Date.
Page No.

Tutorial - 2

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

→ values after execution

1^{st} time → $i = 1$

2^{nd} time → $i = 1+2$

3^{rd} time → $i = 1+2+3$

4^{th} time → $i = 1+2+3+4$

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

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

$$\Rightarrow i^2 < n$$

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

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

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

Date.
Page No.

* Recurrance Relation

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

Let $T(n)$ denote the time complexity of $F(n)$.

In $F(n-1)$ and $F(n-2)$ time will be $T(n-1)$ and $T(n-2)$. We have one more addition to sum our results.

For $n>1$

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

For $n=0$ & $n=1$, no addition occurs

$$\therefore T(0) = T(1) = 0$$

Let $T(n-1) \approx T(n-2) \quad — ②$

Addition of ② & ①

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

Using backward substitution

$$\therefore T(n-1) = 2 \times T(n-2) + 1$$
$$T(n) = 2 \times [2 \times T(n-2) + 1] + 1$$
$$= 4T(n-2) + 3$$

We can substitute

$$T(n-2) = 2 \times T(n-3) + 1$$
$$T(n) = 8 \times T(n-3) + 7$$

General equation -

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

Date.
Page No.

for $T(0)$

$n-k=0 \rightarrow k=n$

Substituting values in ⑤

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

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

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

Space complexity $\rightarrow O(N)$

Reason - The function calls are executed sequentially. Sequential execution guarantees that the stack size will never exceed the depth of calls for first $F(n-1)$ it will create N stack

3. (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 ++;
```

Date.
Page No.

```
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;
    quicksort (arr, 0, n-1);
    return 0;
}
```

Date.
Page No.

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 CountPrimes (int n){
    if (n<2)
        return 0;
    bool * non_prime = new bool [n];
    non_prime [1] = true;
    int numNonPrime = 1;
    for (int i=2; i<n; i++){
        if (nonPrime [i])
            continue;
        int j = i * 2;
        while (j<n) {
            if (! nonPrime [j]) {
```

Date.
Page No.

```
            nonPrime [j] = true;
            numNonPrime ++;
         }
         j += i;
      }
   }
   return (n-1) - numNonPrime;
}
```

4. $T(n) = T(n/4) + T(n/2) + Cn^2$

using master's Theorem -

we can assume $T(n/2) >= T(n/4)$

Equation can be rewritten as

$T(n) <= 2T(n/2) + Cn^2$

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

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

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

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

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

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

5.

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

Date.
Page No.

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 :-

$$\Rightarrow n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}$$

$$\Rightarrow n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right)$$

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

$$\Rightarrow n \times \log n$$

Time complexity = $O(n\log n)$

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

with iterations -
i take values
for 1st iteration → 2
for 2nd iteration → 2^k
for 3rd iteration → $(2^k)^k$
|
|
for n iteration → $2^{k \log_k (\log(n))}$

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

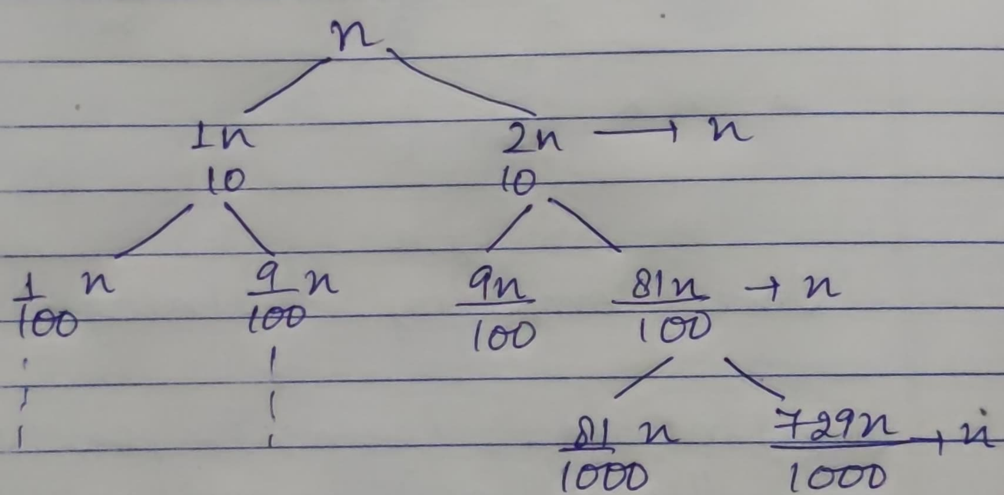
$$2^{k \log k (\log(n))} = 2^{\log n} = n$$

Each iteration takes constant times

∴ Total iteration = $\log k(\log(n))$

Time complexity - $O(\log(\log(n)))$

7.



If we split in this manner

Recurrance relation

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

When first branch is of size $9n/10$ & second one is $n/10$.

Showing the above using recursion tree approach calculating values.

Date.
Page No.

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 value
$= O(n \times \log\log n)$ (upper bound)
$= \Omega(n \log_{10} n)$ (lower bound)
$\rightarrow \boxed{O(n \log n)}$

8. a) $100 < \log(\log n) < \log(n) < \sqrt{n} < n < n\log(n)$
$< \log^{\wedge}2(n) < \log(\lfloor n) < n^2 < 2^n <$
$\lfloor n < 4^n < 2^{2^n}$

b) $1 < \log(\log(n)) < \sqrt{\log(n)} < \log(n) <$
$2\log(n) < \log(2n) < n < n\log(n) <$
$\log(\sqrt{n}) < 2n < 4n < n^2 < \lfloor n < 2(2^n)$

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