Date.
Page No.

Tutorial 2

Q1

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

values after evolution

1st time → $i=1$

2nd time → $i=1+2$

3rd time → $i=1+2+3$

4th time → $i=1+2+3+4$

ith time → $i = (1+2+3+4 \cdots i) < n$

$\Rightarrow i(i+1)/2 < n$

$\Rightarrow i^2 < n \Rightarrow i - \sqrt{n}$

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

Q2 Recurrence 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 $T(n-1)$ & $T(n-2)$

$T(n) = T(n-1) + T(n-2) + 1$ — (1)

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

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

Let $T(n-1) \simeq T(n-2)$

Putting 2 in 1 we get

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

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

Using backword sub

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

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

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

We can sub $T(n-2) = 2T(n-3)+1$

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

General equation

$T(n) = 2^k T(n-k) + (2^k - 1)$ — (3)

For $T(0)$, $n-k=0$, $k=n$

Sub value in 3

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

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

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

Space complexity $= O(N)$.

Reason:- The function calls are executed sequentially. The depth of calls for first F(n-1) it will make N stack frames the other F(n-2) will create N/2 so the largest is N

Q3: O(n logn)

```
#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;
        swap(arr[pivotind], arr[start]);
          int i = start, j = end;
        while(i < pivotind && j > pivotind)
        {
             while(arr[i] <= pivot) i++
          }
         while(arr[j] > pivot) j--;
         if(i < pivot_ind && j > pivotind)
              swap(arr[i++], arr[j--]);
         }
     }
        return pivotind;
  }
```

Date.
Page No.

```
void quick(int arr[], int start, int
                         end)
    if (start >= end) return;
    int p = partition(arr, start, end)
    quick (arr, start, p-1);
    quick (arr, p+1, end);
}
```

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++)
              printf("*");
    }
```

O(log(log(n)))

```
int countPrime(int n)
{
    if(n<2) return 0;
    bool[] nonprime = new bool[n];
    nonprime[1] = true;
```

Date.
Page No.

```
int numNonPrime = 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;
}
```

Q4 $T(n) = T(n/4) + T(n/2) + cn^2$

Using master's theorem

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

Eq can be rewritten as

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

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

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

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

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

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

Date.
Page No.

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

Q5 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

$n + n/2 + n/3 + \dots + n/n$

$n(1 + 1/2 + 1/3 + \dots 1/n) \Rightarrow$

$\Rightarrow n \sum_{K=2}^{n} 1/k$

$= n \log n$

$= O(n \log n)$

Q6 for (int i=2, i <= n; i= pow(i,k))
{
}

with iterations
i take values
for 1 iteration = 2
for 2 iteration = 2^k
for 3 iteration $\Rightarrow (2^k)^k$
for n iteration $\Rightarrow 2^{k(\log k \log(n))}$
last term must be less than equal to n

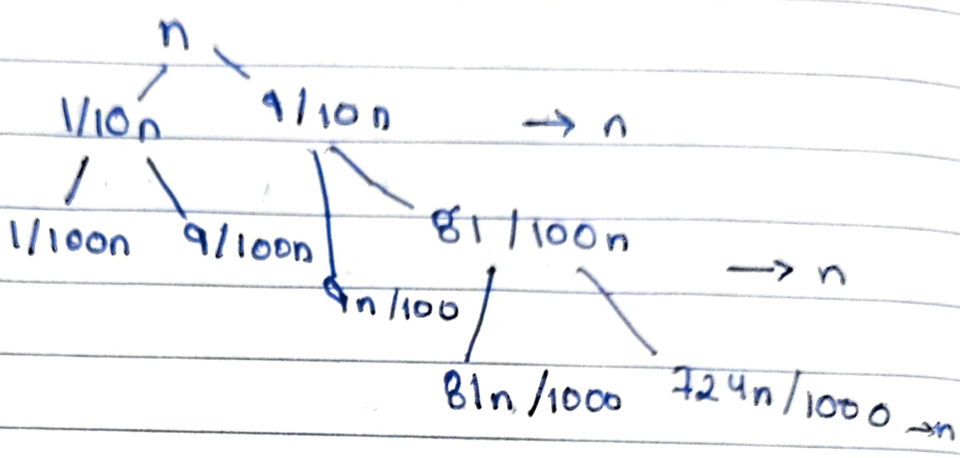
$2^{k \log \log(n))} = 2^{\log n} = n$

10

Each iteration takes constant time
∴ Total iteration = $\log_k \log(n)$

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

Q7



If we split in this manner
recurrence relation = $T(n) = T(9n/10) + T(n/10) + O(n)$
First branch is of size $9n/10$ & second one is $n/10$.
Solve the above using recursion tree approach calculating values.
At 1st level, value = n
At 2nd level, value = $9n/10 + n/10 = n$
Values remain same at all levels i.e
Time complexity = summation of all values

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

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

$\Rightarrow O(n \log n)$