Quick Sort
Quick Sort: a randomized sorting algorithm

QSort(L):

choose $p$ from $L$ at random
partition $L$ into 3 sublists: $L_{<p}, L_{=p}, L_{>p}$

if $\text{length}(L_{<p}) > 0$ then $L_{<p} \leftarrow \text{QSort}(L_{<p})$
if $\text{length}(L_{>p}) > 0$ then $L_{>p} \leftarrow \text{QSort}(L_{>p})$
return $\text{append}(L_{<p}, L_{=p}, L_{>p})$

Practical implementation:

• use the fast, in-place partition algorithm from last time
Intuition:

we split the problem into two problems of “roughly equal” size (in linear time) and then solve both of them

reminds us of the recurrence $T(n) \leq 2T(n/2) + O(n)$

Master Theorem says: $T(n) = O(n \log n)$

BUT . . . this is not rigorous: the splitting step is probabilistic, and we may get some “bad” splits
Quick and dirty analysis of Quick Sort

Idea: leverage our randomized Quick Select analysis

Imagine that $L$ is sorted, and for each $k = 1, \ldots, n$, we can consider the $k$th element $x_k$ in the sorted list

Let $C_k$ be the number of levels at which $x_k$ occurs in the recursion tree

For $j = 0, 1, \ldots$, let $S_j :=$ number of items at level $j$

$W :=$ number of comparisons $\leq \sum_{j\geq 0} S_j \leq \sum_{k=1}^{n} C_k$
Let $C_k$ be the number of levels at which $x_k$ occurs in the recursion tree.

For $j = 0, 1, \ldots$, let $S_j := \text{number of items at level } j$.

$W := \text{number of comparisons } \leq \sum_{j \geq 0} S_j \leq \sum_{k=1}^{n} C_k$.

**Key observation:** the distribution of $C_k$ is precisely the same as the distribution of the recursion depth of $QSelect(L, k)$ (plus 1).

**Idea:** from $x_k$’s point of view, we are just running $QSelect(L, k)$.

**Therefore:** $E[C_k] = O(\log n)$ for each $k$, and

$$E[W] \leq \sum_{k=1}^{n} E[C_k] = O(n \log n).$$
Expected Depth of Quick Sort Recursion

Let $D := \text{depth of the recursion tree for } QSort \text{ on inputs of length } n$

**Theorem:** $E[D] = O(\log n)$

**Notes:**

The $QSelect$ depth analysis does not apply — again, $E[\max\{X, Y\}] \not\leq \max\{E[X], E[Y]\}$

$E[D]$ can also be viewed as the average height of a randomly built binary search tree.
The recursion tree in more detail . . .

\[ N_i := \text{size of node } i \]
\[ \mathcal{L}_j := \text{set of indices at level } j \]
\[ T_j := \sum_{i \in \mathcal{L}_j} N_i^2 \]

The \( N_i \)'s and \( T_j \)'s are random variables

**Claim:** \( E[T_j] \leq (2/3)^j n^2 \) for \( j = 0, 1, 2, \ldots \)
**Proof of claim:** \( E[T_j] \leq (2/3)^j n^2 \) for \( j = 0, 1, 2, \ldots \)

Let’s first prove that \( E[T_1] \leq (2/3)n^2 \)

\[ T_1 = N_2^2 + N_3^2 \]

Imagine the items are in \( L \) are sorted

Let \( R \) be the index of the pivot in the sorted list

\( R \) is uniformly distributed over \( \{1, \ldots, n\} \)

\( N_2 \leq R - 1 \) and \( N_3 \leq n - R \)

\[
E[(R - 1)^2] = \sum_{i=1}^{n} (i - 1)^2 / n = \frac{1}{n} \sum_{i=0}^{n-1} i^2 \\
\leq \frac{1}{n} \int_0^n x^2 \, dx = \frac{1}{n} \cdot \frac{n^3}{3} = \frac{n^2}{3}
\]
The distribution of $n - R$ is the same as that of $R - 1$

Thus, $E[N_2^2] \leq n^2/3$, $E[N_3^2] \leq n^2/3$, and $E[T_1] = E[N_2^2] + E[N_3^2] \leq (2/3)n^2$

More generally, consider any node $i$ in the tree

“Law of total expectation”:
\[
E[N_{2i}^2] = \sum_m E[N_{2i}^2 \mid N_i = m] \Pr[N_i = m] \\
\leq \sum_m \left(\frac{m^2}{3}\right) \Pr[N_i = m] = \left(\frac{1}{3}\right) E[N_i^2]
\]

Similarly, $E[N_{2i+1}^2] \leq (1/3) E[N_i^2]$

This shows: $E[T_{j+1}] \leq (2/3) E[T_j]$ for $j \geq 0$
Implies claim: $E[T_j] \leq (2/3)^j n^2$ for $j \geq 0$ (induction)

**Tail sum formula:** $E[D] = \sum_{j \geq 1} \Pr[D \geq j]$

**Observe:** $D \geq j \iff T_j \geq 1$

**Markov:** $\Pr[T_j \geq 1] \leq E[T_j] \leq (2/3)^j n^2$

A calculation almost identical to that for QSelect

Setting $j_0 := \lceil \log_{3/2}(n^2) \rceil$:
// = least $j$ such that $(2/3)^j n^2 \leq 1$

$$E[D] = \sum_{j \geq 1} \Pr[D \geq j]$$

$$= \sum_{j=1}^{j_0-1} \Pr[D \geq j] + \sum_{j=j_0}^{\infty} \Pr[D \geq j]$$

$$\leq \log_{3/2}(n^2) + 3 = O(\log n)$$
Since the work per level is $O(n)$, this gives another proof that the expected running time of $QSort$ is $O(n \log n)$

But . . . constants are suboptimal

Homework develops alternative analyses of $QSelect$ and $QSort$ with optimal constants