Tuesday, February 20, 2024 8:22 AM

Part II: ALGORITHMS

"step-by-step procedures designed to solve specific problems …"

CS: Computer Prog. that is efficient (fast)

Efficient: Time (#ops) and/or Space. (#bytes.) RAM Model.

$O(\cdot)$ — U.B. $\quad \Omega(\cdot)$ — L.B.

Ex: Sorting

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

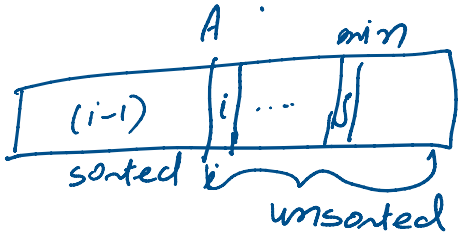
reorder s.t.: $A[1] \le A[2] \le \ldots \le A[n]$

eg.: 50, 80, 45, 18, 96, 35, 75, 25

o/p: 18, 25, 35, 45, 50, 75, 80, 96

Alg'm 1: Selection Sort

idea: Find next smallest number, remove, repeat.



1. for i=1 to n {
   // find ith smallest & put it at A[i].

2. min = i
   … (i+1) to n {

2. min = i

$O(n-i)$ [3. for j = (i+1) to n {

4. if A[j] < A[min] then min = j

}

5. swap A[i] & A[min].

}

Hide constant factors.

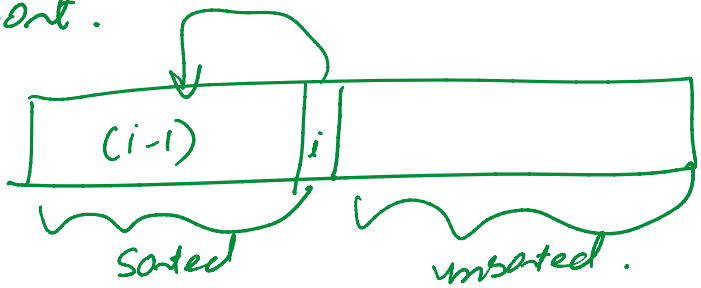
Running Time:

steps 3-4 (i): $O(n-i) = \underbrace{c}_{\text{constant}} \cdot (n-i)$

Total: $O\left(\sum_{i=1}^{n} (n-i)\right) = O\left((n-1) + (n-2) + \cdots + 1\right)$

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

Alg'm 2: Insertion Sort.



1. for i = 1 to n.

// put A[i] in it's "right" position in the sorted subarray A[1 ... (i-1)]

(omit details)

Running Time: $O(n^2)$

Alg'm 3: Merge Sort (von Neumann '40s)

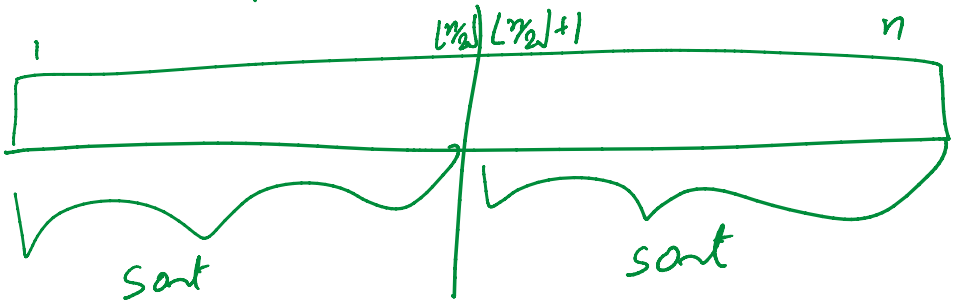
Divide & Conquer.

Recursion.

Divide & conquer.

Recursion.

idea:



Merge

eg.: 50, 80, 45, 18 | 96, 35, 75, 25

sort ↓ sort ↓

~~18~~, ~~45~~, ~~50~~, ~~80~~ | ~~25~~, ~~35~~, ~~75~~, 96

Merge.

18, 25, 35, 45, 50, 75, 80, 96

MergeSort (A[1 .... n])

Time T(n) 1. if n=1 then return.

$T(\lfloor n/2 \rfloor)$ → 2. MergeSort (A[1 .... $\lfloor n/2 \rfloor$])

$T(\lfloor n/2 \rfloor + 1)$ → 3. MergeSort (A[$\lfloor n/2 \rfloor$+1, ..., n])

O(n) → 4. Merge A[1 ..., $\lfloor n/2 \rfloor$], A[$\lfloor n/2 \rfloor$+1 ... n] into sorted array A[1 ... n].

Merge B[1 ... l], C[1 ... k] into

Merge $B[1 \dots \ell]$, $C[1 \dots k]$

$D[1 \dots \ell+k]$

$O(k+\ell)$

1. $i=1$, $j=1$
2. for $d=1$ to $k+\ell$
3. if $B[i] \le C[j]$ then $D[d] = B[i]$, $i++$
   else $D[d] = C[j]$, $j++$

Check ($i \le \ell$ & $j \le k$) else Handle separately.

Runtime:

$$T(n) = 2T(n/2) + c \cdot n$$
$$= O(n \log n)$$

How to solve recursion:

Approach 1: Unrolling:

$$T(n) = 2T(n/2) + c \cdot n \leftarrow$$
$$= 2\left(2T\left(\frac{n}{4}\right) + c \cdot n/2\right) + cn$$
$$= 4T\left(\frac{n}{4}\right) + 2c \cdot n$$
$$= \vdots \quad \text{k-steps of unrolling}$$

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

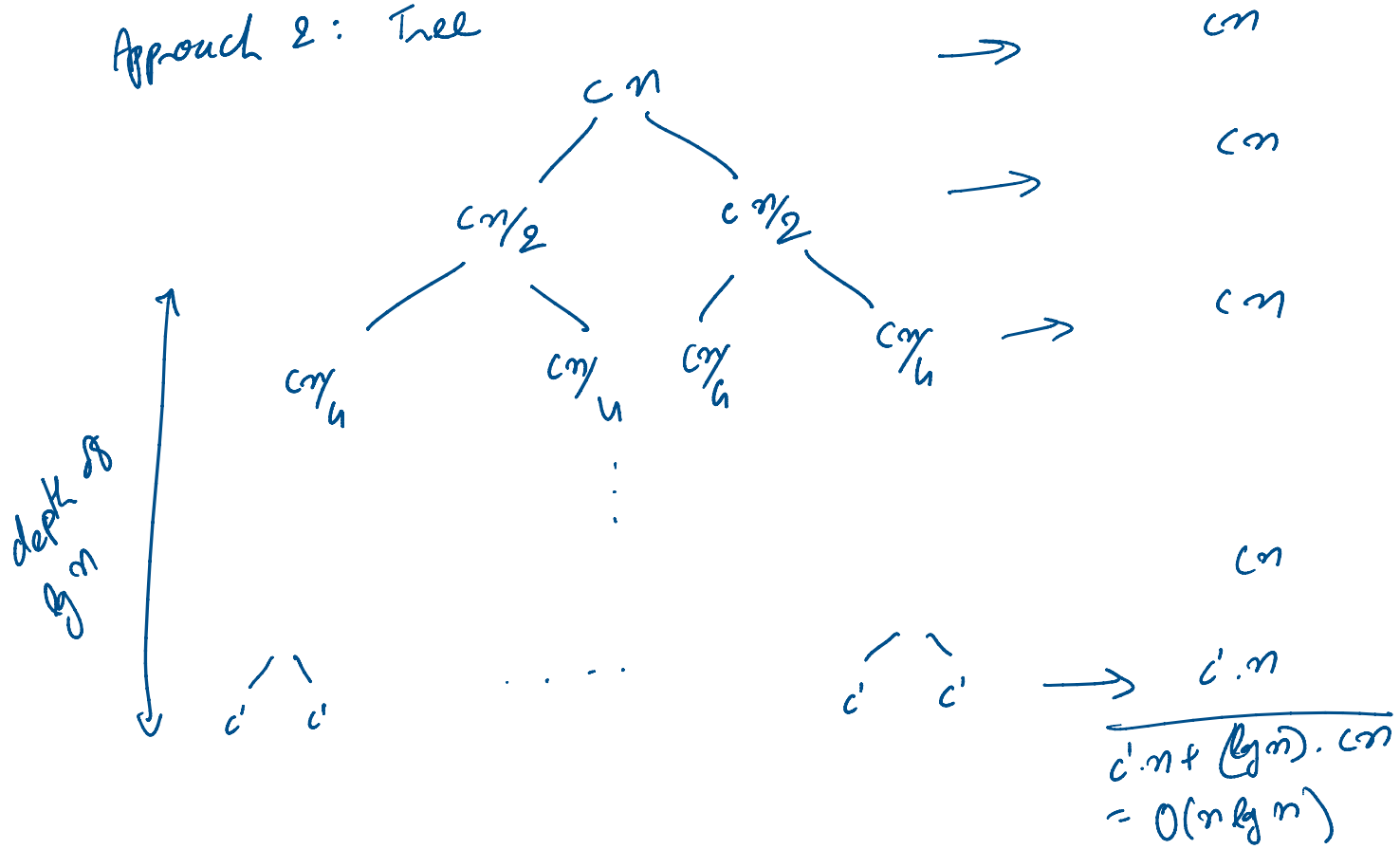
$\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

$$= 2^{\log_2 n} \underbrace{T(1)}_{c'} + \log_2 n \cdot cn$$

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

$$= O(n \cdot \log_2 n)$$

Approach 2: Tree



Approach 3: Guess & prove by induction.

Why $O(\cdot)$?

... about asymptotics & ignore constants!

Why $O(\cdot)$.

Why care about asymptotics & ignore constants?

eg.

	$O(n \lg n)$	vs.	$O(n^2)$
10^9 ops/sec	$10\, n \lg n$		n^2
$n = 10^6$	$10^7 \times 20$		10^{12} ops
	0.22 sec		1000 sec
			~ 15 min

↓

$O(n^2)$	vs	$O(2^n)$
n^2	vs	2^n
~ 15 min		~ 10^{13} years.

Heap Sort: (look up) insertion sort w/ smart data structure

$O(n \lg n)$

QuickSort $(A[1 \ldots n])$ → $T(n)$

0. If $n=1$ then return.
1. Pick a pivot x.

How? ← Randomized.

$O(n)$ {
2. Compare x w/ every $A[i]$

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

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

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

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

Recurrence: $T(n) = T(l) + T(n-l) + c \cdot n$

Runtime: $T(n) = T(l) + T(n-l) + c \cdot n$

wish: $l = n/2$

$$T(n) = 2\,T(n/2) + c \cdot n$$
$$= O(n \cdot \log n)$$

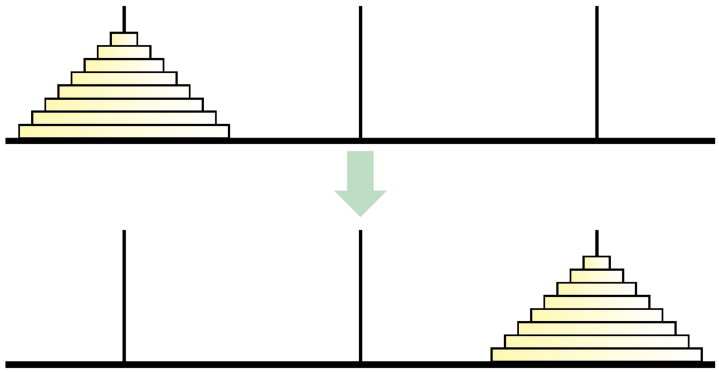
worst: $l = 1$

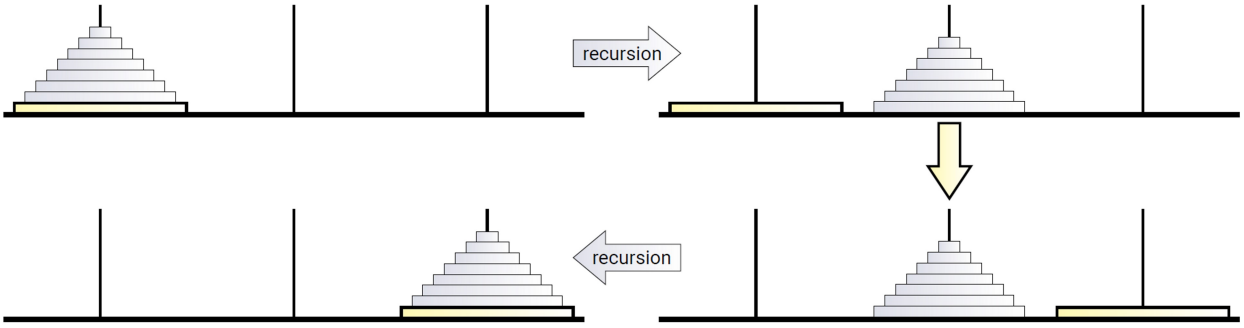
$$T(n) = T(1) + T(n-1) + c \cdot n$$
$$= T(n-1) + O(n)$$
$$= O\left(\sum_{i=1}^{n} (n-i)\right) = O(n^2)$$

Rmk: Running time may depend on the i/p. (size n)

"worst-case" running time

worst i/p of size n.




recursion
recursion