Tutorial-2

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

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

→ values after execution;

1st time → $i=1$

2nd time → $i=i+2$

3rd time → $i=1+2+3$

for i^{th} time → $i = (1+2+3+\cdots+i) < n$

$$\Rightarrow \frac{i(i+1)}{2} \; n \Rightarrow i^2 < n$$

$$\Rightarrow i < \sqrt{n}$$

Time complexity $= O(\sqrt{n})$

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?

→
```
int fib(int n){
    if (n<=1)
        return n;
        return fib(n-1) + fib(n-2);
}
```

Recurrence Relation →

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

Let $T(n)$ be the time complexity of $F(n)$. In $f(n-1)$ & $F(n-2)$ time will be $T(n-1)$ and $T(n-2)$, we have one more addition to sum our resuets.

for $n>1$

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

for $n=0$ & $n=1$, no addition occurs $\Rightarrow T(0) = T(1) = 0$

Let $T(n-1) \simeq T(n-2)$ — ②

Adding eq ② in ①

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

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

using backward Substitution

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

$T(n) = 2(2T(n-2)+1)+1 = 4T(n-2)+3$

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

$T(n) = 8T(n-3)+7$

$\Rightarrow T(n) = 2^k + (n-k) + (2^k - 1)$ — ③

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

eq^n ⑤ $\Rightarrow T(n) = 2^n + (0) + 2^n - 1$

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

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

Space Complexity = $O(n)$

Reason → The func. calls are executed sequentially. Sequentially execution guarantees that the stack size will never exceed the depth of calls for first f(n-1) it will create N stack.

3. WAP which have complexity - n(logn), n^3, log(log n).

→ i) O(n log n) -

```
#include<iostream>
using namespace std;
int partition (int arr[], int s, int e){
    int pivot = arr[s];
    int count = 0;
    for (int i=s; i<=e; i++){
        if (arr[i] <= pivot)
            count++;
    }
```

```
int pivot-ind = s+ count;
swap (arr [pivot - ind], arr [s]);
int i=s, j=e;
while (i< pivot- ind && j> pivot-ind){
    while (arr[i]<= pivot)
        i++;
    while (arr[j] > pivot)
        j--;
    if (i< pivot - ind && j> pivot - ind)
        swap (arr[i++], arr [j--]);
    }
    return pivot- ind ;
}
void quick (int arr [], int s, int e ){
    if (s== e)
        return;
    int p= partition (arr, s, e);
    quicksort (arr, s, p-1);
    quicksort (arr, p+1 , e);
}
int main (){
    int arr [] = {6, 8, 5, 2, 1}
    int n= 5;
    quick sort (arr, 0, n-1);
    return 0;
```

(ii) O(n3)

```
int main (){
  int n= 10;
  for (int i=0; i<n; i++){
     for(int j=0; j<n; j++){
        for (int K=0; K<n; k++){
           printf (" #");

   } }}
   return 0;
}
```

(iii) $O(\log(\log n))$

```
int countprime (int n){
    if (n< 2)
        return 0;
    bool* non-prime = new bool [n];
    non-prime [1]= true;
    int num-non-prime =1;
    for (int i = 2; i<n; i++){
        if (non-prime [i])
            continue;
        int j= i * 2;
        while (j<n){
            if (! non-prime [j]){
                non-prime[j] = true;
                num-non-prime ++;
            }
            j+ = i;
        }
    }
    return (n-1) - num-non-prime;
}
```

4. Solve the following recurrence relation

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

→ using master's theorem -

Assume $(T(n/2) >= T(n/4))$

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

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

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

Also, $T(n) >= cn^2 \Rightarrow T(n) >= \Theta(n^2)$

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

$\because T(n) = O(n^2)$ & $T(n) = \Omega(n^2)$

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

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
}}}
```

→ for i=1, Inner loop is executed n times.

for i=2, inner loop is executed n/2 times.

for i=3, Inner loop is executed n/3 times.

It is forming a series -

$n + n/2 + n/3 + \cdots + n/n$

$n[1 + 1/2 + 1/3 + \cdots + 1/n]$

$\Rightarrow n \times \sum_{k=1}^{n} \frac{1}{k} \Rightarrow n \times \log n$

Time Complexity = $o(n \log n)$

6) What should be the time complexity of

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

where k is a constant

→ with iterations -

i takes values

for 1st iteration → 2

for 2nd iteration → 2^k

for 3rd iteration → $(2^k)^k = 2^{k^2}$

for n iteration → $2^{k^{\log_k(\log(n))}}$

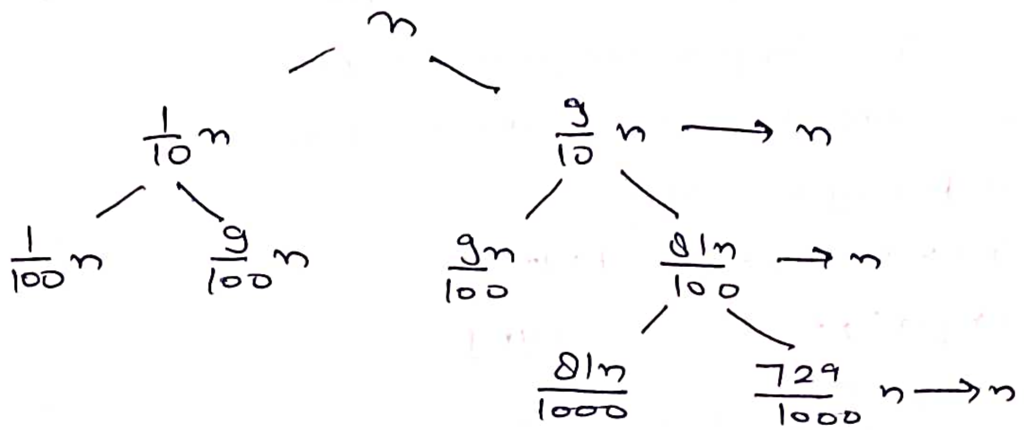
∵ last term must be less than or equal to n.

So, there are in total $\log_k(\log(n))$ many iterations & each iteration takes a constant time to run.

∴ Time complexity = $o(\log(\log(n)))$

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

→



If we split in this manner

Recurrence relation:

$$T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + O(n)$$

when first branch is of size $9n/10$ and second one is $n/10$.

Showing the above using recursion tree approach

Calculating values.

At 1st level, value $= n$

At 2nd level, value $= \frac{9n}{10} + \frac{n}{10} = n$

Value remains same at all levels i.e. n

Time complexity = summation of values

$\Rightarrow O(n \log n)$ [upper bound]

$\Rightarrow \Omega(n \log_{10} n)$ [lower bound]

$\Rightarrow O(n \log n)$

Q) Arrange the following in increasing order of rate of growth.

a) n, n!, logn, log logn, root(n), log(n!), nlogn, log^2(n), 2^n, 2^(2^n), 4^n, n^2, 100

→ $100 < \log(\log n) < \log n < \sqrt{n} < n < n\log n < \log^{2}(n) < \log(2n) < n2 < 2^n < n! < 4^n < 2^{(2^n)}$

b) 2(2^n), 4n, 2n, 1, log(n), log(log(n)), $\sqrt{\log(n)}$, log2n, 2 log(n), n, log(n!), n!, n2, nlog(n)

→ $1 < \log(\log n) < \sqrt{\log n} < \log n < 2\log n < \log(2n) < n < n\log n < \log\sqrt{n} < 2n < 4n < n2 < n! < 2(2^n)$

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

→ $96 < \log_8 n < n\log_6 n < \log_2 n < n\log_2 n < \log(n!) < 5n < 8n2 < 7n2 < n! < 8^{2n}$