TOPICS

1. Asymptotic Analysis

* Time Complexity
* Space Complexity

1. Array
2. Stack
3. Queue
4. Searching

* Linear Search
* Binary Search

1. Sorting

* Selection Sort
* Bubble Sort
* Insertion Sort
* Heap Sort
* Quick Sort
* Merge Sort
* Bucket Sort
* Radix Sort

1. Linked list

* Single Linked List
* Circular Linked List
* Doubly Linked List

1. Hashing
2. Tree
3. Graph

Asymptotic Analysis

* we evaluate the performance of an algorithm in terms of input size.
* how the time (or space) taken by an algorithm increases with the input size.
* **Three cases to analyse an algorithm** :

1. *Best Case* : we calculate upper bound on running time of an algorithm. We must know the case that causes maximum number of operations to be executed.
2. *Average Case* : we take all possible inputs and calculate computing time for all of the inputs.
3. *Worst Case* : we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed.

* **Types of Asymptotic Analysis :**

[ Any notation can be used to describe the worst case , best case or average case ]

1. *Theta notation (Θ)* : bounds a function from above and below, so it defines exact asymptotic behavior

Θ(g(n)) = {f(n): there exist positive constants c1, c2 and n0 such

that 0 <= c1\*g(n) <= f(n) <= c2\*g(n) for all n >= n0}

1. *Big O notation* : defines an upper bound of an algorithm.

O(g(n)) = { f(n): there exist positive constants c and

n0 such that 0 <= f(n) <= c\*g(n) for all n >= n0}

1. *Ω notation* : defines lower bound of an algorithm

Ω (g(n)) = {f(n): there exist positive constants c and

n0 such that 0 <= c\*g(n) <= f(n) for all n >= n0}.

* **Properties :**
* If f(n) is O(g(n)) then a\*f(n) is also O(g(n)) ; where a is a constant.
* Transitive, If f(n) is O(g(n)) and g(n) is O(h(n)) then f(n) = O(h(n)) .
* Reflexive, If f(n) is given then f(n) is O(f(n)). Since MAXIMUM VALUE OF f(n) will be f(n) ITSELF !
* Symmetric, If f(n) is Θ(g(n)) then g(n) is Θ(f(n)) .
* Transpose, If f(n) is O(g(n)) then g(n) is Ω (f(n)).
* If f(n) = O(g(n)) and f(n) = Ω(g(n)) then f(n) = Θ(g(n))
* If f(n) = O(g(n)) and d(n)=O(e(n)) , then f(n) + d(n) = O( max( g(n), e(n) ))
* If f(n)=O(g(n)) and d(n)=O(e(n)) , then f(n) \* d(n) = O( g(n) \* e(n) )
* **DRAWBACKS :** It ignores small input sizes. At small input sizes, constant factors or low order terms could dominate running time