# CS & IT ENGINEERING

## Algorithms

Introduction to Algorithms and Analysis

Lecture No.- 09

By- Dr. Khaleel Khan Sir

# Recap of Previous Lecture

- **Topic** Problem Solving with ASNs
- **Topic** Analysis of Recursive Algorithms
- **Topic**
- **Topic**
- **Topic**

# Topics to be Covered

**Topics**

Framework for Analysing Recursive Algo

Loop Complexities

# Topic: Asymptotic Notations

$$T(n) = 8T(n/2) + c, \quad n > 1$$
$$= a, \quad n = 1$$

$$T(n) = 8T(n/2) + c \quad - (1)$$
$$T(n/2) = 8T(n/4) + c \quad - (2)$$

$$T(n) = 8\left(8T(n/4) + c\right) + c$$
$$= 64 \cdot T(n/4) + 9c \quad - (3)$$
$$= 8^2 \cdot T\left(n/2^2\right) + \left(2^3 + 1\right)c \quad - (4)$$
$$\vdots$$
$$= 8^k \cdot T\left(n/2^k\right) + \left(2^{k+1} + 1\right)c$$

$$T(n) = 8^{\log_2 n} \cdot T(1) + \cdots$$
$$= n^{\log_2 8} \cdot c + \cdots$$
$$= n^3 + \cdots$$
$$O(n^3) \checkmark$$

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

$$T(n/2) = 2 \cdot T(n/4) + n/2 \cdot \log n/2 \quad -②$$

$$T(n) = 2\left[2 \cdot T(n/4) + n/2 \cdot \log n/2\right] + n \cdot \log n \quad -③$$

$$= 4 \cdot T(n/4) + n \cdot \log_2 n/2^1 + n \cdot \log n/2^0$$

$$= 2^2 \cdot T(n/2^2) + n \cdot \log_2 n/2^1 + n \cdot \log n/2^0$$

$$\vdots$$

$$= 2^K \cdot T(n/2^K) + n \cdot \log n/2^{K-1} + \cdots + n \cdot \log n/2^0 \quad -④$$

$$= 2^K \cdot T(n/2^K) + n\left[\log n/2^{K-1} + \log n/2^{K-2} + \cdots + \log n/2^0\right]$$

$$= 2^K \cdot T(1) + n \cdot \left[\log\left(\frac{n^K}{2^{0+1+2+\cdots+K-1}}\right)\right]$$

$$= 2^K \cdot T(1) + n\left[\log n^K - \log_2 2^{\frac{K(K-1)}{2}}\right]$$

$\frac{n}{2^K} = 1$

$n = 2^K$

$K = \log n$

$$= 2^K \cdot c + n\left[K \cdot \log n - \frac{K(K-1)}{2}\right]$$

$$= 2^K c + n\left[2K \cdot \log n - K^2 + K\right]$$

$$= n \cdot c + \frac{n}{2}\left[2\log^2 n - \log^2 n + \log n\right]$$

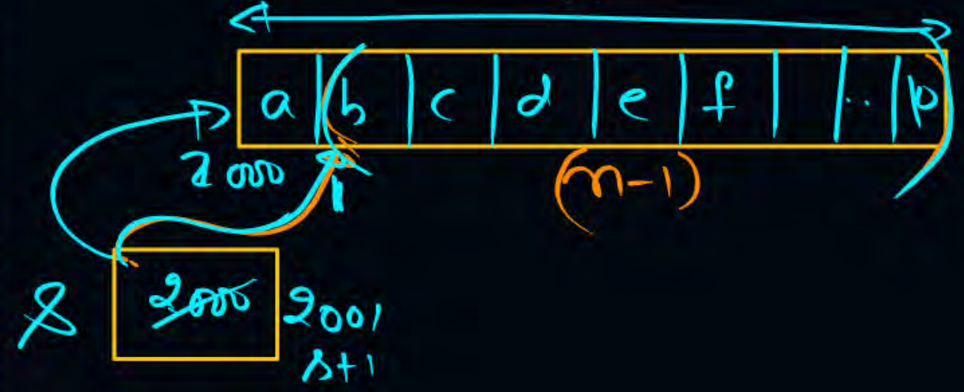
$$= cn + \frac{n}{2}\left[\log^2 n + \log n\right]$$

$$= cn + \frac{1}{2} n \log^2 n + \frac{n \cdot \log n}{2}$$

$$O(n \cdot \log^2 n)$$

Let 's' be a pointer to a String of length 'n' chars
What is the Time Complexity of this Program;

8.
```
void abc(char *s)
{
   if (*s != '\0')
   {
      printf("%c",*s);
      abc(s + 1);
      abc(s+1);
   }
}
```




$T(n) = c \quad , n=1$

$= a+b+2\cdot T(n-1) \quad , n>1$

$T(n) = 2\cdot T(n-1)+d \quad , n>1 \quad d>0$

$\rightarrow O(2^n)$ ✓

```
integer A[n];
Algorithm RSum(A, n)
{
    if (n = 1) return (A[1]);
    else return (RSum(A, n-1) + A[n]);
}
```

$T(n)$

$T(n-1)$ $\quad$ $a$

$$T(n) = \begin{cases} c, & n = 1 \\ = T(n-1) + d, & n > 1 \end{cases}$$

$\rightarrow O(n)$ ✓

```
Algorithm DO_it(n)
{
    if (n=1) return;
    else
        return (DO_it(n-1)+n);
}
```


$$T(n) = c \quad , n = 1$$

$$= a + T(n-1) + b, \quad n > 1$$

$$= \underline{\underline{T(n-1) + d}}$$

$$O(n)$$

for i ← 1 to n ⟺ for(i=1; i<=n; ++i)

{ S1; S2; } ⟹ Total times the loop repeats : 'n'

c=0;

for i ← 1 to n : n

c=c+1; O(1)

$$T(n) = \sum_{i=1}^{n} O(1)$$

Time:

| | |
|---|---|
| i=1 | O(1) |
| i=2 | O(1) |
| i=3 | O(1) |
| i=4 | O(1) |
| ⋮ | |
| i=n | O(1) |
| | (n) |

Time Complexity depends on No. of times the loop is repeat (n) & the Complexity of the statements in the body of loop

```
for i ← 1 to n        } (n+1)
    c = c+1;          } O(n)
```
(c = n)

$n + c \Rightarrow O(n)$

---

c = 0

```
for i ← 1 to n
    c = c + i;
```

$c = (1 + 2 + 3 + 4 + \cdots + n) = \frac{n(n+1)}{2} = O(n^2)$

1) Time : $O(n)$

2) Value(c) : $O(n^2)$

```
for i ← 1 to n/2
    c = c+1
```
: Time : $O(n)$, Value : $O(n)$

```
for i ← 1 to n
    break;
```
: O(1)

```
Mead(n);
for i ← 2 to n
    if(n % i == 0)
        break;
```

Body

→ Worst Case : O(n)

→ Best Case : O(1)

```
for i ← 1 to n
    B(n);
```

Time : $O(n * O(B(n)))$

(i) if $B(n) = O(1) \Rightarrow O(n)$

(ii) if $B(n) = O(\log n) \Rightarrow O(n \cdot \log n)$

(iii) if $B(n) = O(n) \Rightarrow O(n^2)$

c = 0

```
for(i=1; i<=n; ++i);
```

i = n+1

c = c+i; "1"

Time : O(n)

Value of c = n+1

---

c = 0;

```
while(1)
{
    c = c+1;
}
```

Time : ∞

---

## While - loop :

```
while(condition)
{
    s1;
    s2;
    .
    sk;
}
```

```
i = 1;
while(i <= n)
{
    i = i+1;
}
```

'n'

O(n)

## Nested loops:

(i) Independent

```
for(i=1; i<=n; ++i)
{
    for(j=1; j<=n; ++j)
    {
        c=c+1;
    }
}
```

The inner loop runs $n$ times.

| | | |
|---|---|---|
| i=1 | j: 1 to n | : n |
| i=2 | j: 1 to n | : n |
| i=3 | j: 1 to n | : n |
| ⋮ | | ⋮ |
| i=n | " | : n |

$n \times n = n^2$

## Mutually Exclusive loops

```
{
1. for i ← 1 to n  : n
       c=c+1;

2. for j ← 1 to m  : m
       k=k*2;

}
```

Time: $O(n+m)$

$: O(\max(n, m))$

$O(n^2)$

```
for i ← 1 to n : n
{
    a) B(n) : logn
    b) for j ← 1 to n/2 } O(n)
           c = c+1;
       k = 0
    c) while(k <= n) } O(n)
           ++k;
}
```

if $B(n) = O(\log n)$

Time : $O(\quad)$

Per It of 'i' : $(\log n + O(n) + O(n))$

: $O(n)$

Total_time $= n * [\log n + n + n]$

$= n \cdot \log n + n^2 + n^2 = O(n^2)$

# Topic : Running Times of Program Segments with Loops:

Loop-Complexities

1.
```
for i ← 1 to n
    c = c + 1;
```
$O(n)$

2.
```
for i ← 1 to n
    for j ← 1 to n/2
        c = c + 1;
```
$n$, $n/2$

$(n * \frac{n}{2}) = O(n^2)$

3.
```
for i ← 1 to n          : n
    for j ← 1 to n/4    : n/4
        for k ← 1 to n  : 1
            break;
```
Time : $n * \frac{n}{4} * 1$

$: O(n^2)$

# THANK - YOU