Tuesday, February 22, 2022 10:47 AM

Part II: ALGORITHMS

How to solve a given problem fast / efficiently ?

time (#steps) & space.

RAM Model.

Ex: Sorting

Given n numbers $A[1], A[2], \ldots, A[n]$

reorder s.t. $A[1] \le A[2] \le \ldots \le A[n]$.

e.g., 50, 80, 42, 10 | 99, 75, 35, 25

o/p 10 25, 35, 42, 50, 75, 80, 99

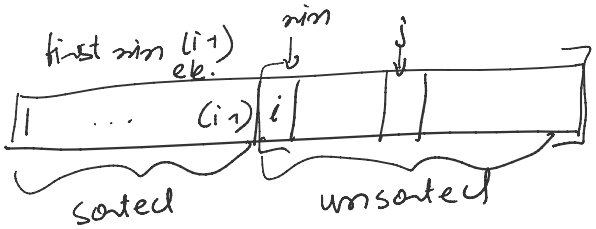
10, 42, 50, 80 | 25, 35, 75, 99

Algorithm I: Selection sort.

idea: select the min/smallest element, "remove", repeat.

1. for $i = 1$ to n.

   // find min $A[i] \ldots A[n]$ & place it at $A[i]$.



2. min = i

[3. for j = (i+1) to n

[4. if $A[j] \le A[min]$ then min = j

5. swap $A[i]$ with $A[min]$.

Hides constant factors.

Running time!

Hides constant factors.

lines 3-4 : $(n-i) * 2 = O(n-i)$

Total time : $O(n-1) + O(n-2) + \cdots + O(1)$

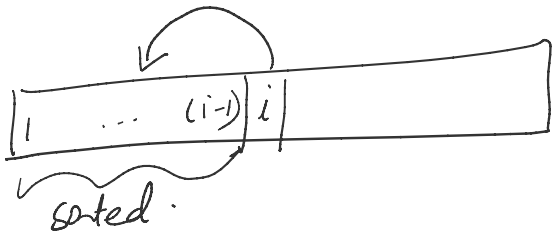
$= O\left(\sum_{i=1}^{n-1} (n-i)\right)$

$= O\left(\frac{n(n-1)}{2}\right) = O\left(\frac{n^2}{2} - \frac{n}{2}\right) = O(n^2)$

Algo 2 : Insertion Sort.

for $i=1$ to n

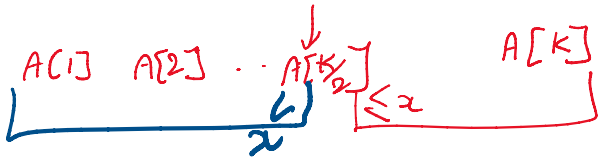
insert $A[i]$ into sorted list $A[1 \ldots (i-1)]$



$O(\log_2 i)$ → 1. find correct position of $A[i]$ in $A[1 \ldots (i-1)]$

$O(i)$ → 2. place it there

Binary search.



$\frac{n}{2^k} = 1$

$k = \log_2 n$

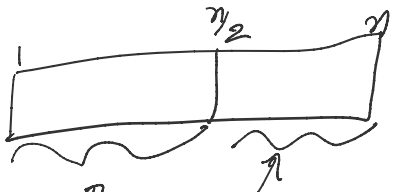
Running time : $O(n^2)$ again!

★ Algo 3 : Mergesort

... recursion.

★ Algo 3 : Mergesort

idea/paradigm : Divide & Conquer, recursion.



Mergesort $(A[1 \dots n])$ {

1. if $n=1$ then return.

$T(\frac{n}{2})$ → 2. Mergesort $(A[1 \dots \lfloor \frac{n}{2} \rfloor])$

$T(\frac{n}{2})$ → 3. Mergesort $(A[\lfloor \frac{n}{2} \rfloor + 1 \dots n])$

$O(n)$ → 4. Merge 2 sorted arrays $A[1 \dots \lfloor \frac{n}{2} \rfloor]$ & $A[\lfloor \frac{n}{2} \rfloor + 1 \dots n]$

$m_1 = m_2 = \frac{n}{2}+1$

Merging $B[1 \dots m_1]$, $C[1 \dots m_2]$ into $D[1 \dots m_1+m_2]$

idea : scan both left to right & keep taking smaller element.

B: 10, 42, 50, 80

C: 25, 35, 75, 99

D[10, 25, 35 ...]

$O(m_1+m_2)$ {

1. $i=1$, $j=1$
2. for $k=1$ to m_1+m_2
3. if $(B[i] \le C[j]$ & $i \le m_1$ OR $j \ge (m_2+1))$ then $D[k] = B[i]$; $i++$;
   else $D[k] = C[j]$; $j++$;
4. }

Runtime :

$T(n)$ = Mergesort on n elements

$$= \begin{cases} c' & \text{if } n=1 \\ 2T\left(\frac{n}{2}\right) + cn & \text{if } n \geq 2 \end{cases}$$

$$= O(n \cdot \log n)$$

How to solve recurrences?

Approach 1: Unrolling

$$T(n) \underset{1}{=} 2T\left(\frac{n}{2}\right) + cn$$

$$= 2\left[2T\left(\frac{n}{4}\right) + \frac{cn}{2}\right] + cn$$

$$\underset{2}{=} 4T\left(\frac{n}{4}\right) + 2cn$$

$$= 4\left[2T\left(\frac{n}{8}\right) + \frac{cn}{4}\right] + 2cn$$

$$\underset{3}{=} 8T\left(\frac{n}{8}\right) + 3cn$$

$$\vdots$$

$$= 2^k T\left(\frac{n}{2^k}\right) + k\,cn$$

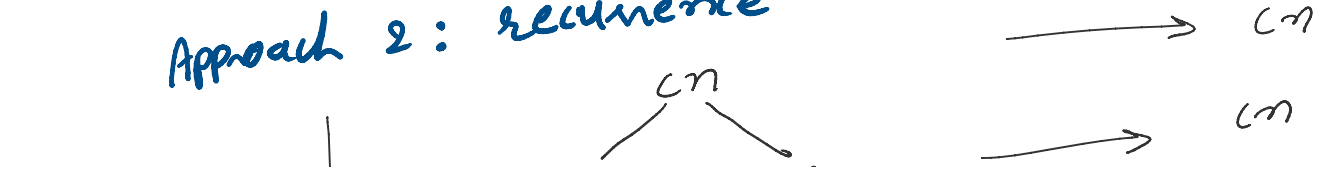
Pick $k = \log_2 n$

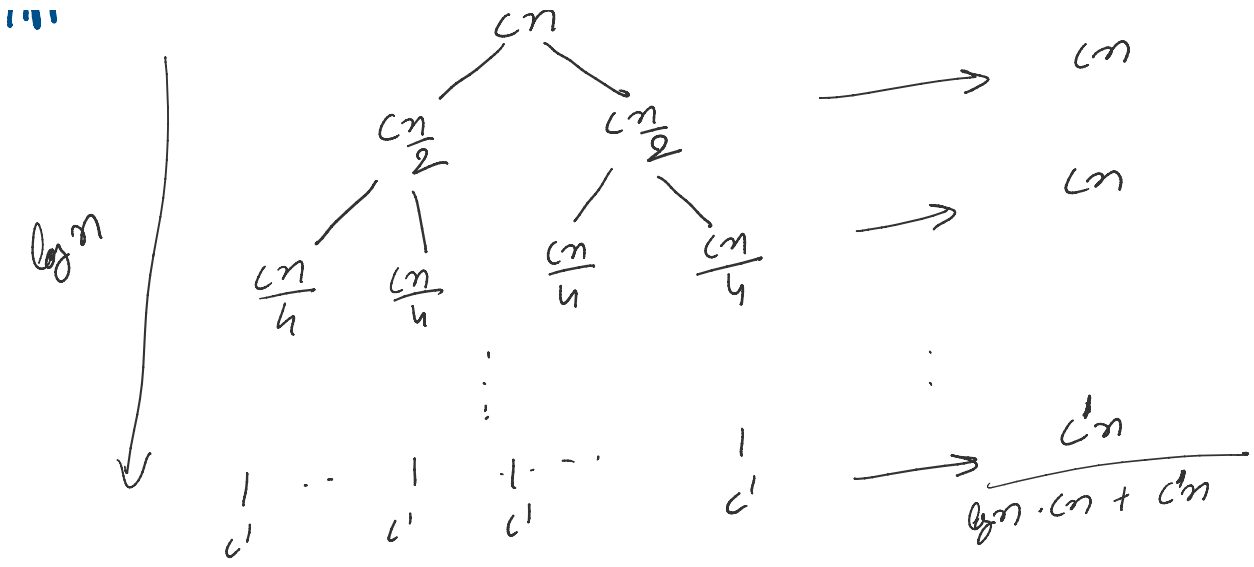
$$= n\,T(1) + \log n \cdot cn$$

$T(1) = c'$

$$= O(n \cdot \log n)$$

Approach 2: recurrence tree





Approach 3: Guess & verify by induction!

Note: Rate of increase is more important than const 3.

eg. $20 n \log n$ vs. $2n^2$

speed = 10^9 ops/sec. $n = 10^6$ ~ 0.45 sec

$2n^2$: 2000 sec. $\sim$ 30 min

eg. $20 n^2$ vs 2^n ← exponential

$n = 100$ $\frac{2}{10^4}$ sec

2^{100}

$\frac{10^{30}}{10^9} = 10^{21}$ secs

$\approx 10^{13}$ years.

Note: - Running time may vary over diff i/ps of size n.

Note: - Running time may vary over diff i/ps of size n.

- We will analyze running time on the worst i/p i.e.

"Worst-case analysis"

Lower bound on running-time for any sorting comparision based is $\Omega(n \cdot \lg n)$

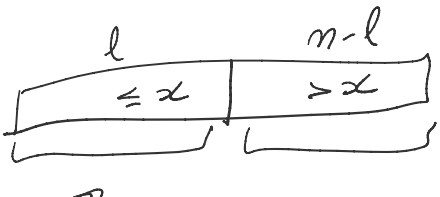
Other sorting Methods: Heapsort (selection sort using smart datastructures)

Divide & conquer Method

Quicksort.

Quicksort $(A[1 \ldots n])$

0. if $n=1$ return
1. Pick a pivot x
2. Partition A into 2 parts



$\{A[i] \mid A[i] \leq x\} \rightarrow A[1 \ldots l]$

$\{A[i] \mid A[i] > x\} \rightarrow A[l+1 \ldots n]$

$O(n)$

$T(l)$ → 3. Quicksort $(A[1 \ldots l])$

$T(n-l)$ → 4. Quicksort $(A[l+1 \ldots n])$

Runtime:

$$T(n) = T(l) + T(n-l) + O(n)$$

ideally we want $l = \frac{n}{2}$

$T(n) = O(n)$

Ideally we want $\ell = \frac{n}{2}$

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

$$= O(n \cdot \log n)$$

Worst-case $\ell = 1$ always

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

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

$$= O(n^2)$$

$c_1 n$

|

$c_2(n-1)$

⋮

c_n

$$cn + c_2(n-1) + \cdots + c_n$$

$$\leq \left(\sum_{i=1}^{n} c_i\right)\left(\frac{n^2}{2}\right) = O(n^2)$$