Lecture 17: Analyzing sorting algorithms

- Time complexity of insertion & selection sort
- Divide & conquer sorting: Mergesort & Quicksort

Announcements:
- Solutions posted
**Insertion Sort:**

- **A** iterations of the loop:
  - find position for \( x \) in sorted part
  - time proportional to size of sorted portion
  - move everything after \( x \) to the right

\( n \) worst case

\[
\frac{1}{2} + 2 + 3 + 4 + \cdots + n = \frac{n(n+1)}{2}
\]

is \( O(n^2) \)

**Selection Sort:**

- A sorted smallest elt of \( a \)

\( i \) happens if array is reverse-sorted to start.
Selection sort:

\[ i = 0 \rightarrow n \]

\[ \text{smallest} \rightarrow ? \]

\[ n \text{ iterations of loop,} \]

\[ \text{in each iteration, find min of } j \text{ have to examine every elt of } \]

\[ \text{unsorted portion.} \]

\[ \text{always takes } n-i \text{ steps.} \]

\[ \text{swap } j \text{ constant time} \]

1\text{st iter: } n-1 \text{ steps}

2\text{nd iter: } n-2 \text{ steps}

\vdots

n\text{th iter: } 0 \text{ steps}

\[ \frac{(n-1)+(n-2)+\cdots+2+1+0}{2} = \frac{n(n-1)}{2} = \frac{n^2}{2} \]

is \( O(n^2) \) steps.

\[ \text{for } (i=0; i<n; i++) \]

\[ \text{for } (j=0; j<n; j++) \]

\[ \text{for } (k=0; k<n; k++) \]

\[ x = x+1 \]

\[ \text{\( n^2 \) steps} \]
Divide & Conquer:

take a big problem, split into smaller problems

Solve subproblems, put solutions together.

Sort:

start
?

end
?

\[ n/2 \]

Walk through both arrays, copying smallest \( a_0 \) into big array.

Recursion:

* sorts elements

\[
\text{mergeSort}(a, \text{start}, \text{end}) = \begin{cases}
    \text{mergeSort}(a, \text{start}, \text{mid}) & \text{if } \text{end} - \text{start} < 2 \\
    \text{mergeSort}((a, \text{start}, \text{mid}), (a, \text{mid}, \text{end})) & \text{otherwise}
\end{cases}
\]

base case: sort array of size 0, 1 or 2
$\log n$ layers, each layer is half size of previous layer.

Total: $O(n \log n)$ steps

better than $O(n^2)$
Quick sort

0. Partition
1. Recursively sort
2. Profit

(linear time)
(need to rearrange to partition)

worst case for quick sort:

O(n^2) = n + (n-1) + (n-2) + \ldots + 0

randomly select a pivot, on average you get close to \( \log n \) performance.