

## Algorithm Analysis

- to find out resources like time and space required to run the algorithm

### Exact Analysis

- find out exact time and space required to run the algorithm
- time (nS, uS, mS) - it is dependent on type of machine, no of processes at that time
- space (Bytes, kb, mb) - it is dependent on data type of variables, type of machine (architecture)

### Approximate Analysis

- find out approximate budget of time and space required to run the algorithm
- Asymptotic analysis
  - mathematical way of finding approximate time and space requirement of the algorithm
- "Big O" notation is used to indicate space and time requirement (complexity)

## Time Complexity

- find no of iterations of the loop used in an algorithm
- time is directly proportional to iterations of the loop

### 1. find factorial of a number

```
int findFactorial(int n){  
    int fact = 1;  
    for(i = 1 ; i <= n ; i++)  
        fact *= i;  
    return fact;  
}
```

No. of itr = n

Time  $\propto$  n

$T(n) = O(n)$

### 2. Print 2D array on console

```
void print2DArray(int arr[][], int row, int col){  
    for(i = 0 ; i < row ; i++)  
        for(j = 0 ; j < col ; j++)  
            sysout(arr[i][j]);  
}
```

Total itrs = n \* n

Time  $\propto$  n<sup>2</sup>

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

### 3. Find sum of two numbers

```
int findSum(int num1, int num2){  
    int sum = num1 + num2;  
    return sum;  
}
```

irrespective of values inside num1 & num2, this algorithm will take same/constant amount of time.

$$T(n) = O(1)$$

### 4. Print table of given number

```
void printTable(int n){  
    for(int i = 1; i <= 10; i++)  
        sysout(i * n);  
}
```

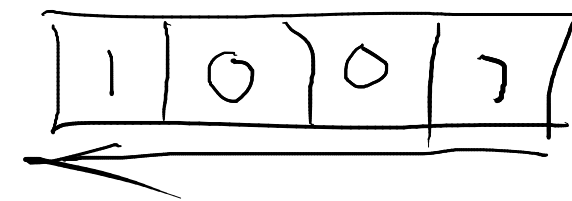
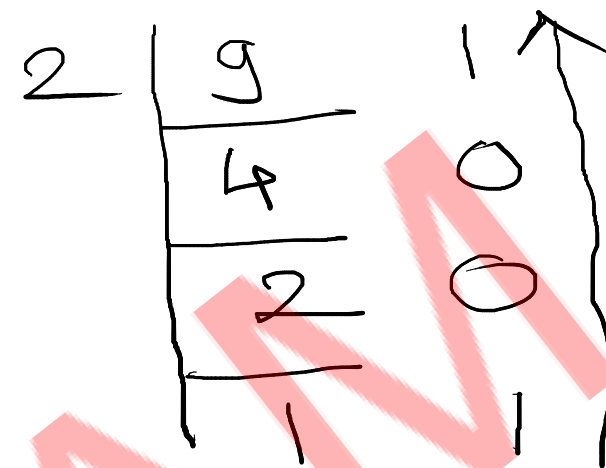
constant/fixed itrs  
constant time  
requirement

$$T(n) = O(1)$$

## 5. Print binary of given decimal

```
void printBinary(int n) {
    while(n > 0) {
        sysout(n % 2);
        n /= 2;
    }
}
```

n	n > 0	n % 2
9	T	1
4	T	0
2	T	0
1	T	1
0	F	



$$(9)_{10} = (1001)_2$$

$$n = 9, 4, 2, 1, 0$$

$$n = n, n/2, n/4, n/8, n/16$$

$$= n/2^0, n/2^1, n/2^2, n/2^3, n/2^4 \text{ itr}$$

$$= n/2^0, n/2^1, n/2^2, n/2^3, \dots, n/2^{\text{itr}}$$

for last value 1 of n, condition is true

$$n/2^{\text{itr}} = 1$$

$$2^{\text{itr}} = n$$

$$\text{itr} \log 2 = \log n$$

$$\text{itr} = \frac{\log n}{\log 2}$$

$$\text{Time} \propto \frac{\log n}{\log 2}$$

$$T(n) = O(\log n)$$

# Space Complexity

- find total space required

**Total space** = **input space** + **Auxillary space**  
(Actual space of data) (Extra space required to process actual data)

Find sum of array elements

```
int findSum(int arr[], int n){  
    int sum = 0;  
    for(int i = 0 ; i < n ; i++)  
        sum += arr[i];  
    return sum;  
}
```

Input variable = arr

Auxillary variables = n, sum, i

Input space = n

Auxillary space = 3

Total space = n + 3

$S(n) = O(n)$   $n \gg \gg$

Auxillary space complexity  $\Rightarrow S(n) = O(1)$

## Linear Search

- |                        |  |                            |
|------------------------|--|----------------------------|
| <b>1. Best case</b>    | <b>- key is found at initial locations</b> | <b>- <math>O(1)</math></b> |
| <b>2. Average case</b> | <b>- key is found at middle locations</b>  | <b>- <math>O(n)</math></b> |
| <b>3. Worst case</b>   | <b>- key is found at last locations</b>    | <b>- <math>O(n)</math></b> |
|                        | <b>- key is not found</b>                  |                            |