Time Complexity

1. What is Time Complexity

Time complexity represents how the runtime of an algorithm grows with the input size. It helps evaluate the efficiency of an algorithm. In interviews and competitive programming, understanding time complexity ensures your solutions run within time limits.

---

2. What is Big O (and Other Notations)

Big O notation describes the **worst-case** time complexity of an algorithm.

- **Big O (O)** → Worst-case
- **Big Theta (Θ)** → Average-case
- **Big Omega (Ω)** → Best-case
  In interviews and online judges, Big O (worst-case) is the most important.

---

3. Rules for Big O Notation

- Ignore constants: O(2N) → O(N)
- Drop non-dominant terms: O(N + N²) → O(N²)

---

4. Constant Time Example: `O(1)`

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    int sum = (n * (n + 1)) / 2;
    cout << sum << endl;
```

```
    return 0;
}
```

The code given below is also an example of constant time complexity.

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    for(int i=0;i<1000000;i++)
    {
        // some operation
    }
    return 0;
}
```

---

5. Linear Time Example: `O(N)`

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    int sum = 0;
    for (int i = 1; i <= n; i++)
    {
        sum += i;
    }
    cout << sum << endl;
    return 0;
}
```

---

6. Examples of `O(N + M)`

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n, m;
    cin >> n >> m;
    int a[n], b[m];
    for (int i = 0; i < n; i++)
        cin >> a[i];
    for (int i = 0; i < m; i++)
        cin >> b[i];
    int c[n + m];
    for (int i = 0; i < n + m; i++)
    {
        if (i < n)
            c[i] = a[i];
        else
            c[i] = b[i - n];
    }
    for (int i = 0; i < n + m; i++)
        cout << c[i] << " ";
    return 0;
}
```

---

7. Code Examples of `O(N^2)` and `O(N^3)`

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < n; i++)
    {
```

```
        for (int j = 0; j < m; j++)
        {
            cout << i << " " << j << endl;
        }
    }
    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n, m, o;
    cin >> n >> m >> o;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            for (int k = 0; k < o; k++)
            {
                cout << i << " " << j << " " << k <<
endl;
            }
        }
    }
    return 0;
}
```

---

8. Code Example of `O(logN)` and Proof

```
int bsearch(int a[], int n, int tar)
```

```
{
    int l=0,r=n-1,mid;
    while(l<r)
    {
        mid=(l+r)/2;
        if(a[mid]==tar) return mid;
        if(a[mid]>tar) r=mid-1;
        else l=mid+1;
    }
    return -1;
}
```

Proof of O(logN):
Each iteration cuts the search space in half → `N → N/2 → N/4 → ... → 1`
`N/2^0 → N/2^1 → N/2^2 → ... → N/2^K`

Total steps depend on `K`, and when `N/2^K = 1`:
→ `2^K = N`
→ $\log_2(2^K) = \log_2(N)$
→ $K = \log_2(N)$ [since $\log_a(a^X) = X$]

So the complexity is `O(logN)`

9. Practice O(N) Complexity

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i *= 2)
    {
        for (int j = 1; j <= i; j++)
        {
```

```
            cout << j << endl;
        }
    }
    return 0;
}
```

Proof of Time Complexity

Consider a nested loop where:

- The inner loop runs for `1, 2, 4, 8, ..., N` iterations.
- The outer loop runs log2(N) times.

The total number of iterations in the inner loop over all outer loop runs forms a geometric series:

`S = 1 + 2 + 4 + 8 + ... + 2^(k-1)`

where:

- a = 1 (the first term),
- r = 2 (the common ratio),
- k = log2(N) (the number of terms).

Using the geometric series sum formula:

`S = a * (r^k - 1) / (r - 1)`

Substitute the values:

`S = 1 * (2^(log2(N)) - 1) / (2 - 1) = 2^(log2(N)) - 1`

Recall that:

`2^(log2(N)) = N`

Therefore:

`S = N - 1`

Ignoring constants in Big-O notation, the overall time complexity is: **`O(N)`**