32.2 Quick Select

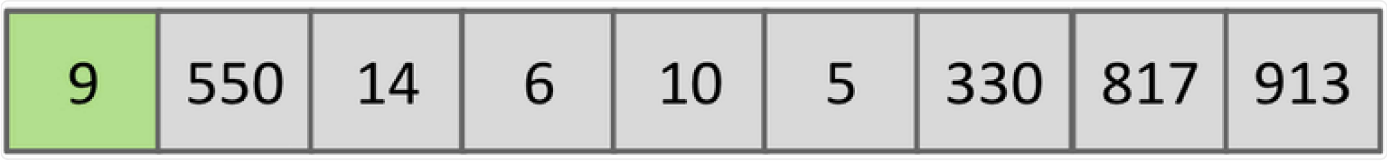
Dissatisfied with the result that Quicksort is unable to completely defeat Mergesort and truly claim the title for "fastest comparison-based sorting" algorithm, let's shoot our one last shot to overturn the result: use Quick Select to identify the median.

Quick Select Algorithm

Goal: find the median using partioning.

Key idea: Median of an length n array will be around index n /2.

1. Initialize array with the leftmost item as the pivot.



Initial Array

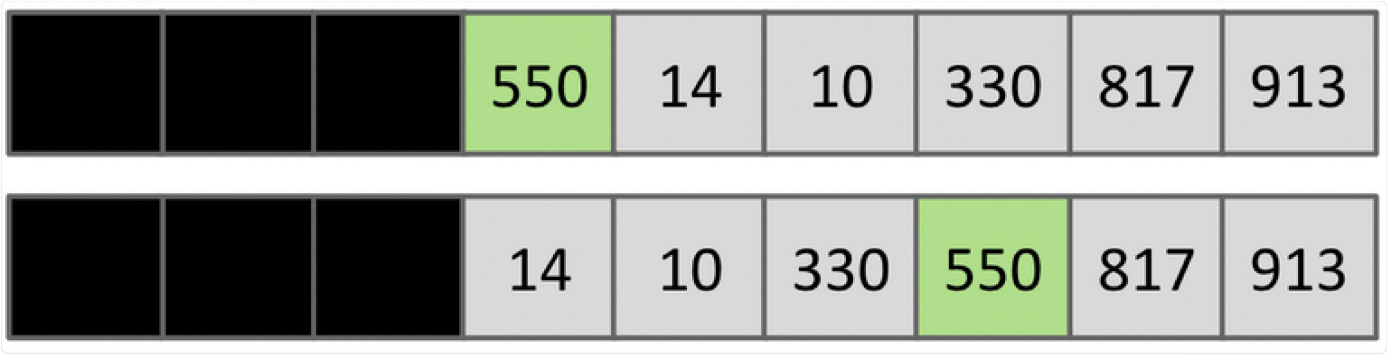
For this example, the index of the median should be 9 / 2 = 4.

2. Partition around pivot.

Pivot lands at index 2

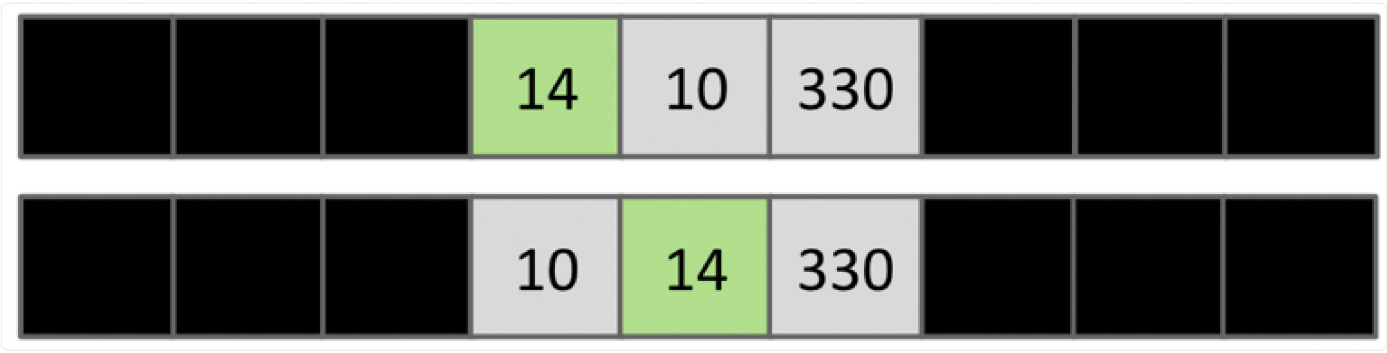
Since the pivot is at index 2, which is not the median. Therefore, need to continue. However, will only partition the **right** subproblem because median can't be to the left! (index n/2 > 2)

3. Partition the subproblem. Repeat the process.



Pivot is at index 6, which means that it is not at the median. Continue.

4. Stop when the pivot is at the median index.



Since pivot is at index 4, we are done.

Quick Select Performance

Worst Case Runtime

The worst case is when the array is in sorted order, which yields a runtime of $\theta(N^2)$.

[1 2 3 4 5 6 7 8 9 10 … N]

[1 **2 3 4 5 6 7 8 9 10 … N**]

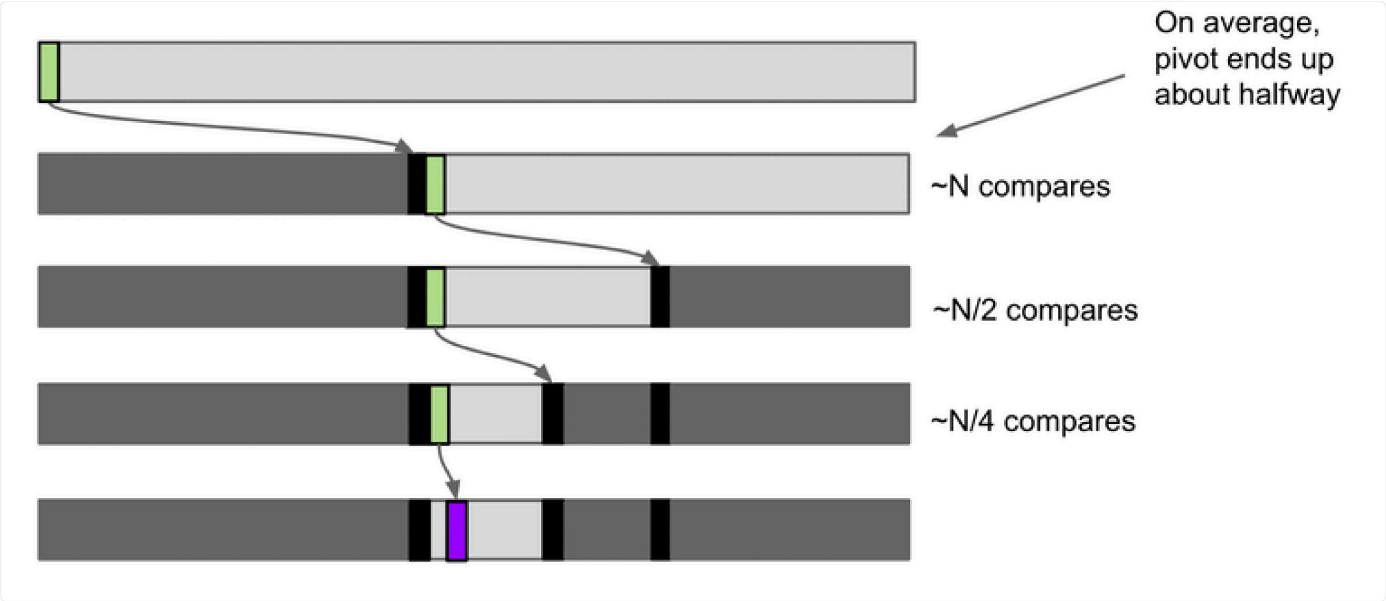
[1 2 **3 4 5 6 7 8 9 10 … N**]

…

[1 2 3 4 5 … **N/2** … N]

Expected Runtime

The expected runtime for the Quick Select Algorithm is $\theta(N)$. Here's an intuitive diagram that justifies this result:



Using one of our favorite sums, we can see the runtime is:

$$N + N/2 + N/4 + ... + 1 = \theta(N)$$

Compared to MergeSort?

Unfortunately, even using Quickselect to find the exact median, the resulting algorithm is still quite slow.

Team Mergesort wins, sadly.

Previous
32.1 Quicksort Flavors vs. MergeSort

Next
32.3 Stability, Adaptiveness, and Optimization

Last updated 1 year ago