Time Complexity

Data Structures – Coding Course

Complexity

- Time and space complexity tell us **how algorithms scale**
- Helps choose **efficient algorithms** for large inputs
- What is Space Complexity?
- What is Time Complexity?

Notation

- Big O (O) – Upper bound: Worst-case growth
- Theta (Θ) – Tight bound: Exact asymptotic growth
- Omega (Ω) – Lower bound: Best-case growth

Let's make hands dirty

```
for (int i = 0; i < 100; i++) cout << i;
```

O(1)

```
for (int i = 0; i < n; i++) cout << arr[i];
```

O(n)

```
int n=200;
for (int i = 0; i < n; i++) cout << arr[i];
```

O(1)

```
for (int i = 0; i < n; i+=2) cout << arr[i];
```

O(n)

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        cout << i + j;
```

$O(n^2)$

```
for (int i = 0; i < n; i++)
    for (int j = i; j < n; j++)
        cout << i << j;
```

$O(n^2)$

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n*n; j++)
        cout << i + j;
```

$O(n^3)$

```
for (int i = 1; i < n; i *= 2) cout << i;
```

O(log n)

```
int i = n;
while (i > 0) i /= 2;
```

O(log n)

```
for (int i = n; i > 0; i /= 3) cout << i;
```

O(log n)

```
for (int i = 0; i < n; i++)
    for (int j = 1; j < n; j *= 2)
        cout << i << j;
```

O(nlog n)

```
for (int i = 1; i * i <= n; i++) cout << i;
```

$O(\sqrt{n})$

```
int f(int n) {
    if (n == 0) return 1;
    return f(n - 1) + f(n - 1);
}
```

$O(2^n)$

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < sqrt(n); j++)
        cout << i + j;
```

$O(n\sqrt{n})$

```
for (int i = 1; i < n; i *= 2)
    for (int j = 0; j < i; j++)
        cout << j;
```

O(n)

```
for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j += i)
        cout << j;
```

$O(n \log n)$

```
for (int i = 0; i < n; i++)
    for (int j = i; j < i + 5 && j < n; j++)
        cout << i + j;
```

O(n)

```
int k = 1;
while (k * k <= n) {
    for (int j = 0; j < k; j++) cout << j;
    k++;
}
```

$O(n\sqrt{n})$

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n / 2; j++)
        cout << i + j;
```

$O(n^2)$

Let's Compare…

- $O(n \log n)$ or $O(n^2)$?

$\rightarrow$ **$O(n \log n)$**

- $O(2^n)$ or $O(n^3)$?

$\rightarrow$ **$O(n^3)$**

- $O(n^2)$ or $O(2^n)$?
- $\rightarrow$ **$O(n^2)$**
- $O(n \log n)$ or $O(n + \log n)$?
- $\rightarrow$ **$O(n + \log n)$**

- $O(n!)$ or $O(2^n)$?

$\rightarrow$ **$O(2^n)$**

- $O(\sqrt{n})$ or $O(\log n)$?

$\rightarrow$ **$O(\log n)$**

- $O(n^3)$ or $O(n^2 \log n)$?

$\rightarrow$ **$O(n^2 \log n)$**

- $O(n \log n)$ or $O(\sqrt{n})$?

$\rightarrow$ **$O(\sqrt{n})$**

- $O(\log_2 n)$ or $O(\log_{10} n)$?

$\rightarrow$ **same order**

$$O(1) < O(\log \log n) < O(\log n)$$

$$< O(\sqrt{n}) < O(n) < O(n \log n)$$

$$< O(n\sqrt{n}) < O(n^2) < O(n^2 \log n)$$

$$< O(n^3) < O(2^n) < O(3^n)$$

$$< O(n!) < O(n^n)$$

Speed Comparison: Python vs C++

- Raw python vs Libraried python!
- Cpp perform 100x faster! But…

n	O(n) Python	O(n) C++	$O(n^2)$ Python	$O(n^2)$ C++
10^3	~0.001 s	~0.0001 s	~0.01 s	~0.001 s
10^4	~0.01 s	~0.001 s	~1 s	~0.1 s
10^5	~0.1 s	~0.01 s	~100 s	~10 s
10^6	~1 s	~0.1 s	~10 000 s	~1 000 s