Design and Analysis of Algorithms

NAME : AKSHIT SINGH

SECTION : F

ROLL No: 42

COURSE: B.Tech CSE

Tutorial-2

1. What is the time complexity of below code and how?

```
void fun(int n)
{
    int j = 1, i = 0;
    while (i<n){
      i = i+j;
      j++; }}
```

Sol →

$$\left.\begin{array}{ll} j=1 \; ; & i=1 \\ j=2 & i=1+2 \\ j=3 & i=1+2+3 \end{array}\right] \text{m-level}$$

for (i)

$\therefore$ $1+2+3+\ldots+ < n$

$\therefore$ $1+2+3+m < n$

$\therefore$ $\frac{m(m+1)}{2} < n$

$m = \sqrt{n}$

By summation method

$\Rightarrow \sum_{i=1}^{m} 1 \Rightarrow 1+1+\ldots\ldots+\sqrt{n}$ times

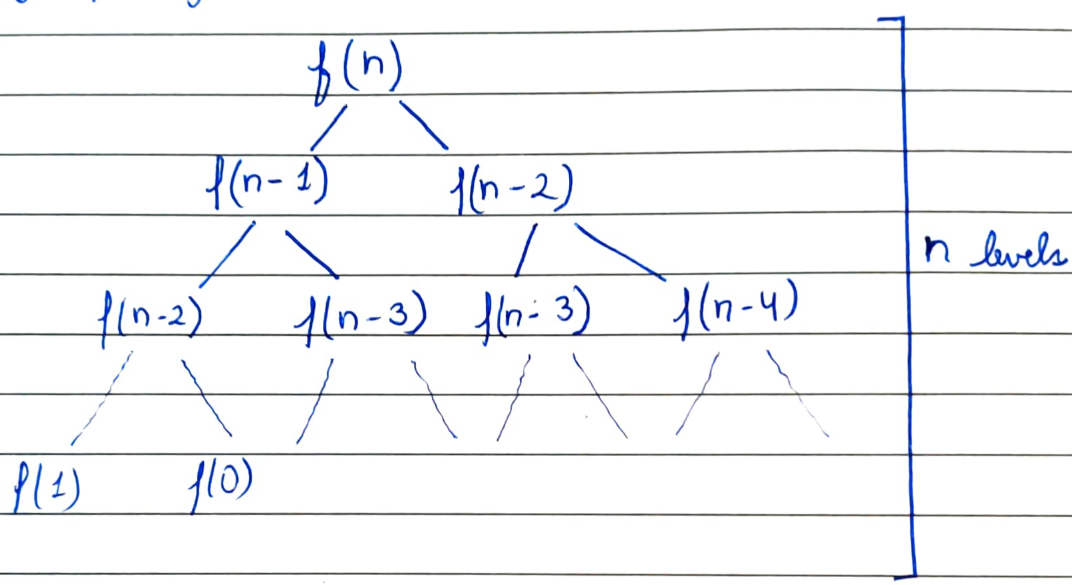
$\boxed{T(n) = \sqrt{n}}$ → Ans

2. Write recurrence relation for the recursive function that prints Fibonacci series. Solve the recurrence relation to get time complexity of the program. What will be the space of this complexity & why?

Sol → For Fibonacci Series

$f(n) = f(n-2) + f(n-2)$ $\qquad f(0) = 0$

By forming a tree $\qquad f(1) = 1$



∵ At every function call we get 2 function calls

∴ for n levels

We have = $2 \times 2 \ldots$ n times

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

MAXIMUM SPACE

Considering Recursive Stack:

no. of calls maximum = n

For each call we have space complexity $O(1)$

$$\boxed{\therefore T(n) = O(n)}$$

Without considering Recursive stack:
each call we have time complexity O(1)

$\boxed{\therefore T(n) = O(1)}$

3. Write programs which have complexity:
n(log n), n³, log(log n)

1) n log n → Quick Sort

```
void quicksort (int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition (arr, low, high);
        quicksort (arr, low, pi - 1);
        quicksort (arr, pi+1, high);
    }
}

int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1)
    for (int j = low ; j <= high - 1 ; j++)
    {
        if (arr[i] < pivot)
        {
            i++;
            swap( & arr[i], & arr[j])
        }
    }
    swap( & arr [i+1], & arr [high]);
    return (i + 1);
}
```

2) n^3 → Multiplication of 2 square matrix

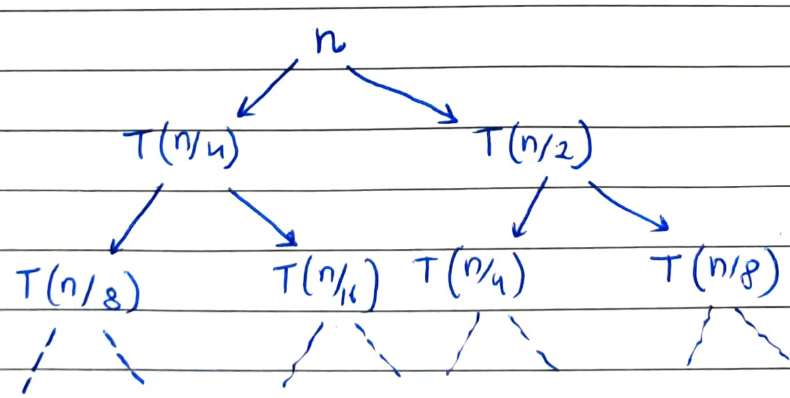
```
for(i=0; i<r ;i++)
  for (j=0; j<c2 ; j++)
    for (k=0; k<c1 ; k++)
    {
        res[i][j] += a[i][k] * b[k][j];
    }
```

3) $\log(\log x)$

```
for (i=2; i<n ; i = i*i)
  {
      count++;
  }
```

4) Solve the following Recurrence Relation

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



At level

$0 \rightarrow Cn^2$

$1 \rightarrow \frac{n^2}{4^2} + \frac{n^2}{2^2} = \frac{5Cn^2}{16}$

$2 \rightarrow \frac{n^2}{8^2} + \frac{n^2}{16^2} + \frac{n^2}{4^2} + \frac{n^2}{8^2} = \left(\frac{5}{16}\right)^2 n^2 C$

$\vdots$

$\text{max level} = \frac{n}{2^k} = 1$

$k = \log_2 n$

$$T(n) = c\left(n^2 + (5/16)n^2 + (5/16)^2 n^2 + \dots + (5/16)^{\log n} n^2\right)$$

$$T(n) = Cn^2\left[1 + \left(\frac{5}{16}\right) + \left(\frac{5}{16}\right)^2 + \dots + \left(\frac{5}{16}\right)^{\log n}\right]$$

$$T(n) = Cn^2 \times 1 \times \left(\frac{1 - (5/16)^{\log n}}{1 - (5/16)}\right)$$

$$T(n) = Cn^2 \times 11/5 \times \left(1 - \left(\frac{5}{16}\right)^{\log n}\right)$$

$T(n) = O(n^2 C)$

$O(Cn^2)$ Ans

5. What is the time complexity of following fun()?

```
int fun(int n){
  for (int i= 1; i<=n; i++){
    for (int j=1 ; j<n ; j+=i){
      // Some O(1) task
}}}
```

for	i	j
	1	1
	2	1 + 3 + 5
	3	1 + 4 + 7
	⋮	
	n	1 + 5 + 7

$\therefore j = (n-1)/i$ times

$$\sum_{i=1} \frac{(n-1)}{i}$$

$$\therefore T(n) = \frac{(n-1)}{1} + \frac{(n-1)}{2} + \frac{(n-1)}{3} + \dots + \frac{(n-1)}{n}$$

$$T(n) = n\left[1 + 1/2 + 1/3 + \dots + 1/n\right]$$

$$= n\log n - \log n$$

$$\underline{T(n) = O(n\log n)}$$

6. What should be time complexity of

```
for (int i=2; i<=n; i=pow[i,k])
{
    // Some O(1)
}
```

where k is a constant

→ for i

2^1

2^k

2^{k^2}

2^{k^3}

⋮

2^{k^m}

where

$2^{k^m} <= n$

$k^m = \log_2 n$

$m = \log k \log_2 n$

$\therefore \sum_{i=1}^{m} 1$

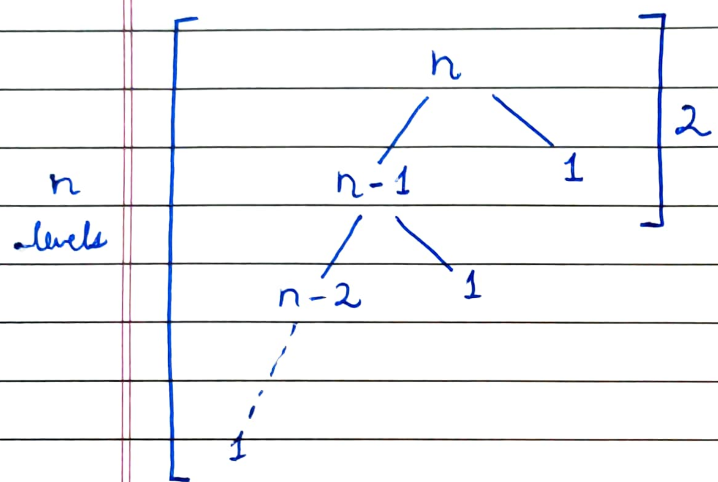
$1 + 1 + 1 + \ldots m$ times

$T(n) = O(\log_k \log n)$ → Ans

7. Write a recurrence Relation when quick sort repeatedly divides the array in to two parts of 99% and 1%. Derive the time complexity in this case. Show the recursion tree while deriving time complexity and find the difference in height of both the extreme parts. What do you understand by this analysis?

→ Given algorithm divides array in 99% and 1% part

$\therefore T(n) = T(n-1) + O(1)$



'n' work is done at each level

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

$= n \times n$

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

Lowest height $= 2$

highest height $= n$

$\boxed{\%\ \text{difference} = n-2}$ $n > 1$

The given algorithm produces linear result

8. Arrange the following in increasing order of Rate of growth

a) $n, n!, \log n, \log\log n, \text{root}(n), \log(n!), n\log n, \log^2(n), 2^n, 2^{2^n}, 4^n, n^2, 100$

$\rightarrow 100 < \log\log n < \log n < (\log n)^2 < \sqrt{n} < n < n\log n < \log(n!) < n^2 < 2^n < 4^n < 2^{2^n}$

b) $2(2^n), 4n, 2n, 1, \log(n), \log(\log(n)), \sqrt{\log(n)}, \log 2n, 2\log(n), n, \log(n!), n!, n2, n\log(n)$

$\rightarrow 1 < \log\log n < \sqrt{\log n} < \log n < \log 2n < 2\log n < 2\log n < n < n\log n < 2n < 4n < \log(n!) < n^2 < n! < 2^{2^n}$

c) $8^{2n}, \log_2(n), n\log_6(n), n\log_2(n), \log(n!), n!, \log_8(n), 96, 8n2, 7n3, 5n$

$\rightarrow 96 < \log_8 n < \log 2n < 5n < n\log_6(n) < n\log_2 n < \log(n!) < 8n^2 < 7n^3 < n! < 8^{2n}$