Quantum minimum finding (Dürr–Høyer)

Turn minimum finding into a sequence of coherent searches for a better threshold.

Undergraduate 21 min lesson Dr. Charith Jayasekara

By the end, you'll be able to

- Trace threshold updates in a four-item list.
- Distinguish unknown marked-count search from a fixed Grover schedule.
- Account for coherent value access and verification.

Search for a better candidate, then lower the threshold

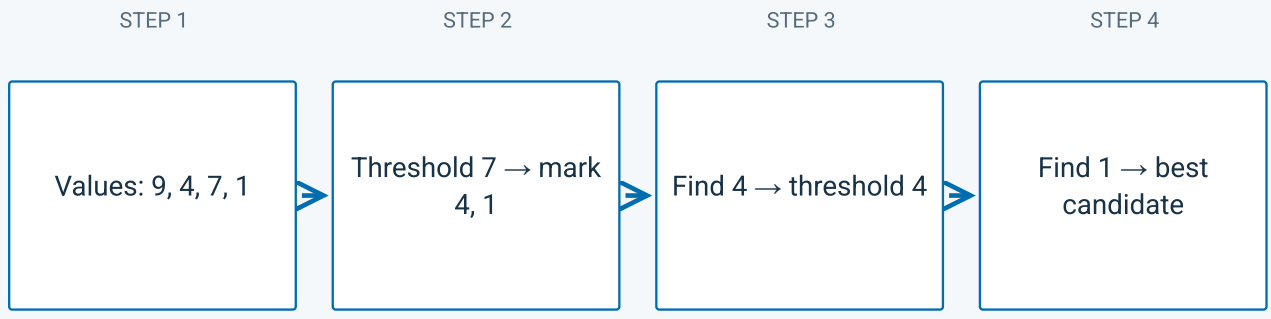
Grover search asks for an item satisfying a yes-or-no predicate. Minimum finding turns an ordered-value problem into a sequence of such predicates. Keep a current candidate, mark every index whose value is smaller, search for a marked index and replace the candidate when one is found. Repeating this idea carefully yields a quadratic query improvement for an unstructured table.

The method does not sort the whole table. Its requested output is one index attaining the minimum. That smaller output requirement matters: printing an entire sorted list would be a different task with an output cost. The algorithm also needs coherent access to values or comparisons, which an ordinary database does not automatically provide.

Review Grover interference, oracles and complexity. A useful classical prerequisite is understanding a running minimum: scan entries while retaining the smallest seen so far.

WORKED DIAGRAM · AA02

Lowering a running threshold



For values (9,4,7,1), threshold 7 marks indices 1 and 3. After choosing value 4, only index 3 is better. A valid stopping rule still needs a confidence analysis.

Specify inputs, outputs and ties

The input is a table of N ordered values, supplied through an oracle that can be evaluated coherently. A comparison with a classical threshold defines the marked set. The output is an index of a minimum, with a declared success probability. Count value-oracle or comparison-oracle calls, and separately count reversible arithmetic, workspace and data-access costs.

For equal values, decide whether any minimum index is acceptable or whether a deterministic tie-breaking rule is required. Comparing pairs consisting of value and index gives a total order if needed. The pedagogical version below uses distinct values so that ranks are unambiguous. The mathematical query analysis concerns an unstructured access model; exploit useful structure if the application supplies it.

Work a four-item example

Let the values at indices zero through three be 9, 4, 7 and 1. Suppose the initial random candidate is index two, with value seven. The predicate “value is less than seven” marks indices one and three. A quantum search subroutine may return either. If it returns index one, update the threshold to four; the next predicate marks only index three. Finding it sets the threshold to one.

If the first search instead returns index three, the minimum is already the current candidate. The algorithm cannot simply infer that fact from failing to see a better value once. A search with an unknown number of marked items needs an appropriate schedule and stopping rule. No marked item, one marked item and many marked items behave differently under a fixed Grover iteration count.

The example shows why the threshold improves while the algorithm remains randomised. It may visit values seven, four, one, or move directly from seven to one. It need not inspect the value nine. Each accepted update strictly improves the candidate, but a missing improvement does not by itself certify optimality.

The threshold changes which indices are marked; it does not change the stored list.

$$(v_0,v_1,v_2,v_3)=(9,4,7,1),\quad f_t(j)=[v_j<t],\qquad \{j:f_7(j)=1\}=\{1,3\},\quad \{j:f_4(j)=1\}=\{3\}.$$

The bracket is one when its comparison is true and zero otherwise. The current value t becomes a stricter threshold after a better candidate is found. A failed search observation alone does not prove that no better index exists.

Why the total query scale improves

When many smaller items exist, finding any one is relatively cheap. As the threshold approaches the minimum, fewer items remain marked, so each improvement becomes more expensive. The analysis combines these stages rather than charging a full square-root-N search for every possible update. The resulting bounded-success query scale is order square root of N; independent repetition can reduce failure probability.

Dürr and Høyer’s minimum-finding result formalises this strategy using a search subroutine that handles an unknown number of solutions. Its guarantee concerns oracle access to an unstructured table, with a controlled time budget and success amplification. It should not be paraphrased as certainty after one short run. Read the original minimum-finding paper.

For a classical unstructured table, establishing a minimum requires linear query scale in the worst case because an unexamined entry could be smaller. A sorted table, a convex objective or a special data structure changes the baseline. Minimum finding does not claim a quadratic improvement over every specialised classical optimisation method.

Turn the idea into an implementable subroutine

Prepare a uniform superposition of indices. Reversibly load or compute the corresponding value, compare it with the current threshold, flip a phase on better indices and uncompute the value and comparison workspace. Apply the search reflections. Measure an index, evaluate its value classically and accept it only if it improves the current candidate. A classical controller updates the threshold between searches.

Uncomputation is essential. If the value register remains correlated with the index while a diffusion intended for the index alone is applied, the simple search analysis may fail. Treat the entire prepare–mark–uncompute process as the oracle task description. The coherent comparison's precision must also match how close two values can be. Approximate real-valued objectives need a declared comparison tolerance.

Limits that affect an application

The table may be cheap to scan classically but expensive to access coherently. Building a data structure can cost order N or more before any minimum-finding query is issued. That preprocessing may be amortised across many tasks, but it cannot simply vanish from an end-to-end comparison. Noise and fault tolerance add another layer to each coherent search call.

The method returns a candidate index, not a proof certificate that every other value is larger. Its probabilistic guarantee comes from the algorithm and assumptions. In a practical workflow, preserve the randomisation schedule, comparison accuracy and confidence target. If an application requires exact certification, account for how that certificate is produced and checked.

Practice with answers

Exercise 1. If the initial candidate in the example is value nine, which indices are marked? If the candidate is value four, which are marked?

> Check the worked answer
>
> Threshold nine marks indices one, two and three. Threshold four marks only index three. The marked set depends on the current candidate, so its size changes between searches.

Exercise 2. Why not run one Grover iteration repeatedly until no improvement is observed?

> Check the worked answer
>
> A fixed iteration count can under-rotate or overshoot depending on the unknown marked fraction. A failed trial does not establish that no smaller item exists. A correct minimum-finding method needs a search schedule, overall budget and confidence analysis appropriate to an unknown number of improvements.

Check your understanding

PAUSE & THINK

With threshold 4 in the example, which index is marked?

- [ ] 0
- [ ] 2
- [ ] 3

Answer: 3. Only the value 1 at index 3 is strictly smaller than 4. A failed search trial alone does not certify that no better item exists.

Sources

- Read the original minimum-finding paper