

Name : Jayesh Singh

Section : D

Roll Number : 11

Date

## TUTORIAL - 2

1) What is time complexity of below code and how?

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

⇒

i	j	
0	1	(initial)
1	2	
3	3	1 + 2 + 3 + ...
6	4	
10	5	
15	6	
21	7	
28	8	
36	9	
45	10	
55	11	
66	12	
78	13	
91	14	
105	15	
120	16	
136	17	
153	18	
171	19	
190	20	
210	21	
231	22	
253	23	
276	24	
300	25	
325	26	
351	27	
378	28	
406	29	
435	30	
465	31	
496	32	
528	33	
561	34	
595	35	
630	36	
666	37	
703	38	
741	39	
780	40	
820	41	
861	42	
903	43	
946	44	
990	45	
1035	46	
1081	47	
1128	48	
1176	49	
1225	50	
1275	51	
1326	52	
1378	53	
1431	54	
1485	55	
1540	56	
1596	57	
1653	58	
1711	59	
1770	60	
1830	61	
1891	62	
1953	63	
2016	64	
2080	65	
2145	66	
2211	67	
2278	68	
2346	69	
2415	70	
2485	71	
2556	72	
2628	73	
2701	74	
2775	75	
2850	76	
2926	77	
3003	78	
3081	79	
3160	80	
3240	81	
3321	82	
3403	83	
3486	84	
3570	85	
3655	86	
3741	87	
3828	88	
3916	89	
4005	90	
4095	91	
4186	92	
4278	93	
4371	94	
4465	95	
4560	96	
4656	97	
4753	98	
4851	99	
4950	100	

for  $i^{\text{th}}$  time  $\Rightarrow i = (1+2+3+\dots+i) < n$

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

$$\Rightarrow i^2 < n$$

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

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

```

2) int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}

```

Recurrence Relation

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

Let  $T(n)$  denote time complexity of  $F(n)$ .  
 For  $n \geq 1$

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

For  $n=0$  &  $n=1$ , no addition occurs

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

$$\text{Let } T(n-1) \approx T(n-2) \quad \text{--- (2)}$$

$$\text{(2) in (1)}$$

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

Using backward substitution:

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

$$\begin{aligned}
 T(n) &= 2 \times [2 \times T(n-2) + 1] + 1 \\
 &= 4T(n-2) + 3
 \end{aligned}$$

we can substitute

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

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

General Equation:

$$T(n) = 2^k \times T(n-k) + (2^k - 1) \quad \text{--- (3)}$$

$$\text{for } T(0)$$

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

substituting values 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:

Function calls are executed sequentially. Sequential execution guarantees that stack size will never exceed the depth of calls.

3)  $O(n \log n)$

// Merge Sort

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

```
void merge (int *array, int l, int m, int r)
{
    int i, j, k, nl, nr;
    nl = m - l + 1;    nr = r - m;
    int lar[nl], rarr[nr];
    for(i=0; i<nl; i++)
        lar[i] = array[l+i];
    for(j=0; j<nr; j++)
        rarr[j] = array[m+1+j];
    i=0; j=0; k=l;
```

```
while (i < n1 || j < n2)
{
    if (arr1[i] <= arr2[j])
    {
        arr[k] = arr1[i];
        i++;
    }
    else
    {
        arr[k] = arr2[j];
        j++;
    }
    k++;
}
```

```
void merge_sort(int *arr, int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;
        merge_sort(arr, l, m);
        merge_sort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
```

$O(N^3)$

```
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++)
            {
                cout << "Hey" << endl;
            }
        }
    }
}
```



$O(\log(\log n))$

```
int countprimes (int n) {  
    if (n < 2) return 0;
```

```
    boolean [] nonprime = new boolean [n];  
    nonprime [1] = true;
```

```
    int numnonprime = 1;
```

```
    for (int i = 2; i < n; i++) {  
        if (nonprime [i]) continue;
```

```
        int j = i * 2;
```

```
        while (j < n) {
```

```
            if (! nonprime [j]) {  
                nonprime [j] = true;
```

```
                numnonprime ++;
```

```
            }
```

```
            j += i;
```

```
        }
```

```
    }
```

4)  $T(n) = T(n/4) + T(n/2) + n^2$   
Using Master's Theorem,

$$T(n) \leq 2T(n/2) + n^2$$

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

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

$$\text{Also, } T(n) \geq n^2 \Rightarrow T(n) \geq O(n^2)$$

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

5) for  $i=1 \rightarrow j=1, 2, 3, \dots, n$

for  $i=2 \rightarrow j=1, 3, 5, \dots$

for  $i=3 \rightarrow j=1, 4, 7, \dots$

$$T(n) = n + n/2 + n/3 + \dots$$
$$= n(1 + 1/2 + 1/3 + \dots)$$

$$\Rightarrow \underline{\underline{O(n \log n)}}$$

6)

for first iteration  $i = 2$ 2nd iteration  $i = 2^k$ 3rd iteration  $i = (2^k)^k = 2^{k^2}$ 

⋮

nth iteration  $i = 2^{k^i} = n$ 

using logarithm,

$$\underline{\underline{i = \log_k (\log n)}}$$