Name - Shreya Murari
Section - D
Roll no. - 38

Design and Analysis of Algorithms

Tutorial - 2

1. What is the time complexity of below code and how?

```
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^{th} time → $i = (1+2+3+4+ \ldots i) < n$

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

$\Rightarrow i^2 < n$

$\Rightarrow i = \sqrt{n}$

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

Date
Page

Q2. Write recurrence relation for the recursive function that prints Fibonacci series. Solve the recurrence relation to get time complexity of the program. What will be the space complexity of this program and why?

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 $T(n-1)$ and $T(n-2)$. We have one more addition to sum our result

For $n > 1$

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

For $n=0$ and $n-1$, no addition occurs

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

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

Putting ② in ①

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

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

Using Backward substitution

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

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

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

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

Second equation :-

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

For $T(0)$

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

substituting value in 3

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

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

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

space complexity $\Rightarrow O(N)$

Reason :

The function calls are execute sequentially. sequential execution guarantees that the stack size will exceed the depth of calls. For $F(n-1)$ it will create N stack frames, the other $F(n-2)$ will create $N/2$, so the longest is N

Date
Page

Q3- Write program which have complexity - n (log n), n^3, log (log n)

(i) O(n log n) :-

```
# include <iostream>
using namespace std;

int partition( int w[], int start, int end)
  {
    int pivot = w [start];
    int count = 0;
for ( int i = start+1; i <= end; i++) {
  if (w[i] <= pivot) count ++;
 }
   int pivot_ind = start + count;
   swap (w [pivot_ind], w [start]);

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

```
void quick (int w[], int start, int end) {
 if ( start >= end)
     return ;
  int P = position(w , start , end) ;
  quicksort ( w, start P-1);
   quicksort  (w, P+1, end );
 }
int main()
{
   int w[] = { 6,8,5,2,1}
      int n = 5;
 quicksort (w , 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++) {
     printf ("*");
  }
  }  }
  return 0;
}
```

(iii) O(log(log n)) :-

```
int countPrimes (int n) {
 if (n>2)
   return 0;
boolean[] nonprime = new boolean[n];
  nonprime [1] = true;

int numPrimes = 1;
  for (int i=2; i<n; i++) {
   if (nonPrime [i] )
     continue;
     j = i * 2;
while (j <n) {

 if (! nonPrime [j ]) {
       nonprime [j] = true;
       numNonPrimes ++ ;
       }
        j += i ;
     }
      }
  return (n-1) - numNonPrime;
     }
```

Q4 Solve the following recurrence relation $T(n) = T(n/4) + T(n/2) + cn^2$

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

Using Master's Theorem

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

Equation can be rewritten as

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

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

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

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

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

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

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

Q5. What is the time complexity of following function fun()?

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

For $i = 2$, inner loop is executed n times.
For $i = 2$, inner " " " $n/2$ "
" $i = 3$, " " " " $n/3$ "

It is forming a series :-

$$n + \frac{n}{2} + \frac{n}{3} + \dots + \frac{n}{n}$$

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

$$\Rightarrow n \times \sum_{K=2}^{n} \frac{1}{K}$$

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

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

Date
Page

Q6. what should be the time complexity of
for (int i = 2; i <= n; i = pow (i, k))
{
// some O(1) expressions
}
where k is a constant

for (int i = 2 ; i<=n ; i = Pow (i, k)
{
// some O (1) express --
}
with iterations :-
i take values

for 1st iteration → 2
" 2nd " → 2^k
" 3rd " → $(2^k)^k$
for n iterations → $2^{k \log_k (\log n)}$
∵ last tin must be less than an equal to n

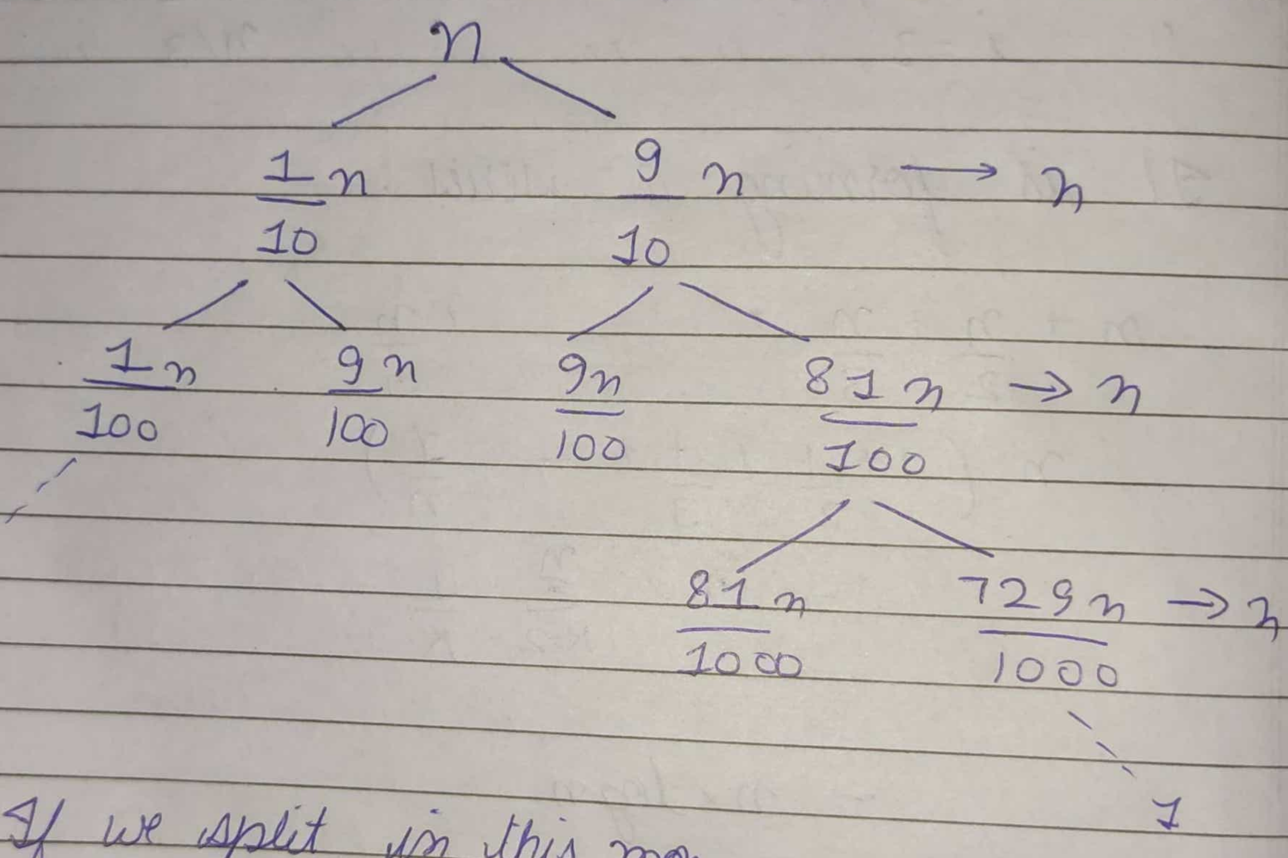
$$2^{k \log_k (\log(n))} = 2^{\log n} = n$$

Each iteration takes constant time

∴ Total iteration = $\log_k(\log(n))$

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

Q7. Write a recurrence relation when quick sort repeatedly divides the array in to two parts of 99% and 1%. Derive the time complexity in this case. Show the recursion tree while deriving time complexity & find the difference in heights of both the extreme parts. What do you understand by this analysis?



If we split in this man.

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

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

Solving the above using recursion tree approach
Calculating values

At 1^{st} level, value $= n$
At 2^{nd} 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)

$\Rightarrow O(n \log n)$ Ans