Data Structures

2. Giving suitable example discuss Quick sort and Bubble sort algorithm and its category (In place or out-of-place sorting algorithm). Discuss its time complexity and compare it with merge and insertion sort. Also make program of each algorithms.

Submitted By

Name : Harsh Soni

Roll No.: 11912082

Branch: IT

Q2

```
QUICK_SORT (arr, p, q):        [INPLACE ALGORITHM]
1  if p<q:
2     r=PARTITION (arr, p, q).
3     QUICK_SORT (arr, p, r-1)
4     QUICK_SORT (arr, r+1, q).

PARTITION (arr, p, q):
1  x = arr[q]
2  i = p-1
3  for (j ⟵ p to q-1 :
4     if arr[j] < x :
5        swap (arr[j], arr[++i])
6  arr[++i] = x
7  return i
```

A Qs ...

* Since, the partition fun^n is traversing through the array once, hence the time complexity of partition $= \Theta(n)$.

$P(n) = \Theta(n)$ { Time complexity of partition }.

* BEST CASE TIME COMPLEXITY OF QUICK SORT.

Best case occurs when we have a balanced split (i.e we get merely each time we partition).

So, the reccurrence relation becomes.

$T(n) = T(\lfloor n/2 \rfloor - k) + T(\lfloor n/2 \rfloor - k) + \Theta(n)$

$T(n) \approx 2T(n/2) + \Theta(n)$

{ $T((n-1)/2) + T(\frac{(n-1)}{2} - 1) + \Theta(n)$

a.) using masters theorem,

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

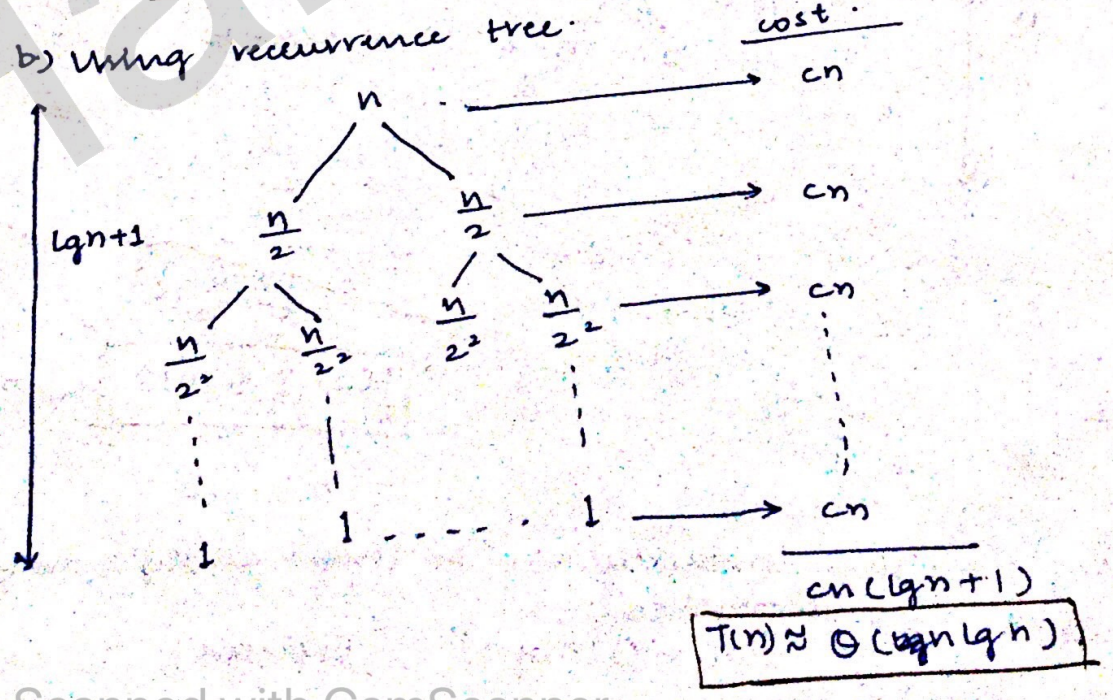
where $a = b = 2$,

$f(n) = \Theta(n)$.

$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$.

$\Rightarrow \boxed{T(n) = \Theta(n \lg n)}$

b) Using reccurrence tree.



* WORST CASE TIME COMPLEXITY OF QUICK SORT.

The worst case occurs when the ~~is~~ splits are extremely unbalanced.

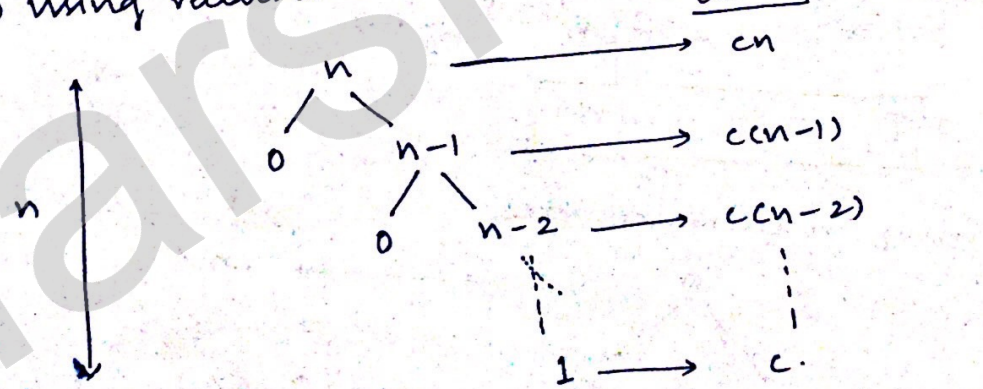
$$T(n) = T(0) + T(n-1) + \Theta(n).$$
$$= T(n-1) + \Theta(n).$$

a.) using Backtracking

$$T(n) = T(n-1) + \Theta(n)$$
$$= T(n-2) + \Theta(n-1) + \Theta(n)$$
$$= \sum_{k=1}^{n} \Theta(k)$$
$$= \Theta(\Sigma k).$$

$$\boxed{T(n) = \Theta(n^2)}$$

b.) using recurrence tree.



$$cn\left[\frac{n(n-1)}{2}\right]$$

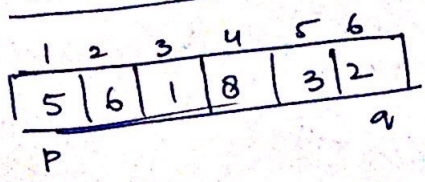
$$\Rightarrow \boxed{T(n) \approx \Theta(n^2).}$$

* Avg case time complexity can be found easily using above two relations

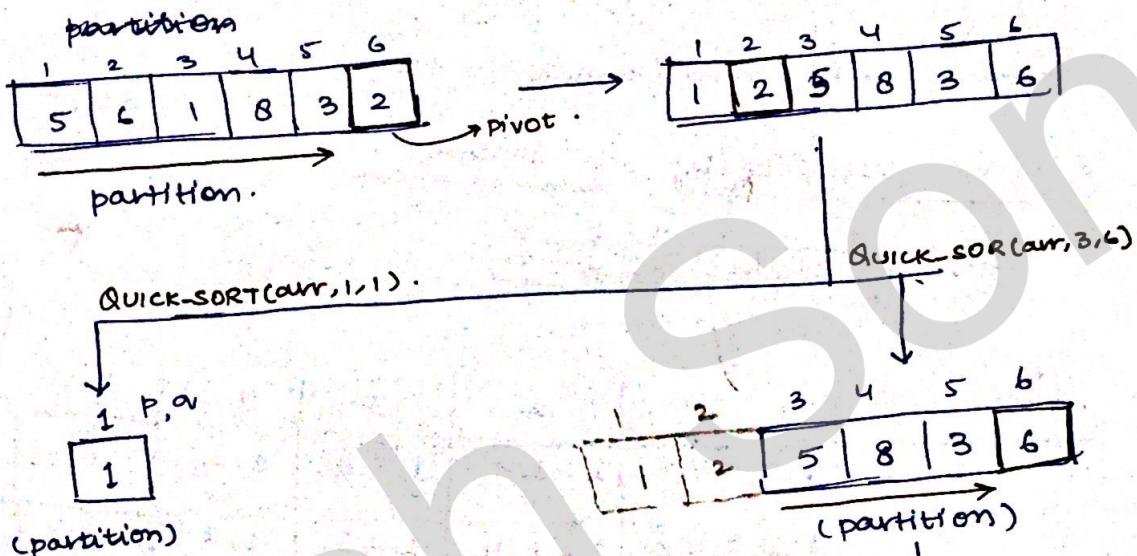
$$T(n) = O(n \lg n). \quad \{\text{Expected}\}.$$

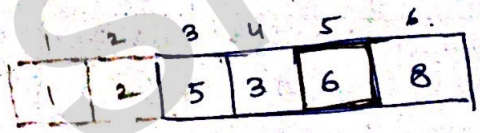
* eg of QUICK SORT

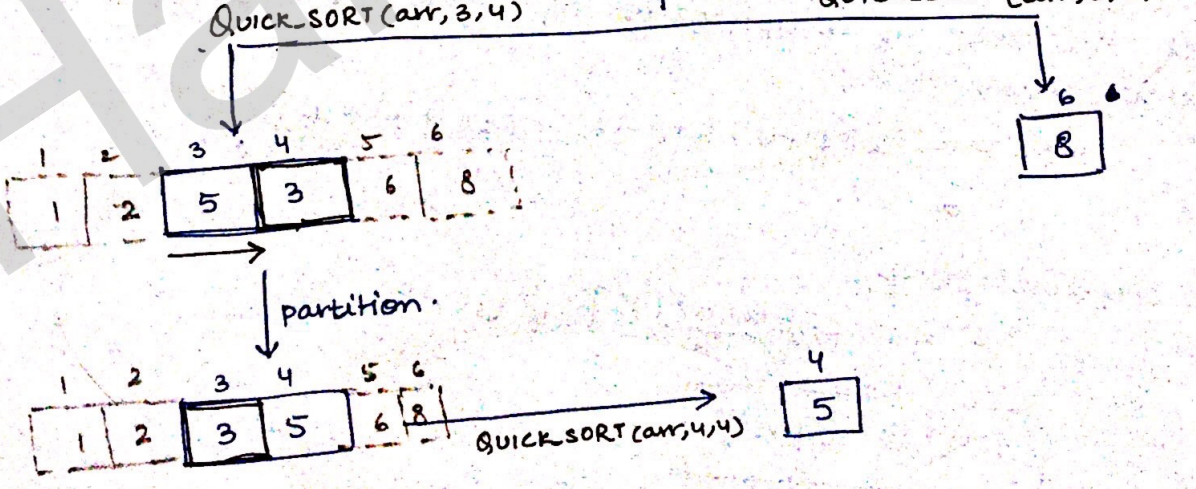
eg:



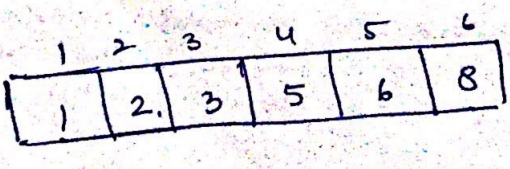
(NOTE the imbalance partitioning)







=>



For level 2,

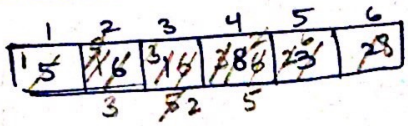
[INPLACE ALGORITHM]

```
BUBBLE_SORT(arr).
1 for(i ← 1 to n):
2     for(j ← 1 to n-i):
3         if arr[j] > arr[j+1]:
4.            swap(arr[j], arr[j+1]).
```

~~∴ swap take O(1) / constant time~~

* Time complexity of Bubble sort

eg:



i	1	2	3	4	5	6
j	1,2,3,4,5	1,2,3,4	1,2,3	1,2	1	—
comparisons	1,2,3,4,5 ⑤	1,2,3,4 ④	1,2,3 ③	1,2 ②	1 ①	—
swaps	0,1,2,3,4 ④	1,1,2,3 ③	0,1,2 ②	0,0 ⓪	0 ⓪	—

Let the cost of computation of lines 3 & 4 be C_1 and C_2 respectively, then.

$$T(6) = C_1(5+4+3+2+1) + C_2(4+3+2).$$

$$=$$

⇒ Generalizing,

$$T(n) = C_1\left[\frac{n(n+1)}{2}\right] + O(n^2).$$

$$\boxed{T(n) \approx O(n^2)}$$

* NOTE : Since, the comparisions will be made irrespective of the arrangement of elements, ∴ best case time complexity of bubble sort = $O(n^2)$.

* MERGE SORT Vs QUICK SORT Vs INSERTION SORT Vs BUBBLE SORT.

	MERGE SORT	QUICK SORT	: INSERTION SORT	BUBBLE SORT.
TIME COMPLEXITY	$O(n \lg n)$	$\Theta(n^2)$ $*O(n \lg n)$	$O(n^2)$	$O(n^2)$.
CATEGORY	OUT-OF-PLACE [$O(n)$]	INPLACE	INPLACE	INPLACE
SPACE COMPLEXITY (EXTRA)	$O(n)$	—	—	—
EFFECT OF ALREADY SORTED I/P	—	—	ACHIEVES BEST CASE [$O(n)$]	— .

* → Randomized Quick sort can be used to achieve an expected running time of $O(n \lg n)$ moreover an avg case time complexity argument of $O(n \lg n)$ can also be made for quick sort.