Time & Space Complexity

1. What is Space Complexity

There are two types of space in a program: **input space** and **auxiliary space**.

Space complexity refers to the **auxiliary space**, which is the extra space or temporary space used by an algorithm **excluding** the space required for the input.

---

2. `O(1)` Space Complexity

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

---

3. `O(N)` Space Complexity

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
```

```
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];
    int b[n];
    for (int i = 0; i < n; i++)
        b[i] = a[i];
    for (int i = 0; i < n; i++)
        cout << b[i] << " ";
    return 0;
}
```

---

4. `O(N^2)` Space Complexity

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    int a[n][n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            a[i][j] = i + j;
        }
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << a[i][j] << " ";
        }
```

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

---

5. Check if a number is Prime in O(N) time O(1) space

```
#include <bits/stdc++.h>
using namespace std;
bool isPrime(int n)
{
    if (n <= 1)
        return false;
    for (int i = 2; i < n; i++)
    {
        if (n % i == 0)
            return false;
    }
    return true;
}
int main()
{
    int n;
    cin >> n;
    if (isPrime(n))
        cout << "yes" << endl;
    else
        cout << "no" << endl;
    return 0;
}
```

The code given below is also **O(N)**.time complexity

```
#include <bits/stdc++.h>
using namespace std;
```

```
bool isPrime(int n)
{
    if (n <= 1)
        return false;
    for (int i = 2; i <= n / 2; i++)
    {
        if (n % i == 0)
            return false;
    }
    return true;
}
int main()
{
    int n;
    cin >> n;
    if (isPrime(n))
        cout << "yes" << endl;
    else
        cout << "no" << endl;
    return 0;
}
```

---

6. Check if a number is Prime in `O(sqrt(N))` time `O(1)` space

```
#include <bits/stdc++.h>
using namespace std;
bool isPrime(int n)
{
    if (n <= 1)
        return false;
    for (int i = 2; i * i <= n; i++)
    {
        if (n % i == 0)
            return false;
```

```
    }
    return true;
}
int main()
{
    int n;
    cin >> n;
    if (isPrime(n))
        cout << "yes" << endl;
    else
        cout << "no" << endl;
    return 0;
}
```

---

7. Examples of `O(N * logN)` time complexity

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) // O(N)
    {
        for (int j = 1; j <= n; j *= 2) // O(logN)
        {
            cout << j << endl;
        }
    }
    return 0;
}
```

---

8. Code Examples of `O(N)` time and `O(N)` space complexity for `Recursion`

```
#include <bits/stdc++.h>
using namespace std;
int fact(int n)
{
    if (n == 0)
        return 1;
    return n * fact(n-1);
}
int main()
{
    int n;
    cin >> n;
    cout << fact(n);
    return 0;
}
```

---

9. Code Examples of `O(2^N)` time and `O(N)` space complexity for `Recursion`

```
#include <bits/stdc++.h>
using namespace std;
int fib(int n)
{
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return fib(n - 1) + fib(n - 2);
}
int main()
{
    int n;
```

```
    cin >> n;
    cout << fib(n) << endl;
    return 0;
}
```

---

10. Code Examples of `O(N^2)` time and `O(N^2)` space complexity for `Recursion`

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

---

11. Code Example of `O(N * logN)` and Proof

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
```

```
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n / i; j++)
        {
            cout << j << endl;
        }
    }
    return 0;
}
```

Proof of Time Complexity

Consider a nested loop where:

- The inner loop runs for `N, N/2, N/3, N/4, N/5, ..., 1` iterations.
- The outer loop runs N times.

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

`S = 1 + 1/2 + 1/3 + 1/4 + ... + 1/N`

`Time complexity for a harmonic series is log(N)`

So, the overall time complexity is: **`O(N * logN)`**

Proof of Harmonic Series

When `N = 100`,

The series will be: `1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6 + ... + 1/100`

Here,

```
1 = 1
1/2 + 1/3 < 1
1/4 + 1/5 + 1/6 + 1/7 < 1
1/8 to 1/15 < 1
1/16 to 1/31 < 1
```

```
1/32 to 1/63 < 1
1/64 to 1/100 < 1
```

So, the total summation is approximately `log2(N)`

---

12. 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 N, N/2, N/4, N/8, ..., 1 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 = N + N/2 + N/4 + N/8 + ... + 1
```

where:

- a = N (the first term),
- r = 1/2 (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 = N * ((1/2)^(log2(N)) - 1) / ((1/2) - 1)
```

```
= 2N [1 - (1 / 2^(log2(N)))]
```

Recall that:

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

Therefore:

```
S = 2N [1 - (1 / N)]
= 2(N - 1)
= O(N)
```

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