Neeraj Singh Bhandari 03 Sec-H

Tutorial-2

1.> Void fun (int n) {

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

1.> Value after execution incremented by 1, 2, 3, 4 ... until it becomes greater than or equal to n

The value of i $\frac{x(x+1)}{2}$ after x iteration

$i^2 < n$

$i = \sqrt{n}$

Time Complexity $= O\sqrt{n}$

2.> int fib (int n) {

```
if (n<=1)
  return n;
return fib(n-1)+ fib(n-2);
```

3

Recurrence Relation

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

Let T(n) denote the time complexity of F(n)

In f(n-1) and F(n-2) time will be T(n-1) and T(n-2). We have one more addition to sum over results.

For $n \geq 1$

$T(n) = T(n-1) + T(n-2) + 1$ — ①

For $n=0$ & $n=1$, no addition occur :- $T(0) = T(1) = 0$

Let $T(n-1) = T(n-2)$ — ②

Addition of ⑪ in ①

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

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

Using backward substitution

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

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

$\Rightarrow 4T(n-2) + 3$

We can substitute

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

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

General equation

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

$\Rightarrow 2^n + 2^n - 1$

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

Space Complexity $\rightarrow O(n)$

Reason :- The function calls are executed sequencially. Sequencially execution guarantees ~~that the~~ stack size will never exceed the depth of calls for first $f(n-1)$ it will create n stack.

(i) O(n logn)

```
# include <iostream>
using namespace std;
int partition (int arr [], int s, int e){
    int pivot = arr[s];
    int count = 0;
    for (int i = s; i<=e; i++){
        if (arr[i]<=pivot)
            Count++;
    }
    int pivot = s+ count;
    Swap (arr [pivot], arr [i]);
    int i = s, j=e;
    while (i < pivot&& j>pivot){
        while (arr[i] <= pivot)
            i++;
        while (arr [j]>pivot)
            j--;
        if (i< pivot && j> pivot){
    }
    return piv;
}
void quick (int arr [], int s, int e){
    if (s=e)
        return;
    int p= partition (arr, s,e);
    quick= (arr, s, p-1);
    quick(arr, p+1, e);
}
int main (){
    int arr [] = {6,2,5,8,1}
    int n= 5;
    quick (arr, 0, n-1);
    return 0;
}
```

(ii) O(N^2)

```
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("*");
   3
 3 3
 return 0;
3
```

(iii) O(log log n)-

```
int Count Primes (int n)<
  if (n<2)
   return 0;
 bool* non-prime = new bool [n];
  non-prime [1] = true;
  int num non prime = 1;
  for (int i = 2; i<n; i++)<
    if (non Prime [i])
      Continue;
   int j = i*2;
    while (j<n)<
    if (!non prime [j])<
      non prime [j] = true;
      num non prime++;
      3
     j+=1;
   3
  3
  return (n-1)- num non Prime;
3
```

4.) $T(n) = T(n/4) + T(n/2) + Cn^2$

Using master's Theorem -

Assume $T(n/2) >= T(n/4)$

Equations can be rewritten as

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

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

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

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

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

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

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

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

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

Ans 5

```
int Fun(int n){
  for(int i=1; i<=n; i++){
    for(int j=1; j<n; j++){
      int sum+=i;
    }
  }
}
```

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

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

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

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

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

6. For (int i = 2; i<; i = pow(i,k)) {

// Some O(1) expression

}

With iterations

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

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

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

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

$\therefore$ for last term be less than or equal to n

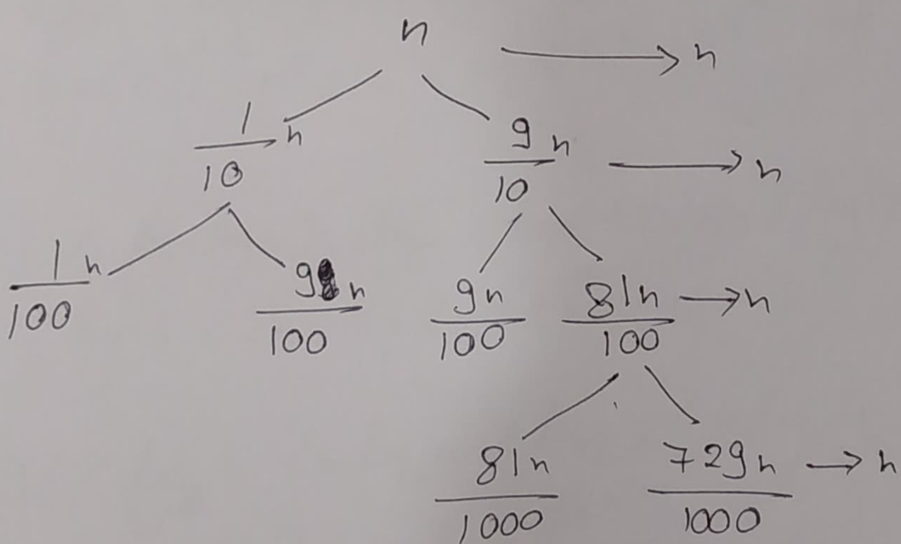
$$2^{K \log K (\log(n))} = 2^{\log n} = n$$

Each iteration takes constant time

$\therefore$ Total iteration $= \log K (\log(n))$

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

7



If we split in this manner

Recurrance Relation

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

...en first branch is of size $9n/10$ & second one
...s $n/10$.
...owing the above using recursion tree approch calculating
...lues

- 1st level, value $= n$
- 2nd level, value $= \frac{9n}{10} + \frac{n}{10} = n$

...alue remains same at all levels i.e n
Time Complexity = submissition of values

$\Rightarrow O(n \times \log(n))$ [upper bound]
$\Rightarrow \Omega(n \times \log(n))$ [Lower bound]
$\Rightarrow \Theta(n \log(n))$

...s 8

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

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

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