TC & SC's of all Codes (Practical)

① Binary search –

TC = { O(log₂n) → Avg case ; Θ(1) → Best case ; O(n) → Worst case }

SC = { O(1) → for iterative ; O(log n) → for recursive }

$\log_2 n \approx \log n$

Process of getting TC

Basic substitution method

$T(n) = aT(n/b) + f(n)$

$T(n) = T(n/2) + 1$ → recurrence relation for binary search

$a = 1, b = 2, f(n) = 1$

$T(n) = T(n/2) + 1$

$= (T(n/4) + 1) + 1$

$= T(n/4) + 2$

$= (T(n/8) + 1) + 2$

$= T(n/8) + 3$

⋮

$T(n/2^k) + k$

[if $n = 2^k$ then $T(1) = \text{const}$]

$\therefore n = 2^k$

$\therefore k = \log_2 n$

$\therefore TC = O(\log_2 n)$

↓

Avg case

applying master's theorem:

$T(n) = aT(n/b) + f(n)$ } recurrence relation

$T(n) = T(n/2) + 1$ → recurrence relation for BS

$f(n) = 1, a = 1, b = 2$

$f(n)$	$n^{\log_b a}$
1	$n^{\log_2 1}$
	$= n^0$
	$= 1$

$\therefore TC = O(n^{\log_b a} * \log_b n)$

$= O(n^{\log_2 1} * \log_2 n)$

$= O(1 * \log_2 n)$

$= O(\log_2 n)$

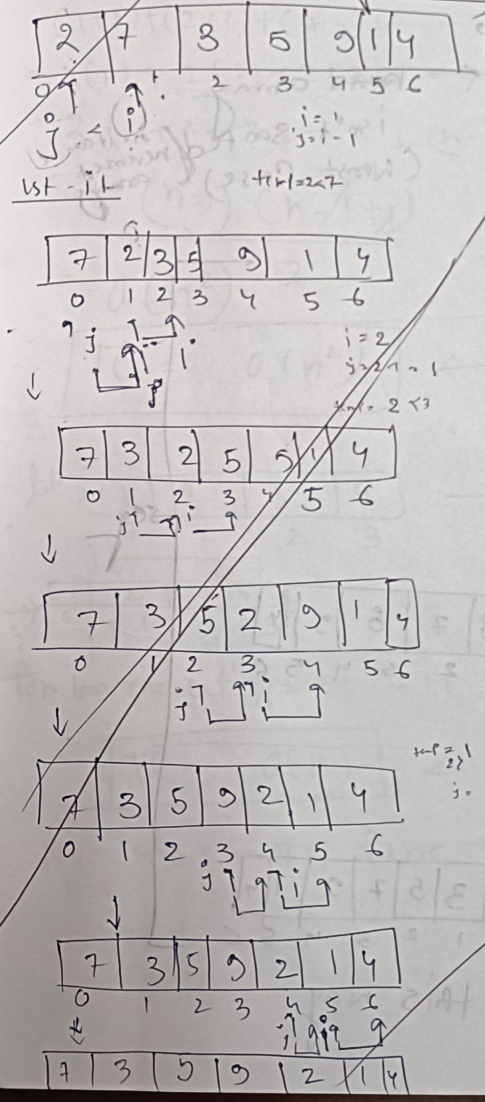
$\therefore TC = O(\log_2 n)$

② Insertion Sort

TC = { $O(n^2)$ → worst case ; $O(n)$ → Best case }

SC =

Process of calculating TC



② Insertion sort

SC = O(1)

TC = ? O(n²) → Worst case

O(n) → Best case

Process of calculate TC

2	7	3	5	9	1	4
0	1	2	3	4	5	6

← ~~bond~~ array isn't sorted (or it can be reversed array) (Worst Case)

1st - it

i = 1, j = i - 1 = 0

temp = 7 ≯ 2

2	7	3	5	9	1	4
0	1	2	3	4	5	6

j i

j = 1, i = 2, temp = 3

3 < 7 ~

2	7	7	5	9	1	4
0	1	2	3	4	5	6

j i

→

2	3	7	5	9	1	4
0	1	2	3	4	5	6

7 ≯ 2

i = 3, j = 2, temp = 5 < 7

2	3	7	7	9	1	4
0	1	2	3	4	5	6

j i

→

2	3	5	7	9	1	4
0	1	2	3	4	5	6

5 ≯ 3, 5 ≯ 2 ..... this way

seq

read from there, almost

∴ When it = 1, no. of comparisons = (1+1) (1 comp, 1 movement)

When it = 2 ---- = (2+2)

When it = 3 ---- = (3+3)

⋮

When it = n-1 ---- = (n-1)

∴ total no. of comparisons,

(1+1) + (2+2) + (3+3) + ... + ((n-1) + (n-1))

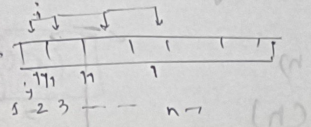
= 2(1) + 2(2) + 2(3) + ... + 2(n-1)

= 2(1 + 2 + 3 + ... + (n-1))

$= 2 * \frac{(n-1)(n-1+1)}{2}$

$= O(n^2)$

∴ TC = $O(n^2)$ ← at worst case



let,

1	2	3	4
0	1	2	3

→ array is sorted (Best case)

i

1	2	3	4
0	1	2	3

j j j

for loop executing = 1 + 1 + 1 + ... = O(n)

∴ TC = O(n) ← at best case

No movements here just comparisons.

③ Quick sort **

TC = { $O(n\log_2 n)$ → best case ; $O(n^2)$ → worst case }

SC = { $O(\log_2 n)$ → Best case ; $O(n)$ → worst case }

Process of calculating tc, sc – eg copy

tc,

recurrence relation = $T(n) = 2T(n/2) + O(n)$ (for 2 quick sorts; for partition) → for best case

applying master's theorem,

$f(n) = O(n)$ <=> $n^{\log_b a} = n^{\log_2 2} = n$

$\therefore TC = O(n^{\log_b a} * \log_b n)$

$= O(n * \log_2 n)$

$= O(n\log_2 n)$

$TC = O(n\log_2 n)$ → for best case

recurrence relation = $T(n) = T(n-1) + T(0) + O(n)$ (T(0): const, nothing left; for pivot)

~~applying master's the~~ → for worst case

$= T(n-1) + c \cdot n$ – ① [∵ T(0) + O(n) = const = const (c) multiplied with n]

$= T(n-2) + c(n-1) + c \cdot n$

$= T(n-3) + c(n-2) + c(n-1) + c \cdot n$

$T(n-k) + c(n-k+1) \ldots + c \cdot n$

∴ $n-k = 1$, then $T(1)$

$\therefore T(1) + c \cdot 2 + c \cdot 3 + \cdots + c(n-1) + c(n)$ (T(1): 1 unit)

$= c(1 + 2 + \cdots + n)$

$= c \cdot \frac{n(n+1)}{2}$ [∵ $\frac{(n-1)(n-1+1)}{2}$]

$= O(n^2)$

$TC = O(n^2)$ ← for worst case

④ merge Sort **

TC = $O(n\log_2 n)$ ↓ for all Cases | SC = $O(n)$

~~For~~ eg - copy

Process of calculating tc, sc,

tc, recurrence relation, $T(n) = 2T(n/2) + O(n)$ (for 2 merge sorts; for merging)

applying master's theorem, $a = 2, b = 2, f(n) = O(n)$

$f(n) = O(n)$ <=> $n^{\log_b a} = n^{\log_2 2} = n$ ∴ $= O(n^{\log_b a} * \log_b n)$

$= O(n * \log_2 n)$

$= O(n\log_2 n)$

∴ $TC = O(n\log_2 n)$

SC, height of the tree = $\log_2 n$ (∵ no. of levels = n)

There are n no. of eles, that are taking k amount of space

∴ $k * \log_2 n = O(k\log_2 n)$ → for merge sort

we are merging n no. of eles ∴ $O(n)$ → for merging

∴ $SC = O(n)$ [∵ $O(n) > O(k\log_2 n)$]

(5) max heapify (to create max heap)

i) TC = $\leq O(\log_2 n)$ | SC = $O(\log_2 n)$

Eg - from copy

Process TC, height of the tree = $\log_2 n$

There are 2 comparisons depending upon which node has max ele.

If we apply max heapify at root max no. of comparisons is going to be happen (= max TC).

$\therefore$ max TC = $2 * \log_2 n$

$O(\log_2 n)$

But TC can be lessen than it.

$\therefore$ $TC \leq O(\log_2 n)$

SC, height of the tree(heap) = $\log_2 n$

there are n no. of eles(i.e, n, largest, rest of nodes), that are taking a constant (k) amount of memory

$\therefore$ $S(n) = k * \log_2 n$

$\therefore$ $SC = O(\log_2 n)$

ii) create max heap

TC = $O(n)$ | SC = $O(\log_2 n)$

Process,

TC,

at a particular height (h) the max. no. of nodes = $\left\lceil \frac{n}{2^{h+1}} \right\rceil$

If we apply mh at leaf node - no. of comp. = 0 → O(0)

Parent of leaf node - - = 1 → O(1)

Parent of that nod - - = 2 → O(2)

at h → O(h) = This is also the TC of any particular node at any particular level.

now, mh executes for all nodes,

$\therefore$ the max amount of time required,

$$\sum_{h=0}^{\log_2 n} \frac{n}{2^{h+1}} * O(h)$$

$$= \sum_{h=0}^{\log_2 n} \frac{n}{2^{h+1}} \cdot c.h \quad [\because O(h) \text{ is } h \text{ mul. some const.}]$$

$$= \frac{cn}{2} \sum_{h=0}^{\log_2 n} \frac{h}{2^h}$$

$$= \frac{cn}{2} \sum_{h=0}^{\infty} \frac{h}{2^h}$$

$$\leq \left(\frac{cn}{2} \sum_{h=0}^{\infty} \frac{h}{2^h} \right)$$

$$\leq \left(\frac{c \cdot n}{2} \cdot 2 \right)$$

$\therefore$ $TC = O(n)$

SC → height of tree = $\log_2 n$

n no. of eles contain k amount of memories,

$\therefore$ $S(n) = k * \log_2 n$

$\therefore$ $SC = O(\log_2 n)$

④ extract max ele

TC = $O(\log_2 n)$ | SC = $O(\log_2 n)$

as mh's ↓ = $O(\log_2 n)$

eg: copy

⑦ i) Increase key [for all eg: copy]

TC = $O(\log_2 n)$ | SC = $O(\log n)$ → ~~for both best & worst case~~ (max heapify → any way is needed)

[we will perform this o/p from key to root as, the Right of tree is $\log_2 n$, so, TC $O(\log_2 n)$]

ii) Decrease key

TC = $O(\log n)$ (max heapify needed) | SC = $O(1)$

iii) insert key

TC = $O(\log n)$ (mh-ify needed) | SC = $O(1)$

iv) Delete key

TC = $O(\log_2 n)$ (mh needed) | SC = $O(\log n)$ → (mh needed)

[∵ max heapify is applied, & it's TC = $O(\log_2 n)$]

v) Heap Smt

TC = $O(n \log_2 n)$ | SC = $O(1)$

Process,

TC, we have to perform mh for atleast n/2 times & TC of mh is $\log_2 n$, ∴ $T(n) = \frac{n}{2} + \log_2 n$

⑧ mm for large dim

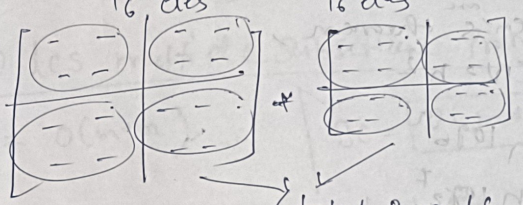
TC = $O(n^3)$ | SC = $O(n^2)$

Process;

TC, recurrence relation, $T(n) = \begin{cases} O(1) \text{ or } 1 ; \text{if } n \le 2 \\ 8T(n/2) + n^2, \text{if } n > 2 \end{cases}$

recur eg, 4×4 mat * 4×4 mat

(A) 16 eles, (B) 16 eles



total 8 muls ~~& 16 elems~~

↑ to perform mul for dim n/2, the time taken = $T(n/2)$ & there are 8 muls so, $8T(n/2)$

↓ for mat. add

now applying master's theorem,

a = 8, b = 2, $f(n) = n^2$

$f(n) = n^2$

$n^{\log_b a} = n^{\log_2 8} = n^3$

∴ TC = $O(n^3)$

[Normal matrix mul, 3 for loops, ∴ TC = $O(n^3)$, SC = $O(n^2)$]

(9) Strassen's mm

TC = $O(n^{2.81})$ | SC = $O(n^2)$

Process, tc, recurrence relation,

$$T(n) = \begin{cases} 1 & ; \text{if } n \le 2 \\ 7T(n/2) + n^2 & ; \text{if } n > 2 \end{cases}$$

7 mults needed here; n^2 for matadd.

now applying master's theorem,

$a = 7,\ b = 2,\ f(n) = n^2$

$f(n) = n^2$

$n^{\log_b a}$
$= n^{\log_2 7}$
$= n^{2.81}$

$\therefore$ tc = $O(n^{2.81})$

(10) Knapsack (fractional)

TC = $O(n \log n)$ | SC = $O(n)$

[n = no. of objs]

(11) Prim's algo

TC = $O(E * \log n)$ | SC = $O(E + V)$

[E = no. of edges] [E = no. of edges, V = no. of vertices (nodes)]

Process, tc, Say, there are n no. of nodes (vertices)

$O(n) + O(1) + O(n \log n) + O(E * \log n) \Rightarrow TC = O(E \log n)$ [$\because n \le E$]

[Best tc of Prim's algo is by using fibonacci heap
• tc without using min heap = $O(n^2)$ = $O(n \log n)$]

DP

(12) lcs

TC = $O(m+n)$ | SC = $O(m+n)$

[m = len(str1)
n = len(str2)]

(13) lcs multiple subsequence Print.

TC = $O(m*n)$ | SC = $O(m*n)$

(14) matrix chain multiplication

TC = $O(n^3)$ | SC = $O(n^2)$

Process, tc

tc of min scalar mul. table = $O(n^2)$
tc of split = $O(n)$
$\therefore$ tc = $O(n^2) * O(n)$
tc = $O(n^3)$

[for every dp approached algo tc = no. of cells
eg - (12), (15) etc]

(15) 0/1 Knapsack [If tc < 2^n, use fractional knapsack]

TC = $O(n * w)$ | SC = $O(n*w)$

[n = no. of objs
w = weight
n*w = no. of cells]

Eg - copy

[Without dp approach
tc = $O(n * 2^n)$]

Ex, relations —

$$1 \leq \log n \leq \sqrt{n} \leq n \leq n + \log n \leq n^2 \leq n^3 \cdots \leq n^n \leq 2^n \leq$$

$O(1)$

$$3^n \cdots \leq n^n$$