Time & Space Complexity

* Time Complexity

→ Amount of time taken by an algorithm to run as a function of length of input.

→ Here time taken is not actual time, it's CPU operations

* Space Complexity

→ Amount of space taken by an algorithm of length of input.

* Case analysis of Algorithm

1.) Worst case analysis (Mostly used)

→ We calculate the upper bound on the running time of an algorithm. We must know the case that causes a maximum number of operations to be executed

2.) Best Case analysis (Very rarely used)

→ We calculate the lower bound on the running time of an algorithm. We must know the case that causes minimum number of operations to be executed.

3) Average case analysis

→ We take all possible inputs & calculate the computing time for all the inputs. Sum all the calculated values and divide the sum by the total number of inputs.

Asymptotic Notation

* Big O notation (Worst case)

→ Big O notation is a powerful tool used in computer science to describe the time complexity or space complexity of algorithm.

→ Big O is a way to express the upper bound of an algorithm's time or space complexity.

→ It provides an upper limit on the time taken by an algorithm in terms of size of the input. We mainly consider the worst case scenario of the algorithm to find its time complexity in terms of Big O

→ It is denoted as O(f(n)), where f(n) is a function that represents the number of operations (steps) that an algorithm performs to solve a problem of size n

Properties

① ~~Reflexivity~~ ① Reflexivity

If $f(n)$ is function, Then $f(n) = O(f(n))$

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

② Transitivity

If $f(n) = O(g(n))$

$\rightarrow g(n) = O(h(n))$

Then $f(n) = O(h(n))$

③ Constant factor

For any constant $K > 0$ (positive)

if $f(n) = O(g(n))$ Then $\Rightarrow Cf(n) = O(g(n))$

④ Sum rule

→ if $f(n) = O(g(n))$ & $k(n) = O(h(n))$, Then the largest one will dominate

Thus $f(n) + k(n) = \max(O(g(n)), O(h(n)))$

Eg $O(n^2) + O(n^3) + O(\log n) + O(n) = O(n^3)$

⑤ Product rule

if $f(n) = O(g(n))$ & $h(n) = O(k(n))$

Then $f(n) * h(n) = O(g(n) * k(n))$

Eg.) $f(n) = n^2$ $g(n) = n^4$

$\therefore f(n) * g(n) = O(n^6)$

⑥ Composition rule

if $f(n) = O(g(n))$ & $g(n) = O(h(n))$

Then $f(g(n)) = O(h(n))$

Big O: Complexities

① Constant time: $O(1)$

```
>> int a = 5;
```

② Linear time: $O(n)$ (Linear search)

```
for (int i=0; i<N; i++){
}
```

③ Logarithmic time: $O(\log n)$ (Binary search)

```
for (int i=n; i>1; i=i/2)
```

PAGE NO.:

④ Quadratic time: $O(N^2)$

```
for ( ){
  for ( ) {}
}
```

⑤ Cubic time : $O(N^3)$

```
for( ) {
   for( ) {
      for{
      }}}
```

⑥ Polynomial time : $O(n^k)$

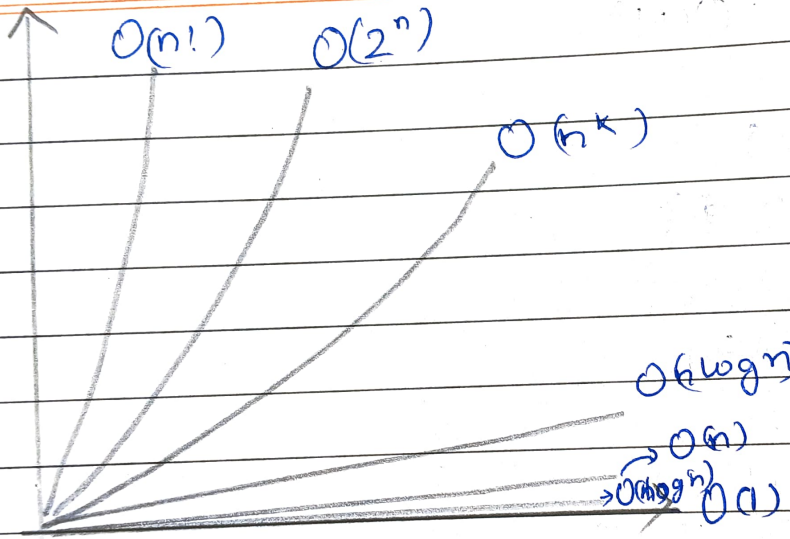
→ Just like quadratic, cubic --- upto power k by forming nested loop of k times.

⑦ Exponential Time : $O(2^n)$

Exponential time complexity means that the running time of an algorithm doubles with each addition of to the input data set.

⑧ Factorial time : $O(n!)$

The running time of an algorithm grows factorially with the size of the input.



Order of Complexity

⇒ $O(1) < O(\log N) < O(\sqrt{N}) < O(N) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(N!) < O(N^n)$