TLE Eliminators

Time Complexity

→ CP
→ DSA
→ Interviews

- Viraj Chandra

Goal

To understand:

- Elementary Operations
- Time Complexity
- Big-O Notation
- Evaluating TC of an algorithm
- Evaluating expected TC based on constraints of the problem

O(100)

What is an Elementary Operation?

An operation that takes **constant time** is called elementary operation.

- **Arithmetic Operations**
  - A + B , A - B , A * B , A / B
- **Comparison of Primitive Datatypes (int, float, char etc.)**
  - A > B , A < B , A == B
- **Input and Output of Primitive Datatypes (int, float, char etc.)**
  - cin >> A , cout << A

NOTE: 10^8 elementary operations ≈ 1 second of time

Quiz 1

Is the following an elementary operation?

1

```cpp
int a, b, c, d;
cin >> a >> b >> c >> d;
cout << (a + b * c) / d << endl;
```

YES

2

```cpp
string s, t;
cin >> s >> t;
if (s < t)
    cout << "s is smaller than t" << endl;
```

NO

$n \to 1000, 10000$ $TC = O(n^2)$

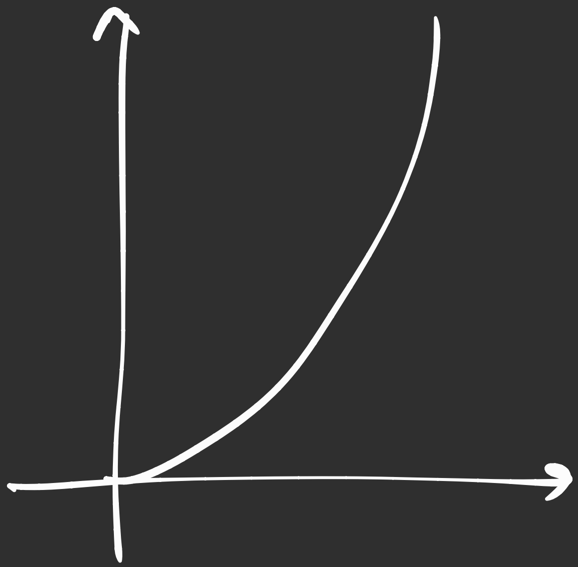
What is Time Complexity?

Time complexity is a function to describe **the approximate amount of operations** an algorithm requires for the given input.

We can calculate **approximate execution time of code** using time complexity and constraints.

We will understand more about calculating time complexity in the next slides.

$$f(\underbrace{x}_{\text{input}}) = \underbrace{x^2}_{\text{output}}$$



input

Constraits

$n \rightarrow 1 \leq n \leq 1000$

$a[i] \rightarrow 1 \leq a[i] \leq 10^6$

no. of elementary operations wrt to n

3 ways

① atleast $\qquad 1$

② atmost $\qquad n^2$

③ average $\qquad \dfrac{n^2+1}{2}$

```
for( 1 to n )
{ for ( 1 to n )
  { if(rand == 10) → stop
    { // come out of everything }
  }
}
```

Big-O Notation

$O(n)$

atmost
worst
upperbound

Tetha Notation

$\Theta(n)$

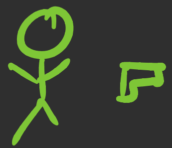
avg
middle

Omega Notation

$\Omega(n)$

atleast
best
lowerbound

1 to 10



$X = 10$

$$\underbrace{n + n + n + n + \dots\dots}_{n \text{ times}} n$$

$$\Rightarrow n^2$$

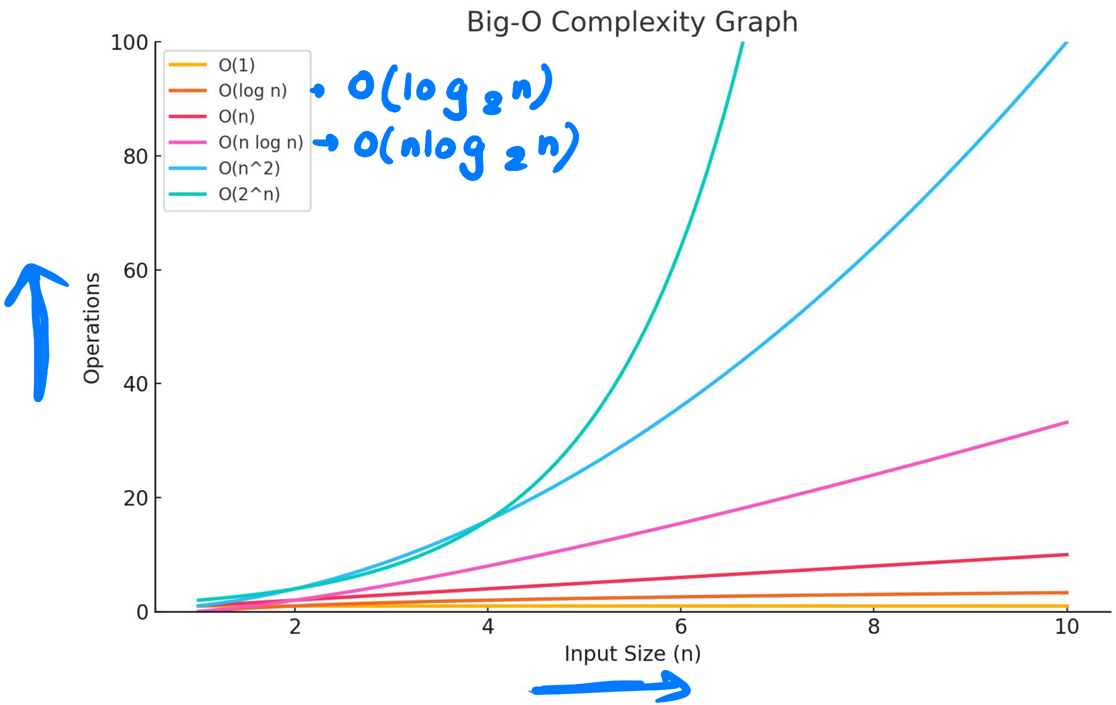
Big-O Notation

Big-O of an algorithm is a function to **calculate the worst case time complexity** of the algorithm.

It is written as **O(worst case time complexity)**

Big-O is used to calculate the approximate **upper bound** of the algorithm. It expresses how the run time of the algorithm grows relative to the input.

Big-O Notation



Rules for Big-O Notation

- Should not have constants.
  - $O(N + K) = O(N)$
- Should not have constant factors.
  - $O(N / 2) = O(N)$
- Only include the fastest growing function for each variable.
  - $O(N^2 + N) = O(N^2)$
- Can never be 0. Has to be at least $O(1)$.

Example: $2N^2+4N+4(M^3+5)+10$ = ? $O(N^2+M^3)$

Quiz 2

1. $(N + M) / K \rightarrow O(N+M)$

2. $N * (N + 1) / 2 \rightarrow O(N^2 + N/2) \rightarrow O(N^2)$

3. $N + \underline{MN^2 + M^2N + M} \rightarrow O(MN^2 + M^2N)$

4. $N^3 / 64 + \underline{20N + (32NM)^2} \rightarrow O(\underline{N^2M^2} + \underline{N^3})$

Calculate TC of an Algorithm

Time complexity usually depends on:

- **Loops**
- **STL** - to be covered in the upcoming classes
- **Recursion**

Time Complexity of recursive algorithms will not be covered.

Calculate TC of an Algorithm

In Big-O notation, when you have nested loops, the total time complexity is calculated by **multiplying the number of iterations of each loop.** The time complexity here is **O(n*m).**

```cpp
int n = 5, m = 3;
// Outer loop runs 'n' times
for (int i = 0; i < n; i++) {
    // Inner loop runs 'm' times for each iteration of the outer loop
    for (int j = 0; j < m; j++) {
        cout << "* ";
    }
    cout << "\n";
}
```

$O(n*m)$

Quiz 3

Find the time complexity of the following code snippets in Big-O notation

```
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n / 2; j++)
    {
        // Elementary Operation
    }
}
```

$$O\left(n * \frac{n}{2}\right)$$

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

Quiz 3

```
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        for (int k = 0; k < n; k++)
        {
            // Elementary Operation
        }
    }
}
```

$O(n^3)$

Quiz 3

```
for (int i = 12; i <= n - 123; i += 5)
{
    for (int j = 6; j <= m * 2; j += 321)
    {
        for (int k = 4023; k > 23; k -= 16)
        {
            // Elementary Operation
        }
    }
}
```

$O(n*m)$

Quiz 3

$i = \overbrace{1, 2, 4, 8, 16}\ldots\ldots \quad 2^x < n$

```
for (int i = 1; i < n; i *= 2)
{
    // Elementary Operation
}
```

$2^x \approx n$

$\log_2 2^x \approx \log_2 n$

$\underline{x \approx \log_2 n}$

Cheatsheet

Expected Time Complexity

Feasible Big-O Function	Maximum N	Example Algorithms
O($N!$)	10	All permutations of a list
O(N^3)	400	Multiplication of two matrices
O(N^2)	5000	Square grid, bubble sort, insertion sort
O($N\sqrt{N}$)	10^5	Usually related to factoring
O($NlogN$)	10^6	Merge sort, binary search for N times
O(N)	10^7	Linear search, reversing an array, string comparison
O($\sqrt{N}$)	10^{12}	Factors of a number
O($logN$), O(1)	10^{18}	Binary search, Constant time formulas

Points to Note

- Identify the variables that contribute to time complexity.
- Just because constraints allow slower solutions, doesn't mean we cannot optimise it.
    - **For n = 1000,** both $O(n^2)$ and $O(n)$ will work.
- Test cases matter, unless there's a limit explicitly imposed in the constraints.
    - **Example:** "The sum of n over all test cases do not exceed X"
- The constants and constant factors removed when calculating Big-O still matter.

test file

$n \approx 10^5$ in a testcase

$1s = 10^8$ ops

$O(n)$

$2s = 2 \times 10^8$ ops

ops / testcase $= \dfrac{2 \times 10^8}{10^4} = 2 \times 10^4$ ops

$O(n \log n) \rightarrow$ work $\leq 2 \times 10^8$

10^4 cases

$\downarrow$

$t_1 \quad t_2 \quad t_3 \quad \cdots \quad t_{10^4}$

$n_1 + n_2 + n_3 + \cdots + n_{10^4} \leq 10^5$

Important Links

- https://towardsdatascience.com/essential-programming-time-complexity-a95bb2608cac
- https://www.youtube.com/watch?v=9T|HvipP5yA
- https://www.youtube.com/watch?v=9SgLBjXqwd4
- https://www.youtube.com/watch?v=10DTkS1L2k
- https://adrianmejia.com/most-popular-algorithms-time-complexity-every-programmer-should-know-free-online-tutorial-course/ (advanced)