DESIGN AND ANALYSIS OF ALGORITHM

TUTORIAL-2

Name - DEEPTI PANDEY
SECTION - CST SPL1
CLASS ROLL NO - 21
UNIVERSITY ROLL NO - 2016724.

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++;
  }
}
```

for $i=0$, $i=1$, 2

for $i=1$, $i=3$, $j=3$

$i=3$, $i=6$, $j=4$

$i=6$, $i=10$, $j=5$

⋮

$i=n$, $i=n+j$, $j=j+1$

n

$\sum$

$i=0$

for $i=1$ $(i=0+1)$

for $i=3$ $(i=1+2)$

for $i=6$ $(i=1+2+3)$

⋮

n^{th} time $i=x^2<n$

$\Rightarrow x=\sqrt{n}$

(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 complexity of this program and why? .

Recurrence Relation for Recursive function that prints Fibonacci Series :

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

$$T(n) = T(n-1) + T(n-2) + 1 \quad \text{——} \; (1)$$

putting $T(n-1) = T(n-2)$ in (1),

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

$$= 2 * T(n-1) + 1 \quad \text{——} \; (2)$$

Now, $T(n) = 2 * (2 \times T(n-2) + 1) + 1$

$$= 4 \times T(n-2) + 3 \quad \text{——} \; (3)$$

Now, putting $T(n-2) = 2 \times T(n-3) + 1$

$$T(n) = 2 \times [2 \times [2 \times T(n-3) + 1] + 1] + 1 \quad \text{——} \; (4)$$

$$T(n) = 8T(n-3) + 7 \quad \text{——} \; (5)$$

$$T(n) = 2^k T(n-k) + 2^k - 1 \quad \text{——} \; (5)$$

let $n - k = 0 \Rightarrow k = n$

Now, putting $k = n$ in (5),

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

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

$$T(n) = 2^n + 2^n - 1 \Rightarrow T(n) = O(2^n)$$

The space complexity for this is $O(n)$ as even though the function calls are taking place recursively but they are actually being executed sequentially.

(3) Write programs which have complexity - ~~[illegible]~~ $n(\log n)$, n^3, $\log(\log n)$.

program with time complexity $n(\log n)$ -

```
a=n;
while (a>0)
{ b=a;
  while(b>0)
  { b=b/2;
  }
  a=a-1;
}
```

program with time complexity n^3 :

```
int arr[n1][n2][n2];
for(int i=0; i<n1; i++)
{ for(int j=0; j<n2; j++)
  { for(int k=0; k<n2; k++)
    { printf(arr[i][j][k];
    }
  }
}
```

program with time complexity $\log(\log n)$

```
for (int i=2; i<n; i=pow(i, k))
{ // some O(1) expressions
```

as satements.

}

④ Solve the following recurrence relation $T(n) = T(n/4) + T(n/2) + cn^2$.

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

It can be assumed that $T(n/2) >= T(n/4)$

Now, $T(n) <= 2T(n/2) + cn^2$

$T(n) >= cn^2$

$\Rightarrow T(n) >= O(n^2)$

$\Rightarrow T(n) = \Omega(n^2)$

As, $T(n) = O(n^2)$ and $T(n) = \Omega(n^2)$

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

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

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

Time Complexity of given function,

for, $i=1$, $j = 1, 2, 3, 4, \ldots\ldots, n$

for $i=2$, $j = 1, 3, 5, 7, \ldots\ldots, n/2$

$\vdots$

for $i=n$, $j = 1, n+1, \ldots\ldots$

$$\sum_{j=n}^{1} n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \ldots\ldots +$$

$$= \sum_{j=n}^{1} n\left[1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots\ldots + \frac{1}{n}\right]$$

$$= \cancel{\sum_{j=n}^{1} n\left[1 + \frac{1}{2} + \frac{1}{3}\right.}$$

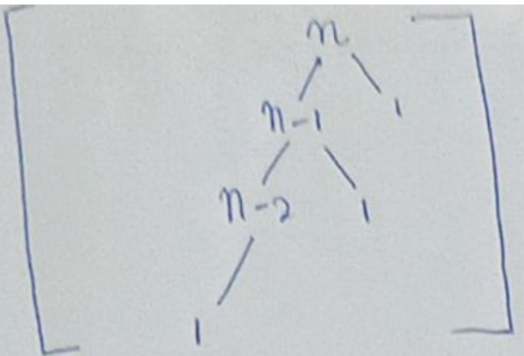
$$= \sum_{j=n}^{1} n[\log n]$$

$$\Rightarrow T(n) = O(n\log n)$$

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

array is divided as 99% and 1%

So, Recurrence Relation, $T(n) = T(n-1) + O(1)$



← Recursion Tree

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

lowest height = 2   highest height = n

difference between the two = n-2   n>1

This gives us linear result.

(8) Arrange the following in increasing order of rate of growth.

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

100 < loglogn < $\log^2(n)$ < log(n) < log(n!) < nlogn < root(n) < n < n! < 2^n < n^2 < 4^n < 2^{2^n}

(b) $2(2^n)$, 4n, 2n, 1, log(n), log(log(n)), $\sqrt{\log(n)}$, log2n, 2log(n), n, log(n!), n!, n^2, nlog(n).

1 < log(log(n)) < $\sqrt{\log(n)}$ < logn < log2n < 2logn < n! < log(n!) < nlog(n) < n < 2n < 4n < n^2 < $2(2^n)$

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

⑥ What should be the time complexity of following fun()?

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

where K is a constant.

for $i = 2, 2^k, \cancel{2k^2, 2k}\ 2^{k^2}, 2^{k^3}, \ldots, 2^{k^i}$

So, $2^{k^i} = n$

Taking log,

$\log_2 2^{k^i} = \log n$

$\Rightarrow k^i = \log n$

Taking log with base k,

$i \log_k k = \log(\log n)$

$\Rightarrow i = \log(\log n)$

$\therefore$ Time Complexity $= \log(\log n) * (O(1))$

$= \log(\log n)$