Quantum Min/Max Searching

Lecture 3

Mingyu Lee
Seoul National University

red1108@snu.ac.kr
+82 010-2857-3320 (only text) / +01 530-760-8690

2026.04.24(FRI) 16:30 KST

Welcome back

Week 3. Thanks for coming back!

Last two weeks we studied about

- Quantum Amplitude Amplification
- Quantum Amplitude Estimation
- Quantum Counting

This week we will see more practical applications of these primitives.

- Exponential searching (BBHT): Grover when t is unknown
- Dürr–Høyer minimum-finding algorithm, $O(\sqrt{N})$
- Extension: find k smallest items in $O(\sqrt{kN})$

All slides remain on GitHub: `https://github.com/red1108/lectures`

Outline

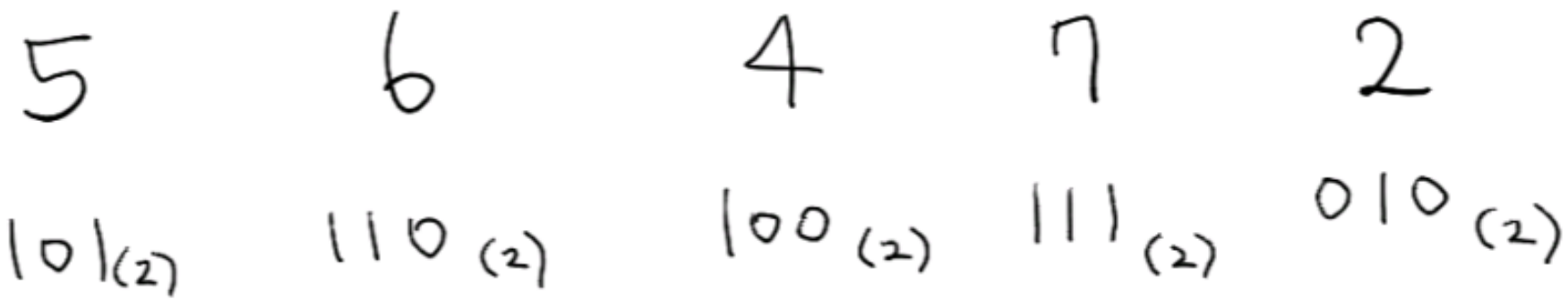
Outline

Grover Recap

- Given $f : \{0,1\}^n \to \{0,1\}$ and the promise that at least one x satisfies $f(x) = 1$
- Grover finds such x in $O(\sqrt{N})$ queries, where $N = 2^n$
- Classically we need $\Theta(N)$, and BBBV shows $\sqrt{N}$ is optimal

Now, instead of "marked or not", let's learn an algorithm that finds the min/max!

Finding the Minimum



From Search to Optimization

- Problem: given values $T[0], \ldots, T[N-1]$, find

$$y^* = \arg \min_{0 \le y \le N-1} T[y]$$

- Classically: $\Theta(N)$ comparisons are necessary and sufficient
- Quantum question: can we do it in $O(\sqrt{N})$?

Yes. Dürr and Høyer (1996) gave a surprisingly short algorithm [DH96].

The paper is only **2 pages long**.

Today's Algorithm

A quantum algorithm for finding the minimum*

Christoph Dürr† Peter Høyer‡

July 1996

1 Introduction

Let $T[0..N-1]$ be an unsorted table of N items, each holding a value from an ordered set. For simplicity, assume that all values are distinct. The minimum searching problem is to find the index y such that $T[y]$ is minimum. This clearly requires a linear number of probes on a classical probabilistic Turing machine.

Here, we give a simple quantum algorithm which solves the problem using $\mathcal{O}(\sqrt{N})$ probes. The main subroutine is the quantum exponential searching algorithm of [2], which itself is a generalization of Grover's recent quantum searching algorithm [3]. Due to a general lower bound of [1], this is within a constant factor of the optimum.

2 The algorithm

Our algorithm calls the quantum exponential search-

We first give the minimum searching algorithm, then the proof of the probability of success.

Quantum Minimum Searching Algorithm

1. Choose threshold index $0 \le y \le N-1$ uniformly at random.
2. Repeat the following and interrupt it when the total running time is more than $22.5\sqrt{N} + 1.4 \lg^2 N$.[1] Then go to stage 2(2c).
   (a) Initialize the memory as $\sum_j \frac{1}{\sqrt{N}}|j\rangle|y\rangle$. Mark every item j for which $T[j] < T[y]$.
   (b) Apply the quantum exponential searching algorithm of [2].
   (c) Observe the first register: let y' be the outcome. If $T[y'] < T[y]$, then set threshold index y to y'.

This is what we will learn today.

Why Min Finding Matters

- It is one of the simplest non-trivial quantum building blocks
- Appears as a subroutine in many algorithms:
  - k-nearest neighbor
  - clustering, classification
  - shortest path, MST on graphs

It is a simple application of Grover.

Outline

Why Use Exponential Searching?

Recall how Grover works.

- To find a marked item with high probability, we apply near $\left\lfloor \frac{\pi}{4}\sqrt{N/t} \right\rfloor$ Grover iterations, where t is the number of marked items.
- Grover assumes we **know t in advance**.

So we would first have to estimate t with quantum counting (runs in $O(\sqrt{N})$, lecture 2) and then feed it into Grover.

Two Ways to Handle Unknown t

There are two ways to run Grover without knowing t.

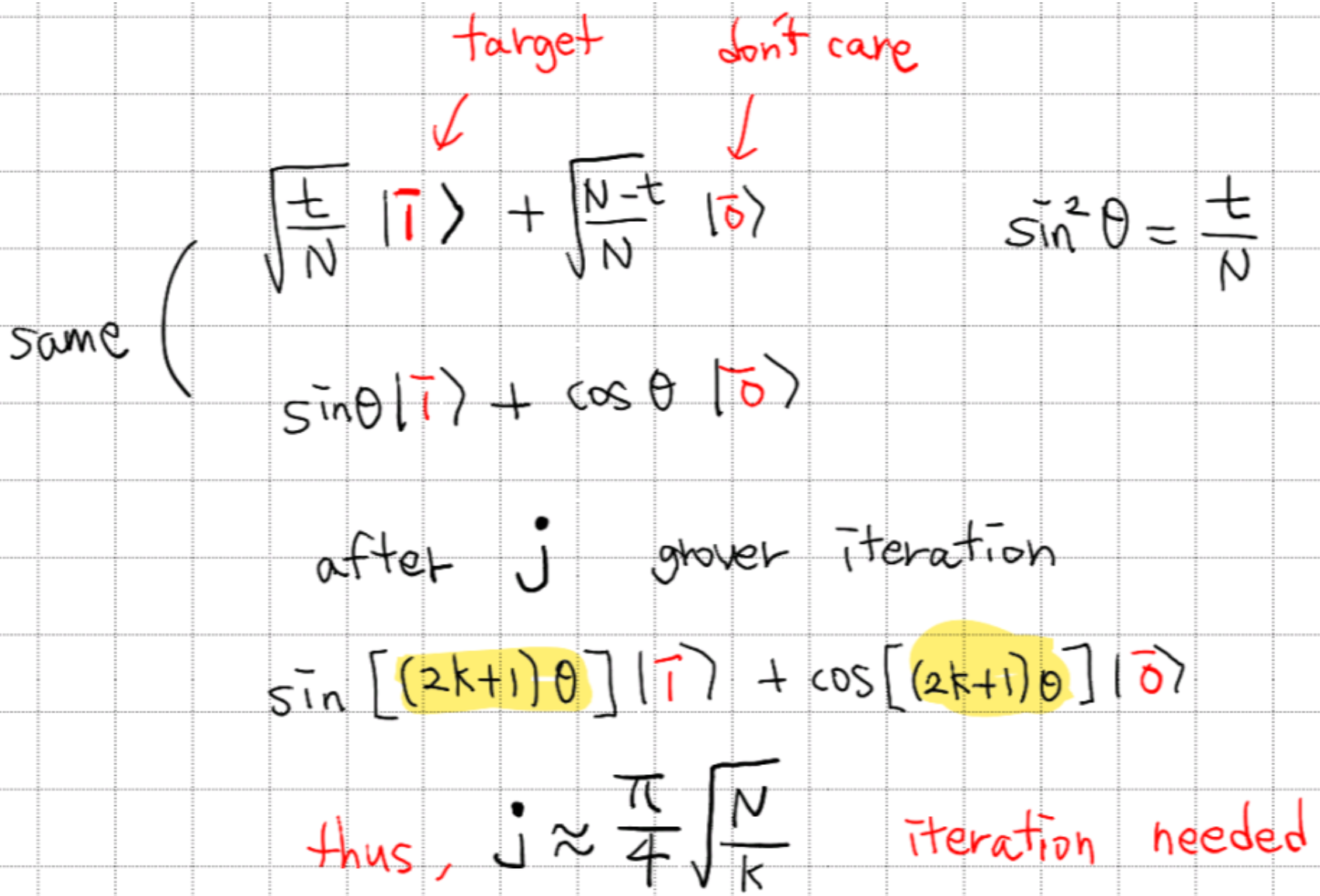
- **Counting + Grover**: first guess t with quantum counting, then run Grover.
- **Exponential searching**: works without knowing t.

Both take the same time, $O(\sqrt{N/t})$ per call.

Exponential searching is simpler. That is why most people use it as the subroutine.

Quiz time!

Grover when t Is Known



The Catch: t Unknown

- If we just use $m \approx \frac{\pi}{4}\sqrt{N}$ (good for $t = 1$), the success probability is **not** monotone in j
- For $N = 2^{20}$ and $t = 1$, $m = 804$ iterations succeed almost certainly
- For $N = 2^{20}$ and $t = 4$, $m = 804$ iterations succeed with probability $< 10^{-6}$

The wrong t drastically affects the success probability.

The Averaging Trick

Idea: don't pick a single j, randomize over a window.

Lemma 1 (BBHT, Lemma 2)

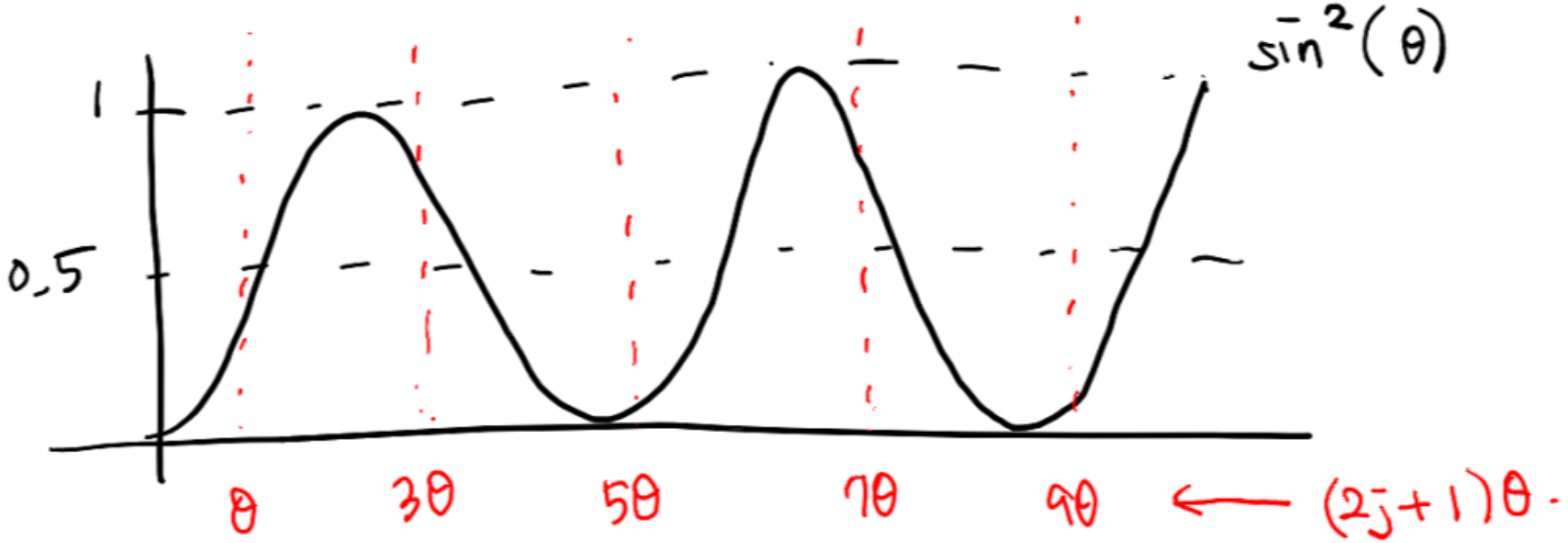
Let $\sin^2\theta = t/N$ *(unknown) and pick* j *uniform in* $\{0, \ldots, m-1\}$. *Apply* j *Grover iterations and measure. Then*

$$P_m = \frac{1}{2} - \frac{\sin(4m\theta)}{4m\sin(2\theta)}.$$

In particular, if $\mathbf{m \geq 1/\sin(2\theta)}$**, then** $\mathbf{P_m \geq 1/4}$**.**

Proof: average $\sin^2((2j+1)\theta)$ over j via the cosine-sum identity.

Lemma 1: Visual



If we sample j from $[m]$ (m is sufficiently large)

success prob ~ 0.5.

if $m \geq \frac{1}{\sin 2\theta}$, we get success prob $\geq \frac{1}{4}$

The BBHT Exponential Searching Algorithm

Inputs: oracle f, no knowledge of t. Choose any $\lambda \in (1, 4/3)$.

1. Set $m \leftarrow 1$
2. Pick j uniformly at random in $\{0, \ldots, m-1\}$
3. Apply j Grover iterations to the equal superposition; measure i
4. If $f(i) = 1$, return i
5. Else set $m \leftarrow \min(\lceil \lambda m \rceil, \sqrt{N})$ and go to step 2

The window m grows by a factor of λ until it covers the critical scale $m_0 = 1/\sin(2\theta) \approx \frac{1}{2}\sqrt{N/t}$, where every round succeeds with probability $\geq 1/4$.

BBHT Runtime

Theorem 2 (BBHT, Theorem 3)

For any $1 \leq t \leq 3N/4$, the algorithm returns a marked item in expected

$$O\left(\sqrt{N/t}\right)$$

Grover iterations, even though t is unknown.

Let $m_0 = 1/\sin(2\theta)$. Each round in round s uses $m = \lceil \lambda^{s-1} \rceil$ as its window and costs $\frac{m}{2}$ Grover iterations on average.

- **Phase 1 (ramp-up)**: m grows by factor λ until $m \geq m_0$. Geometric sum gives total cost $\leq \frac{1}{2} \cdot \frac{\lambda}{\lambda - 1} m_0$
- **Phase 2 (critical stage)**: each round succeeds with prob. $\geq 1/4$. The dominant (last) round has cost $\leq \frac{\lambda}{8 - 6\lambda} m_0$

Plugging in λ

Adding the two phases,

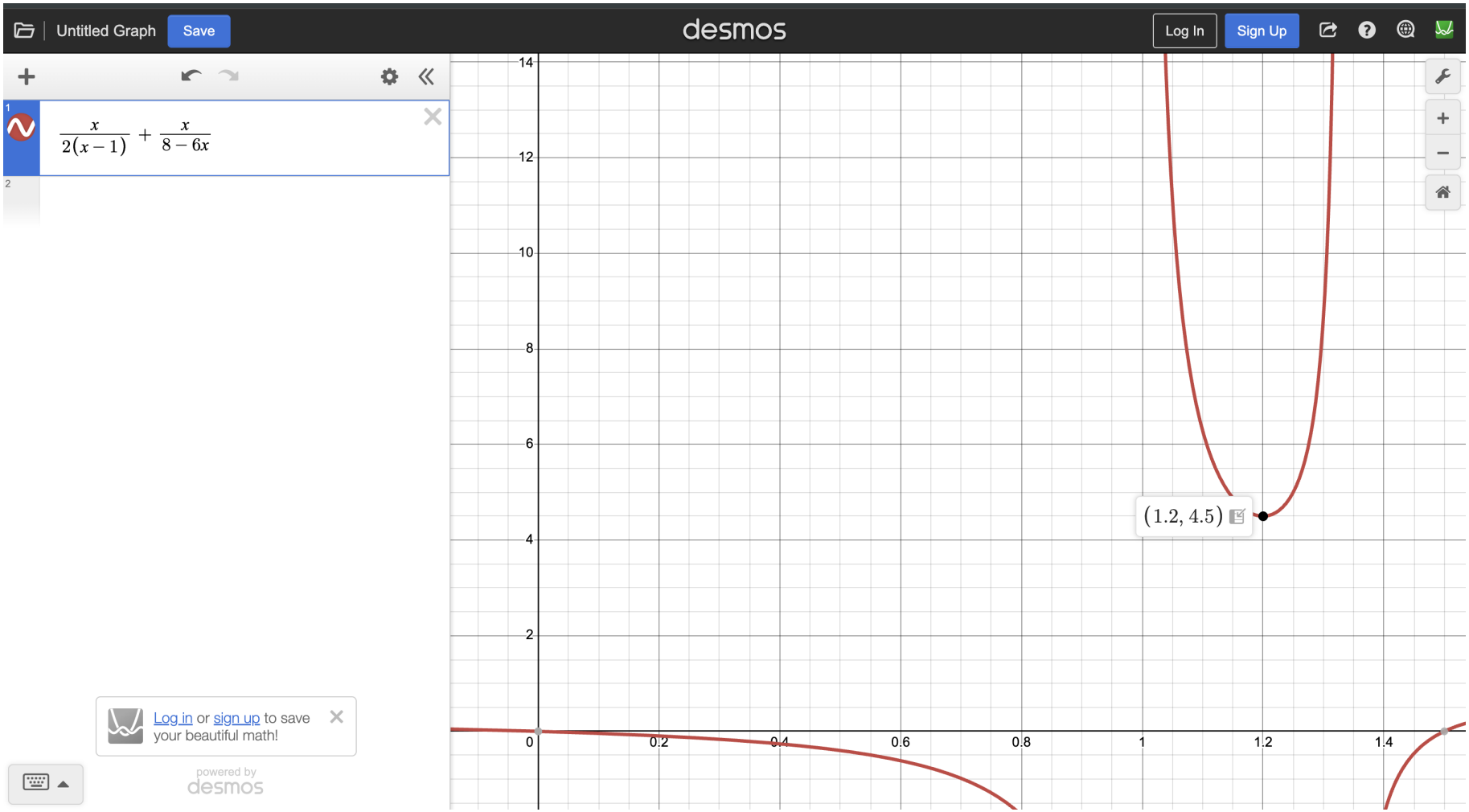
$$(\text{expected iterations}) \;\le\; C(\lambda)\, m_0, \qquad C(\lambda) \;=\; \frac{1}{2} \cdot \frac{\lambda}{\lambda - 1} \;+\; \frac{\lambda}{8 - 6\lambda}.$$

Now, substitute $C(6/5) = \frac{9}{2}$.

Using $m_0 \le \sqrt{N/t}$ (valid for $1 \le t \le 3N/4$):

The algorithm uses at most $\frac{9}{2} m_0 \le \frac{9}{2}\sqrt{N/t}$ Grover iterations in expectation, with no prior knowledge of t.

$C(\lambda)$ over $(1, 4/3)$



We need at most $\frac{9}{2} m_0 \leq \frac{9}{2}\sqrt{N/t}$ Grover iterations.

What About $t = 0$?

- If no item is marked, this algorithm never halts.
- The ramp-up gets stuck at $m = \sqrt{N}$.
- At $m = \sqrt{N}$, *if* there were even one marked item ($t \geq 1$), it should find it with probability $\geq 1/4$.
- We can just set the **time-out** as r.
- The false-negative probability is at most $(3/4)^r$.

Black-Box Interface

For the rest of the lecture, treat exponential searching as a primitive:

ExpSearch(f)

Input: oracle f on $\{0, \ldots, N-1\}$ with $t \geq 1$ marked items.

Output: a uniformly random marked index.

Cost: $\frac{9}{2}\sqrt{N/t}$ expected Grover iterations.

Two facts we will use:

- output is uniform over marked items
- each call is independent; cost depends only on the current t

Outline

What We Will Cover

Idea: keep picking any element smaller than the current one. Eventually you land on the minimum.

Each "pick a smaller one" step is one call to ExpSearch.

Total cost: $O(\sqrt{N})$ **Grover iterations.**

Covered on the Whiteboard

This section is covered by handwriting.

Outline

The k-Minima Problem

Definition 3 (k-minima)

Given $T[0..N-1]$ and $k \leq N$, return the indices of the k smallest values.

- Naive classical(heap): scan all N items, keep the k smallest $\Rightarrow O(N \log k)$
- Advanced classical(median of medians, optimal): Worst case $O(N)$.
- Naive quantum: run Dürr–Høyer k times $\Rightarrow O(k\sqrt{N})$

Can we do **better than** $k\sqrt{N}$?

Yes: $O(\sqrt{kN})$ is achievable and optimal.

The k-Minima Problem

SIAM J. COMPUT.
Vol. 35, No. 6, pp. 1310–1328

QUANTUM QUERY COMPLEXITY OF SOME GRAPH PROBLEMS*

CHRISTOPH DÜRR†, MARK HEILIGMAN‡, PETER HØYER§, AND MEHDI MHALLA¶

Abstract. Quantum algorithms for graph problems are considered, both in the adjacency matrix model and in an adjacency list-like array model. We give almost tight lower and upper bounds for the bounded error quantum query complexity of CONNECTIVITY, STRONG CONNECTIVITY, MINIMUM SPANNING TREE, and SINGLE SOURCE SHORTEST PATHS. For example, we show that the query complexity of MINIMUM SPANNING TREE is in $\Theta(n^{3/2})$ in the matrix model and in $\Theta(\sqrt{nm})$ in the array model, while the complexity of CONNECTIVITY is also in $\Theta(n^{3/2})$ in the matrix model but in $\Theta(n)$ in the array model. The upper bounds utilize search procedures for finding minima of functions under various conditions.

Key words. graph theory, quantum algorithm, lower bound, connectivity, minimum spanning tree, single source shortest paths

AMS subject classifications. 81-04, 68W20, 68R10, 05C85

DOI. 10.1137/050644719

1. Introduction. A primary goal of the theory of quantum complexity is to determine when quantum computers may offer a computational speed-up over classical computers. Today there are only a few results which give a polynomial time quantum algorithm for some problem for which no classical polynomial time solution is known. We are interested in studying the potentials for speed-up for problems for which there already are efficient classical algorithms. Basic graph problems are interesting

The first k-minima algorithm was proposed in 2006 [DHHM06], but the paper is 19 pages long... A simpler version appeared in 2019 [MIK19].

A Well-Known Trick: $\sqrt{kN}$ Scaling

Think of finding marked items one by one:

$$\underbrace{\sqrt{N/k}}_{\text{1st}} + \underbrace{\sqrt{N/(k-1)}}_{\text{2nd}} + \cdots + \underbrace{\sqrt{N}}_{k\text{-th}}$$

Bound the sum by an integral:

$$\sum_{t=1}^{k} \sqrt{\tfrac{N}{t}} \leq \sqrt{N}\left(1 + \int_{1}^{k} t^{-1/2}\, dt\right) = O(\sqrt{kN}).$$

Each find gets cheaper as k shrinks; the sum telescopes into $\sqrt{kN}$.

Step 1: Run the Finding-Minimum Algorithm

Run FM and **record every threshold update along the way**.

$$T[y_1] > T[y_2] > T[y_3] > \cdots > T[y_m]$$

- Each y_i is a threshold that FM accepted at some round
- The values are **strictly decreasing** by construction
- Total length: $m = O(\sqrt{N})$ (the FM expected runtime)

The chain is essentially a sorted ladder of candidate thresholds.

Step 2: We Don't Need an Exact Threshold

Goal: find an index i such that

$$\left|\{\, x : T[x] < T[i] \,\}\right| \;=\; k.$$

But hitting exactly k is unnecessary. It is enough to find i with

$$k \;\leq\; k' \;\leq\; c \cdot k \qquad \text{for some constant } c.$$

- This means O(k') = O(k) is good enough.
- Why? Discuss later.

Binary Search on the Threshold Ladder

Recall the FM ladder: $T[y_1] > T[y_2] > \cdots > T[y_m]$, $m = O(\sqrt{N})$.

Two endpoints to anchor the search:

- $T[y_m]$ is the **minimum** $\Rightarrow$ $rank(y_m) - 1 = 0$
- $T[y_{m-k}]$ has at least k items below it $\Rightarrow$ $rank(y_{m-k}) - 1 > k$

So a good threshold sits somewhere in $\{y_{m-k}, \ldots, y_m\}$.

- Binary search over these k candidates: $O(\log k)$ steps
- Each step calls quantum counting once: $O(\sqrt{N})$
- **Total:** $O(\sqrt{N} \log k)$ **to locate** $t_{k'}$**.**

The Process

Pseudocode

repeat:
  Run Phase 1 $\rightarrow$ candidate threshold $t_{k'}$
  Estimate $k' = h(t_{k'})$ via quantum counting
  if $k' \leq c \cdot k$: break (success)
  else: retry
Run Phase 2 with threshold $t_{k'}$

- Phase 1 alone may overshoot; quantum counting checks the size
- Each retry is independent, so a constant number suffice w.h.p.

Phase 2: Extract All k' Marked Items

We have a threshold $t_{k'}$ with exactly k' items below it.

Pull them out one by one:

- Run $\textsc{ExpSearch}(f_{t_{k'}})$ to get one marked index
- Unmark it (remove from the search space)
- Repeat until the marked set is empty

After k' rounds we have collected all k' items.

Total query complexity?

Phase 2: Cost Analysis

Each call to ExpSearch depends on the current marked count:

$$\underbrace{\sqrt{N/k'}}_{\text{round 1}} + \underbrace{\sqrt{N/(k'-1)}}_{\text{round 2}} + \cdots + \underbrace{\sqrt{N/1}}_{\text{round } k'}$$

This is exactly the $\sqrt{kN}$ trick from before:

$$\sum_{t=1}^{k'} \sqrt{\frac{N}{t}} \;=\; O(\sqrt{k'N}).$$

Since $k' = O(k)$ by Phase 1's guarantee:

$$O(\sqrt{k'N}) \;=\; O(\sqrt{kN}).$$

Trimming k' Down to k

Phase 2 returns $k' \geq k$ items. We only want the smallest k.

- Once we have the k' values in classical memory, just sort and pick the smallest k
- Worst case: $O(k' \log k')$ classical comparisons
- This is **classical** work — not counted in query complexity

The query complexity is set by Phases 1 and 2.

Final Complexity Analysis

Stage	**Cost**	**Source**
Phase 1: FM ladder	$O(\sqrt{N})$	Dürr–Høyer
Phase 1: binary search	$O(\sqrt{N}\log k)$	QC, $\log k$ steps
Phase 2: extract k'	$O(\sqrt{kN})$	$\sum_t \sqrt{N/t}$
Trim to k	free	classical

$$O(\sqrt{N}) + O(\sqrt{N}\log k) + O(\sqrt{kN}) = \boxed{O(\sqrt{kN})}$$

$\sqrt{kN}$ dominates because $\sqrt{k} \geq \log k$ for all $k \geq 1$.

Quiz time!

Outline

References I

[AK99] Ashish Ahuja and Sanjiv Kapoor.
A quantum algorithm for finding the maximum, 1999.

[BBBV97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh V. Vazirani.
Strengths and weaknesses of quantum computing.
SIAM Journal on Computing, 26(5):1510–1523, 1997.

[BBHT98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp.
Tight bounds on quantum searching.
Fortschritte der Physik, 46(4-5):493–505, 1998.

[BHMT02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp.
Quantum amplitude amplification and estimation.
In *Quantum Computation and Quantum Information*, volume 305, pages 53–74. American Mathematical Society, 2002.

References II

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp.
Quantum counting.
In *International Colloquium on Automata, Languages, and Programming*, pages 820–831. Springer, 1998.

[DH96] Christoph Dürr and Peter Høyer.
A quantum algorithm for finding the minimum, 1996.

[DHHM06] Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla.
Quantum query complexity of some graph problems.
SIAM Journal on Computing, 35(6):1310–1328, 2006.

[MIK19] Kohei Miyamoto, Masakazu Iwamura, and Koichi Kise.
A quantum algorithm for finding k-minima, 2019.