METROPOLITAN TIRANA UNIVERSITY

Course: Data Structures and Algorithms

The Power of Algorithms

Evis Plaku

Motivation for the Course

Kylie Jenner
@ikyliejenner

Can you guys please recommend books that made you cry?

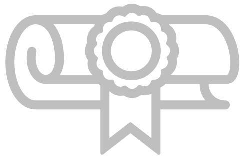
Saransh Garg @saranshgarg
Replying to @ikyliejenner

Data Structures and Algorithms in Java (2nd Edition) 2nd E
by Robert Lafore (Author)

- Challenging but essential
- Think like a programmer
- Interview must have. Real impact

Course Organization

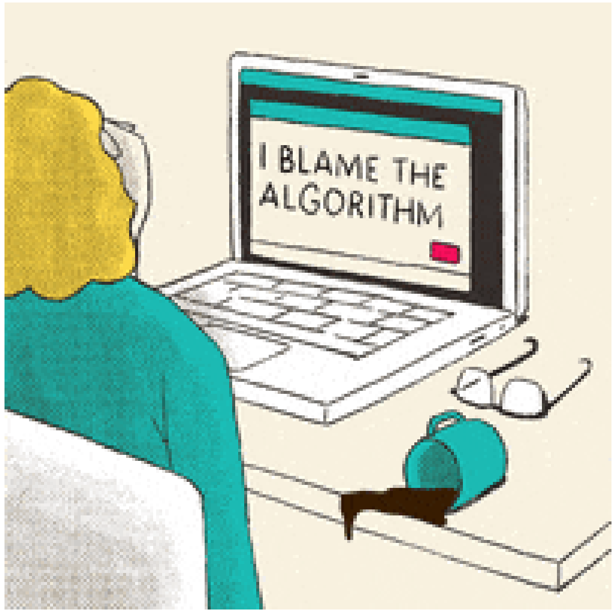
- 📅 Lectures on Tuesday | Seminars on Thursday
- 📚 Final Exam: 50 points
- 📝 Continuous Evaluation: 50 points
    - 💡 Active participation: 10 points
    - 🖊️ Midterm: 20 points
    - 💻 Labs: 20 points



What is an Algorithm

A step by step procedure to solve a problem efficiently

- **Precise**: follows a well-defined sequence of steps
- **Efficient:** optimized for time and resources
- **Generalizable:** solves multiple instances of the problem

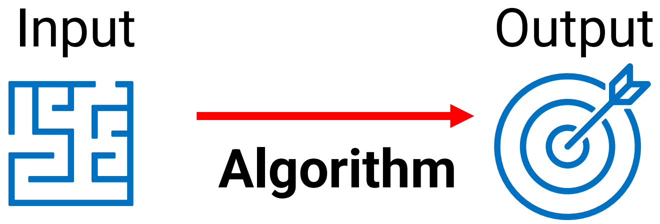


Algorithms: the Recipes of Problem Solving

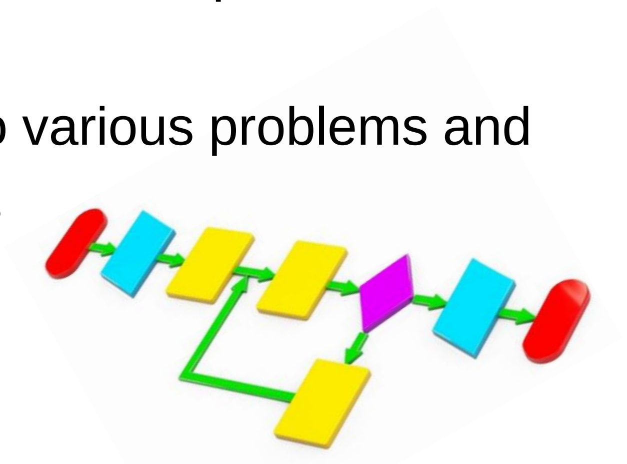
METROPOLITAN TIRANA UNIVERSITY

- A systematic, step-by-step guide for solving problems

- **Transforms input to output:** processes data to produce results

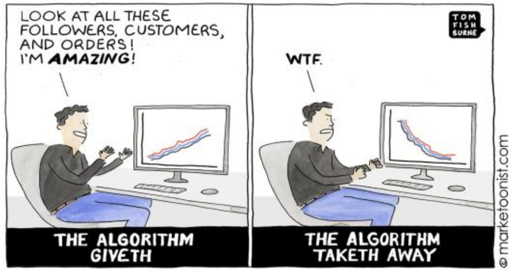


- Applies to various problems and scenarios



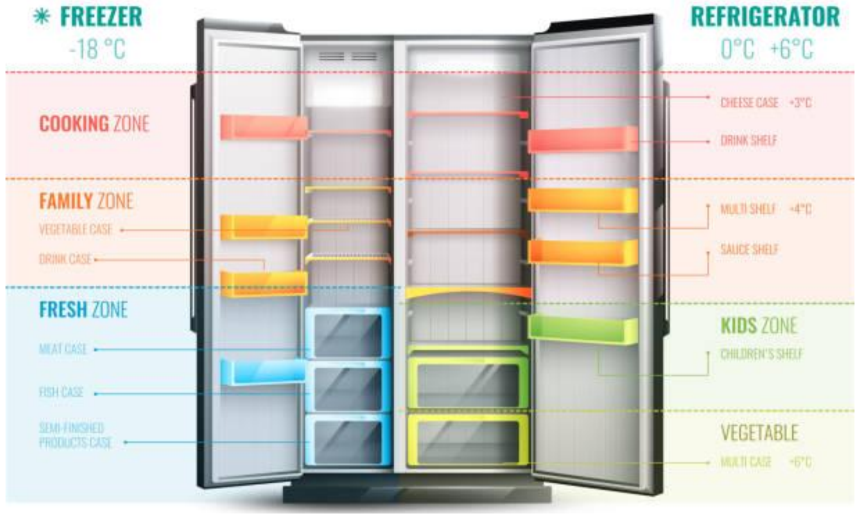
What Makes an Algorithm Good

- **Deterministic**: each step follows logically, ensuring consistent results
- **Unambiguous**: every operation is clearly defined and without confusion
- **Finite**: it terminates after a defined number of steps
- **Correctness:** it reliably produces the desired output for valid inputs
- **Scalable**: works well with varying sizes of input data


LOOK AT ALL THESE FOLLOWERS, CUSTOMERS, AND ORDERS! I'M AMAZING!
THE ALGORITHM GIVETH
WTF.
THE ALGORITHM TAKETH AWAY
TOM FISHBURNE
© marketoonist.com

Why do we need algorithms?

The Fridge Organization Problem



Algorithms help to
sort things out

Finding Your Socks in the Laundry

Algorithms can
search for you

The Friend who Always Overpacks

Algorithms can **minimize storage**
and **optimize choices**

Get inspired by other disciplines

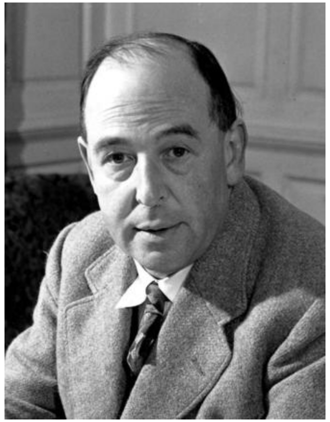
C. S Lewis on Rules of Writing

- Always try to use the language so as to make quite clear what you mean and make sure your sentence couldn’t mean anything else

- Always prefer the plain direct word to the long, vague one

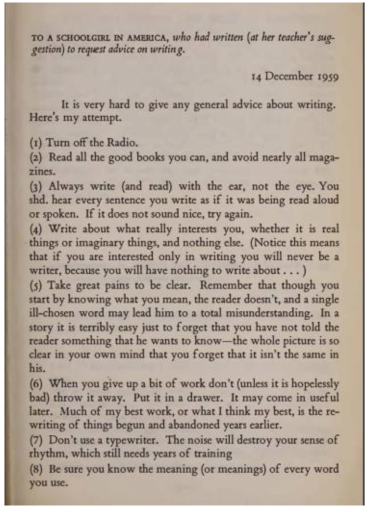
- Never use abstract nouns when concrete ones will do. If you mean 'More people died,' don’t say 'Mortality rose.’”

- Don’t use words too big for the subject



C. S. Lewis, British writer
29.11.1898 – 22.11.1963

Rules of Writing
Lewis’ letter to young school girl



TO A SCHOOLGIRL IN AMERICA, *who had written (at her teacher's suggestion) to request advice on writing.*

14 December 1959

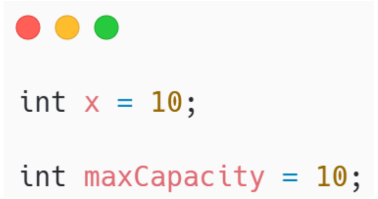
It is very hard to give any general advice about writing. Here's my attempt.

(1) Turn off the Radio.
(2) Read all the good books you can, and avoid nearly all magazines.
(3) Always write (and read) with the ear, not the eye. You shd. hear every sentence you write as if it was being read aloud or spoken. If it does not sound nice, try again.
(4) Write about what really interests you, whether it is real things or imaginary things, and nothing else. (Notice this means that if you are interested only in writing you will never be a writer, because you will have nothing to write about . . .)
(5) Take great pains to be clear. Remember that though you start by knowing what you mean, the reader doesn't, and a single ill-chosen word may lead him to a total misunderstanding. In a story it is terribly easy just to forget that you have not told the reader something that he wants to know—the whole picture is so clear in your own mind that you forget that it isn't the same in his.
(6) When you give up a bit of work don't (unless it is hopelessly bad) throw it away. Put it in a drawer. It may come in useful later. Much of my best work, or what I think my best, is the re-writing of things begun and abandoned years earlier.
(7) Don't use a typewriter. The noise will destroy your sense of rhythm, which still needs years of training
(8) Be sure you know the meaning (or meanings) of every word you use.

Rules of Writing Good Algorithms

METROPOLITAN TIRANA UNIVERSITY

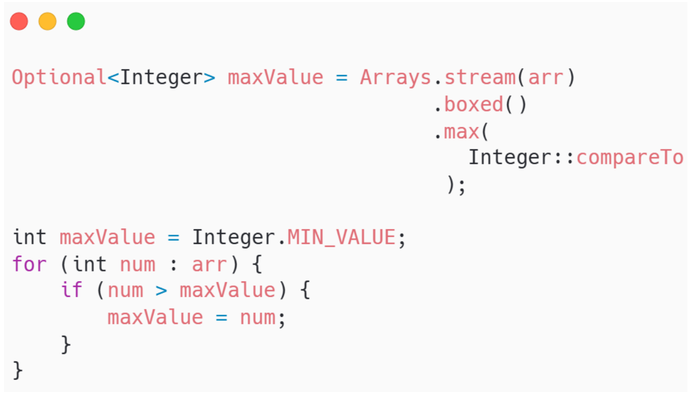
- Be clear and precise in your code



```java
int x = 10;

int maxCapacity = 10;
```

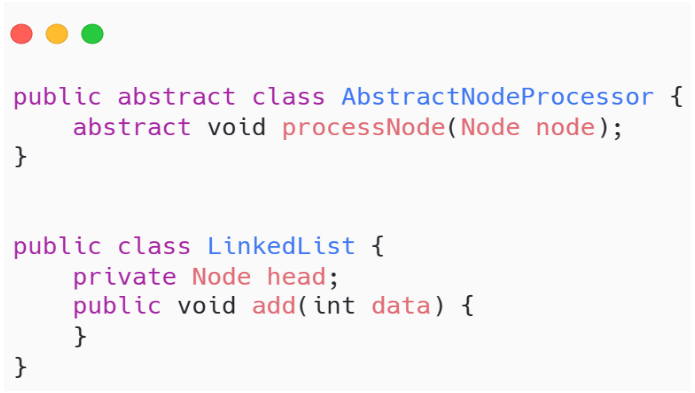
- Prefer simple, direct solutions over complex ones



```java
Optional<Integer> maxValue = Arrays.stream(arr)
                                   .boxed()
                                   .max(
                                      Integer::compareTo
                                    );

int maxValue = Integer.MIN_VALUE;
for (int num : arr) {
    if (num > maxValue) {
        maxValue = num;
    }
}
```

- Use concrete concepts over abstract ones when possible

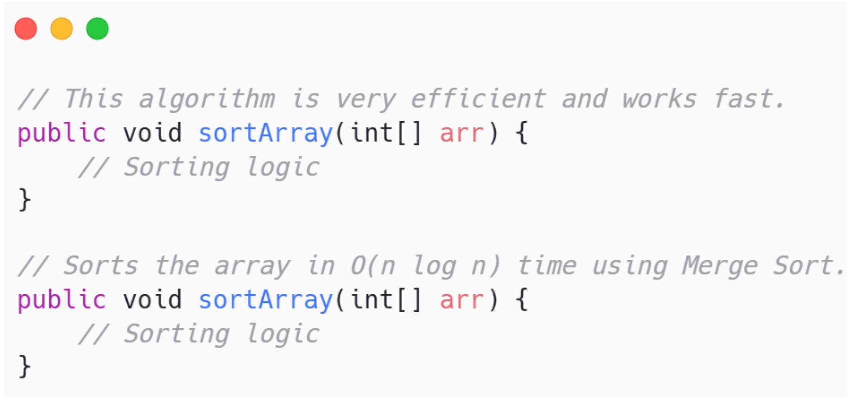


```java
public abstract class AbstractNodeProcessor {
    abstract void processNode(Node node);
}

public class LinkedList {
    private Node head;
    public void add(int data) {
    }
}
```

Rules of Writing Good Algorithms

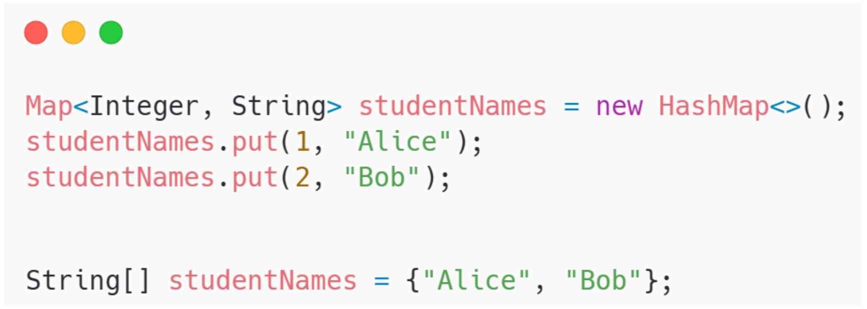
- Show algorithm efficiency, don’t just describe it



```java
// This algorithm is very efficient and works fast.
public void sortArray(int[] arr) {
    // Sorting logic
}

// Sorts the array in O(n log n) time using Merge Sort.
public void sortArray(int[] arr) {
    // Sorting logic
}
```

- Choose appropriate data structures, not overkill solutions



```java
Map<Integer, String> studentNames = new HashMap<>();
studentNames.put(1, "Alice");
studentNames.put(2, "Bob");


String[] studentNames = {"Alice", "Bob"};
```

Stand for your principles

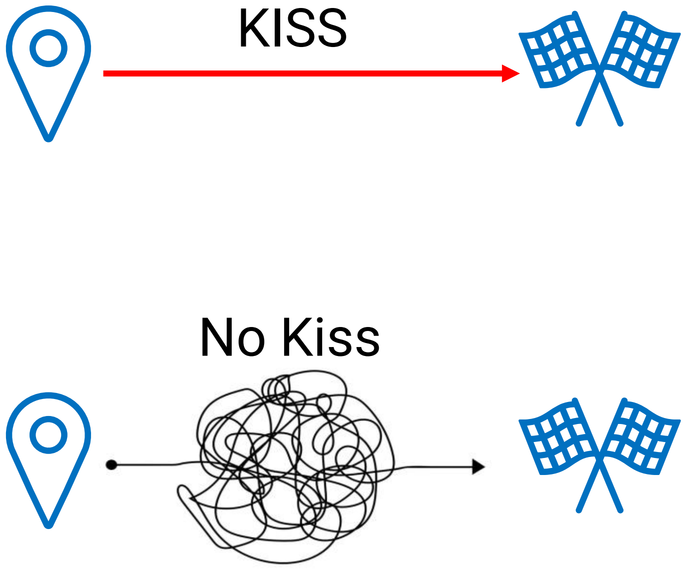
KISS Principle

METROPOLITAN TIRANA UNIVERSITY

Keep It Simple, Stupid

- First, choose the most simple and efficient data structure or algorithm **that gets the job done**

- Don’t overcomplicate

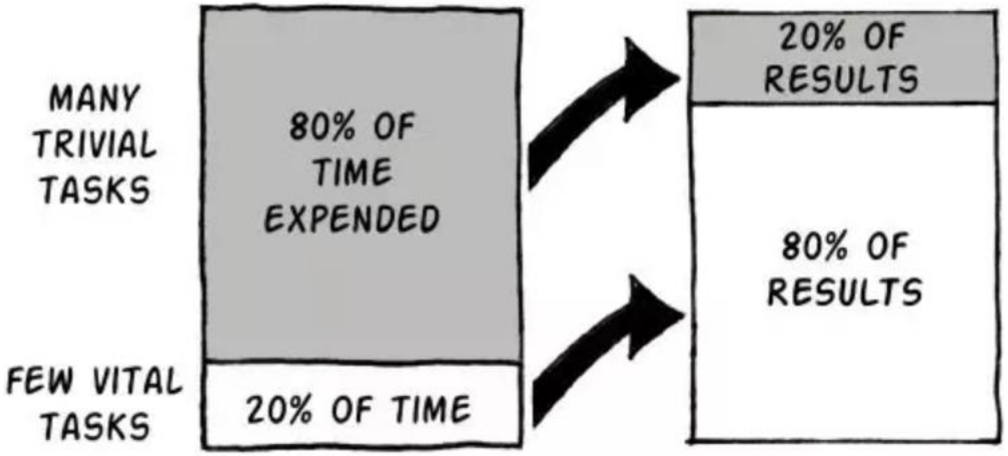


Pareto Principle (80 / 20 rule)

20% of actions often yield 80% of results

- Optimize first parts of the code with most impact

- Don’t micro optimize

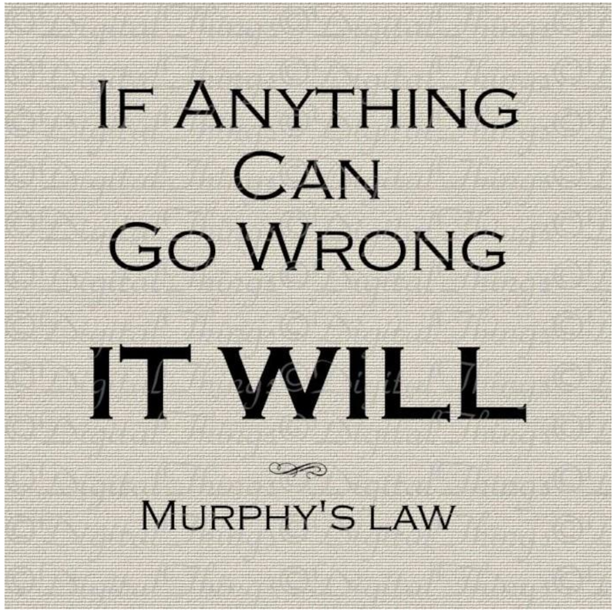


Murphy's Law

Anything that can go wrong, will go wrong

- Always consider edge cases (empty input, large values, null references etc)

- Test for extreme cases


IF ANYTHING CAN GO WRONG

IT WILL

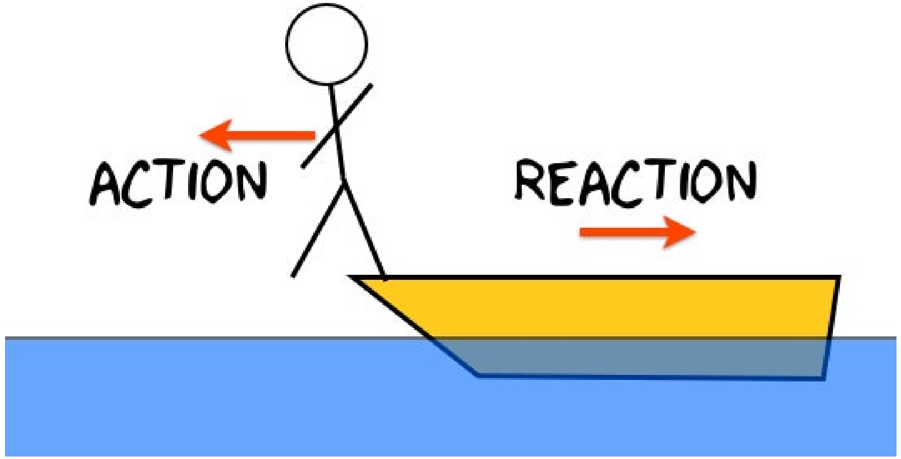
MURPHY'S LAW

Newton’s Third Law

METROPOLITAN TIRANA UNIVERSITY

⇆ For every action, there is an equal and opposite reaction

- Space vs time trade-off
- Optimizing for speed usually requires more memory and vice versa



Sherlock Holmes Deductive Reasoning

METROPOLITAN TIRANA UNIVERSITY

Eliminate impossible options and focus on facts

- When debugging, systematically analyze all inputs and outputs

- Use print statements and dry runs

The Tortoise and the Hare

The slow but steady tortoise wins the race against the fast but inconsistent hare

- Some algorithms may seem fast for small inputs but perform poorly with large ones

- Rely on Big O notation

Understanding Algorithm Efficiency

Time and Space Complexity

Time and Space Complexity

Understanding efficiency helps improve algorithm performance and scalability

- **Optimize performance**: ensure fast execution for large inputs
- **Manage resources**: limit memory usage and storage needs
- **Scalability:** handle increasing input sizes effectively

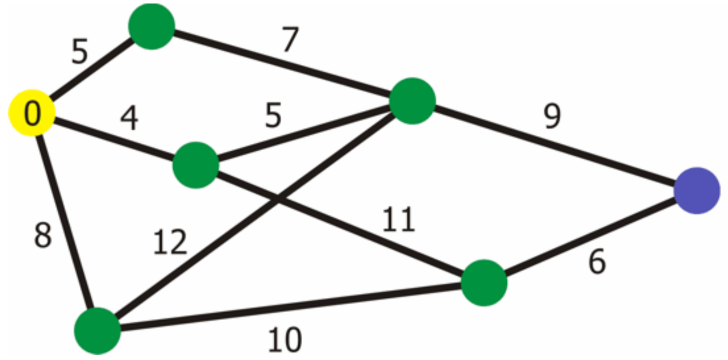


Illustration: a graph algorithm

Time and Space Complexity

METROPOLITAN TIRANA UNIVERSITY

Time complexity	Space complexity
• Time to run algorithm	• Memory required
• Based on input size	• Accounts for all storage
• Behavior for large inputs	• Memory usage growth
• Big O notation	• Big O notation

Optimizing one often compromises the other

Balance is key

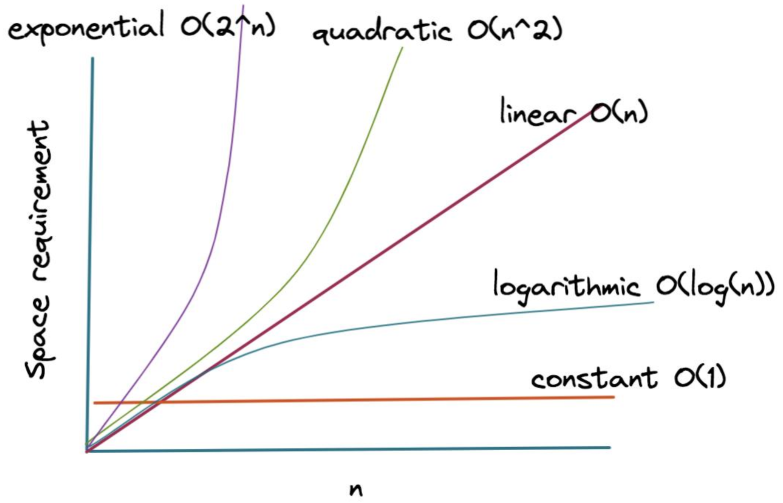
Big O Notation

A mathematical notation to describe algorithm performance

- Describes time/space complexity growth

- Usually focuses on upper-bound behavior

- Helps compare algorithms regardless of hardware/environment



Big O Mathematical Formulation

$$O(f(n) = \{g(n) : \exists c > 0, n_0 \geq 0\}$$

$$such\ that\ for\ all\ n \geq n_0, |g(n)| \leq c \cdot f(n))$$

- $f(n)$ algorithm's time or space complexity
- $g(n)$ function that describes the growth rate
- c is a constant factor, and n_0 is the input size where the behavior starts to hold true

expresses how the algorithm's resource usage grows as the input size (n) increases

Best, Average and Worst Time Complexity

- **Best Time (Ω notation)**: minimum performance for optimal inputs

$$\boxed{T_{best} = \Omega(f(n))}$$

- **Average Time (Θ notation)**: expected performance over typical inputs

$$\boxed{T_{average} = \Theta(f(n))}$$

- **Worst Time (O notation)**: maximum performance for worst-case inputs

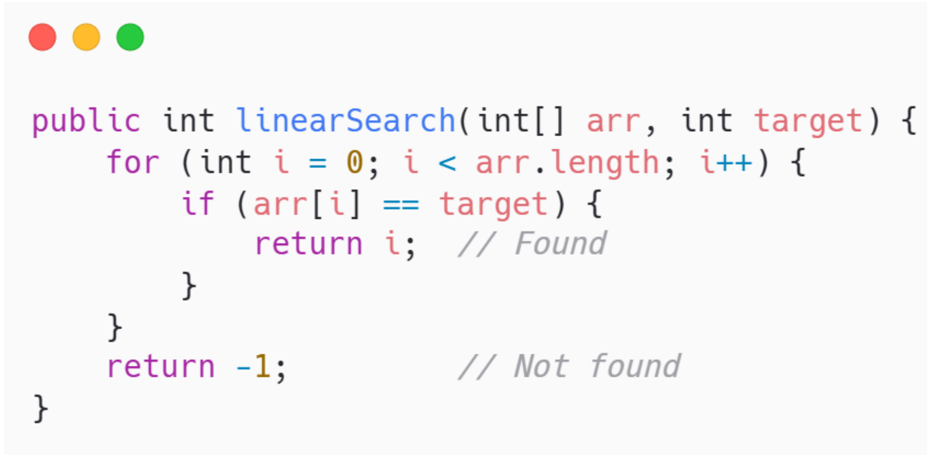
$$\boxed{T_{worst} = O(f(n))}$$

☞ These notations capture the range of possible execution times

Understanding Complexity through Examples

METROPOLITAN TIRANA UNIVERSITY

Linear search



```java
public int linearSearch(int[] arr, int target) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == target) {
            return i;  // Found
        }
    }
    return -1;         // Not found
}
```

Checking if an element is part of an array

- Element is the first one.

$$T_{best} = \Omega(1)$$

- Target is somewhere *in middle*

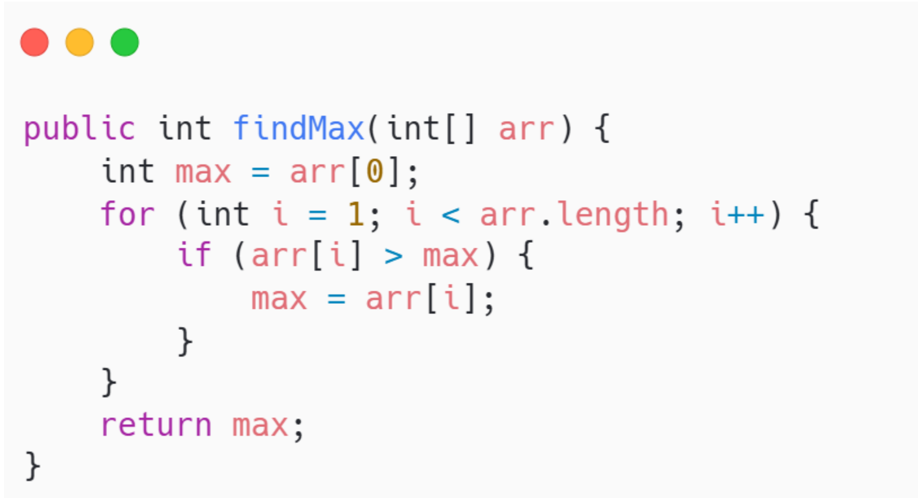
$$T_{average} = \Omega(n)$$

- Target is the last element (or not present)

$$T_{worst} = O(n)$$

Understanding Complexity through Examples

Finding maximum element in a list



```
public int findMax(int[] arr) {
    int max = arr[0];
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}
```

Best case = Average Case = Worst Case

$$\Omega(n) = \Theta(n) = O(n)$$

- All elements must be checked to ensure maximum is found
- What if array is already sorted?

Common Big O Time Complexities

- $O(1)$	Constant Time	Checking if number is even Accessing an element in an array
- $O(\log n)$	Logarithmic Time	Looking up a word in a dictionary Finding a contact in a sorted phonebook
- $O(n)$	Linear Time	Counting occurrences of words in a book Checking if a list contains a value
- $O(n \log n)$	Linearithmic time	Sorting a list of names Merging large log files by timestamp

Common Big O Time Complexities

- $O(n^2)$	Quadratic Time	Comparing all students for similarity Checking for duplicate names in list
- $O(2^n)$	Exponential Time	Generating all subsets of a set Solving Towers of Hanoi
- $O(n!)$	Factorial time	Finding all possible seat arrangements Brute force to solve traveling salesman

Avoid **quadratic, exponential, and factorial** solutions for large inputs!

Key Takeaways

- Algorithms solve problems step-by-step and optimize efficiency

- Big O notation measures growth rate, not exact execution time

- Choosing the right algorithm prevents inefficiency in large-scale problems

Think efficiency:

better algorithms save

time and resources!

Helpful Resources on Algorithms & Complexity

- 📖 Big-O Cheat Sheet: Quick reference for complexities

- 🎥 CS50 Lecture on Algorithms (Harvard): Beginner-friendly explanation

- 📚 VisuAlgo: Interactive algorithm visualizations

- 📝 GeeksforGeeks Data Structures & Algorithms
In-depth explanations and examples

- 💻 HackerRank Algorithms Challenges
Practice problems for learning

Practice

Devise a pseudocode solution and analyze its time complexity

- Given a list of people's heights, determine the tallest person

- Given a text message and a word, check if the word appears in the message

- Given a list of food item prices, calculate the total cost

Practice

- Given a list of gas stations along a highway with distances from the starting point, find the closest one with fuel available

- Given a list of votes (each representing a candidate's name), determine how many votes each candidate received

- Given a list of packages with estimated delivery times, sort them so that they are delivered in the correct order

Quote of the Week

METROPOLITAN TIRANA UNIVERSITY

A slow algorithm isn’t wrong, it’s just waiting for a faster computer to be invented.