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 → = 1+2+3+4

for ith time → $i = (1+2+3+ \ldots +i) < n$

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

$$= i^2 < n$$

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

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

x———x

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) & f(n-2) time will be T(n-1) & T(n-2).

We have one more ~~[illegible]~~ 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

i. $T(0) = T(1) = 0$

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 Eq^n.

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

for $T(0)$

$n-k = 0 \implies k = n$

Subsituting Value in ③

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

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

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

Space Complexity $= O(n)$

Date: __/__/__
Page:

Reason :- The funⁿ calls are execute Sequentially. This execution guarantees that the stack size will exceed the depth of calls for 1st f(n-1) it will create N stack frames, the other f(n-2) will create N/2. So the longest in N.

Ques-3

i) **O (n log n) :-**

```
#include <iostream>
Using namespace std;
int partition [int arr[], int Start, int end)
{
    int first = arr[start];
    int count = 0;
    for (int i = (start + 1); i <= end ; i++)
    {
        if (ar[i] <= pivot)
        count ++;
    }
    int pivot_nd = Start + count ;
    Swap [arr[pivot - nd], ar[start]);
    int i = Start, j = end ;-
    while [ i < pivot_nd $$ j > pivot_nd )
    {
        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);
  quicksort (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) <u>O(N^3)</u> :-

```
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 ("*");  }}}
return0; }
```

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 num non primes = 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;
         num nonprime ++;
      }
        j += i;
     }
   }
   return (n-1) - num non prime;
}
```

Ques-4

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

Using Master's Method / theorem

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

Eqn can be rewritten as

$T(n) <= 2T(n/2) + n^2 \Rightarrow T(n) \cancel{<} = O(n^2)$

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

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

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

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

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

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

$\Rightarrow n + n/2 + n/3 + \ldots + n/n$

$= n(1 + 1/2 + 1/3 + \ldots + 1/n)$

$= n \times \sum_{k=1}^{n} 1/k$

$= n \times \log n$

$\Rightarrow$ Time Complexity $= O(n \log n)$

Ques - 6

```
for (int i=2; i<=n; i = pow(i,k) )
{
   // Some O(1) expression or statement.
}
```

with iteration —

i takes values

for 1st iteration $\rightarrow 2$

for 2nd iteration $\rightarrow 2^k$

for 3rd iteration $\rightarrow (2^k)^k$

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

∴ last term must be less than or equal to n

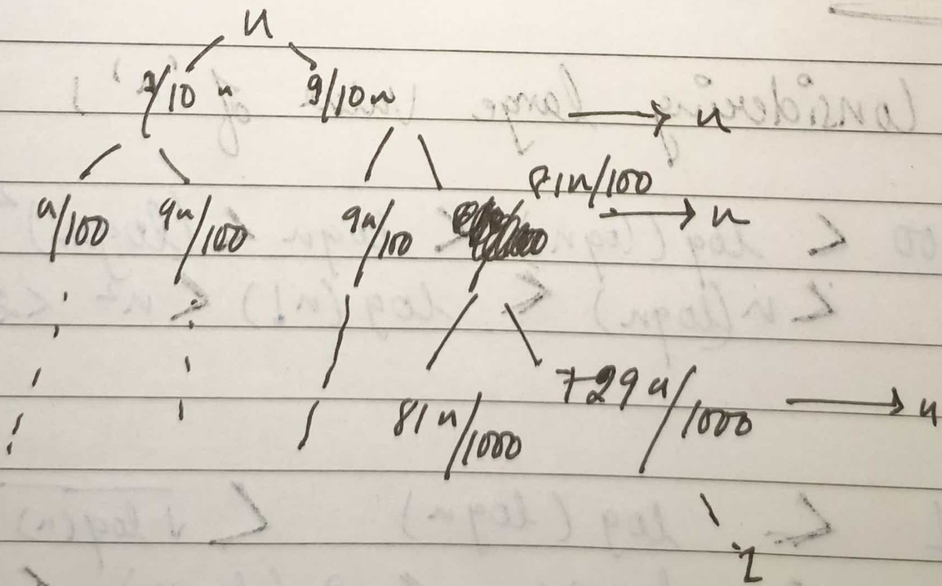
takes

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

Each iteration takes constant time

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

Ques - 7



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

where 1^{st} branch is of size $9n/10$ & second one is $n/10$

Solving the above using recursion tree approach
Calculating Values
At 1^{st} level, value $= n$
At 2^{nd} level, value $= 9n/10 + n/10 = n$

Value remains same at all levels i.e n
Time Complexity = Summation of values
$= O(n \times \log_{10/9} n)$ [upperbound]
$= \Omega(n \log_{10} n)$ [lower bound]
$\Rightarrow O(n \log n)$

Ques - 8

Considering large value of 'n',

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

*enlivo

Date: ___/___/___

Page:

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