# ANALYSIS OF ALGORITHMS

COMP 2230-02 | WINTER 2026

Instructor : Shivani Tyagi, Department of Computing Science

## In Previous Class

**Data** is any raw or unstructured piece of information that can be processed to get some information.

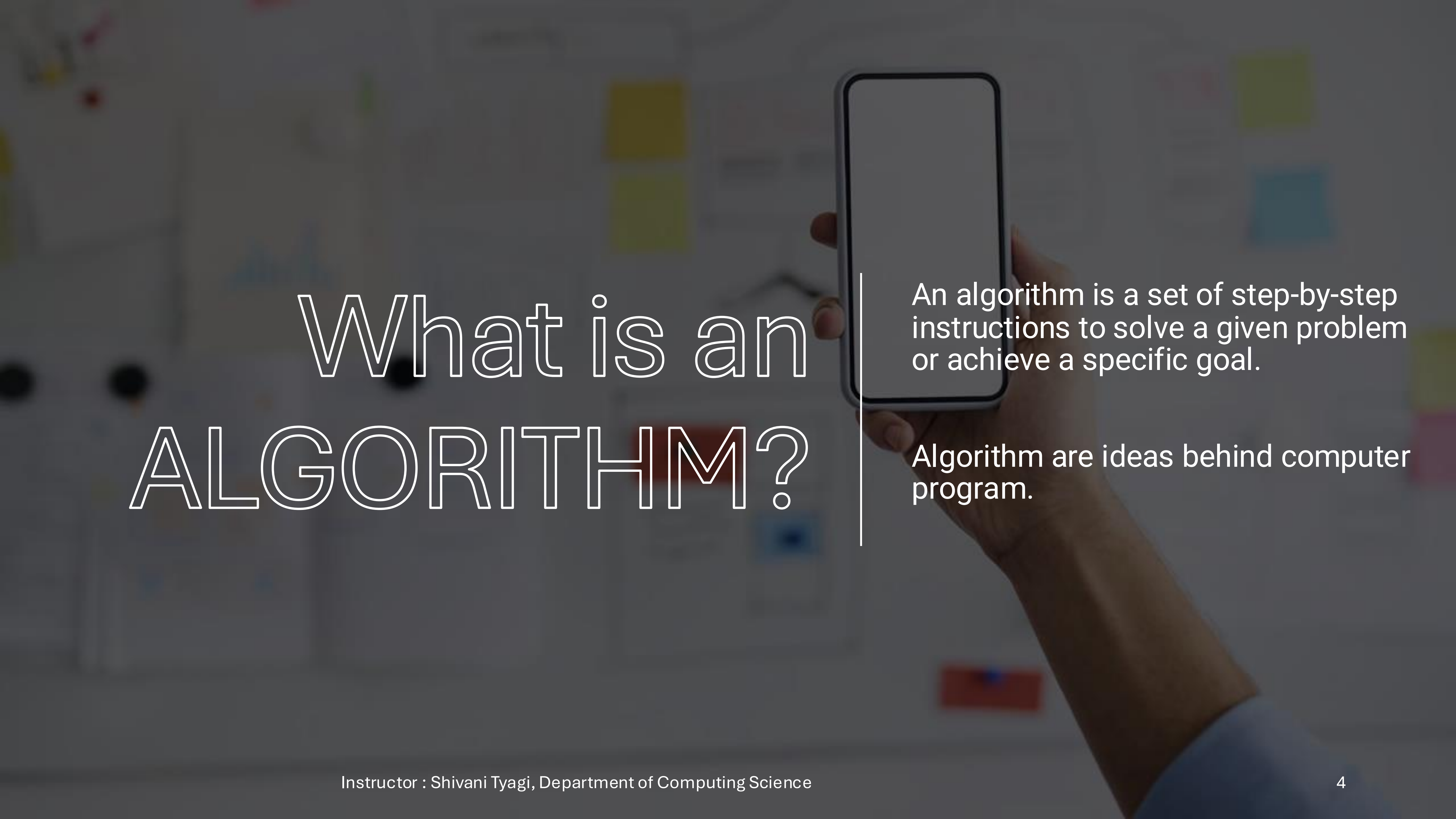
**Data Structures** is about how data can be stored in different structures.

**Algorithms** is about how to solve different problems, often by searching through and manipulating data structures.

Learning about **Data Structures and Algorithms (DSA)** will help us to use large amounts of data to solve problems efficiently.

# Learning Objectives

1. Who do we decide the problem is an "ALGORITHM"?
2. Properties of Algorithm
3. Mapping Examples to Properties
4. Why do we need to ANALYZE an ALGORITHM?
5. What to ANALYZE?
6. Algorithm Efficiency , Problem Size & Growth Function
7. Asymptotic Notations/Complexity
8. Big-Oh Notation & How to calculate it?



## What is an ALGORITHM?

An algorithm is a set of step-by-step instructions to solve a given problem or achieve a specific goal.

Algorithm are ideas behind computer program.

# Examples of an ALGORITHM

1. **Directions to somebody's house**
   **(Step-by-step instructions with a clear start and end)**
2. **A recipe for cooking a cake**
   **(Ordered steps, inputs = ingredients, output = cake)**
3. **Morning routine**
   **Wake up → brush teeth → shower → get dressed → leave**
4. **ATM cash withdrawal**
   **Insert card → enter PIN → choose amount → collect cash**
5. **Traffic light system**
   **Green → Yellow → Red → repeat (with timing rules)**
6. **Logging into an email account**
   **Enter username → enter password → click login**

# Examples of an ALGORITHM

| | |
|---|---|
| Sorting | Sorting books on a shelf : Compare titles → arrange alphabetically |
| Making | Making a phone call : Unlock phone → dial number → press call |
| Selecting | Online food ordering : Select restaurant → choose items → add to cart → pay |
| Washing | Washing clothes in a washing machine : Load clothes → add detergent → select mode → start |
| Crossing | Crossing a road safely : Look left → look right → look left again → cross |
| Evaluating | Exam paper evaluation : Read answer → compare with solution → assign marks |
| Parking | Parking a car : Check space → reverse slowly → adjust → stop |

## What is the deciding criteria to check if it is an ALGORITHM?

# Properties of ALGORITHMS

- INPUT – what the algorithm takes in as input.
- OUTPUT – what is the goal.
- DEFINITENESS – the steps are defined.
- CORRECTNESS - should produce the correct output.
- FINITENESS - the steps required should be finite.
- EFFECTIVENESS - each step must be able to be performed in a finite amount of time.

## Mapping Examples to Properties

| Example | Input | Output | Definiteness | Correctness | Finiteness | Effectiveness |
|---|---|---|---|---|---|---|
| Directions to somebody's house | Starting location, destination | Arrival at the house | Step-by-step route clearly defined | Following steps gets you to the house | Finite number of steps | Each step (turn, go straight) can be performed |
| Recipe for cooking a cake | Ingredients, utensils | Cooked cake | Steps for mixing, baking, cooling are clearly defined | Following steps produces cake | Finite steps (mix → bake → cool) | Each step can be performed in real time |
| Morning routine | Wake-up time | Ready to leave for day | Steps like brush teeth, shower, get dressed | Following steps prepares a person | Finite steps | Each step executable in finite time |
| ATM cash withdrawal | Bank card, PIN, withdrawal amount | Cash dispensed | Steps: insert card → enter PIN → choose amount → collect cash | Correct cash is given | Finite steps | Each step can be done in real time |
| Traffic light system | Current light status | Next light in cycle | Green → Yellow → Red → repeat | Changes follow correct sequence | Steps repeat in cycles, but each transition is finite | Each light change happens in finite, timed duration |

# Mapping Examples to Properties

| Example | Why It’s Not a Proper Algorithm | Which Component Fails |
|---|---|---|
| **“Do whatever you feel like today”** | No clear steps; outcome unpredictable | Definiteness, Correctness |
| **“Keep searching for a lost pen forever”** | | |
| **“Think of a number and guess it”** | | |
| **“Write an essay whenever inspiration strikes”** | | |
| **“Make the best cake instantly without a recipe”** | | |
| **“Solve world hunger in one step”** | | |

## Mapping Examples to Properties

| Example | Why It's Not a Proper Algorithm | Which Component Fails |
|---|---|---|
| **"Do whatever you feel like today"** | No clear steps; outcome unpredictable | Definiteness, Correctness |
| **"Keep searching for a lost pen forever"** | No finite number of steps; might never end | Finiteness |
| **"Think of a number and guess it"** | | |
| **"Write an essay whenever inspiration strikes"** | | |
| **"Make the best cake instantly without a recipe"** | | |
| **"Solve world hunger in one step"** | | |

# Mapping Examples to Properties

| Example | Why It's Not a Proper Algorithm | Which Component Fails |
|---|---|---|
| **"Do whatever you feel like today"** | No clear steps; outcome unpredictable | Definiteness, Correctness |
| **"Keep searching for a lost pen forever"** | No finite number of steps; might never end | Finiteness |
| **"Think of a number and guess it"** | Steps not well-defined, effectiveness impossible | Definiteness, Effectiveness |
| **"Write an essay whenever inspiration strikes"** | | |
| **"Make the best cake instantly without a recipe"** | | |
| **"Solve world hunger in one step"** | | |

## Mapping Examples to Properties

| Example | Why It's Not a Proper Algorithm | Which Component Fails |
|---|---|---|
| **"Do whatever you feel like today"** | No clear steps; outcome unpredictable | Definiteness, Correctness |
| **"Keep searching for a lost pen forever"** | No finite number of steps; might never end | Finiteness |
| **"Think of a number and guess it"** | Steps not well-defined, effectiveness impossible | Definiteness, Effectiveness |
| **"Write an essay whenever inspiration strikes"** | No input/output defined; steps unclear | Input, Output, Definiteness |
| **"Make the best cake instantly without a recipe"** | | |
| **"Solve world hunger in one step"** | | |

## Mapping Examples to Properties

| Example | Why It's Not a Proper Algorithm | Which Component Fails |
|---|---|---|
| **"Do whatever you feel like today"** | No clear steps; outcome unpredictable | Definiteness, Correctness |
| **"Keep searching for a lost pen forever"** | No finite number of steps; might never end | Finiteness |
| **"Think of a number and guess it"** | Steps not well-defined, effectiveness impossible | Definiteness, Effectiveness |
| **"Write an essay whenever inspiration strikes"** | No input/output defined; steps unclear | Input, Output, Definiteness |
| **"Make the best cake instantly without a recipe"** | Steps impossible to perform; vague | Effectiveness, Definiteness |
| **"Solve world hunger in one step"** | | |

## Mapping Examples to Properties

| Example | Why It’s Not a Proper Algorithm | Which Component Fails |
|---|---|---|
| **“Do whatever you feel like today”** | No clear steps; outcome unpredictable | Definiteness, Correctness |
| **“Keep searching for a lost pen forever”** | No finite number of steps; might never end | Finiteness |
| **“Think of a number and guess it”** | Steps not well-defined, effectiveness impossible | Definiteness, Effectiveness |
| **“Write an essay whenever inspiration strikes”** | No input/output defined; steps unclear | Input, Output, Definiteness |
| **“Make the best cake instantly without a recipe”** | Steps impossible to perform; vague | Effectiveness, Definiteness |
| **“Solve world hunger in one step”** | Not executable in finite time; outcome unclear | Effectiveness, Finiteness, Correctness |

## Hands-on Activity 1

Look at each task. Decide if it meets all the algorithm components. Discuss in pairs why it is or isn't an algorithm.

GROUP 1 : Playing a song on a music app

GROUP 2 : Waiting for inspiration to start a painting

GROUP 3 : Washing Hands

GROUP 4 : Deciding to wear something

GROUP 5 : Following a GPS to reach a destination

GROUP 6 : Guessing lottery numbers

# Possible Solutions

| Group | Task | Algorithm? | Explanation |
|---|---|---|---|
| Group 1 | Playing a song on a music app | ✅ Yes | Steps are finite, definite, and executable: open app → select song → play. Input = song choice; output = song plays. |
| Group 2 | Waiting for inspiration to start a painting | | |
| Group 3 | Washing hands | | |
| Group 4 | Deciding to wear something | | |
| Group 5 | Following a GPS to reach a destination | | |
| Group 6 | Guessing lottery numbers | | |

# Possible Solutions

| Group | Task | Algorithm? | Explanation |
|---|---|---|---|
| Group 1 | Playing a song on a music app | ✅ Yes | Steps are finite, definite, and executable: open app → select song → play. Input = song choice; output = song plays. |
| Group 2 | Waiting for inspiration to start a painting | ❌ No | Steps are not definite or finite; output is unpredictable; cannot guarantee effectiveness. |
| Group 3 | Washing hands | | |
| Group 4 | Deciding to wear something | | |
| Group 5 | Following a GPS to reach a destination | | |
| Group 6 | Guessing lottery numbers | | |

# Possible Solutions

| Group | Task | Algorithm? | Explanation |
|---|---|---|---|
| Group 1 | Playing a song on a music app | ✅ Yes | Steps are finite, definite, and executable: open app → select song → play. Input = song choice; output = song plays. |
| Group 2 | Waiting for inspiration to start a painting | ❌ No | Steps are not definite or finite; output is unpredictable; cannot guarantee effectiveness. |
| Group 3 | Washing hands | ✅ Yes | Steps are clear and finite: wet → soap → scrub → rinse → dry. Input = hands, Output = clean hands. |
| Group 4 | Deciding to wear something | | |
| Group 5 | Following a GPS to reach a destination | | |
| Group 6 | Guessing lottery numbers | | |

# Possible Solutions

| Group | Task | Algorithm? | Explanation |
|---|---|---|---|
| Group 1 | Playing a song on a music app | ✅ Yes | Steps are finite, definite, and executable: open app → select song → play. Input = song choice; output = song plays. |
| Group 2 | Waiting for inspiration to start a painting | ❌ No | Steps are not definite or finite; output is unpredictable; cannot guarantee effectiveness. |
| Group 3 | Washing hands | ✅ Yes | Steps are clear and finite: wet → soap → scrub → rinse → dry. Input = hands, Output = clean hands. |
| Group 4 | Deciding to wear something | ❌ No | Choice is vague, steps not defined; no clear algorithm; output depends on mood, not systematic procedure. |
| Group 5 | Following a GPS to reach a destination | | |
| Group 6 | Guessing lottery numbers | | |

# Possible Solutions

| Group | Task | Algorithm? | Explanation |
|---|---|---|---|
| Group 1 | Playing a song on a music app | ✅ Yes | Steps are finite, definite, and executable: open app → select song → play. Input = song choice; output = song plays. |
| Group 2 | Waiting for inspiration to start a painting | ❌ No | Steps are not definite or finite; output is unpredictable; cannot guarantee effectiveness. |
| Group 3 | Washing hands | ✅ Yes | Steps are clear and finite: wet → soap → scrub → rinse → dry. Input = hands, Output = clean hands. |
| Group 4 | Deciding to wear something | ❌ No | Choice is vague, steps not defined; no clear algorithm; output depends on mood, not systematic procedure. |
| Group 5 | Following a GPS to reach a destination | ✅ Yes | Steps are definite, finite, and effective: start → follow route → reach destination. Input = starting location + destination; output = arrival. |
| Group 6 | Guessing lottery numbers | | |

# Possible Solutions

| Group | Task | Algorithm? | Explanation |
|---|---|---|---|
| Group 1 | Playing a song on a music app | ✅ Yes | Steps are finite, definite, and executable: open app → select song → play. Input = song choice; output = song plays. |
| Group 2 | Waiting for inspiration to start a painting | ❌ No | Steps are not definite or finite; output is unpredictable; cannot guarantee effectiveness. |
| Group 3 | Washing hands | ✅ Yes | Steps are clear and finite: wet → soap → scrub → rinse → dry. Input = hands, Output = clean hands. |
| Group 4 | Deciding to wear something | ❌ No | Choice is vague, steps not defined; no clear algorithm; output depends on mood, not systematic procedure. |
| Group 5 | Following a GPS to reach a destination | ✅ Yes | Steps are definite, finite, and effective: start → follow route → reach destination. Input = starting location + destination; output = arrival. |
| Group 6 | Guessing lottery numbers | ❌ No | Steps are not definite, not correct, not effective; output cannot be guaranteed; no finite procedure ensures success. |

If a task has a defined **input**, produces a correct **output**, has **finite, unambiguous steps**, and is **effective**, then it can be implemented as a computer program

## Why we need ALGORITHM ANALYSIS?

- Writing a working program is not good enough.
- The program may be inefficient!
- If the program is executed on BIG Data / Large Datasets, then the running time becomes an issue.

## Why we need ALGORITHM ANALYSIS?

Algorithm analysis is the process of evaluating an algorithm in terms of efficiency, correctness, and resource usage before implementing it.

**BETTER** algorithm save **TIME, SPACE & COST**, making programs **FASTER** and more **SCALABLE**.

## What to ANALYZE?

CORRECTNESS – does the input/output relation matches the algorithm requirement

AMOUNT OF WORK DONE – basic operations to do task finite amount of time

AMOUNT OF SPACE USED – memory used

SIMPLICITY & CLARITY – verification & implementation

OPTIMALITY – is it impossible to do better?

## What to ANALYZE?

**TIME COMPLEXITY :** Amount of computer time it needs to execute the program to get the intended result.

**SPACE COMPLEXITY :** Memory requirements based on the problem size.

## ALGORITHM EFFICIENCY

**Definition:** The **efficiency of an algorithm** measures how effectively it uses **resources**, usually **CPU time**.

**Algorithm analysis** helps categorize algorithms based on efficiency.

## ALGORITHM EFFICIENCY

**Everyday Example: Washing Dishes**

Washing a dish → 30 seconds

Drying a dish → 30 seconds

**Total per dish** = 60 seconds = 1 minute

**n dishes → n minutes**

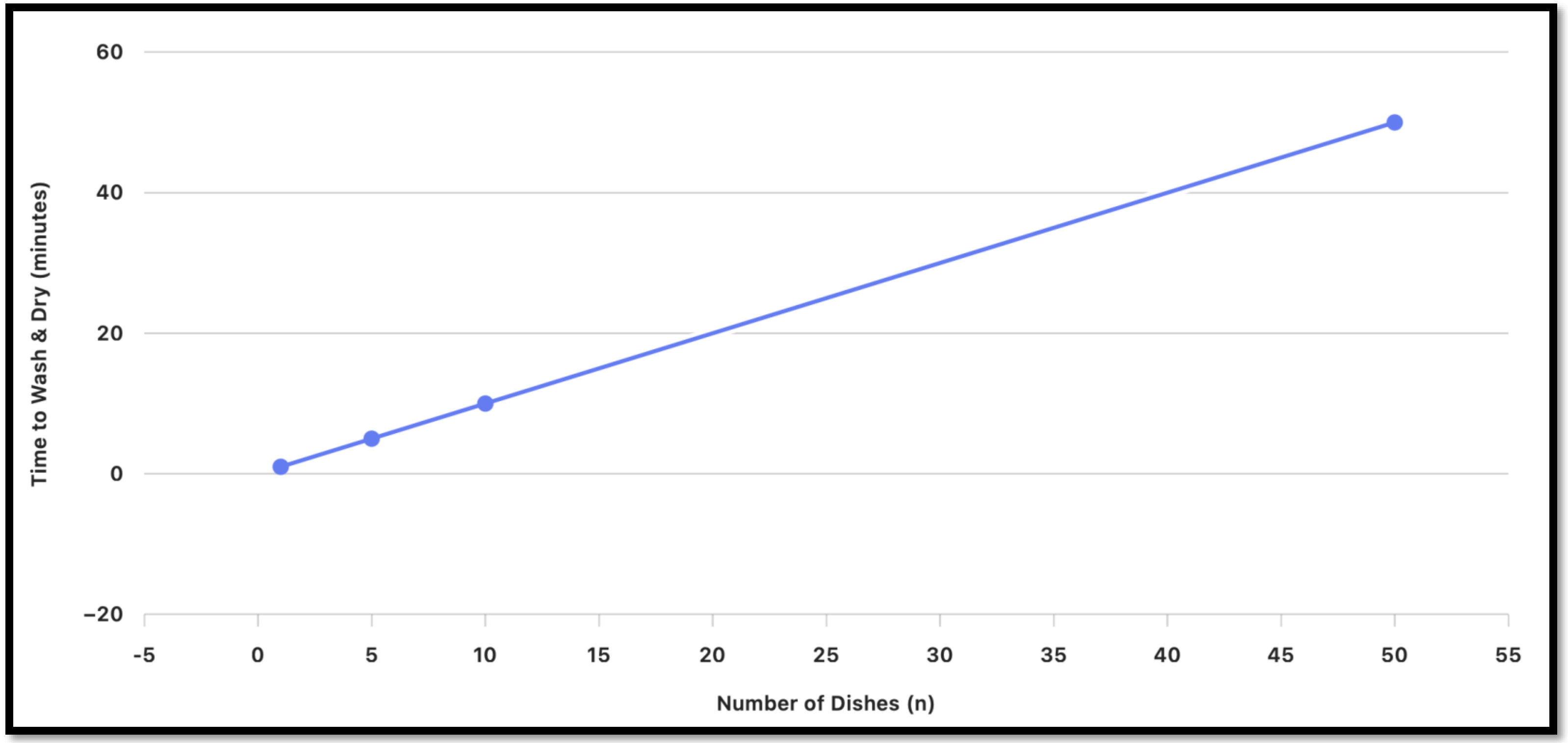
# ALGORITHM EFFICIENCY

"EFFICIENCY TELLS US HOW RESOURCES SCALE WITH INPUT SIZE"

| Dishes (n) | Time to wash & dry (minutes) |
|---|---|
| 1 | 1 |
| 5 | 5 |
| 10 | 10 |
| 50 | 50 |

## ALGORITHM EFFICIENCY

Efficiency grows **linearly** with input in this example → **O(n)** time complexity.



T(n)=k·n

Where:

- $T(n)$ = Time to wash & dry
- $n$ = Number of dishes
- $k$ = Constant time per dish (in this example, $k = 1$ minute)

# Hands-On Activity 2

TASK : Finding Your Favorite Snack

**Scenario:**

- There’s a small snack cupboard with 10 different snacks.
- You want to find your favorite snack.

## Checking one by one

There are **10 snacks**. It takes **15 seconds** to check each snack. If you find your favorite snack on the **k-th try**, the time taken is?

## Checking one by one

There are **10 snacks**. It takes **15 seconds** to check each snack. If you find your favorite snack on the **k-th try**, the time taken is?

| Position (k) | Time (seconds) | Time (minutes) |
|---|---|---|
| 1 | 15 | 0.25 |

$T(k) = 15 * k$ seconds

Or

$T(k) = (15 * k)/60$ minutes

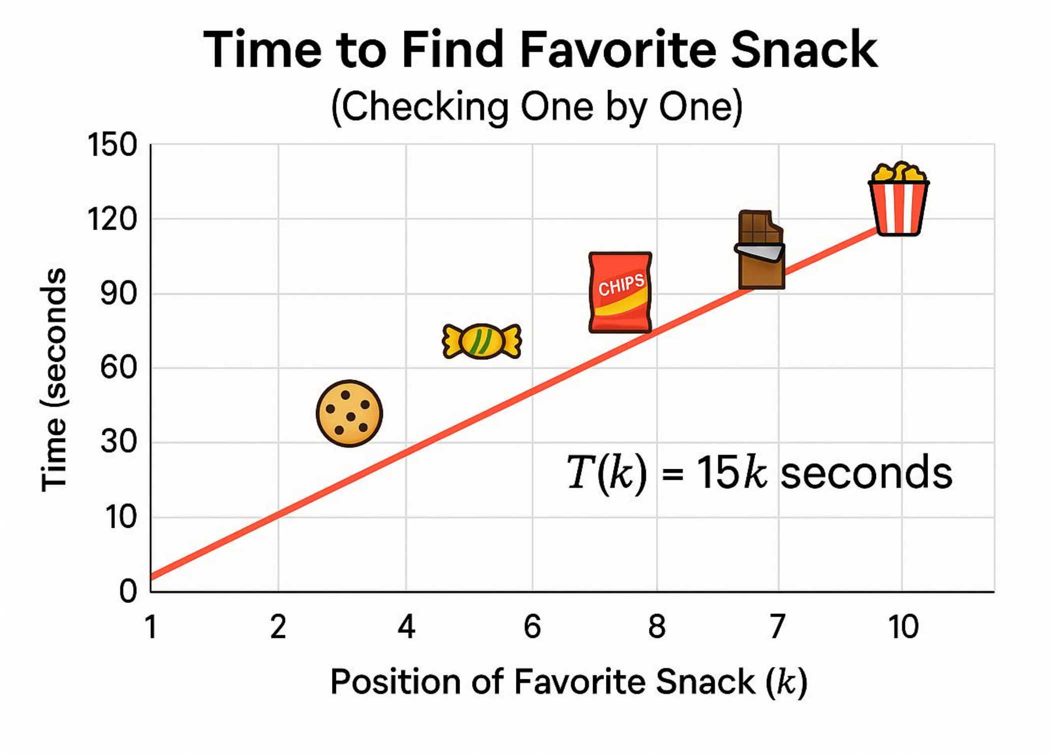
## Checking one by one

There are **10 snacks**. It takes **15 seconds** to check each snack. If you find your favorite snack on the **k-th try**, the time taken is?

| Position (k) | Time (seconds) | Time (minutes) |
|---|---|---|
| 1 | 15 | 0.25 |
| 2 | 30 | 0.5 |
| 3 | 45 | 0.75 |
| 4 | 60 | 1 |
| 5 | 75 | 1.25 |
| 6 | 90 | 1.5 |
| 7 | 105 | 1.75 |
| 8 | 120 | 2 |
| 9 | 135 | 2.25 |
| 10 | 150 | 2.5 |

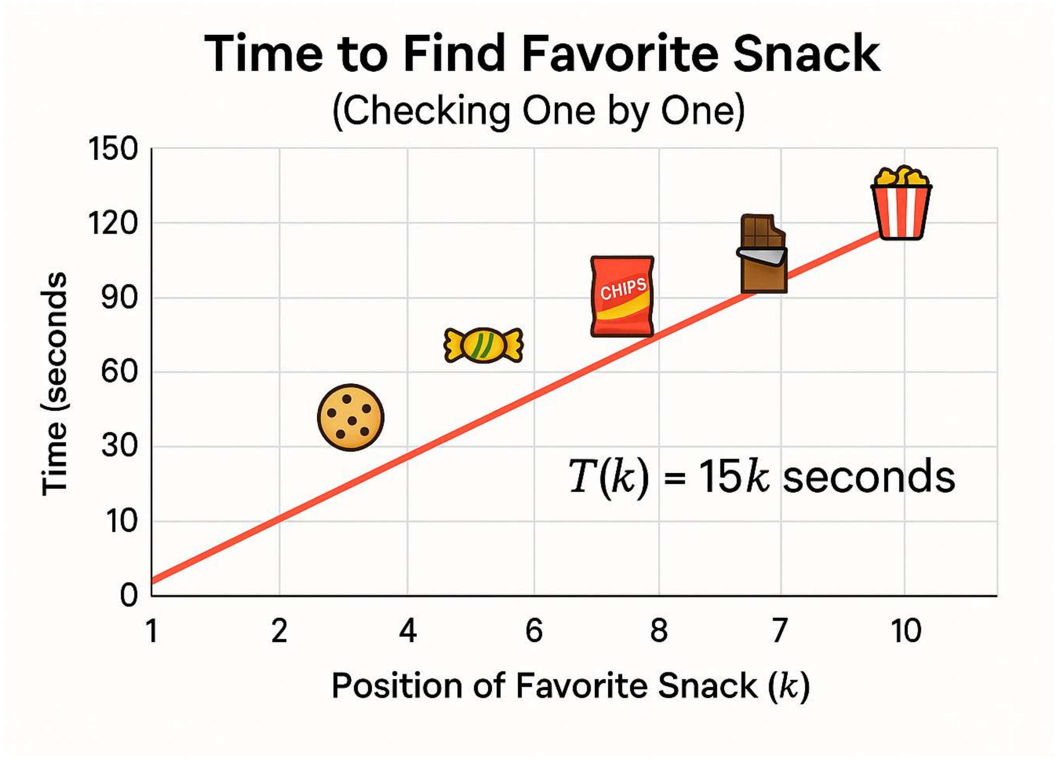
# ALGORITHM EFFICIENCY

Now what would be the WORST-CASE and the BEST-CASE scenarios ?



# ALGORITHM EFFICIENCY

Now what would be the WORST-CASE  and the BEST-CASE scenarios ?



You find your favorite snack **on the first try**.

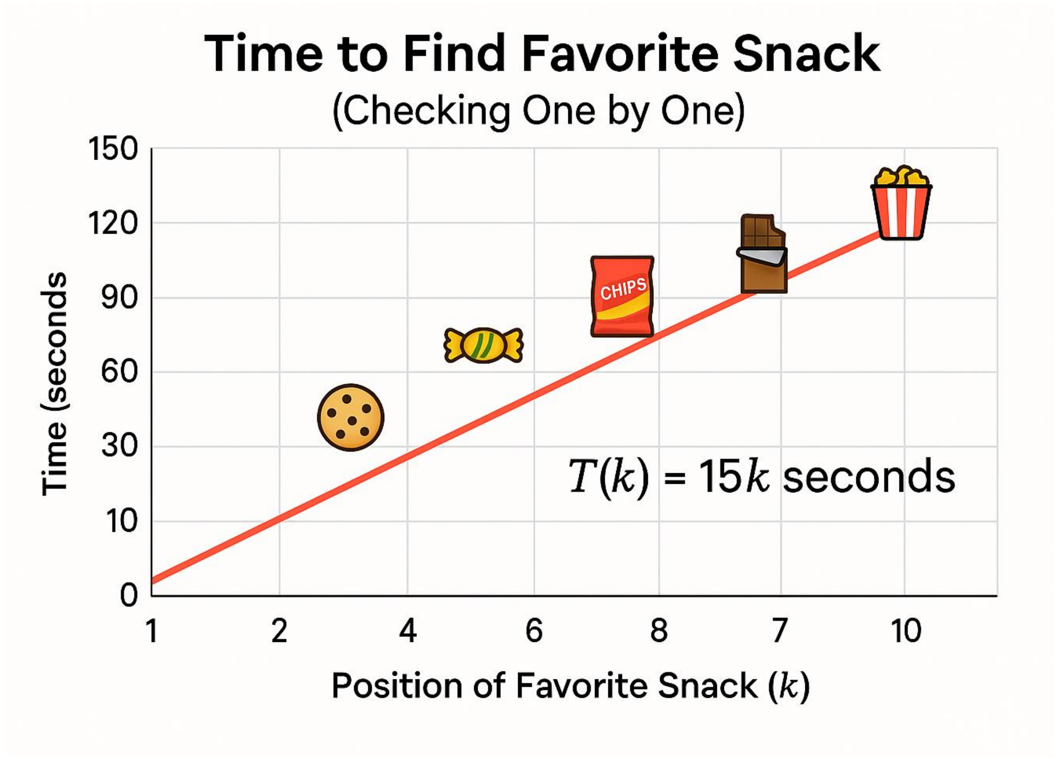
T(1) = 15 seconds / 0.25 mins

This is the **minimum time** possible.

In algorithm terms:
Best case = O(1) (constant time) because you only check one snack.

# ALGORITHM EFFICIENCY

Now what would be the WORST-CASE and the BEST-CASE scenarios ?



Your favorite snack is **last in the cupboard (10th position).**

T(10)=15×10=150 seconds / 2.5 mins

This is the **maximum time** possible.

In algorithm terms:
Worst case = O(n) because you check all snacks.

## PROBLEM SIZE

For every algorithm we want to analyze, we need to define the size of the problem

The dishwashing problem has a size *n* – number of dishes to be washed & dried

The finding snacks problem has a size *n* – number of snacks in the cupboard.

## PROBLEM SIZE

For a search algorithm, the size of the problem is the size of the search pool

For a sorting algorithm, the size of the program is the number of elements to be sorted

## PROBLEM SIZE

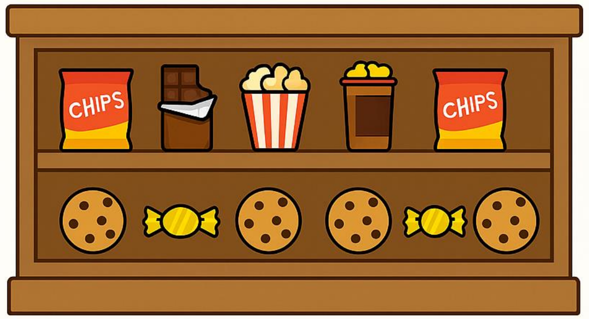
For every algorithm we analyze, we define the **size of the problem**:

**Dishwashing Problem:**
**Size = n** → number of dishes to be washed/dried.

**Search Problem:**
**Size = n** → number of items in the search pool.

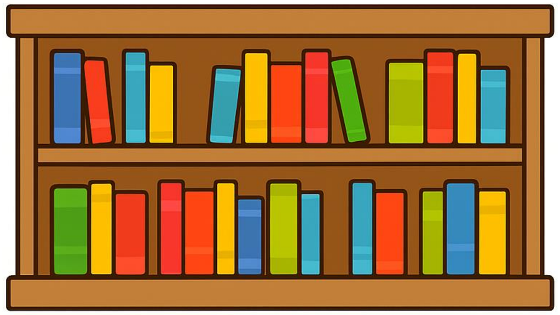
**Sorting Problem:**
**Size = n** → number of elements to be sorted.

What is the search size if you looking for your favorite snack?



10 snacks

What is the search size if you looking for your favorite book?



20 books

## GROWTH FUNCTIONS

We must also decide what we are trying to efficiently optimize

*time complexity* – CPU time

*space complexity* – memory space

CPU time is generally the focus

A growth function shows the relationship between the size of the problem (n) and the value optimized (time)

# ASYMPTOTIC NOTATION

Asymptotic notations are the terminology that enables meaningful statements about **time & space complexity**.

**To calculate running time of an ALGORITHM**

## ASYMPTOTIC NOTATION

The time required by the given algorithm falls under **THREE** types:

1. **Best-case time or the minimum time** required in executing a program.
2. **Average-case time or the average time** required in executing a program.
3. **Worst-case time or the maximum time** required in executing program

## ASYMPTOTIC NOTATION

**Best Case: Denoted by Ω (Omega)**
→ **Represents the minimum time** an algorithm will take.

**Average Case: Denoted by Θ (Theta)**
→ **Represents the average time** for an algorithm.

**Worst Case: Denoted by O (Big O)**
→ **Represents the maximum time** an algorithm will take.

## Big-Oh Notation

In computing science, Big-Oh Notation denoted by O is used to classify algorithm according to how their run time and space requirement grows as the input size grows.

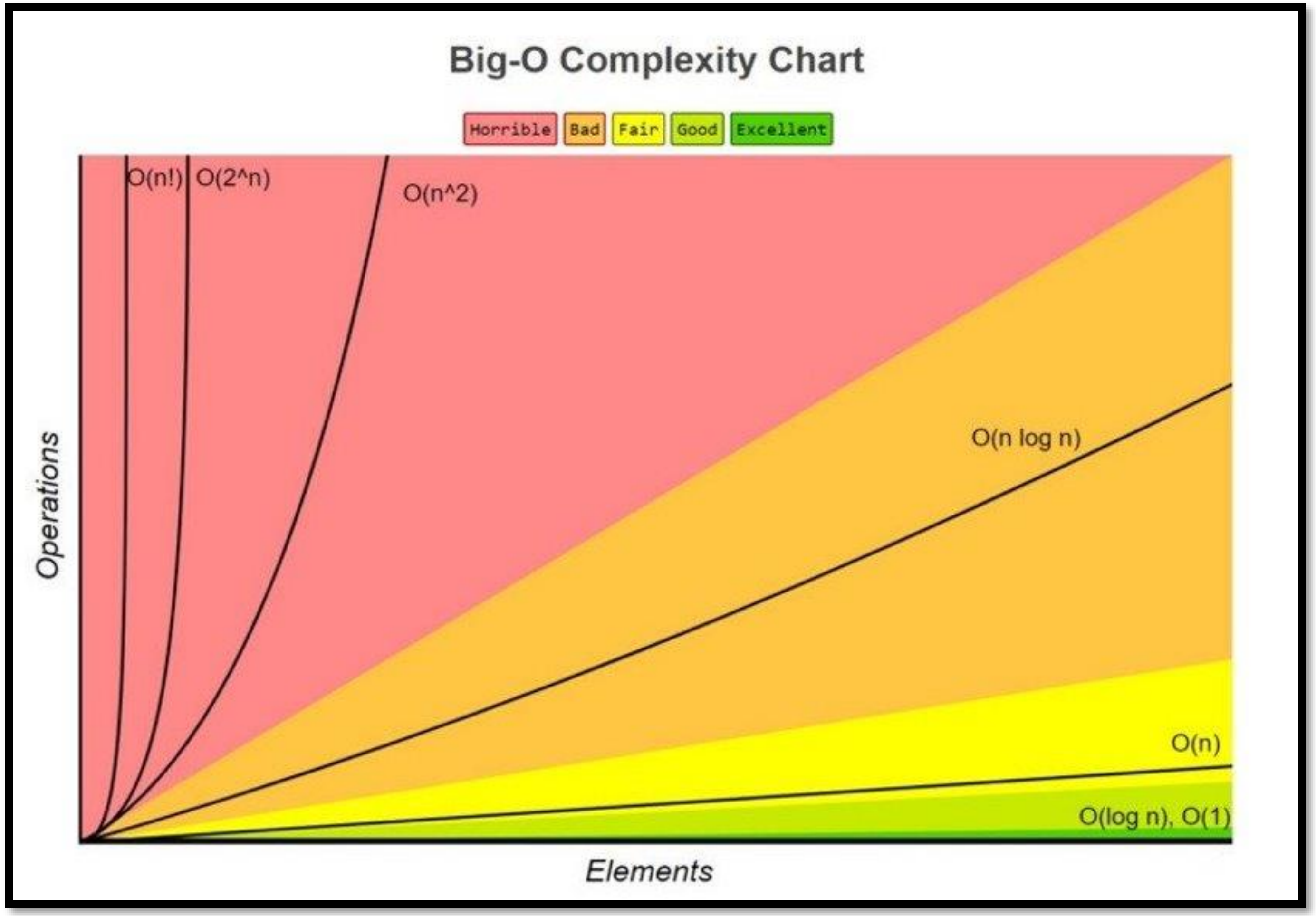
Big-Oh Notation is really concerned with " WHAT HAPPENS WHEN WE SCALE IN TERMS OF DATA"

## Why is it important for us to understand Big-Oh Notation?

## Big-Oh Notation Definition

- Given two functions **f(n)** and **g(n)**, we say that **f(n)** is **O(g(n))** if there exist constants **c > 0** and **n0 >=** 0 such that **f(n) <= c*g(n)** for all **n >= n0**.
- **f(n)** is **O(g(n))** if **f(n)** grows no faster than **c*g(n)** for all n >= n0, where c and n0 are constants.

F(n) = O(g(n))



## Example 1 : Adding numbers

```
int total = 0;

for (int i = 0; i < n; i++) {
    total += 1;              // same as total = total + 1
    System.out.println(i);
}
```

# How do we calculate Big-Oh Notation?

First, we will get the count of operations an algorithm is going to take.

| Statement/Operation | | Frequency |
|---|---|---|
| total = 0 | | |
| for(i=0,i<n, i++) | | |
| total+=1 | print(i) | |

## How do we calculate Big-Oh Notation?

First, we will get the count of operations an algorithm is going to take.

| Statement/Operation | | Frequency |
|---|---|---|
| total = 0 | | 1 |
| for(i=0,i<n, i++) | | |
| total+=1 | print(i) | |

# How do we calculate Big-Oh Notation?

First, we will get the count of operations an algorithm is going to take.

| Statement/Operation | | Frequency |
|---|---|---|
| total = 0 | | 1 |
| for(i=0,i<n, i++) | | n |
| total+=1 | print(i) | 1+1=2 |

# How do we calculate Big-Oh Notation?

Second, we put together the steps count into f(n) function

| Statement/Operation | | Frequency |
|---|---|---|
| total = 0 | | 1 |
| for(i=0,i<n, i++) | | n |
| | | 1+1 =2 |
| total+=1 | print(i) | |

**F(n) = O(g(n))**

F(n) = 1+ 2 * n = 2n + 1

# How do we calculate Big-Oh Notation?

Third, we ignore the lower order terms and consider only highest order term.

| Statement/Operation | | Frequency |
|---|---|---|
| total = 0 | | 1 |
| for(i=0,i<n, i++) | | n |
| | | 1+1 =2 |
| total+=1 | print(i) | |

F(n) = 1+ 2 * n = 2n + 1

**2n**

# How do we calculate Big-Oh Notation?

Fourth, we ignore the constant associated with the highest order term.

| Statement/Operation | | Frequency |
|---|---|---|
| total = 0 | | 1 |
| for(i=0,i<n, i++) | | n |
| | | 1+1 =2 |
| total+=1 | print(i) | |

F(n) = 1+ 2 * n = 2n + 1

**n**

# How do we calculate Big-Oh Notation?

The final step, the value remaining after the 4 step is = O(f(n)).

| Statement/Operation | | Frequency |
|---|---|---|
| total = 0 | | 1 |
| for(i=0,i<n, i++) | | n |
| total+=1 | print(i) | 1+1 =2 |

F(n) = 1+ 2 * n = 2n + 1

**Therefore, the Big Oh value O(n)**

## Example 2

***$f(n) = 3n^2 + 2n + 1000Logn + 5000$***

# Example 2

***$f(n) = 3n2 + 2n + 1000Logn + 5000$***

*After ignoring lower order terms, we get the highest order term as $3n^2$*

*After ignoring the constant 3, we get $n^2$*

*Therefore the Big O value of this expression is $O(n^2)$*

## Example 3

***$f(n) = 3n^3 + 2n^2 + 5n + 1$***

## Example 3

***$f(n) = 3n^3 + 2n^2 + 5n + 1$***

*After ignoring lower order terms, we get the highest order term as $3n^3$*

*After ignoring the constant 3, we get $n^3$*

*Therefore the Big O value of this expression is $O(n^3)$*

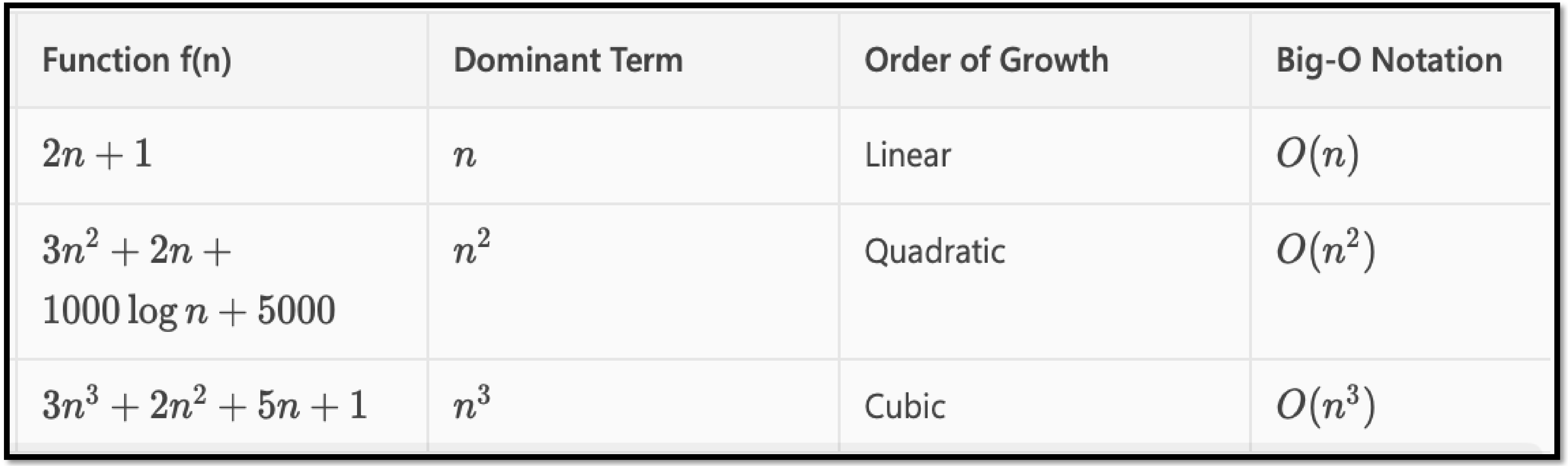
## Dominant Term in Asymptotic Notation

Asymptotic complexity is based on the ***dominant term*** of the growth function – the term that increases most quickly as n increases.

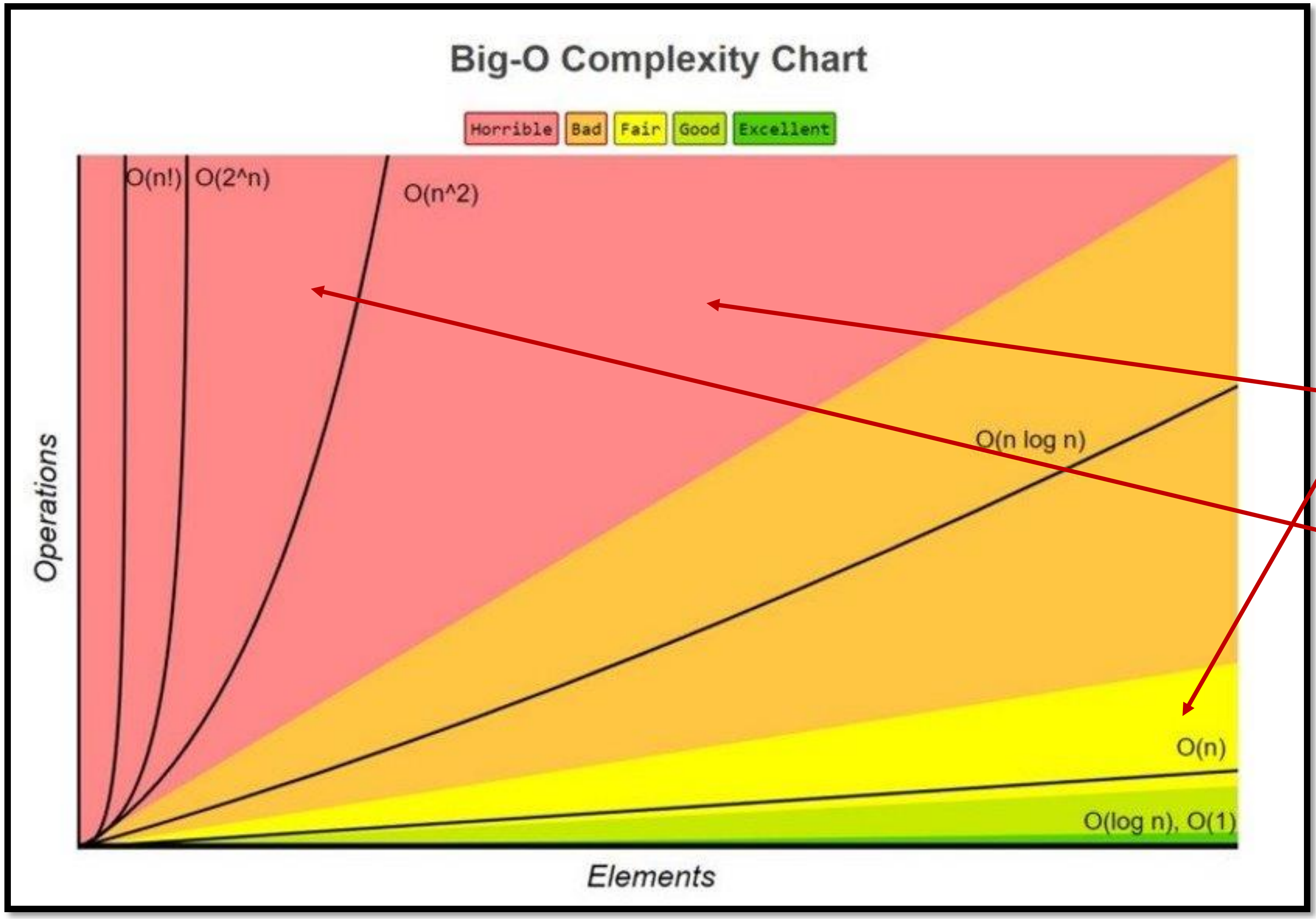
- In Example 1, the highest order term was – n that is our ***Dominant Term***
- In Example 2, the highest order term was – $n^2$ that is our ***Dominant Term***
- In Example 3, the highest order term was – $n^3$ that is our ***Dominant Term***

## Order of Growth in Asymptotic Complexity

Order of growth describes how fast an algorithm's running time increases as input size (n) grows, based on the dominant term of its function.



| Function f(n) | Dominant Term | Order of Growth | Big-O Notation |
|---|---|---|---|
| $2n + 1$ | $n$ | Linear | $O(n)$ |
| $3n^2 + 2n + 1000 \log n + 5000$ | $n^2$ | Quadratic | $O(n^2)$ |
| $3n^3 + 2n^2 + 5n + 1$ | $n^3$ | Cubic | $O(n^3)$ |



**Example 1 : O(n)**

**Example 2 : O($n^2$)**

**Example 3 : O($n^3$)**

## Example : If Else Statement

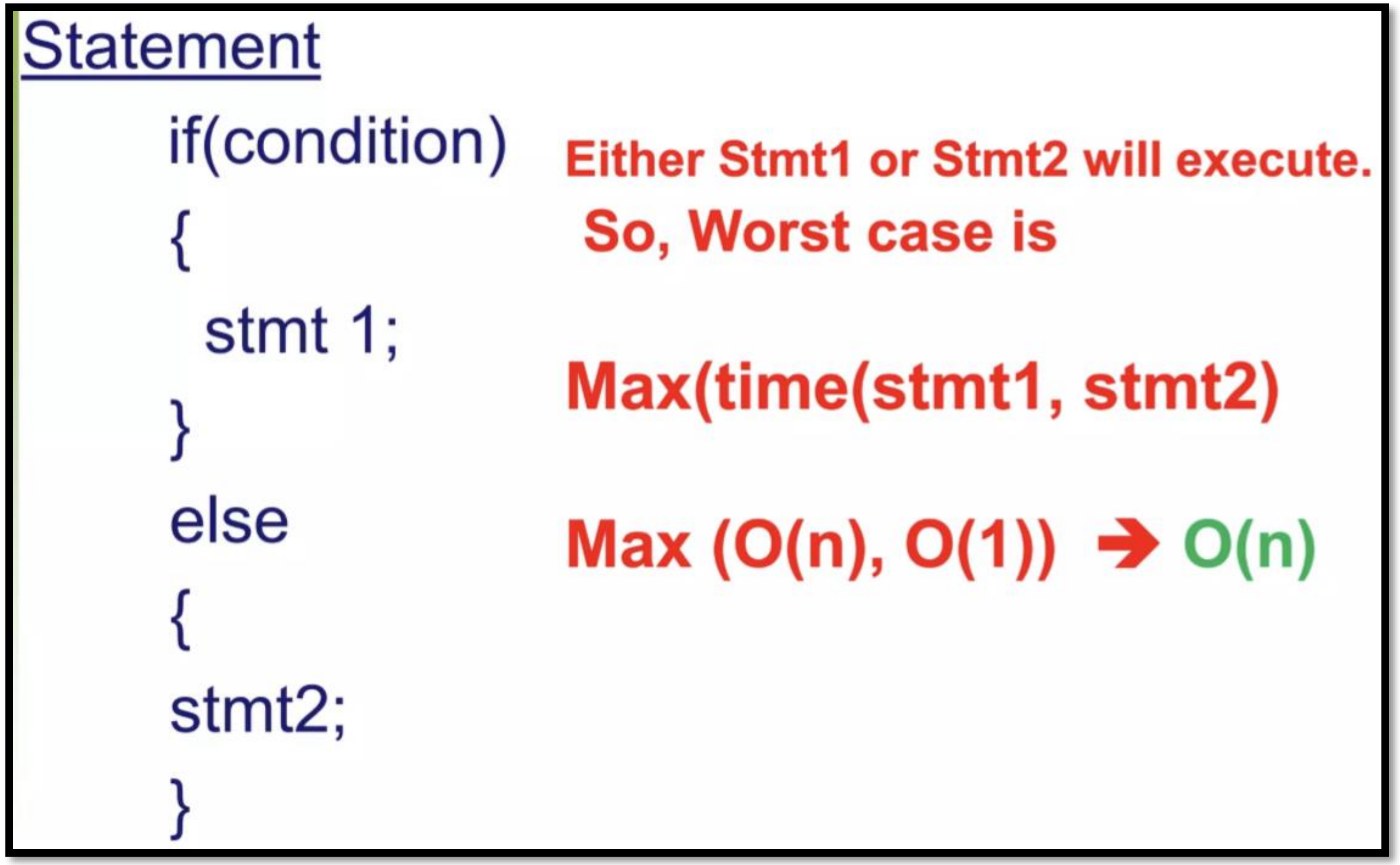
```
if (condition) {
   Statement1;
} else {
   Statement2;
}
```

**Either Stmt 1 or Stmt 2 will execute.**

## Solution

Because only one statement will be executed in if else, we will consider

**Max(time(st1), time(st2)) ~ Max(O(n), 1)**



Statement

```
if(condition)
{
  stmt 1;
}
else
{
stmt2;
}
```

**Either Stmt1 or Stmt2 will execute.**
**So, Worst case is**

**Max(time(stmt1, stmt2)**

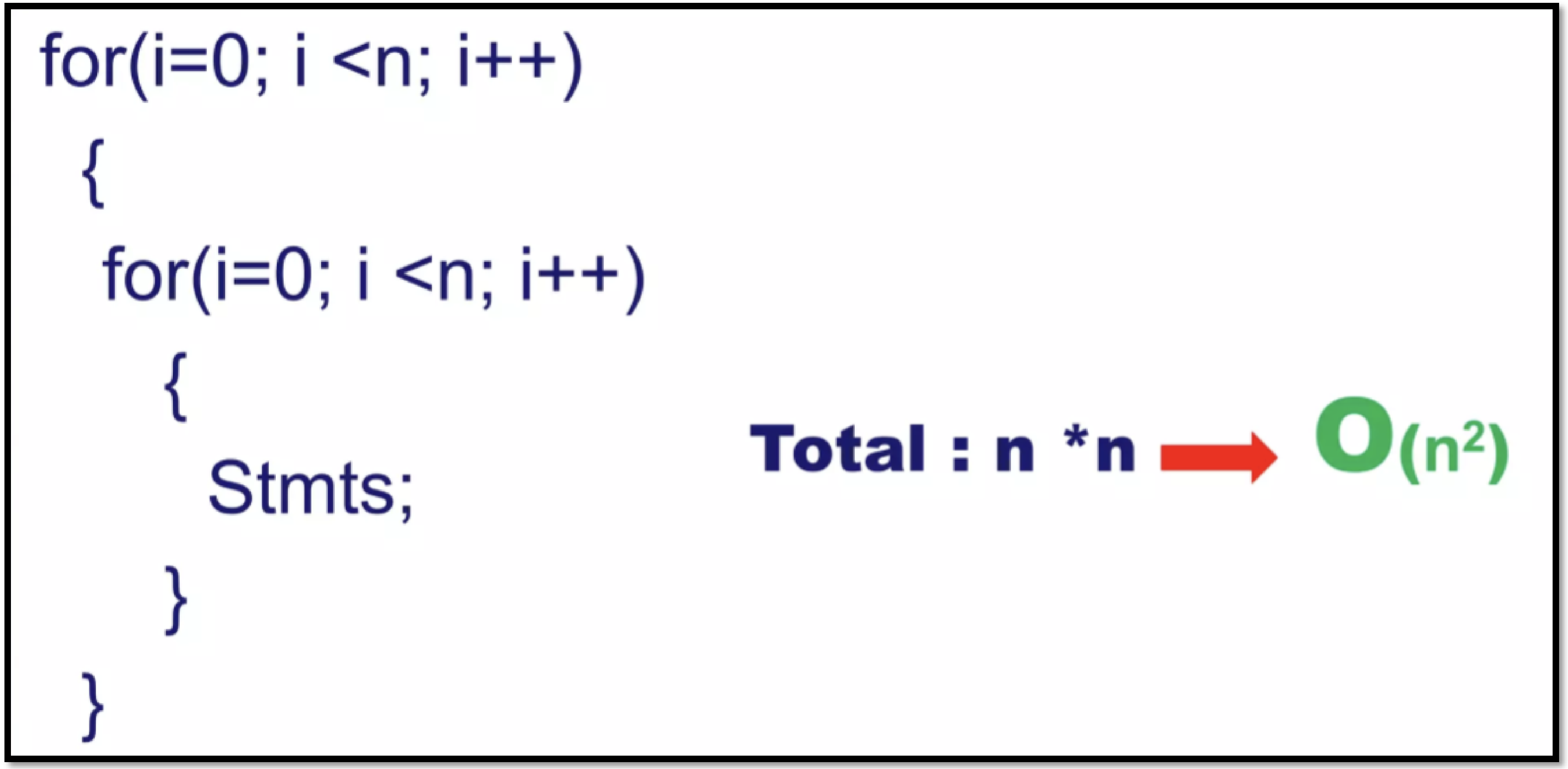
**Max (O(n), O(1)) ➔ O(n)**

## Example : Nested For Statement

```
for (condition) {
   for (condition) {
               Statements;
}
}
```

**Note: For all the "for loop" statement multiply by the product**

## Solution



```
for(i=0; i <n; i++)
 {
  for(i=0; i <n; i++)
    {
      Stmts;
    }
 }
```

**Total : n *n** $\longrightarrow$ $O(n^2)$

# In the Next Class?

| | |
|---|---|
| Implement | Hands-on Lab Work : Implement algorithms discussed in theory (Use lab computers for coding and testing.) |
| Run | Run Programs : Execute sample algorithms with varying input sizes. Observe actual time and space usage. |
| Analyze | Analyze Complexity : Compare empirical results with theoretical Big-O. Plot graphs for performance trends. |
| Share | Discussion & Q&A : Share findings and clarify doubts in next session. |