Announcements

- HW 1 up, due in class next Friday
  (may still work in groups)

- Thursday office hours next week may move - stay tuned...
Recursion: Quicksort

Downside of MergeSort - space!
Hard to do in place.
(Also harder to code...)

Simpler divide + conquer: Quicksort

Idea? Select a "pivot"
Pseudocode

\textbf{QuickSort}(A[1..n]):
\begin{align*}
\text{if } (n > 1) \\
&\text{Choose a pivot element } A[p] \\
&k \leftarrow \text{Partition}(A, p) \\
&\text{QuickSort}(A[1..k - 1]) \\
&\text{QuickSort}(A[k + 1..n])
\end{align*}

But how to partition?
\textbf{Partition}(A[1..n], p):
if (p \neq n)
    swap A[p] \leftarrow A[n]

i \leftarrow 0; \quad j \leftarrow n
while (i < j)
    repeat i \leftarrow i + 1 until (i = j or A[i] \geq A[n])
    repeat j \leftarrow j - 1 until (i = j or A[j] \leq A[n])
    if (i < j)
        swap A[i] \leftarrow A[j]

if (i \neq n)
    swap A[i] \leftarrow A[n]
return i

\Rightarrow \Theta(n)
Proof: (sketch)

Very similar to mergesort:

* First show partition works given any array as input + any p.
* Then use induction on entire array.
Analysis:

Depends on choice of pivot:

\[ Q(n) = O(n) + Q(k-1) + Q(n-k) \]

Worst case:

\[ Q(n) = O(n) + O(n) + O(n-1) \]

\[ = n + (n-1) + \ldots + 1 \]

\[ = O(n^2) \]
A recursive strategy: backtracking

Idea: Build up a solution iteratively.

Setting: an algorithm needs to try multiple possibilities.

Strategy: make a recursive call on each possibility.

Downside: slow
Ex: Subset Sum

Given a set $X$ of positive integers and a target $t$, is there a subset of $X$ which sums to $t$?

Ex: $X = \{8, 6, 7, 5, 3, 10, 9\}$

$t = 15$

Yes

$\{8, 7\}$

$\{10, 5\}$

$\{6, 9\}$

Ex: $t = 20$

Yes

$\{8, 7, 5\}$
How could we look at things incrementally (or recursively)?

Set up: take an item \( x \in X \).

Two possibilities:
- \( x \) is in subset
- \( x \) is not

- \[ \exists y \in \mathbb{Y} \]

\[ t \]

- \[ \exists y \in \mathbb{Y} \]

\[ t - x \]
Careful —
that is the recursive case!

What is missing?
(i.e. when are we done?)

- If $X$ is empty, can't hit any $t \geq 0$
- If $t < 0$, done
Pseudo code

\textbf{SubsetSum}(X[1..n], T):
    if \(T = 0\)
        return \textbf{TRUE}
    else if \(T < 0\) or \(n = 0\)
        return \textbf{FALSE}
    else
        return (\textbf{SubsetSum}(X[2..n], T) \lor \textbf{SubsetSum}(X[2..n], T - X[1]))

(tail recursion)
Correctness

**IS:** Either \( X[i] \) is in subset or not (if subset summing to \( T \) exists).

My code tries both possibilities.

\[ \rightarrow \text{Base case: } T = 0 \Rightarrow \text{true} \]
\[ \phi \text{ sums to 0} \]
\[ T < 0 \Rightarrow \text{no set sums to } T \]
\[ X \text{ is empty - can't hit positive} \]
Runtime:

\[ S(n) = 5 + 2S(n-1) \]

\[ s_n = 2s_n + 5 \]

\[ x - 2 = 0 \quad \text{poly of degree 0} \]

\[ S(n) = c_1 2^n + c_2 \]

(use base cases to check \( c_1 \neq 0 \))

\[ c_1 2^0 + c_2 = 1 \]

\[ c_1 2^1 + c_2 = 5 \]

\[ S(n) = \Theta(2^n) \]
Side note: brute force
- try every subset ($2^n$)
- for each, sum values and check = $T$
  $O(n)$

$\implies O(n \cdot 2^n)$