Lecture 01

Outline

1. Algorithms
    a. What?
    b. Properties
        i. Be correct
        ii. Be unambiguous
        iii. Give the correct solution for all cases
        iv. Be simple
        v. It must terminate
    c. Applications
        i. Data retrieval
        ii. Network Routing
        iii. Sorting
        iv. Searching
        v. Shortest paths in a graph
    d. Pseudocodes
    e. Analysis of the algorithms
        i. Why?
            1. To compare
            2. Predict the growth of the run time
        ii. Idea: To predict resource usage
            1. Memory
            2. Logic Gates
            3. Computational Time
        iii. Scenarios
            1. Best case
            2. Worst case
            3. Average case
        iv. Analysis Methods
            1. Operation count method
                a. For time complexity.
                b. Select one or more operations.
                c. Considers the time spent on chosen operations

            2. Step Count Method (RAM Model)
                a. Assume instructions are executed one after another. (No simultaneous operations)
                b. One step = each single memory access
                c. Running time = Sum of the steps
                d. **Problems with RAM Model Analysis**
                    i. Differ number of steps with different architecture.

ii. It is difficult to count the exact number of steps in the algorithm

3. Exact Analysis
4. Asymptotic Notations

Note: I wrote most of the notes. But on some points, to make things simpler I've asked ChatGPT for help. These ChatGPT parts are included in text boxes and they are great for further reading.

Algorithms

What is an Algorithm?

- An **algorithm** is a **step-by-step procedure,** or **a set of well-defined rules** designed to solve a specific problem. It must be **clear, finite, and effective**.

You missed (A step)

Imagine you're at your **grandma's house**, and she asks you to make her **legendary pancakes**. 🍳🥞

You panic—**you have no idea how to make pancakes!** 😱

Grandma hands you **her secret recipe**:
1 Take **1 cup of flour** 🥣
2 Add **1 egg** 🥚
3 Pour **1/2 cup of milk** 🥛
4 Mix until smooth 🔄
5 Heat a pan and pour batter 🍳
6 Flip when golden brown 🔄
7 Serve with syrup and enjoy! 🍰 😋

🎯 **Congratulations! You just followed an algorithm!**

✅ **Clear steps** (no guessing needed).
✅ **Correct solution for all cases** (as long as you follow the steps).
✅ **Terminates after a finite number of steps** (or you'll just have infinite batter 😆).

🚨 **But what if you skip a step?**

- Forget the flour? **Congratulations, you made an omelet!** 🍳
- Pour the milk first? **Now you have a mess!** 😳
- Never stop flipping? **Your pancake is now charcoal!** 💀 🔥

👉 **Moral of the story:** A **good algorithm** ensures you get the **right result every time**, no matter who follows it!

Note: ChatGPT generated this for me. I don't love it.

Properties

- Algorithms have some properties (or call them standards).
- Without these properties algorithms won't work or won't produce correct results.
- These are the properties of an algorithm.

1. Be correct.
2. Be unambiguous.
3. Give the correct solution for all cases.
4. Be simple.
5. It must terminate.

Be correct

- The algorithm should provide the correct answer.

Be unambiguous

- Every step of the algorithm should be clear.

Give the correct solution for all cases

- Algorithms should always give the correct solution for every possible scenario. Not just a specific one.

Be simple

- Algorithms should be in the simplest form possible, which will boost efficiency instead making a mess.

It Must Terminate

- The algorithm should never run into an infinite loop. It should be terminate after giving the desired output.

1. Correctness ✅ (It must be correct)

💡 **An algorithm should always give the right answer.**

🔹 Imagine you ask **Google Maps** for the shortest route home.
🔹 Instead of guiding you properly, it takes you through **a haunted forest, over a mountain, and across a river with crocodiles.** 🐊💀

🎯 **Moral:** A wrong algorithm is worse than NO algorithm! It **must always give the correct solution.**

---

2. Unambiguity 🎭 (Clear Instructions, No Confusion)

💡 **Every step should be clear—no room for misinterpretation.**

🔹 Imagine a **robot waiter** that follows instructions **literally**:
🧑‍🍳 You say: *"Put butter on toast."*
🤖 Robot: *Puts an entire butter packet on top of the bread.* 🧈🍞
🧑‍🍳 You say: *"Cut the bread in half."*
🤖 Robot: *Cuts the table in half.* 🪵✂️

🎯 **Moral: Each step should be precise!** A good algorithm doesn't leave room for confusion.

---

3. Generality 🌍 (Works for ALL Cases)

💡 **It should work in every possible scenario—not just specific ones.**

🔹 If your **pancake recipe** only works with *organic flour, mountain spring water, and eggs from a golden goose*... guess what? It's useless. 🥞🚫

🔹 A **good algorithm** should handle **all kinds of valid inputs** and always give the correct output.

🎯 **Moral: A good algorithm must be reliable, no matter the situation!**

4. Simplicity 🧠 (Keep It Short & Sweet)

💡 **The shorter & more efficient, the better!**

🔹 Imagine if **unlocking your phone** required 20 steps instead of just entering a passcode. 📱
🔹 Or if **ordering food** required **filling out a 10-page application form**. 😳

🎯 **Moral:** If an algorithm is **too complex**, people will hate it. **Keep it simple and efficient!**

5. Termination ⏳ (It Must End Eventually)

💡 **An algorithm should NOT run forever—it must reach a result.**

🔹 Imagine you're **waiting in line for coffee**, and the barista just keeps **stirring and stirring... forever!** ☕😵
🔹 Or imagine Google **keeps loading the same search results forever**. You'd throw your laptop, right? 😂

🎯 **Moral: If an algorithm doesn't stop, it's broken!**

Applications

- There are many applications of algorithms, but we can identify these main categories.

    1. Data retrieval
    2. Network Routing
    3. Sorting
    4. Searching
    5. Shortest paths in a graph

Pseudocodes

Overview

- A method for writing algorithms in a readable format.
- Similar to programming languages but more expressive.
- Focuses on logic, not software engineering details.

Pseudocode Conventions

- Written in English with indentation.
- Each instruction on a separate line.
- Uses loops and conditional statements.
- **//** for comments, **=** for assignments.
- Arrays use square brackets (A[i]), and ".." represents a range (A[1..i]).

Analysis of Algorithms

- The purpose of analyzing is to predict resource usage and to compare the efficiency of different approaches for a problem.
- What are the resources that an algorithm will use?
    - Memory
    - Logic Gates
    - Computational Time

Best, Average or Worst?

Assume we have an *n* sized input. The number of steps that it takes to get the desired output is known as the case. There are three cases.

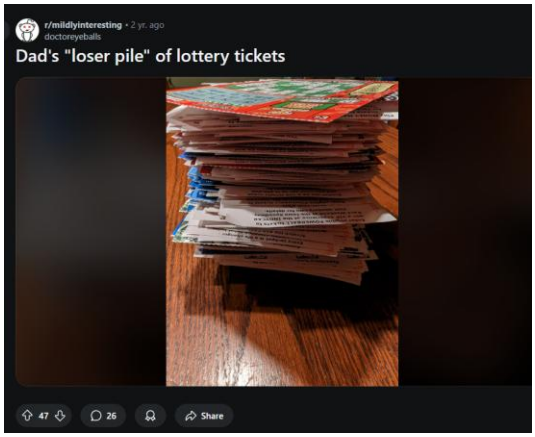
1. Best case
    - In this case we consider the minimum number of steps that are required to get the desired output.
2. Average Case
    - In average case we consider the average number of steps that required to get the desired output.
3. Worst Case
    - In the worst case we consider the maximum number of steps that are required to get the desired output.

We can use **supermarket queue analogy** to explain this.

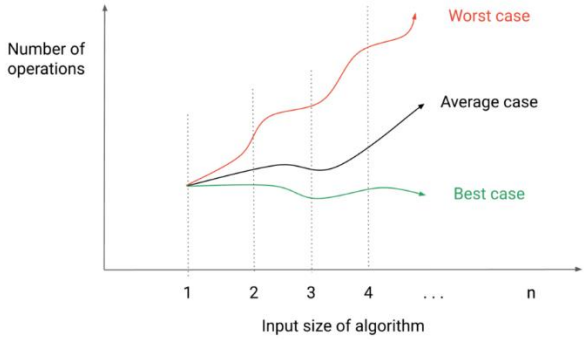
Instant Lottery Analogy for Case

- Imagine you bought 1000 scratchcards (Instant Lotteries). Now you make a card stack from it and start scratching them one by one.
- The seller says, "**One of these cards has a $100,000 jackpot**!"

- In the **best-case** scenario, the first lottery you pick will have the 100000$ prize in it.
- In the **average case** scenario, you might nearly scratch 500 lotteries and find the 100000$ ticket.
- Imagine the seller lied to you and there is no 100000$ ticket in that 1000 lotteries. You'll have to scratch 1000 lotteries in the **worst-case** scenario.
- And to make matters worser your son posted this. 👇



Here is the graph of the time complexity (We'll get into that later).



Analysis Methods

Okay so, now we know that there are those average, best, worst cases. Now we want a way to measure the complexity of an algorithm.

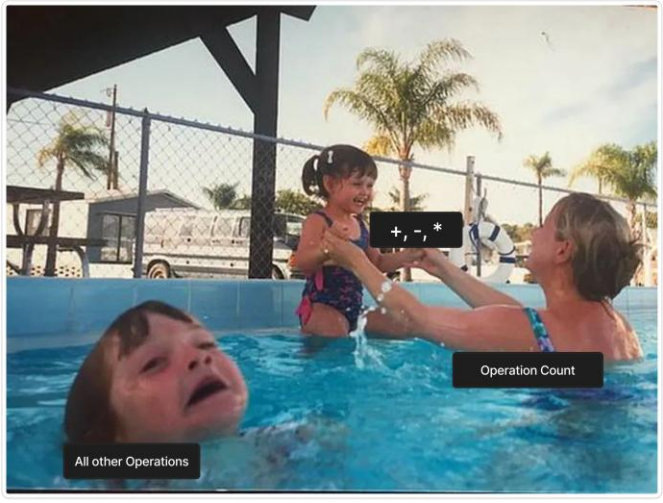
There are few analysis methods to do that. However, in the first lecture we just have to discuss only two of them. Which are,

1. Operation Count
2. Step Count

Operation Count

This analysis method is used to measure the **time complexity**. In operation count method we choose operations to consider. And then we get the time that required to run each operation.

Let's say we choose addition, multiplication and subtraction operations.



Now, consider the following pseudocode.

```
DEFINE a, b and c AS INTEGER
a = 10
b = 5
c = (a * 10)
c = c – 2
c = (c + 2) * 3
DISPLAY c
```

Now your geek friend gave you the time spent on each operation as follows.

1. Addition: 1ms
2. Subtraction: 1ms
3. Multiplication: 2ms
4. Assigning: 3ms
5. Memory access: 2ms

First you must get rid of the last two operations. You don't need that as you threw them away (As your Ex dumped you 💀).

Then, we have to take a look at how many times these operations were used. Lets get back into the code.

```
DEFINE a, b and c AS INTEGER
a = 10
b = 5
c = (a * 10)
c = c – 2
c = (c + 2) * 3
DISPLAY c
```

We've used multiplication, addition and subtraction a few times. Let's go line by line.

- c = (a * 10): Multiplication Once → 2ms
- c = c – 2: Subtraction Once → 1ms
- c = (c + 2) * 3 → ?
    - Here is the twist. In operation count we assume operations in same line executed parallelly (same time). Hence, we have to choose the highest time-consuming operation from the line.
    - Here,
        - Addition takes 1ms
        - Multiplication takes 2ms
    - So, we consider only multiplication operation time, which is 2ms.
- We describe time complexity in $T(n)$ notation. So, in this case time complexity equals,

$$T(n) = 2ms + 1ms + 2ms$$

$$T(n) = 5ms$$

Step Count (RAM Model)

- Now you know operation count method is ~~racist~~ not very efficient. Therefore, we use another analysis called RAM Model analysis.
- In RAM model analysis you don't need your geek friend to tell how much time it takes to execute a single operation.
- However, we make these assumptions.
    - Assume a generic One Processor
    - There are NO parallel executions
    - Operations like +, -, / takes exactly one step.
    - Each memory access takes one step.

$$Running\ time = Sum\ of\ steps$$

Let's dive into an example. Let's take the exact code we used earlier.

```
a = 10
b = 5
c = (a * 10)
c = c – 2
c = (c + 2) * 3
DISPLAY c
```

- a = 10 → One step (Assigning Operation)
- b = 10 → One step (Assigning Operation)
- c = (a * 10) → Three steps
    - c = → (Assigning)
    - a → (Memory Access)
    - (a * 10) → (Multiplication)
- c = c – 2 → Two Steps
    - c = → (Assigning)
    - (c - 2) → (Subtraction)
- c = (c + 2) * 3 → Three Steps
    - c = → (Assigning)
    - (c + 2) → (Addition)
    - * 3 → Multiplication

- DISPLAY c → One Step (Memory Access)

$$T(n) = 1 + 1 + 3 + 2 + 3 + 1$$
$$T(n) = 11$$

However, it's a little bit tricky when it comes to loops. Before moving into advanced questions let's look at a simple for loop.

```
for i = 1 to 3
      sum = sum + i
print i
```

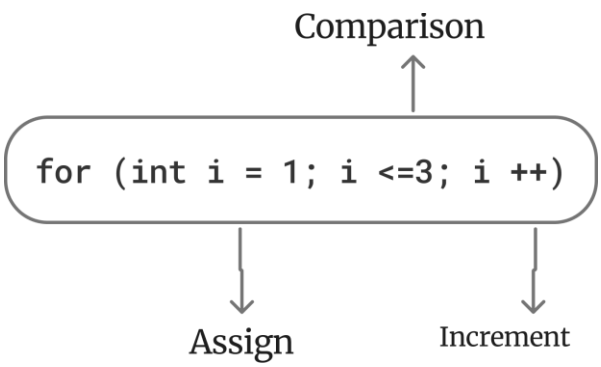
Let's consider this code line by line.

- for i = 1 to 3
  In this single line there are three operations.
  - i = 1
  - i <= 3
  - i = i +1

  And these operations will happen again and again. Let's try to list them.

i = 1		1 step	Assign
1 <= 3		1 step	Comparison
i = 1 + 1	1 + 1	1 step	Increment
	i = 2	1 step	Assign
2 <=3		1 step	Comparison
i = 2 + 1	2 + 1	1 step	Increment
	i = 3	1 step	Assign
3 <= 3		1 step	Comparison
i = 3 + 1	3 + 1	1 step	Increment
	i = 4	1 step	Assign
4 <= 3		1 step	Comparison
Total Steps		11 Steps	

When the loop executes above operations happen. The for loop in java looks like this.



When looking at the table, you can see.

Assignment happened: 4 times

Comparison happened: 4 times

Increment happened: 3 times

The loop runs when i = 1,2,3 (3 times).

Let's take this 3 (How many times loop runs). We can now count steps as,

$$Assign = n + 1$$

$$Comparison = n + 1$$

$$increment = n$$

Hence, $Steps = 2(n + 1) + n$

```
for i = 1 to 3          → 11 Steps
      sum = sum + i     → 2 * 3 Steps
print i                 → 1 step
```

Therefore, the total step count is, $11 + 6 + 1 = 18$.

Now how does this happen inside a while loop?

```
i = 1            → 1 Step
while i <= 10    → 11 steps (1,2,3,4,5,6,7,8,9,10,11)
      print i    → 10 * 1 steps
      i = i + 1  → 10 * 2 steps
```

$$Steps = 1 + 11 + 10 + 20$$

$$Steps = 42$$

Problems with RAM Model

- Different architectures hardware takes different number of steps to perform the same task.
- It's difficult to count the exact number of steps in some algorithm. (Sorting algorithms, Searching algorithms)