NAME - RADHIKA SAINI
SECTION - D
ROLL NUMBER - 31

DESIGN AND ANALYSIS OF ALGORITHMS
TUTORIAL - 2

Ques (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++;
    }}
```

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

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

$\Rightarrow i^2 < n$

$\Rightarrow i = \sqrt{n}$

Time Complexity = $O(\sqrt{n})$ Answer.

Page :
Date: / /

Ques (20) 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 $n > 1$

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

for $n=0$ and $n-1$, 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 substitution

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

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

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

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

for $T(0)$

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

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 executed sequentially, sequential execution guarantees that the stack size will exceed the depth of Calls. For $F(n-1)$ it will create N stack frame the other $F(n-2)$ will create $N/2$, so the longest is N

Page :
Date: / /

Ques (30) Write program which have Complexity - n[logn], n^3, log(logn)

(i) O(n logn) :-

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

int partition (int w[], int start, int end)
{
    int pivot = w[start];
    int count = 0;
for (int i= start; 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_int && j > pivot_int) {
    swap(w[i++], w[j--]);
}}
return pivot_int;
}
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 j=2; i<n ; i++) {
        if (nonPrime[i])
            Continue;
            j = i+2;
    while (j<n) {
    if (! nonPrime [j])
        {   nonPrime [j] = true;
            numNonPrime ++;
            }
                j+=i;
        }
    }
    return (n-1) - numNonPrime;
    }
```

Page :
Date: / /

Ques (4) 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 :-

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

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

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

Also,

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

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

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

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

Page :
Date: / /

Ques (5) What is the time Complexity of following function func() ?

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

for i=2, 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 :-

$$n + n/2 + n/3 + \text{-----} + n/n$$

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

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

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

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

Ques⑥ 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=2 ; i<=n ; i = pow(i,k)
{
    // some O(1) express---
}
```

with iterations

i' take values

for 1st iteration → 2
for 2nd iteration → 2^k
for 3rd iteration → $(2^k)^k$
for n iterations → $2^k \log_k(\log n)$ /

∵ last it in must be less than or equal to n

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

Each iteration takes constant time

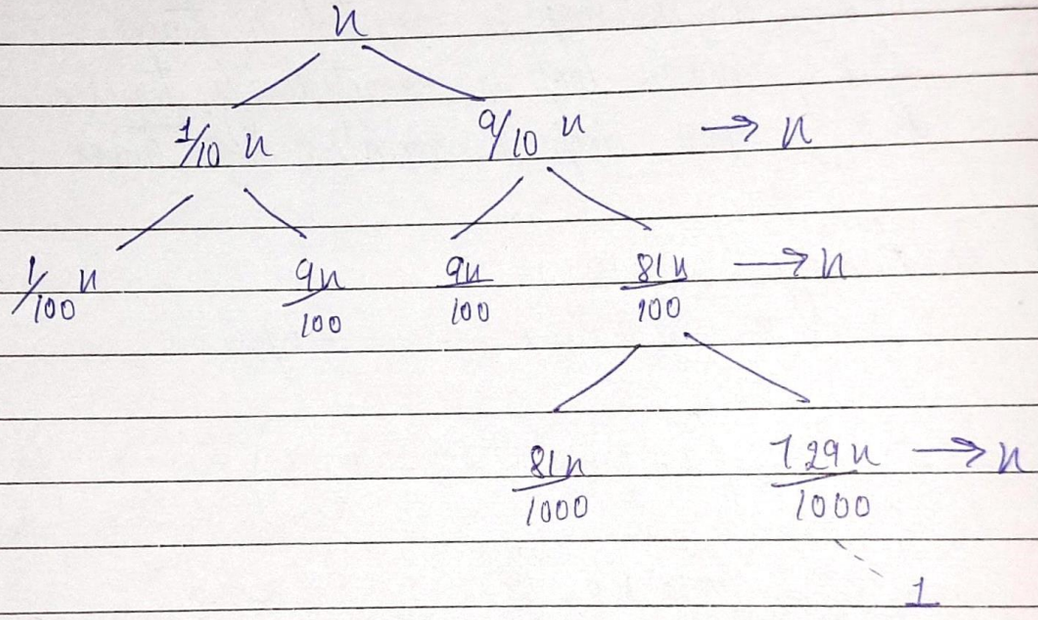
∴ Total iteration = $\log_k(\log(n))$

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

Page :
Date : / /

Ques (9) Write a recurrence relation when quick sort repeatedly divided 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 and find the difference in height of both the extreme parts. What do you understand by this analysis?

→



If we split in this.

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 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)

= $O(n \log n)$ Answer