Quantum Adversary (Upper) Bound

Shelby Kimmel*

January 4, 2011

Abstract

We propose a method for upper bounding the general adversary bound for certain boolean functions. Due to the the tightness of query complexity and the general adversary bound [5], this gives an upper bound on the quantum query complexity of those functions. We give an example where this upper bound is smaller than the query complexity of any known quantum algorithm.

1 Introduction

The general adversary bound has proven to be a powerful concept in quantum computing. It has recently been shown to be tight with respect to the quantum query complexity of any function [5]. However, as a tool for calculating query complexities, it is somewhat limited, as the bound for even simple, few-bit functions must be calculated numerically [3, 7].

Here we suggest a new way to use the general adversary bound. Since it is tight with respect to quantum query complexity, any upper bound on the general adversary bound is also an upper bound on quantum query complexity. Finding a comprehensive method of upper bounding the general adversary bound is likely very difficult. However, using the composition property of the general adversary bound, we are able to upper bound the general adversary bound of a boolean function w, given an algorithm for the composed function w^d. This bound is useful in only limited cases, but we will give an example of a function where this upper bound is smaller than the query complexity of the best known quantum algorithm.

Before we can state the main theorem, we need a few definitions.

Definition 1.1. [3] *The general quantum adversary bound for a function w is*

$$ADV^{\pm}(w) = \max_{\Gamma \neq 0} \frac{\|\Gamma\|}{\|\max_i \Gamma \star D_i\|} \tag{1}$$

where Γ is a symmetric matrix with $\Gamma[x,y] = 0$ if $w(x) = w(y)$, $\|M\|$ is the spectral norm of the matrix M, D_i is a matrix whose entries are 0 or 1 such that $D_i[x,y] = 1$ if the i^{th} bits of x and y are not equal, and zero otherwise. $\Gamma \star D_i$ denotes entry-wise multiplication.

*Center for Theoretical Physics, Massachusetts Institute of Technology, Email: skimmel@mit.edu. Supported by NSF Grant No. DGE-0801525, *IGERT: Interdisciplinary Quantum Information Science and Engineering.*

A function w is boolean if $w : S \to \{0,1\}$ with $S \subseteq \{0,1\}^n$. Given a function $w : S \to \{0,1\}$ with $S \subseteq \{0,1\}^n$, and a natural number d, we define w^d, "w composed d times," recursively as $w^d = w \circ (w^{d-1}, \dots, w^{d-1})$, where $w^1 = w$.

Now we can state our main theorem:

Theorem 1.2. (Main Theorem) *Suppose we have a boolean function w that is composed d times, represented by w^d. Suppose we have a quantum algorithm for w^d that takes $\tilde{Q}$ queries. Then $Q(w) = O(\tilde{Q}^{1/d})$, where $Q(w)$ is the bounded-error quantum query complexity of w.*

The proof of Theorem 1.2 appears in Section 2.

One might think that Theorem 1.2 is a bit useless because:

- One is unlikely to have an algorithm for w^d without having an algorithm for w.
- If one has an algorithm for w, then the algorithm for w^d is likely created by composing the algorithm for w, in which case one expects the query complexity of the new algorithm to be at least $\tilde{J}^d$, where $\tilde{J}$ is the query complexity of the algorithm for w.

Luckily for us, the second point is not always correct. If there is a quantum algorithm for w that uses $\tilde{J}$ queries, where $\tilde{J}$ is not optimal (i.e. is larger than the bounded error quantum query complexity of w), then the number of queries used when the algorithm is composed d times can be much less than $\tilde{J}^d$. If this is the case, and if the non optimal algorithm for w is the best known, Theorem 1.2 promises that there exists an algorithm for w that uses potentially much less than $\tilde{J}$ queries, even though no explicit algorithm yet exists. We give an example of such an algorithm in Section 3.

Examples where Theorem 1.2 proves the existence of an algorithm that uses fewer queries than the best known algorithm may not be common; nevertheless, the theorem is interesting as an inspiration for future upper bounds on query complexity obtained by upper bounding the general adversary bound.

2 Proof of Theorem 1.2

We need two lemmas to prove the Main Theroem:

Lemma 2.1. (Høyer, Lee, and Špalek [3]) *For any boolean function $w : S \to \{0,1\}$ with $S \subseteq \{0,1\}^n$ and natural number d,*

$$ADV^{\pm}(w^d) \geq (ADV^{\pm}(w))^d. \tag{2}$$

(This bound can be improved to equality [5], but we don't require equality for our theorem.)

Lemma 2.2. (Lee, Mittal, Reichardt, and Špalek [5]) *For any function $w : S \to E$, where $S \in D^n$, with E and D finite sets, then the bounded-error quantum query complexity of w, $Q(w)$, satisfies*

$$Q(w) = \Theta(ADV^{\pm}(w)). \tag{3}$$

Proof of Main Theorem. Given an algorithm for w^d that requires $O(\tilde{Q})$ queries, by Lemma 2.2,

$$ADV^{\pm}(w^d) = O(\tilde{Q}). \tag{4}$$

Combining Eq. 4 and Lemma. 2.1, we have

$$(ADV^{\pm}(w))^d = O(\tilde{Q}). \tag{5}$$

Raising both sides to the $1/d^{th}$ power, we obtain

$$ADV^{\pm}(w) = O(\tilde{Q}^{1/d}). \tag{6}$$

At this point, we have the critical upper bound on the general adversary bound of w. Finally, using Lemma 2.2 again, we have

$$Q(w) = O(\tilde{Q}^{1/d}). \tag{7}$$

□

3 Example for Which Theorem 1.2 is Useful

In this section we will describe a function, called the 1-fault NAND tree, for which Theorem 1.2 gives a better upper bound on query complexity than any known quantum algorithm. The 1-fault NAND tree function was proposed by Zhan et al. [9] as a way of obtaining a superpolynomial speed-up from a boolean formula with a promise on the inputs. We will first define a NAND tree, and then explain the promise of the 1-fault NAND tree problem.

The NAND tree is a complete, binary tree of depth n, where each node is assigned a bit value. The leaves are assigned arbitrary values, and any internal node d is given the value NAND$(v(d_1), v(d_2))$, where d_1 and d_2 are the children of d, and $v(d_i)$ denotes the value of that node. To evaluate the NAND tree, one must find the value of the root given an oracle for the values of the leaves. (The NAND tree is equivalent to solving NAND^n, although the composition we will be considering is not the composition of the NAND function, but of the NAND tree as a whole.) For arbitrary inputs, Farhi et al. showed that there exists an optimal algorithm to solve the NAND tree in $2^{n/2}$ queries [1]. Classically, the best algorithm requires $2^{.753}$ queries [8]. However, we will consider the 1-fault NAND tree, for which there is a promise on the form of the inputs.

Definition 3.1. (1-fault NAND Tree [9]) *Consider a NAND tree of depth n, with values assigned to all nodes. Then to each node d, with child nodes d_1 and d_2, we assign an integer $\kappa(d)$ such that:*

- $\kappa(d) = 0$ *for leaf nodes.*
- $\kappa(d) = \max_i \kappa(d_i)$, *if* $v(d_1) = v(d_2)$
- *Otherwise* $v(d_1) \neq v(d_2)$. *Let* d_i *have value* 0. *Then* $\kappa(d) = 1 + \kappa(d_i)$.

A tree satisfies the 1*-fault condition if* $\kappa(d) \leq 1$ *for all nodes* d *in the tree.*

Notation: When $v(d_1) \neq v(d_2)$ we call the node d a **fault**. (Since NAND$(0,1) = 1$, fault nodes must have value 1, although not all 1-valued nodes are faults.) When $v(d_1) = v(d_2)$ we call d **trivial**.

The 1-fault condition is a limit on the amount and location of faults within the tree. A 1-fault NAND tree is such that given a path from the root to a leaf, if that path (moving in the direction from the root towards a leaf) encounters a fault node and then passes through the 0-valued child of the fault node, there can be no further fault nodes on the path. For instance, if every path from the root to a leaf contains at most one fault, then the tree is a 1-fault tree. An example of a 1-fault NAND tree is given in Figure 1.

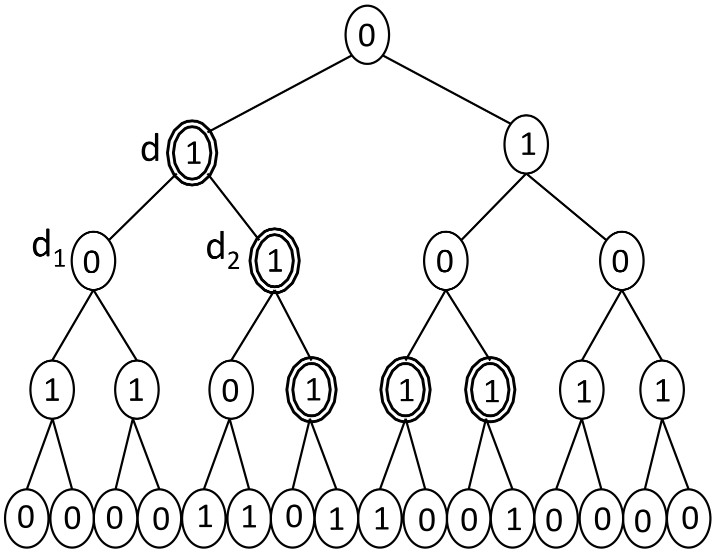


Figure 1: An example of a 1-fault NAND tree of depth 4. Leaves are the bottom row of nodes, and the root is at the top. Fault nodes are marked by a double circle. Consider the node labeled d. It is a fault since one of its children (d_1) has value 0, and one (d_2) has value 1. Notice that among d_1 and nodes descending from d_1, there are no further faults, as required by the 1-fault condition. However, at d_2 there is a fault, which is allowed as long as there are no further faults at the 0-valued child of d_2 or among any of the descendants of the 0-valued child of d_2.

Zhan et al. [9] propose a quantum algorithm for an n-level, 1-fault NAND tree that requires $O(n^2)$ queries to an oracle for the leaves. However, when the 1-fault NAND tree is composed $\log n$ times, they find an algorithm that requires $O(n^3)$ queries. (Here we see an example where the number of queries required by an algorithm composed d times does not scale exponentially in d, which is critical for applying Theorem 1.2.) By applying Theorem 1.2 to the algorithm for the 1-fault NAND tree composed $\log n$ times, we find that an upper bound on the query complexity of the 1-fault NAND tree is $O(1)$. Not only is this a large improvement over the $O(n^2)$ queries of Zhan et al., but we have not yet found an algorithm that solves for the 1-fault NAND tree in a constant number of queries in all cases. Thus we have a upper bound on the query complexity without an algorithm that achieves that query complexity. Zhan et al. prove $\Omega(\log n)$ is a lower bound on the classical query complexity of the 1-fault NAND tree.

Actually, Zhan et al. apply their methods to a broader range of "fault" trees, where instead of NAND, the evaluation tree is made up of what they call direct boolean functions. For this more general case, the 1-fault direct boolean tree composed $\log n$ times can be solved by a quantum computer in $O(n^2 w^{\log n}) = O(n^{2+\log w})$ queries, where w is independent of n and depends only on the specific direct boolean function used in the tree. Applying Theorem 1.2 to this more general case again gives an upper bound of $O(1)$ for the 1-fault direct boolean evaluation tree, while Zhan et al. prove $\Omega(\log n)$ is a lower bound on the classical query complexity of such trees.

The NAND tree is the simplest case, and so it is here that we will attempt to create an $O(1)$ query quantum algorithm. There are two regimes for which we are able to solve the 1-fault NAND tree in a constant number of queries: when there is exactly one fault on every path from the root to a leaf (quantum regime), and when there is a large majority of leaves with either value 0 or value 1 (classical regime). In the appendix we will give an algorithm which outputs the correct result with bounded error if the input is promised to fall into one of these two regimes. We note that the algorithm for the quantum regime (called the Quantum Haar Transform (QHT) algorithm) is of braoder interest than the use described here, although its usefulness in oracle problems was inspired by the knowledge that there must exist an algorithm that solves the 1-fault NAND tree in a constant number of queries; the QHT algorithm and related oracle problems are described in further detail in [4].

4 Conclusions and Future Work

In this paper, we have created a method for upper bounding quantum query complexity using the general adversary bound. Using this method, we show that the 1-fault NAND tree can always be solved in $O(1)$ queries (as well as the 1-fault direct boolean evaluation tree). While we have not yet found an algorithm which matches this bound, we have made partial progress towards an algorithm, and the knowledge that there must exist a constant-query algorithm for the 1-fault NAND tree has inspired a new algorithm and series of oracle problems [4].

We would like to find other examples where Theorem 1.2 is useful, although we suspect that 1-fault evaluation trees are a somewhat unique case. We hope to find a complete quantum algorithm for the 1-fault NAND tree which can be solved in a constant number of queries. Finally, this work suggests that new ways of upper bounding the general adversary bound could give us a second window into quantum query complexity in addition to creating algorithms.

5 Acknowledgements

Thanks to Bohua Zhan, Avinatan Hassidim, Eddie Farhi, and Andy Lutomirski for helpful discussions and for reading drafts of this paper. Thanks to Scott Aaronson for useful conversations, for teaching a great class, and for giving me the opportunity to be on his blog!!

References

[1] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum algorithm for the hamiltonian nand tree. *Theory of Computing*, 4(1):169–190, 2008.

[2] A Haar. Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, 69(3):331–371, 1910.

[3] Peter Hoyer, Troy Lee, and Robert Spalek. Negative weights make adversaries stronger. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 526–535, New York, NY, USA, 2007. ACM.

[4] S. Kimmel. Speed from repitition. *Arxiv-eprints*, 2010.

[5] T. Lee, R. Mittal, B. W. Reichardt, and R. Spalek. An adversary for algorithms. *ArXiv e-prints*, November 2010.

[6] Yves Nievergelt. *Wavelets Made Easy.* Birkhäuser, Washington, 1999.

[7] Ben W. Reichardt and Robert Spalek. Span-program-based quantum algorithm for evaluating formulas. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 103–112, New York, NY, USA, 2008. ACM.

[8] Michael Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:29–38, 1986.

[9] B Zhan, S. Kimmel, and A. Hassidim. Super-polynomial quantum speed-ups for boolean evaluation trees with hidden structure. *Arxiv-eprints*, 2010.

A Quantum Regime

We will describe a quantum algorithm for solving the 1-fault NAND tree in a constant number of queries when there is exactly one fault node in each path from the root to a leaf. We say that inputs that satisfy this condition are in the quantum regime. An example of a 1-fault-per-path tree is given in Figure 2b.

This algorithm involves making a measurement in the Haar wavelet basis [2, 6]. The Haar transform is based on the following step-like function:

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \leq t < 1/2 \\ -1 & \text{if } t \leq 1/2 < 1 \\ 0 & \text{otherwise.} \end{cases}$$

Then on the 2^n dimensional Hilbert space, with standard basis states $\{|j\rangle\}$, $0 \leq j < 2^n$, the (un-normalized) Haar basis consists of the states $\{|\phi_0\rangle, |\psi_{k,l}\rangle\}$:

$$|\phi_0\rangle = \sum_{j=0}^{2^n-1} |j\rangle$$

$$|\psi_{k,l}\rangle = \sum_{j=0}^{2^n-1} \psi(2^{-k}j - l)|j\rangle \tag{8}$$

where $1 \le k \le n$, and $0 \le l < 2^{n-k}$.

Now we suppose that the input to the 1-fault NAND tree is given to us in the form of a phase-flip oracle O such that $O|j\rangle = (-1)^{f(j)}|j\rangle$ where $f(j)$ is the value of the j^{th} leaf of the tree. Now we can state the algorithm, called the Quantum Haar Transform (QHT) algorithm:

(1) Starting with $|0\rangle$, apply a Hadamard to all bits, giving $|\xi\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle$

(2) Apply the phase flip oracle, giving $|\xi_f\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} (-1)^{f(j)}|j\rangle$

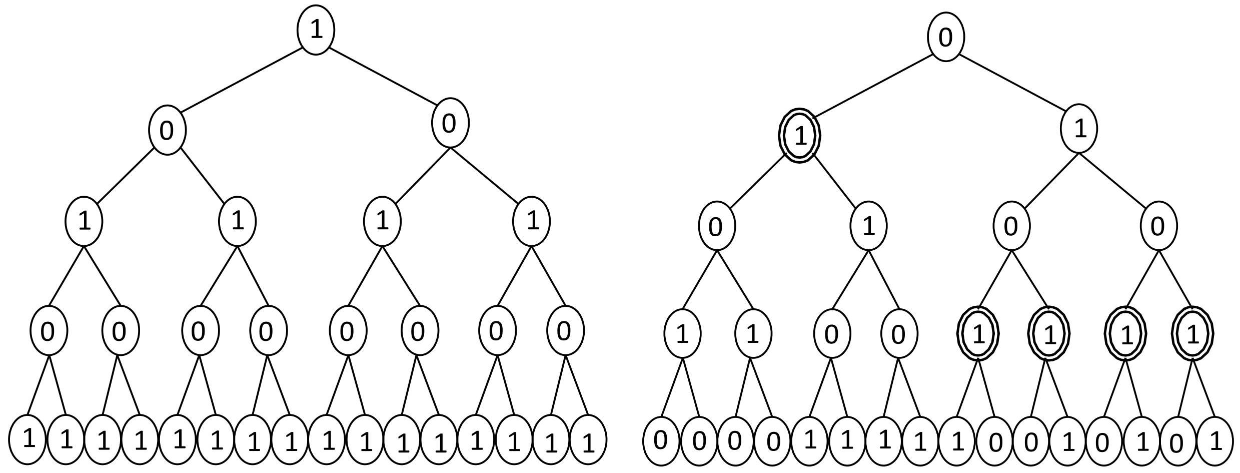
(3) Measure $|\xi_f\rangle$ in the Haar basis.

For an in-depth discussion of this algorithm, see [4].

To see why this algorithm works, we need to consider the values of the leaves of a NAND tree with one fault node per path, as shown in Figure 2b. Notice that when there are no faults in a NAND tree, as in Figure 2a, then if the root has value 1, then all even depth nodes have value 1, and all odd depth nodes have value 0. Likewise, if the root has value 1, then all even depth nodes have value 1, and all odd depth nodes have value 0. Further notice that faults can only occur at nodes with value 1 (since $\text{NAND}(0,1) = 1$). Therefore, on any path with only one fault, that fault must occur at even depth if the root of the tree has value 1 or at odd depth if the root has value 0. This means that determining the depth of a fault in such a tree gives the value of the root. For example, in Figure 2b, all faults occur at odd depth (in particular at depth 1 and depth 3), since the root is 0-valued.

Now consider the leaves descending from a fault node d when there are no further faults at any nodes descending from d. If the fault is at height i, then it will have 2^i leaves descending from it. Because one of the children of the fault node has value 0, and one has value 1, the 2^{i-1} leaves descending from one child will all have the same value, r, and the next 2^{i-1} leaves descending from the other child will have the opposite value $\neg r$. For example, in Figure 2b, the fault at height $i = 3$ has $2^i = 2^3 = 8$ leaves descending from it. The first $2^{i-1} = 2^2 = 4$ of these leaves have value 0, while the second four of these leaves have value 1.

Consider the state $|\xi_f\rangle$, which is a superposition of all standard basis states, where each standard basis state corresponds to a leaf in the tree, and the phase of the state corresponds to the value of the leaf. For a moment only consider the subset of states in $|\xi_f\rangle$ corresponding to the leaves descending from a fault node d at height i. In this subset, we have a superposition



(a) NAND tree with no faults (b) NAND tree with one fault per path

Figure 2: Two examples of 1-fault NAND trees with depth 4. Figure (a) shows a tree with no faults, and Figure (b) shows a 1-fault-per-path tree. In Figure (a), note that at each depth, all nodes have the same value, and that value depends on the value of the root and whether the node is at odd or even depth. In Figure (b), since the root is 0, all faults occur at odd depth. Also note the leaves descending from each fault; the first half of the leaves descending from a fault node have the same value, which is different from the value of the next half of the leaves descending from the node.

where the first 2^{i-1} states in the superposition have phase $(-1)^r$ and the next 2^{i-1} states have phase $(-1)^{\neg r}$. But this is exactly (up to a phase) a Haar basis state. In particular, if d is the m^{th} node (from left to right) at height i, it is the state $\pm|\psi_{i,m-1}\rangle$. Thus we can rewrite the state $|\xi_f\rangle$ in terms of the Haar basis as

$$|\xi_f\rangle = \frac{1}{\sqrt{2^n}} \sum_b (-1)^{r_b} |\psi_{i_b,m_b-1}\rangle, \tag{9}$$

where b is a fault node with height i_b, with position m_b (when counting from left to right among all nodes at height i_b), and with r_b the value of the first 2^{i_b-1} leaves descending from b.

By the orthogonality of the Haar basis states, in running the above algorithm, one will always obtain an outcome $|\psi_{i_b,m_b-1}\rangle$, where i_b is the height of one of the faults in the tree. But since faults must occur at even depth if the root of the tree is one, or at odd depth if the root of the tree is 0, computing PARITY$(n - i_b)$ gives the value of the root of the tree (after a single quantum query).

B Classical Regime

In this section we will describe inputs to the 1-fault NAND tree such that the root can be evaluated with high probability by classically querying a constant number of leaves. We call these inputs the classical regime. The 1-fault NAND tree reduces to the majority function when there are not an equal number of leaves with value 0 and value 1 (see Lemma B.1 below). Our classical regime will thus be inputs where more than three quarters of the leaves have the same value. In this case, random classical querying of the leaves will with high probability indicate which value has the majority, and hence give the value of the tree.[1]

There is a clear separation between the quantum and classical regimes, since in the quantum regime, there are an equal number of leaves with value 0 and value 1. Given the promise that the input is either in the classical or quantum regime, one can distinguish which regime the input is in, with high probability, by running the quantum QHT algorithm several times. This works because the probability of obtaining outcome $|\phi_0\rangle$ is related to the extent that one value has the majority among the leaves. If one value has a large majority, then there is a high probability of obtaining outcome $|\phi_0\rangle$.[2] Once it is clear which regime the input falls into, either the results of the QHT algorithm can be read off, if it is in the quantum regime, or the classical algorithm (random querying of leaves) can be run.

First we will prove several lemmas necessary for the algorithm, and then we will describe the algorithm itself.

Lemma B.1. *If the majority of the leaves of a* 1*-fault NAND tree have value* g*, then an even depth, 1-fault NAND tree has value* g*, and an odd depth tree has value* $\neg g$*.*

Proof. We will only prove the case where the tree has a majority of leaves with value 1, but the case for a majority of leaves with value 0 is analagous. Our proof follows by induction. Note that we will treat odd and even depth trees separately. For the odd-depth base case, a depth-1 tree with more than 1/2 of the leaves with value 1 has leaves $(1, 1)$ and has root 0. For the even-depth base case, a depth-2 tree with the majority of leaves having value 1 has leaves that are either $(1, 1, 1, 1)$ or $(1, 1, 1, 0)$ (and permutations). In any case, the root has value 1.

We assume for induction that all trees of depth $2k$, $(k \geq 1)$, with a majority of leaves with value 1, have root value 1. Now consider a depth $(2k + 1)$ tree, T, with more than 1/2 of the leaves having value 1. Suppose for contradiction that T's root has value 1.

Notice each of the two child nodes of the root of T is the root of a depth $2k$ tree. We call these two trees subtrees. Suppose one of the subtrees has $r_1 \cdot 2^{2k}$ leaves with value 1 and the other subtree has $r_2 \cdot 2^{2k}$ leaves with value 1. We know $r_1 + r_2 > 1$ because the majority of T's leaves have value 1. Thus at least one of the subtrees of depth $2k$ has more than 1/2 of its leaves having value 1, and by our inductive assumption it therefore has a root with value 1. Because the value of the root of T is the NAND of the values of its children, and one of the children has value 1 and the root has value 1, the other child must have value 0.

Therefore the root of T is a fault node (since one of its children is 0-valued, and one is 1-valued). Thus in the subtree rooted at the 0-valued child, there can be no faults, by definition of the 1-fault NAND tree. But if there are no faults in a depth $2k$ tree with

[1]The classical regime can be extended to the case where a constant proportion of the leaves have a majority, instead of 3/4. We choose 3/4 to have a specific number to work with. We also suspect it is at the edge of the region where a quantum algorithm would become useful.

[2]One could also classically randomly query a constant number of leaves to see which regime the oracle is in, but using the quantum algorithm to differentiate hints at a way to bridge the gap between the quantum and classical regimes, which one must do to have a complete algorithm for the 1-fault NAND tree.

root 0, all leaves in the tree must be 0. Since the subtree rooted at the 0-valued child only has 0-valued leaves, even if the other subtree had only 1-valued leaves, there is no way for there to be a majority of 1-valued leaves in T, contradicting our assumption. Thus a tree of depth-$(2k+1)$ with a majority of 1-valued leaves must have root with value 0.

Now we use the fact that all depth $(2k+1)$ trees with a majority of 1-valued leaves have root value 0 to prove the result for $(2k+2) = 2(k+1)$ level trees. Suppose we have a depth $(2k+2)$ tree with a majority of leaves having value 1. For contradiction, we assume that this tree has root value 0. Again we can look at the two subtrees rooted at depth 1. Because of the form of the NAND function, both of the subtrees must have roots with value 1. But by the same argument as in the $(2k+1)$ case, we know that at least one of the subtrees has more than 1/2 of its leaves with value 1. But from the previous paragraph, we know such a subtree must have value 0 at its root, leading to a contradiction. □

Lemma B.2. *The probability of obtaining outcome $|\phi_0\rangle$ is larger than* 1/4 *when the QHT algorithm is run, if and only if at least* 3/4 *of the leaves have the same value.*

Proof. Using a normalized $|\phi_0\rangle$, $\langle\phi_0|\xi_f\rangle$ is the number of leaves that have value 0 minus the number of leaves that have value 1, divided by 2^n. There are 2^n leaves, and leting p be the fraction of the leaves have value 0, $\langle\phi_0|\xi_f\rangle = (1-2p)$. The probability of measuring $|\phi_0\rangle$ is $|\langle\phi_0|\xi_f\rangle|^2$, and this is larger than 1/4 if and only if $p \leq 1/4$ or $p \geq 3/4$. □

Lemma B.3. *The probability of obtaining outcome $|\phi_0\rangle$ is* 0 *when the QHT algorithm is run if there is exactly* 1 *fault on each path from the root to a leaf (i.e. in the quantum regime).*

Proof. From Section A, if there is a fault with no further faults at nodes descending from it, then the first half of the leaves descending from the fault have value r, and the second half have value $\neg r$. Since every path has just one fault, the number of leaves with value 0 and value 1 are eqaul. Thus the probability of measuring $|\xi_f\rangle$ is $|\langle\phi_0|\xi_f\rangle|^2 = 0$, since $\langle\phi_0|\xi_f\rangle$ is proportional to the number of leaves with value 0 minus the number of leaves with value 1. □

Given the promise that the input is either in the classical regime or the quantum regime, the following algorithm will evaluate a 1-fault NAND tree correctly with probability at least 2/3:

1. Run the QHT algorithm 8 times
2. If $|\phi_0\rangle$ is never obtained, and every outcome state $|\psi_{k,l}\rangle$ has k's with the same parity p, then using Lemmas B.3 and B.2 and the binomial distribution, with probability $> 2/3$, the input is in the quantum regime. Output $PARITY(n-p)$, for reasons described in Section A.
3. If outcome $|\phi_0\rangle$ is measured or if the parity of the states don't match, we know with certainty that we are in the classical regime. Randomly query 5 leaves. If value r has the majority, and if the tree has even depth, output r, while if the tree has odd depth, output $\neg r$. Since the probability of choosing a leaf with value r is at least .75 at each query due to the promise on the oracle, by the binomial distribution, the probability that at least 3 of the leaves queried have value r is $> 2/3$.

We thus have a bounded error quantum algorithm using a constant number of queries for the 1-fault NAND tree with the promise that the tree is in the classical regime or the quantum regime.