8. Average-Case Analysis of Algorithms + Randomized Algorithms
Array $A[1]..A[n]$

for $i = 1..n-1$ {
    $T = A[i]$
    $j = i-1$
    while $j >= 0$ && $T < A[j]$ {
        $A[j] = T$
        $j = j-1$
    }
    $A[j+1] = T$
}

Sorted

Unsorted

"compare"

"swap"
Run Time

Worst Case: $O(n^2)$

( $\sim n^2$ swaps; $\#\text{compares} = \#\text{swaps} + n - 1$)

“Average Case”

? What’s an “average” input?

One idea (and about the only one that is analytically tractable): assume all $n!$ permutations of input are equally likely.
permutations & inversions

A permutation $\pi = (\pi_1, \pi_2, ..., \pi_n)$ of 1, ..., n is simply a list of the numbers between 1 and n, in some order.

(i,j) is an inversion in $\pi$ if $i < j$ but $\pi_i > \pi_j$

E.g.,

$$\pi = (3 \ 5 \ 1 \ 4 \ 2)$$

has six inversions: (1,3), (1,5), (2,3), (2,4), (2,5), and (4,5)

Min possible: 0:

$$\pi = (1 \ 2 \ 3 \ 4 \ 5)$$

Max possible: n choose 2:

$$\pi = (5 \ 4 \ 3 \ 2 \ 1)$$

Obviously, the goal of sorting is to remove inversions
Swapping an adjacent pair of positions that are out-of-order decreases the number of inversions by exactly 1.

So..., number of swaps performed by insertion sort is exactly the number of inversions present in the input. Counting them:

\[
I_{i,j} = \begin{cases} 
1 & \text{if } (i, j) \text{ is an inversion} \\
0 & \text{if not}
\end{cases}
\]

\[
I = \sum_{i<j} I_{i,j}
\]

\[
E[I] = E \left[ \sum_{i<j} I_{i,j} \right] = \sum_{i<j} E[I_{i,j}]
\]
There is a 1-1 correspondence between permutations having inversion \((i,j)\) versus not:

$$
\begin{align*}
\pi & \quad (\cdots \ a \ \cdots \ b \ \cdots) \\
\pi' & \quad (\cdots \ b \ \cdots \ a \ \cdots)
\end{align*}
$$

So:

$$
E[I_{i,j}] = P(I_{i,j} = 1) = \frac{1}{2}
$$

$$
E[I] = \sum_{i<j} E[I_{i,j}] = \sum_{i<j} \frac{1}{2} = \binom{n}{2} \cdot \frac{1}{2}
$$

Thus, the expected number of swaps in insertion sort is \(\binom{n}{2}/2\) versus \(\binom{n}{2}\) in worst-case. I.e.,

The average run time of insertion sort (assuming random input) is about half the worst case time.
average-case analysis of quicksort

Quicksort also does swaps, but *non* adjacent ones.

Recall method:

Array $A[1..n]$

“pivot” = $A[1]$

“Partition” (O(n) compares/swaps) so that:

$\{A[1], ..., A[i-1]\} < \{A[i] == \text{pivot}\} < \{A[i+1], ..., A[n]\}$

recursively sort $\{A[1], ..., A[i-1]\}$ and $\{A[i+1], ..., A[n]\}$
quicksort run-time

Worst case: already sorted (among others) –

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

\[ = n + (n-1) + (n-2) + \ldots + 1 = \frac{n(n+1)}{2} \]

Best case: pivot is always median \( \Rightarrow \sim n \log_2 n \)

Average case: ？

Below. Will turn out to be \( \sim 40\% \) slower than best

Why?

Random pivots are “near the middle on average”
average-case analysis

Assume input is a random permutation of 1, ..., n, i.e., that all n! permutations are equally likely

Then 1st pivot \( A[1] \) is uniformly random in 1, ..., n

Important subtlety:

- pivots at all recursive levels will be random, too,
- (unless you do something funky in the partition phase)
Let $C_N$ be the average number of comparisons made by quicksort when called on an array of size $N$. Then:

$$C_0 = C_1 = 0 \quad \text{(list of length } \leq 1 \text{ is already sorted)}$$

In the general case, there are $N-1$ comparisons: the pivot vs every other element (a detail: plus 2 more for handling the “pointers cross” test to end the loop). The pivot ends up in some position $1 \leq k \leq N$, leaving two subproblems of size $k-1$ and $N-k$.

$$C_N = N + 1 + \frac{1}{N} \sum_{1 \leq k \leq N} (C_{k-1} + C_{N-k}) \quad \text{for } N \geq 2,$$

$I/N$ because all values $1 \leq k \leq N$ for pivot are equally likely.

\[ C_N = N + 1 + \frac{1}{N} \sum_{1 \leq k \leq N} (C_{k-1} + C_{N-k}) \quad \text{for } N \geq 2, \]

\[ C_N = N + 1 + \frac{2}{N} \sum_{1 \leq k \leq N} C_{k-1}. \]

\[ NC_N - (N - 1)C_{N-1} = N(N + 1) - (N - 1)N + 2C_{N-1}. \]

\[ NC_N = (N + 1)C_{N-1} + 2N. \]
\[ NC_N = (N + 1)C_{N-1} + 2N. \]

\[
\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1}
\]

\[ = \frac{C_{N-2}}{N-1} + \frac{2}{N} + \frac{2}{N+1} \]

\[ = \vdots \]

\[ = \frac{C_2}{3} + \sum_{3 \leq k \leq N} \frac{2}{k+1}. \]

\[ \frac{C_N}{N+1} \approx 2 \sum_{1 \leq k < N} \frac{1}{k} \approx 2 \int_1^N \frac{1}{x} \, dx = 2 \ln N, \]

\[ 2N \ln N \approx 1.39N \log N \]
So, average run time, averaging over randomly ordered inputs, = $\Theta(n \log n)$.

\textbf{A worst case input is still worst case: $n^2$ every time}

(Is real data random?)

Is it possible to improve the worst case?
another idea: randomize the algorithm

Algorithm as before, except pivot is a *randomly selected* element of $A[1]...A[n]$ (at top level; $A[i]..A[j]$ for subproblem $i..j$)

Analysis is the same, but conclusion is different:

On *any* fixed input, average run time is $n \log n$, 
*averaged over repeated (random) runs of the algorithm.*

There are no longer any “bad inputs”, just “bad (random) choices.” Fortunately, such choices are improbable!
Summary

Average Case Analysis:

1. for algorithm A, choose a sample space S and probability distribution P from which inputs are drawn

2. for \( x \in S \), let \( T(x) \) be the time taken by A on input \( x \)

3. calculate, as a function of the “size,” \( n \), of inputs,
   \[ \sum_{x \in S} T(x) \cdot P(x) \]
   which is the expected or average run time of A

For sorting, distrib is usually “all n! permutations equiprobable”

Insertion sort: \( E[\text{time}] \propto E[\text{inversions}] = \binom{n}{2}/2 = \Theta(n^2) \), about half the worst case

Quicksort: \( E[\text{time}] = \Theta(n \log n) \) vs \( \Theta(n^2) \) in worst case; fun with recurrences, sums & integrals
Randomized Algorithms:

1. for a randomized algorithm $A$, input $x$ is fixed, just as usual, from some space $I$ of possible inputs, but the algorithm may draw (and use) random samples $y = (y_1, y_2, \ldots)$ from a given sample space $S$ and probability distribution $P$.

2. for any $x \in I$ and any $y \in S$, let $T(x, y)$ be the time taken by $A$ on input $x$ when $y$ is sampled from $S$.

3. calculate, as a function of the “size,” $n$, of inputs,
   $$\max_{x \in I} \sum_{y \in S} T(x, y) \cdot P(y)$$
   which is the expected or average run time of $A$ on a worst-case input.

Randomized Quicksort: choosing pivots at random, $E[\text{time}] = \Theta(n \log n)$ for any input. (For every input, there are some rare random choice sequences causing $n^2$ time.)
Worst-case analysis is much more common than average-case analysis because it’s often easier to get meaningful average case results, a reasonable probability model for “typical inputs” is critical, but may be unavailable, or difficult to analyze. As with insertion sort, the results are often similar. But in some important examples, such as quicksort, average-case is sharply better. Randomized algorithms are very important in many areas; sometimes easier to argue that bad stuff is rare than to deterministically circumvent it. (Fascinating open problem: is this intrinsic?)