\*\* **Time Complexity**



\***Time Complexity Chart:**

****

****

**Lower the Time Complexity of a program more efficient the program looks.**

N4 + N2 + 100 is approximately equal to N4.

In a program algorithm, there are 3 types of analysis:

1. Best Case-- Defines the input for which the algorithm takes the least time (fastest time to compile). Input is the one for which the algorithm runs the fastest.
2. Worst case-- Defines the input for which the algorithm takes greatest time (slowest time to complete). Input is the one for which the algorithm runs the slowest.
3. Average case-- Provides a prediction about the running time of the algorithm. Run the algorithm many times, using many different inputs that come from some distribution that generates these inputs, compute the total running time (by adding the individual times), and divide by the number of trials.

NOTE:-Lower Bound ≤ Average Time ≤ Upper Bound

**\*\* BIG -O Notation: -**

A function is said to be O(g(n)) if there exists constant C and constant n0 such that **0 ≤ f(n) ≤c. g(n) for all n≥n0**

In simple words, O(g(n)) is the set of functions with smaller or the same order of growth as g(n). For example- O(n2) includes O(1), O(n), O(nlogn).

O(1) = 100, 150, 34539, 1 etc.

O(n) = 3n+100, 400, 2n-1 etc.

O(nlogn) = 8nlogn, 3n-10, 100n, 100 etc.

O(n2) = 5n2, 2n2+3n+1, 6nlogn, 50 etc.

**Example-1** Find upper bound for f(n) = 3n + 8

**Solution:** 3n + 8 ≤ 4n, for all n ≥ 8

∴ 3n + 8 = O(n) with c = 4 and n0 = 8

**Example-2** Find upper bound for f(n) = n2 + 1

**Solution:** n2 + 1 ≤ 2 n2, for all n ≥ 1

∴ n2 + 1 = O(n2) with c = 2 and n0 = 1

**Example-3** Find upper bound for f(n) = n4 + 100 n2 + 50

**Solution:** n4 + 100 n2 + 50 ≤ 2 n4, for all n ≥ 11

∴ n4 + 100 n2 + 50 = O(n4) with c = 2 and n0 = 11

**Example-4** Find upper bound for f(n) = 2n3 – 2n2

**Solution:** 2 n3 – 2n2 ≤ 2n3, for all n > 1

∴ 2n3 – 2n2 = O(n3) with c = 2 and n0 = 1

**Example-5** Find upper bound for f(n) = n

**Solution:** n ≤ n, for all n ≥ 1

∴ n = O(n) with c = 1 and n0 = 1

**Example-6** Find upper bound for f(n) = 410

**Solution:** 410 ≤ 410, for all n > 1

∴ 410 = O(1) with c = 1 and n0 = 1

**\*Application:**

int search(int arr[],int n, int key)

{

for(int i=0; i<a; i++)

if(arr[i]==key)

return I;

return -1;

}

The above function is O(n).

Now even if the element is present at 1st position time taken will be 1 ie. constant but still it comes under the O(n) notation.

Big O is the upper bound on the order of growth.

Big Omega is the lower bound on the order of growth.

Big theta is the exact order of growth.

**\*\* BIG -Omega Notation: -**

A function is said to be Ω (g(n)) if there exists constant C and constant n0 such that **0 ≤c. g(n) ≤ f(n) for all n≥n0**

**Example-1** Find lower bound for f(n) = 3n + 8

**Solution:** 2n ≤ 3n+8, for all n ≥ 0

∴ 3n + 8 = O(n) with c = 2 and n0 = 0

**Example-2** Find lower bound for f(n) = n2 + 1

**Solution:** n2 ≤ n2 +1, for all n ≥ 0

∴ n2 + 1 = O(n2) with c = 1 and n0 = 0

**Example-3** Find lower bound for f(n) = 2n4 + 100 n2 + 50

**Solution:** n4 ≤ 2n4 + 100 n2 + 50, for all n ≥ 0

∴ 2n4 + 100 n2 + 50 = O(n4) with c = 1 and n0 = 0

**\*\* BIG -Theta Notation: -**

A function is said to be θ(g(n)) if there exists constant C and constant n0 such that **0 ≤c. g(n) ≤ f(n) for all n≥n0**

**NOTE:** O (log n!) ≤ O(nlogn)