

Algorithms

BCS1110

Dr. Ashish Sai



Week 2 Lecture 1



bcs1110.ashish.nl



EPD150 MSM Conference Hall

Algorithms

Part 1: The Beginning

Algorithms

The problem

- The solution
- Math
- History

The problem

- There are times when the world is one way, and you would like it to be another



The problem (Example 1)

- I am hungry and want a brownie
- But there are no brownies in the house
- I can go online and find a recipe
- Follow it step by step
→ now there are brownies 🍫

The problem (Example 2)

- I need to get somewhere
- I go online and get directions
- If I follow them → I reach my destination



The solution

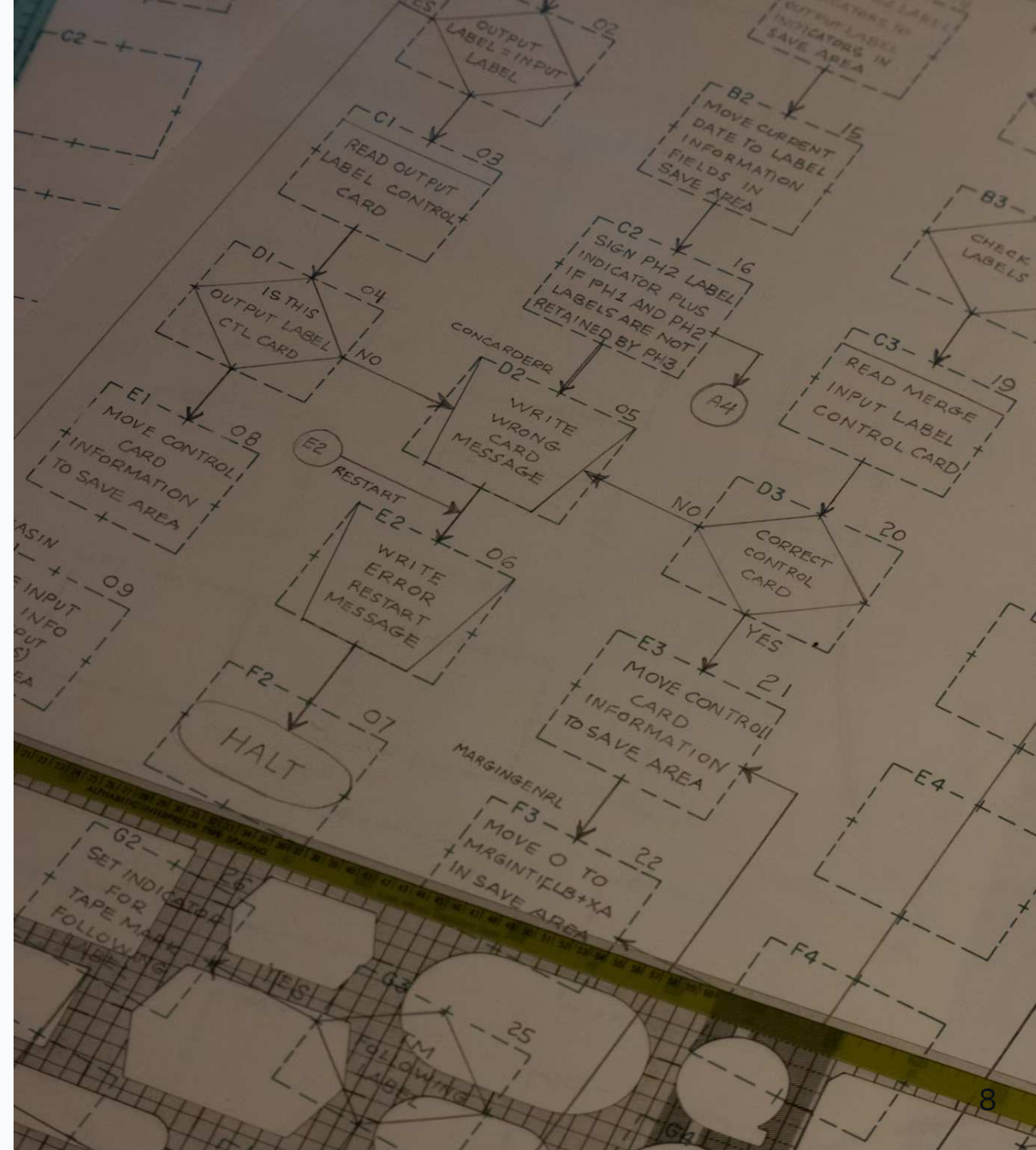
- Recipes & directions are useful because:
 - Easy to follow
 - Minimal knowledge needed
 - Produce the desired result

The solution (definition)

☞ An algorithm is:

"A set of rules that precisely defines a sequence of operations."

- Recipes and directions are algorithms
- Programs are a **subset** of algorithms, not all of them



The solution

Recipes and directions
look different

- Here is a recipe

The solution

- And here are
directions for
traveling from one
place to another

The solution (**similarities**)

Despite differences, they share:

- **Prerequisites** (must be true before starting)
- **Steps** (ordered actions)
- **Done state** (how to know you're finished)

The solution (brownie recipe)

- **Prerequisites**

- Ingredients in correct amounts
- Oven hot enough for long enough

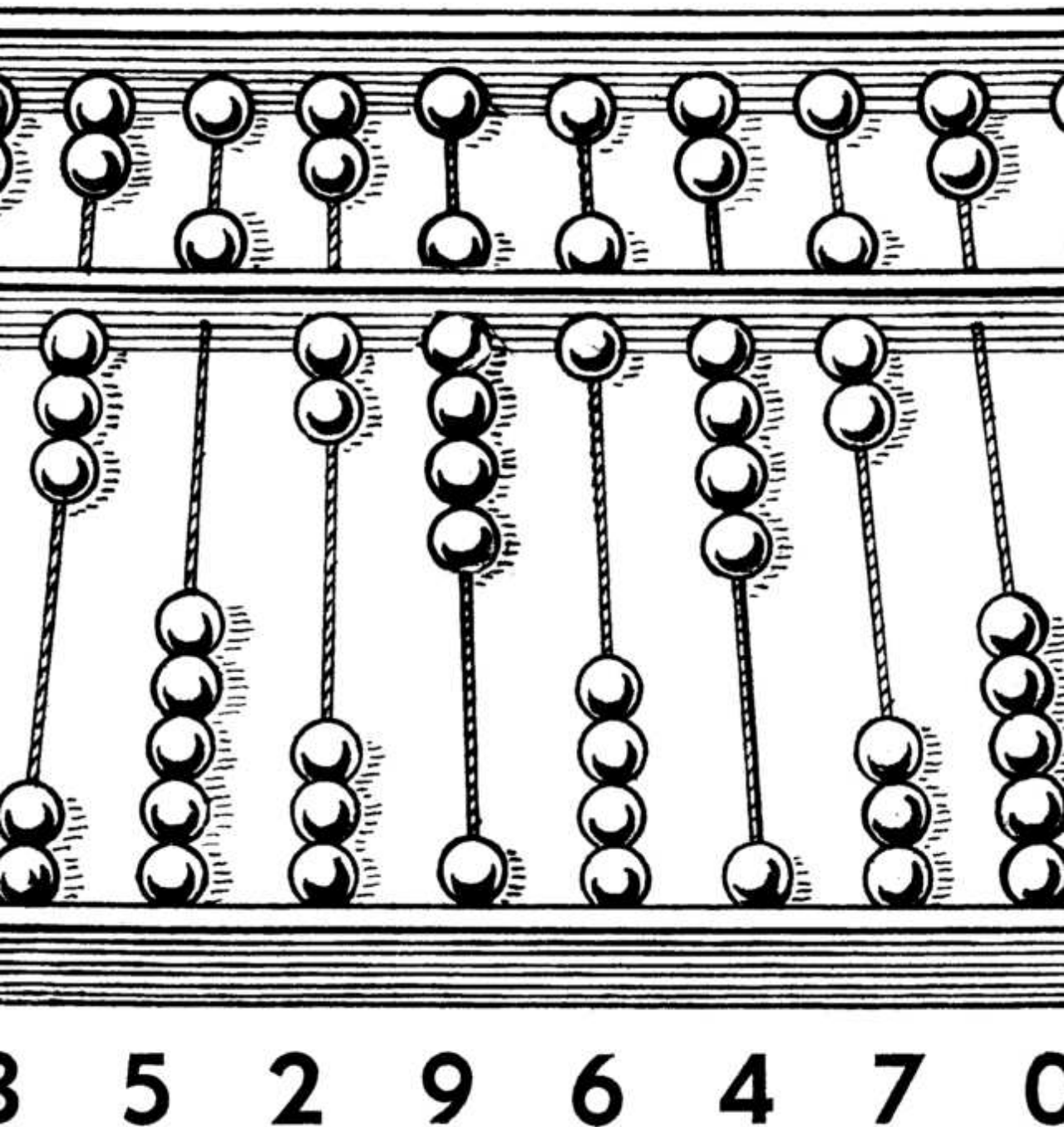
- **Steps**

- Directions laid out in order

- **Done**

- 20–22 minutes at 350°F
- Then cooled off

- Same applies for travel directions



Math ÷


- Every field has algorithms → math is no different
- Examples: multiplying, factoring, square roots, testing primes
- Early perspective: mathematicians were casual
 - ☒ Correct answers were "good enough"
 - Algorithms were just one tool (with logic & proofs)

Math (deeper questions) ?

Over time, mathematicians asked:

- 🔑 What are the **minimum prerequisites**?
- 🪜 What are the **simplest steps**?
- ⌚ Will the algorithm always **finish**?
- 🎯 How can we be sure it gets the **right answer**?

History (preface & early civs)

- Cursory overview (not exhaustive)
- Early civilizations: Egyptians, Babylonians, Indians
 - Basic ops: $+$, $-$, \times , \div
 - Advanced: factorization, square roots
 - Tools:  abacus



Greeks & Romans 🇬🇷🇮🇹

- Euclid's algorithm (GCD), Sieve of Eratosthenes (primes)
 - 📏 Euclid: axiomatic proofs linked to algorithms
- Tools: paper, pen, diagrams
- Romans: focused on engineering
 - But Roman numerals made algorithms cumbersome
- After Greeks → little progress until Arabic texts reached Europe (~1200s)



Indians & Arabs 🇮🇳📖

- Indians: negative numbers, zero, infinity, decimal system, quadratics, combinatorics
 - 🧠 Brahmagupta (7th c.): rules for negatives (add, subtract, multiply)
- Arabs: algebra, induction, irrational numbers → standardized problem statements
 - 💡 Concepts + algorithms = usable mathematics





Al-Khwarizmi & impact 📖

- 825 CE: al-Khwarizmi wrote:
 - *Kitāb al-ḥisāb al-hindī* (Indian computation)
 - *Kitāb al-jam' wa'l-tafrīq* (Addition/subtraction)
 - *Al-Jabr wa'l-Muqābalah* (Algebra)
- Latin translations (~12th c.) → revolutionized Europe
- Introduced arithmetic, decimals, algebra
 - "Algorithm" = from *Algoritmi*
- 🚀 Boosted Europe: caught up mathematically, reduced geometry dominance, showed notation's power
 - Tools: astrolabe, calculators

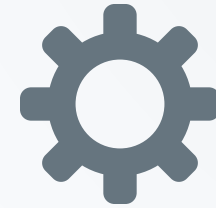
Europe & beyond ⚙️

- Europeans advanced further, but calculations were 😴 boring
- **Applications:** navigation 🌊, physics ⚛️, engineering 🏗️, war ⚔️

Mechanical devices (clocks) → idea of **mechanical calculators**

- New key question:
 - 👉 How to describe an algorithm so a **machine** can perform it?

Algorithms

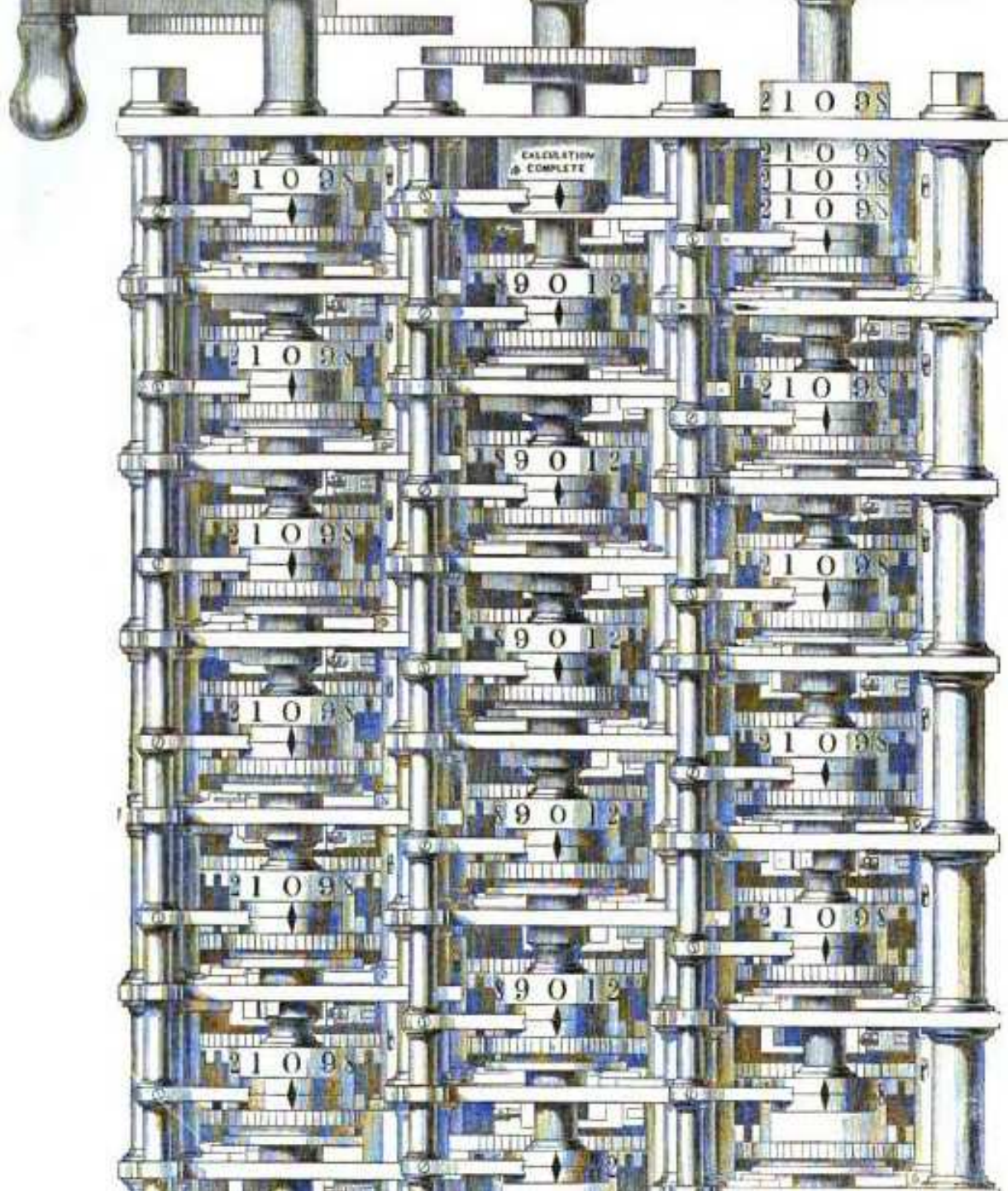


Part 2: Machines

The problem 🤯

- Hand calculation = boring + error-prone
- Logarithms especially tedious
- Machines existed for: $+$ $-$ \times \div
- But surrounding tasks stayed **manual**:
 - Choosing operations & values
 - Handling errors (e.g. divide by zero)
 - Repetition & bookkeeping
- Imagine: mathematician cranking dials, pressing buttons, writing down results





Babbage's idea

- Charles Babbage (1830–1870):
Analytical Engine
- Breakthrough → feed a **list of instructions** instead of manual resetting
- Example program:
 1. Take two numbers a , b
 2. Add them
 3. Repeat 5 times: add result to itself
 4. Output final result
- Computes: $6 \times (a + b)$
- Big change: machine could both **follow algorithm & do math**

Ada Lovelace 💡

- Saw true potential: beyond arithmetic
- Programmed Bernoulli numbers → tedious by hand, feasible by Engine
- Challenge: algorithms had to be **fully explicit**
- Ada had to:
 - Translate algorithm into machine-level detail
 - Explain clearly to others
- Later: Turing gave formal foundations 🧠
- Debate continues: how revolutionary was this step?



Programs

- From here: **programs & programming**
 - Program = algorithm detailed for a machine
 - Programming = creating programs
- Programming is hard, requires languages
- BUT: programs are a poor way to **communicate algorithms** to people
 - Hard to read, especially large ones
 - Raises questions:
 - What counts as “explaining”?
 - What does it mean to “understand”? (links to AI & philosophy)

Programs → Flowcharts



- Programs not ideal for humans → need **another way**
- Enter: **flowcharts**

```
Documents / UnityProjects / SeminarKlausurSimulation - Kopie / Assets / Scripts / Objects.cs
public float surroundingTemperature = 20f;

public float surfaceArea;
public float heatingFactor = 10f;

public float coolingFactor = 1f;

public float rayAmount = 3000;

public Gradient VisualTemperature;

private void Start()

    gameObject.GetComponent<Renderer>().material.color = VisualTemperature.Evaluate(0f);

    var GameController = GameObject.FindWithTag("GameController");
    temperature = GameController.GetComponent<gameController>().StartingTemperature;
    surroundingTemperature = GameController.GetComponent<gameController>().StartingTemperature;

    // getting the Ammount of rays emmitet to calculate the heating factor
    rayAmount = GameObject.Find("Rayemmitter").GetComponent<Raycast>().RayAmount;
    print("RayAmmount: " + rayAmount);

private void Update()

    coloring();
    adaptTemperature();

private void coloring()

    gameObject.GetComponent<Renderer>().material.color = VisualTemperature.Evaluate(temperature / 30f);

private void adaptTemperature()

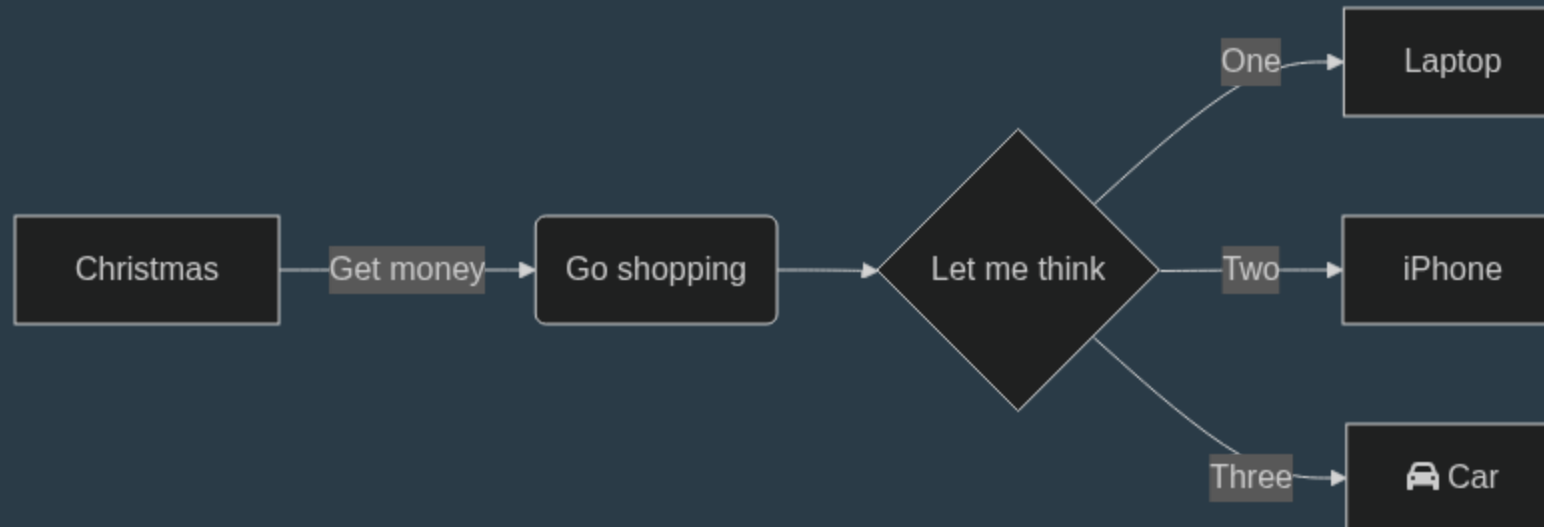
    float TempDif = temperature - surroundingTemperature;
    temperature -= TempDif * coolingFactor * Time.deltaTime;

public void HeatingUp()

    if (temperature < maxTemperature)
    {
        temperature += DayCycle.DaytimeMultiplier * heatingFactor / surfaceArea / (rayAmount * 3f);
    }
}
```

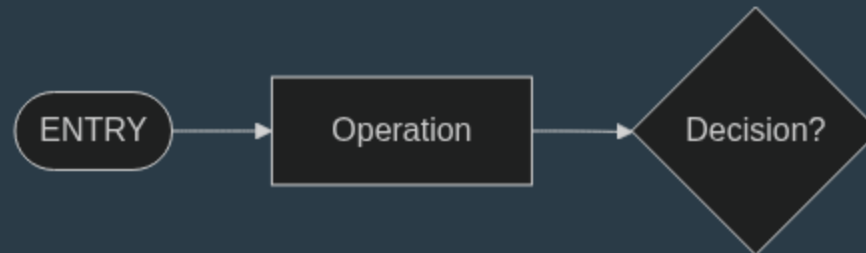
Flowcharts

- Graphical representation of algorithms
- Easier to read than raw code
- Use boxes, diamonds, arrows, and text



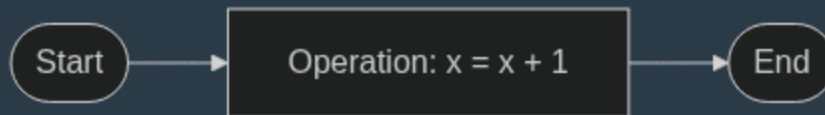
Flowcharts (conventions)

- Conventions:
 - Entry → start point
 - Box → operation (read input, arithmetic, assign)
 - Diamond → decision (yes/no branch)
 - Lines → show flow, can carry conditions/data



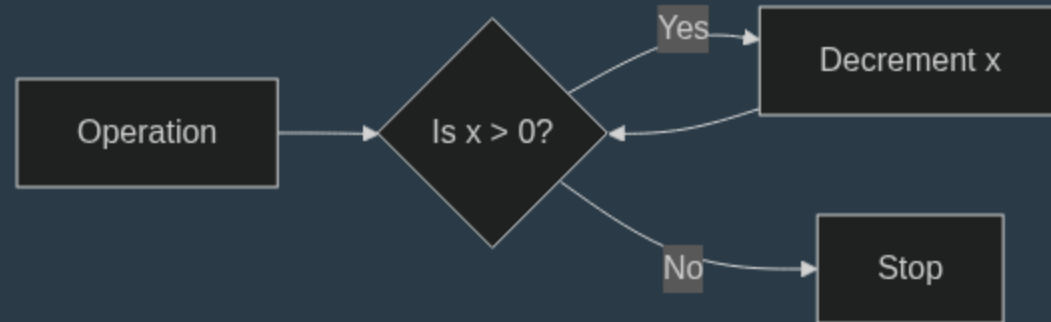
Flowcharts (operations)

- The **box** = operation
 - Simple action: input, calculation, assignment



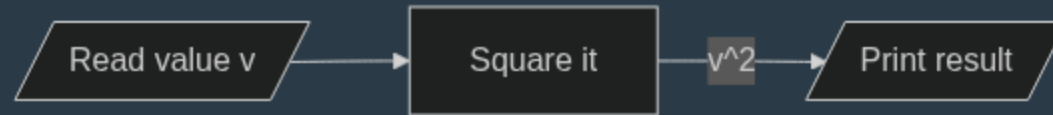
Flowcharts (decisions)

- The **diamond** = decision
 - Compare something → branch Yes/No



Flowcharts (lines)

- Lines may include:
 - Condition to be true for path
 - Data passed along that path



Flowcharts (styles & UML)

- Many conventions exist → all similar
- No one “right” layout → goal is **communication**
- Flowcharts can:
 - Be docs, planning tools, or debugging aids
 - Split large projects among teams
- UML (Unified Modeling Language) = flowchart's more formal cousin
 - Used in high-precision projects

Pseudocode

- Closer to code but still human-readable
- Looks like code, won't run
- Flexible, less standardized than flowcharts
- Example:

```
for i ← 1 to n do  
    read x  
    s ← s + x  
output s
```

Pseudocode (examples)


Informal

```
Read a, b, c
max ← a
If b > max then max ← b
If c > max then max ← c
Print max
```

Formal

```
input:  $n \geq 0$ 
output:  $n!$ 
p ← 1
for i ← 2 to n do
    p ← p · i
return p
```

Pseudocode vs Code

- Ideally, pseudocode  code translation is straightforward

Pseudocode

```
for i ← 1 to n do
    read x
    s ← s + x
output s
```

Java

```
for (int i=0; i<n; i++) {
    int x = sc.nextInt();
    s = s + x;
}
System.out.println(s);
```

Pseudocode (guidelines)



- Omit language-specific quirks
- Keep general, map to any language
- Adapt for audience (math-heavy vs general)
- Example:

Math-friendly:

```
input:  $a_1 \dots a_n$   
 $s \leftarrow 0$   
for  $i \leftarrow 1..n$  do  
     $s \leftarrow s + a_i$   
output  $s$ 
```

General audience:

```
Read a list of numbers  
Add them up  
Print the total
```


Big Ideas in Algorithms

Idea 1

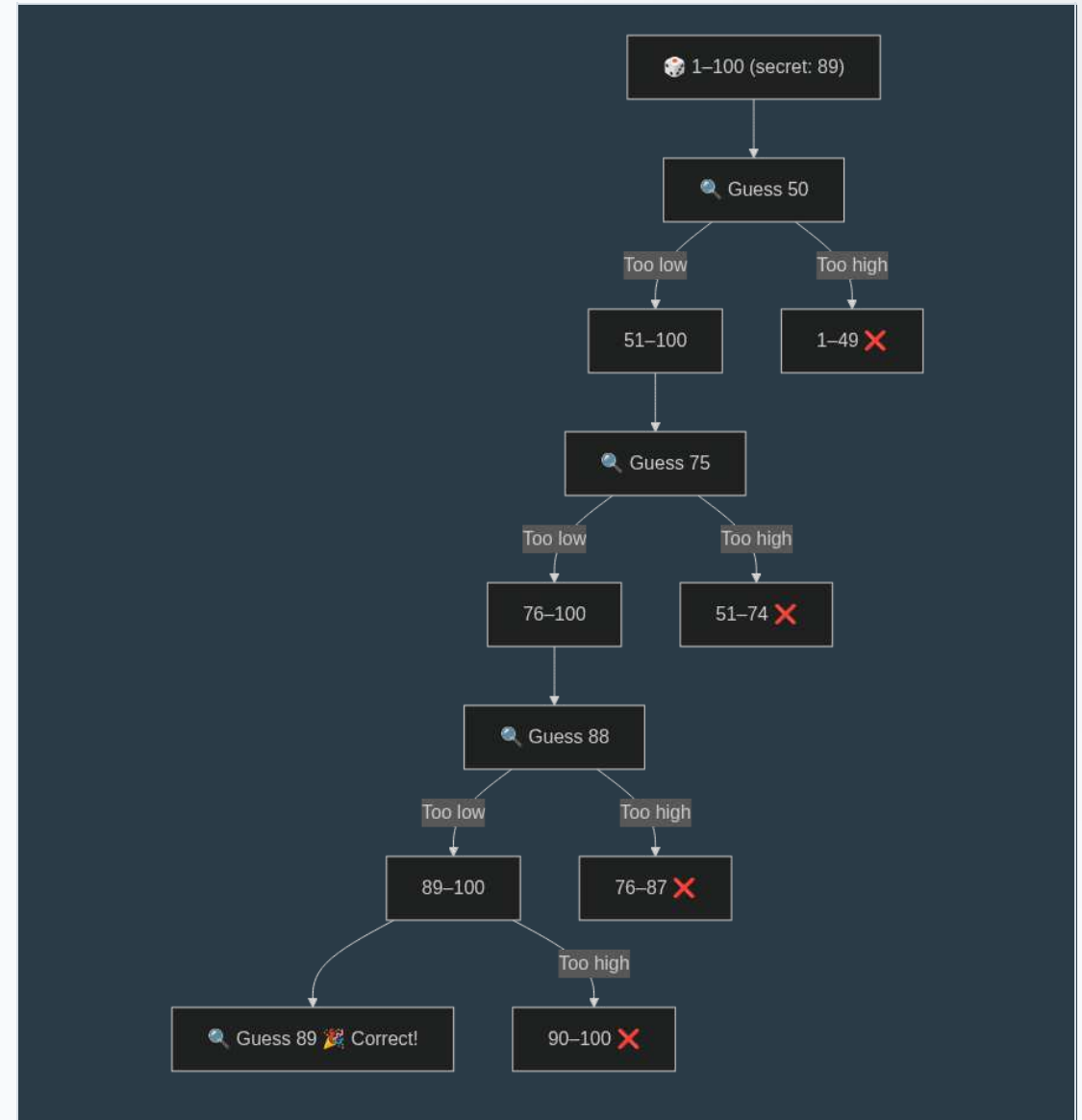
- Imagine I pick a number between **1 and 100** 🎲 .
- You can ask: *"Is it higher or lower?"*
- 🤔 If you guess randomly, it could take 100 tries.

⚡ Can you do better?

Idea





Binary Search

- Always guess the **middle** of the range.
- Each time, cut the possibilities in **half**.
- Find the secret number in just **7 tries**. 🤖



Why it matters ?

Binary Search

- Searching in a dictionary 
- Finding songs in your playlist 
- Quickly locating files on a laptop 
- Finding players with similar rank in multiplayer games 

Idea 2

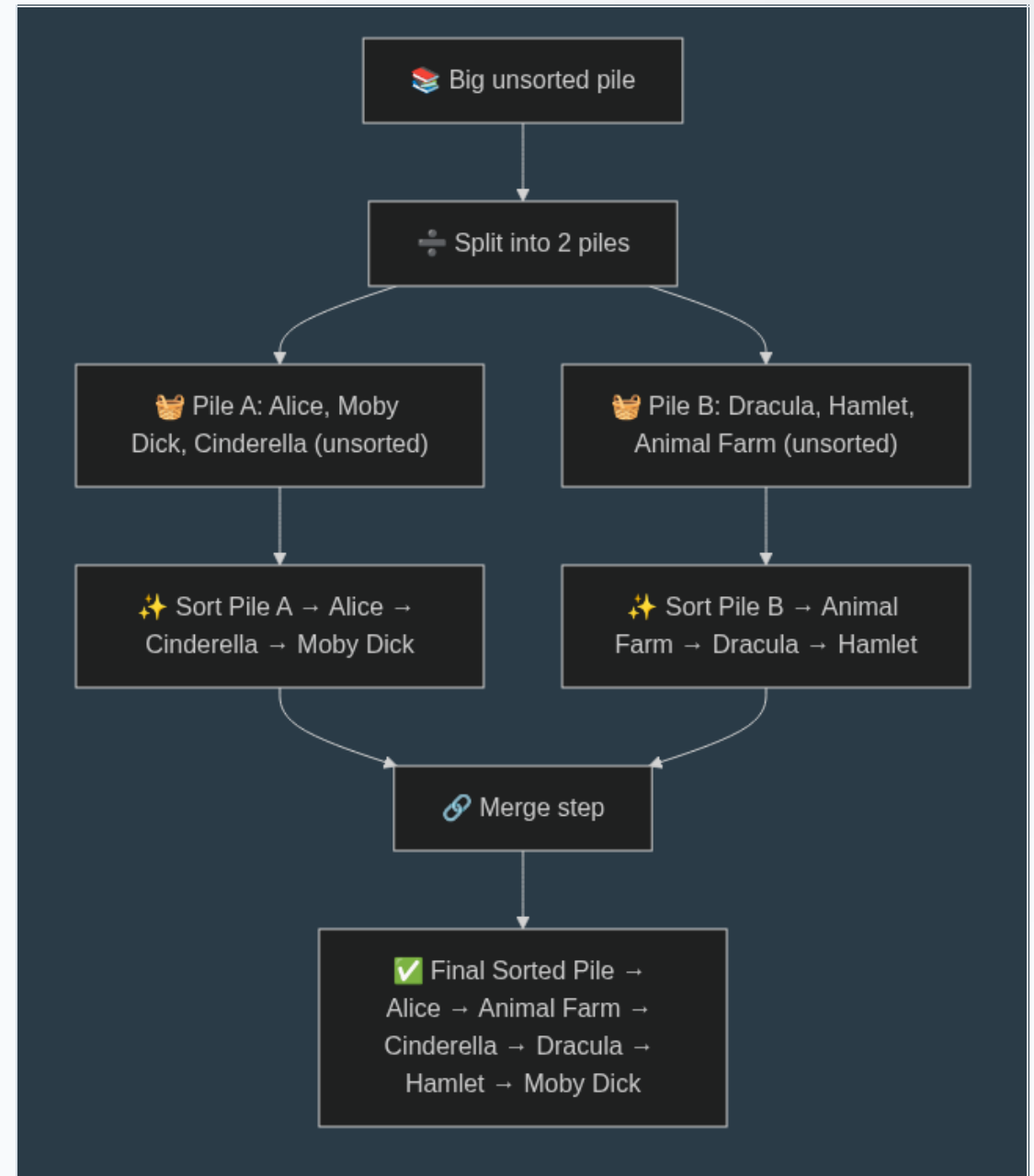
Problem

- You have a pile of books all jumbled together. 📖
- Goal: Put them in order (alphabetically by title).
- Sorting the whole pile at once is messy. 😵

Idea

Merge Sort

- Split the pile into smaller piles.
- Sort each small pile individually (easy!).
- Merge piles
- Continue until you have one big, sorted pile. ✓




Why it matters

- Organizing playing cards 🃏
- Sorting names at graduation 🎓
- Sorting contacts on your phone ⚙️

Idea 3

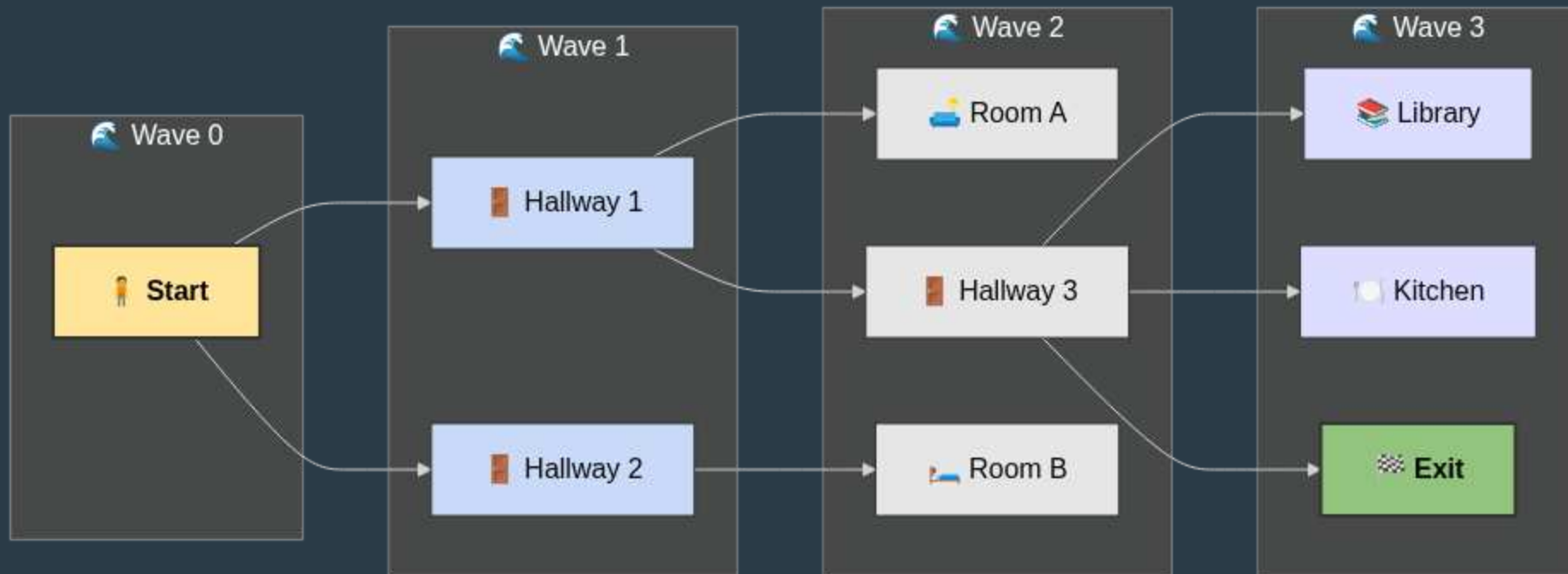
Problem

- A **fire alarm** rings in your ICS class (very unrealistic scenario). 
- You want the **closest exit**.
- What's the safest way to explore?




Idea

Breadth-First Search (BFS)

- Explore in **waves**:
 - First check all rooms next to you.
 - Then the rooms next to them.
 - Keep going until you find the exit.
- Guarantees the **shortest way out**.



Why it matters

- Social networks: “You and Sam are 3 steps apart” 
- GPS: finding shortest bus routes 
- Emergency drills: nearest exit 

Idea 4

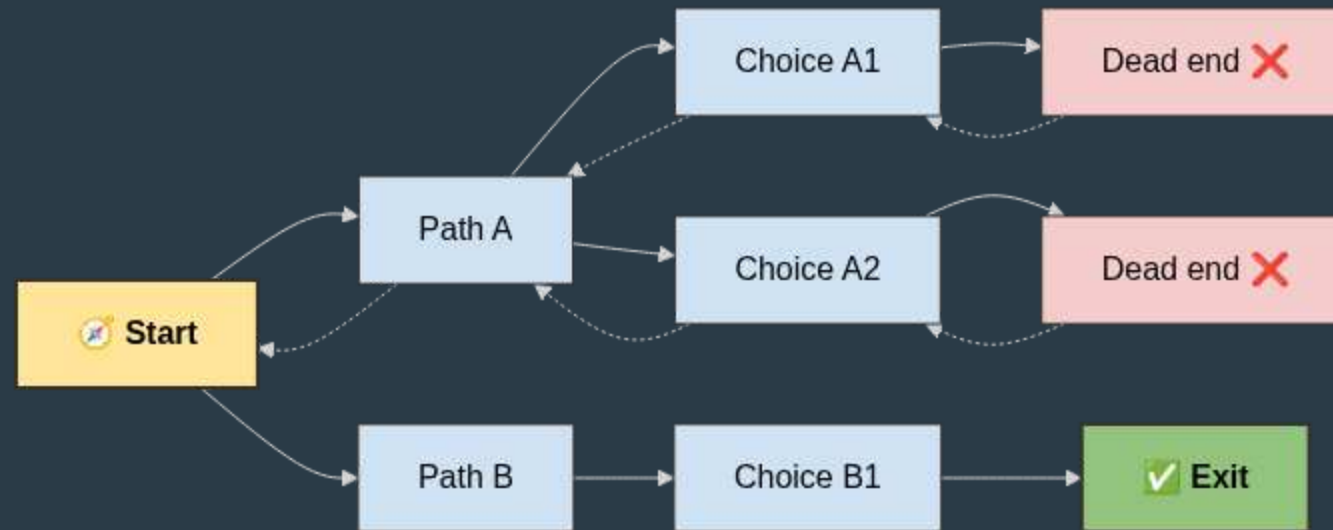
Problem

- You're in a **maze**. 🌀
- You try one path → it's a dead end.
- What do you do?




Idea

Backtracking

- Explore one path as far as possible.
- If blocked → **step back**.
- Try another path until you succeed.



Why it matters

- Solving Sudoku 
- Escape rooms 
- Crossword puzzles 
- ~~Autocomplain~~ Autocomplete

Idea 5

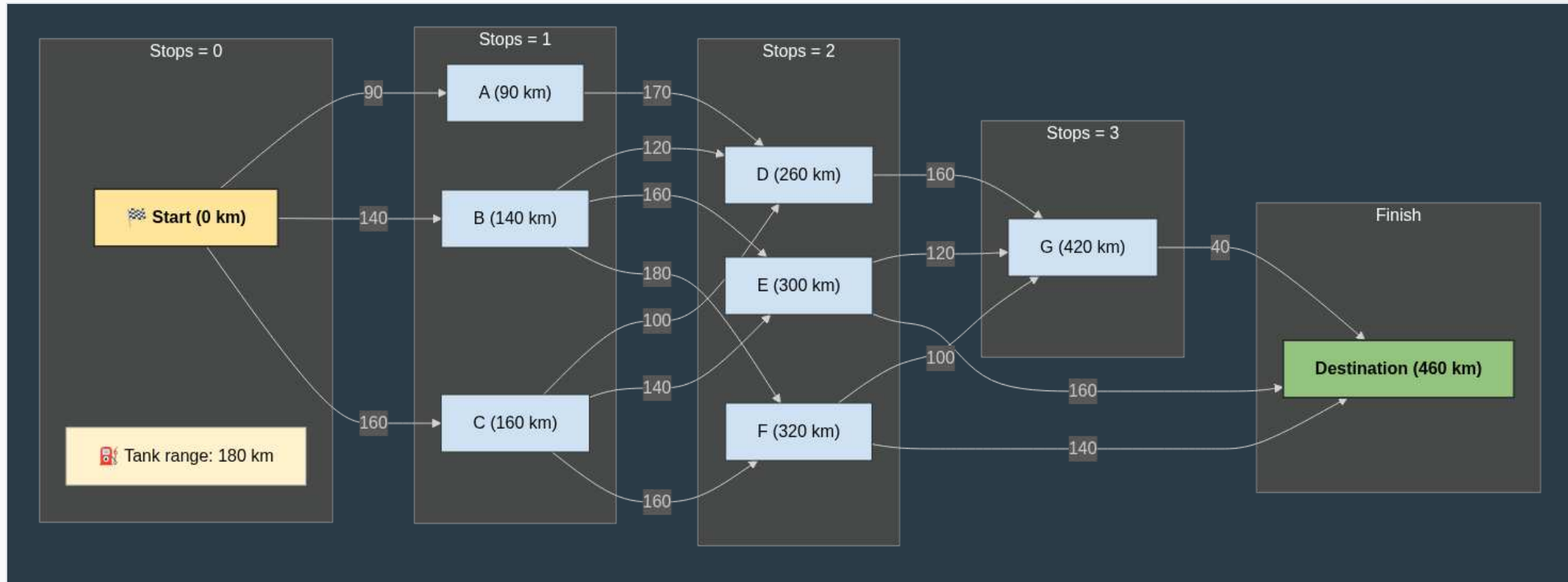
Problem

- You're planning a **road trip across Europe** 🚗.
- You want to stop for gas as few times as possible.
- There are many stations, but you can only drive a limited distance per tank.
- How do you plan?





Idea

Dynamic Programming

- Think of smaller trips first:
 - Best way to drive 100 km.
 - Then 200 km.
 - Then 300 km.
- Reuse these answers to plan the full trip.
 - 👉 Build up the solution step by step.



Why it matters

- Travel planning 
- Budget optimization 
- Scheduling projects 
- DNA/protein analysis 

Idea 6

Problem

You're playing a **dice** game. 🎲

- The math is messy.

How can you find
your chances of
winning?



Idea

Monte Carlo

- Play the game many times.
- Count how often you win.
- Probability = wins ÷ trials.

Why it matters

- Casinos & gambling 🎰
- Predicting weather ☀️☁️
- Stock market risks 📈📉
- Movie recommendations 🍿

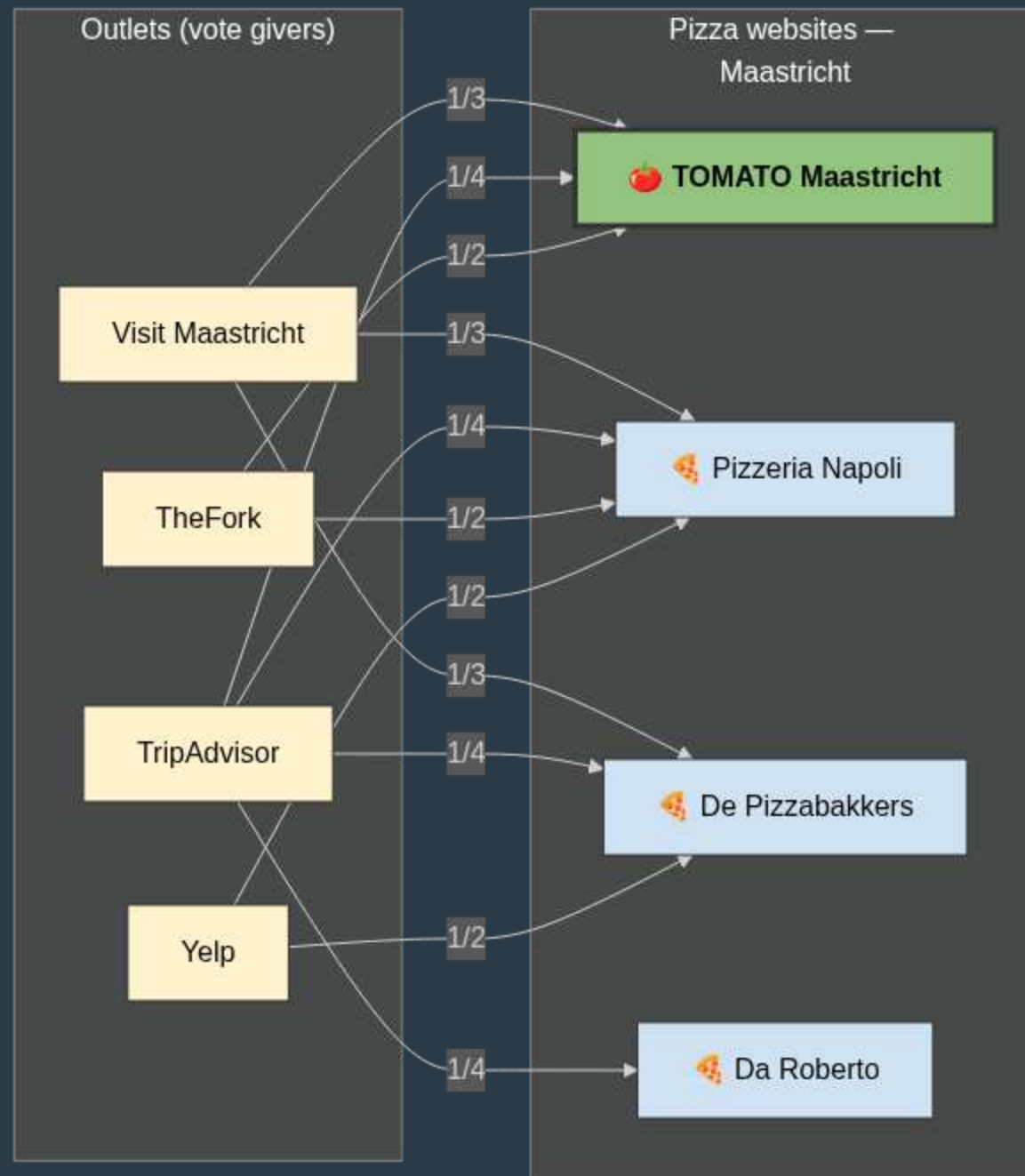
Idea 7

- You search “best pizza near me” 🍕.
- Thousands of results.
- How does Google decide who's first?




Idea

PageRank

- Each website **votes** by linking to others.
- Votes from popular sites count more.
- Pages with most “authority” rise to the top.



Why it matters

- Search engines 
- Instagram influencers 
- Academic paper citations 

What we learned 🎓

- **Binary Search:** halve the problem each time.
- **Merge Sort:** divide work, then combine.
- **BFS:** ripple out to find shortest path.
- **Backtracking:** try → undo → try again.
- **Dynamic Programming:** build answers from smaller ones.
- **Monte Carlo:** use randomness to estimate.
- **PageRank:** trust flows through links.

**See you in
the next
lecture!**

