Algorithm Design and Analysis

A Day in the Life of a MERN Stack Developer

You are a MERN stack developer at XYZ Corp, tasked with improving a job portal that connects candidates to potential employers. The portal processes vast amounts of data from resumes to job descriptions.

Your challenge is to optimize data handling, making the platform faster and more efficient.

To elevate the user experience, you decide to refine the algorithms that handle data storage and retrieval. This includes selecting the right sorting algorithms for job listings and designing efficient search algorithms for matching candidates to jobs.

In this lesson, you will learn a few concepts that will help you to find a solution to the above scenario.

Learning Objectives

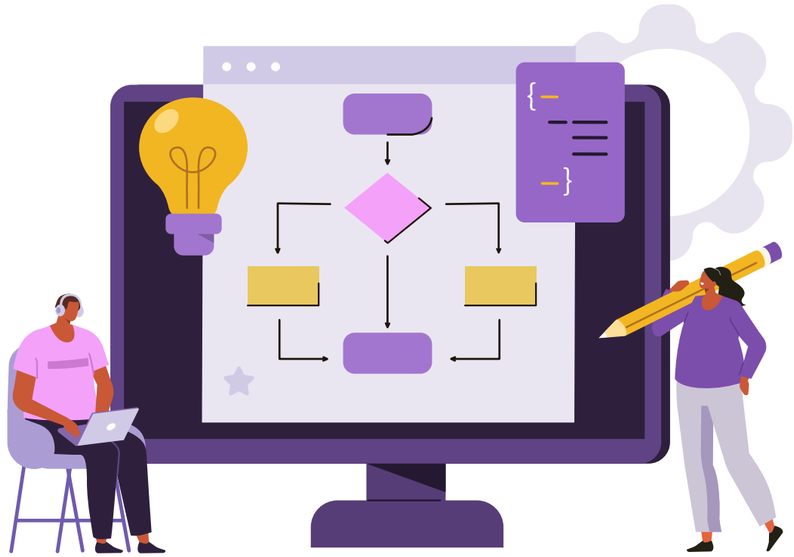
By the end of this lesson, you will be able to:

- Analyze algorithm efficiency using Big O notation and complexity measures for performance optimization
- Master the divide and conquer strategy for solving complex problems efficiently
- Apply algorithms for effective computational problem resolution and logic development
- Integrate algorithms with data structures for more efficient data handling and processing
- Explore design principles for creating structured and efficient algorithms for robust applications

Introduction to Algorithms

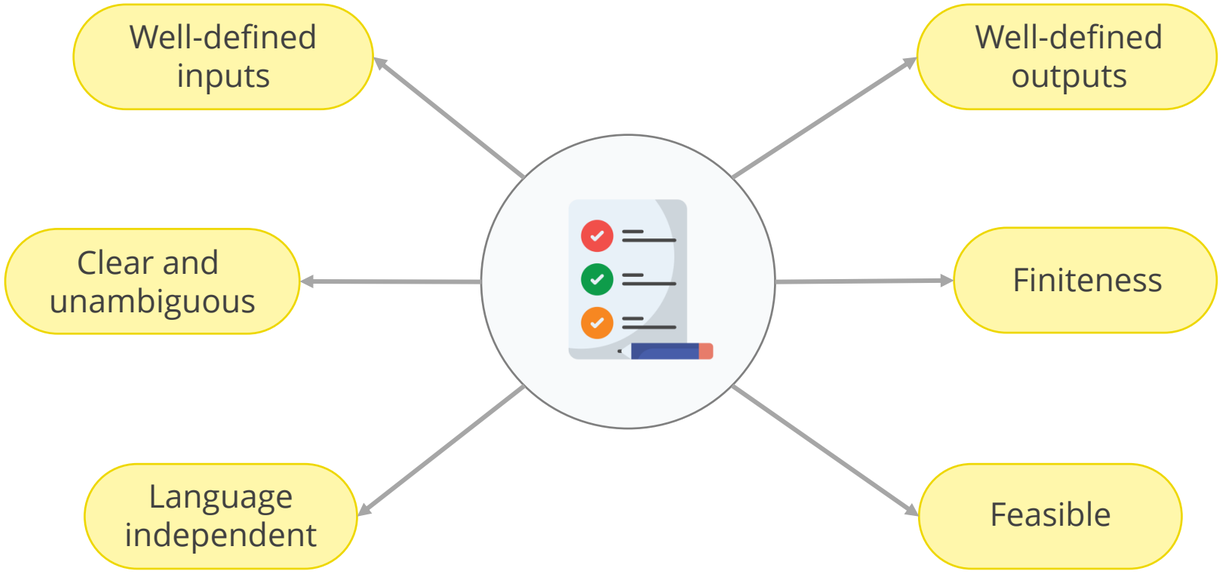
What Is an Algorithm?

It is a set of well-defined instructions or a step-by-step procedure designed to perform a specific task or solve a particular problem.



Characteristics of Algorithms

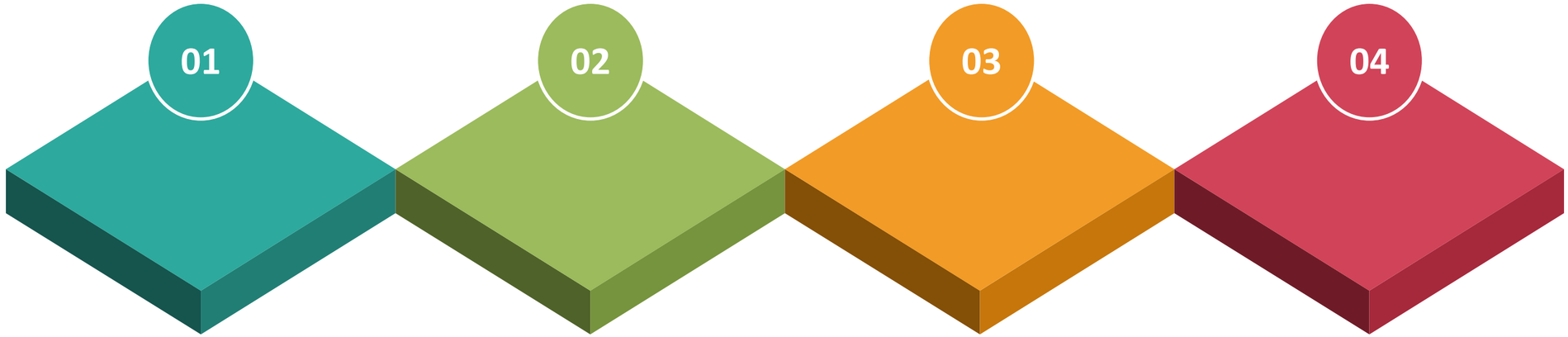
It is fundamental qualities that define their functionality and effectiveness.
Here are the key characteristics:



JavaScript algorithms integrate standard algorithmic design principles, adapted to align with the distinct characteristics and application contexts of the JavaScript language.

Importance of an Algorithm

It plays a critical role in various aspects of computing and problem-solving. Their importance can be highlighted in several key areas:



Problem-solving

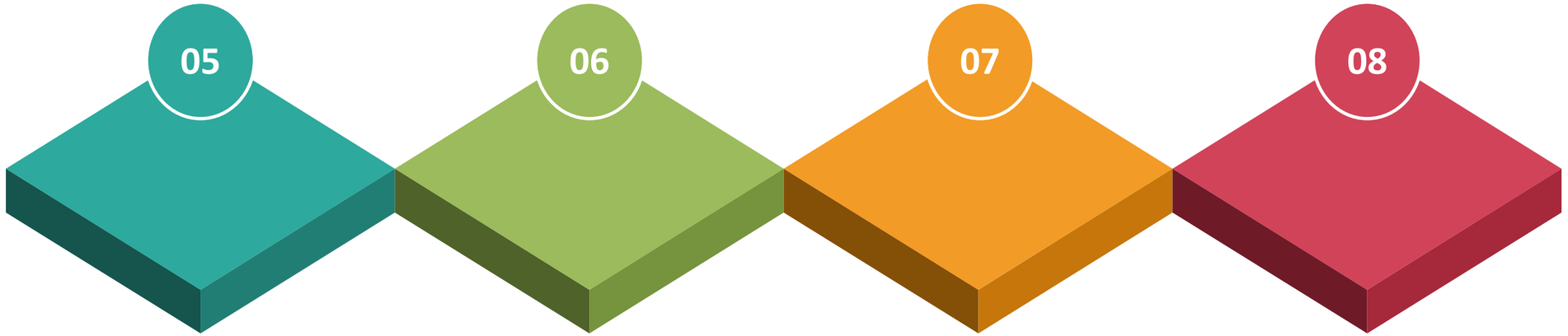
Efficiency and performance

Consistency and reliability

Automation

Importance of an Algorithm

It plays a critical role in various aspects of computing and problem-solving. Their importance can be highlighted in several key areas:



Data process

Foundation for software development

Technological innovation

Scalability

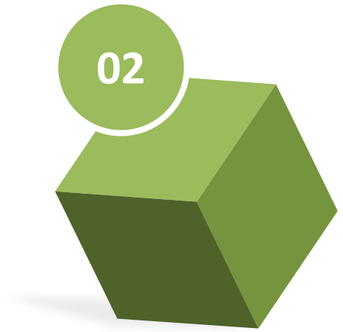
Algorithms are crucial in the digital domain, forming the core components of computer science, information technology, and the progress of digital advancements.

Representation of Algorithms

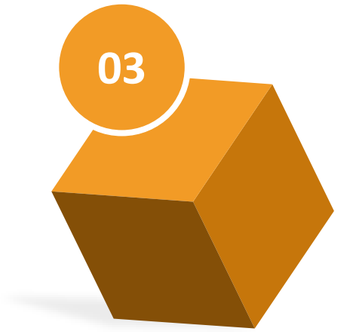
Pseudocode: Describe Algorithms

Pseudocode is a key tool in software development that plays several important roles.

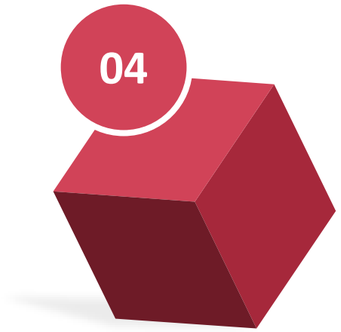
Structure and clarity



Problem-solving focus



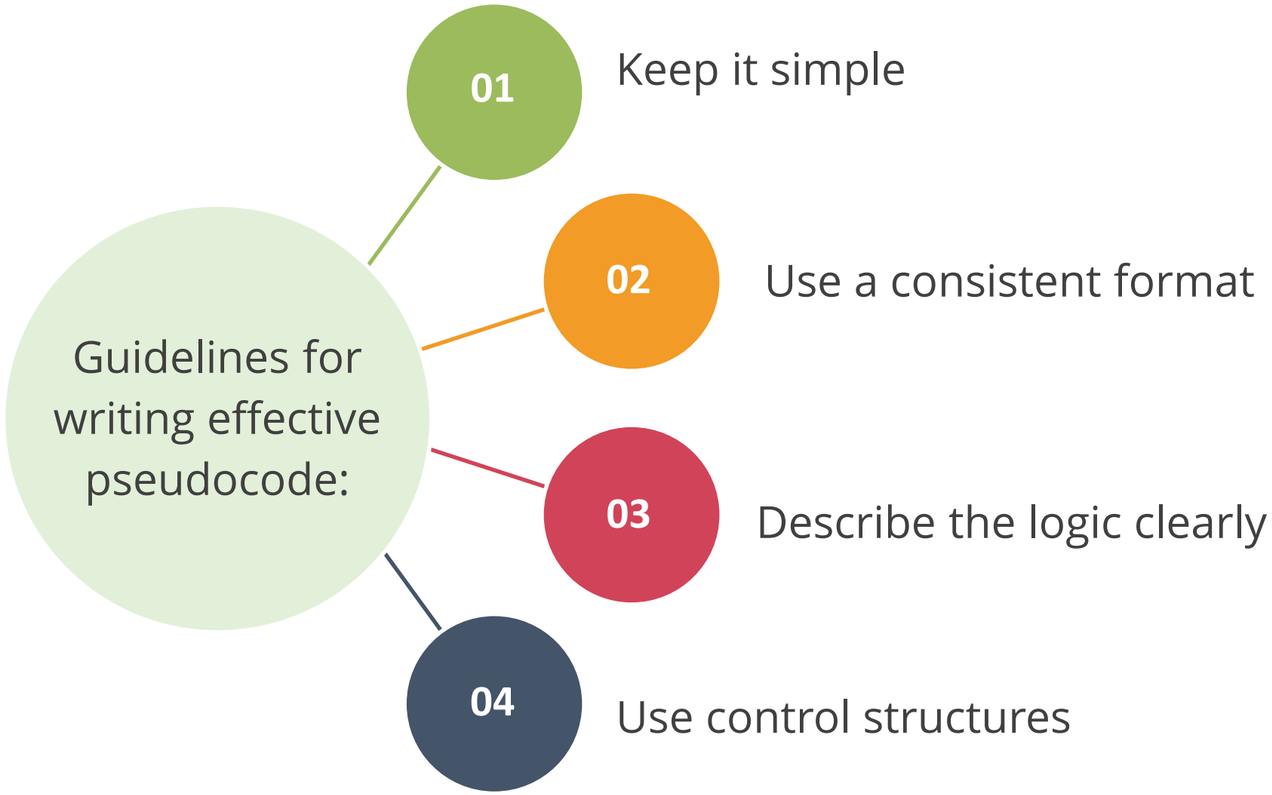
Communication tool



Planning and design

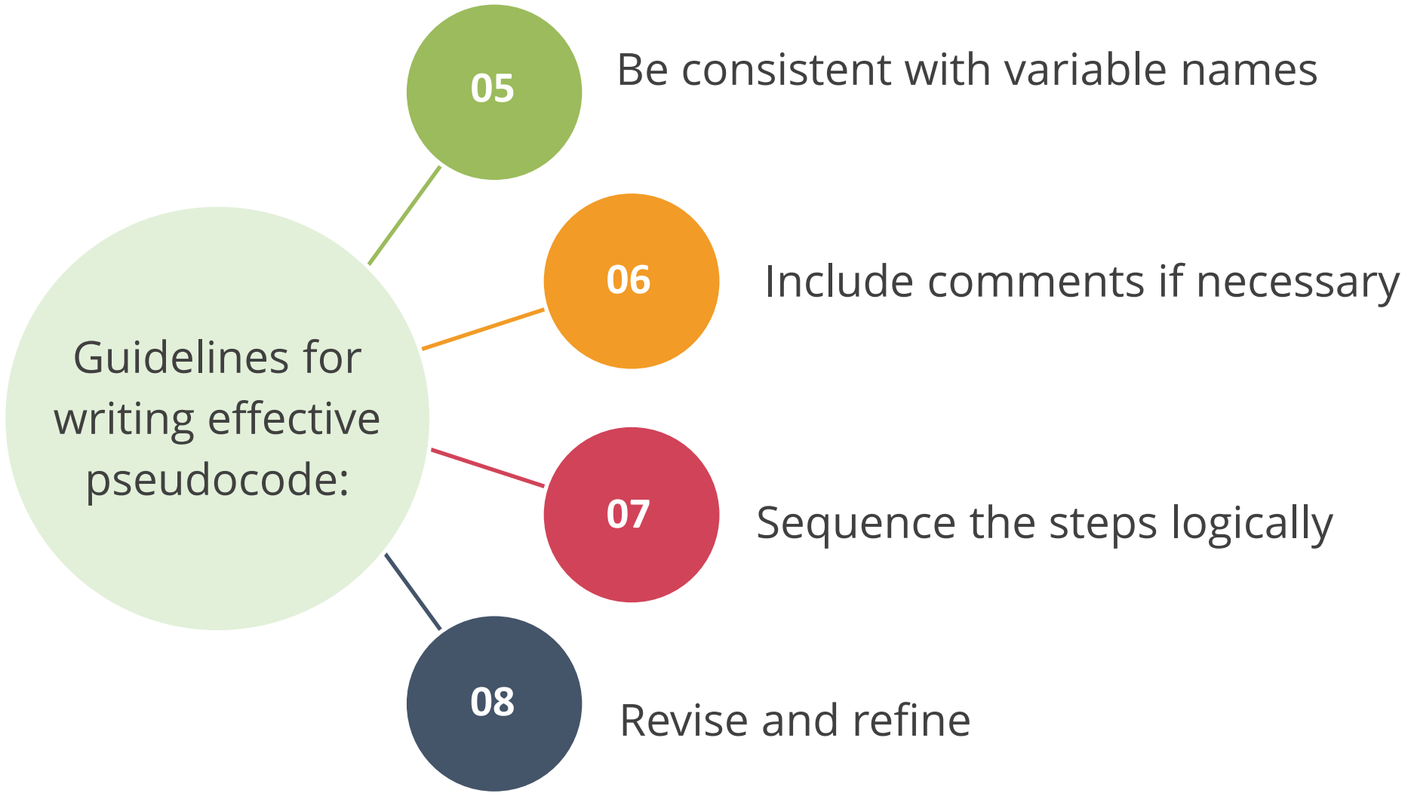
Rules for Writing a Pseudocode

It is essential for planning a program's logic; it should be simple and logical, avoiding a complex syntax.



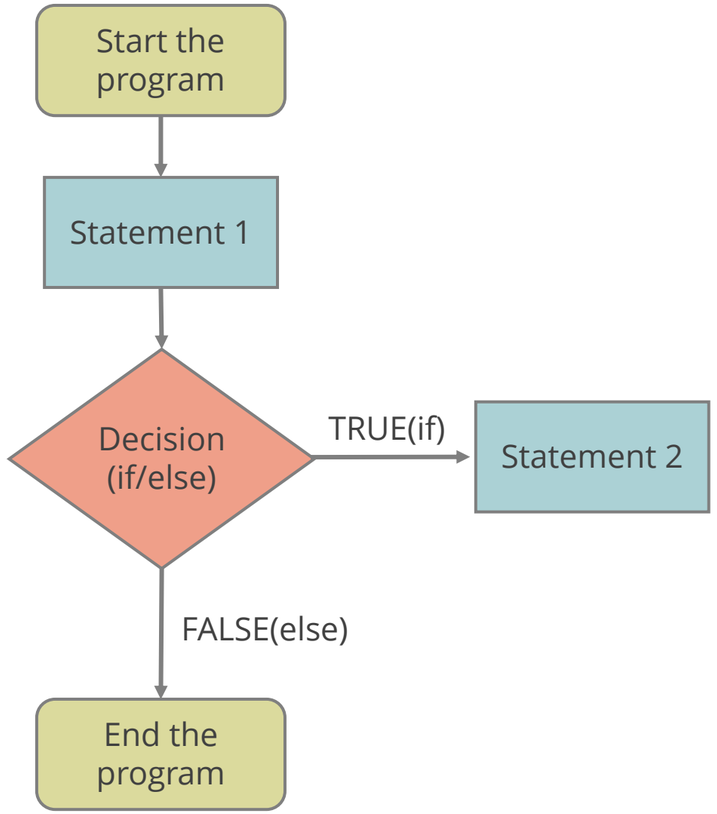
Rules for Writing a Pseudocode

It is essential for planning a program's logic; it should be simple and logical, avoiding a complex syntax.



Representation of Algorithm

Representing an algorithm in a flowchart is a visual way to depict the steps a program performs to achieve the desired output.



Algorithm Analysis and Design Techniques

Analyzing the Efficiency of an Algorithm

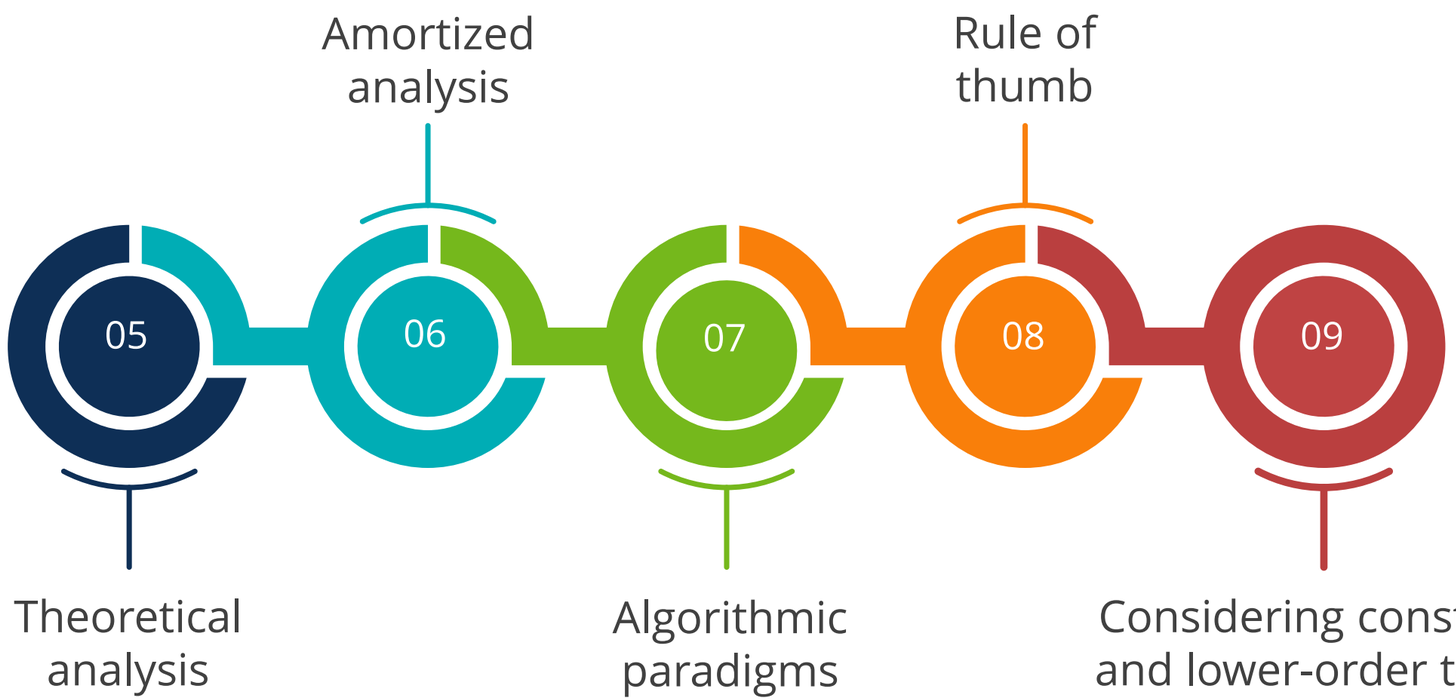
It typically involves understanding the time and space complexity, which is crucial for evaluating the performance.

Here are the key aspects involved in this analysis:

Analyzing the Efficiency of an Algorithm

It typically involves understanding the time and space complexity, which is crucial for evaluating the performance.

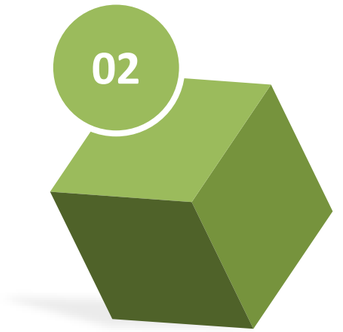
Here are the key aspects involved in this analysis:



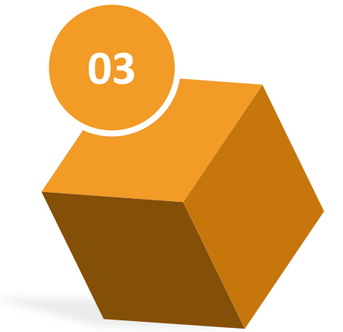
Time Complexity: Measuring Algorithm Efficiency

It measures an algorithm's completion time based on input length, estimating its efficiency as input size increases.

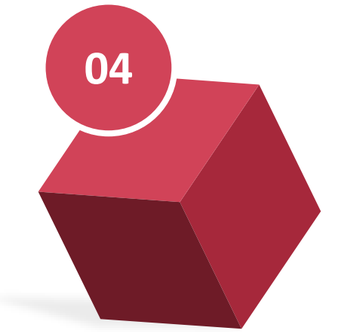
Growth rate



Nested loops and operations



Divide and conquer

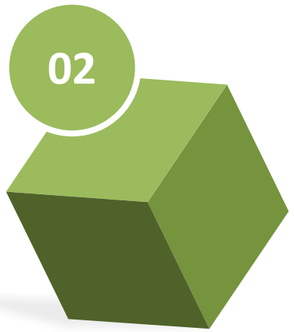


Logarithmic time

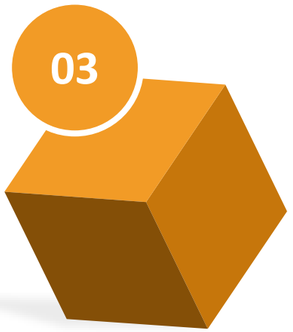
Space Complexity: Assessing Memory Usage

It denotes the memory requirement of an algorithm for its execution, evaluating the overall memory usage in correlation with the size of the input.

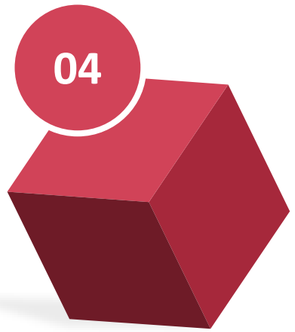
Input and auxiliary space



Recursive algorithms



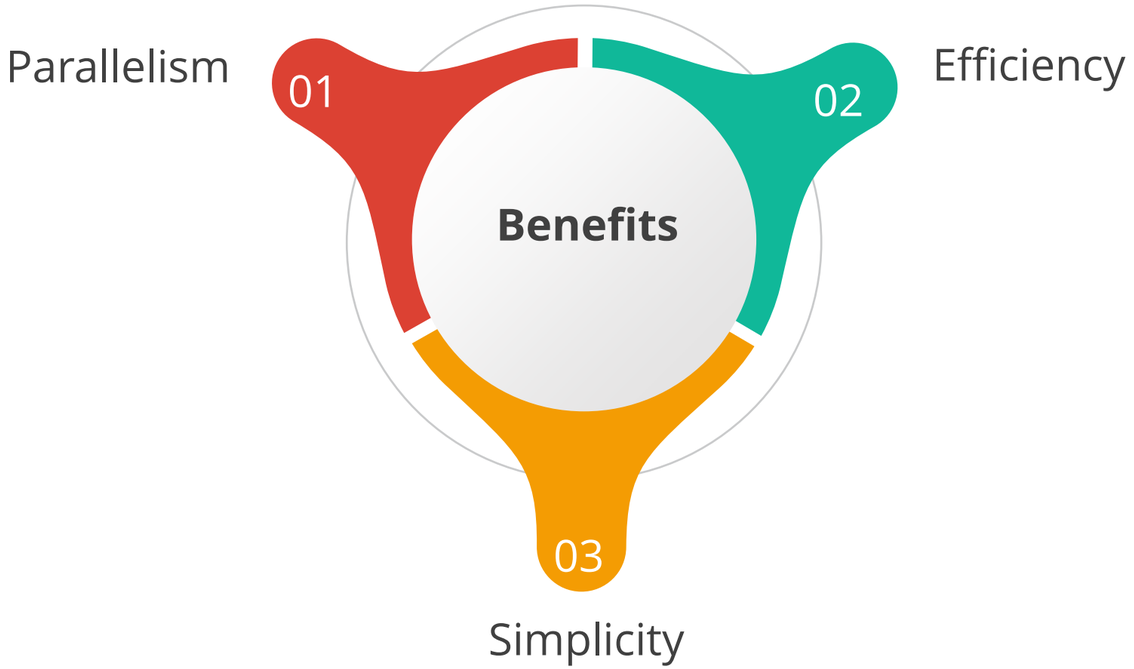
Static and dynamic memory



Amortized space complexity

Divide and Conquer

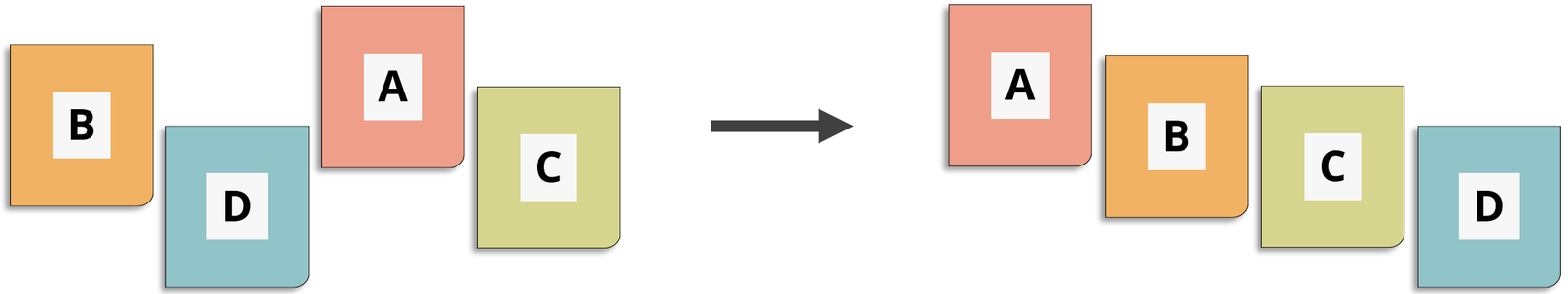
It is a method that breaks down a problem into smaller, similar sub-problems, solves them individually, and then combines these solutions.



Sorting Algorithms

Sorting Algorithms

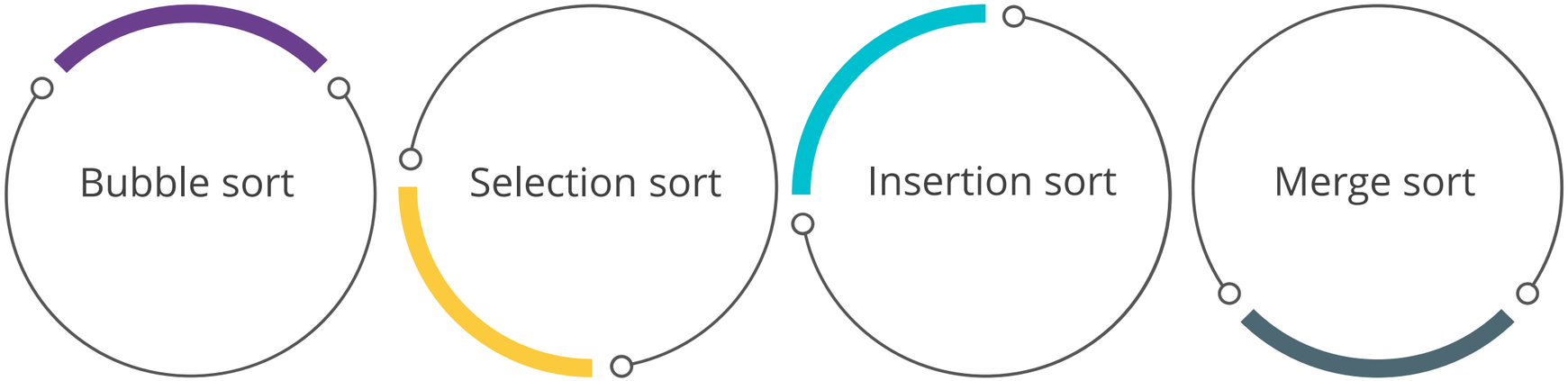
These are the methods used to order the elements of a list in a particular sequence (most commonly in ascending or descending order).



The importance of sorting lies in the fact that it makes data searching easier and more efficient.

Types of Sorting Algorithm

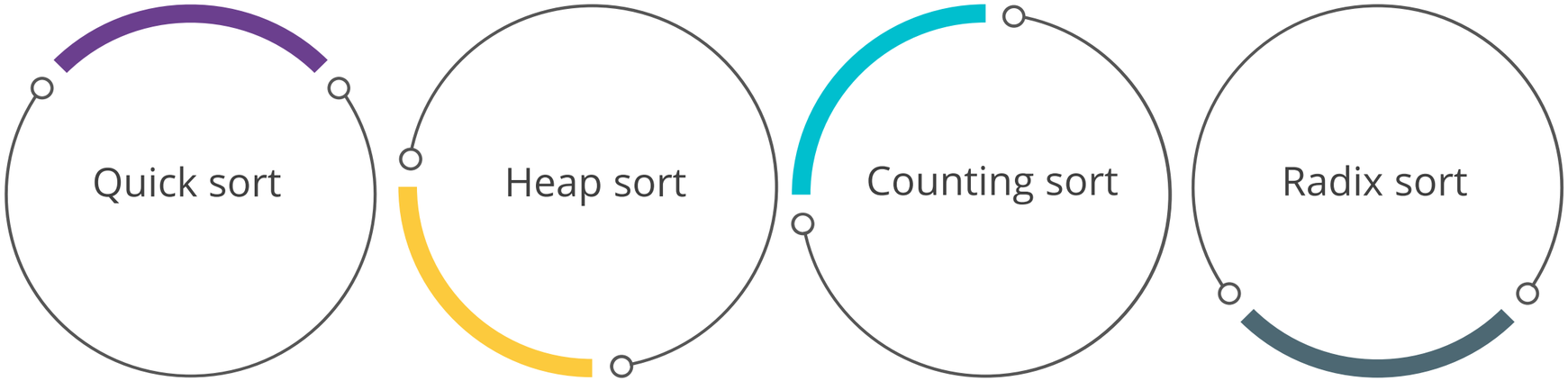
There are various sorting algorithms, each with its own mechanics and efficiency. Here are some commonly used ones:



The choice of sorting algorithm depends on the size and nature of the data, the computational complexity requirements, and specific constraints like memory usage.

Types of Sorting Algorithm

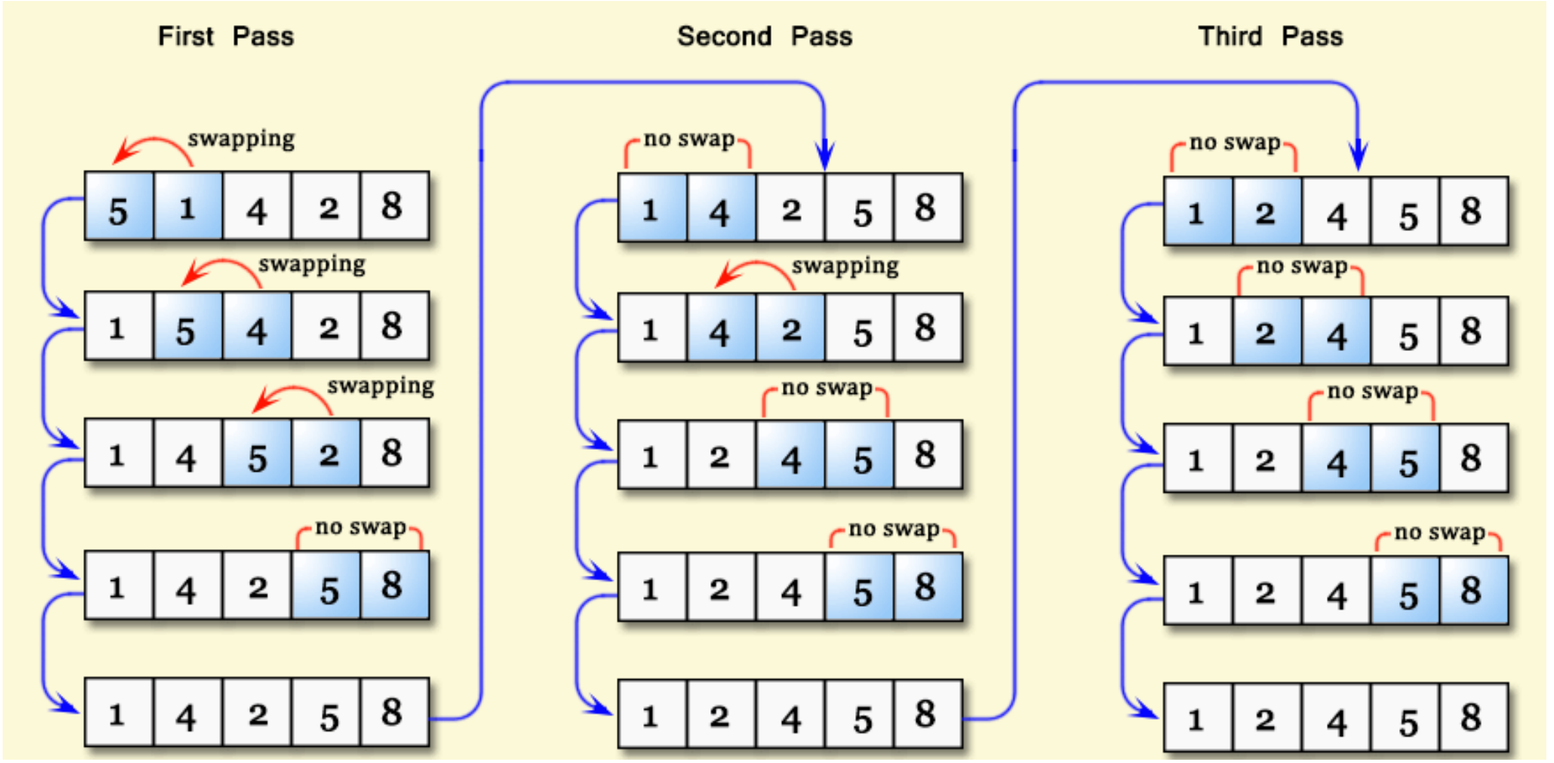
There are various sorting algorithms, each with its own mechanics and efficiency. Here are some commonly used ones:



Certain algorithms are more efficient for small lists, while others are better for large lists and maintain the relative order of equal elements.

Bubble Sort

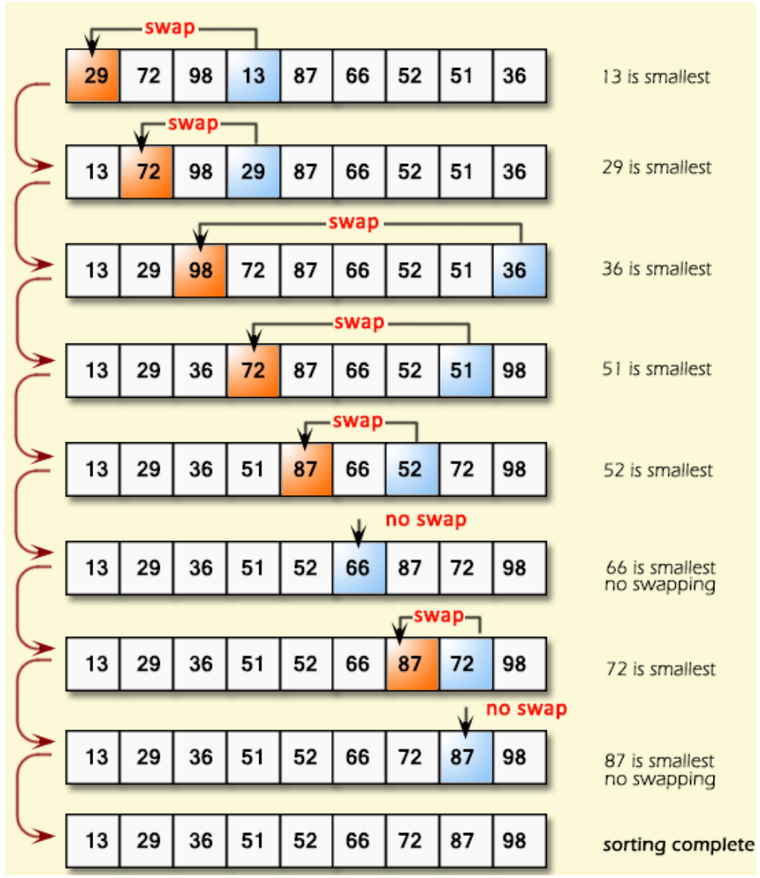
This sorting algorithm continuously compares and swaps adjacent elements, moving smaller ones to the array's start in each pass.



Source: https://www.w3resource.com/javascript-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-7.php

Selection sort

This algorithm sorts elements by repeatedly finding and moving the minimum (or maximum) element from the unsorted to the sorted section.



Source: https://www.w3resource.com/javascript-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-5.php

Assisted Practice

Implementing Bubble Sort Algorithm **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the bubble sort algorithm and explain its time and space complexity using JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Assisted Practice

Implementing Selection Sort Algorithm **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the selection sort algorithm and explain its time and space complexity using JavaScript.

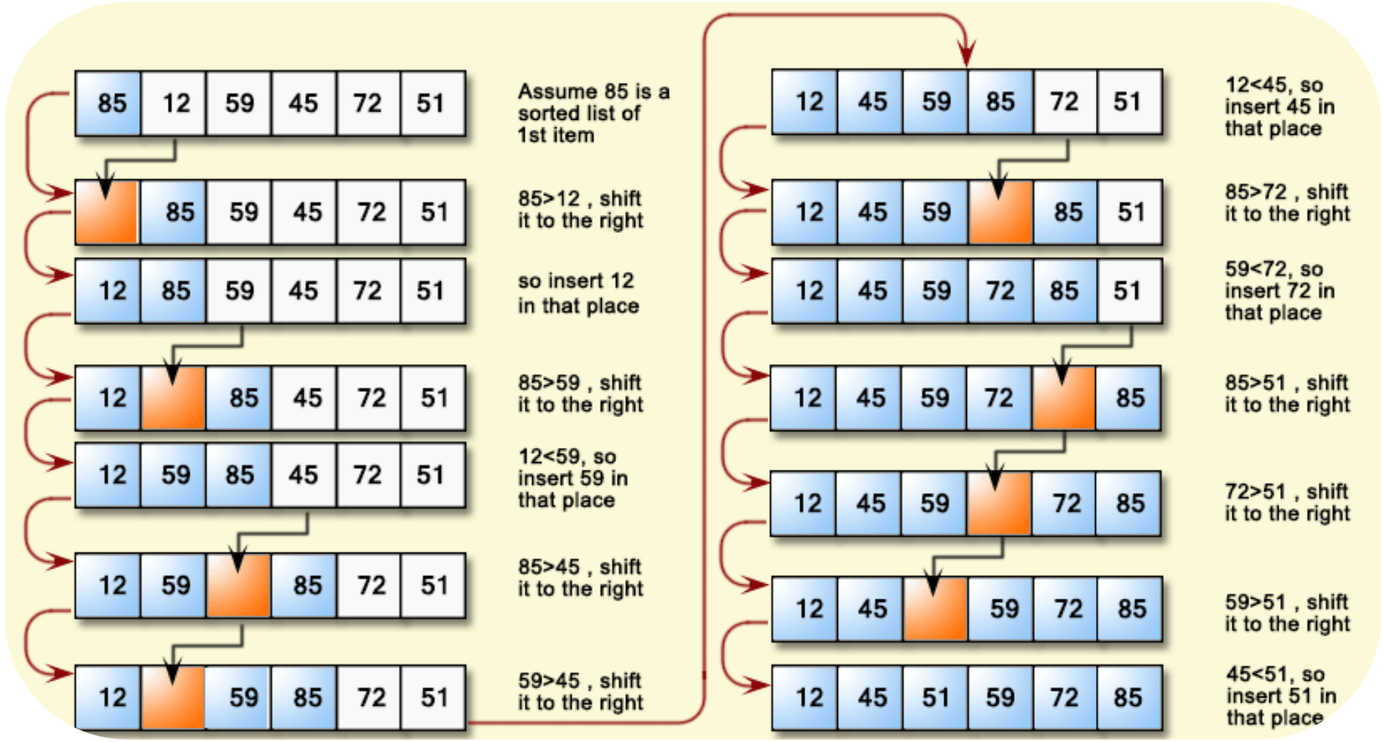
Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Insertion sort

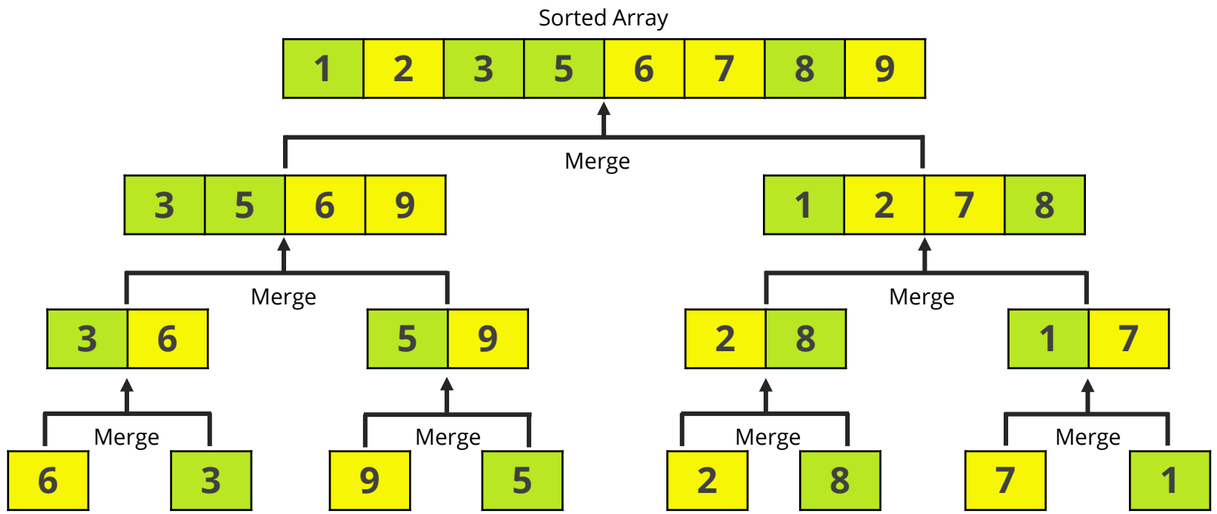
It is a simple and intuitive sorting algorithm that builds the final sorted array (or list) one item at a time.



Source: https://www.w3resource.com/javascript-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-4.php

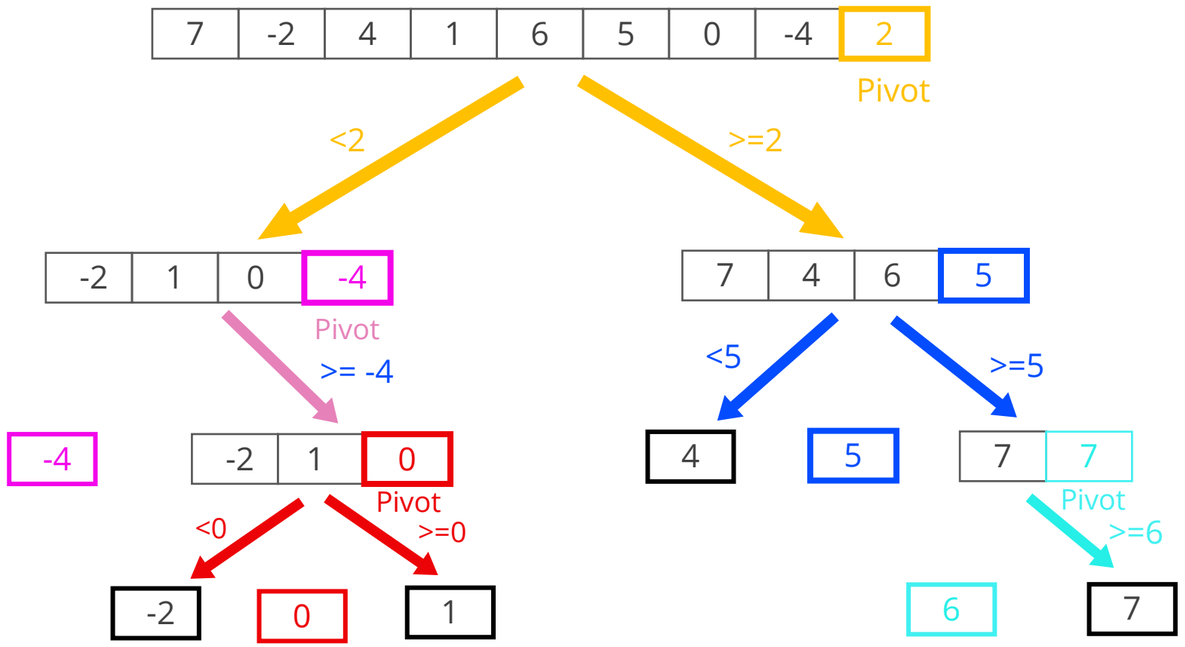
Merge sort

It is a dependable and effective sorting technique, notable for its steady efficiency and use of the divide and conquer method in its design.



Quick sort

It is an efficient sorting algorithm well-known for effectively handling large datasets, utilizing a divide and conquer strategy for optimal average-case performance.



Assisted Practice

Implementing Insertion Sort Algorithm **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the insertion sort algorithm and explain its time and space complexity using JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Assisted Practice

Implementing Merge Sort Algorithm **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the merge sort algorithm and explain its time and space complexity using JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Assisted Practice

Implementing Quick Sort Algorithm **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the quick sort algorithm and explain its time and space complexity using JavaScript.

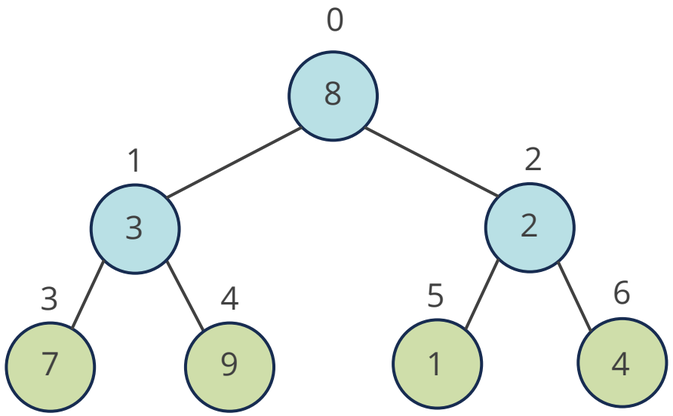
Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Heap sort

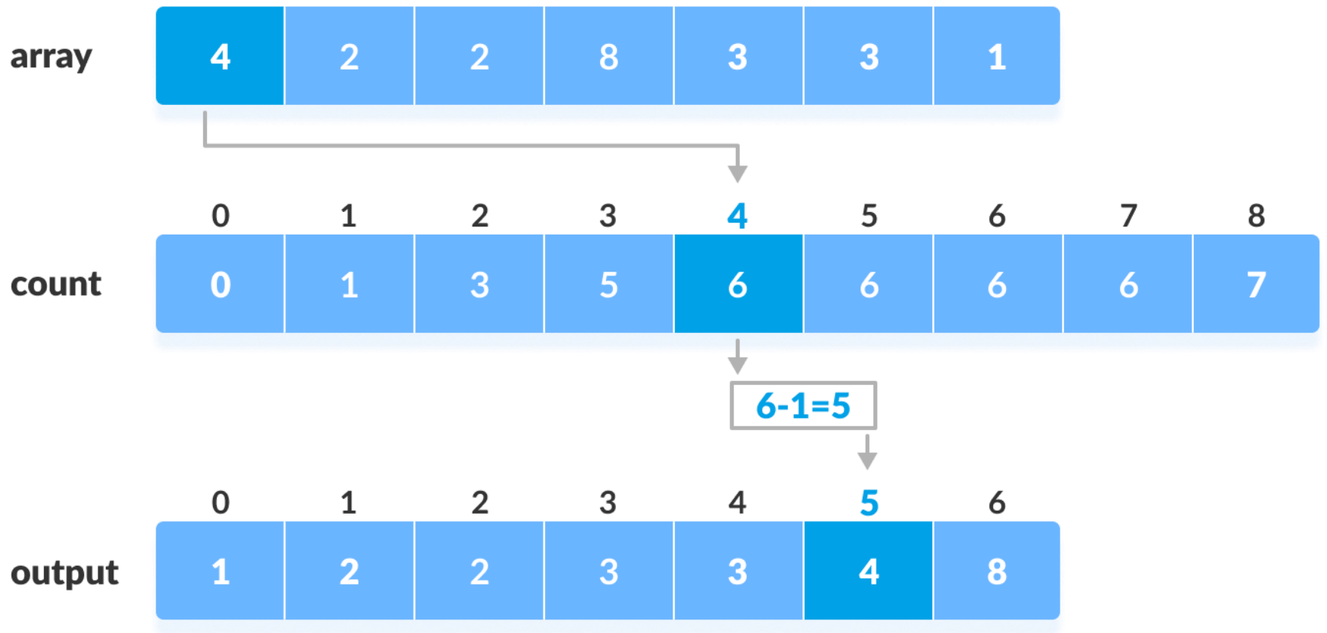
It is a comparison-based sorting algorithm that uses a binary heap data structure to organize elements.



8	3	2	7	9	1	4
0	1	2	3	4	5	6

Counting sort

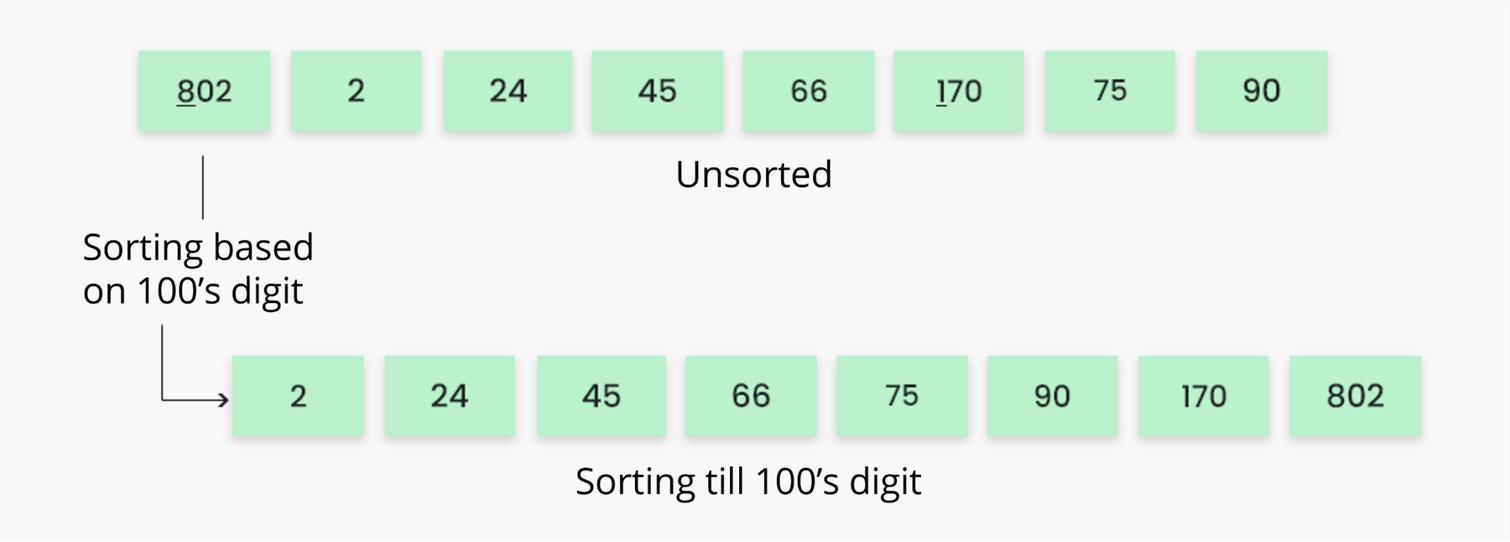
It operates by counting the number of objects that possess distinct key values, then doing arithmetic to calculate the positions of each key in the output sequence.



Source: https://www.programiz.com/dsa/counting-sort

Radix Sort

It is non-comparison-based sorting algorithm that sorts data with integer keys by grouping keys by the individual digits, which share the same significant position and value.



Source: https://www.geeksforgeeks.org/radix-sort/

Assisted Practice

Implementing Heap Sort Algorithm **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the heap sort algorithm and explain its time and space complexity using JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Assisted Practice

Implementing Count Sort Algorithm **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the count sort algorithm and explain its time and space complexity using JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Assisted Practice

Implementing Radix Sort Algorithm **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the radix sort algorithm and explain its time and space complexity using JavaScript.

Assisted Practice: Guidelines

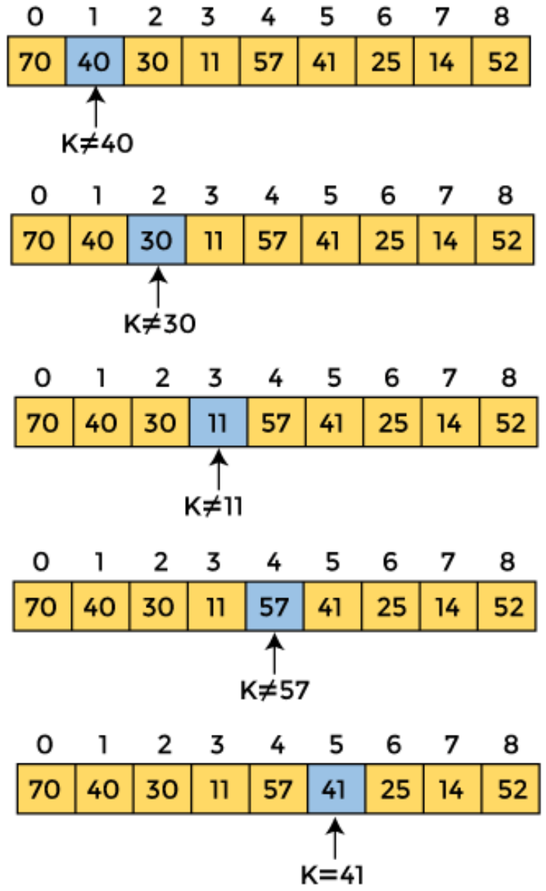
Steps to be followed:

1. Create and execute JS file

Searching Algorithms

Linear Search

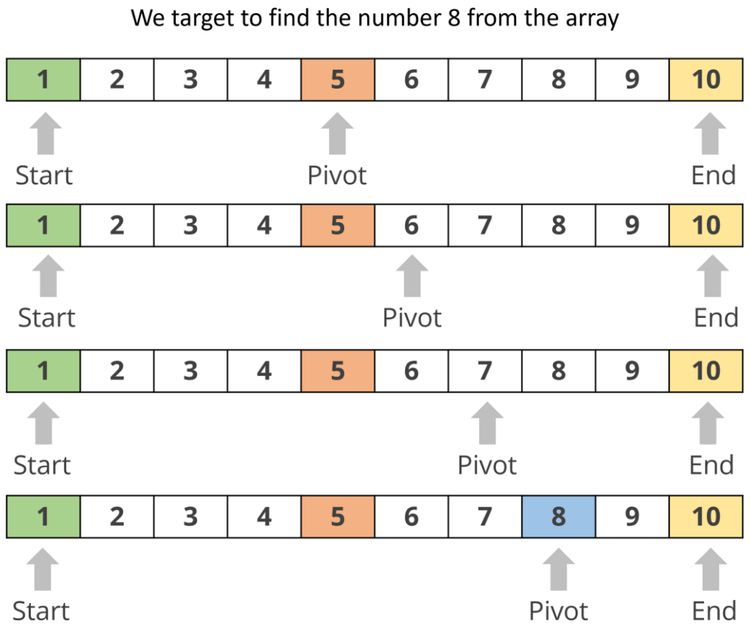
It is a method for finding a particular value in a list by checking each element in the list sequentially until the desired value is found or the list is completely searched.



Source: https://www.javatpoint.com/linear-search

Binary Search

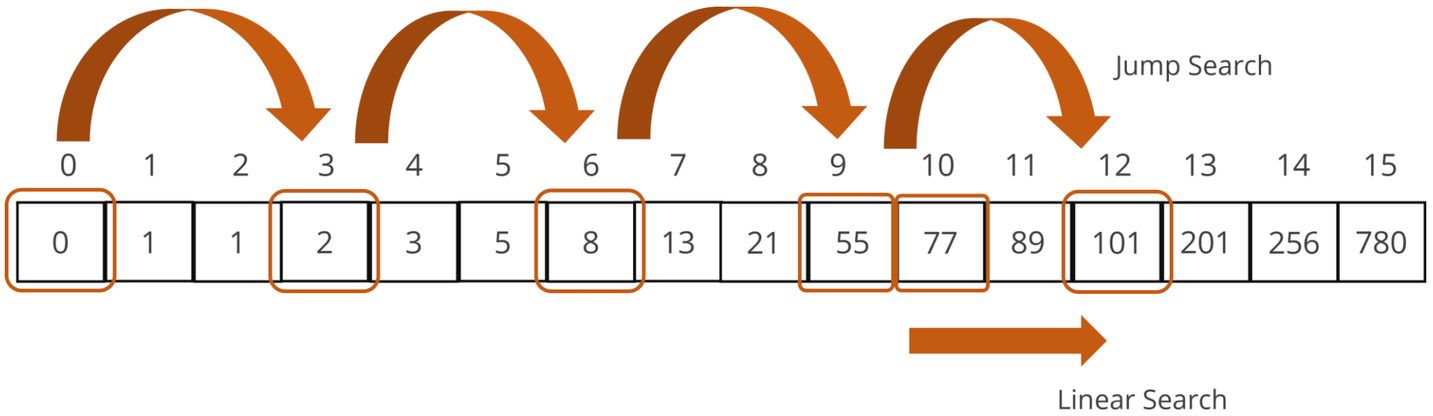
This algorithm finds an item by repeatedly dividing the array in half and checking if the element exists in the JavaScript array.



Start searching from the center and move right since the target number is greater than the middle value.

Jump Search

It efficiently locates an element in a sorted array by skipping set intervals and then performing a linear search within a narrowed down range.



The color boxes highlight the positions where the algorithm checks the elements.

Assisted Practice

Implementing Linear Search Algorithm **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the linear search algorithm and explain its time and space complexity using JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Assisted Practice

Implementing Binary Search Algorithm **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the binary search algorithm and explain its time and space complexity using JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Assisted Practice

Implementing Jump Search Algorithm **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the jump search algorithm and explain its time and space complexity using JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Key Takeaways

- Algorithms are crucial in computer science for solving problems; they are precise, have a clear end point, and are practical, offering specific solutions for established inputs and outputs.
- Pseudocode simplifies algorithm descriptions without specific syntax, while flowcharts visually map out the steps for easier understanding and troubleshooting.
- Sorting are crucial in computing, includes various algorithms like bubble, selection, insertion, merge, quick, heap, counting, and radix sort, each tailored for different situations.
- Bubble sort swaps neighboring elements, selection sort finds the minimum, and insertion sort gradually builds a sorted list.

Thank you