{0}(20)^{k-1} \\
&\subseteq (\exists n^{x_1})(\forall n^{x_2}) \dots (Q_{k-1} n^{x_{k-1}})\mathsf{TS}[n^{cx_{k-1}}] && 1^k020\mathbf{2}0(20)^{k-2} \\
&\dots && 1^k02020\mathbf{(20)^{k-2}} \\
&\subseteq \mathsf{TS}[n^{\alpha c^2 x_1}].
\end{aligned}$$

In order to ensure a contradiction at the end, we will set $x_1 := \frac{d}{\alpha c^2}$. In order to ensure that the first application of the wiggle rule is proper, we require that the $\{x_i\}$ satisfy

$$\alpha c(d - x_1 - \dots - x_k) < \frac{\alpha c}{\alpha c - 1} x_k \iff d - x_1 - x_2 - \dots - x_k < \frac{1}{\alpha c - 1} \cdot x_k. \tag{4}$$

In order to ensure that we can apply the wiggle rule properly in all other steps, we require that the $\{x_i\}$ satisfy

$$\alpha c^2 x_i < \frac{\alpha c}{\alpha c - 1} \cdot x_{i-1} \tag{5}$$

for all $2 \leq i \leq k$.

The next claim lets us find valid speedup parameters $\{x_i\}$ for certain values of c depending on the number of iterations k that we allow in our annotation. The expression will look somewhat hairy, but fortunately most of it will disappear in the limit as k becomes large.

▷ Claim 23. Pick $\epsilon > 0$, and let $\tau := \frac{1-\epsilon}{c(\alpha c-1)}$. There are choices of the speedup parameter x_i such that the annotation $\boldsymbol{1^k 0 (20)^k}$ implies a contradiction for all c satisfying

$$\alpha c^2 - \frac{\tau^k - 1}{\tau - 1} < \frac{\tau^k}{\alpha c - 1}. \tag{6}$$

Proof of Claim. Let $x_i := \left(\frac{1-\epsilon}{c(\alpha c-1)}\right)^{i-1} x_1 = \left(\frac{1-\epsilon}{c(\alpha c-1)}\right)^{i-1} \frac{d}{c^2}$ for $i \geq 2$. Observe that all the x_i are positive and satisfy the constraints of (5) for all i. If we take d to be sufficiently large we can ensure that every exponent in the proof exceeds 1. Therefore, if we can show that our selection of $\{x_i\}$ satisfies the first constraint, (4), we will have a valid sequence of rules and, by our choice of x_1 above, have derived a contradiction.

Plugging our choice of $\{x_i\}$ into (4) yields the constraint

$$d - \frac{d}{\alpha c^2}\left(1 + \frac{1-\epsilon}{c(\alpha c)} + \cdots + \left(\frac{1-\epsilon}{c(\alpha c-1)}\right)^{k-1}\right) < \frac{1-\epsilon}{\alpha c - 1}\left(\left(\frac{1-\epsilon}{c(\alpha c-1)}\right)^{k-1} \frac{d}{\alpha c^2}\right). \tag{7}$$

This is equivalent to

$$\alpha c^2 - \frac{\tau^k - 1}{\tau - 1} < \frac{\tau^k}{\alpha c - 1}, \tag{8}$$

as desired. ◁

The lemma condition $c > \frac{1+\sqrt{1+4\alpha}}{2\alpha}$ means that $(c(\alpha c - 1))^{-1} < 1$. Now, let $\epsilon := \frac{1}{k}$ and allow $k \to \infty$ in (6). Since $\tau \to (c(\alpha c-1))^{-1}$ as $k \to \infty$, we see that (6) becomes

$$\alpha c^2 - \frac{c(\alpha c - 1)}{c(\alpha c-1) - 1} < 0 \iff \alpha^2 c^3 - \alpha c^2 - 2\alpha c + 1 < 0$$

in the limit. More formally, we have shown that for every c satisfying this constraint, if we take k to be large enough, then our choice of $\{x_i\}$ satisfies (4). The leading coefficient is positive so we satisfy all the constraints (i.e. the alternation-trading proof shows a contradiction) when c lies between the largest and second largest roots of this cubic. Implicitly, we require $c < \frac{1+\alpha}{\alpha}$ so that we can apply Lemma 19 and $c > \frac{1+\sqrt{1+4\alpha}}{2\alpha}$ so that the undesired terms in (6) vanish. By Lemma 22, these are satisfied when $\frac{1+\sqrt{1+4\alpha}}{2\alpha} < c < r_1$. Furthermore, if $\mathcal{C} \not\subseteq \mathsf{TS}[n^{c'}]$ for some c' then we automatically have $\mathcal{C} \not\subseteq \mathsf{TS}[n^c]$ for all $c < c'$. ◀

From Main Theorem 1, we immediately obtain the following corollary.

▶ **Corollary 24** ([25]). *For $c < 2\cos(\frac{\pi}{7}) \approx 1.801$, $\mathsf{NTIME}[n] \not\subseteq \mathsf{TS}[n^c]$.*

Proof. Taking $\alpha = 1$ (as this is normal slowdown) yields $c^3 - c^2 - 2c + 1 < 0$. The largest root is $2\cos(\frac{\pi}{7})$, so we have $\mathsf{NTIME}[n] \not\subseteq \mathsf{TS}[n^{2\cos(\frac{\pi}{7})-o(1)}]$ ◀

4 A Generic Slowdown Rule From Grover's Algorithm

Now that we've proved the general case, we turn to an application. Observe that any application of a speedup rule with parameter x results in an alternating complexity class whose final quantifier is over $x \log n$ bits, as this quantifier indexes over the n^x states that it receives from the penultimate quantifier. For example,

$$\mathsf{TS}[n^d] \subseteq (\exists n^x)(\forall x \log n)^1 \mathsf{TS}[n^{d-x}].$$

Normally, we could remove the last quantifier with a slowdown rule, yielding the inclusion

$$\mathsf{TS}[n^d] \subseteq (\exists n^x)\mathsf{TS}[n^{c \cdot \max(x, d-x, 1)}]. \tag{9}$$

We could also remove the quantifier by having the deterministic verifier try all n^x possible strings it could receive, but this yields the useless inclusion $\mathsf{TS}[n^d] \subseteq (\exists n^x)\mathsf{TS}[n^d]$. However, if we allow alternating complexity classes with quantum verifiers rather than deterministic verifiers, we can remove the last quantifier more efficiently. In particular, we can think of the last quantifier as a search problem over a space of size $N := n^x$. Classically, no blackbox algorithm can search over N items with fewer than N queries in the worst case, but in the quantum setting Grover's algorithm lets us do this in $O(\sqrt{N})$ queries! This gives us, for some informal notion of quantum time,

$$\mathsf{TS}[n^d] \subseteq (\exists n^x)\mathsf{QTIME}[n^{d-\frac{x}{2}}]. \tag{10}$$

This method of quantifier removal allows us to remove quantifiers more efficiently than (9) when c is large at the cost of introducing a quantum verifier. However, the quantum verifier can be replaced with a deterministic verifier, under the assumption $\mathsf{QCMATIME}[n] \subseteq \mathsf{TS}[n^c]$! Ultimately, we will find that combining a speedup (adding one quantifier), (10) (removing one quantifier), and the assumption (removing one quantifier) yields a generic slowdown rule with $\alpha = \frac{2}{3}$.

4.1 Review of Grover's Algorithm

Given (quantum) query access to a function $f \colon [N] \to \{0, 1\}$, Grover's algorithm tells us whether or not there exists an $\alpha \in [N]$ such that $f(\alpha) = 1$. For a given f, let $S := \{\alpha \in [N] \colon f(\alpha) = 1\}$. It turns out, per [12], that the probability of success of Grover search after j iterations is $\sin^2(2(j+1)\theta)$ where $\theta = \sin^{-1}\sqrt{\frac{|S|}{N}}$. Regardless of $\frac{|S|}{N}$, a sufficiently large random number of iterations should succeed with probability roughly $\frac{1}{2}$, which is the average value of $\sin^2 x$. The following lemma of [5] formalizes this.

▶ **Lemma 25** ([5]). *Let k be an arbitrary positive integer and let j be an integer chosen uniformly at random from $[0, k-1]$. If we observe the register after applying j Grover iterations starting from the uniform state, the probability of obtaining a solution is at least $\frac{1}{4}$ when $k \geq \frac{1}{\sin 2\theta}$.*

▶ **Corollary 26.** *Let j be an integer chosen uniformly at random from $[0, (\sin \frac{2}{\sqrt{N}})^{-1}]$. If we observe the register after applying j Grover iterations starting from the uniform state, the probability of obtaining a solution is at least $\frac{1}{4}$.*

4.2 Using Grover's Algorithm to Invert RAMs

In order to apply Grover's algorithm to remove the quantifier in expressions like $(\exists \log n)\mathsf{TS}[n^d]$ – specifically, to perform Grover diffusion – we must be able to implement the relevant function $f \in \mathsf{TS}[n^d]$ in our quantum computational model. The following lemma converts the classical random-access machine of f to a "normal form" that is more amenable to implementation in a quantum computer. The workspace of A includes an input address register used to specify which bit of the input the machine wishes to read and an input query bit which stores the result of the most recent query of the input. Assume without loss of generality that the first $\lceil \log n \rceil$ bits of the workspace hold the input address register, and bit $\lceil \log n \rceil + 1$ holds the current bit of input being read.

▶ **Lemma 27** (Normal Form Circuit for Time-Space Bounded Computation). *Let f be a function computed by a random-access machine A in time $t(n)$ and space $s(n)$ on inputs of length n. Then, there exists a sequence of uniform Boolean circuits $\mathcal{C}_1, \ldots, \mathcal{C}_r$, such that:*

1. *for all $1 < i \leq r - 1$, $\mathcal{C}_i$ has s inputs and s outputs,*
2. *$\mathcal{C}_1$ has no inputs and s outputs,*
3. *$\mathcal{C}_r$ has s inputs and 1 output,*
4. *$\sum_{i=1}^{r} |\mathcal{C}_i| = O(ts^2)$ where $|\mathcal{C}_i|$ denotes the size of $\mathcal{C}_i$,*

and furthermore

$$f(y) = \mathcal{C}_r \circ R \circ \mathcal{C}_{r-1} \circ \cdots \circ R \circ \mathcal{C}_1, \tag{11}$$

where $R \colon \{0,1\}^{s(n)} \to \{0,1\}^{s(n)}$, on input $z \in \{0,1\}^{s(n)}$, sets $z_{\lceil \log n \rceil + 1} = y_{z_1 z_2 \ldots z_{\lceil \log n \rceil}}$ and leaves the remainder of z unchanged. Here, we write s_j to denote the j^{th} bit of a string s.

In Lemma 27, the circuits $\mathcal{C}_i$ are used to simulate steps in the workspace of the machine, and the function R perform random accesses to the n-bit input y, by setting its $(\lceil \log n \rceil + 1)^{\text{th}}$ output bit to the bit of y whose index is specified by the first $\lceil \log n \rceil$ bits. Note that y is the original length-n input to f and n may be much larger than $s(n)$. Lemma 27, along with Corollary 28 allow us to translate a RAM to a quantum circuit with $s(n)$ wires and with intermittent QRAM calls: the Boolean circuits $\{\mathcal{C}_i\}$ can be replaced with quantum circuits in the usual way and each R can be replaced by a QRAM call. Proofs of Lemma 27 and Corollary 28 can be found in the full version.

▶ **Corollary 28.** *Let $f : \{0,1\}^m \to \{0,1\}$ be a function computable in classical time t and space s by a random-access machine. Then, the transformation $|y\rangle |0\rangle \mapsto |y\rangle |f(y)\rangle$ can be implemented by a quantum computer with quantum random-access to its input in time $O(ts^2)$.*

We now state the main lemma of this subsection.

▶ **Lemma 29** (Grover's Algorithm for Time-Space Bounded Computation). *Given a function $f : \{0,1\}^m \to \{0,1\}$ that can be computed in classical time t and space s by a random-access machine, there is a quantum algorithm taking time $O(2^{\frac{m}{2}}(ts^2 + m))$ that finds a value $\alpha \in \{0,1\}^m$ such that $f(\alpha) = 1$, assuming one exists, with probability at least $\frac{2}{3}$. In particular, when $m = x \log n, t = n^d, s = O(\log n)$, we obtain a time bound of $\tilde{O}(n^{d+\frac{x}{2}})$.*

Proof. (Sketch) We run Grover's algorithm. Per Corollary 26, we need $O(2^{\frac{m}{2}})$ Grover iterations. Grover diffusion is dominated by the cost of implementing the function itself, which by Corollary 28 is $O(ts^2)$. Inversion about the mean takes $O(m)$ time. ◀

4.3 Proving Main Theorem 2

We are now in a position to apply Grover's algorithm to prove a generic slowdown rule.

▶ **Lemma 30.** *If* $\exists \cdot \mathsf{BQTIME}_{\frac{1}{3},1}[n] \subseteq \mathsf{TS}[n^c]$ *then*

$$\ldots(Q_{k-1}n^{a_{k-1}})^{b_{k-1}}(Q_k n^{a_k})^{b_k}\mathsf{TS}[n^d] \subseteq \ldots(Q_{k-1}n^{a_{k-1}})^{b_{k-1}}\mathsf{TS}[n^{c\cdot\max(a_k,b_k,b_{k-1},1,\frac{2d}{3})}].$$

Therefore, under the assumption $\exists \cdot \mathsf{BQTIME}_{\frac{1}{3},1}[n] \subseteq \mathsf{TS}[n^c]$ we may apply a generic slowdown rule with parameters $\alpha = \frac{2}{3}$ and c. Lemma 30 corresponds to the following sequence of operations:

1. Classical Speedup (Corollary 5)
2. Grover's algorithm to invert a classical function with classical input (Lemma 29)
3. Direct application of $\exists \cdot \mathsf{BQTIME}[n] \subseteq \mathsf{TS}[n^c]$

Proof. By classical speedup, we have

$$\ldots(Q_k n^{a_k})^{b_k}\mathsf{TS}[n^d] \subseteq \ldots(Q_k n^{\max(a_k,x)})^{\max(b_k,x)}(Q_{k+1}x\log n)^{b_k}\mathsf{TS}[n^{d-x}]. \tag{12}$$

Consider the function $g\colon \{0,1\}^{n^{\max(b_k,x)}} \times \{0,1\}^{x\log n} \to \{0,1\}$ which implements the $\mathsf{TS}[n^{d-x}]$ verifier from the right-hand-side of (12) given as inputs the $n^{\max(b_k,x)}$ bits of output from the k^{th} stage/quantifier and the $(x\log n)$-bit string chosen by the $(k+1)^{\text{th}}$ stage/quantifier. We will apply Lemma 29 to the function $g_z := g(z,\cdot)$, where z is the output from the k^{th} stage of the class (not the $(k+1)^{\text{th}}$ stage!). In particular, we can use Grover's algorithm to search over the space of possible values of the last quantifier of $(x\log n)$ bits. Thus, the inputs to g_z are strings of length $m = x\log n$. The runtime of g_z is $O(ts^2)$ by Corollary 28, as g_z just needs to evaluate the verifier on z (the output of the k^{th} stage) and a length m-input. Therefore, applying Lemma 29,

$$\subseteq \ldots(Q_{k-1}n^{a_{k-1}})^{b_{k-1}}(Q_k n^{\max(a_k,x)})^{\max(b_k,x)}\mathsf{BQTIME}[n^{d-\frac{x}{2}}].$$

Without loss of generality, suppose that $Q_k = \exists$. Then, we can continue the sequence of inclusions as follows:

$$\begin{aligned}&\subseteq \ldots(Q_{k-1}n^{a_{k-1}})^{b_{k-1}}\exists \cdot \mathsf{BQTIME}[n^{\max(a_k,b_k,x,d-\frac{x}{2})}]\\ &\qquad\subseteq \ldots(Q_{k-1}n^{a_{k-1}})^{b_{k-1}}\mathsf{TS}[n^{c\cdot\max(b_{k-1},a_k,b_k,x,d-\frac{x}{2},1)}].\end{aligned}$$

(Note we need a 1 in the maximum in the last equation, because our assumption $\exists \cdot \mathsf{BQTIME}_{\frac{1}{3},1}[n] \subseteq \mathsf{TS}[n^c]$ only implies $\exists \cdot \mathsf{BQTIME}[n^\delta] \subseteq \mathsf{TS}[n^c]$ for $\delta < 1$.) To minimize the exponent, we take $x = \frac{2a_0}{3}$, yielding the exponent in the lemma statement. ◀

▶ Remark 31. Note that in the proof of Lemma 30 we do not need to account for the the n^{b_k} exponent on the $(k+1)^{\text{th}}$ stage/quantifier when computing the the runtime of g_z when preparing to apply Lemma 29. This is because the quantifier $(Q_{k+1}x\log n)^{b_k}$ on the right-hand-side of (12), which arises due to a speedup rule, has a b_k in the exponent only because it needs to copy and pass on the output of the k^{th} stage. This is normally necessary because the verifier on the right-hand-side of (12) needs to run from configuration to configuration on the original verifier's input and hence needs to be able to access the original input (i.e., the input to the verifier on the left-hand-side of the inclusion). However, by our definition of g_z, we don't need to worry about the cost of copying the output from the k^{th} stage z is itself the output of the k^{th} stage. We don't need to copy and pass on the original input because g_z already has it! Put differently, because we're collapsing two stages into one, we don't need to expend time on computation that's only used to pass input along.

Note that every possible proof involving slowdown and Grover's algorithm (Lemma 29) can be carried out using Lemma 30 as the only rule for removing quantifiers, as we can only apply Lemma 29 between a speedup and a slowdown. (More details on why are in the full version.) Plugging $\alpha = \frac{2}{3}$ into Main Theorem 1, we obtain the following corollary.

▶ **Corollary 32.** $\exists \cdot \mathsf{BQTIME}_{\frac{1}{3},1}[n] \not\subseteq \mathsf{TS}[n^c]$ *for* $c < \frac{3+\sqrt{3}}{2} \approx 2.366$.

Since $\exists \cdot \mathsf{BQTIME}[n] \subseteq \mathsf{QCMATIME}[n]$, this proves Main Theorem 2.

5 Lower Bounds Against BPTS

We turn to proving lower bounds against randomized algorithms. In this section, we will use both Corollary 5 and Corollary 8 as speedup rules. The randomized slowdown rule is straightforward.

▶ **Lemma 33** ("Randomized" Slowdown Rule). *Assuming* $\exists \cdot \mathsf{BPTIME}[n] \subseteq \mathsf{BPTS}[n^c]$, *we have*

$$(Q_1 n^{a_1})^{b_1} \dots (Q_k n^{a_k})^{b_k} \mathsf{BPTS}[n^d] \subseteq (Q_1 n^{a_1})^{b_1} \dots (Q_{k-1} n^{a_{k-1}})^{b_{k-1}} \mathsf{BPTS}[n^{c \cdot \max(d, b_k, a_k, b_{k-1})}].$$

▶ **Lemma 34.** *Suppose that, under the assumption* $\exists \cdot \mathsf{BPTIME}[n] \subseteq \mathsf{BPTS}[n^c]$ *for some* c, *there is an alternation-trading proof with at least two inclusions proving that* $\mathsf{BPTS}[n^d] \subseteq \mathsf{BPTS}[n^{d'}]$ *for* $d < d'$. *Then, the assumption is false.*

A proof of this lemma can be found in the full version. Recall that by Lemma 15, we may suppress some of the exponents when writing alternating complexity classes. We will do so throughout the remainder of this section.

Now that we have the preliminaries out of the way, let us prove some lower bounds. The first part of Main Theorem 3 is an immediate corollary of the following theorem.

▶ **Theorem 35.** $\exists \cdot \mathsf{BPTIME}[n] \not\subseteq \mathsf{TS}[n^c]$ *for* $c < r_1$, *where* $r_1 \approx 1.466$ *is the largest root of the polynomial* $x^3 - x^2 - 1 = 0$.

Proof. Our analysis is similar to the proof of Main Theorem 1. The bound will arise by applying the annotation $\mathbf{1^k 0^{k+2}}$ with the appropriate speedup parameters as $k \to \infty$. We will choose parameters $\{x_i\}$ so that the following sequence of inclusions is valid.

$$\begin{array}{lll}
\mathsf{BPTS}[n^d] & \subseteq (\exists n^1)(\forall n^{x_1})(\exists n^{x_2}) \dots (Q_k n^{x_k})(Q_{k+1} n^{x_k}) \mathsf{TS}[n^{(d - x_1 - x_2 - \dots - x_k)}] & \mathbf{1^k}0^{k+2} \\
& \subseteq (\exists n^1)(\forall n^{x_1})(\exists n^{x_2}) \dots (Q_k n^{x_k}) \mathsf{BPTS}[n^{c(d - x_1 - x_2 - \dots - x_k)}] & 1^k\mathbf{0}0^{k+1} \\
& \subseteq (\exists n^1)(\forall n^{x_1}) \dots (Q_k n^{x_k}) \mathsf{BPTS}[n^{c x_k}] & \\
& \subseteq (\exists n^1)(\forall n^{x_1}) \dots (Q_{k-1} n^{x_{k-1}}) \mathsf{BPTS}[n^{c^2 x_k}] & 1^k0\mathbf{0}0^k \\
& \subseteq (\exists n^1)(\forall n^{x_1}) \dots (Q_{k-1} n^{x_{k-1}}) \mathsf{BPTS}[n^{c x_{k-1}}] & \\
& \dots & 1^k00\mathbf{0^{k-1}}0 \\
& \subseteq (\exists n^1) \mathsf{BPTS}[n^{c^2 x_1}] & \\
& \subseteq \mathsf{BPTS}[n^{c^3 x_1}] & 1^k0^{k+1}\mathbf{0} \\
& \subseteq \mathsf{BPTS}[n^d] &
\end{array}$$

In order for all of the above inclusions to hold, we take

$$x_1 := \frac{d}{c^3}, \quad x_i := (c(1-\epsilon))^{1-i} \cdot x_1 = \frac{d(1-\epsilon)^{1-i}}{c^{i+2}}$$

for $\epsilon := \frac{1}{k}$. This automatically satisfies all the constraints except for the one corresponding to the third line, which requires

$$\begin{aligned}
&c(d - x_1 - x_2 - \cdots - x_k) \leq cx_k \\
\iff &1 - \frac{1}{c^3}\left(\frac{1-\epsilon}{c}\right)^{k-1} - \frac{1}{c^3}\sum_{i=0}^{k-1}\left(\frac{1-\epsilon}{c}\right)^{i} < 0 \\
\iff &1 - \frac{1}{c^3}\left(\frac{1-\epsilon}{c}\right)^{k-1} - \frac{1}{c^3}\frac{1-\frac{1-\epsilon}{c^k}}{1-\frac{1-\epsilon}{c}} < 0.
\end{aligned}$$

As $k \to \infty$, several terms vanish. We are left with

$$1 - \frac{1}{c^3(1-\frac{1}{c})} < 0 \iff c^3 - c^2 - 1 < 0.$$

The largest root is at $c \approx 1.466$. ◀

When we are allowed to introduce quantum operations and use Lemma 30, we can do slightly better than this.

▶ **Theorem 36.** $\exists \cdot \mathsf{BQTIME}[n] \not\subseteq \mathsf{TS}[n^c]$ *for* $c < 1.5$.

The second part of Main Theorem 3 is an immediate corollary of the previous theorem.

Proof. (Sketch) Let $d > 1.5$. We have the following sequence of containments:

$$\begin{aligned}
\mathsf{BPTS}[n^d] &\subseteq (\exists n^1)(\forall n^{\frac{2d}{3}})(\exists \frac{2d}{3}\log n)^1\mathsf{TS}[n^{\frac{d}{3}}] \\
&\subseteq (\exists n^1)(\forall n^{\frac{2d}{3}})\mathsf{BQTIME}[n^{\frac{2d}{3}}] \\
&\subseteq (\exists n^1)\mathsf{BPTS}[n^{\frac{2cd}{3}}],
\end{aligned}$$

where we've used Lemma 29 in the second inclusion. When $c < 1.5$, we have $\frac{2c}{3} < 1$. Thus, we can repeat this procedure until we derive the inclusion

$$\begin{aligned}
\mathsf{BPTS}[n^d] &\subseteq (\exists n^1)\mathsf{BPTS}[n^{\frac{d}{c}}] \\
&\subseteq \mathsf{BPTS}[n^{d'}]
\end{aligned}$$

for $d' < d$. To get a contradiction, we can apply the ideas of Lemma 34 modulo some additional technical details that we relegate to the full version. ◀

References

1 Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1), February 2009. doi:10.1145/1490270.1490272.

2 Sanjeev Arora and Boaz Barak. *Computational complexity: A modern approach*. Cambridge University Press, Cambridge, 2009. doi:10.1017/CBO9780511804090.

3 Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $\mathcal{P} =?\mathcal{NP}$ question. *SIAM J. Comput.*, 4(4):431–442, 1975. doi:10.1137/0204037.

4 Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *STOC'13—Proceedings of the 2013 ACM Symposium on Theory of Computing*, pages 111–120. ACM, New York, 2013. doi:10.1145/2488608.2488623.

5 Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998.

6 Sergey Bravyi, David Gosset, and Robert König. Quantum advantage with shallow circuits. *Science*, 362(6412):308–311, 2018.

7 Samuel R Buss and Ryan Williams. Limits on alternation trading proofs for time–space lower bounds. *computational complexity*, 24(3):533–600, 2015.

8 Scott Diehl. Lower bounds for swapping arthur and merlin. In *APPROX-RANDOM*, 2007.

9 Scott Diehl. Lower bounds for swapping arthur and merlin. In Moses Charikar, Klaus Jansen, Omer Reingold, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 449–463, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

10 Scott Diehl and Dieter van Melkebeek. Time-space lower bounds for the polynomial-time hierarchy on randomized machines. *SIAM J. Comput.*, 36(3):563–594, 2006. doi:10.1137/050642228.

11 Lance Fortnow, Richard J. Lipton, Dieter van Melkebeek, and Anastasios Viglas. Time-space lower bounds for satisfiability. *J. ACM*, 52(6):835–865, 2005. doi:10.1145/1101821.1101822.

12 Lov K Grover. A fast quantum mechanical algorithm for database search. *arXiv preprint*, 1996. arXiv:quant-ph/9605043.

13 Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *STOC'19—Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1115–1124. ACM, New York, 2019. doi:10.1145/3313276.3316411.

14 Yael Tauman Kalai, Omer Paneth, and Lisa Yang. Delegation with updatable unambiguous proofs and ppad-hardness. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 652–673, Cham, 2020. Springer International Publishing.

15 Ravindran Kannan. Towards separating nondeterminism from determinism. *Math. Systems Theory*, 17(1):29–45, 1984. doi:10.1007/BF01744432.

16 Clemens Lautemann. BPP and the polynomial hierarchy. *Inform. Process. Lett.*, 17(4):215–217, 1983. doi:10.1016/0020-0190(83)90044-3.

17 Dylan M. McKay and Richard Ryan Williams. Quadratic time-space lower bounds for computing natural functions with a random oracle. In *10th Innovations in Theoretical Computer Science*, volume 124 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 56, 20. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2019.

18 V. Nepomnjascii. Rudimentary predicates and turing calculations. *Doklady Mathematics*, 11:1462–1465, 1970.

19 Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information.* Cambridge University Press, Cambridge, 2000.

20 Noam Nisan. RL $\subseteq$ SC. *Comput. Complexity*, 4(1):1–11, 1994. doi:10.1007/BF01205052.

21 Ran Raz and Avishay Tal. Oracle separation of BQP and PH. In *STOC'19—Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 13–23. ACM, New York, 2019. doi:10.1145/3313276.3316315.

22 Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997. doi:10.1006/jcss.1997.1494.

23 Dieter van Melkebeek and Thomas Watson. Time-space efficient simulations of quantum computations. *Theory Comput.*, 8:1–51, 2012. doi:10.4086/toc.2012.v008a001.

24 Adam Bene Watts, Robin Kothari, Luke Schaeffer, and Avishay Tal. Exponential separation between shallow quantum circuits and unbounded fan-in shallow classical circuits. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 515–526. ACM, 2019. doi:10.1145/3313276.3316404.

25 R. Ryan Williams. Time-space tradeoffs for counting NP solutions modulo integers. In *In Proceedings of the 22nd IEEE Conference on Computational Complexity*, pages 70–82. IEEE, 2007.

26 Ryan Williams. Inductive time-space lower bounds for sat and related problems. *Journal of Computational Complexity*, 15:433–470, 2006. doi:10.1007/s00037-007-0221-1.

27 Ryan Williams. Alternation-trading proofs, linear programming, and lower bounds. *ACM Trans. Comput. Theory*, 5(2):Art. 6, 49, 2013. doi:10.1145/2493246.2493249.