Vashchandra

Sec - E

62

2014154

Assignment - 02

Design and Analysis of algorithms

TCS - 505

Q-1

Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis. The main idea of asymptotic analysis is to have a measure of the efficiency of algorithms that don't depend on machine specific constants and doesn't require algorithms to be implemented and time taken by the programs to be compared.

→ Following are the asymptotic notations that are mostly used:-

① O (Big O notation):- It defines and upper bound of an algorithm, it bounds a function only from above.

② Ω notation:- Big omega notation provide the lower bound of function.

③ Θ notation:- Theta notation represent both upper and lower bound of function.

eg -> Let take example of insertion sort :

-> It take linear time in best case and quadratic time in worst case.

∴ we can conclude :

$O(n^2)$

$\Theta(n^2)$ for worst case.

$\Theta(n)$ for best case

$\Omega(n)$

Q-2 here the complexity of code will be

$\Theta(\log n)$

Q-3 here $T(n) = \begin{cases} 3T(n-1) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

$$T(n) = 3T(n-1)$$
$$= 3.(3T(n-2))$$
$$= 3^2 T(n-2)$$
$$= 3^3 T(n-3)$$
$$\vdots$$
$$3^n T(n-n) = 3^n$$

Q-4 here $T(n) = \begin{cases} 2T(n-1)-1 & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

$$T(n) = 2T(n-1)-1$$
$$= 2.(2T(n-2)-1)-1$$
$$= 2^2.(T(n-2))-2-1$$

$= 2^2 (2T(n-3) - 1) - 2 - 1$

$= 2^3 T(n-3) - 2^2 - 2^1 - 2^0$

$= 2^n T(n-n) - 2^{n-1} - 2^{n-2} \ldots 2^3 \ldots 2^2 - 2^1 - 2^0$

$= 2^n - (2^n - 1)$

$= 2^n - 2^n + 1 = 1$

$\therefore T(n) = 1$

Ans-5 here $S = S + i$

if k is total number of iterations taken by the program, then while loop terminate.

$1 + 2 + 3 + \ldots + k = [k(k+1)/2] > n$

$\therefore k = O(\sqrt{n})$

Ans-6 $O(\sqrt{n})$

Ans-7 here j loop is executing "log n" times and k loop is also executing "log n" times and i loop is executing "n/2" times

So Time complexity $= O(n \log_2 n)$

Ans-8 here $O(n^2)$

Ans-9 here inner will form

$n + n/2 + n/3 \ldots + n$

So $n(1 + 1/2 + 1/3 + \cdots 1)$

$\therefore O(n \log n)$

Ans-10 Here

$n^k \quad a^n$

$k >= 1 \qquad a > 1$

Taking $k = a = 2$

$\therefore n^2 \quad 2^n$

$n^2 = O(2^k)$

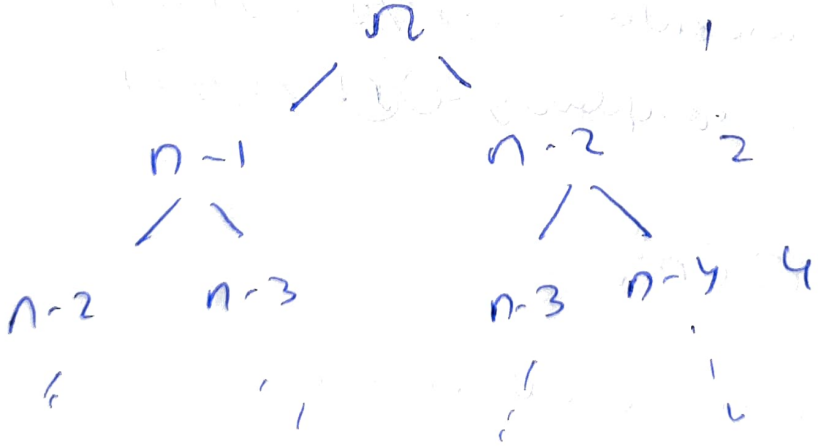
$\therefore n^k = O(a^n)$

Ans-11 Here answer will be $O(\sqrt{n})$ and for loop it is same as loop of question-5

Ans-12 Here Recurrence relation is

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

Taking recurrence tree.



$= 1 + 2 + 4 + \cdots + 2^n$

here $a = 1, r = 2$

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

$$\therefore O(2^{n+1}) = O(2 * 2^n = O(2^n)$$

space complexity : $O(n)$

This is because maximum stack frame is equal to n. only function is called like this

$$f(n-1) + f(n-2)$$

$\therefore$ $f(n-2)$ is called. when we get the return value from $f(n-1)$

$\therefore$ it is equal to $O(n)$

Ans 13 $n \log n$

```
for (i=1; i<n; i++)        (n)
{
    for (j=1; j<=n; j=j+1)  (log n)
    {
        printf ("*");
    }
}
```

```
n^3 <- for (i=1; i<n; i++)   (n)
{
    for (j=1; j<n; j++)      (n)
    {
        for (k=1; k<n; k++)  (n)
        {
            printf ("#");
        }
    }
}
```

log log n int func(int n)
{ if (n <= 2)
return 1;

else
return (func(floor(sqrt(n))) + n);

}

Ans-14 $T(n) = T(n/4) + T(n/2) + cn^2$

We assume
$T(n/2) >= T(n/4)$

$\therefore T(n) = 2T(n/2) + cn^2$

Apply Master Method.

$a = 2, \; b = 2$

$k = \log_b a = \log_2 2 = 1$

$n^k = n \quad , f(n) = n^2$

$\therefore \Theta(n^2)$

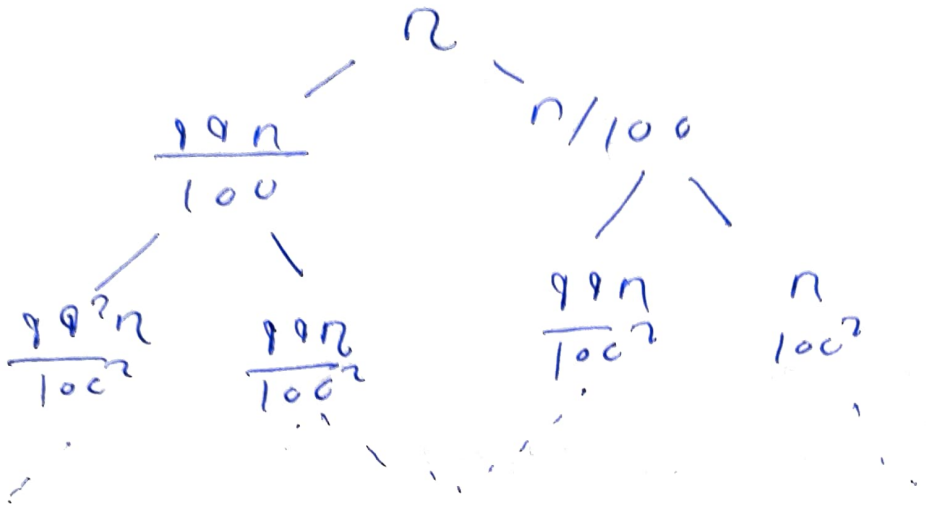
as $T(n) <= \Theta(n^2)$

$T(n) = \Theta(n^2)$

Ans-15 $n(\log n)$

Ans-16 if k is a constant greater than 1
then $T.C = O(\log \log n)$

Ans-17 here $T(n) = T\left(\frac{99n}{100}\right) + T(n/100)$



If we take larger branches i.e. $\frac{99n}{100}$

$$T.C = \log_{\frac{100}{99}} n \cong \log n$$

We can say that the base of log does not matter as it only a matter of constant.

Ans-18

(a) $100 \log \log n \quad \log n \cdot \sqrt{n} \quad n \log n! \quad n \log n \quad n^2$

$2^n \cdot 2^{n/4} \quad n!$

(b) $\log \log n \quad \sqrt{\log n} \quad \log n \quad 2 \log n \quad \log_{2n} \cdot n \quad 2n \quad n$

$\log n! \quad n \log n \quad n^2 \quad 2 \cdot (2^n) n!$

(c) $\log_5 n \cdot \log_2 n \quad 5n \quad \log n! \quad n \log n! \quad 8 \log_2 n \cdot$

$5n^2 \quad 7n^3 \quad 8^n \cdot n!$

Ans-19 Linear search (array, key)

for i in array

if value = = key

return i

Ans-20

Iterative insertion sort:-

insertion sort. (arr, n)

for from i = 1 to i = n-1

- Pick element arr [i] and insert it into sorted sequence arr [0 - - - i-1]

Recursive insertion sort

insertion sort. (arr, n)

{

if n <= 1

return

recursively sort n-1 elements

insertion sort. (arr, n-1)

Pick last element arr [i] and insert it into sorted sequence arr [0 - - - i-1]

}

insertion sort considers one input element per iteration. - Then produces a partial solution with all considered - every but more elements.

∴ It is called online sorting algorithm.

Aug- 20/11/22

Considering only 3 sorting now, as we got letters till now:

Algo	Best	Average	Worst	S.C	Stable	In place	order
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✓	X
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✓	✓
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	X	✓	X

Q-23 Binary search

A ← sorted Array
n ← size of Array
x ← Value to be searched

while x not found

if upper bound < lower:
    Exit :: X does not exist
    set mid part = lowerbound + (upper bound - lower bound)/2

    if A[mid part] < x
        lower bound = mid part + 1

    if A[mid part] > x
        upper bound = mid part - 1

    if A[mid part] = x
        Exit :: x found at mid part

	Time complexity	space complexity
Linear	$O(n)$	$O(1)$
Binary search (Recursive)	$O(\log n)$	$O(\log n)$
Binary Search (Iterative)	$O(\log n)$	$O(1)$

Ans 2 here $T(n) = T(n/2) + C$