Ishant Sehrawat
CSE-A
03620802719

Assignment - 1

Ques 1. What is quick sort? Explain the algorithm & apply it to the following list : 15, 10, 35, 9, 3, 25, 10

Quick Sort works on the divide & conquer algorithm
following is the algorithm:

- Consider an array 'S' of size 's'.
- if size ≤ 1 → sorted
- Pick any element 'v' as pivot
- Partition S - {v} into 2 groups S_1, S_2

S_1 = { all elements < v }
S_2 = { all elements > v }

- Return quick (S_1), v, quick sort (S_2)

Partition Technique

					P

i → ← j

→

elem < P	P	elem > P

i will search for element < P
j will search for element > P

15	10	35	9	3	25	10

i = left, j = right, pivot

i++
j--

3	10	9	35	15	25	10

i → ← j

Swap a[i] & pivot

3	10	9	10	15	25	35

$\underbrace{3 \quad 10 \quad 9}_{S1} \quad \underbrace{10}_{pivot} \quad \underbrace{15 \quad 25 \quad 35}_{S2}$

Time Complexity comparison

	Best	Avg	Worst
Quicksort	O(n log n)	O(n log n)	O(n²)
Mergesort	O(n log n)	O(n log n)	O(n log n)

Ques7. Solve the following recurrence

1) $T(n) = 2T(n/2) + n$ by substitution method

$$T(n) = 2T(n/2) + n \quad \text{—— (1)}$$
$$T(n/2) = 2T(n/2^2) + n/2 \quad \text{—— (2)}$$
$$T(n/2^2) = 2T(n/2^3) + n/2^2 \quad \text{—— (3)}$$

$$T\left(\frac{n}{2^{k-1}}\right) = 2T\left(\frac{n}{2^k}\right) + \frac{n}{2^{k-1}}$$

Assuming $n = 2^k$ & $T(1) = 1$,

$$T(n) = n + 2\left(\frac{n}{2} + 2\left(\frac{n}{2^2} + \cdots 2\left(\frac{n}{2^{k-1}} + 2(1)\right)\right)\right)\ldots$$

$$T(n) = n + n(k-1) + 2^{k-1} \times 2$$
$$= nk + 2^k$$

Now, $n = 2^k \Rightarrow k = \log_2 n$

$$\boxed{T(n) = n * \log_2 n + n}$$

$O(n \log n)$

Scanned with CamScanner

2) $T(n) = 3T(n/4) + n$ by recurrence tree method

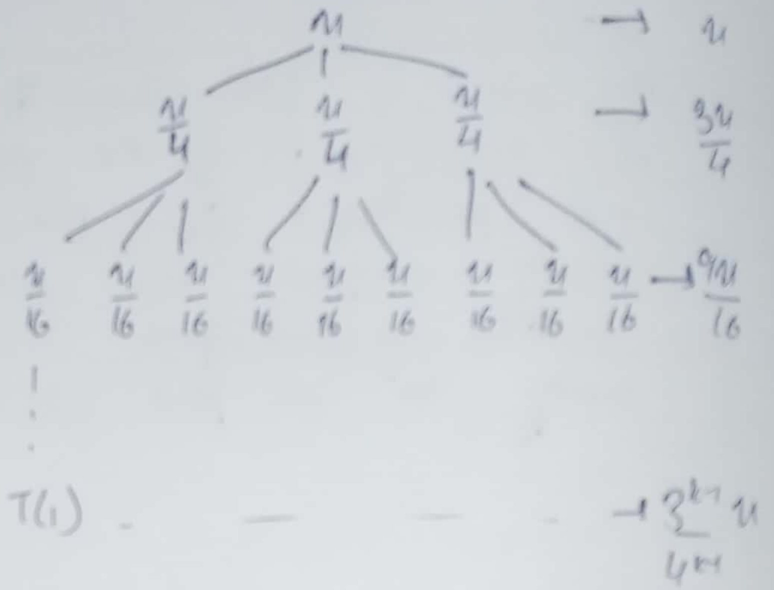
<u>Recursive Call</u>

$T(n)$

$T(n/4)$

$T\left(\frac{n}{16}\right) = T\left(\frac{n}{4^2}\right)$

<u>Tree</u>



$n = 4^k$ $\quad T\left(\frac{n}{4^k}\right)$

$k = \log_4 n$

Assuming $T(1) = 1$

$$T(n) = n\left[1 + \frac{3}{4} + \frac{9}{16} + \dots \left(\frac{3}{4}\right)^{k-1}\right] + \cancel{3k}\ 3^k$$

$$= n\left[\frac{1}{1 - 3/4}\right] + 3^{\log_4 n}$$

$$= n[4] + n^{\log_4 3}$$

$$\boxed{T(n) = 4n + n^{\log_4 3}}$$

$\Rightarrow O(n)$

3) $T(n) = 4T(n/2) + n^2$ by Master Method

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

$a = 4$, $b = 2$, $f(n) = n^2$

$a \geq 1$, $b > 1$

if $h(n)$	$U(n)$
$n^r, r > 0$	$O(n^r)$
$n^r, r < 0$	$O(1)$
$\log(n)^i, i \geq 0$	$\frac{(\log_2 n)^{i+1}}{i+1}$

$T(n) = n^{\log_b a} [U(n)]$

$U(n)$ depends on $h(n) = \frac{f(n)}{n^{\log_b a}}$

$T(n) = n^{\log_2 4} [U(n)]$

$h(n) = \frac{n^2}{n^{\log_2 4}} = \frac{n^2}{n^2} = 1 = (\log n)^0$

$, U(n) = \frac{(\log_2 n)^{0+1}}{0+1} = (\log_2 n)$

$$\boxed{T(n) = n^2 \cdot (\log_2 n)}$$

$O(n) = n^2 (\log_2 n)$

Ques 3- Write a short note asymtotic notation

Asymptotic notation describe how much time an algorithm takes, There are 3 asymptotic for analysing time complexity.

① Big - O notation (Big Oh)

② Big - Ω notation (Big Omega)

③ Big - Θ notation (Big Theta)

- Big - O Notation (Upper Bound) - Describes the worst case running time of the program.
- Big - Ω Notation (Lower Bound) - Describes the best case running time of the program
- Big - Θ Notation → Bounds a function from above & below, so it defines exact asymptotical behaviours

★ Big O : $0 \le f(n)$

★ Big Ω : $0 \le c * g(n) \le f(n)$

★ Big Θ : $(c_1) * g(n) \le f(n) \le (c_2) * g(n)$