Announcements

* HW 1 due on Thursday (Aug 28)
* Office Hours in Klaus 2108 (see Canvas or Piazza for Google Calender)
* Exam 1 on Sep 18 (instead of Sep 16)
* Recordings on Canvas, Handwritten on Piazza

Plan for today: Complete Median finding, Dynamic Prog

Median Finding

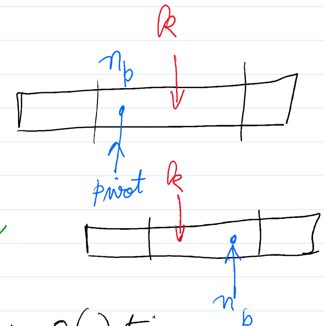
Given n numbers $A[0, \ldots n-1]$. ← Say distinct for simplicity

Given $k \in \{1, 2, \ldots, n\}$

Goal: Find the k-th largest number ← $(k-1)$ smaller #s & $n-k$ large #s

Sorting takes $\Theta(n \log n)$ time. Can we do it faster?

Thm: We can achieve this in $O(n)$ time.
Blum, Pratt, Rivett, Tarjan - 1972



Idea: (1) Find a $\underbrace{\text{"good pivot"}}_{\geq \frac{3n}{10} \text{ elements larger } \& \geq \frac{3n}{10} \text{ elems smaller}}$ recursively. ← $T\left(\frac{n}{5}\right)$

(2) Argue that we can drop one of sides of the pivot in $O(n)$ time
Pf: Calculate $\underbrace{\text{how many elems}}_{n_p}$ are less than pivot p

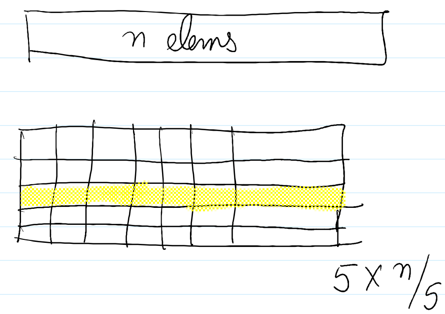
Case 1: $n_p > k$
Drop all $e > p$ & find k-th larg in rem

Case 2: $n_p \leq k$
Drop all $e < p$ & find $(k - n_p)$ th largest in remain

(3) Get Recursion: $T(n) \leq \underbrace{T\left(\frac{7n}{10}\right)}_{n - \frac{3n}{10} \text{ remaining}} + \underbrace{T\left(\frac{n}{5}\right)}_{\text{Finding Pivot}} + \underbrace{O(n)}_{\text{Merge operation}}$

$n - \frac{3n}{10}$ remaining elms   Finding pivot   Merge operation

Exercise: Prove this recursion gives $T(n) = O(n)$.



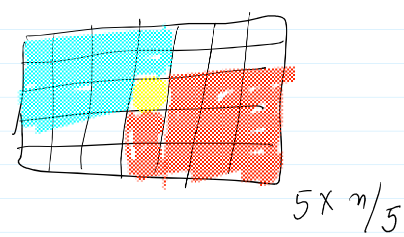
$O(n)$ Step 1: Construct $5 * \frac{n}{5}$ array

$O(n)$ Step 2: Sort each colm of this array

$T(\frac{n}{5})$ Step 3: Recursively find the median of the column-medians.

Claim: This returned median of medians is a good pivot.

\# elms < median of median is at least $\geq \frac{n}{10} \times 3 = 30\%\, n$



Takeaways on D & C

(1) Break pb into smaller subproblems

(2) Recursively solve each subproblem

(3) Combine all subprob solns to obtain soln to orig prob

Examples: Sorting, Max-difference, Median

Dynamic Programming

Idea: Solve large pbs by using solns to smaller pbs.

(1) Define subproblem indexed $i, j, \ldots$ ← Similar to Induction Hypothesis

(2) Solve the subproblems & store its soln in a array $M[i, j, \ldots]$

(3) Use soln of subproblems to solve current problem   ... integer - tuple...

(2) Solve the subproblems ... $[\ ,\ , \dots]$

(3) Use soln of subproblems to solve current problem

soln could be integer, tuple, ...

(Similar sounding to D & C but subproblems more interleaved, rather than being disjoint)

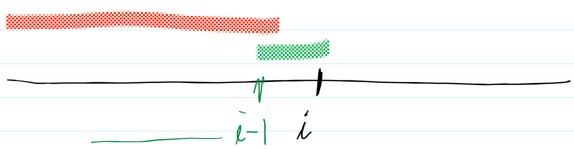
Example 1: Longest Contiguous Sum

n numbers $A[0, \dots, n-1]$ ← could be +ve or −ve

Find a subinterval $\{x, x+1, \dots, y\}$ to $\max \sum_{i=x}^{y} A[i]$

E.g., $5, 15, -30, \underbrace{10, -5, 40, 10}_{55}$

Define: Let $M[i]$ = Sum of longest contig in $A[0, \dots i]$ containing $A[i]$.



Plan: (1) Fill array $M[\,]$ by induction

(2) Given array $M[\,]$, find the soln $= \max_i \{M[i], 0\}$ ← $\Theta(n)$ time

Claim: We can compute $M[\,]$ in $\Theta(n)$ time ⇒ overall $\Theta(n)$ runtime

Pf by induction:

I.H. is that we have filled $M[0, \dots i]$ correctly

Base Case: $M[0] = A[0]$

Ind Step: $M[i] = \max\{A[i],\ A[i] + M[i-1]\}$

↑ $O(1)$ time

⇒ $O(n)$ time to compute $M[\,]$.

Extending to finding Longest Contig Seq

- Redefine $M[i]$ to store not only sum of LCS
  but also whether $M[i]$ took $M[i-1]$ or not.
- In the end, when computing $\max_i \{M[i], 0\}$, trace back the soln.

Example 2: Longest Common Subseq

Given 2 arrays $A[0, \ldots n-1]$
$B[0, \ldots, n-1]$

Find subsets of indices X, Y with $|X| = |Y|$
and $A[X_i] = B[Y_i]$

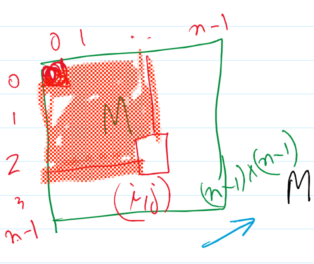
E.g: A B C D E F G $\quad X = \{1, 2, 3, 6\}$

B C D G K A B $\quad Y = \{0, 1, 2, 3\}$

Define: $M[i,j]$ = Length of LCS using $A[0, \ldots i]$ & $B[0, \ldots j]$

Base Case: $M[0,0] = \begin{cases} 1 & \text{if } A[0] = B[0] \\ 0 & \text{otherwise} \end{cases}$ $\quad M[i,0] = \{ \quad \}$

$M[0,j] = \{ \quad \}$

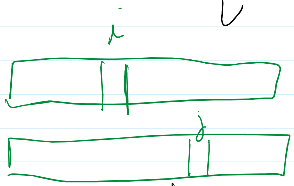


2-d array

$$M[i,j] = \begin{cases} 1 + M[i-1, j-1] & \text{if } A[i] = B[j] \\ \max\{M[i-1,j], M[i,j-1]\} & \text{o.w.} \end{cases}$$

← if we match last numbers

we need to drop at least one of $A[i]$ or $B[j]$.



Time = $\Theta(n^2)$ since $O(1)$ per entry

Time = $\Theta(n^2)$ since $O(1)$ per entry