Naive Quicksort

1. Partition around a pivot item, e.g. leftmost item.
2. Quicksort left side, all keys ≤ pivot.
3. Quicksort right side, all keys ≥ pivot.

Demo

32 15 2 17 19 26 41 17 17

Partition(32)

≤ 32

≥ 32

32's final position

Quick sort Case Analysis

Sort | Best-Case | Worst-Case | Space | Stable | Notes
--- | --- | --- | --- | --- | ---
Selection Sort | Ω(N) | Θ(N²) | Θ(1) | No | Slow in practice.
Heapsort | Ω(N log N) | Θ(N log N) | Θ(1) | No | Best for small or almost sorted inputs.
Merge Sort | Ω(N log N) | Θ(N log N) | Θ(N) | Yes | Fastest stable sort.
Insertion Sort | Ω(N) | Θ(N²) | Θ(1) | Yes | Best for small or almost sorted inputs.
Naive Quicksort | Ω(N log N) | Θ(N²) | Θ(N) | Yes | 2x or more slower than merge sort.
Java Quicksort | Ω(N) | O(N²) | ? | No | Fastest comparison sort.
Argument 1: 10% Case

Suppose the pivot is always at least 10% from either edge (not to scale).

N/10 9N/10
N/100 9N/100 9N/100 81N/100

Work at each level is in \( O(N) \).
Height is about \( \log_{\frac{10}{9}} N \in O(\log N) \).
Overall: \( O(N \log N) \).

Argument 2: Binary Search Tree Analogy

Random insertion into a binary search tree is expected to take \( O(N \log N) \) time.

Optimizing Quicksort

Naive Quicksort
Recursive Depth. \( \Omega(\log N) \), \( O(N) \).

Pivot choice. Leftmost item. \( \Theta(1) \)
Common worst-case: sorted array!

Partitioning. Allocate a new array. \( \Theta(N) \)
Slow but stable.
Common worst-case: all duplicates!

Java Quicksort
5x or more faster.
Recursive Depth. \( \Theta(\log N) \), \( O(N) \).

Pivot choice. Approximate median. \( \Theta(1) \)
Resilient to worst-case inputs.

Partitioning. Long-distance swaps. \( \Theta(N) \)
In-place, fast, but unstable.
3-way partition to handle duplicates.

Goal. Find the median item in \( O(N) \) time.
Reduces to the selection problem.

Selection. Given an array of \( N \) items, find item of rank \( K \).

For median, find \( K = N / 2 \).

How difficult is this problem?
- Why is the time complexity of selection in \( O(N) \)?
- Describe an \( O(N \log N) \) runtime algorithm for selection with any \( K \).
- Describe an \( O(N) \) runtime algorithm for selection with \( K = 0, 1, 2 \).
If selection reduces to sorting, which of the following statements about problem difficulty is true?

- Selection ≤ Sorting
- Sorting ≤ Selection
- Selection > Sorting
- Sorting > Sorting

Not sure

Unfortunate reality: Quicksort with quickselect pivots is significantly slower than merge sort.

Goal. Find the approximate median item in Θ(1) time.

Median-of-3. Pick 3 items and take the median of the sample.

if (a < b)
    if (b < c) return b;
else if (a < c) return c;
else return a;
else
    if (a < c) return a;
else if (b < c) return c;
else return b;

Median-of-3 Decision Tree
Hoare Partitioning

**Hoare partitioning.** In-place, unstable partitioning algorithm. Initialize an int \( L \) and an int \( G \).

- \( L \). Left pointer that loves small items < pivot.
- \( G \). Right pointer that loves big items > pivot.

**Idea.** Walk towards each other, swapping anything they don’t like.

End result is that things on left are “small” and things on the right are “large”.

Hoare partitioning improves real-world runtime and space complexity.
Asymptotic time complexity still depends on pivot choice!

Dual-Pivot Quicksort

If classic quicksort is analogous to BSTs, then dual-pivot quicksort is analogous to 2-3 trees.

<table>
<thead>
<tr>
<th>Sort</th>
<th>Best-Case</th>
<th>Worst-Case</th>
<th>Space</th>
<th>Stable</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection Sort</td>
<td>( \Theta(N^2) )</td>
<td>( \Theta(N^2) )</td>
<td>( \Theta(1) )</td>
<td>No</td>
<td>Slow in practice.</td>
</tr>
<tr>
<td>Heapsort</td>
<td>( \Theta(N) )</td>
<td>( \Theta(N \log N) )</td>
<td>( \Theta(1) )</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Merge Sort</td>
<td>( \Theta(N \log N) )</td>
<td>( \Theta(N \log N) )</td>
<td>( \Theta(1) )</td>
<td>No</td>
<td>Fastest stable sort.</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>( \Theta(N) )</td>
<td>( \Theta(N^2) )</td>
<td>( \Theta(1) )</td>
<td>Yes</td>
<td>Best for small or almost sorted inputs.</td>
</tr>
<tr>
<td>Naive Quicksort</td>
<td>( \Theta(N \log N) )</td>
<td>( \Theta(N^2) )</td>
<td>( \Theta(N) )</td>
<td>Yes</td>
<td>2x or more slower than merge sort.</td>
</tr>
<tr>
<td>Java Quicksort</td>
<td>( \Theta(N) )</td>
<td>( \Theta(N^2) )</td>
<td>( O(\log N) )</td>
<td>No</td>
<td>Fastest comparison sort.</td>
</tr>
</tbody>
</table>