

Data Structures and Algorithms



Algorithm Design and Analysis



Engage and Think



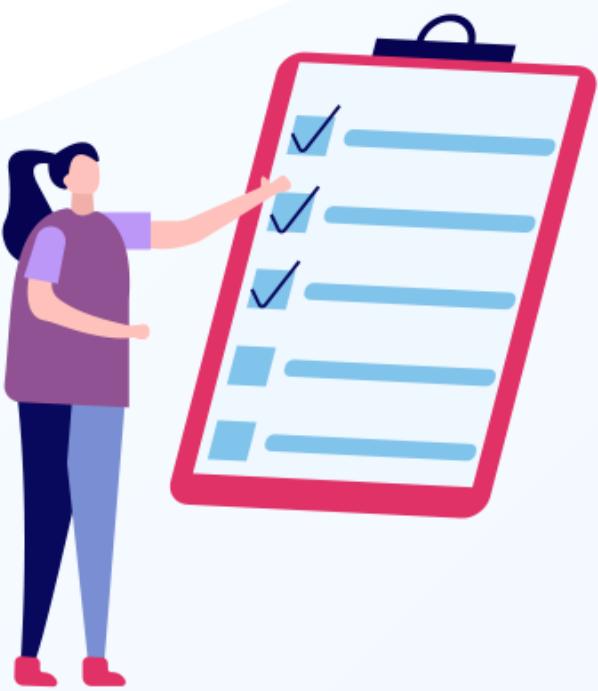
When you shop on Amazon, the process feels almost effortless. You search for an item, and the results appear instantly. Then, you add a product to your cart, enter your delivery address, and complete the payment. The steps are always fast, accurate, following a clear, consistent order.

Have you ever wondered how Amazon decides which products to show first and how it displays results so quickly every time you search?

Learning Objectives

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

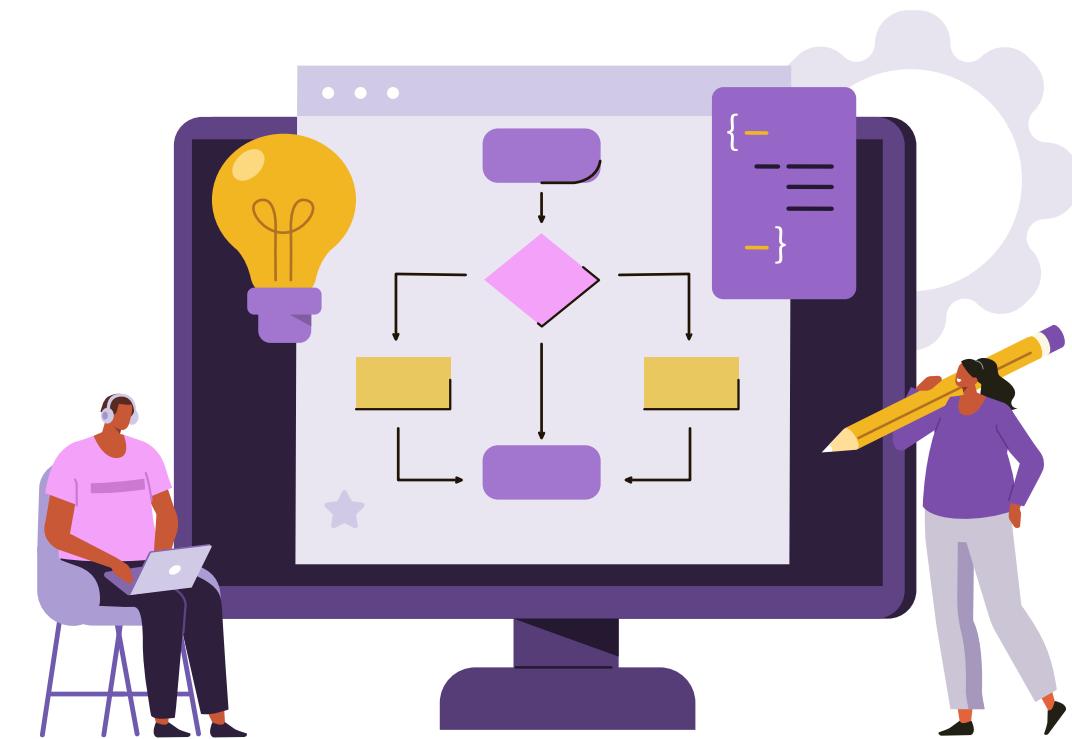
- Apply algorithms to solve structured computational problems
- Integrate algorithms with data structures to enhance processing efficiency
- Analyze algorithms using Big O to assess time and space performance
- Implement divide and conquer to handle complex problems efficiently
- Design efficient algorithms for scalable and high-performance applications



Introduction to Algorithms

What Is an Algorithm?

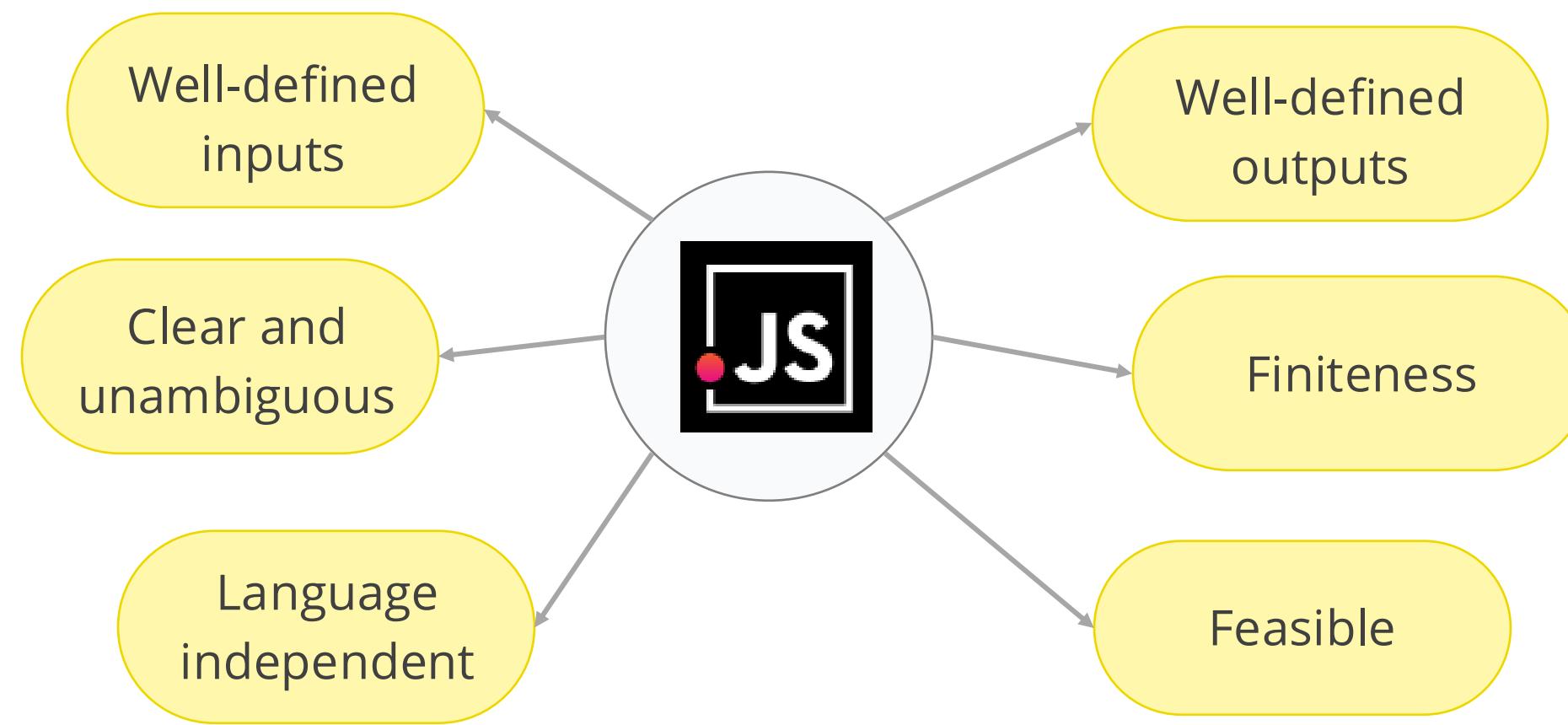
It is a set of well-defined instructions, or a step-by-step procedure designed to perform a specific task or solve a particular problem.



For example, *like a cooking recipe*, an algorithm provides step-by-step instructions to solve a problem.

Characteristics of Algorithms

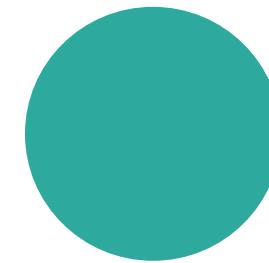
An algorithm's effectiveness is determined by its fundamental qualities. Here are the key characteristics that define a well-structured algorithm:



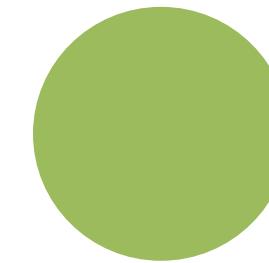
Algorithms follow standard design principles and are adapted to suit different programming languages and application contexts, ensuring efficiency and scalability.

Importance of an Algorithm

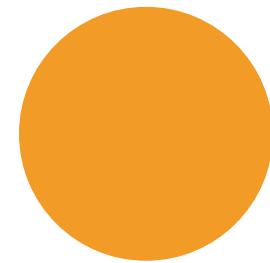
It plays a critical role in various aspects of computing and problem-solving.



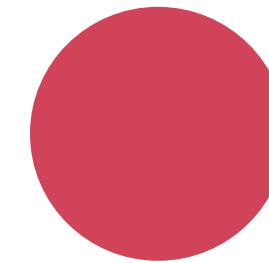
Solving problems
effectively



Improving efficiency
and performance



Ensuring consistency
and reliability



Automating complex
tasks

Algorithms play a vital role in computing and technology. To ensure accuracy and efficiency, they must be **represented in a structured format** before implementation.

Quick Check



A software engineer is designing an automated system to calculate monthly utility bills for customers. The program follows a defined sequence of steps to process user data, compute charges, and generate a final bill. However, during testing, the engineer notices that the program does not stop and keeps running indefinitely.

What fundamental algorithm characteristic is missing in the program?

- A. Precision
- B. Infinite steps
- C. Definitive outcome
- D. Finite steps

Representation of Algorithms

Representation of Algorithms

Algorithms must be represented in a structured way to ensure clarity, accuracy, and efficiency before implementation. The two main types include:

```
define function factorial(n)
if n==1 or n==0
return 1
else n * factorial(n - 1);

main function
num = 5;
Print "Factorial of "num" is",
call function factorial(num)
```

Pseudocode

Uses structured, human-readable statements to outline the steps of an algorithm logically

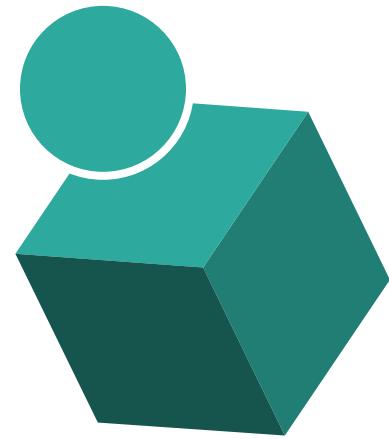


Flowcharts

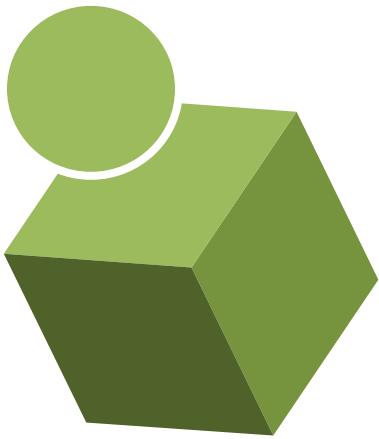
Uses graphical symbols to visually depict the sequence of operations and decision-making in an algorithm

Understanding Pseudocode

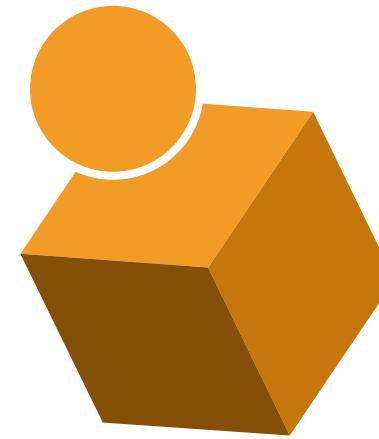
Pseudocode is a key tool in software development that plays several important roles. Its key benefits include:



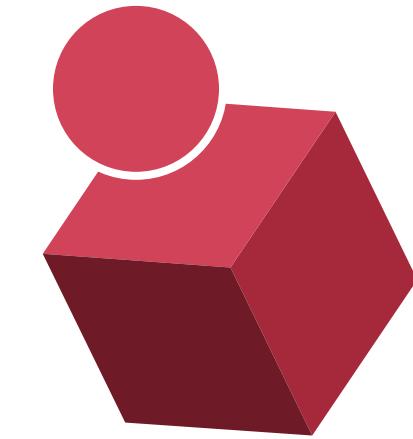
Providing structure
and clarity



Focusing on
problem-solving



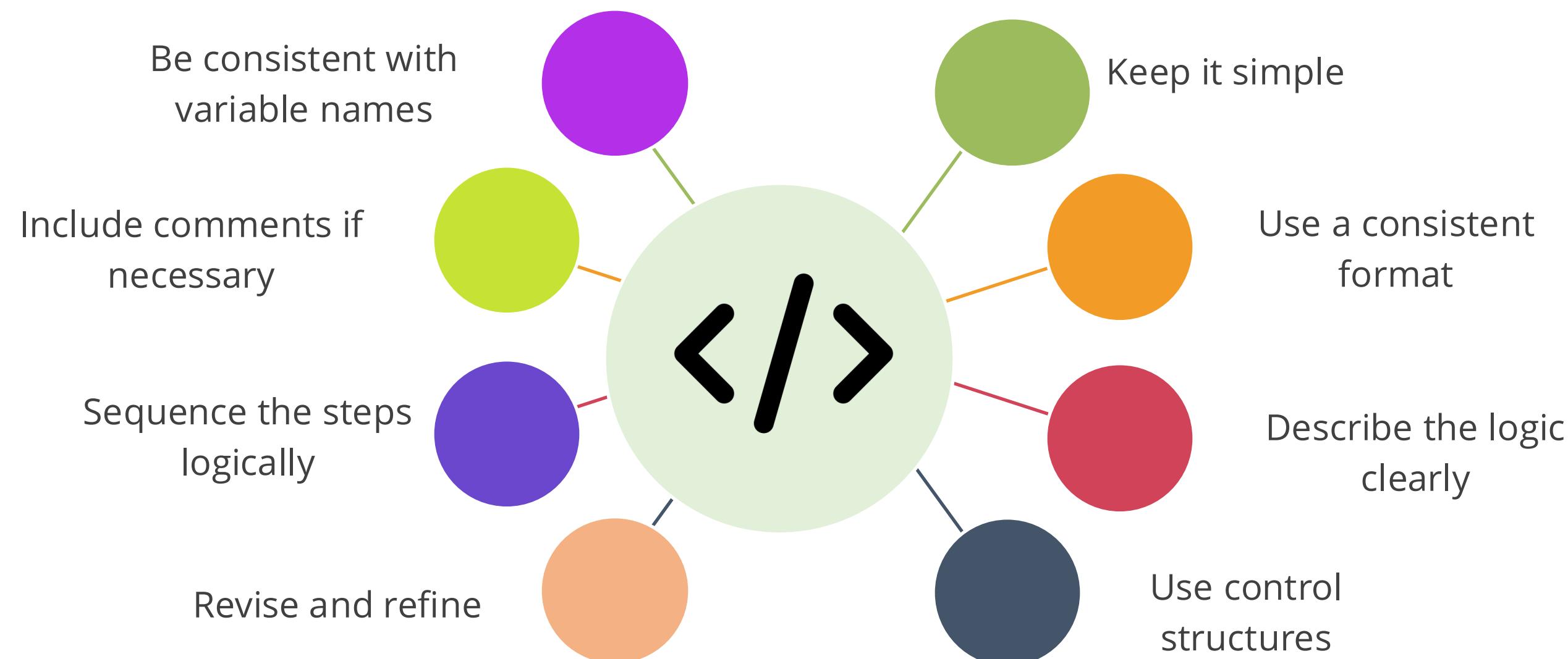
Enhancing
communication



Supporting planning
and design

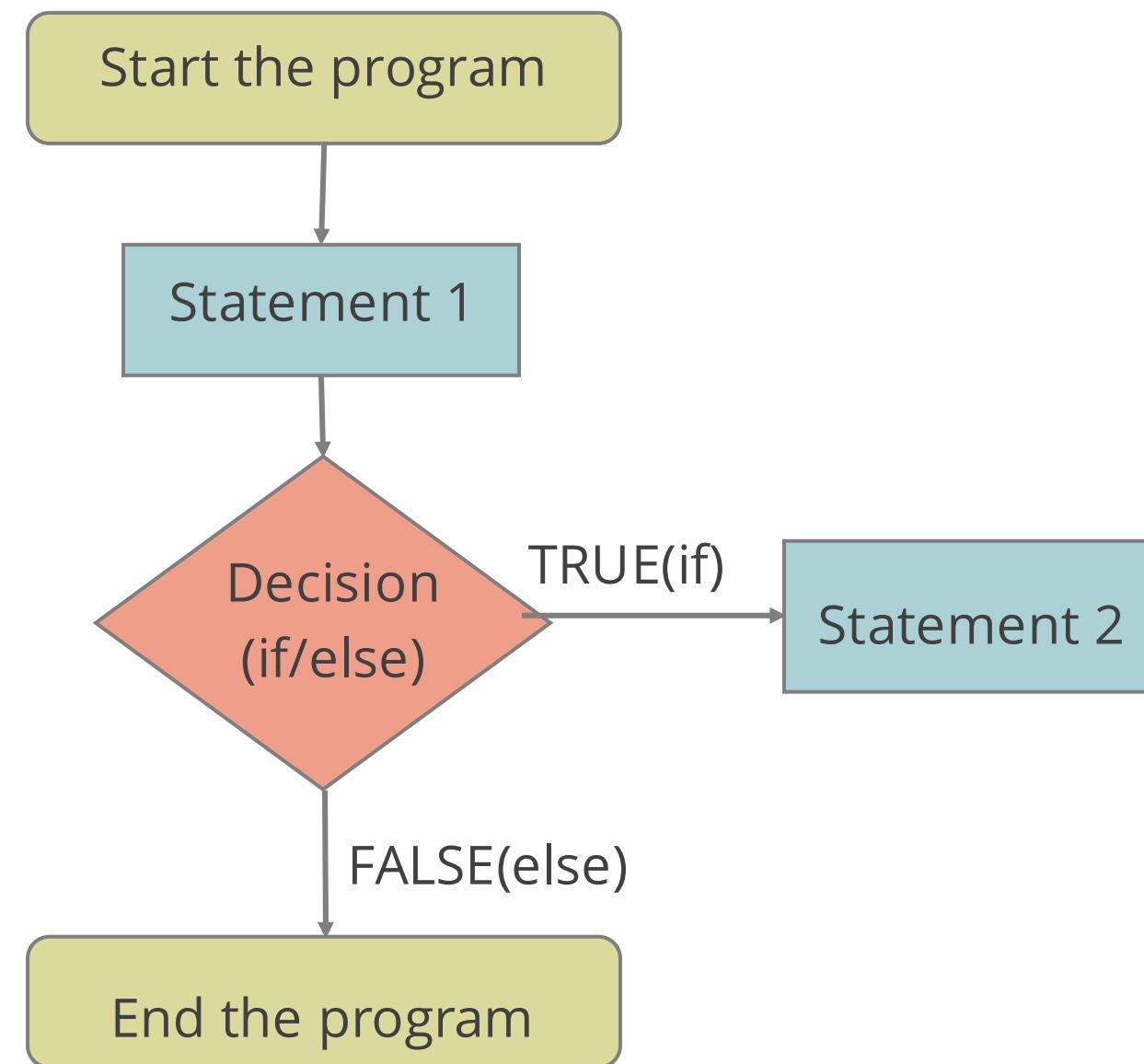
Rules for Writing a Pseudocode

Planning a program's logic should be simple and clear, avoiding complex syntax for easy understanding and implementation.



Understanding Flowcharts

Flowcharts visually represent algorithms, making it easier to understand the sequence of steps and decision-making in a process. The flow charts are visual representations of steps in shapes connected by arrows.



Rules for Making Flowcharts

Designing a clear and effective flowchart requires following structured guidelines to ensure accuracy, readability, and logical flow.

Use standard symbols

Follow industry-standard shapes for start, process, decision, and end

Maintain logical flow

Ensure steps follow a top-to-bottom or left-to-right direction for clarity

Keep It simple

Avoid unnecessary complexity; use only essential steps

Ensure proper connections

Use arrows to indicate clear progression between steps

Avoid crossed lines

Arrange elements to minimize intersecting paths for readability

Quick Check



A developer is designing a new software module and needs a clear way to represent the sequence of operations before writing the actual code. They want to create a structured diagram to illustrate the decision-making process and data flow within the algorithm.

What tool should the developer use to represent the steps of the program?

- A. Programming code
- B. Visual representation of an algorithm's flow
- C. Textual description of the algorithm
- D. Final output of the algorithm

Understanding Types of Algorithms

Types of Algorithms

Algorithms help solve problems efficiently in computing by following a step-by-step approach. The two types of algorithms are:



Sorting algorithms

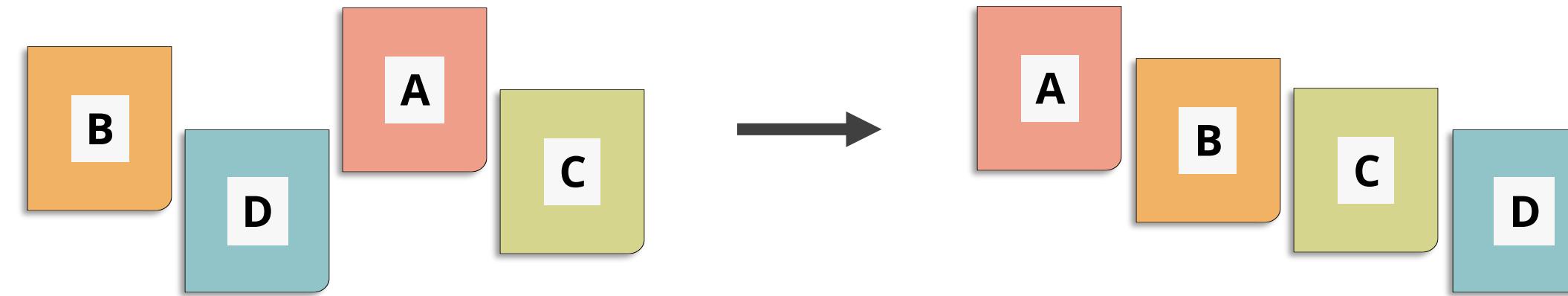


Searching algorithms

Sorting Algorithms

Sorting Algorithms

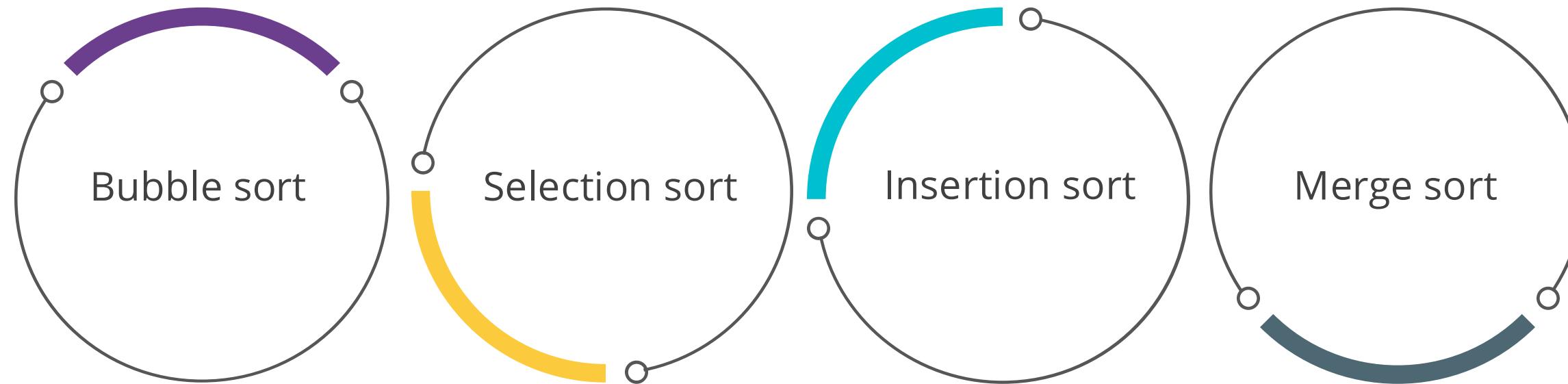
These are the methods used to order the elements of a list in a particular sequence (most commonly in ascending or descending order).



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

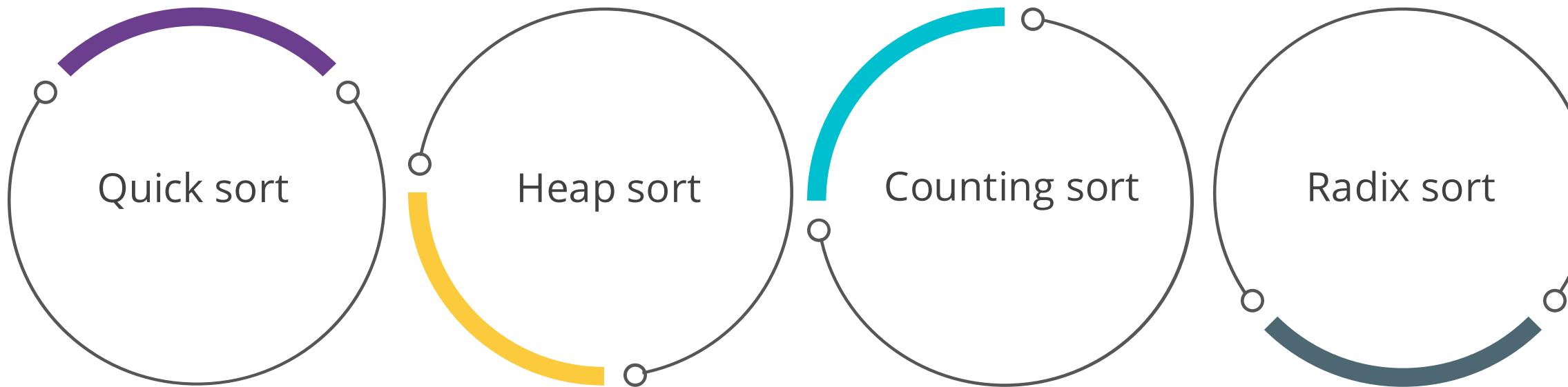
Types of Sorting Algorithms

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.

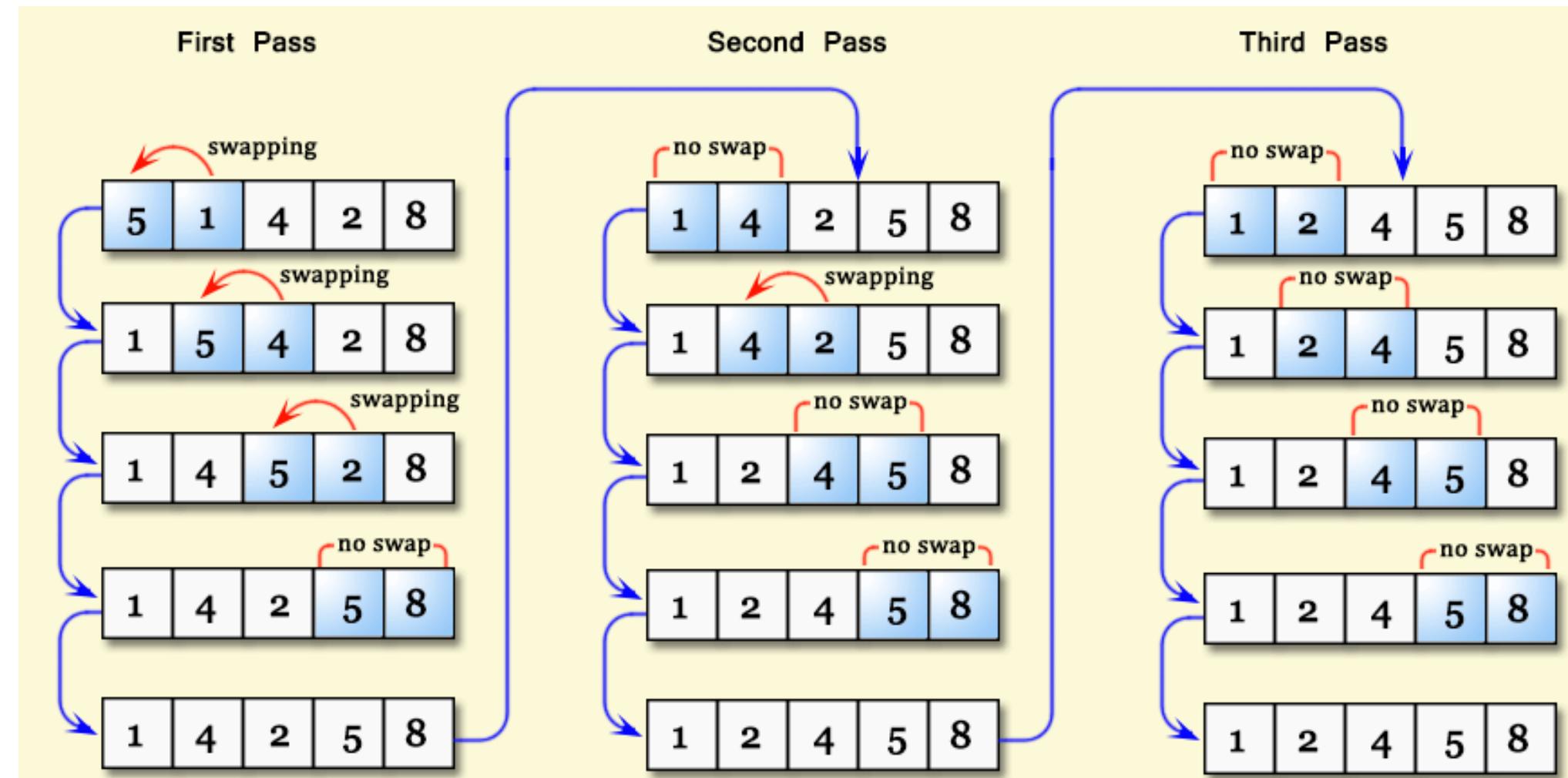
Types of Sorting Algorithms



Certain algorithms are more efficient for small lists, while others perform 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.



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.

Outcome:

By the end of this demo, you will be able to implement the bubble sort algorithm in JavaScript, sort an array, analyze its time and space complexity, and measure its execution time.

Note: Refer to the demo document for detailed steps:
[01_Implementing_Bubble_Sort_Algorithm](#)

ASSISTED PRACTICE

Assisted Practice: Guidelines

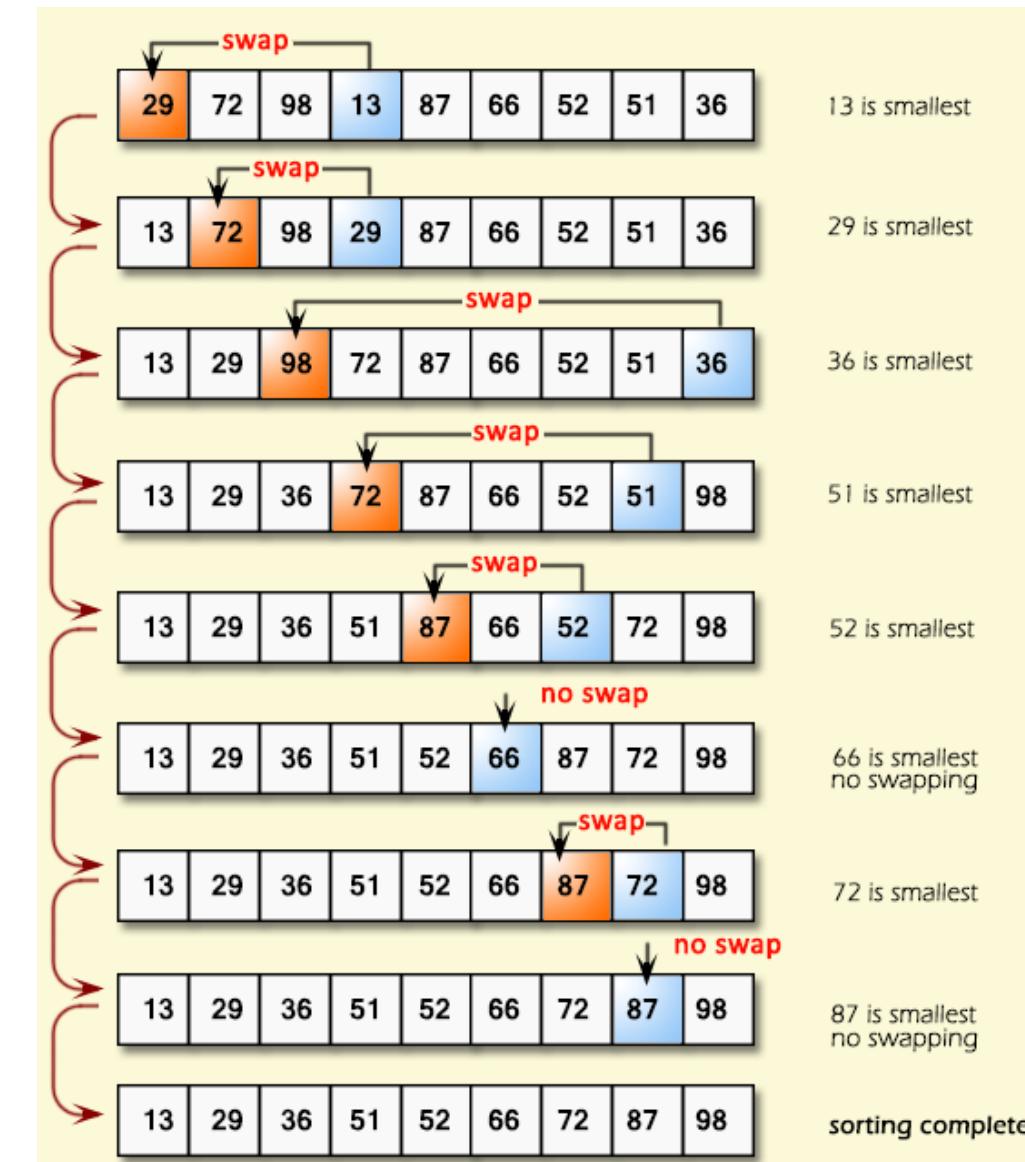


Steps to be followed:

1. Create and execute the JS file

Selection Sort

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



Assisted Practice



Problem Statement:

Duration: 15 Min.

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

Outcome:

By the end of this demo, you will be able to implement the selection sort algorithm in JavaScript, sort array elements efficiently, and evaluate its time and space complexity.

Note: Refer to the demo document for detailed steps:
[02_Implementing_Selection_Sort_Algorithm](#)

Assisted Practice: Guidelines

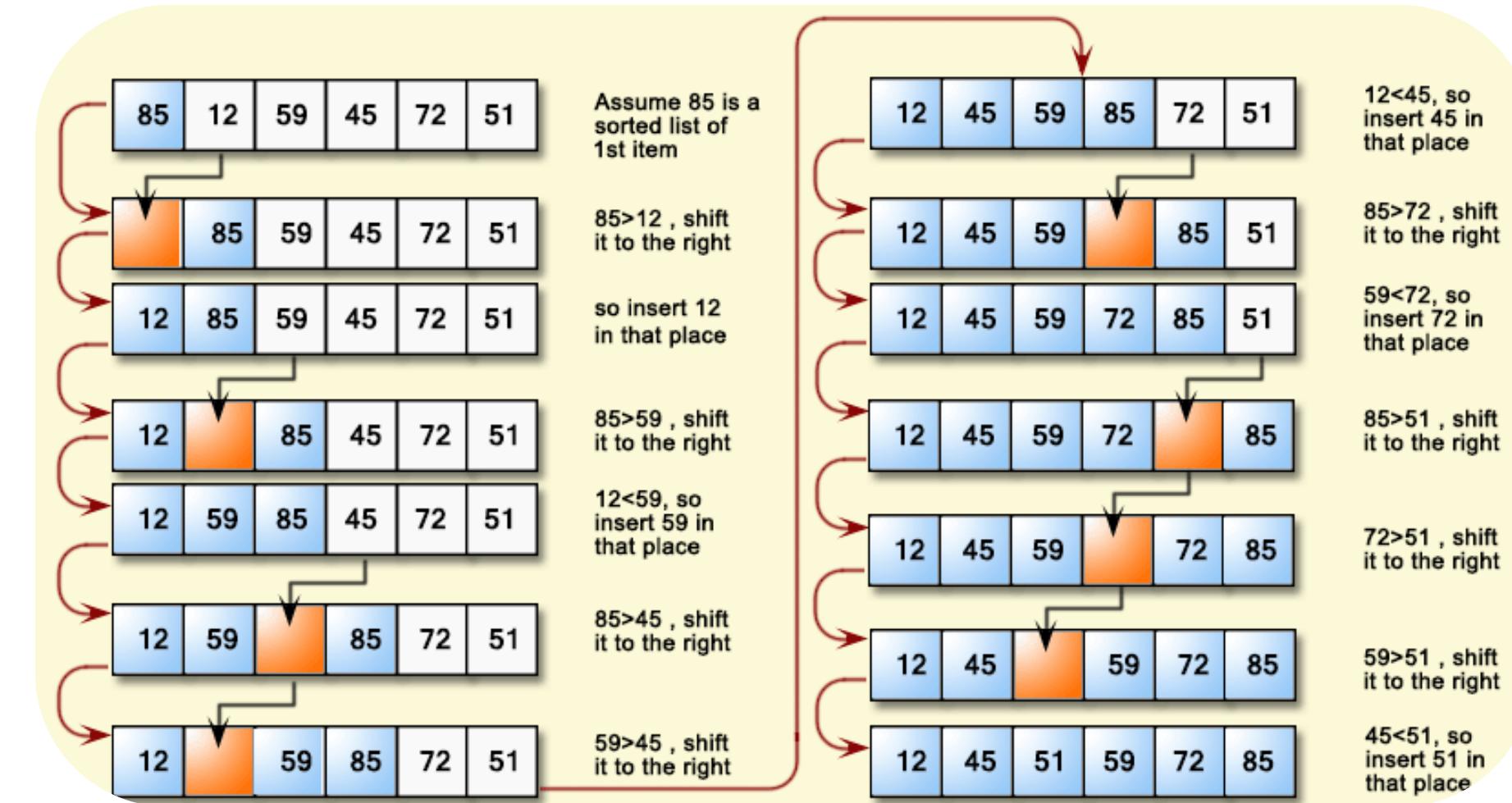


Steps to be followed:

1. Create and execute the 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.



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.

Outcome:

By the end of this demo, you will be able to implement the insertion sort algorithm in JavaScript, sort an array efficiently, and evaluate its time and space complexity.

Note: Refer to the demo document for detailed steps:
[03_Implementing_Insertion_Sort_Algorithm](#)

ASSISTED PRACTICE

Assisted Practice: Guidelines

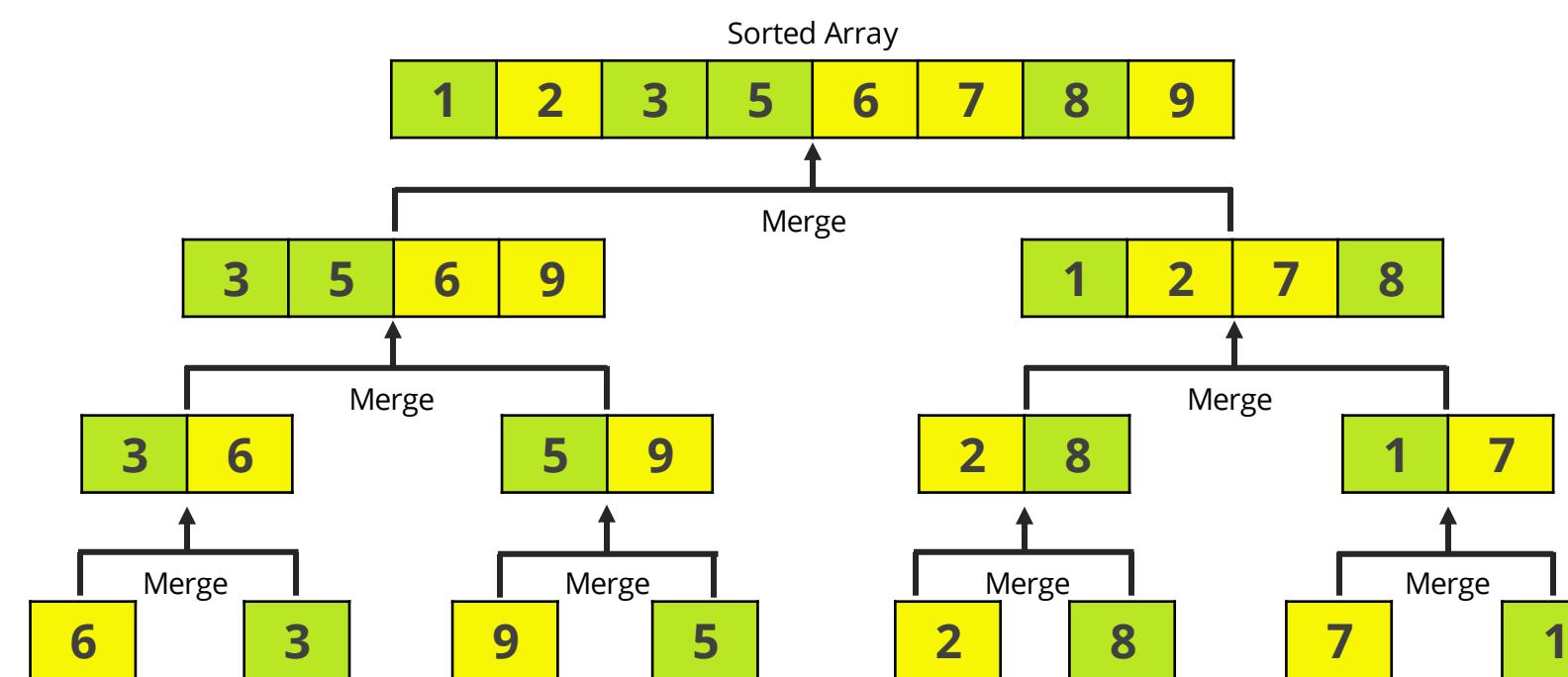


Steps to be followed:

1. Create and execute the JS file

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.



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.

Outcome:

By the end of this demo, you will be able to implement the merge sort algorithm in JavaScript, apply the divide-and-conquer approach, and analyze its time and space complexity.

Note: Refer to the demo document for detailed steps:
[04_Implementing_Merge_Sort_Alogrithm](#)

ASSISTED PRACTICE

Assisted Practice: Guidelines

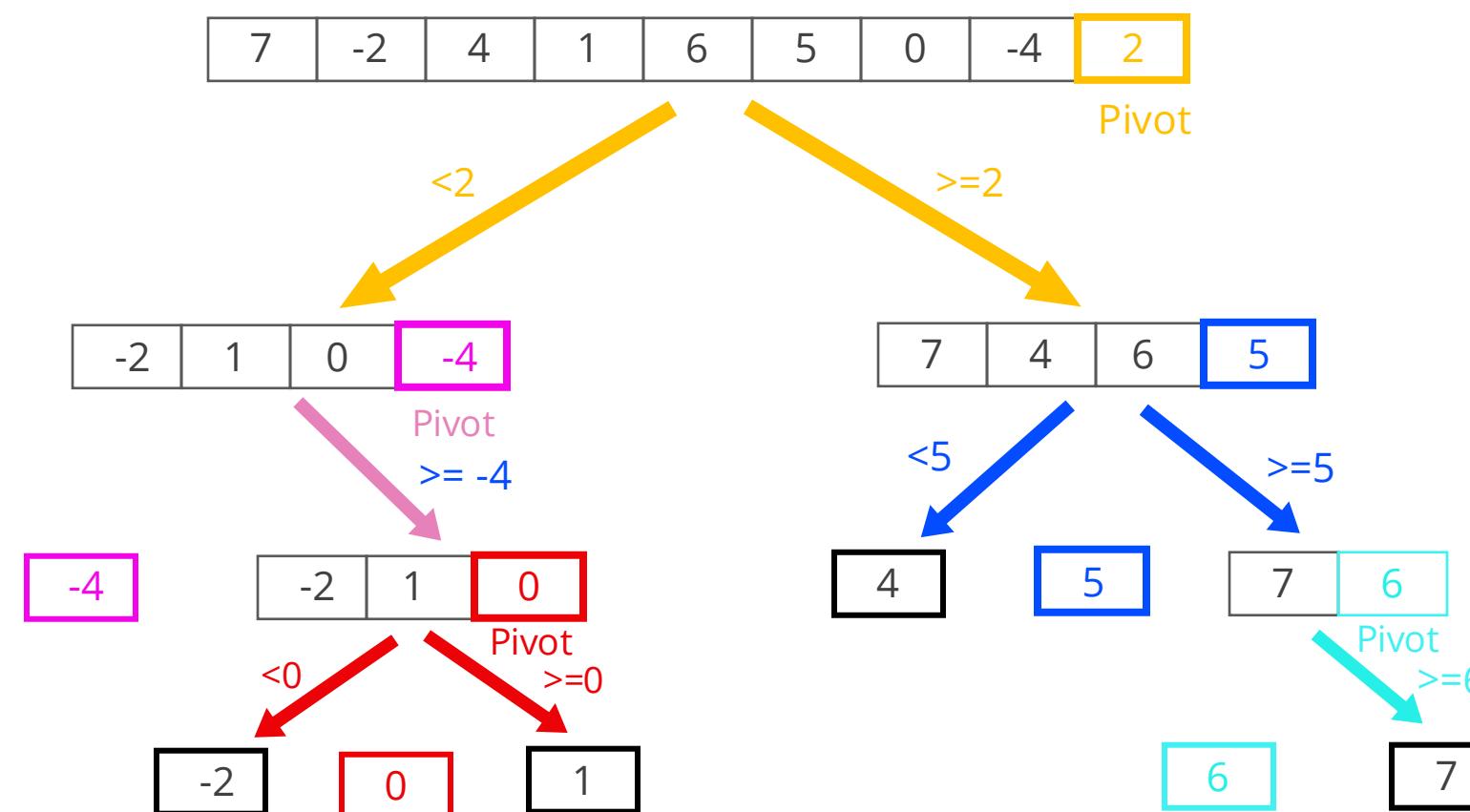


Steps to be followed:

1. Create and execute the JS file

Quick sort

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



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.

Outcome:

By the end of this demo, you will be able to implement the quick sort algorithm in JavaScript, sort arrays efficiently, and analyze its worst-case time and space complexity.

Note: Refer to the demo document for detailed steps:
[05_Implementing_Quick_Sort_Alogrithm](#)

ASSISTED PRACTICE

Assisted Practice: Guidelines

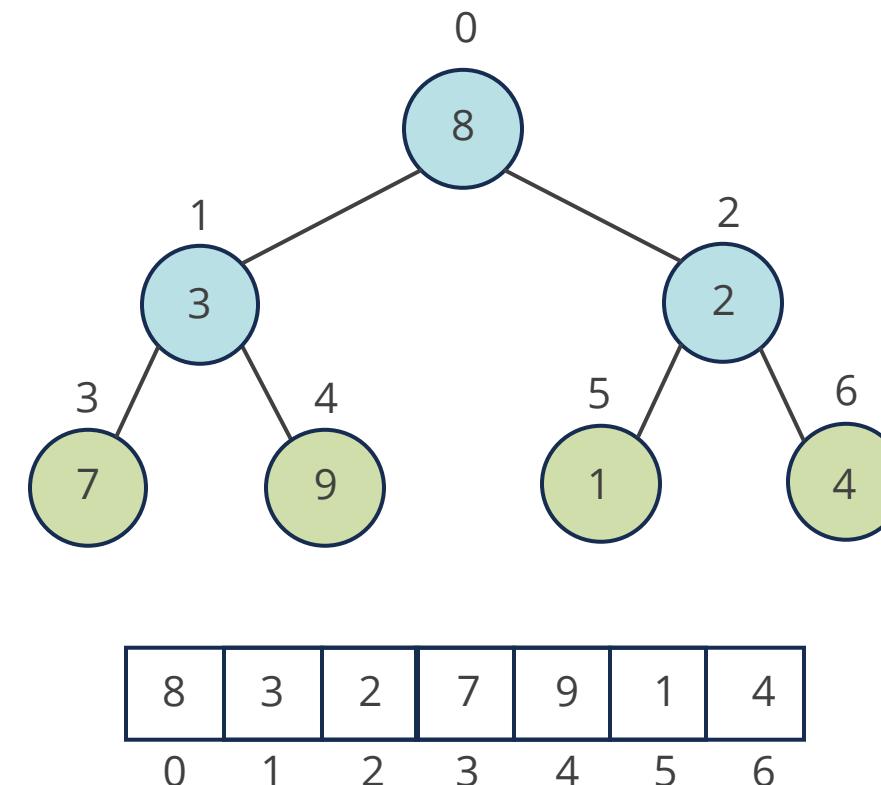


Steps to be followed:

1. Create and execute the JS file

Heap Sort

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



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.

Outcome:

By the end of this demo, you will be able to implement the heap sort algorithm in JavaScript, sort arrays efficiently, and analyze its time and space complexity.

Note: Refer to the demo document for detailed steps:
[06_Implementing_Heap_Sort_Algorithm](#)

Assisted Practice: Guidelines

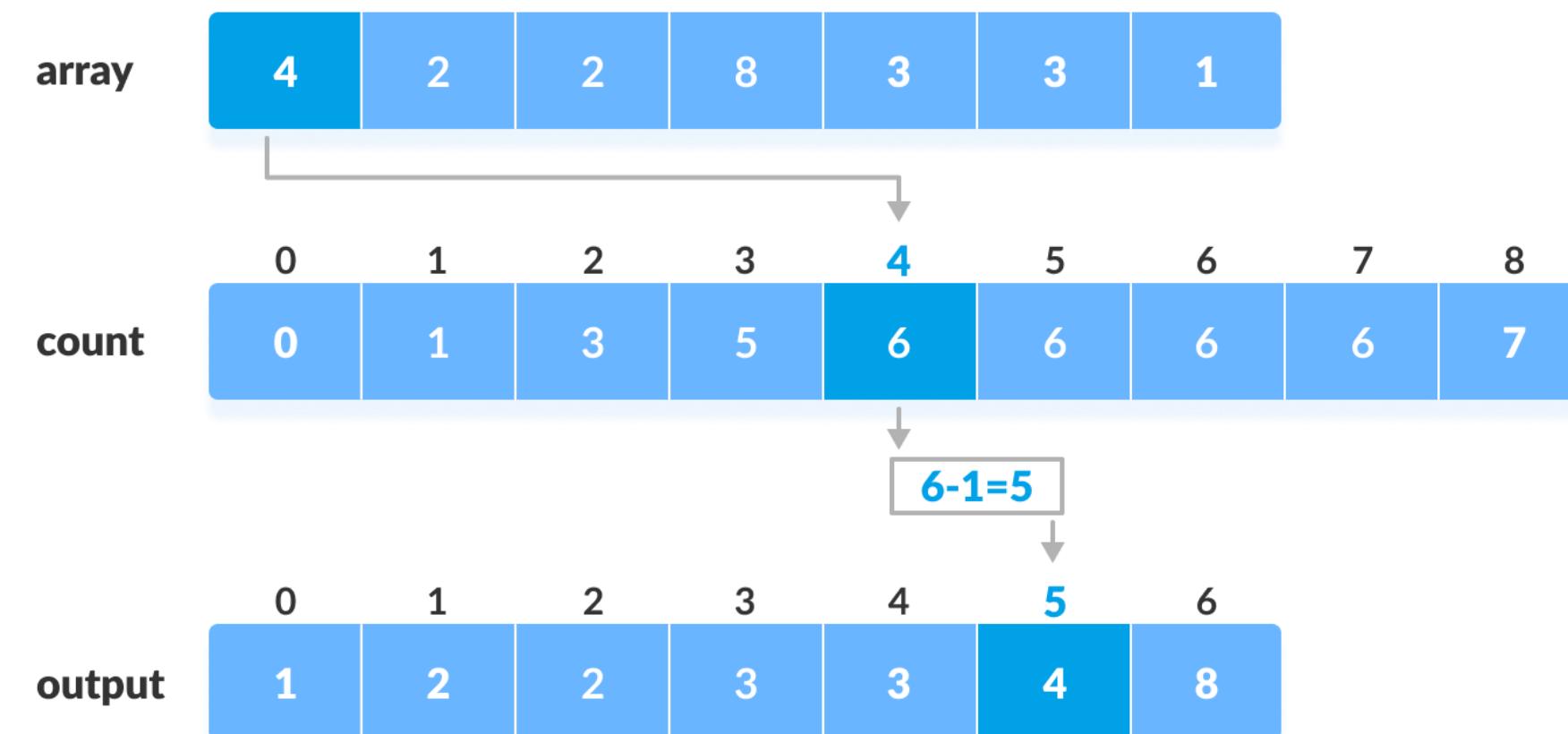


Steps to be followed:

1. Create and execute the JS file

Counting sort

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



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.

Outcome:

By the end of this demo, you will be able to implement the count sort algorithm in JavaScript, sort arrays efficiently, and analyze its time and space complexity.

Note: Refer to the demo document for detailed steps:
[07_Implementing_Count_Sort_Algorithm](#)

ASSISTED PRACTICE

Assisted Practice: Guidelines

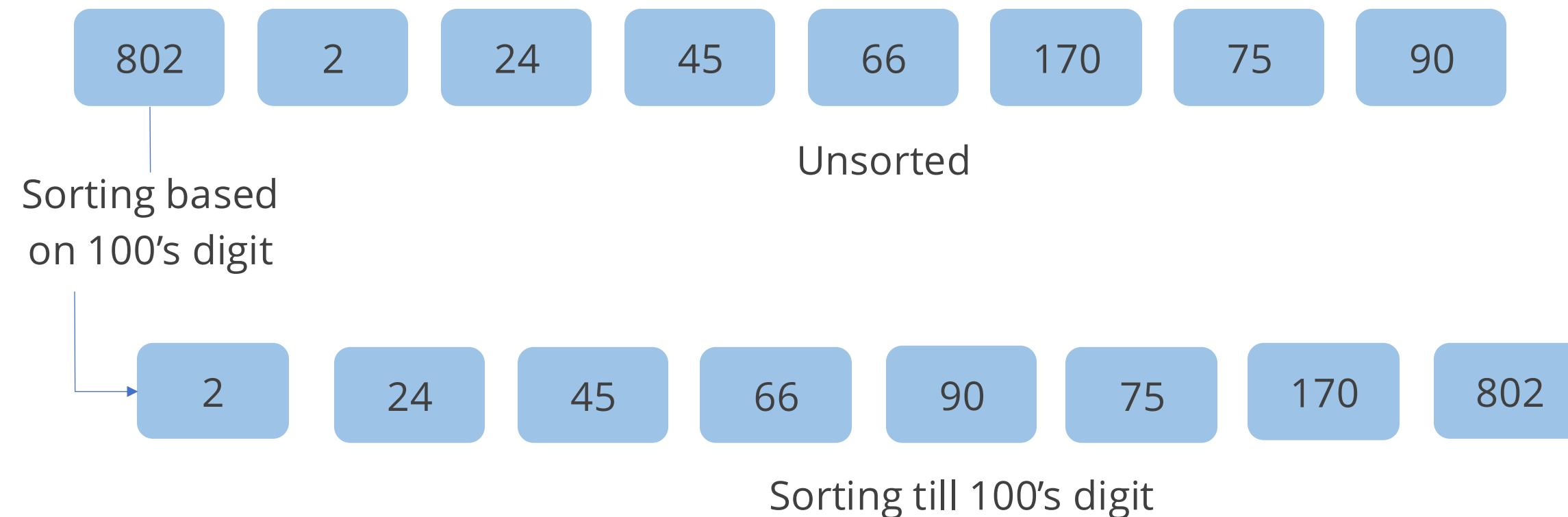


Steps to be followed:

1. Create and execute the JS file

Radix Sort

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



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.

Outcome:

By the end of this demo, you will be able to implement the radix sort algorithm in JavaScript, sort arrays efficiently, and evaluate its time and space complexity.

Note: Refer to the demo document for detailed steps:
[08_Implementing_Radix_Sorting_Algorithm](#)

ASSISTED PRACTICE

Assisted Practice: Guidelines



Steps to be followed:

1. Create and execute the JS file

Quick Check



A team of developers is working on a simple educational tool that demonstrates how sorting works. They need an algorithm that sorts data by repeatedly comparing and swapping adjacent elements until the entire list is in order. The algorithm should be easy to visualize, making it suitable for beginners.

Which sorting algorithm should they use?

- A. Merge sort
- B. Bubble sort
- C. Quick sort
- D. Insertion sort

Searching Algorithms

Searching Algorithms

Searching algorithms help locate specific elements in a dataset efficiently. They are classified based on their searching approach and efficiency.

Linear search

Scans each element sequentially and works on both sorted and unsorted data

Binary search

Repeatedly divides sorted data in half to locate the target

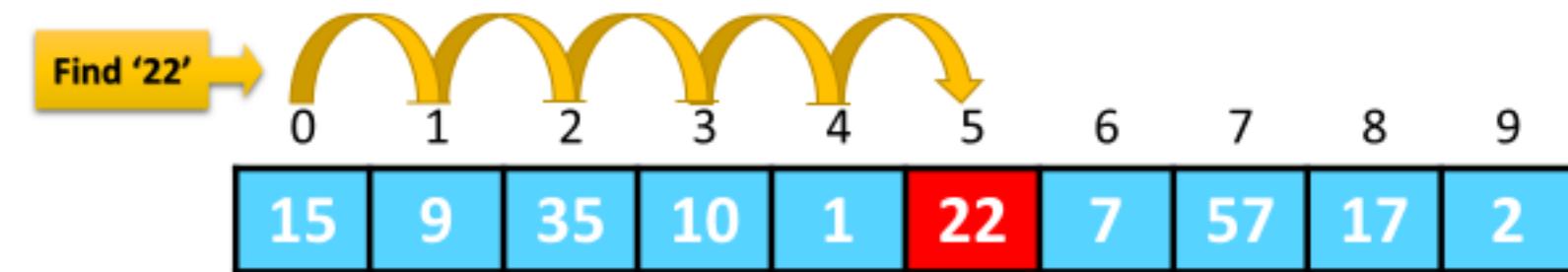
Jump search

Skips ahead by fixed steps and then performs a linear search within a block

Linear Search

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

Linear Search Algorithm



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.

Outcome:

By the end of this demo, you will be able to implement the linear search algorithm in JavaScript, search array elements efficiently, and analyze its time and space complexity.

Note: Refer to the demo document for detailed steps:
[09_Implementing_a_Linear_Search_Algorithm](#)

Assisted Practice: Guidelines

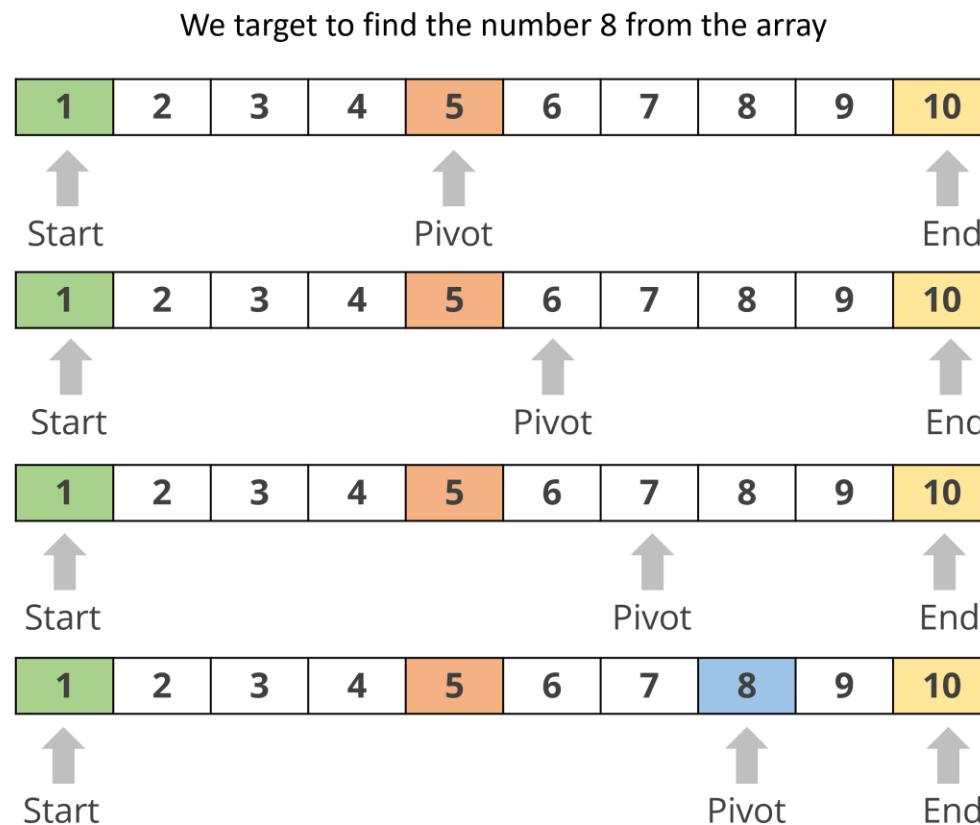


Steps to be followed:

1. Create and execute the JS file

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

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.

Outcome:

By the end of this demo, you will be able to implement the binary search algorithm in JavaScript, locate elements in a sorted array efficiently, and evaluate its time and space complexity.

Note: Refer to the demo document for detailed steps:
[10_Implementing_a_Binary_Search_Algorithm](#)

ASSISTED PRACTICE

Assisted Practice: Guidelines

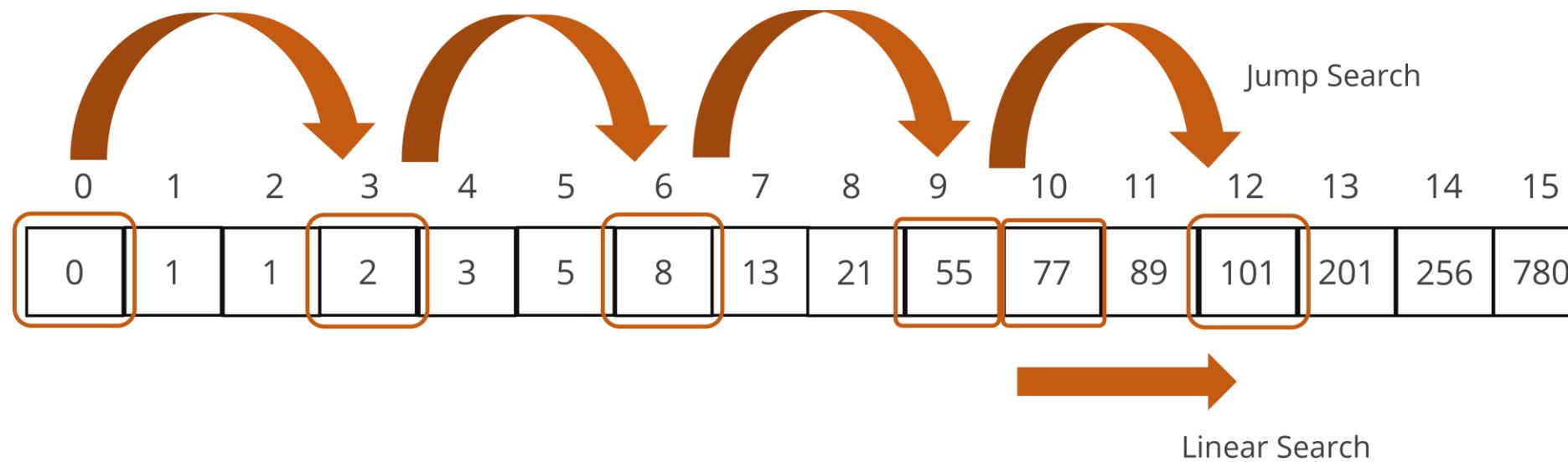


Steps to be followed:

1. Create and execute the JS file

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.

Different algorithms serve different purposes. Identifying the best fit requires **analyzing algorithms and evaluating design techniques**.

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.

Outcome:

By the end of this demo, you will be able to implement the jump search algorithm in JavaScript, perform efficient searches in arrays, and evaluate its time and space complexity.

Note: Refer to the demo document for detailed steps:
[11_Implementing_Jump_Search_Algorithm](#)

Assisted Practice: Guidelines



Steps to be followed:

1. Create and execute the JS file

Quick Check



A software engineer is building a contact search feature for a mobile app. The contact list is stored in an unsorted format, and users need to find a specific name quickly. Since sorting the list before every search is not practical, the engineer chooses a search method that checks each contact one by one until the desired name is found.

Which search algorithm should the engineer use?

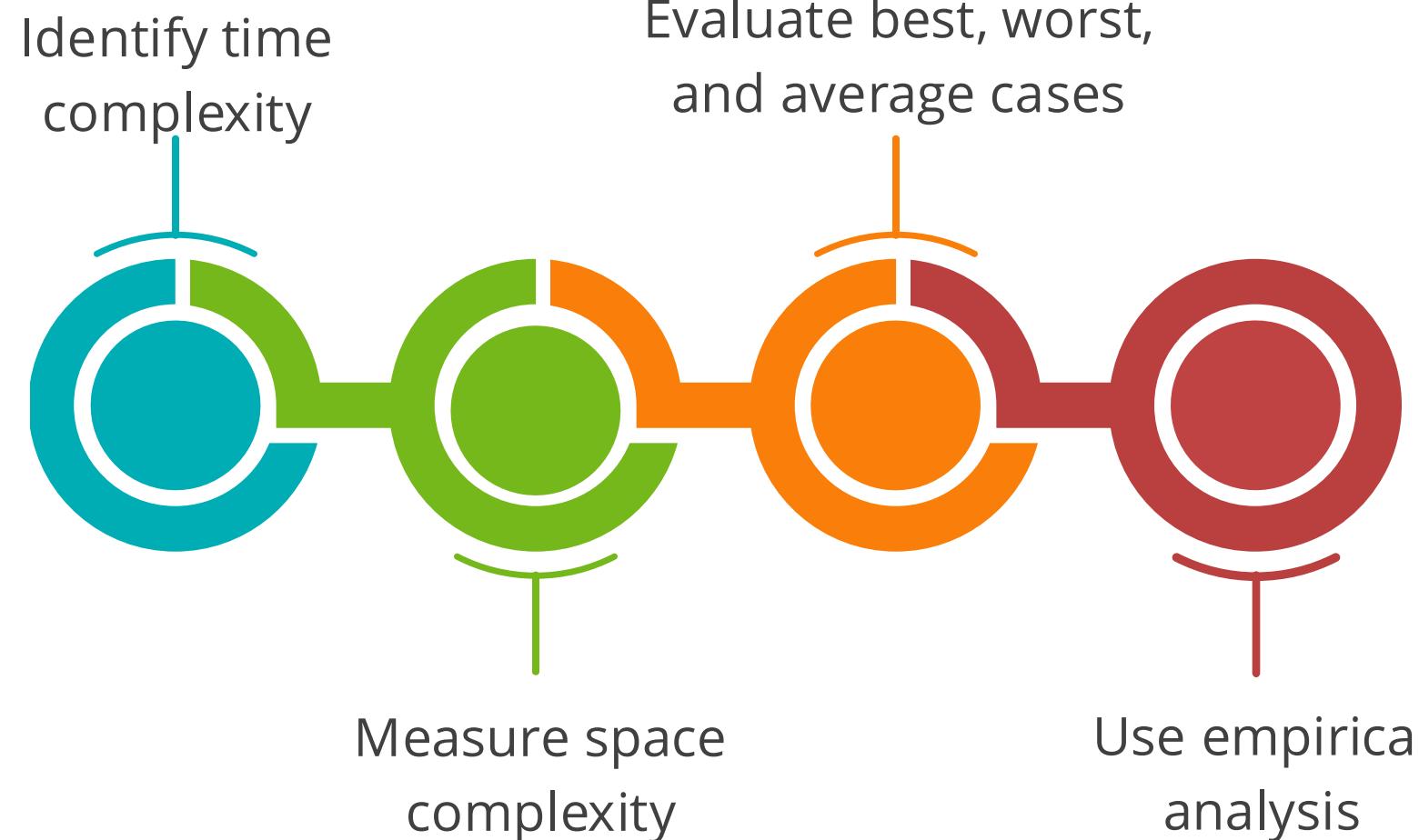
- A. Binary search
- B. Jump search
- C. Linear search
- D. Exponential search

Analyzing an Algorithm's Efficiency

Analyzing the Efficiency of an Algorithm

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

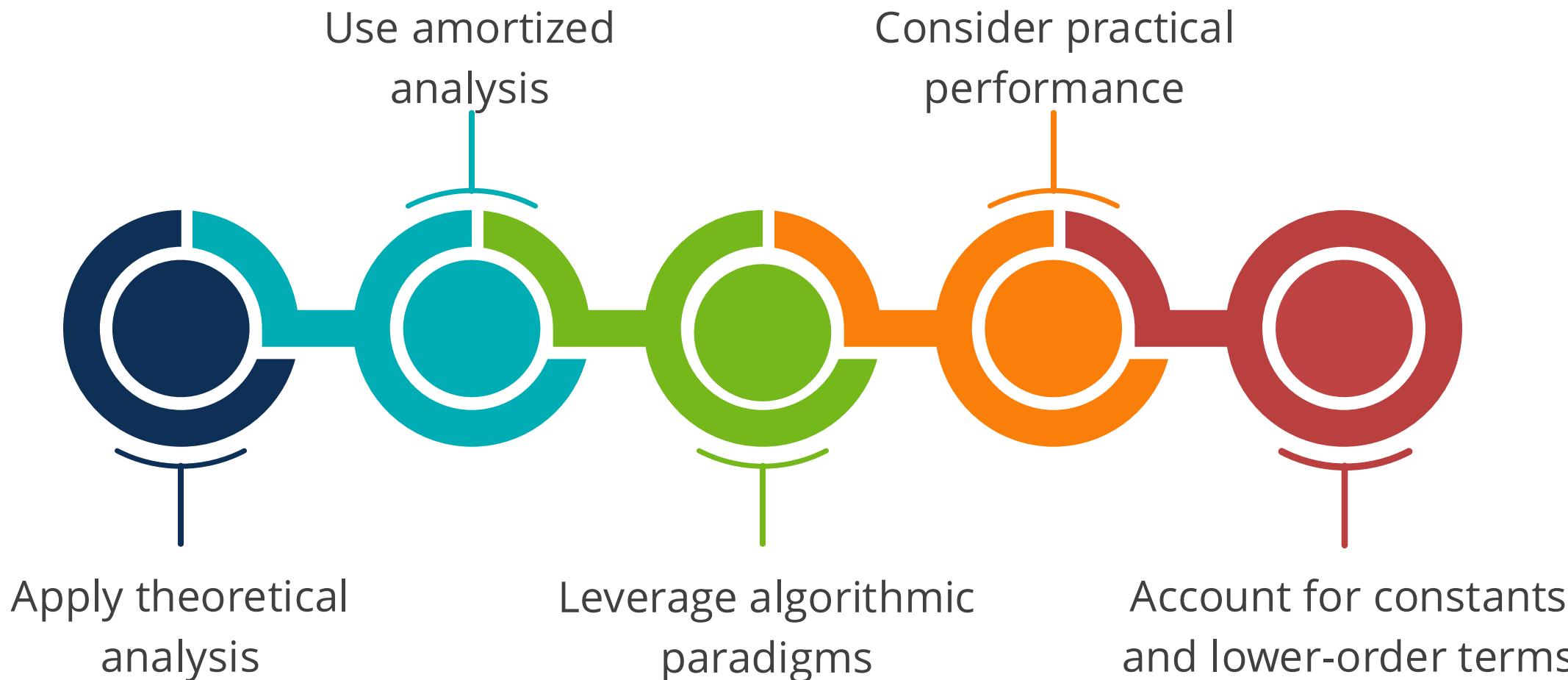
Here are the key aspects involved in this analysis:



Analyzing the Efficiency of an Algorithm

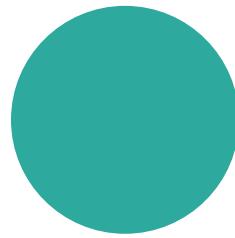
It typically involves understanding the time and space complexity, which is crucial for evaluating the performance.

Here are the key aspects involved in this analysis:

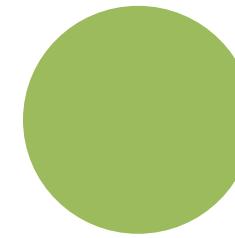


Time Complexity

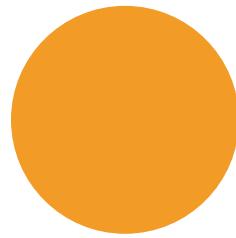
It measures an algorithm's completion time based on input length, estimating its efficiency as input size increases. Key factors include:



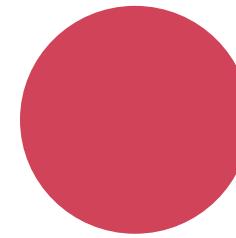
Tracking growth rate



Evaluating nested loops and operations



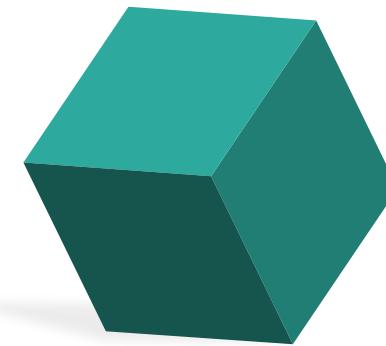
Applying divide and conquer



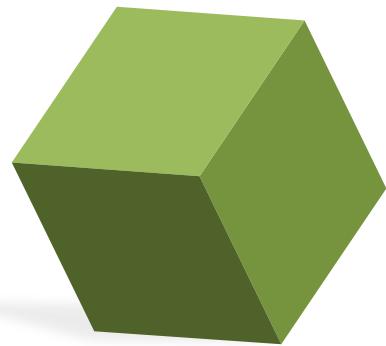
Optimizing logarithmic time

Space Complexity

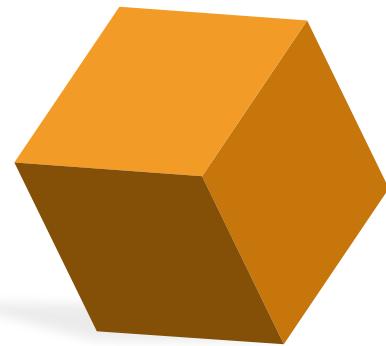
It denotes the memory requirement of an algorithm for its execution, evaluating the overall memory usage in correlation with the size of the input. Key aspects involved in evaluating include:



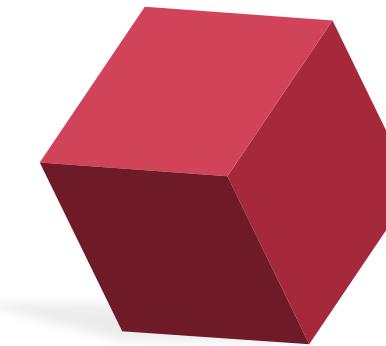
Allocating memory
for input and
auxiliary space



Managing memory
in recursive
algorithms



Using static and
dynamic memory
allocation



Optimizing amortized
space complexity

Quick Check



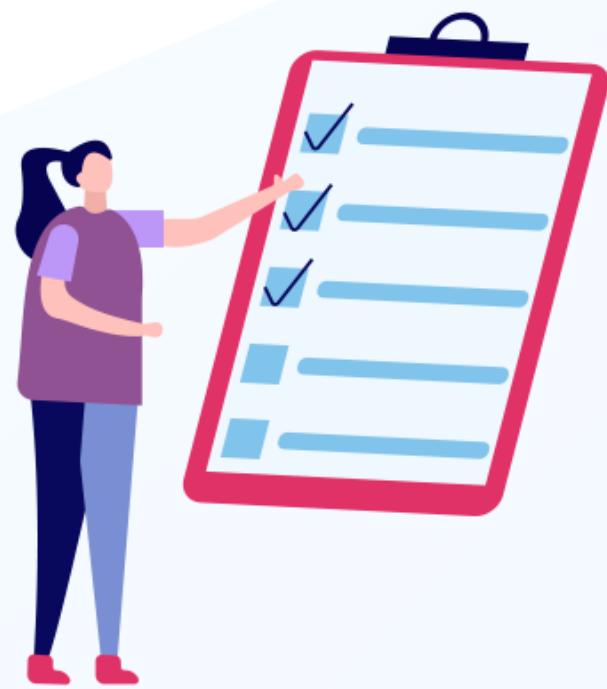
A software engineer is designing a program that processes a list of customer transactions. As the number of transactions increases, the program takes significantly longer to execute. The engineer wants to evaluate how the runtime of the program grows as more transactions are added.

What measure should the engineer use to analyze this growth?

- A. The amount of memory used
- B. The number of lines of code
- C. The time it takes to run as a function of input size
- D. The cost of the algorithm

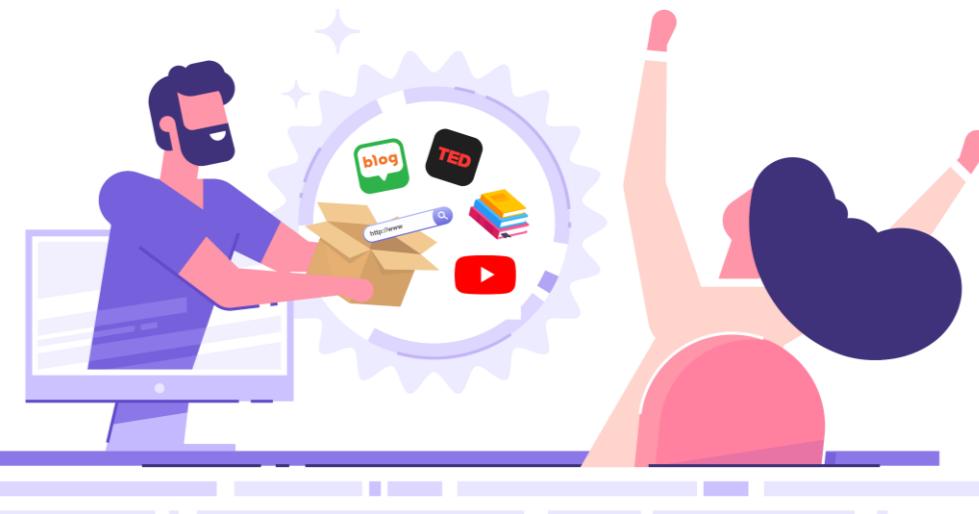
Key Takeaways

- Algorithms provide step-by-step instructions to solve problems efficiently in computing, ensuring precise and structured solutions.
- 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.
- Search algorithms like linear, binary, and jump search offer different ways to locate data efficiently.
- Bubble sort swaps neighboring elements, selection sort finds the minimum, and insertion sort gradually builds a sorted list.



Additional Resources

- [Data Structures and Algorithms](#)
- [DSA](#)
- [DSA Interview Questions](#)



Thank You