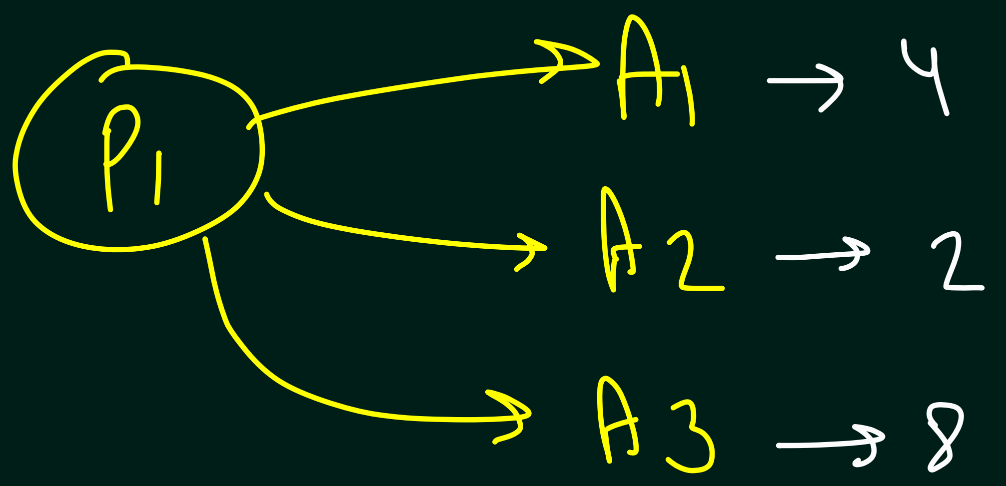
Time & Space Complexity

In This Lecture

1. What is Time Complexity?
2. The Big O Notation
3. Time Complexity of Loops
4. Time Complexity of Recursive Functions
5. Time Constraints and Time Complexity
6. Space Complexity

What is Time Complexity

The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.



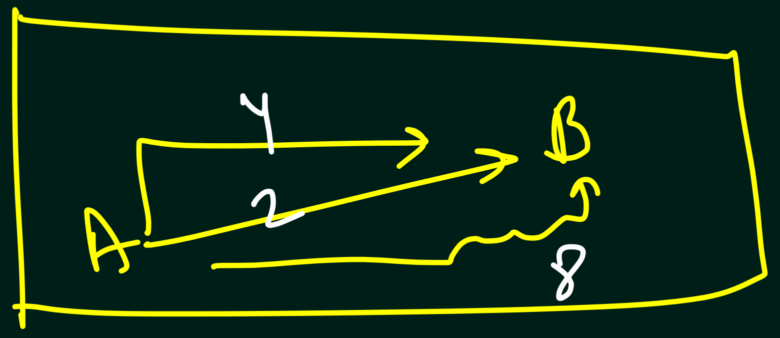
seconds

A o slow

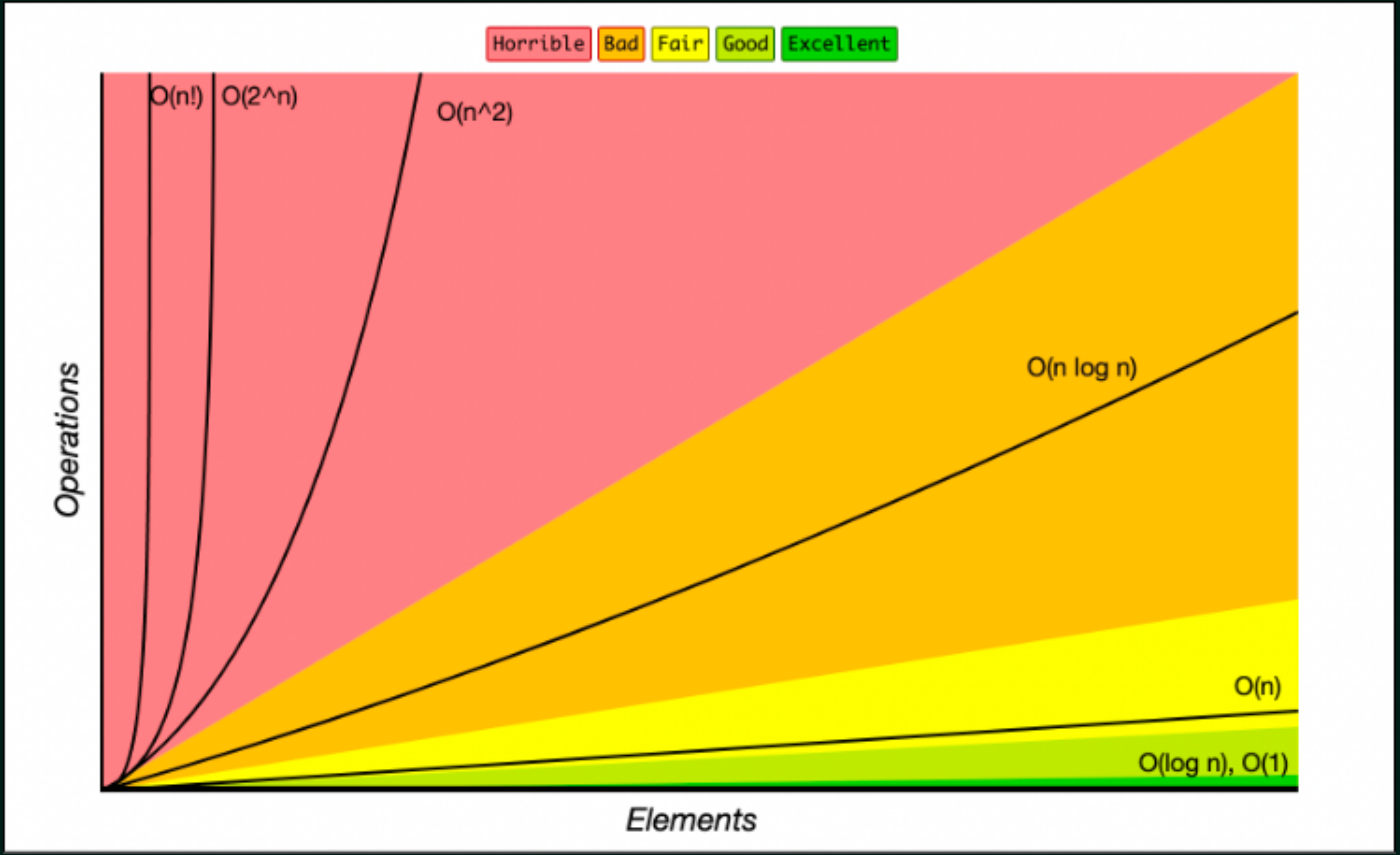
B o fast

Standard

C o super fast



The Big O Notation

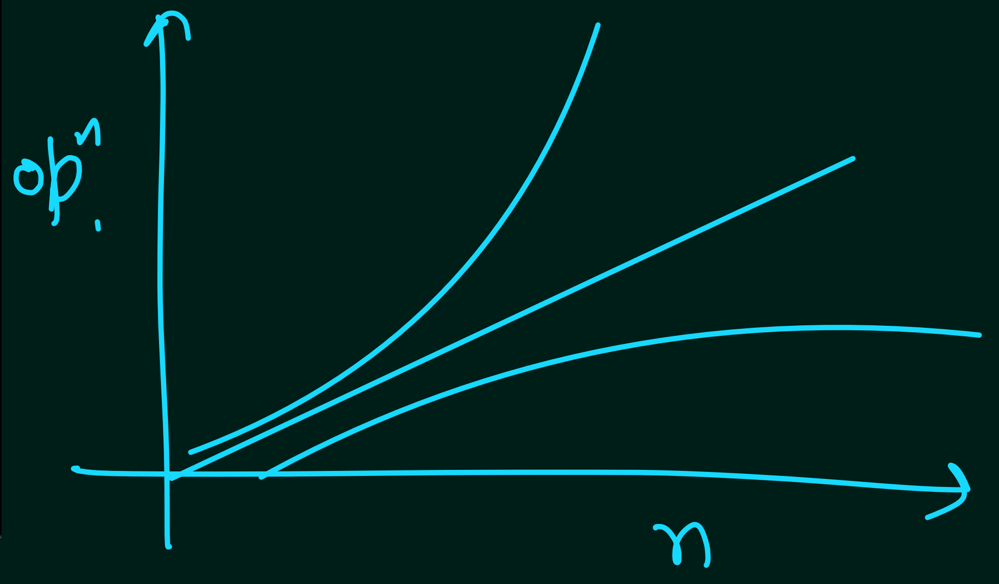


Asymptotic Notations

✓ → $O(\ )$ worst case

x → $\Theta(\ )$ Average case

x → $\Omega(\ )$ Best case



1, 2, 3 ... N

① for (1, 2, 3 .. N)

sum += i

$\boxed{O(N)}$

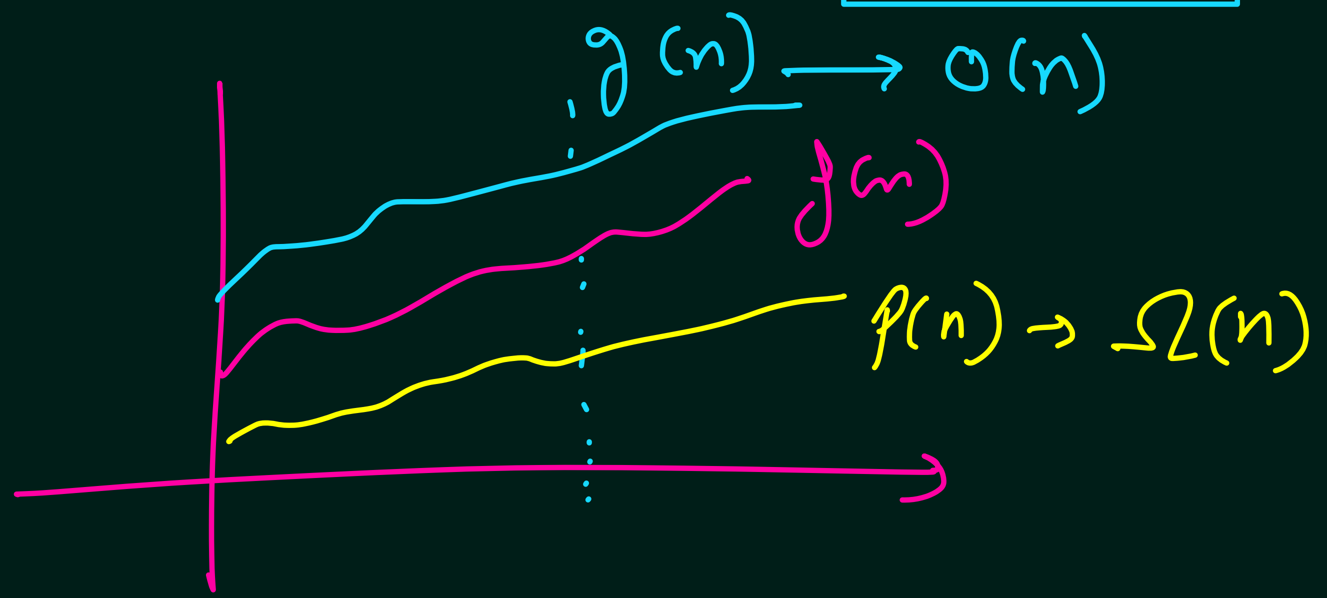
② $sum(n) = \frac{n(n+1)}{2}$

$\boxed{O(1)}$

1 o/p → 1 ms

→ 10^3 ms

→ 10^6 ms



$O(n) = g(n) > f(n)$

Time Complexity of Various Functions

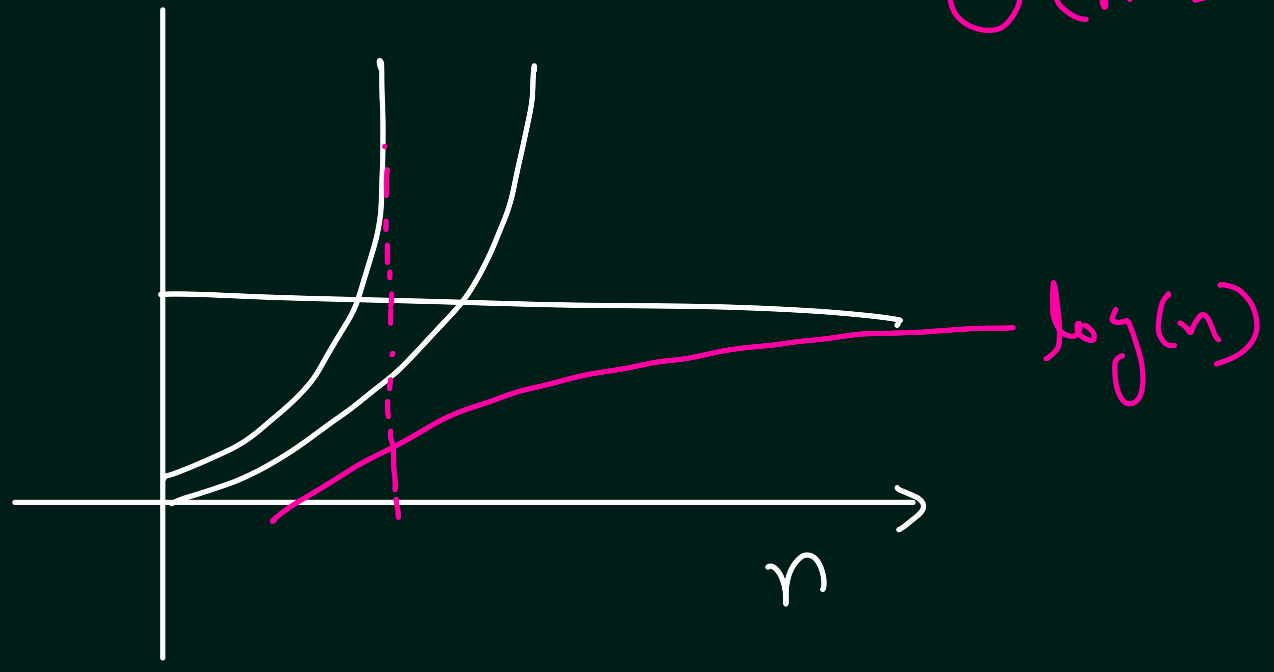
Q1. T(n) = $n^2 + \boxed{\cancel{4}n^3} + 1000$

Q2. T(n) = $40n + \boxed{n^2} + 12 \rightarrow O(n^2)$

Q3. T(n) = $n^2 + \boxed{2^n} \rightarrow O(2^n)$

Q4. T(n) = $20\log n(n) + 4n + \boxed{\cancel{3}n^2}$

$O(n^2)$



① Find the largest growing factor

② Remove the constant

$\boxed{O(n^3)}$

Time Complexity of Recursive Functions

Q1. $T(n) = T(n-1) + C$

Q2. $T(n) = 2T(n-1) + C$

Q3. $T(n) = 2T(n-1) + nC$

Q4. $T(n) = 2T(n/2) + C$

```
function(n) {
    sum = sum + n;
    function(n-1);
}
```

$T(n) = T(n-1) + C$

Time Complexity of Recursive Functions

CODING SHUTTLE

(1) $T(n) = (T(n-1)) + C$

$= ((T(n-2)) + C) + C$

$= ((T(n-3)) + C) + C + C$

$n - k = 0$

$\Rightarrow k = n$

$T(0) = 1$

$T(1) = 1$

$T(n) = T(n-k) + kC$

$T(n) = T(0) + nC$

$[T(n) = 1 + \boxed{\cancel{n}C}] \longrightarrow \boxed{O(n)}$

Time Complexity of Recursive Functions

2

$$T(n) = 2T(n-1) + C$$

$$= 2[2T(n-2) + C] + C$$

$$= 2[2[2T(n-3) + C] + C] + C$$

$$T(n) = 2^k T(n-k) + kC$$

$$= 2^n T(0) + nC$$

$T(0) = n - k = 0$

$k = n$

$T(0) = 1000$

(let's assume)

$$\boxed{T(n) = \boxed{2^n * 1\cancel{0}00} + nC} \rightarrow O(2^n)$$

Time Complexity of Recursive Functions

CODING SHUTTLE

3 $T(n) = 2T(n/2) + C$

$= 2[2T(n/4) + C] + C$

$= 2[2[2T(n/8) + C] + C] + C$

$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kC$

$= 2^{\log_2 n} T(1) + \log_2 n \, C$

$T(n) = \boxed{n * 1} + \log_2 n * C$

$O(n)$

$T(1) = \frac{n}{2^k} = 1$

$n = 2^k$

$\log_2 n = \log_2 2^k$

$= k \log_2 2$

$\boxed{k = \log_2 n}$

$x = 2^{\log_2 n}$

$\log_2 x = \log_2 2^{\log_2 n}$

$= \log_2 n \log_2 2$

$\log_2 x = \log_2 n$

$\boxed{x = n} = 2^{\log_2 n}$

Time Constraints and Time Complexity

Let n be the main variable in the problem.

- If n ≤ 12, the time complexity can be O(n!). Hardest
- If n ≤ 25, the time complexity can be O(2^n).
- If n ≤ 100, the time complexity can be O(n^4).
- If n ≤ 500, the time complexity can be O(n^3).
- If n ≤ 10^4, the time complexity can be O(n^2).
- If n ≤ 10^6, the time complexity can be O(n log n).
- If n ≤ 10^8, the time complexity can be O(n).
- If n > 10^8, the time complexity can be O(log n) or O(1). Smallest

10^8 Operations Rule

Time Constraints and Time Complexity

Space Complexity

The space Complexity of an algorithm is the total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space and space used by input.

$O(2n)$

$\hookrightarrow O(n)$

$O(n)$

$\hookrightarrow N$

```
function(n) {
// int a[] = new int[n];
// int b[] = new int[n];
   int c[] = new int[10];
   int d[][] = new int[n][n];
```

$O(10) \rightarrow \boxed{O(1)}$

$\boxed{O(n^2)} \leftarrow$ int d[][]