Date:
Page No.

TUTORIAL 2

Q1. Void fun (int n) {
int j=1, i=0;
while (i<n) {
i = i+j;
j++;
}
}

Values after execution
1st time → i = 1
2nd time → i = 1+2
3rd time → i = 1+2+3
4th time → i = 1+2+3+4
For i time → i = (1+2+3+...i) < n

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

$\Rightarrow i^2 < n$

$i = \sqrt{n}$

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

(ii) O (log (log n)).
int countPrimes (int n) {
if (n<2) return 0;
boolean[] notPrime = new boolean[n];

Date.
Page No.

Ans 2.

```
nonprime[1] = true;
int numnonprimes = 1;
for(int i=2; i<n; i++){
    if(nonprime[i]) continue;
  int j = i*2;
  while(j<n){
      if(!nonprime[j]){
          nonprime[j] = true;
      numnonprime++;
      }
      j += i;
  }
}
return (n-1) - numnonprime;
}
```

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

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

$\Rightarrow n\left(1 + 1/2 + 1/3 \cdots 1/n\right)$

$\rightarrow n \times \sum_{R=2}^{n} \frac{1}{R}$

$\Rightarrow n \times \log n$

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

Date.
Page No.

Ans 2. Recurrance Relation

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

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

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

for $n \geq 1$.

$T(n) = T(n-1) + T(n-2) + 1$ — ①

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

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

Let $T(n-1) \approx T(n-2)$ — ②

Putting ② in ①

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

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

~~2 x~~.

Using backward substitution

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

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

$= 4 \times T(n-2) + 3$

We can substitute $T(n-2) = 2 \times T(n-3) + 1$

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

General Equation

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

For $T(0)$

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

substituting values in (2)

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

$2^n + 2^n - 1$

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

Space complexity → $O(N)$

Reason → The 2^n calls are execute sequentially. Execution guarantees that the stack size will not exceed the depth of calls. for first $F(n-1)$ it will execute N stack frames, the other $F(n-2)$ will execute $N/2$. so the largest is N.

Ans 3. $O(n \log n)$ -

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

```
int pivot_ind = start + count - 1;
swap(&arr[pivot_ind], &arr[start]);
  int i = start, j = end;
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 start, int end) {
  if (start >= end) return;
  int p = partition(arr, start, end);
  quicksort(arr, start, p-1);
  quick_sort(arr, p+1, end);
}
int main()
{
  int arr[] = { 6, 8, 5, 2, 1 }
  int n = 5;
  quick sort(arr, 0, n-1);
  return 0;
}
```

(ii) $O(n^3)$ -

```
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+1){
        printf("*");
    }
}
}
return 0;
}
```

Ans 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

$T(n) <= 2T(n/2) + cn^2$
$T(n) < O(n^2)$
$T(n) = O(n^2)$
Also $T(n) >= cn^2 \Rightarrow T(n) >= O(n^2)$
$\Rightarrow T(n) = \Omega(n^2)$
$T(n) = O(n^2)$ & $T(n) = \Omega(n^2)$
$T(n) = O(n^2)$ Ans

Ans 5

Ans 6

Date.
Page No.

Ans

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

for $i=1$, inner loop is executed n times
for $i=2$, " " " $n/2$ "
for $i=3$, " " " $n/3$ "

It is forming a series -

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

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

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

$\Rightarrow n \times \log n$

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

Ans 6

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

n^2)

Date.
Page No.

With iterations :-

i take values

for 1st iteration → 2

for 2nd " → 2^k

for 3rd " → $(2^k)^k$

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

∵ last term must be less than or equal to n

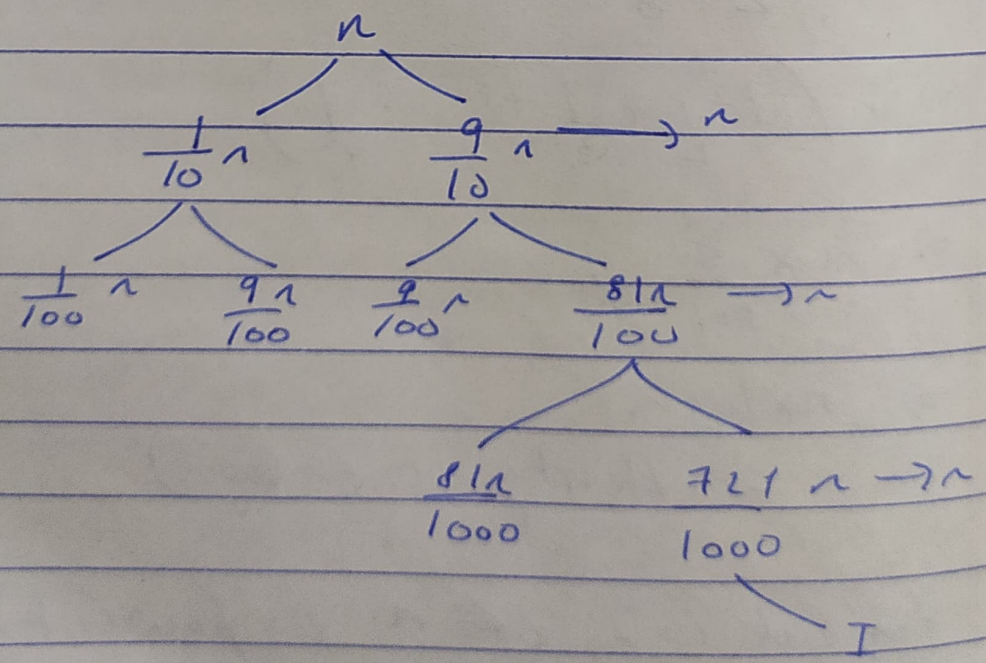
$2^{k \log_k (\log(n))} = 2^{\log n} = n$

Each iterat^n takes constant times

∴ Total iteration = $\log_k (\log(n))$

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

7)



Page No.

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 $9n/10$ & second one is $n/10$. Solving 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

$= O(n \times \log_{10/9} n)$ (upper bound)

$= \Omega(n \log_{10} n)$ (lower bound)

$= O(n \log n)$