CSE 630 lecture 8 Aug 21

h.w. assignment problems for submission on Aug 25 (Monday) from Tutorial sheet 1 (not practice probs)

3, 6, 7, 9

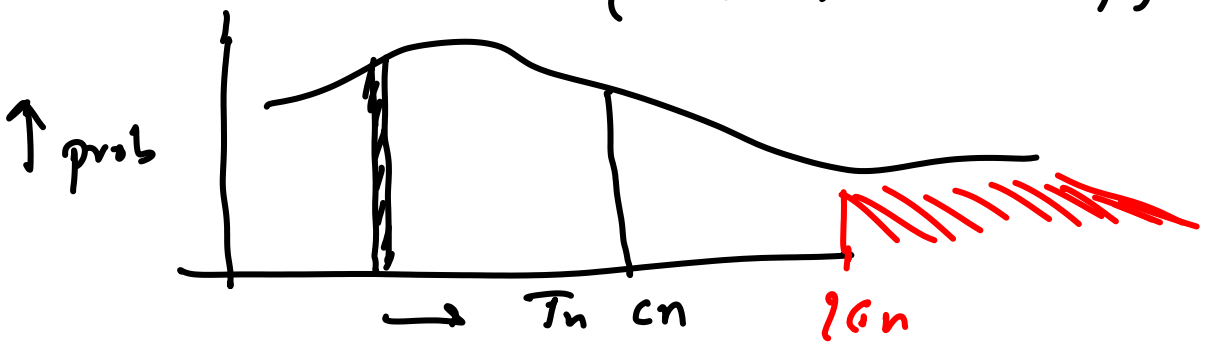
→ What about deterministic algorithms

The previous randomized select algorithms returns the correct element in expected time $O(n)$.

$E[T_n] = c \cdot n$ c is a constant

$\Rightarrow$ $Prob[T_n \geq 2 \cdot (c \cdot n)] \leq \frac{1}{2}$

(Markov ineq)

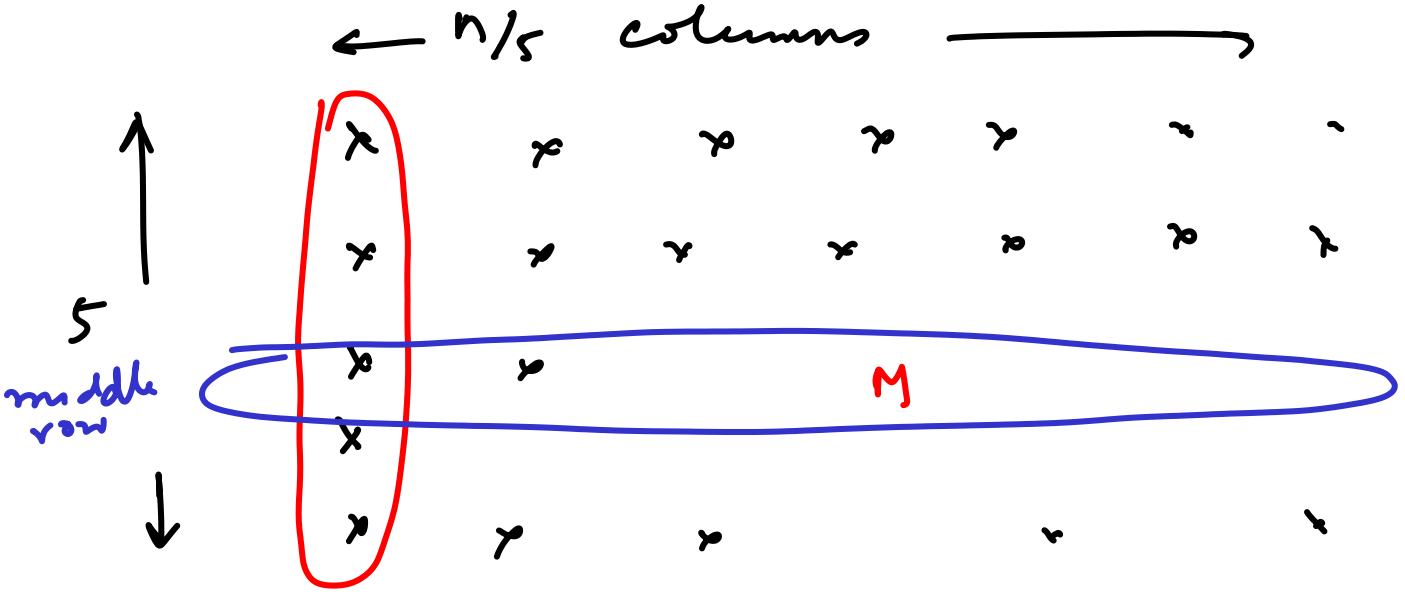


$Pr[T_n \geq \alpha \cdot E[T_n]] \leq \frac{1}{\alpha}$, $\alpha \geq 1$

How do we find a 'good' pivot (splitter)

"Prune and Search"

We group the elements in columns of size 5



We sort the columns : $O(1) \cdot \frac{n}{5} = O(n)$

After sorting consider the middle row of $\frac{n}{5}$ elements. Find M: median of the $\frac{n}{5}$ elements

Observation $\frac{n}{10}\frac{n}{4} \leq \text{Rank}(M,S) \leq \frac{3n}{4}$ ~~$9n/10$~~

Suppose $T(n)$ is the time for selection among n elements (worst over K)

$$T(n) \leq O(n) + T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right)$$

- $O(n)$: initial sorting of columns
- $T\left(\frac{n}{5}\right)$: recursively find median
- $T\left(\frac{3n}{4}\right)$: recursively call on select at most $3n/4$

$T(1) = O(1)$

$T(n) \leq c \cdot n$ c is some constant

→ hw exercise Verify using induction

→ what if we used $T\left(\frac{9n}{10}\right)$ instead of $T\left(\frac{3n}{4}\right)$

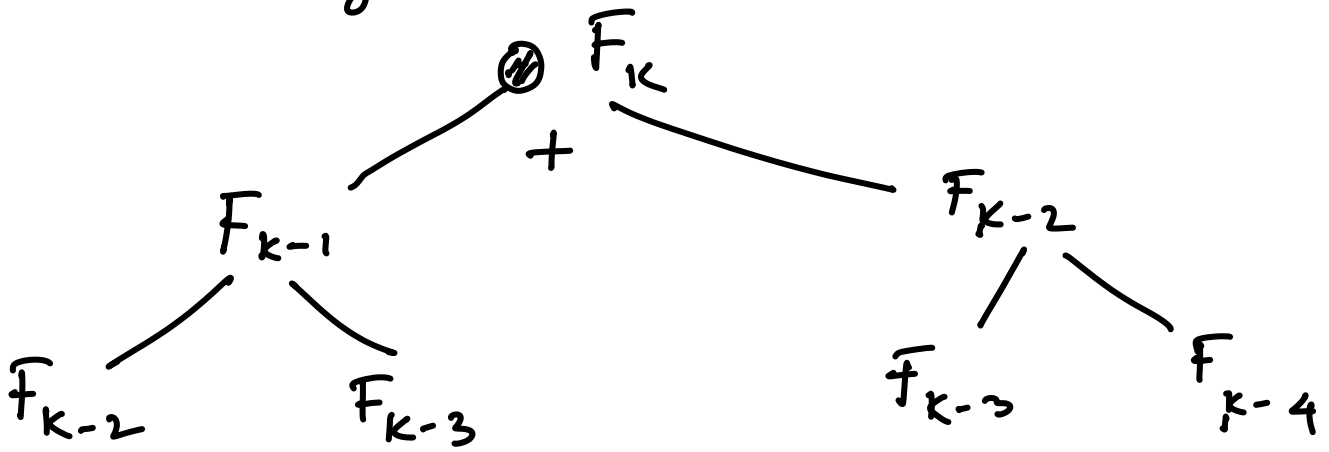
How do we compute Fibonacci numbers

i.e. given k, find F_k

$F_0 = 0 \quad F_1 = 1 \quad F_2 = 1, \quad F_3 = 2 \ldots$

$F_i = F_{i-1} + F_{i-2}$

- Apply the recurrence



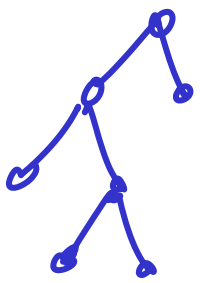
$F_0 \quad F_1 \quad \ldots \quad F_1 \quad F_0$

$T(n)$ = cost of computing F_n

$$T(n) = T(n-1) + T(n-2) + O(1)$$

$T(n)$ is roughly the # internal nodes

In a binary tree (0 or 2 children) # internal nodes = # Leaf nodes − 1



What is the # leaf nodes in the recursion tree for Fibonacci?

"Roughly" - the value of F_K

$$F_n = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right)$$

$$\phi = \frac{1+\sqrt{5}}{2} \qquad \hat{\phi} = 1 - \phi$$

$$\sim 2^n$$

Too large for realistic computation

What is a better way?

Iteratively compute F_K

from $F_0, F_1, F_2, \ldots F_3 \ldots F_K$

Each iteration takes $O(1)$ time

$\Rightarrow$ $O(K)$ operations

Each operation is addition of two Fibonacci nos

$|F_n| \sim n$

# of bits

The last addition will take $\Omega(n)$ -time

Suppose addition takes $O(|x|+|y|)$ time to add $x+y$

Then computing Fibonacci nos will take time $\sum_{i=1}^{n} O(i) = O(n^2)$

Running -time is a function of the input size

Input size for computing F_n

$|n| = \log_2 n$

The lower bound for any algorithm is (input size + output size)

The running time of iterative Fib is $(\text{output} + \text{input size})^2$

$\searrow n$ bits

Can we compute Fibonacci numbers faster?

(Time for extra lecture)

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} F_{n-2} \\ F_{n-3} \end{bmatrix}$$

$$= \cdots \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^{k} \cdot \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$$

$k \sim n$

How quickly can we compute A^k

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

In about $\log n$ matrix multiplications we need to do more detailed analysis of each multiplication because the individual entries could be quite large!