Tutorial 2

Ques 1

```
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 ith time → $i=(1+2+3+....i)<n$

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

$$= i^2 < n$$

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

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

Ques 2

Recurrence Relation

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

let $T(n)$ denote the time complexity of $f(n)$

for $f(n-1)$ and $f(n-2)$ time will be $T(n-1)$ & $T(n-2)$. we have one more addition to sum & result. for $n>1$

$T(n) = T(n-1) + T(n-2) + 1$ — 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

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

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

using Backward Subsitution

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

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

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

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

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

General Equation -

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

for $T(0)$

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

Subsituting values in ③

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

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

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

Space complexity $= O(N)$

Reason -

The function calls are execute sequentially. This ~~seo~~ execution ~~gaat~~ guarantees that the stack size will exeed the depth of calls. for 1st f(n-1) it will creat N stack frames, the other F(n-2) will create N/2. So the longest is N.

Ques 3 (i) O(n log n) -

```
#include <iostream>.
using namespace std;
int partition (int arr[] ,int start, int end)
{
  int first = ar[start];
  int count = 0;
  for(int i = (start +1); i<=end; i++)
  { if (ar[i] <= pivot)
    count++;
  }
  int pivot_nd = start + count ;
  swap (ar[pivot_nd], ar [start]);
  int i = start, j = end;
  while (i < pivot_end && j > pivot ind)
  { while (arr[i] < = pivot )
    { i++;
    }
    while (ar[j] > pivot)
    {j--;
    }
```

```
if (i< pivot_int && j >pivot _int)
{
 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);
 quicksort (arr, p+1, end);
}
int main ()
{
 int arr[]={6,0,5,2,1};
 int n=5;
 quicksort (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(*x*); }}}
return 0; }
```

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

```
int Count Prime (int n)
{ if (n<2)
  return 0;
  boolean [] non prime = new boolean [n];
  non prime [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])
       { non prime [j] = true;
         numnonprime ++;
       }
       j+ = i;
     }
  }
  return (n-1) - numnonprime;
}
```

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

using Master's theorum

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

Equation can be rewritten as

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

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

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

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

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

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

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

Ques 6 for (int i=2); i<=n; i = pow(i, K))

{
// same O(1) expressions or statements
}

With iterations -

i take values

for 1^{st} iteration $\rightarrow 2$

for 2^{nd} iteration $\rightarrow 2^K$

for 3^{rd} iteration $\rightarrow (2^K)^K$

for n iterations $\rightarrow 2^{K \log_K(\log(n))}$

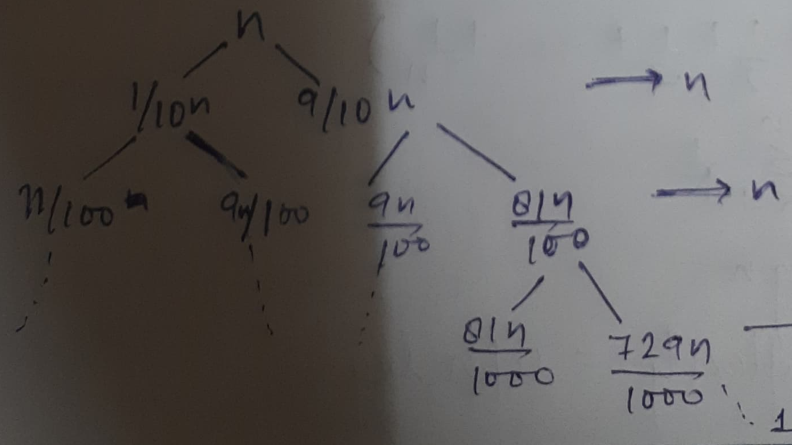
$\because$ last term must be less than or equal to n

$2^{K \log_K(\log(n))} = 2^{\log n} = n$

Each iterations takes constant time

$\therefore$ Total iterations $= O(\log(\log(n)))$

Ques 7



If we split in this Recurrence Relation → $T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + O(n)$

where 1st branch is of size $\frac{9n}{10}$ and second one is $\frac{n}{10}$

Solving the above using recursion tree approach calculating values

At 1st level, value $= n$

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

Values 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]

$\Rightarrow O(n \log n)$

Ques 5

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 $= O(n \log n)$

Ques 8

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

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

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