Probability and Computation

K. Sutner

Carnegie Mellon University

1. Order Statistics

- Circuit Evaluation
- Yao's Minimax Principle
- More Randomized Algorithms *

Let $\mathcal{U}$ be some ordered universe such as the integers, rationals, strings, and so forth.

It is easy to see that for any set $A \subseteq \mathcal{U}$ of size n there is a unique order isomorphism

$$[n] \longleftrightarrow A$$

$\rightarrow$: $\mathrm{ord}(k, A)$

$\leftarrow$: $\mathrm{rk}(a, A)$

Note that $\mathrm{ord}(k, A)$ is trivial to compute if A is sorted.

Computation of $\mathrm{rk}(a, A)$ requires to determine the cardinality of $A^{\leq a} = \{ z \in A \mid z \leq a \}$ (which is easy if A is a sorted array and we have a pointer to a).

Sometimes it is more convenient to use ranks $0 \leq r < n$.

Randomized Quicksort

Recall randomized quicksort. For simplicity assume elements in A are unique.

- Pick a pivot $s \in A$ uniformly at random.
- Partition into $A^{<s}$, s, $A^{>s}$.
- Recursively sort $A^{<s}$ and $A^{>s}$.

Here A is assumed to be given as an array. Partitioning takes linear time (though is not so easy to implement in the presence of duplicates).

Let X be the random variable: size of $A^{<s}$. Then

$$p_i = \Pr[X = i] = 1/n$$

where $i = 0, \ldots, n-1$, $n = |A|$.

Ignoring multiplicative constants we get

$$t(n) = \begin{cases} 1 & \text{if } n \leq 1, \\ \sum_{i<n} p_i(t(i) + t(n-i-1)) + n & \text{otherwise.} \end{cases}$$

Simplifying

$$
\begin{aligned}
t(n) &= 1/n \sum_{i<n} \big(t(i) + t(n-i-1)\big) + n \\
&= 2/n \sum_{i<n} t(i) + n
\end{aligned}
$$

$$
\begin{aligned}
n \cdot t(n) &= 2 \sum_{i<n} t(i) + n^2 \\
(n+1) \cdot t(n+1) &= 2 \sum_{i \le n} t(i) + (n+1)^2
\end{aligned}
$$

$$
t(n+1) = (n+2)/(n+1) \cdot t(n) + (2n+1)/(n+1)
$$

which comes down to

$$
t(n) = \frac{n+1}{n} \cdot t(n-1) + 2.
$$

$t(n) = n + 1/n \cdot t(n-1) + 2$ can be handled in to ways:

- Unfold the equation a few levels and observe the pattern.
- Solve the homogeneous equation $h(n) = n + 1/n \cdot h(n-1)$: $h(n) = n + 1$. Then construct t from h – see any basic text on recurrence equations.

Either way, we find

$$t(n) = (n+1)/2 + 2(n+1) \sum_{i=3}^{n+1} 1/i = \Theta(n \log n)$$

Random versus Deterministic Pivots

Random pivot:

$$\mathsf{Pr}[X = k] = 1/n \qquad k = 0, \ldots, n-1$$

$$\mathsf{E}[X] = (n-1)/2$$

$$\mathsf{Var}[X] = (n^2 - 1)/12$$

Median of three:

$$\mathsf{Pr}[X = k] = \frac{6k(n-k-1)}{n(n-1)(n-2)} \qquad k = 1, \ldots, n-2$$

$$\mathsf{E}[X] = (n-1)/2$$

$$\mathsf{Var}[X] = ((n-1)^2 - 4)/20$$

While selection seems somewhat easier than sorting, it is not clear that one can avoid something like $O(n \log n)$ in the process of computing ord(k, A).

The following result was surprising.

Theorem (Blum, Floyd, Pratt, Rivest, Tarjan, 1973)

Selection can be handled in linear time.

The algorithm is a perfectly deterministic divide-and-conquer approach. Alas, the constants are bad.

Alternatively, we can use a randomized algorithm to find the kth element quickly, on average.

Given a collection A of cardinality n, a rank $0 \leq k < n$. Here is a recursive selection algorithm:

- Permute A in random order, yielding $a_0, a_1, \ldots, a_{n-1}$; set $B = \text{nil}$.
- Pick a pivot $s \in A$ at random and compute $A^{<s}$ and $A^{>s}$. Let $m = |A^{<s}|$.
- If $k = m$ return s.
- If $k < m$ return $\text{ord}(k, A^{<s})$.
- If $k > m$ return $\text{ord}(k - m - 1, A^{>s})$.

Correctness is obvious, for the running time analysis divide $[n]$ into bins of exponentially decreasing size: bin k has the form

$$B_k = [n \cdot (3/4)^k, n \cdot (3/4)^{k+1}]$$

where we ignore the necessary ceilings and floors, as well as overlap.

Note that with probability $1/2$ the cardinality of the selection set will move (at least) to the next bin in each round. But then it takes 2 steps on average to get (at least) to the next bin.

Hence the expected number of rounds is logarithmic and the total running time therefore linear.

- Order Statistics

2. Circuit Evaluation

- Yao's Minimax Principle

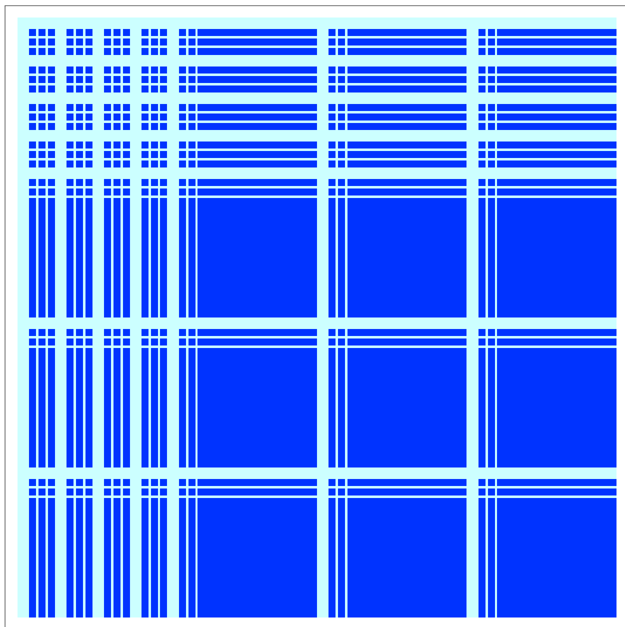
- More Randomized Algorithms *

Here is a highly simplified model of a game tree: we only consider Boolean values $\mathbf{2} = \{0, 1\}$ and represent the two players by alternating levels of "and" and "or" gates (corresponding to min and max).

More precisely, define Boolean functions $T_k : \mathbf{2}^{4^k} \to \mathbf{2}$ by

$$T_1(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$$

$$T_{k+1}(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4) = T_1\left(T_k(\boldsymbol{x}_1), T_k(\boldsymbol{x}_2), T_k(\boldsymbol{x}_3), T_k(\boldsymbol{x}_4)\right)$$



The Challenge: Given a truth assignment $\alpha : \boldsymbol{x} \to \mathbf{2}$, we want to evaluate the circuit T_k reading as few of the bits of α as possible (think of α as a bitvector of length 4^k).

We may safely assume that we always read the input bits from left to right.

For example, $x_1 = x_2 = 0$ already forces output 0 and we do not need to read x_3 or x_4 when evaluating T_1.

Skipping a single bit in T_1 may sound irrelevant, but skipping a whole subtree in T_3 is significant (16 variables).

Critical parameters:

$$R = \text{ output value}$$

$$S = \# \text{ variables read}$$

Augmented Truth Table

x_1	x_2	x_3	x_4	R	S
0	0	0	0	0	2
0	0	0	1	0	2
0	0	1	0	0	2
0	0	1	1	0	2
0	1	0	0	0	4
0	1	0	1	1	4
0	1	1	0	1	3
0	1	1	1	1	3
1	0	0	0	0	3
1	0	0	1	1	3
1	0	1	0	1	2
1	0	1	1	1	2
1	1	0	0	0	3
1	1	0	1	1	3
1	1	1	0	1	2
1	1	1	1	1	2

Essential Part

x_1	x_2	x_3	x_4	R	S
0	0	.	.	0	2
0	0	.	.	0	2
0	0	.	.	0	2
0	0	.	.	0	2
0	1	0	0	0	4
0	1	0	1	1	4
0	1	1	.	1	3
0	1	1	.	1	3
1	.	0	0	0	3
1	.	0	1	1	3
1	.	1	.	1	2
1	.	1	.	1	2
1	.	0	0	0	3
1	.	0	1	1	3
1	.	1	.	1	2
1	.	1	.	1	2

And Probability?

Think of choosing a truth assignment for x_1, x_2, x_3, x_4 at random. R and S are now discrete random variables.

Here is the PMF in the uniform case:

$R \backslash S$	1	2	3	4
0	0	1/4	1/8	1/16
1	0	1/4	1/4	1/16

$$\mathsf{E}[R] = 9/16 \approx 0.56$$

$$\mathsf{E}[S] = 21/8 \approx 2.63$$

Lemma

$\mathsf{E}[S_k] = 3^k = n^{\log_4 3} \approx n^{0.79}$

Proof.

Homework.

How about input with bias $\Pr[x = 1] = p$ for some $0 \leq p \leq 1$?

This is the bias for the original inputs at the input level of the circuit.

Note that this question is really inevitable: we have to worry about the influence of T_1 gates, even if the original bias is just $1/2$.

Table for Biased Input

x_1	x_2	x_3	x_4	R	S	Pr
0	0	0	0	0	2	q^4
0	0	0	1	0	2	pq^3
0	0	1	0	0	2	pq^3
0	0	1	1	0	2	p^2q^2
0	1	0	0	0	4	pq^3
0	1	0	1	1	4	p^2q^2
0	1	1	0	1	3	p^2q^2
0	1	1	1	1	3	p^3q
1	0	0	0	0	3	pq^3
1	0	0	1	1	3	p^2q^2
1	0	1	0	1	2	p^2q^2
1	0	1	1	1	2	p^3q
1	1	0	0	0	3	p^2q^2
1	1	0	1	1	3	p^3q
1	1	1	0	1	2	p^3q
1	1	1	1	1	2	p^4

It follows that for input with bias $\Pr[x = 1] = p$ we have

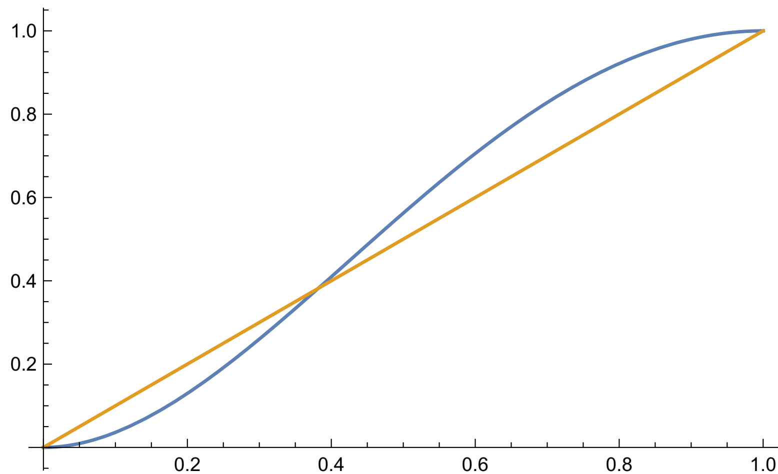
$$\mathsf{E}[R_1] = \Pr[R_1 = 1] = p^2(4 - 4p + p^2)$$

Sanity check: $p^2(4 - 4p + p^2)\ [p \mapsto 1/2] = 9/16$.

Claim

T_1 increases the bias for $p \geq (3 - \sqrt{5})/2 \approx 0.38$.

This is vaguely plausible since both "and" and "or" are monotonic. See the next plot.


1.0
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1.0

- Order Statistics

- Circuit Evaluation

3. Yao's Minimax Principle

- More Randomized Algorithms *

So What?

So the canonical lazy algorithm has $\mathsf{E}[S_k] = 3^k \approx n^{0.79}$.

This may sound good, but it would be nice to have a lower bound that indicates how good this result actually is.

It would be even nicer to have some general method for doing this.

One can use understanding of the performance of deterministic algorithms to obtain lower bounds on the performance of probabilistic algorithms. To make this work, focus on Las Vegas algorithms: the answer is always correct, but the running time may be bad, with small probability.

Given some input I, a Las Vegas algorithm A makes a sequence of random choices during execution. We can think of these choices as represented by a choice sequence $C \in \mathbf{2}^\star$.

Given I and C, the algorithm behaves in an entirely deterministic fashion: $A(I; C)$.

Fix some input size n once and for all (unless you love useless subscripts).

$$\mathcal{I} = \text{collection of all inputs of size } n$$

$$\mathcal{A} = \text{collection of all deterministic algorithms for } \mathcal{I}$$

It is clear that $\mathcal{I}$ is finite, but it requires a fairly delicate definition of "algorithm" to make sure that $\mathcal{A}$ is also finite.

Exercise

Figure out how to make sure $\mathcal{A}$ is finite.

We can think of a Las Vegas algorithm A as a probability distribution on $\mathcal{A}$: with some probability the algorithm executes one of the deterministic algorithms in $\mathcal{A}$.

This works both ways: given a probability distribution on $\mathcal{A}$ we can think of it as a Las Vegas algorithm (though this is not the way algorithm design works typically).

In the following, we are dealing with two probability distributions:

σ for the algorithm,

τ for the inputs.

We'll indicate selections according to these distributions by subscripts.

Theorem (Yao 1977)

$$\min_{A \in \mathcal{A}} \mathsf{E}[T_A(I_\tau)] \leq \max_{I \in \mathcal{I}} \mathsf{E}[T_{A_\sigma}(I)]$$

Thus, the average case (wrt τ) running time of the best deterministic algorithm is a lower bound for the expected (wrt σ) running time of the corresponding Las Vegas algorithm on the worst input.

The proof is by computation: show that $\sum_{(A,I)} \mathsf{Pr}[A]\mathsf{Pr}[I]T_A(I)$ separates the two values. Note that we are not assuming independence!

There is a natural Las Vegas algorithm to evaluate T_k: at every node in the tree, pick a subtree at random, evaluate it and then determine whether the other tree also needs to be evaluated.

From what we have seen, this algorithm will evaluate $O(n^{0.79})$ variables on average on any input $I \in \mathbf{2}^{4^k}$.

According to Yao's Minimax Principle we have to construct a random instance and determine the expected number input variabls read by any deterministic evaluation algorithm.

Dead in the Water

So we need to understand $\mathcal{A}$, the class of all deterministic algorithms, for evaluating T_k .

How on earth are we ever going to understand this class of algorithms? We know some of them, but who knows what kind of cockamamie methods there are?

Exercise

The performance of any deterministic algorithm can be matched or beaten by a top-down lazy algorithm.

This is not obvious, think about the necessary argument. At any rate, we only need to consider these algorithms to get the lower bound for Yao.

A simple computation shows that

$$T_1(x_1, x_2, x_3, x_4) = (x_1 \mathbin{\overline{\vee}} x_2) \mathbin{\overline{\vee}} (x_3 \mathbin{\overline{\vee}} x_4)$$

So, we can think of T_k as a homogeneous nor-tree of depth $2k$.

If we provide input to a nor gate with bias p, then the output has bias $(1-p)^2$.

The equation $(1-p)^2 = p$ has solution $p_0 = (3-\sqrt{5})/2 \approx 0.38$ and is visible as a fixed point in the graph in a previous slide.

Cost in Deterministic Algorithm

Let S_d be the cost of evaluating a node at depth d in the nor tree with bias p_0 by some top-down lazy method.

$$\begin{aligned} \mathsf{E}[S_d] &= p_0 \cdot \mathsf{E}[S_{d-1}] + (1 - p_0) \cdot 2 \cdot \mathsf{E}[S_{d-1}] \\ &= (2 - p_0)\, \mathsf{E}[S_{d-1}] \\ &= (1 + \sqrt{5})/2\, \mathsf{E}[S_{d-1}] \end{aligned}$$

It follows that $\mathsf{E}[S_{2k}] \approx n^{0.69}$, so no Las Vegas algorithm can do better than that in the worst case (i.e., on the worst possible input).

- Order Statistics
- Circuit Evaluation
- Yao's Minimax Principle

4. More Randomized Algorithms *

Occasionally the construction of a data structure can be simplified significantly if on assumes the input is sufficiently random: one can then build the data structure in a very brute-force, step-by-step manner that requires no complicated ideas and is fast on average.

For example, suppose we wish to construct a sorted list B from a given list A.

- Permute A in random order, yielding $a_1, a_2, \ldots, a_n$; set $B =$ nil.
- for $k = 1, \ldots, n$: insert a_k into B, in the proper place.

This looks like insertion sort, so why bother?

Because it isn't: we are going to maintain an additional data structure, a table that determines for each $x \in A - B$ which interval I defined by B element x belongs to. Moreover, for each interval I the table provides a list of all the elements in the interval.

Given the table, the insert step plus maintenance of the table can be handled in $O(|I|)$ steps.

So we need to find the expected value of the sum of the lengths of the intervals that we insert into.

A Trick: Going Backwards

Here is a trick that sometimes makes the argument a bit easier: run the algorithm backwards.

Here, going backwards in stage k means this: we randomly pick one of the k elements in B and remove it. Since the points in B are random, we should expect intervals of size n/k.

But then the total number of steps will about $nH_n = \Theta(n \log n)$, the best a comparison based sorting algorithm can do.

Alas, in practice, maintaining the table is cumbersome, so in the Real World this method is not competitive.

A set $A \subseteq \mathbb{R}^2$ is convex iff for all $x, y \in A$, the line segment $[x, y]$ is contained in A.

Note that $[x, y] = \{ \lambda x + (1 - \lambda) y \mid 0 \leq \lambda \leq 1 \}$.

Given an arbitrary set A, the convex hull of A is defined to be the least convex set containing A:

$$\mathsf{ch}(A) = \bigcap \{ C \mid A \subseteq C, C \text{ convex} \}.$$

This is a hull operation:

- $A \subseteq \mathsf{ch}(A)$.
- $\mathsf{ch}(\mathsf{ch}(A)) = \mathsf{ch}(A)$.

Note that the definition as stated is impredicative and hence not too useful (ch(A) is one of the sets on the right hand side). Here is a better one:

$$\text{ch}(A) = \{ \sum \lambda_i a_i \mid \sum \lambda_i = 1, 0 \leq \lambda_i, a_i \in A \}$$

The $\sum \lambda_i a_i$ are called convex combinations.

In particular when A is finite, say $A = \{a_1, \ldots, a_n\}$, we can obtain the hull as

$$\text{ch}(A) = \{ \sum \lambda_i a_i \mid \sum \lambda_i = 1, 0 \leq \lambda_i \}$$

Some of the a_i can be expressed as convex combinations of others, so the problem comes down to identifying $B \subseteq A$ such that $\text{ch}(B) = \text{ch}(A)$ but no proper subset works.

Hence a reasonable output format for the convex hull is to return a list

$$b_1, b_2, \ldots, b_m$$

of extremal points, obtained by traversing them in clockwise order, starting at the "top-left" point.

As a consequence of our output convention, we get a lower bound: we can use the convex hull to sort. To see why, suppose we have integral or rational numbers $x_1, \ldots, x_n$.

Define points $a_i = (x_i, x_i^2)$ on the parabola $y = x^2$.

Since the parabola is convex one can read the sorted list off the convex hull of A.

We will now match this bound with a randomized incremental algorithm to construct the hull.

For simplicity assume that A contains no collinear points.

- Permute A in random order, yielding $a_1, a_2, \dots, a_n$;
- Let $B = (a_1, a_2, a_3)$, let c be the centroid of this triangle.
- for $k = 4, \dots, n$: insert a_k into B:
  - if $a_k \in \text{ch}(B_{k-1})$ do nothing
  - otherwise modify B_{k-1} to include a_k.

As before, we will need to maintain additional information: for each point $a \in A - B$ the edge of the convex hull of B that intersects the line segment $[c, a]$.

In the opposite direction, we need for each edge a list of all the corresponding points.

Updating B may require the removal of $O(n)$ points from B, but the total number of removals is bounded by $2n$: we insert at most $2n$ points and we can charge for removal at the moment of insertion.

So the critical part is the update operation on the edge-points table: we need to process all the points currently associated with the edge that is being removed from the boundary of B_{k-1}.

Using the backward trick, the argument is precisely the same as for the sorting algorithm from above.

Here is a randomized algorithm for selection that uses a few magic numbers. The numbers make sense only when one performs a probabilistic analysis of the algorithm.

Convention: We will systematically ignore ceilings and floors and pretend that various numbers such as $\sqrt{n}$ are integral.

We are given a set $A \subseteq \mathcal{U}$ of n elements and we would like to determine $t = \text{ord}(k, A)$.

To this end, the algorithm selects a "small" subset B of A and works with B. Actually, we sample A with replacement.

Batten down the hatches.

1. Sample A with replacement $n^{3/4}$ times to produce B.
2. Sort B.
3. Let $\kappa = k/n^{1/4}$, $\kappa^- = \max(\kappa - \sqrt{n}, 1)$, $\kappa^+ = \min(\kappa + \sqrt{n}, n^{3/4})$, $b^\pm = \text{ord}(\kappa^\pm, B)$.
4. Compute $r^\pm = \text{rk}(b^\pm, A)$ – note the A.
5. Let
$$A_0 = \begin{cases} \{ x \in A \mid x \leq b^+ \} & \text{if } k < n^{1/4}, \\ \{ x \in A \mid x \geq b^- \} & \text{if } k > n - n^{1/4}, \\ \{ x \in A \mid b^- \leq x \leq b^+ \} & \text{otherwise.} \end{cases}$$
6. if $t \notin A_0$ or $|A_0| > 4n^{3/4}$ return to step 1.
7. Sort A_0 and return $\text{ord}(k - r^- + 1, A_0)$.

- Think of $n = 10^8$ so that $n^{3/8} = 10^6$ and $\kappa = k/100$.
- It is easier to pretend that B is a subset of A cardinality $n^{3/4}$. Alas, picking a subset of this size would make the algorithm more clumsy to implement and harder to analyze.
- In an ideal scenario, the elements in B would be equidistant; in that case we only would need to consider the interval spanned by the immediate neighbors of $\text{ord}(\kappa, B)$ in B. By going out to $\sqrt{n}$ we hope to compensate for the fact that B is not regularly placed.
- Let's count comparisons. The only part that is expensive is step (4), the total damage is $2n + o(n)$.
- The test in (6) is not impossible: we use the order isomorphism and check $r^- \leq k \leq r^+$ instead.

Lemma

The Crazy Selection algorithm terminates after one round with probability $1 - O(n^{-1/4})$.

Proof. Unfortunately, there are several cases to consider. For simplicity, we deal only with

$$A_0 = \{\, x \in A \mid b^- \leq x \leq b^+ \,\}$$

and show that $t \notin A_0$ is unlikely. $t \notin A_0$ means $t < b^-$ or $t > b^+$. In the first case we must have

$$\#(x \in B \;\big|\; x \leq t) < \kappa^-$$

and in the other case

$$\#(x \in B \;\big|\; x \leq t) > \kappa^+.$$

This suggests to consider to random variable

$$X = \#(x \in B \,\big|\, x \leq t)$$

which can be written as an indicator variable sum $X = \sum_{i=1}^{n^{3/4}} X_i$ where $X_i = 1$ iff the ith element in B is $\leq t$.

Note that we sample A with replacement and "ith element" means in the order of selection; X is really the number of samples below t (but for intuition think of it as cardinality).

It follows that $\Pr[X_i = 1] = k/n$.

Analysis, contd.

Clearly the X_i are Bernoulli, so we can calculate stats for X as follows:

$$\begin{aligned} \mathsf{E}[X] &= k/n \cdot n^{3/4} = kn^{-1/4} = \kappa \\ \mathsf{Var}[X] &= n^{3/4} \cdot k/n \cdot (1 - k/n) \leq 1/4 \cdot n^{-1/4} \\ \sigma &\leq 1/2\, n^{3/8} \end{aligned}$$

The bound on $\mathsf{Var}[X]$ follows from considering the parabola $x(1-x)$.

By Chebyshev,

$$\mathsf{Pr}[|X - \kappa| \geq \sqrt{n}] \leq \mathsf{Pr}[|X - \kappa| \geq 2n^{1/8}\sigma] = O(n^{-1/4})$$

It follows that $\Pr[t < b^-] = O(n^{-1/4})$.

Essentially the same argument shows that $\Pr[b^+ < t] = O(n^{-1/4})$.

But the probability of the union of the two failure modes is bounded by the sum of the respective probabilities, which is still $O(n^{-1/4})$.

□

Note that the bound $O(n^{-1/4})$ is not overwhelming; we have not even made an attempt to estimate the constants.

We certainly would not want to use a recursive version of the algorithm.