CS 331, Fall 2024

Lecture 2 (8/28)

Today:
- Recursion trees
- Merge Sort
- Selection

Recursion trees (Part II, Section 3)

Aside

Geometric series has form

common ratio

$$a_0 + a_0 r + a_0 r^2 + \dots + a_0 r^k = a_0 \sum_{i=0}^{k} r^i$$

first term

Series evaluates to... (see Lemma 5, Part I)

$$a_0 \sum_{i=0}^{k} r^i = a_0 \frac{r^{k+1} - 1}{r - 1} = \Theta(a_0)$$

"first term dominates"

$r < 1$

$$a_0 \frac{r^{k+1} - 1}{r - 1} = \Theta(a_0 r^k)$$

"last term dominates"

$r > 1$

$$a_0 (k+1)$$

"balanced"

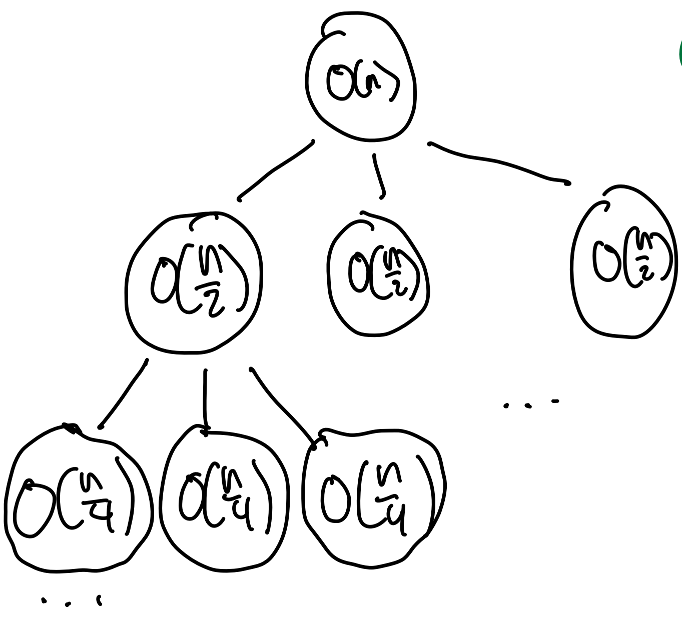
$r = 1$

Idea: write runtime of recursive algo as geometric sequence.

Standard recurrence: $T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$

a: # of subproblems; $\frac{n}{b}$: subproblem size; $f(n)$: base cost

$\boxed{\text{Example}}$ $T(n) = 3T\left(\frac{n}{2}\right) + O(n)$ (multiplication)



Level 0 — Cost: $O(n)$

Level 1 — $O\left(\frac{3n}{2}\right)$

Level 2 — $O\left(\frac{9n}{4}\right)$

Level k — $k = \lceil \log_2(n) \rceil$

Note: there are technically "rounding errors",

we should write $T(n) = a' T\left(\left\lfloor \frac{n}{b} \right\rfloor\right) + (a - a') T\left(\left\lceil \frac{n}{b} \right\rceil\right) \ldots$

e.g. can't split odd-sized lists exactly in half.

OK to ignore in this course, see lecture notes.

General cost of recursion tree

$\boxed{\text{Example}}$ $f(n)=n^c$

level 0	$O(f(n)) \times 1$	$O(n^c)$
level 1	$O(f(\frac{n}{b})) \times a$	$O(n^c)\cdot\frac{a}{b^c}$
level 2	$O(f(\frac{n}{b^2})) \times a^2$	$O(n^c)\cdot\frac{a^2}{b^{2c}}$
$\vdots$		$\vdots$
level k	$O(f(\frac{n}{b^k})) \times a^k$	$O(n^c)\cdot\frac{a^{\log_b(n)}}{n^c}$
$k \approx \log_b(n)$	$O(1) \times n^{\log_b(a)}$	$= O(n^{\log_b(a)})$

In a geometric sequence, first or last term wins.

$\boxed{\text{Aside}}$ We claim $a^{\log_b(n)} = n^{\log_b(a)}$

Proof: $\log(LHS) = \log(a)\cdot\log_b(n)$

$$= \log(a)\cdot\frac{\log(n)}{\log(b)} = \log(n)\cdot\frac{\log(a)}{\log(b)} = \log(RHS)$$

In this class, $f(n)$ almost always of the form

$$f(n) = \Theta\left(n^c \log^d(n)\right) \text{ for } \begin{matrix} c \geq 1 \\ d \geq 0 \end{matrix}.$$

In that case, we can use the...

$\boxed{\text{Master Theorem}}$

Suppose recurrence looks like

$$T(n) = a\,T\left(\frac{n}{b}\right) + \Theta\big(\overbrace{n^c \log^d(n)}^{f(n)}\big)$$

$$\left(\text{common ratio: } \frac{a}{b^c}\right)$$

"root-heavy"

$c > \log_b(a)$: $T(n) = \Theta(f(n))$

"leaves-heavy"

$c < \log_b(a)$: $T(n) = \Theta\left(n^{\log_b(a)}\right)$

"balanced"

$c = \log_b(a)$: $T(n) = \Theta(f(n)\log(n))$

... in general, even if Master Thm. doesn't apply, can still use recursion tree to induce a geometric sequence.

Merge Sort (Part II, Section 6.1)

The most famous recurrence in algos:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n).$$

Balanced case: $T(n) = O(n\log(n))$

e.g. Mergesort(18, 1, 7, 6, 45, 3, 9, 0)

(1) Sort two halves recursively

$$(1\ 6\ 7\ 18)(0\ 3\ 9\ 45)$$

first half: $T\left(\frac{n}{2}\right)$ second half: $T\left(\frac{n}{2}\right)$

(2) Stitch two halves together

$$(1\ 6\ 7\ 18)(0\ 3\ 9\ 45)$$

$$(0\ 1\ 3\ 6\ \ldots)$$

We can implement ② in $O(n)$ time.

Invariant: Smallest element remaining is either smallest in first half, or smallest in second half.

Why? Recursion fairy sorted the halves.
Just keep peeling off the smallest element.

MergeSort(L):

$n \leftarrow |L|$

If $n == 1$: return L

Else:

$L_1 \leftarrow L[1 : \lceil \frac{n}{2} \rceil]$, $L_2 \leftarrow [\lceil \frac{n}{2} \rceil + 1 : n]$

$L_1 \leftarrow$ MergeSort(L_1) Runtime: $T(\frac{n}{2})$

$L_2 \leftarrow$ MergeSort(L_2) Runtime: $T(\frac{n}{2})$

$i_1 \leftarrow 1$, $i_2 \leftarrow 1$ // pointers to current smallest elements

For $i \in [n]$:

Runtime: $O(n)$

If $L_1[i_1] \leq L_2[i_2]$: $L[i] \leftarrow L_1[i_1]$, $i_1 \leftarrow i_1 + 1$

Else: $L[i] \leftarrow L_2[i_2]$, $i_2 \leftarrow i_2 + 1$

Selection (Part II, Section 6.2)

Input: L is a list of n elements in $\mathbb{R}$

i is an index $1 \leq i \leq n$

Output: i^{th} largest element in L.

Example

Selection([10, 7, 16, 3, 2, 80, 1], 4)

= 7

Selection([-500, -30, -7, 2, 50, 70, 100], 4)

= 2 easier because sorted!!!

Observation 1: $O(n \log(n))$ time algo.

Proof: Sort, take i^{th} largest element

Observation 2: $O(n)$ if i is small ($i = O(1)$)

Proof: Compute minimum in $O(n)$ time (just store it). Repeat i times.

Main claim: We can solve selection in

$O(n)$ time, for all $i \in [n]$.

Application: Median in linear time.

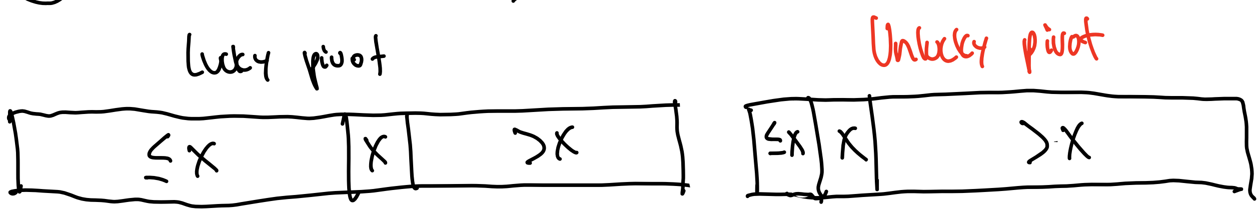
Application: Deterministic QuickSort.

Aside Quicksort is a simple sorting algo.

(1) $x \leftarrow \text{findPivot}(L)$

Easiest choice: random element

(2) $L \leftarrow \text{Pivot}(L, x)$ $O(n)$ time.



(3) Quicksort two halves recursively.

If we're lucky, x is in the middle, recursion depth $O(\log(n))$

If we're unlucky, x is near the edges, recursion depth $O(n)$

How to use pivoting to solve Selection?

Idea: avoid unlucky pivots using recursion.

```
Selection(L, i):
    n ← |L|
    If n == 1: return L[i]
    Else:
①       x ← FindPivot(L)                 // TBD.
②       (k, L) ← Pivot(L, x)             // L is pivoted around x.
                                          //   x = L[k].
        If k == i: Return x
        Else If k > i:                   // Search left half
③           Return Selection(L[1:k-1], i)
        Else:                            // search right half
③           Return Selection(L[k+1:n], i-k)
```

Total cost: $T(n) = P(n) + O(n) + T(??)$

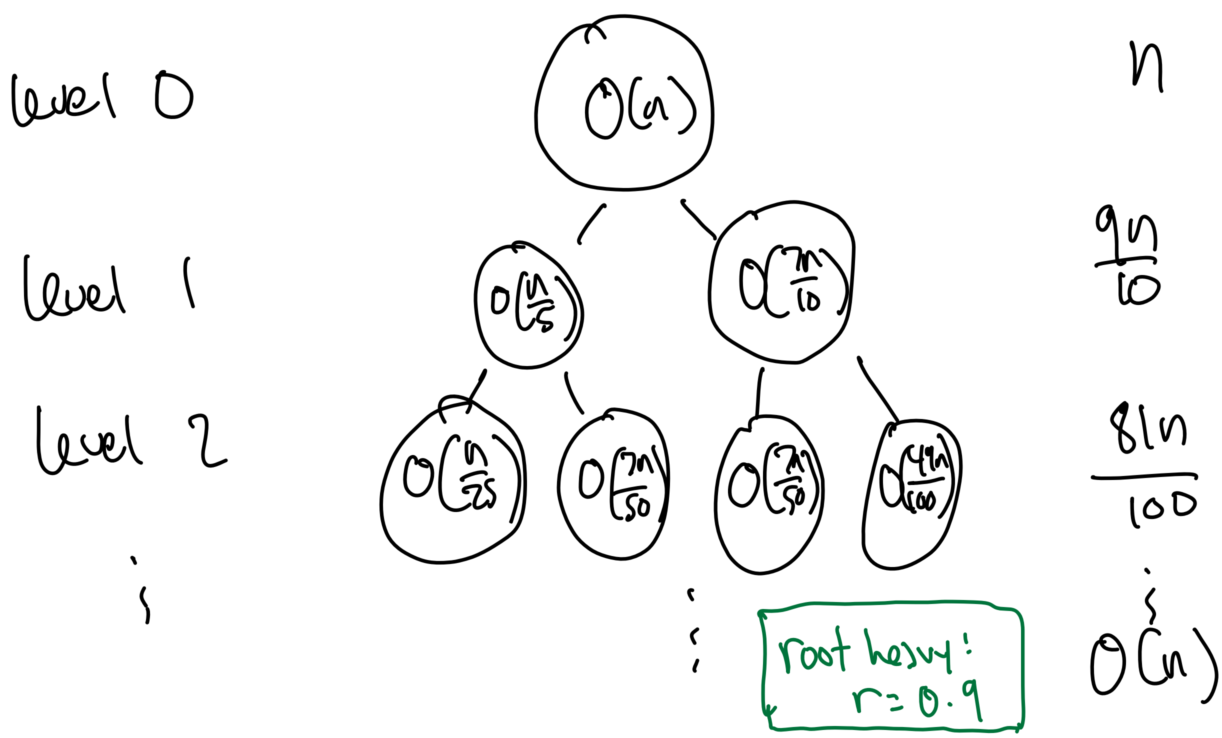
① Cost of FindPivot ② ③ Cost of recursion

Key claim: there is FindPivot implementation w/

- $P(n) \le T\left(\frac{n}{5}\right) + O(n)$.
- ?? $\le \frac{7n}{10}$. (not too unlucky)

Finish runtime analysis:

Total cost: $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$



root heavy!
$r = 0.9$

Takeaway: Selection (Li:) in $O(n)$ time

:)

Proof of Key Claim

"Median of medians" pivot. — Cost

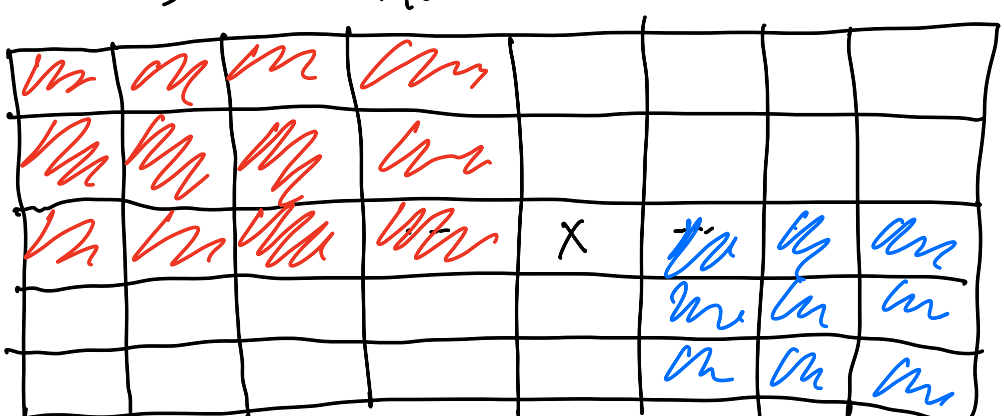
- Split into blocks of size ≤ 5. — $O(n)$
- Compute median of each block — $O(n)$
- Compute median of block medians (selection problem on $\frac{n}{5}$ elements) — $T\left(\frac{n}{5}\right)$

Magical fact: Median of medians index is

$$\in [0.3n, \ 0.7n]$$

$$3/5 \times 1/2 = 3/10.$$

blocks of 5



[red scribble] means $\leq x$, [blue scribble] means $> x$