February 19, 2019 10:59 AM

PART II: ALGORITHMS

how to solve specific problems efficiently

time (# steps) & space in RAM model

Ex: Sorting

Given n numbers $A[1], \dots, A[n]$, reorder them s.t. $A[1] \le A[2] \le \dots \le A[n]$

e.g. 50, 82, 43, 19 | 96, 32, 74, 25

→ 19, 25, 32, 43, 50, 74, 82, 96

19, 43, 50, 82 | 25, 32, 74, 96

Alg'm 1: selectionsort

idea - find smallest, remove, repeat

1. for $i = 1$ to n {
   // find min of $A(i..n)$
2. min = i
3. for $j = i+1$ to n
4. if $A(j) < A(\text{min})$ then min $= j$
5. swap $A(\text{min})$ with $A(i)$

}

$\boxed{i-1}\underbrace{\boxed{\qquad\qquad}}_{A(i..n)}$

What is runtime?

order notation hides const factor

What is runtime? hides const factor

lines 3-4 : $O(n-i)$

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

$= O(n-1+n-2+\ldots+1)$

$= \boxed{O(n^2)}$

Alg'm 2: insertionsort

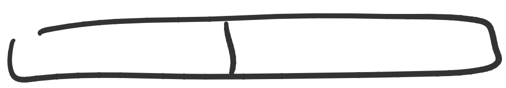
for $i = 1$ to n
  insert $A[i]$ into sorted list $A[1..i-1]$
  (omit details)

$\Rightarrow \boxed{O(n^2)}$ also

Alg'm 3: mergesort

idea - divide & conquer, i.e., recursion

mergesort $(A[1..n])$:



1. if $n=1$ return
2. $T(n/2) \rightarrow$ mergesort $(A[1..\lfloor n/2\rfloor])$
3. $T(n/2) \rightarrow$ mergesort $(A[\lfloor n/2\rfloor+1..n])$
4. $O(n) \rightarrow$ merge 2 sorted lists $A[1..\lfloor n/2\rfloor]$ & $A[\lfloor n/2\rfloor+1..n]$

how to merge $B[1..m]$, $C[1..n]$ into $D[1..m+n]$:

idea - scan from left to right

B: 19, 43, 50, 82
C: 25, 32, 74, 96

left to right

C: 25, 32, 74, 96
$\uparrow$
j

$O(m+n)$ time

$i=1,\ j=1$

for $k=1$ to $m+n$

if $(B[i] \leq C[j])$ or $j=n+1$ then $D[k]=B[i]$, i++
& $i \leq m$
else $D[k]=C[j]$, j++

Runtime:

let $T(n)$ = runtime of mergesort on n elements

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

$\Rightarrow \boxed{O(n \log n)}$ much better than n^2

How to solve recurrence?

approach 1 - unroll

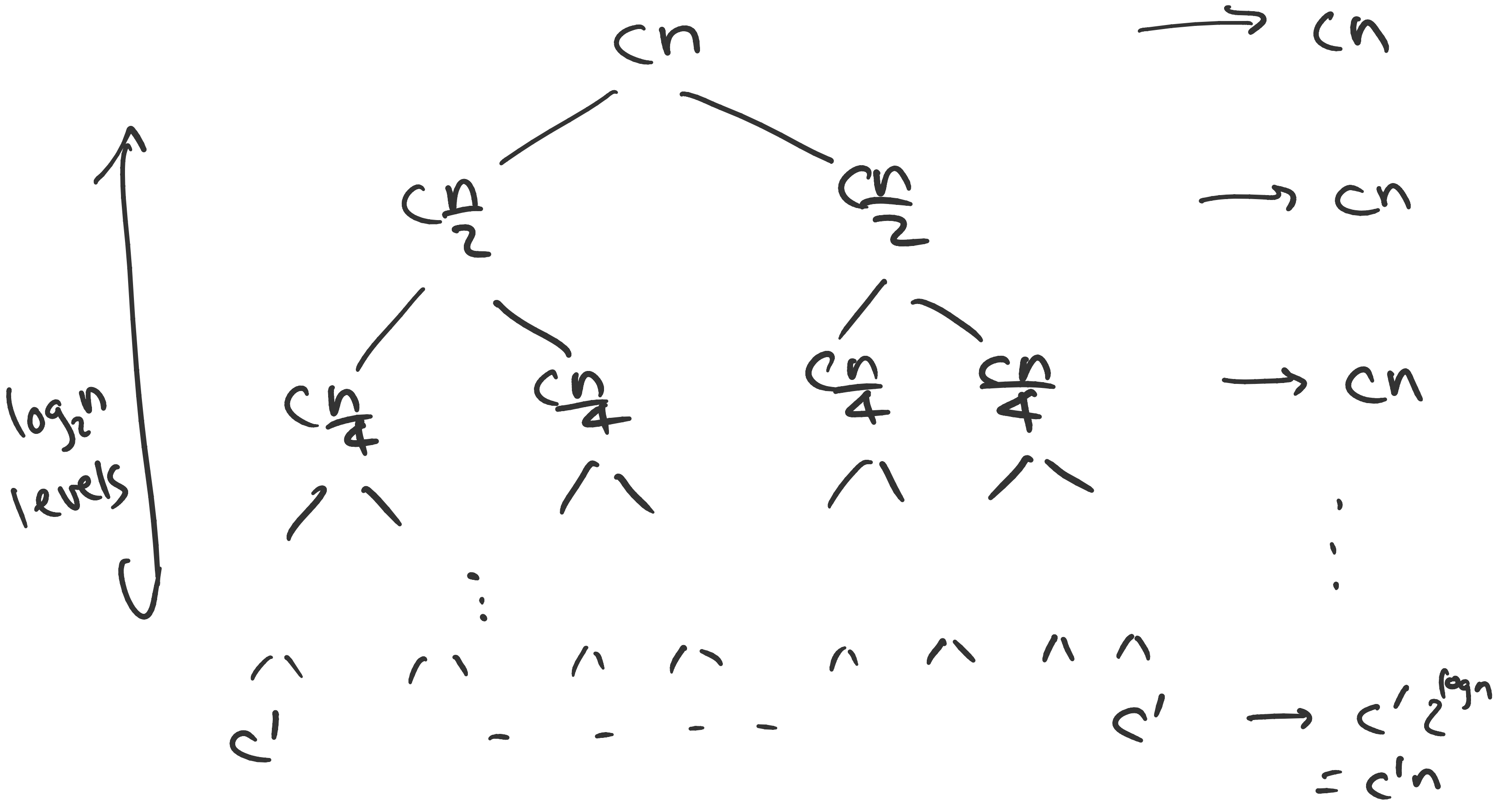
$$\begin{aligned} T(n) &= 2T\left(\tfrac{n}{2}\right) + cn \\ &= 2\left[2T\left(\tfrac{n}{4}\right) + c\tfrac{n}{2}\right] + cn \\ &= 4T\left(\tfrac{n}{4}\right) + 2cn \\ &\ \vdots \\ &= 2^k T\left(\tfrac{n}{2^k}\right) + kcn \\ &= c'n \cancel{T(1)} + cn\log n \\ &= O(n\log n) \end{aligned}$$

set $k = \log_2 n$

approach 2 - recursion tree

approach 2 - recursion tree

same as unrolling, but in picture form



$\boxed{O(n \log n)}$

approach 3 - guess answer!

verify by induction

Rmk - growth rate matters more than const factors! on large input

e.g. compare $20 n \log_2 n$ vs. $2n^2$

say 10^9 ops/sec

$n = 10^6$: 0.4s vs. 2000s

$\sim$ 30min

e.g. compare $2n^2$ vs. 2^n

$n = 100$: $\sim 10^{21}$ sec

$n = 100$: ~ 10 sec

$\sim 10^{13}$ yrs

Rmk: - runtime may vary over diff. inputs of size n

- will analyze worst-case runtime as a function of input size n

---

Other Alg'ms for sorting:

- refined selectionsort w. data structures $\Rightarrow$ heapsort also $O(n \log n)$ time

- different divide & conquer

quicksort$(A[1..n])$:

  if $n=1$ return

how? $\rightarrow$ Pick a pivot x

  partition A into 2 parts:

$O(n)$ time $\rightarrow$ $\{A(i) : A(i) \leq x\} \rightarrow A(1..\ell)$

  $\{A(i) : A[i] > x\} \rightarrow A(\ell+1..n)$

  quicksort$(A[1..\ell])$

  quicksort$(A[\ell+1..n])$

Runtime:

$$T(n) = T(\ell) + T(n-\ell) + O(n)$$

Ideal case: $\ell = n/2$

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

Ideal case:

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

bad case: $\ell = 1$

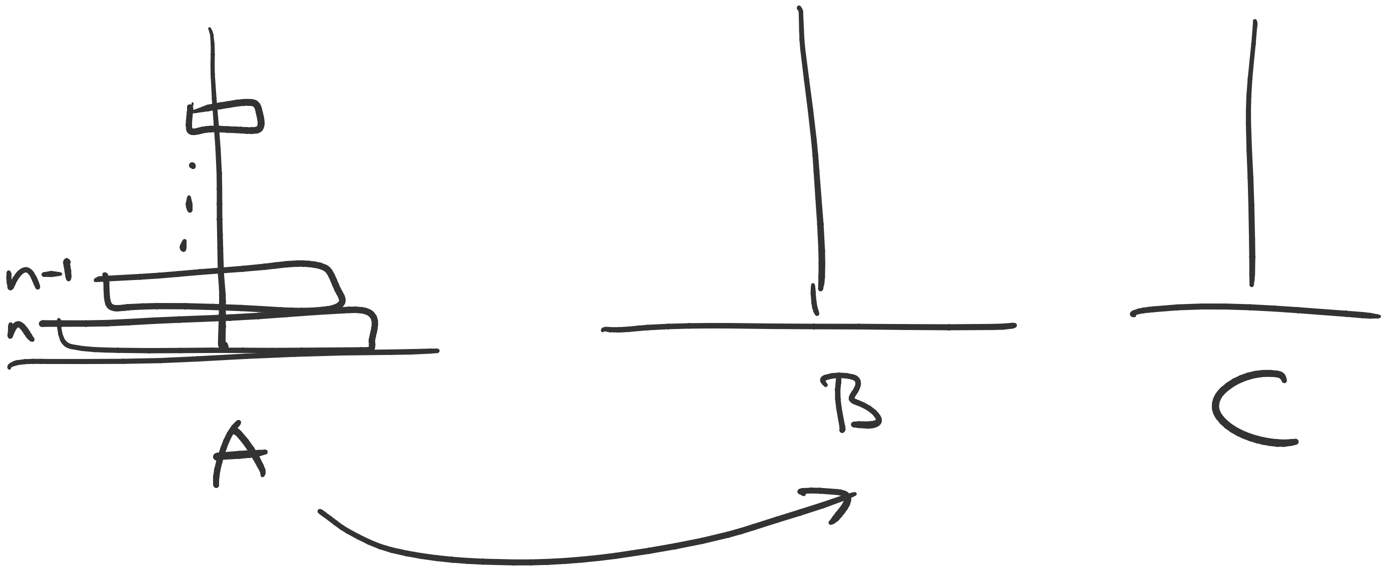
$$T(n) = \cancel{T(1)} + T(n-1) + O(n)$$

$\overset{\text{unroll}}{\Rightarrow}$ $T(n) = O(n + n-1 + n-2 + \dots + 1)$

$= O(n^2)$. unfortunately!

- lower bd $\Omega(n\log n)$
  over all comparison-based alg'ms

Ex2 Tower of Hanoi



can't put ~~smaller~~ larger disk on top of ~~larger~~ smaller

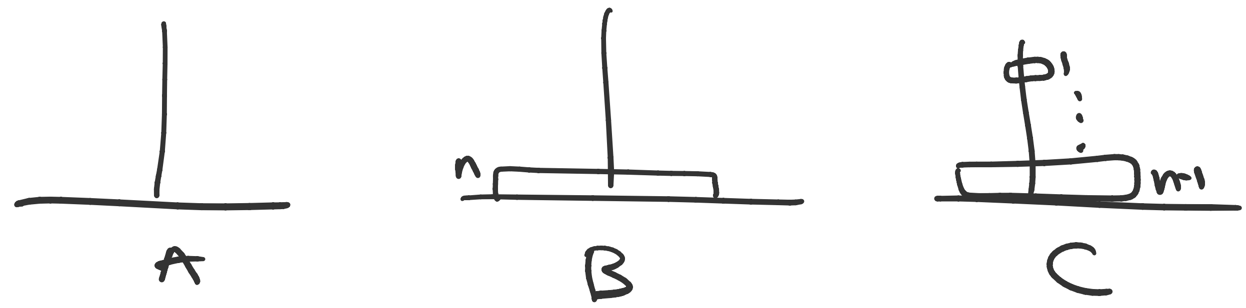
move(n, A, B, C): // C as tmp

  if n=0 return

  move(n-1, A, C, B)

  move disk n from A to B ←

  move(n-1, C, B, A)



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

$\Longrightarrow$ $\underline{O(2^n)}$ best possible!