Sorting Algorithms

Seen insertion sort - good for nearly sorted data but worst-case $\Theta(n^2)$ time

Seen merge sort - worst-case $\Theta(n \log n)$ time ($T(n) = 2T(n/2) + \Theta(n)$)

Today we'll study quicksort
Quicksort

Divide-and-Conquer Alg
Do all the hard work in splitting (a recursive call). No combine.

Divide: Partition array into two
Subarrays where all elements in left portion are less than all elements in right portion
Example of Partition

\[ \ldots 11, 4, 9, 7, 3, 10, 2, 6, 8 \ldots \]

\[ \uparrow \]

\[ i \]

\[ \ldots 6, 4, 9, 7, 3, 10, 2, 11, 8 \]

\[ \uparrow \]

\[ j \]

\[ \ldots 6, 4, 2, 7, 3, \boxed{10, 9, 11, 8} \]

\[ \uparrow \]

\[ j \]

\[ \boxed{6, 4, 2, 7, 3, 8, 9, 11, 10} \]
General invariant maintained

... left

\( \leq \text{pivot} \)
\( \text{not yet processed} \)
\( \geq \text{pivot} \)

... right

\( \text{pivot} \)
Java Code For Partition

Final position of pivot element

int partition(int left, int right, Comparator<? super E> sorter) {
    E pivot = read(right);  // pivot around the right element
    int i = left;
    int j = right;
    while (i < j) {
        while (i < j && sorter.compare(read(i), pivot) < 0)  // pos. i element < pivot
            i++;
        while (j > i && sorter.compare(read(j), pivot) >= 0)  // pos. i element >= pivot
            j--;
        if (i < j)
            swapImpl(i, j);  // swaps pos. i and pos. j elements
    }
    swapImpl(i, right);
    return i;
}
QuickSort (Comparator sorter) {
    quickSortImpl (0, n-1, sorter)
}

\[\text{method of positional collection interface}\]

\[
\text{quickSortImpl (left, right, sorter)} \{ \\
\text{if (left < right)} \{ \\
\text{swap what's in pos right using median-of-three or by random} \\
\text{mid = partition (left, right, sorter);} \\
\text{quickSortImpl (left, mid-1, sorter);} \\
\text{quickSortImpl (mid+1, right, sorter);} \\
\text{end if} \\
\} \\
\} \\
\]
Time Complexity

Asymptotic time complexity for Partition is $\Theta(n)$

Best Case: Pivot is in middle

$$T(n) = 2T\left(\frac{n-1}{2}\right) + \Theta(n) = \Theta(n \log n)$$
What’s the worst-case?

What if pivot is min or max?

What if positional collection was already sorted?

- n-1 elements in left
- no elements in right
- mid + 1, right
Leads to recurrence

\[ T(n) = T(n-1) + \Theta(n) \]

\[
\begin{align*}
\begin{cases}
  \frac{c}{n} & \text{for } n \\
  c(n-1) & \text{for } n-1 \\
  c(n-2) & \text{for } n-2 \\
  \vdots & \text{for } \vdots \\
  c \cdot 2 & \text{for } 2 \\
  c & \text{for } 1 \\
\end{cases}
\end{align*}
\]

Sum

\[ c \left( 1 + 2 + \ldots + n \right) \]

\[ \frac{c \cdot n(n+1)}{2} = \Theta(n^2) \]
How can we pick the partition to try to avoid bad behavior.

**Median-of-three partitioning**

Find median of $a$, $b$, $c$ and then if this is not $c$, swap $c$ with median
The other common solution is

**Randomized Quicksort**

Pick a random element (uniformly) from subarray $+$ swap that with element in rightmost position of the subarray

Show highlights that expected (average) time is $\Theta(n \log n)$
One common optimization is as follows.

1. Terminate when subarray size is relatively small ($n \approx 30$)

2. Run insertion sort after quick sort is done