More on Algorithms

Iterative Improvement

It is a step-by-step process of improving an algorithm in order to make it more efficient, robust and simpler. It can be done so by first starting with a basic solution and gradually building on it.

It is important because:

- Algorithms maybe correct but innefficient
- Optimization of time, space and scalability

While doing this sort of improvement:

- Start with a correct algorithm (the innefficient one)
- Test and analyze its performance
- Identify bottlenecks (unwanted instructions, assignments, nested loops)
- Refine the algorithm by using better strategies and data structures.

Keep doing this until the efficiency and performance reaches a satisfactory level.

Algorithm Correctness

An algorithm is correct if, for any valid input it produces the expected output. It can be found using two parameters:

1. **Partial Correctness**: It basically talks about the logic of the algorithm being correct
2. **Termination**: Talks about if the algorithm terminates to produce an output.

Hence, **Full Correctness** = **Partial Correctness** + **Termination**

To ensure correctness:

- Define problem specifications (Input / Output)
- Use mathematical proofs such as induction
- Test with multiple test cases — Normal, Edge (Smallest / Largest), Invalid

Algorithm Efficiency

Mesaures how well an algorithm is in terms of using resources such as time and memory. A correct algorithm may still be inefficient.

It is important, especially for large scale problems because computers have limited resources, and large inputs may take a very long time (hours or days) to get processed by a slow algorithm.

There are several factors used to check the efficiency of an algorithm:

- **Time Efficiency**
  - Each computation step contributes. It is the number of computational steps.
- **Memory Efficiency**
  - Amount of space / memory occupied in RAM

Common Mistakes in Algorithms

Here are the list of common mistakes made while designing an algorithm:

- Infinite (never-ending) loops
- Wrong base cases
- Off-by-one errors (Index errors mostly)
- Unnessacery complexity
- Ignoring edge cases
- Wrong assumptions
- Lack of clarity in steps

Iterative and Recursive Algorithms

Algorithms involving looping can be done in two ways — Iterative and recursive. Here is a comparison for both methods:

	Iterative	Recursive
Method	Loops	Functions
Space	O(1)	O(n)

	Iterative	Recursive
Time	O(n)	O(n)
Ease of Code	Long	Elegant & Compact
Efficiency	High	Low
Limitations	None	Memory

Summary

- Iterative improvement: start with a correct baseline, measure performance, find bottlenecks, and refine with better strategies or data structures until efficiency is satisfactory.
- Algorithm correctness = partial correctness + termination. Define clear I/O specs, use proofs (e.g., induction), and test with normal, edge, and invalid cases.
- Efficiency focuses on resource use: time (number of computational steps) and memory (RAM consumed). Correctness alone is not enough for large inputs.
- Common pitfalls: infinite loops, wrong base cases, off-by-one errors, unnecessary complexity, ignoring edge cases, wrong assumptions, and unclear steps.
- Iterative vs recursive: loops vs function calls. Iterative typically uses O(1) space and is often more efficient; recursive is concise and elegant but may use O(n) space and be limited by memory.