CS 170

Asymptotics, Divide-and-Conquer Review Session

ASYMPTOTICS

2 commonly used definitions

1. $f(n) = O(g(n))$ if, for $n \to \infty$, $f(n) \le c \cdot g(n)$, for a constant c.

2. If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = O(g(n))$

(not necessary, but sufficient)

big - O : asymptotically $\le$

big - Ω : asymptotically $\ge$

big - Θ : asymptotically $=$

- $O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(2^n) \subset O(3^n) \subset O(n^n)$
- All logarithms are $\Theta(\log n)$ by change of base rule (differ by only a constant factor).
- Asymptotic notation only cares about "highest-growing" terms. For example, $n^2 + n = \Theta(n^2)$.
- Asymptotic notation does not care about leading constants.
- Any exponential with base > 1 grows faster than any polynomial. For example, $n^{100} = O(1.01^n)$, but $n^{100} \neq \Omega(1.01^n)$.
- The base of the exponential matters. For example, $3^n = O(4^n)$, but $3^n \neq \Omega(4^n)$.
- If $f_1 = O(g_1(n))$, $f_2 = O(g_2(n))$, then
  - $f_1 f_2 = O(g_1 g_2)$.
  - $f_1 + f_2 = O(\max\{g_1, g_2\})$.

DIVIDE-AND-CONQUER

Divide-and-conquer problems generally follow the following paradigm:

1. Split the problem up into smaller parts
2. Make $\overset{\downarrow}{a}$ recursive calls to problems of size n/b
3. "glue" the subproblems together to provide solution of original problem.

This formulation lends itself to the canonical recurrence relation:

$$T(n) = a\,T(n/b) + O(n^d)$$

$a \leftarrow$ branching factor

$b \leftarrow$ factor by which subproblem size is reduced

$d \leftarrow$ work done at each subproblem.

MASTER THEOREM

$$T(n) = a\,T(n/b) + O(n^d)$$

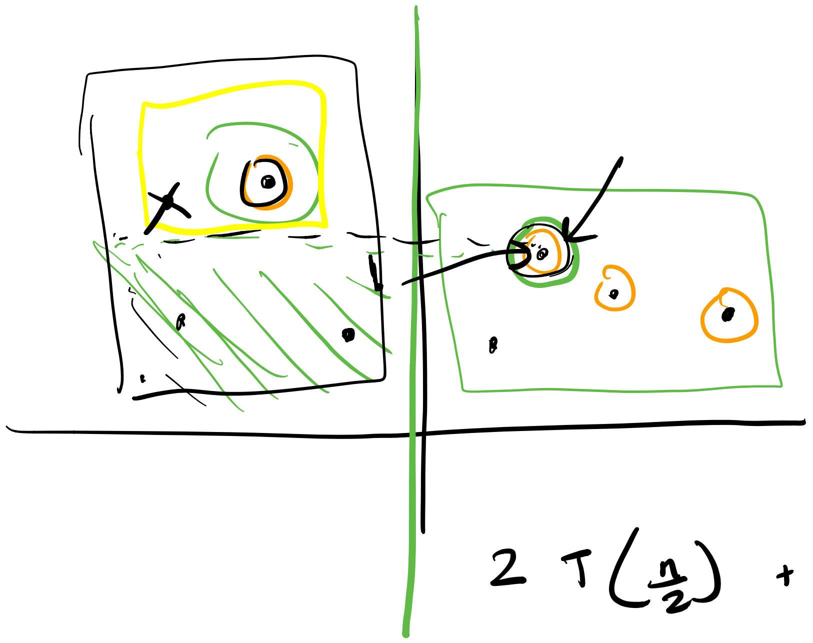
- If $\frac{a}{b^d} < 1$ $(d > \log_b a)$, then $\underline{T(n)} = O(n^d)$
  (root heavy)
- If $\frac{a}{b^d} > 1$ $(d < \log_b a)$, then $T(n) = O(n^{\log_b a})$
  (leaf heavy)
- If $\frac{a}{b^d} = 1$ $(d = \log_b a)$, then $T(n) = O(n^d \log n)$
  (balanced)

PRACTICE PROBLEMS

1 Midterm Prep: Divide and Conquer

Given a set of points $P = \{(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)\}$, a point $(x_i, y_i) \in P$ is Pareto-optimal if there does not exist any $j \neq i$ such that such that $x_j > x_i$ and $y_j > y_i$. In other words, there is no point in P above and to the right of (x_i, y_i). Design a $O(n \log n)$-time divide-and-conquer algorithm that given P, outputs all Pareto-optimal points in P.

(Hint: Split the array by x-coordinate. Show that all points returned by one of the two recursive calls is Pareto-optimal, and that you can get rid of all non-Pareto-optimal points in the other recursive call in linear time).



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

$\Theta(n \log n)$

Extreme Poopulicions

SP19 MT1, Q10

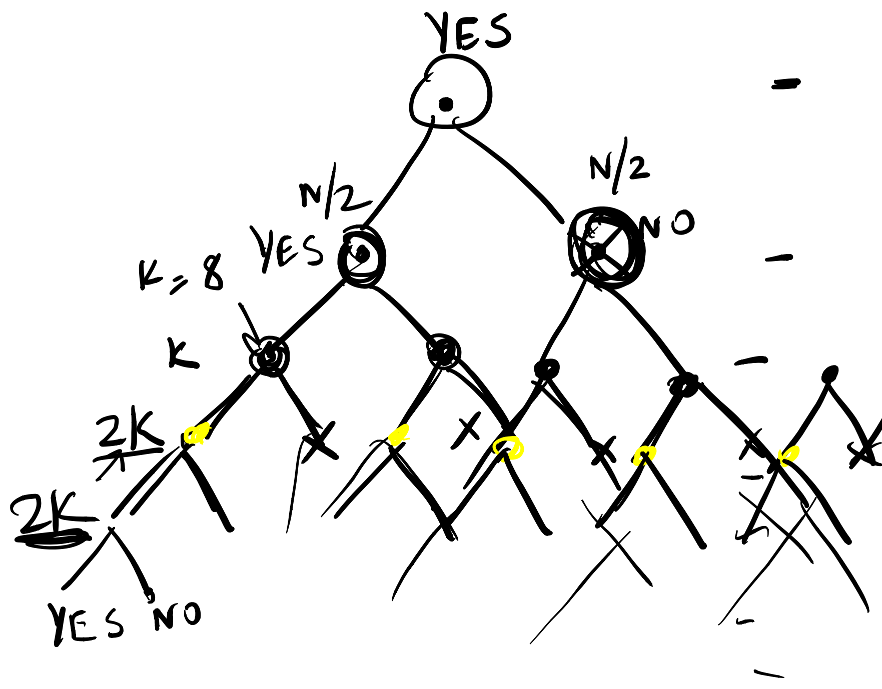
10. **Testing efficiently (8 points)**

At most k people among a population of n people in a city have been afflicted by a deadly virus. The city council needs to identify the k people infected so as to treat them.

The infection can be detected by testing the patient's blood for the virus. Unfortunately, each test is very expensive and the city council would like to minimize the number of tests carried out.

One can mix the blood samples from a subset of people, and test at one shot if at least one in the subset is infected. Specifically, for any subset S of people, $test(S : subset)$ carries out one blood test and returns YES if at least one in S is infected and NO otherwise.

Describe an algorithm that uses $O(k \log n)$ tests to algorithm to find all infected people in the city. (try to use 6 sentences or less)



log n levels

O(K) work per level

$T(n) =$

FORMER DISCUSSION

3 Find the valley

You are given an array A of integers of length N. A has the following property: it is decreasing until element j, at which point it is increasing. In other words, there is some j such that if $i < j$ we have $A[i] > A[i+1]$ and if $i \geq j$ we have $A[i] < A[i+1]$. Assuming you do not already know j, give an algorithm to find j.

For simplicity, you may assume that N is a power of 2.

$$\left\lfloor \frac{N}{2} \right\rfloor, \left\lfloor \frac{N}{2} \right\rfloor + 1$$

$$\underline{\left\lfloor \frac{N}{2} \right\rfloor} < \underline{\left\lfloor \frac{N}{2} \right\rfloor + 1} \ \Big|$$

$$T(n) = T\left(\frac{n}{2}\right) + O(1) \ O(n^0)$$

$$d \qquad \log_b a$$

$$d = 0 \qquad \log_2 1 = 0$$

$$n^d \log n = \log n$$

FA19 MT 1

1. Given k sorted lists, each of length n, give an algorithm to produce a single sorted list containing the elements of all the lists. The runtime should be $O(kn \log k)$.