A&W

QuickSelect: Expected Runtime is $O(n)$

Detailed Walkthrough

Goal of these notes. We prove that randomized QuickSelect — the algorithm that finds the k-th smallest element of an array — runs in $O(n)$ expected time. Two ideas do all the work: (i) a good-pivot argument showing that with probability $\geq 1/2$, one recursive step shrinks the problem size by factor $3/4$, and (ii) a bucketing-and-geometric-series argument that turns this per-step shrinkage into a linear total-cost bound.

Contents

---

1 Setup and notation

The objects we work with throughout:

- The array $A[1..n]$ of n *distinct* elements.
- The query $k \in \{1, \ldots, n\}$: we want the k-th smallest element.
- The cost measure: T, the random number of comparisons made by QuickSelect$(A, 1, n, k)$.

1.1 The algorithm

QuickSelect mirrors QuickSort's partition step but recurses into *only one side* — whichever contains the target rank. Given current slice $A[\ell..r]$ and current target index k within that slice:

1. Pick a random pivot $p \sim \text{Uniform}(\{\ell, \dots, r\})$.
2. Run Partition(A, ℓ, r, p), which returns the pivot's final position t.
3. If $t = \ell + k - 1$: return $A[t]$. Done.
4. If $t > \ell + k - 1$: target is in the left part. Recurse on QuickSelect$(A, \ell, t - 1, k)$.
5. If $t < \ell + k - 1$: target is in the right part. Recurse on QuickSelect$(A, t+1, r, k-(t-\ell+1))$.

Remark 1.1: Why this is faster than sorting

QuickSort would sort the whole array (cost $\Theta(n \log n)$) and then read off the k-th entry. QuickSelect does the same partition step but throws away the half of the array that cannot contain the target, so half the work disappears at each step. The geometric series of remaining work converges, giving $O(n)$.

1.2 The destination

Theorem 1.1 (Expected runtime of randomized QuickSelect)**.** *For any input array A of n distinct elements and any $k \in \{1, \dots, n\}$,*

$$\mathbb{E}[T] \;\le\; 8n \;=\; O(n).$$

The next two sections build up to this bound. We first express T as a sum that we can bucket (Section 2), then bound each bucket using a good-pivot argument (Section 3), and finally collapse everything via a geometric series (Section 4).

2 Expressing the total cost

2.1 Cost is the sum of partition sizes

QuickSelect produces a (random) sequence of recursive calls

$$(\ell_0, r_0, k_0),\ (\ell_1, r_1, k_1),\ \dots,\ (\ell_N, r_N, k_N)$$

with $(\ell_0, r_0, k_0) = (1, n, k)$ and where N is the (random) number of calls before the algorithm halts. Each call invokes Partition once, which costs exactly $r_i - \ell_i$ comparisons (same Partition as in QuickSort). Comparisons happen nowhere else, so:

$$\boxed{T \;=\; \sum_{i=1}^{N} (r_i - \ell_i).}$$

2.2 Bucketing the calls by size

The slice sizes $r_i - \ell_i + 1$ form a non-increasing sequence (each recursive call works on a strict subset of the previous slice). They start at n and trend down to 1. We group calls into geometric-size buckets:

Definition 2.1 (Size buckets)**.** For each $j \geq 1$, let

$$N_j \;:=\; \#\Big\{\, i \,:\, (3/4)^j\, n \,<\, r_i - \ell_i + 1 \,\leq\, (3/4)^{j-1}\, n \,\Big\}.$$

In words: N_j counts the number of calls whose slice size sits in the band $\big((3/4)^j n,\, (3/4)^{j-1} n\big]$.

Remark 2.1: Why these buckets

Bucket j collects all calls whose slice size has been shrunk by roughly factor $(3/4)^{j-1}$ from the original. Bucket 1 is calls of size in $(\frac{3n}{4}, n]$, bucket 2 is calls of size in $(\frac{9n}{16}, \frac{3n}{4}]$, and so on. Every call belongs to exactly one bucket.

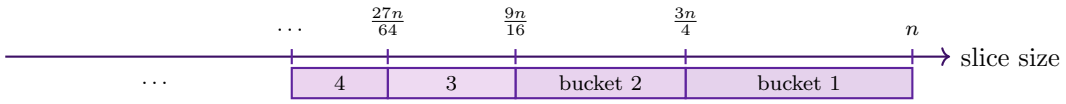
Since each call in bucket j has size at most $(3/4)^{j-1}n$:

$$T \;=\; \sum_{i=1}^{N} (r_i - \ell_i) \;\leq\; \sum_{j=1}^{\infty} N_j \cdot (3/4)^{j-1}\, n.$$

Take expectation (linearity):

$$\boxed{\mathbb{E}[T] \;\leq\; n \cdot \sum_{j=1}^{\infty} \mathbb{E}[N_j] \cdot (3/4)^{j-1}.}$$

What's left. If we can show $\mathbb{E}[N_j] \leq 2$ for every $j \geq 1$, the geometric series in the bound converges to a constant and we're done. That's the entire goal of Section 3.



3 Bounding $\mathbb{E}[N_j]$ via good pivots

We now show $\mathbb{E}[N_j] \leq 2$ for every j. The key concept:

3.1 Good pivots

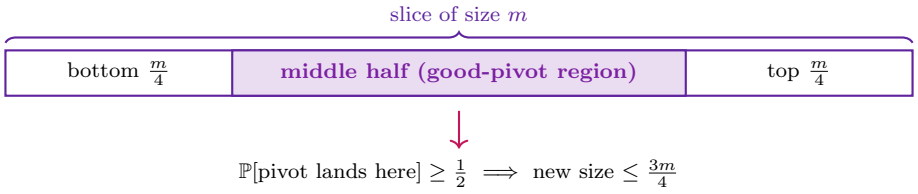
Definition 3.1 (Good pivot)**.** On a slice of size m, call the pivot good if it falls within the *middle half* of the slice — i.e., its rank within the slice is in $\big(\frac{m}{4}, \frac{3m}{4}\big]$.

A good pivot has two properties we'll use repeatedly:

Key Idea 3.1: The good-pivot dichotomy

- Probability: the pivot is uniform over m ranks, and the middle half contains $\geq m/2$ of them, so $\mathbb{P}[\text{good pivot}] \geq \frac{1}{2}$.
- Shrinkage: if the pivot is good, the larger of the two pieces has size $\leq \frac{3m}{4}$. Since QuickSelect recurses into one piece, the new slice has size $\leq \frac{3m}{4}$.

Why "shrinkage"? If the pivot's rank is $p \in (\frac{m}{4}, \frac{3m}{4}]$, the two pieces have sizes $p-1$ and $m-p$. Both are at most $\frac{3m}{4}$, so whichever side we recurse on has size $\leq \frac{3m}{4}$.



3.2 From good pivots to $\mathbb{E}[N_j] \leq 2$

Fix a bucket j. Every call in bucket j has slice size in $\big((3/4)^j n,\ (3/4)^{j-1} n\big]$. *Whenever* a call in bucket j uses a good pivot, the next slice size is $\leq \frac{3}{4} \cdot (3/4)^{j-1} n = (3/4)^j n$, which kicks the next call out of bucket j (into bucket $j+1$ or beyond).

Key Idea 3.2: Each bucket is a geometric trial

While in bucket j, each pivot is good with probability $\geq \frac{1}{2}$ *independently* of past pivots. The number of calls in bucket j is therefore upper-bounded by a geometric random variable with success probability $p \geq \frac{1}{2}$:

$$\mathbb{E}[N_j] \ \leq \ \frac{1}{p} \ \leq \ 2.$$

Remark 3.1: The independence is real

Pivots are drawn independently in each call, so "good or not" at step i is independent of past goodness. This is what justifies modeling N_j as the number of trials until success in a Bernoulli(p) sequence with $p \geq 1/2$.

Example 3.1: Visualizing one bucket

Imagine we are currently in bucket j. At each subsequent call, flip a coin (heads = good pivot, prob. $\geq 1/2$). The number of calls before the first heads — which kicks us out of the bucket — has expectation at most 2. So on average, we spend at most 2 calls per bucket.

4 Putting it together

Plugging $\mathbb{E}[N_j] \leq 2$ into our earlier bound:

$$\mathbb{E}[T] \ \leq \ n \cdot \sum_{j=1}^{\infty} \mathbb{E}[N_j] \cdot (3/4)^{j-1} \ \leq \ 2n \cdot \sum_{j=1}^{\infty} (3/4)^{j-1}.$$

The geometric series sums to a constant:

$$\sum_{j=1}^{\infty} (3/4)^{j-1} \ = \ \frac{1}{1 - 3/4} \ = \ 4.$$

Therefore:

$$\boxed{\mathbb{E}[T] \ \leq \ 2n \cdot 4 \ = \ 8n \ = \ O(n).} \quad \square$$

Watch out 4.1: Common confusions

- "Why middle *half*, not middle third or middle 80%?" Any constant fraction works; the middle half is the cleanest because $p = 1/2$ gives $\mathbb{E}[N_j] \le 2$ immediately. Tweaking the constants only changes $8n$ to a different cn — the $O(n)$ conclusion is unaffected.
- "Why is $\mathbb{E}[N_j] \le 2$ and not $\le 1/p$ for all j at once?" The bound ≤ 2 already *is* $1/p$ with $p = 1/2$. We use the same bound for every bucket because the same probability-$\ge 1/2$ argument applies independently in each.
- "Why don't we get $O(n \log n)$ like QuickSort?" Because QuickSelect recurses into *one* side only. QuickSort recurses into both, so its total work per "level" is $\Theta(n)$ across $\Theta(\log n)$ levels. QuickSelect only does work along a single path, and the geometric shrinkage of that path's sizes makes the total a convergent geometric series.
- Worst case is still $\Theta(n^2)$. The bound is on the *expected* number of comparisons, where the expectation is over the random pivot, not over the input. The worst case (pivot always lands at an extreme rank) still has runtime $\Theta(n^2)$ — but it has vanishing probability.

5 Closing summary

The structure of the argument in one line:

$$T = \sum_i (r_i - \ell_i) \xrightarrow[\text{geometric size buckets}]{\text{step 1}} \mathbb{E}[T] \le n \sum_j \mathbb{E}[N_j](3/4)^{j-1} \xrightarrow[\text{good pivot, } p \ge 1/2]{\text{step 2}} \mathbb{E}[N_j] \le 2$$
$$\xrightarrow[\text{geometric series}]{\text{step 3}} \mathbb{E}[T] \le 8n.$$

Two reusable tricks.

1. Bucket by size. When a recursive algorithm shrinks its input by some factor $\rho < 1$ at each step, group calls into bands of size $\left(\rho^j n, \rho^{j-1} n\right]$. The total cost becomes a geometric series in ρ which converges as long as $\rho < 1$.
2. Good-event argument for geometric distributions. If a "good event" has probability $\ge p$ at each step, the number of steps before the first good event has expectation $\le 1/p$. Combine with linearity of expectation to bound a sum of trials.

Comparison with QuickSort. QuickSort's $O(n \log n)$ analysis was *algebraic*: derive a recurrence for t_n, kill the sum with shift-and-subtract, telescope into a harmonic sum. QuickSelect's $O(n)$ analysis is *probabilistic*: bucket the calls geometrically, apply a good-event argument per bucket, sum a geometric series. The two analyses use entirely different toolkits even though the underlying algorithms differ by one line.