Complexity Classes

Presented by

Name: MD. Abdullah Al Saif
ID: 241311047
Section: B
Semester:4th

Name: Abdullah Atif
ID: 241311051
Section: B
Semester:4th

Introduction

Complexity theory is a part of Theoretical Computer Science. A complexity class is defined in terms of a type of computational problem, a model of computation, and a bounded resource like time or memory.

We usually measure this by:

- **Time Complexity:** How long it takes to solve a problem.
- **Space Complexity:** How much memory it needs.

Importance Of Complexity Classes

It helps in understanding the difficulty of computational problems

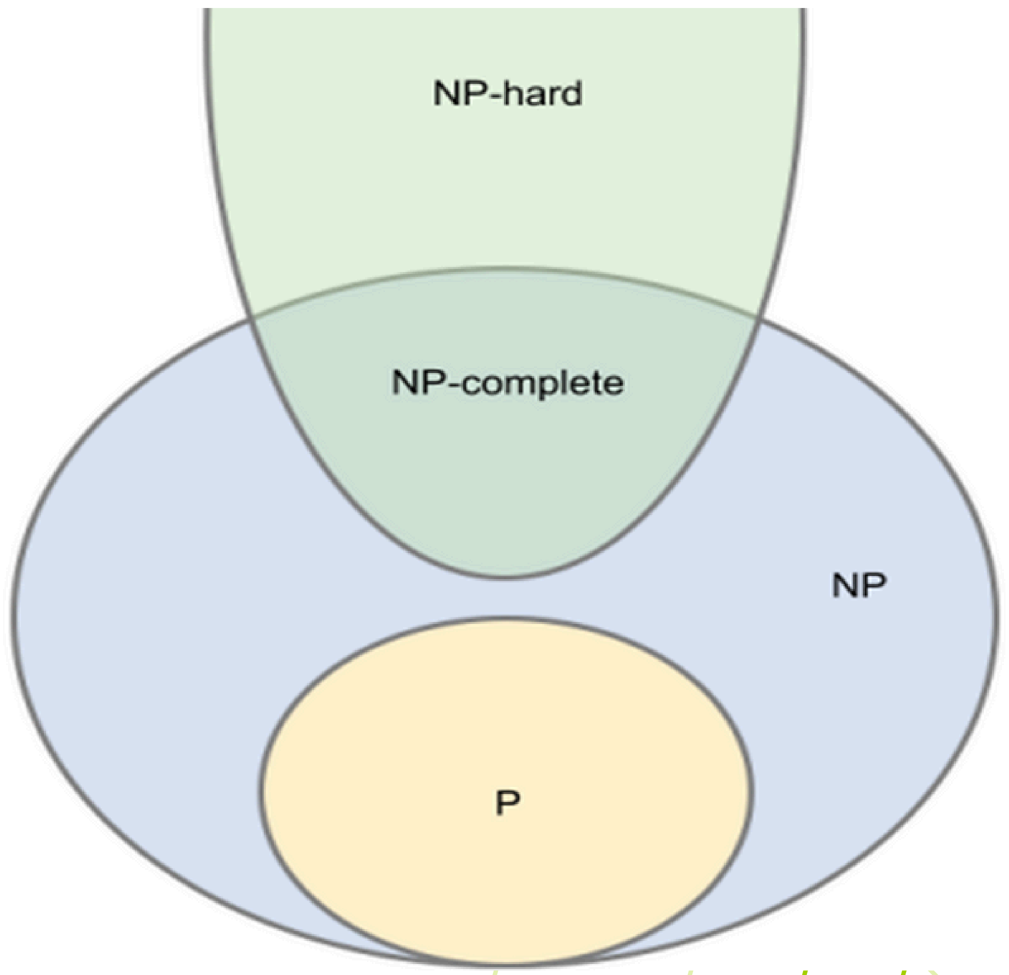
It helps in comparing different algorithms

It helps in identifying intractable problems

It helps in understanding the limits of computation

Major Complexity Classes

- P Class (Polynomial Time)
- NP Class (Non-deterministic Polynomial Time)
- NP - Complete
- NP - Hard



P Class (Polynomial Time)

P class stands for Polynomial Time. It is the collection of decision problems(problems with a "yes" or "no" answer) that can be solved by a deterministic machine (our computers) in polynomial time.

Features:

- The solution to P problems is easy to find.
- P is often a class of computational problems that are solvable within polynomial time (n, n^2, n^3.... n^c).

Examples: Calculating the greatest common divisor.
Finding a maximum matching.

- Merge Sort
- Bubble Sort
- Selection Sort
- Insertion Sort

NP Class (Non-deterministic Polynomial Time)

The NP class stands for Non-deterministic Polynomial Time. It is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time.

Features:

The solutions of the NP class might be hard to find since they are being solved by a non-deterministic machine but the solutions are easy to verify.

Examples:

- Boolean Satisfiability Problem (SAT).
- Hamiltonian Path Problem
- Traveling Salesman Problem (TSP).
- Sudoku puzzles checking

NP-Hard

An NP-hard problem is at least as hard as the hardest problem in NP and it is a class of problems such that every problem in NP reduces to NP-hard.

Features:

All NP-hard problems are not in NP.

It takes a long time to check them. This means if a solution for an NP-hard problem is given then it takes a long time to check whether it is right or not.

A problem A is in NP-hard if, for every problem L in NP, there exists a polynomial-time reduction from L to A.

Examples:

- Halting problem.
- Qualified Boolean formulas.
- No Hamiltonian cycle.

NP-Complete

A problem is NP-complete if it is both NP and NP-hard. NP-complete problems are the hard problems in NP.

Features:

NP-complete problems are special as any problem in NP class can be transformed or reduced into NP-complete problems in polynomial time.

If one could solve an NP-complete problem in polynomial time, then one could also solve any NP problem in polynomial time

Examples:

- Hamiltonian Cycle.
- Satisfiability.
- Vertex cover.

Relation of Complexty Classes

👉 $P \subseteq NP \subseteq \text{NP-Complete} \subseteq \text{NP-Hard}$

P: Problems that can be solved easily (in polynomial time).
NP: Problems whose solutions can be verified in polynomial time.
NP-Complete: The hardest problems in NP (both in NP and NP-Hard).
NP-Hard: At least as hard as NP problems but not necessarily in NP.
If P = NP, then all these classes would be equal — but this is still an unsolved question in computer science.

Applications

Complexity theory has many practical applications in various fields. Here Are Some examples:

- **Algorithm Design**
- **Game Theory**
- **Artificial Intelligence**
- **Cryptography**
- **Computer Hardware Design**
- **Optimization Problems**

Analysis

Subset Sum Problem

--- A Classic Puzzle in Computer Science

Definition:
Given a set of integers and a target sum, determine whether there exists a subset of the given numbers whose sum equals the target.

Real-World Use:

- Used in Cryptography
- Resource Allocation
- Bioinformatics

Examples

Example - 1

Input: Set = {3, 34, 4, 12,5, 2}

Target = 9

Output: subset {4, 5} sums to 9

YES

Example - 2

Input: Set = {1,2,3,4,5}

Target = 16

Output: no subset sums to 16

NO

Fun Fact:

The Subset Sum Problem is like solving a puzzle where you pick numbers from a list to hit a target sum without going over, think of it as a mathematical treasure hunt!

Algorithmic Approach
(Brute Force)

Steps :

- Generate all possible subsets of the given set (including the empty set).
- Compute the sum of each subset.
- Check if any subset's sum matches the target.

Why Brute Force?

- It guarantees finding a solution (if one exists), but it's inefficient for large sets due to the sheer number of subsets.

Algorithmic Approach (Contd)

Visualization Idea :

Imagine flipping a coin for each number in the set: "include" or "exclude." Each combination forms a subset, but with n numbers, then we get 2^n possibilities!

Limitation :

For a set with 30 numbers, we would need to check over 1 billion subsets ($2^{30} \approx 1,073,741,824$), which is obviously a large input and not a practical thing to do!

Pseudocode

```
function subsetSum(set, target):
    for each possible subset S of set:
        if sum(S) == target:
            return true
    return false
```

Complexity Analysis of SSP

Type	Explanation	Value
Time Complexity	For n elements, there are 2^n subsets to check.	$O(2^n)$
Space Complexity	Storing recursion stack or subset data.	$O(n)$
Nature of Growth	Exponential—doubles with each additional element.	Grows very fast

Complexity Analysis (Cont'd)

Illustration :

- For n = 10: $2^{10} = 1{,}024$ subsets (manageable).
- For n = 20: $2^{20} \approx 1$ million subsets (slow).
- For n = 30: $2^{30} \approx 1$ billion subsets (impractical without optimization).

Key Insight :

The exponential growth makes brute force infeasible for large datasets, pushing us to seek smarter solutions.

Classification

Why is SSP NP-Complete?

NP : Problems where a solution can be verified quickly (in polynomial time).
NP-complete: The hardest problems in NP; if we can solve one efficiently, we can solve all NP problems efficiently.

SSP is NP-complete because it can be reduced to other NP-complete problems (e.g., napsack, Partition).

Classification (Cont'd)

Property	Description
Problem Type	Decision Problem (returns "Yes" or "No").
Solution Verification	Easy: Given a subset, summing its elements to check if it equals the target is O(n).
Finding the Subset	Hard: Testing all subsets requires $O(2^n)$ time.
Class	NP-complete: Verifiable in polynomial time, but solving is exponentially hard.

Summary

Concept	Explanation
Problem Name	Subset Sum
Goal	Check if a subset sums to a target
Time Complexity	$O(2^n)$ (brute force)
Space Complexity	$O(n)$ (brute force)
Class	NP-complete
In Short	Easy to verify, hard to solve

Thank You!