# Casual conversation

hi gpt

Hey Abdullah! 👋

How are you doing today?

well you

I'm doing great, thanks for asking! 😄

What are we working on today — coding, HTML learning, or something else?

next week i have presentation about complexity classes. so i want to learn about it deaply to creat slide and present it well.

Perfect! That's a great topic — *Complexity Classes* is one of the most important areas in **Theoretical Computer Science** and **Algorithm Analysis**.

Let's make sure you understand it deeply *and* prepare a solid presentation.

We'll go step-by-step like this:

### 🧠 Step Plan

- 1. Basic Concepts What is Time & Space Complexity?
- **2.** Understanding Big-O,  $\Omega$ ,  $\Theta$  Asymptotic notation.
- 3. What are Complexity Classes? The idea of classifying problems by difficulty.
- 4. Major Classes Explained (with examples):
  - P
  - NP
  - NP-Complete
  - NP-Hard
- 5. Relations between these classes (Venn Diagram style explanation).
- 6. Real-world examples and analogies.

- 7. Slide-by-slide breakdown for your presentation.
- 8. Pitch (how to explain each slide smoothly).

Would you like me to start from **Step 1** (**Basic Concepts of Time and Space Complexity**) so you build a strong foundation first — or jump directly to **Step 3** (**Complexity Classes Overview**) since that's your presentation topic?

step 1

Excellent choice, Abdullah 👏

Let's start from the very base — **Step 1: Basic Concepts of Time and Space Complexity**.

### Step 1: Basic Concepts of Time and Space Complexity

1. What is "Complexity" in Algorithms?

#### **Complexity** means:

How much *time* and *memory* (space) an algorithm needs as the size of the input increases.

So, complexity tells us how efficiently an algorithm solves a problem.

### 2. Two Types of Complexity

Туре	Description	Example
Time Complexity	How much time (steps or operations) an algorithm takes to complete.	Counting how many comparisons occur in sorting an array.
Space Complexity	How much extra memory (RAM) the algorithm uses.	Extra arrays, variables, recursion stack, etc.

### • 3. Input Size and Growth

We represent the size of the input as **n**.

### Example:

- Searching in an array of 10 items  $\rightarrow$  n = 10
- Sorting 1,000 names  $\rightarrow$  n = 1000

When **n** grows, we study how the *number of steps* grows with it.

### 4. Measuring Time Complexity

We don't measure in seconds (because hardware differs).

We measure in **number of operations**, relative to input size.

#### Example:

```
for (int i = 0; i < n; i++) {
   cout << i;
}</pre>
```

 $\Rightarrow$  Runs **n times**  $\rightarrow$  **O(n)** time complexity.

### 5. Common Time Complexities

Notation	Name	Example	Performance
O(1)	Constant	Accessing array element	Fastest
O(log n)	Logarithmic	Binary search	Very fast
O(n)	Linear	Simple loop	Moderate
O(n log n)	Linearithmic	Merge sort	<b>♯</b> Efficient
O(n²)	Quadratic	Nested loops	Slower

Notation	Name	Example	Performance
O(2 <sup>n</sup> )	Exponential	Recursive subset generation	🧨 Very slow
O(n!)	Factorial	Traveling Salesman brute-force	<b>○</b> Impractical

### 6. Space Complexity Example

```
int sum = 0; // uses constant space O(1)
int arr[n]; // uses O(n) space
```

Recursion often increases space use due to call stack.

### 7. Why Do We Study This?

#### Because:

- It helps us choose **faster algorithms**.
- It shows **what's possible** and **what's not** for large data.
- It's the foundation for Complexity Classes (P, NP, etc.).

### In short:

Time and Space Complexity tell us *how scalable* an algorithm is — when the problem grows big.

Would you like me to move to **Step 2: Asymptotic Notations (Big O, Omega, Theta)** next?

That's the mathematical way we *express* these complexities clearly.

step 2

Awesome  $\red{6}$  Abdullah — now that you clearly understand what **time** and **space complexity** are, let's go deep into **Step 2: Asymptotic Notations** (**Big O, \Omega, \Theta**).

This is where we learn the *mathematical language* used to describe algorithm efficiency.

# Step 2: Asymptotic Notations

### • 1. What is Asymptotic Analysis?

When we analyze algorithms, we care about how they perform when **input size** (n) becomes **very large**.

We ignore small details like:

- Exact number of operations
- Machine speed
- Constant multipliers

Instead, we focus on the **growth rate** — how the runtime increases as n increases.

#### 2. The Three Main Notations

Notation	Meaning	Describes	Example
Big O (O)	Upper Bound	Worst-case performance	Sorting worst-case time
Omega (Ω)	Lower Bound	Best-case performance	Best possible runtime
Theta (Θ)	Tight Bound	Average or exact rate	Typical runtime

### 🧠 3. Big O (O) – "At Most This Fast"

Big O gives an **upper limit** — the algorithm *won't be slower than this* (in terms of order).

Example:

```
for (int i = 0; i < n; i++) {
   cout << i;
}</pre>
```

 $\rightarrow$  **O(n)** — runs *at most linearly* with n.

Example with nested loop:

```
for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j++) {
      cout << i + j;
   }
}</pre>
```

 $\rightarrow$  O(n<sup>2</sup>) — grows *quadratically*.

### $\neq$ 4. Omega ( $\Omega$ ) – "At Least This Fast"

Omega gives a **lower limit** — the algorithm *can't be faster than this*.

#### Example:

If a sorting algorithm takes at least  $n \log n$  comparisons in the best case,  $\rightarrow \Omega(n \log n)$ .

### 1 5. Theta (Θ) – "Exactly This Fast"

If the **upper** and **lower** bounds are the same order, we call it **Theta**  $(\Theta)$ .

#### Example:

• Loop from 1 to  $n \to O(n)$  and  $\Omega(n)$  both  $\to$  so  $\Theta(n)$ 

Meaning: It always grows linearly.

### • 6. Visual Understanding

- big O = ceiling (worst case)
- $\leftarrow$  Ω = floor (best case)
- $\leftarrow$  Θ = middle (tight bound)

### • 7. Example Summary

Code	Big O	Omega	Theta
for(i=0;i <n;i++)< td=""><td>O(n)</td><td>Ω(n)</td><td>Θ(n)</td></n;i++)<>	O(n)	Ω(n)	Θ(n)
Nested loop (n×n)	O(n²)	$\Omega(n^2)$	$\Theta(n^2)$
Binary search	O(log n)	Ω(1)	Θ(log n)
Bubble sort	O(n²)	Ω(n)	$\Theta(n^2)$

### • 8. Why It Matters

- **Big O** → tells you the **worst-case performance** (most used).
- $\Omega \rightarrow$  shows **best case**.
- $\Theta \rightarrow$  shows **average case** (or exact bound).

When designing algorithms, we mostly focus on **Big O** since it guarantees how bad things can get.

### In short:

Asymptotic notations help describe how an algorithm's runtime grows with input size — ignoring constants and focusing on scalability.

Would you like me to continue to **Step 3: Complexity Classes Overview (P, NP, NP-Complete, NP-Hard)** next?

That's where we connect all this theory to real algorithm categories like "easy vs impossible problems."

step 3 in details please

Perfect, Abdullah 🔥

Now we're entering the **main heart** of your presentation topic —

Step 3: Complexity Classes Overview (P, NP, NP-Complete, NP-Hard)

Let's go **deep but clear**, so you can easily create slides **and** explain confidently during your presentation.

# Step 3: Complexity Classes Overview

1. What are Complexity Classes?

A **complexity class** groups problems based on **how much time or space** they need to solve **as input grows**.

← Think of them like difficulty levels for computational problems.

We study them mainly using time complexity (how long algorithms take).

### 🔅 2. Basic Idea: Decision Problems

Most complexity classes deal with **decision problems** (yes/no questions).

For example:

- "Is there a path from A to B with cost < 10?"</li>
- "Can this graph be colored using 3 colors?"

"Is there a subset of numbers that sums to 0?"

These are used because they make mathematical classification easier.

### 🗩 3. Major Complexity Classes

Let's go one by one carefully 👇

### (1) Class P (Polynomial Time)

#### • Definition:

All problems that can be **solved efficiently** (in *polynomial time*) by a **deterministic** algorithm (normal computer).

#### Meaning:

You can find the answer quickly — in time like O(n),  $O(n^2)$ ,  $O(n^3)$  ... but not  $O(2^n)$  or worse.

#### • Examples:

- Sorting numbers  $\rightarrow$  O(n log n)
- Finding shortest path (Dijkstra's algorithm)  $\rightarrow$  O(n<sup>2</sup>)
- Searching (Binary Search) → O(log n)

#### • In short:

**V** Easy problems, solvable efficiently.

### (2) Class NP (Nondeterministic Polynomial Time)

#### • Definition:

Problems where, if someone gives you a *solution*, you can **verify** it *quickly* (in polynomial time) — even if finding it might be **very hard**.

#### • Meaning:

It's easy to *check* a solution, but maybe hard to *find* it.

#### Examples:

- Sudoku puzzle: Easy to check if a filled grid is correct, but hard to find the solution.
- Traveling Salesman Problem (TSP): Easy to verify if a given route is short enough,

but finding that route is difficult.

#### • In short:

⚠ Verification is easy; solving might be hard.

### (3) Class NP-Complete (NPC)

#### • Definition:

The hardest problems in NP.

If you can solve **any** NP-Complete problem efficiently, you can solve **all NP problems** efficiently.

#### Two conditions:

- 1. The problem is in NP (verifiable in polynomial time).
- 2. Every NP problem can be *reduced* to it in polynomial time.

#### • Examples:

- Traveling Salesman Problem (decision version)
- Knapsack Problem
- Boolean Satisfiability Problem (SAT)
- Graph Coloring
- Subset Sum

#### • In short:

A Hardest of the hard problems that can still be verified quickly.

### (4) Class NP-Hard

#### Definition:

Problems at least as hard as NP-Complete problems.

They may or may not be in NP (might not even be "decision problems").

#### Meaning:

These are even harder — sometimes you can't even check the solution easily.

#### Examples:

- Optimization version of Traveling Salesman (find the *shortest* route).
- Halting Problem (undecidable).
- Scheduling problems.

#### • In short:

• At least as hard as NP-Complete, maybe harder.

### 🗮 4. Relationship Between the Classes

Here's how they're related conceptually:

CSS

### $P \subseteq NP \subseteq NP$ -Complete $\subseteq NP$ -Hard

- Every problem in P is also in NP
   (if you can solve it fast, you can verify it fast).
- Every **NP-Complete** problem is in **NP**.
- Every NP-Complete problem is also NP-Hard.
- Some NP-Hard problems are not even in NP (too difficult to verify).

### 5. The Famous Open Question: "P vs NP"

Is P = NP?

#### In simple terms:

Are all problems that are easy to verify also easy to solve?

If yes  $\rightarrow$  we could solve many currently impossible problems efficiently.

If no  $\rightarrow$  there will always be hard problems computers can't handle quickly.

∀ Till today, no one knows the answer — it's one of the biggest unsolved problems
 in computer science (worth \$1 million prize).

### of 6. Example to Understand Easily

Problem	Can we Solve Fast?	Can we Verify Fast?	Class
Sorting numbers	<b>✓</b> Yes	<b>✓</b> Yes	Р
Sudoku	X Hard to solve	<b>Z</b> Easy to check	NP
3-SAT	X Hard to solve	Easy to check	NP-Complete
TSP (shortest route)	XX Very hard	<b>Z</b> Easy to check	NP-Hard

### 🧠 7. Real-World Analogy

#### Think of:

- **P:** Solving a jigsaw puzzle with all the steps shown.
- **NP:** Checking if someone else's completed puzzle is correct easy!
- **NP-Complete:** The hardest kind of puzzle where checking is easy, but solving is extremely hard.
- **NP-Hard:** A puzzle that's so hard, even checking might be impossible.

### In summary:

Class	Meaning	Example	Difficulty
P	Solvable in polynomial time	Binary Search, Sorting	Easy
NP	Verifiable in polynomial time	Sudoku, TSP	Hard to solve
NP-Complete	Hardest problems in NP	SAT, Knapsack	Very hard
NP-Hard	At least as hard as NP-Complete	Halting Problem	Possibly unsolvable

Would you like me to move to **Step 4: Relations + Venn Diagram & Visual Explanation** next (for your slides)?

It'll show graphically how P, NP, NP-Complete, and NP-Hard connect — very useful for explaining in presentation.

step 4 also in detail and cleared

Excellent, Abdullah! 🌟

Now that you fully understand **what each class (P, NP, NP-Complete, NP-Hard)** means, let's go into **Step 4**, where we make everything **visual, logical, and crystal clear** — perfect for presentation slides.

# Step 4: Relationship Between Complexity Classes (with Diagrams & Examples)

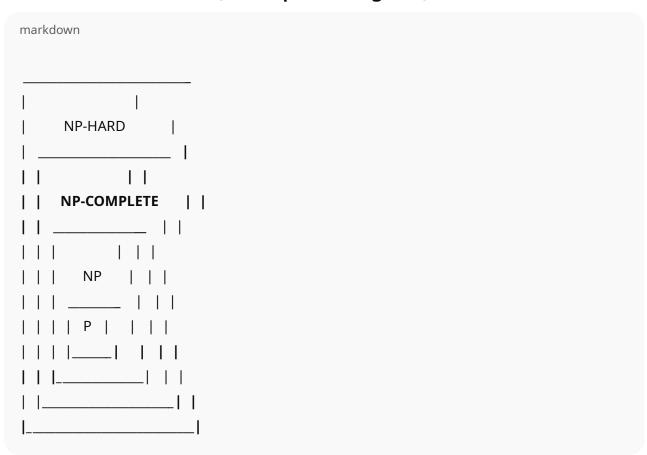
### 1. Why Relationships Matter

All the complexity classes (P, NP, NP-Complete, NP-Hard) are **connected** — they're not separate worlds.

To understand the difference, you need to know:

- Which problems are contained inside others,
- Which ones are harder or equal, and
- What we **don't know yet** (the big mystery: "P = NP?")

### 2. Visual Overview (Conceptual Diagram)



#### **Explanation:**

- **P** is the smallest circle (easy problems).
- **NP** contains **P** (includes all problems that can be *verified* easily).
- NP-Complete problems are the *hardest* ones inside NP.

• **NP-Hard** covers everything that's *as hard or harder* than NP-Complete. (Includes some problems even outside NP.)

### 3. Understanding Each Relation Clearly

### $\square$ P $\subseteq$ NP

- Every problem you can solve quickly, you can also verify quickly. Example:
  - Sorting → You can both sort (solve) and check the order (verify) easily.
- So, all P problems are NP problems, but not all NP problems are P.

### NP-Complete ⊆ NP

- NP-Complete problems belong to NP, meaning they can be verified quickly.
- But they are also the hardest in NP.

If you find a fast (P) algorithm for **even one** NP-Complete problem,

⇒ all NP problems become solvable fast too!

That's why NP-Complete problems are very important — they act like "bridges."

### ■ NP-Hard ⊇ NP-Complete

- All NP-Complete problems are NP-Hard (they're the hardest inside NP).
- But NP-Hard also includes some problems not in NP, like optimization or undecidable problems.

#### Example:

- "Find the shortest route" → NP-Hard
- "Is there a route shorter than 50 km?" → NP-Complete

The second one is *decision version*, so it's in NP.

The first one is optimization version, so NP-Hard (not necessarily in NP).

### 6 Unknown: Is P = NP?

This is the big question.

- If **P** = **NP**, it means every problem that can be verified quickly can also be solved quickly.
- If **P** =/N**P**, it means some problems can be verified fast but not solved fast.

Most experts **believe P ≠**/**NP**, but no one has proven it yet.

**s** The "P vs NP" problem is one of the **Millennium Prize Problems** — worth **\$1 million USD** if solved.

### 4. Real-World Analogy

Let's make it more intuitive 👇

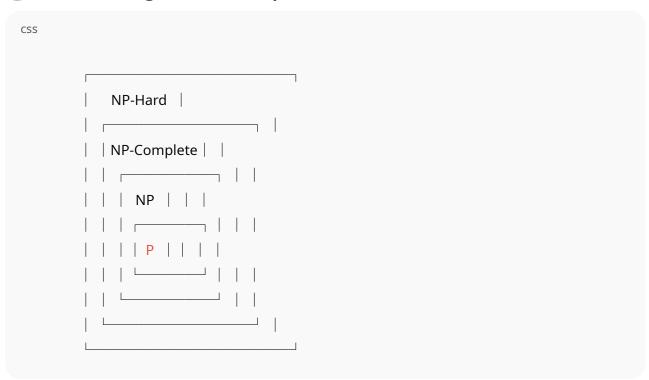
Class	Analogy	Meaning
P	Solving a simple puzzle like a 3×3 Rubik's cube	You can solve it quickly yourself
NP	Checking if a completed big puzzle is correct	Easy to verify if someone shows the solution
NP-Complete	The hardest puzzles that are easy to check	If you solve one, you can solve them all
NP-Hard	A puzzle that may not even have a clear rule or check	Might be impossible to solve or verify efficiently

### **5. Example Table of Problems by Class**

Problem	Class	Can Solve Fast?	Can Verify Fast?	Notes
Sorting Numbers	Р	<b>V</b>	V	Deterministic & efficient
Binary Search	Р	V	V	Simple
Sudoku	NP	×	V	Hard to find, easy to check

Problem	Class	Can Solve Fast?	Can Verify Fast?	Notes
3-SAT	NP-Complete	×	<b>V</b>	Canonical NP- Complete problem
TSP (Decision version)	NP-Complete	×	V	Classical example
TSP (Optimization)	NP-Hard	×	×	Harder than NP
Halting Problem	NP-Hard (undecidable)	×	×	Impossible to solve in general

### 6. Venn Diagram with Explanation



### **Explanation of Circles:**

- **P** = efficiently solvable
- **NP** = efficiently verifiable
- NP-Complete = hardest of NP
- **NP-Hard** = at least as hard as NP-Complete (might not be verifiable)

# 🧠 7. Simple Way to Remember

Class	Key Idea	Think Of
P	Problems easy to solve	Homework you can do fast
NP	Problems easy to check	Checking a friend's answer
NP-Complete	Hardest in NP	Final exam problem
NP-Hard	Harder than NP	Impossible assignment 😄

### 8. Summary for Your Presentation Slide

Title: "Relationships Between Complexity Classes"

#### **Key Points for Slide:**

- 1.  $P \subseteq NP \subseteq NP$ -Complete  $\subseteq NP$ -Hard
- **2.** P = Solvable efficiently
- **3.** NP = Verifiable efficiently
- **4.** NP-Complete = Hardest in NP
- **5.** NP-Hard = At least as hard as NP-Complete
- **6.** "P vs NP" is an unsolved million-dollar problem

Would you like me to move to Step 5: Real-life Examples + Slide-by-Slide

#### **Presentation Outline next?**

That's where I'll help you structure your slides and speech (pitch) to present smoothly in class.

#### Step 5:

do it very carefully, simple, all needs should be included, as a profetional and easy to explain without pitch

Perfect, Abdullah 👏

Now we're at Step 5: Real-life Examples + Slide-by-Slide Presentation Outline written **professionally**, **clearly**, and **in simple language** so you can directly use it for your slides.

No pitch — only perfect **content** and **structure**.



### Step 5: Complete Presentation Outline

**Topic:** Complexity Classes in Computational Theory

Goal: Explain what complexity classes are, their types (P, NP, NP-Complete, NP-Hard),

and how they relate.



### 💻 Slide 1: Title Slide

**Title:** Complexity Classes

Subtitle: Understanding P, NP, NP-Complete, and NP-Hard

**Details:** 

Name: Abdullah Al Mahfuz

Course: [Your course name]

Instructor: [Instructor name]

Date: [Your presentation date]



### 🧠 Slide 2: Introduction to Complexity

**Heading:** What is Computational Complexity?

#### **Points:**

- Computational Complexity studies how efficiently algorithms solve problems.
- It measures:
  - Time Complexity: How long an algorithm takes.
  - Space Complexity: How much memory it needs.
- Helps us understand which problems can be solved efficiently and which cannot.

#### **Example:**

 Sorting 1,000 numbers takes more time than sorting 10 numbers → complexity depends on input size (n).



## Slide 3: Types of Complexity

**Heading:** Two Major Types

Туре	Definition	Example
Time Complexity	Amount of time (steps) required to complete the task.	Number of operations in sorting.
Space Complexity	Extra memory required to complete the task.	Temporary arrays or recursion stack.

#### **Key Point:**

Both time and space grow as input size increases — complexity tells us how fast or slow.

# Slide 4: Asymptotic Notations

**Heading:** Expressing Complexity Mathematically

#### **Main Notations:**

Symbol	Meaning	Used For
O (Big O)	Upper bound	Worst case
Ω (Omega)	Lower bound	Best case
Θ (Theta)	Tight bound	Average case

#### **Example:**

Loop from 1 to  $n \rightarrow O(n)$ ,  $\Omega(n)$ ,  $\Theta(n)$ 

#### Note:

Big O notation is most commonly used to express performance.

### Slide 5: What Are Complexity Classes?

Heading: The Idea of Classes

#### Points:

- Complexity classes group problems by how difficult they are to solve.
- They show which problems computers can solve efficiently and which are too hard.
- Mainly based on **time complexity** using **polynomial time** as a limit of efficiency.

### **Example Question:**

Can we solve this problem fast (like  $n^2$  or  $n^3$ ) — or only very slowly (like  $2^n$ )?

# ✓ Slide 6: Class P (Polynomial Time)

Heading: P - Problems Solvable Quickly

#### **Definition:**

- Set of problems that can be solved efficiently (in polynomial time).
- Algorithms with time like O(n), O(n²), O(n³), etc.

### **Examples:**

- Binary Search  $\rightarrow$  O(log n)
- Merge Sort  $\rightarrow$  O(n log n)
- Dijkstra's Algorithm  $\rightarrow$  O(n<sup>2</sup>)

#### **Key Idea:**

 $\bigvee$  P = Easy to solve and easy to verify.

### Slide 7: Class NP (Nondeterministic Polynomial Time)

Heading: NP - Easy to Verify, Hard to Solve

#### **Definition:**

• Problems where a given solution can be **verified** in polynomial time, but finding that solution may be **hard**.

### **Examples:**

- Sudoku
- Subset Sum Problem
- Traveling Salesman Problem (decision form)

#### **Key Idea:**

⚠ We can check the answer quickly but finding it is difficult.

### Slide 8: Class NP-Complete

**Heading:** NP-Complete – The Hardest in NP

#### **Definition:**

- Problems that are:
  - 1. In NP (easy to verify)
  - 2. As hard as any other NP problem (if you can solve one, you can solve all)

#### **Examples:**

- 3-SAT (Boolean Satisfiability Problem)
- Knapsack Problem
- Graph Coloring
- Traveling Salesman (decision version)

#### **Key Idea:**

A Hardest problems that are still verifiable.

### 🔥 Slide 9: Class NP-Hard

Heading: NP-Hard – At Least as Hard as NP-Complete

#### **Definition:**

- Problems that are at least as hard as NP-Complete.
- They might not even be in NP (not necessarily decision problems).

#### **Examples:**

- TSP (optimization version) find the shortest route.
- Halting Problem check if a program will stop or run forever.

#### **Key Idea:**

Solving or verifying them efficiently may be impossible.

### Slide 10: Relationships Between Classes

**Heading:** How the Classes Are Connected

### Hierarchy:

```
P \subseteq NP \subseteq NP-Complete \subseteq NP-Hard
```

#### **Explanation:**

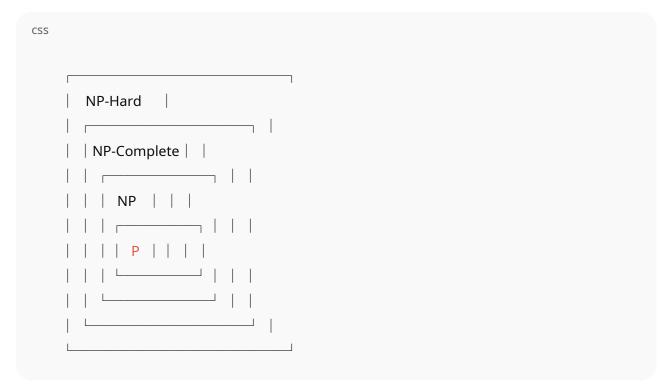
- Every problem in P is also in NP.
- NP-Complete problems are the hardest inside NP.
- NP-Hard includes NP-Complete and problems even harder.

#### Note:

No one knows whether P = NP or  $P \neq NP$  — it's still an unsolved question.

# 🗩 Slide 11: Diagram Representation

**Heading:** Visual Representation



#### Simple Meaning:

- **P:** Easy problems
- NP: Easy to verify

- NP-Complete: Hardest of NP
- **NP-Hard:** Hardest overall (even outside NP)

### Slide 12: Real-World Examples

Class	Problem	Description
P	Sorting, Searching	Easily solvable
NP	Sudoku	Hard to find, easy to verify
NP-Complete	3-SAT, Graph Coloring	Hardest in NP
NP-Hard	TSP (Optimization), Halting Problem	Extremely hard or unsolvable

#### **Key Takeaway:**

Complexity classes help classify real-life computational problems based on their difficulty.



### Slide 13: The "P vs NP" Question

**Heading:** The Biggest Open Problem

#### **Question:**

Is P = NP?

#### If yes:

All hard problems could be solved quickly — major revolution in computer science.

#### If no:

Some problems will always remain hard, no matter how powerful computers get.

#### **Status:**

Still unsolved — one of the Millennium Prize Problems (worth \$1 million USD).

### **6** Slide 14: Summary

Class	Meaning	Example
Р	Solvable efficiently	Sorting, Searching
NP	Verifiable efficiently	Sudoku
NP-Complete	Hardest in NP	3-SAT, Knapsack
NP-Hard	At least as hard as NP-Complete	Halting Problem

#### Core Idea:

- $P \subseteq NP \subseteq NP$ -Complete  $\subseteq NP$ -Hard
- P vs NP is still an open question.
- Understanding these helps design better algorithms.

### Slide 15: Conclusion

**Heading:** Final Thoughts

#### **Points:**

- Complexity classes describe the limits of computation.
- They help identify which problems are feasible and which are not.
- Most real-world efficient algorithms fall under **P**.
- NP, NP-Complete, and NP-Hard define the boundary of what is computationally possible.
- The question "P = NP?" continues to challenge computer science today.

### 📚 Slide 16: References

**Heading:** References (For Academic Credit)

- "Introduction to Algorithms" Cormen, Leiserson, Rivest & Stein
- "Computational Complexity" Christos Papadimitriou
- GeeksforGeeks Complexity Classes
- TutorialsPoint Theory of Computation Notes

Wikipedia – P vs NP Problem

### **V** Done:

This structure is complete, professional, easy to explain, and covers everything definitions, examples, diagrams, hierarchy, and real-world connection.

Would you like me to make a **PowerPoint-style version** of these slides next (readyto-use content for each slide)?

It'll include short bullet points and slide titles formatted exactly like presentation text.

now give me the pitch for the slides in simple and easy to explain

Perfect timing, Abdullah 👍



Now that your slides are fully ready, here's your simple, clear, and confident pitch (spoken explanation) for each slide — so you can present naturally without reading directly from text.

Each pitch sounds like you're talking to your classmates — easy English, but professional.

Let's go one by one 👇



#### Slide 1: Title Slide

#### Pitch:

"Assalamu Alaikum everyone.

Today I'm going to present a very important topic in computer science — *Complexity* Classes.

We'll learn what they mean, how they're connected, and why they matter in algorithm design."



### Slide 2: Introduction to Complexity

#### Pitch:

"Every algorithm we create has two main limits — how fast it runs and how much memory it uses.

This is called *computational complexity*.

It helps us understand which problems computers can solve quickly and which are too slow or even impossible to solve efficiently."



### Slide 3: Types of Complexity

#### Pitch:

"There are two main types:

Time complexity means how long an algorithm takes to run, and **space complexity** means how much memory it needs.

When our input grows bigger, both time and space also grow — that's why studying complexity is so important."



### Slide 4: Asymptotic Notations

#### Pitch:

"To describe algorithm performance mathematically, we use special symbols: Big O for the worst case, Omega for the best case, and Theta for the average or exact case.

For example, a loop running 'n' times has O(n) complexity — meaning it grows linearly with input size."



### Slide 5: What Are Complexity Classes

#### Pitch:

"Now let's talk about complexity classes.

They are groups of problems divided by how difficult they are to solve.

Some problems can be solved very fast, while others take huge amounts of time and these classes help us understand that difference."



### 

#### Pitch:

"Class P contains all the problems that can be solved efficiently — in polynomial time." For example, sorting numbers or finding the shortest path in a graph.

These problems are easy both to solve and to verify, so they are considered 'feasible' for computers."

### Slide 7: Class NP (Nondeterministic Polynomial Time)

#### Pitch:

"Next, we have class NP.

These are problems that are easy to check but hard to solve.

If someone gives you the answer, you can verify it quickly — but finding that answer by yourself might take a very long time.

Sudoku is a great example — checking a solution is easy, but solving it can take hours."

### Slide 8: Class NP-Complete

#### Pitch:

"NP-Complete problems are the hardest problems inside NP.

They are both in NP, and as hard as any other NP problem.

That means if we could solve even one NP-Complete problem quickly, then we could solve all NP problems quickly.

Examples include the 3-SAT problem, Knapsack, and Traveling Salesman (decision version)."



### Slide 9: Class NP-Hard

#### Pitch:

"NP-Hard problems are at least as hard as NP-Complete problems, and sometimes even harder.

They might not even have a quick way to verify the solution.

For example, the optimization version of the Traveling Salesman Problem or the Halting Problem.

These problems are generally considered impossible to solve efficiently."

### 🗭 Slide 10: Relationships Between Classes

#### Pitch:

"Here's how all these classes connect."

P is a subset of NP, NP-Complete lies inside NP, and NP-Hard includes everything that's as hard or harder.

The big unsolved question in computer science is — is P equal to NP?

If yes, it would change everything about how we solve problems."



### 🗩 Slide 11: Diagram Representation

#### Pitch:

"This diagram shows the relationship visually.

The smallest circle is P — easy problems.

NP includes both easy and hard-to-solve but easy-to-check problems.

NP-Complete sits at the edge of NP, being the hardest inside it.

And NP-Hard is the outer layer, containing the toughest of all problems."



### Slide 12: Real-World Examples

#### Pitch:

"Let's look at some real-world examples.

Sorting and searching are P problems — easy for computers.

Sudoku is NP — difficult to solve but easy to check.

3-SAT or Knapsack are NP-Complete — extremely hard.

And optimization problems like TSP or Halting Problem are NP-Hard — often impossible to solve efficiently."



### 💡 Slide 13: The P vs NP Question

#### Pitch:

"The most famous open question is: Is P equal to NP?

If it's true, we could solve every problem that can be checked quickly.

If it's false, some problems will always remain hard.

No one knows the answer yet — and solving it would win a one-million-dollar prize."

### @ Slide 14: Summary

#### Pitch:

"To summarize —

P problems are easy to solve,

NP problems are easy to verify,

NP-Complete are the hardest in NP,

and NP-Hard are at least as hard as NP-Complete.

The key relationship is:

 $P \subseteq NP \subseteq NP$ -Complete  $\subseteq NP$ -Hard."

### Slide 15: Conclusion

#### Pitch:

"In conclusion, complexity classes help us understand the limits of what computers can and cannot do efficiently.

They separate easy problems from impossible ones.

Most real-world algorithms fall into class P, but NP, NP-Complete, and NP-Hard remind us of the challenges that still remain unsolved in computer science."

### Slide 16: References

#### Pitch:

"These are the main sources I used — standard books like *Introduction to Algorithms* and reliable sites like GeeksforGeeks and Wikipedia for better understanding."

### **Tips for Delivery:**

- Speak slowly and clearly.
- Give short pauses between slides.
- Emphasize key words like "easy," "hard," "verify," "solve," "unsolved question."
- You don't need to memorize just understand the ideas so it sounds natural.

Would you like me to now make this into a **PowerPoint (PPTX) file** with slides + your spoken notes (pitch in speaker notes section)?

You could use it directly for class.