

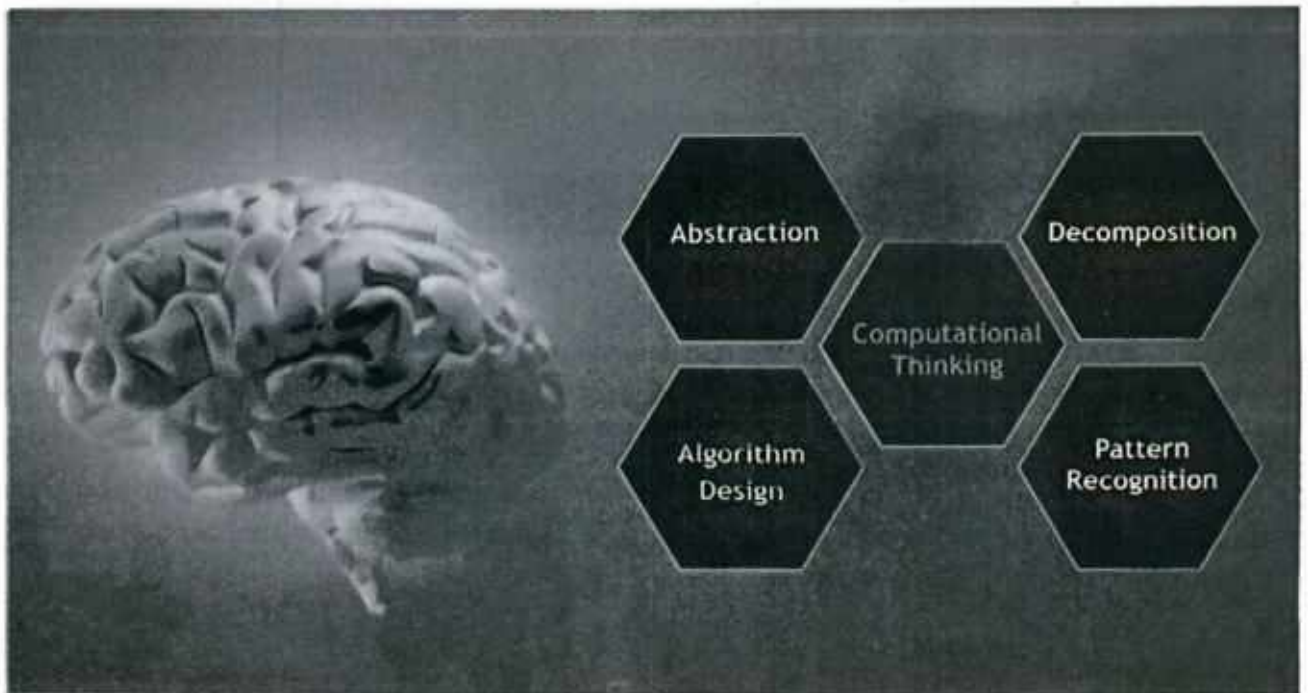
Unit 02

Computational Thinking and Algorithms



After completing this lesson, you will be able to:

- plan, develop, systematically test, and refine computational artifacts for problem solving.
- apply common search, and sort algorithms.

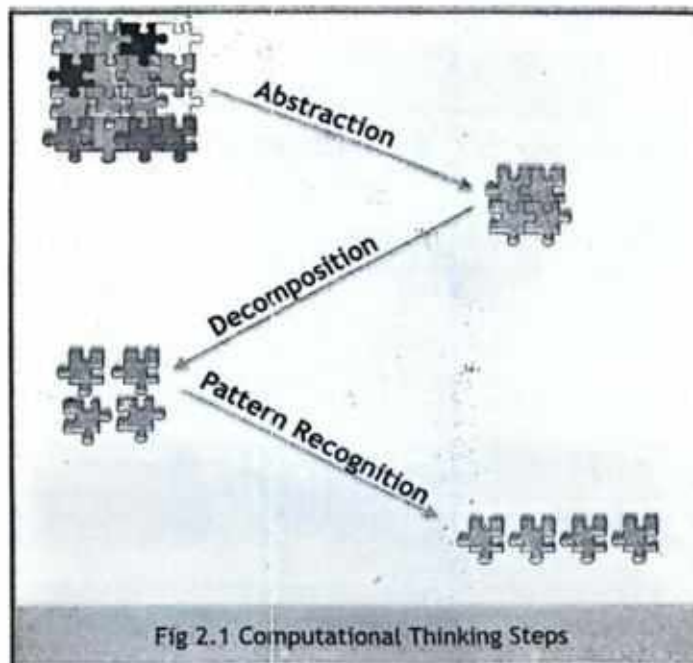


UNIT INTRODUCTION

Computer systems are used to solve real word problems relevant to any topic or aspect of our life. For example education, healthcare, Transportation, economics. A problem is a challenge or situation for which a solution is required. The process of analyzing that situation and accordingly handling it is referred as problem solving.

Computational thinking is a thought process that helps us to understand the problem and solve it in a way that computers do. Computational thinking equires understanding the capabilities of computers. For example, computers process the input in a faster and reliable way and it works by following the input-process-output model.

To solve problems in any field including computer science, we can apply computational thinking. The solution of a problem where computational thinking is applied uses the following properties:



Abstraction:

Simplifying a problem or system by focusing on the most relevant aspects while ignoring unnecessary details. It is about looking at the big picture without getting lost in the tiny details.

Consider the task of Creating a website

Abstraction is to focus on the main design and functionality without getting into the coding details. Think about the-layout, colour scheme, and user interaction, rather than every line of code.

Decomposition:

Breaking down a complex problem into smaller, more manageable sub-problems or tasks.

Consider the task of Developing a mobile app

Decomposition is to break the task into smaller parts like designing the user interface, coding specific features (e.g., login, navigation), and handling data storage. Each part becomes a manageable sub-task.

Pattern Recognition:

Identifying patterns or trends within data, problems, or solutions.

Consider the task where we need to analysing user behaviour on a website,

Pattern Recognition is to identify trends, such as which pages users visit the most or common paths they take. This helps in making improvements based on observed patterns.

Algorithmic Design:

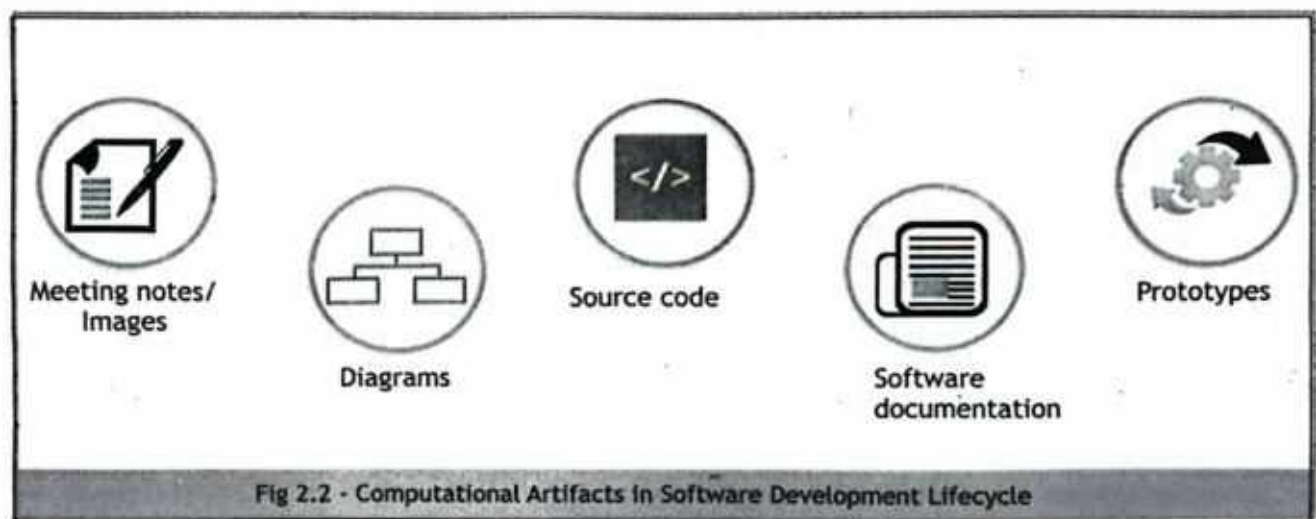
Developing a step-by-step solution to solve a problem.

2.1 Computational Artifacts

Computational artifacts refer to human made objects and systems using computational thinking. An artifact is a byproduct of software. Particularly in computer science, the computational artifacts could include programs, websites, videos, simulations, databases, digital animations, software systems, e-commerce platforms, and mobile applications.

2.1.1 Computational Artifacts in Software Development

In software development, the artifact could be an object that helps to describe the architecture, design, and function of software. There are a range of different artifacts generated during the lifecycle of software development as shown in the Fig 2.2. The artifacts are considered important because they make the process of developing software easy.



The artifacts generated during computational thinking in the software development process are the objects to be used before starting the coding phase. Artifacts define the behavior of a software with the help of some control sequences. By using these artifacts, software developers can easily understand how software works without requiring going into coding complexities.

2.1.2 Computational Solution Design

During the computational thinking, following are the commonly used computational artifacts.

a. Design Methods

- Algorithms
- Flowcharts
- Pseudocode

Algorithms, pseudocodes, and flowcharts serve as a structured and systematic representation of a computational process.

Algorithms: While solving a problem using a computer, the first step is to think about the algorithm. An algorithm is a step-by-step set of instructions that defines how a specific problem should be solved. It is a high-level description of the solution without worrying about the implementation details such as specific programming language. Algorithms are normally written in natural language.

Example

Let's consider a simpler example and write algorithm to add two numbers.

Algorithm: Add Two Numbers

1. Input two numbers, let's call them num1 and num2.
2. Add num1 and num2 together.
3. Store the result in a variable, let's call it sum.
4. Output the value of sum.

Flowchart: Flowchart is another tool used in the process of algorithm design. This provides a visual representation of flow and logic using defined symbols. Flowcharts are mostly helpful in explaining complex loops and decision-making processes.

Example

Flowchart: Add Two Numbers

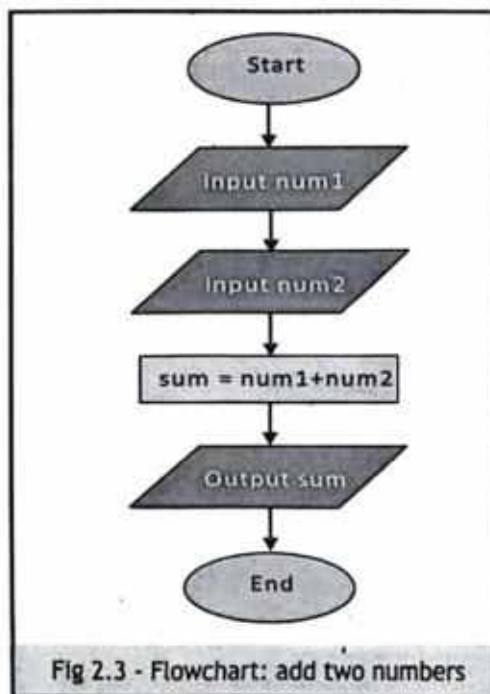


Fig 2.3 - Flowchart: add two numbers

Pseudocode: Once you have devised an algorithm, now it's time to create pseudocode. Pseudocode is an intermediate step between the algorithm and actual programming code. It's a more structured and semi-formal way of representing the algorithm using a combination of natural language and simplified programming constructs. Pseudocode helps in explaining the logic and structure of the algorithm before starting programming. It can be thought of as a detailed plan of how the algorithm should be implemented.

Example

Pseudocode: Add Two Numbers

// Step 1: Input two numbers

Input "Enter the first number: " into num1

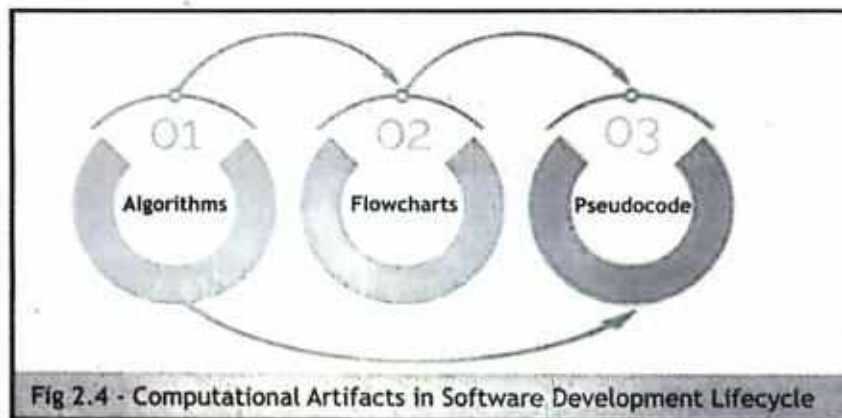
Input "Enter the second number: " into num2

// Step 2: Add the two numbers and store result in sum

sum <- num1 + num2

// Step 3: Output the result

Output "The sum of two numbers is: " sum



However, it is important to note that the above sequence can vary depending on the complexity of the problem and your personal preference. Some programmers may skip flowcharts and go directly from algorithms to pseudocode or even from algorithms to writing code; especially for simpler tasks.

Let's consider another example to check whether a given number is even or odd and write algorithm, pseudocode and also draw its flowchart.

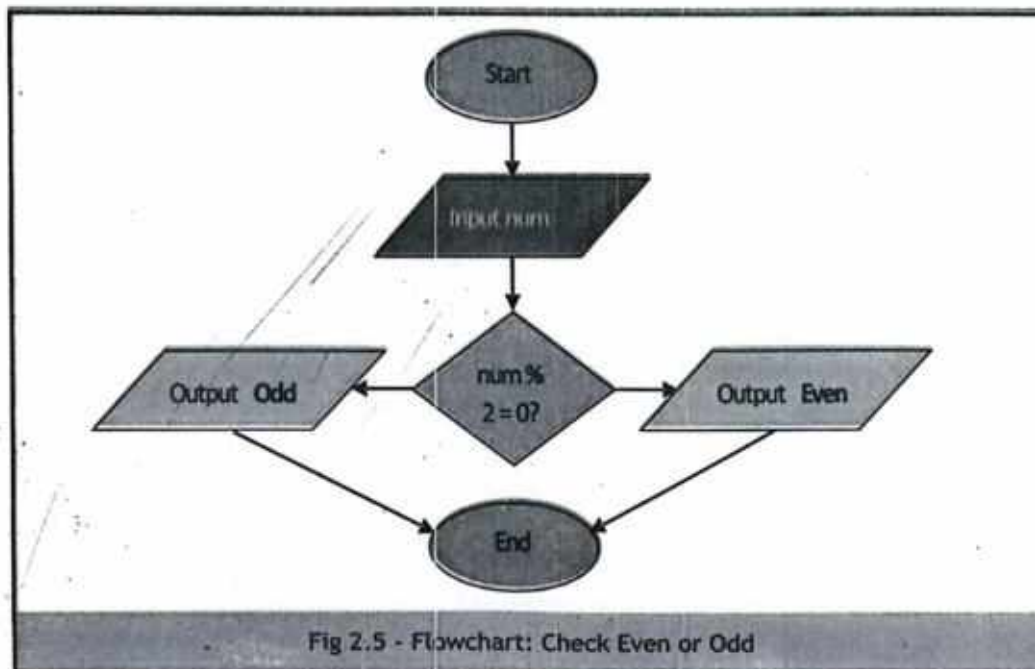
Algorithm: Check Even or Odd

1. Input a number, let's call it num.
2. Check if num is divisible by 2.
3. If the remainder is 0, output "The number is even."
4. If the remainder is not 0, output "The number is odd."

Pseudocode: Check Even or Odd

```
// Step 1: Input a number
    Input "Enter a number: " into num
// Step 2: Check if the number is even or odd
    if num % 2 = 0 then
// Step 3: Output result for even number
        Output "number is even."
    else
// Step 4: Output result for odd number
        Output "number is odd."
    end if
```

Flowchart: Check Even or Odd



b. Algorithm Design

The primary goal of pseudocode is to convey the algorithm's logic in a clear and understandable way, irrespective of specific programming language conventions.

Pseudocode writing is not standardized and there isn't a universal set of rules for its formatting. However, certain conventions are commonly used. Below is a set of guidelines that are often followed, but keep in mind that these are not strict rules, and you can adapt them based on your preferences or the requirements of a specific solution:

Font, Size, Style:

Font: It is typically written in a plain, easy-to-read font.

Size: Use a standard font size that is easy to read, often around 10-12 points.

Style: Italicizing or bolding keywords is sometimes done for emphasis, but it's not a strict rule.

Indentation:

Use consistent indentation to visually represent the structure of your code. Commonly, 2-4 spaces or a tab character is used for each level of indentation.

Case:

While pseudocode is not case-sensitive, using uppercase for keywords and lowercase for variables can enhance readability. For example, `FOR i FROM 1 TO n` or `if condition then`.

Line Numbers:

Including line numbers is optional and depends on personal preference or the requirements of a specific project. Line numbers can aid in discussions about the algorithm.

Comments:

Comments are essential for explaining the logic. They are often denoted by symbols like `//` for single-line comments or `/**/` for multiline comments.

Data Type Keywords:

Some pseudocode styles use data type keywords like `INTEGER`, `STRING`, etc., to indicate the type of a variable. However, this is optional.

Variable Assignments & Declarations:

Variables can be declared using `DECLARE` or simply by mentioning them. Assignments are often denoted using `□` or `:=`.

Common Operators:

Use standard operators such as `+`, `-`, `*`, `/`, `=`, `<`, `>`, `<=`, `>=`, `!=` to represent common operations.

Key Commands:

Key commands like `INPUT`, `OUTPUT`, `FOR`, `IF`, `WHILE` can be used to represent specific actions or control structures.

2.1.3 Planning and Developing a Computational Artifact

The following steps are used for planning and development of an artifact.

- a. Define the problem:** State the problem you are trying to solve in clear and concise terms.
- b. List the inputs** (information needed to solve the problem) and the expected outputs (what the algorithm will produce as a result)
- c. Plan:**
 - i) Outline the logic:** For this purpose, breakdown the problem into smaller problems and

consider how to solve each one.

ii) **Choose Data Structures:** A data structure is a storage to save and organize the data. We have different types of data structures such as array, stack, queue, etc. The choice of data structures is determined by considering the merits and drawbacks associated with each specific data structure.

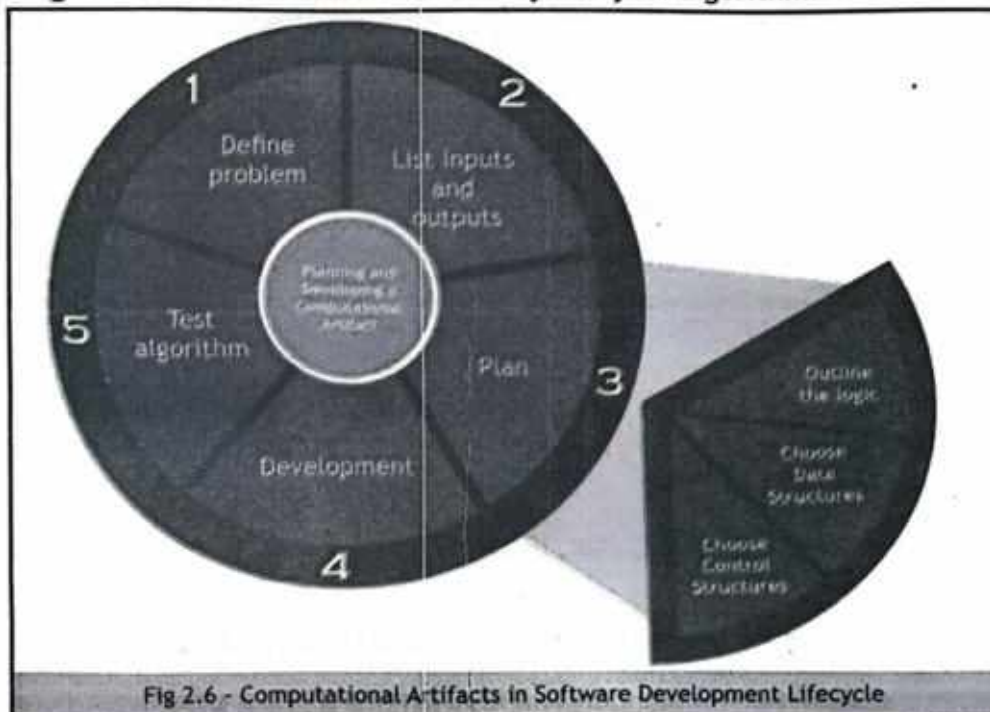
iii) **Choose Control Structures:** enables to determine the order of program execution. The sequential order of execution is disturbed in two aspects:

- **skip some program statements.** This is usually done with the help of conditional statements.
- **repeat one or more program statements.** This is done by using loops.

Those control structures are selected that are most appropriate for over solution.

d. **Development:** Describe the steps needed to convert or manipulate the inputs to produce the outputs. Start at a high level first and keep refining the steps until they are effectively computable operations. Keep the process simplified and remember that the number of steps should be finite.

e. **Test the algorithm:** choose data sets and verify that your algorithm.



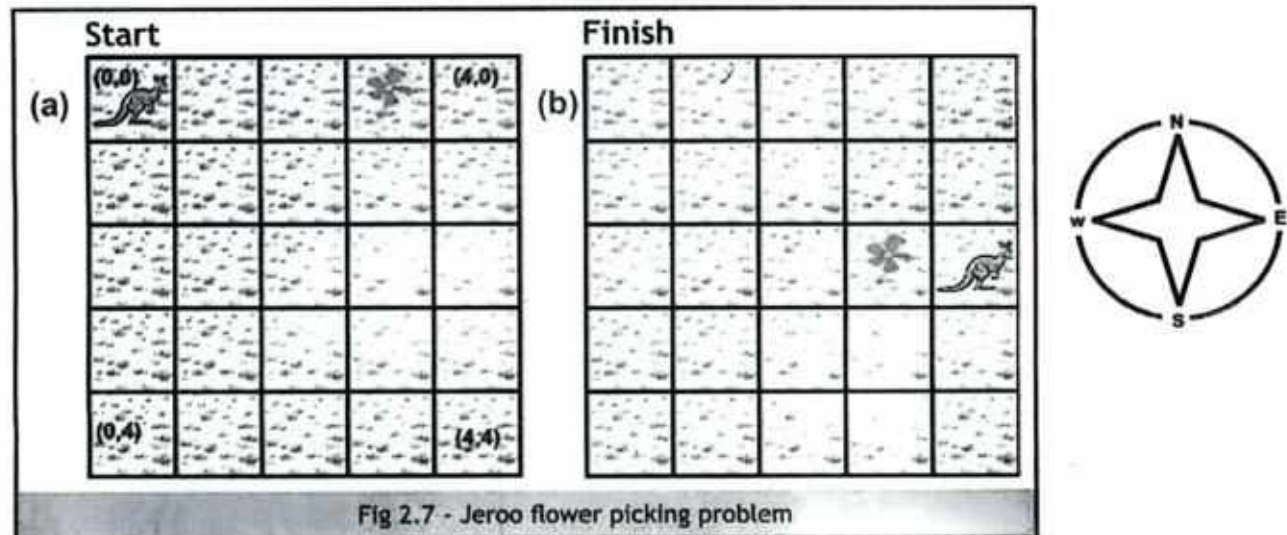
Example

To solve this example, we need to have some domain knowledge first about the problem.

Jeroo is a Kangaroo like animal. A Jeroo can pick flowers from a location and can plant them to some other. The Jeroo moves by hopping in the four directions e.g. forward, backward, right and left. As the Jeroo hop; it must avoid water, nets, and other of it kind.

Problem Definition

Consider the given diagram, where the field is divided into 5x5 matrix and a Jeroo is available at position (0, 0) facing East. The flower is located at (3, 0). Write an algorithm where the Jeroo picks the flower and plant it to another location (3, 2). After its plantation, the Jeroo should move one hop East and stop there. Assume there are no other Jeroos, flowers or nets available in the field. Top left corner of matrix is (0,0) values of x increases horizontally and y downwards.



List of inputs & Outputs

Inputs:

Jeroo starts location (0, 0)
flower current location (3, 0)

Outputs

flower new location (3, 2)
Jeroo should stop one hop East

Plan:

Breakdown the problem:

Pick flower
Put flower
Move East

Development:

Pick Flower

Hop 3 times
take the flower

Put Flower

Turn right
Hop 2 times

Plant the flower

Move East

Turn left

Hop 1 time

After the development of the algorithm, try to refine it by asking yourself the following questions:

- ❶ Does the developed algorithm solve some specific problem or could it may be generalized.

For example:

The algorithm that finds the area of a circle having radius of 3 meters (formula $A=\pi 3^2$) solves a specific problem, but it could be made generalized by using the formula $A=\pi r^2$

- ❷ Can the developed algorithm be made simplified?

For Example:

The formula for calculating perimeter of square is:

Perimeter of square = side + side + side + side

A simple formula would be:

Perimeter of square = $4 \times \text{side}$

2.1.4 Testing Computational Artifacts

The computational artifacts should be tested by considering potential logical errors. For example, you should think what would happen if a user enters invalid or a certain out of range input. Testing and accordingly refinement of solution is the iterative process of a computational artifact.

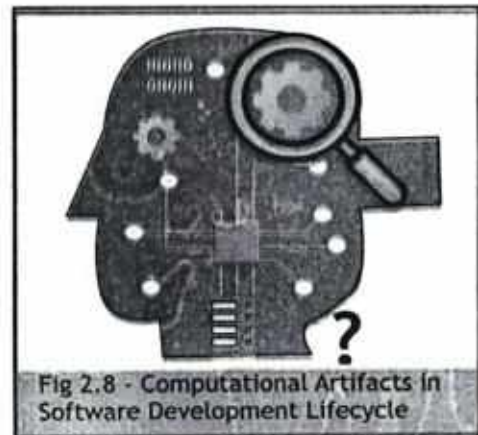


Fig 2.8 - Computational Artifacts in Software Development Lifecycle

a) Tracing an Algorithm

Tracing of algorithm is also known as a "desk check or dry-run". The programmers convert the algorithms to code that is ultimately handed over to computer for compilation and execution. But before going to write code for an algorithm, we manually verify that the algorithm works correctly. For this purpose, trace method is used that hand simulate the execution of your algorithm, keep change track of variable values and flow of logic control.

Tracing of Algorithm involves the following:

- Understand the Algorithm:** Before tracing is started, thoroughly understand the algorithm's logic flow.
- Choose Test Input:** To trace the algorithm it requires input; a complete set of input is therefore chosen at this point.

- c. **Initialization:** Initialize the variables and data structures. This helps in setting up the algorithm in its initial state, where the algorithm has never run before.
- d. **Trace Each Step:** hand execute the algorithm step-by-step, tracking the flow of control. For each step of algorithm, do the following:
 - ⑥ **Track record Inputs:** write the input values and variable values relevant to that step.
 - ⑥ **Perform processing:** Do computation / perform the operation specified in that step of algorithm.
 - ⑥ **Update Variables:** keep an eye on the changes in the variable values and data structures in response to performing operation.
 - ⑥ **Go by Control Flow:** Normally, the steps of algorithms are followed sequentially but sometime this sequence is broken by loops, conditional statements, and function calls. Therefore, we need to track how an algorithm decides which step to execute next.
- e. **Repeat Until Completion:** Tracing the algorithm continues step by step until we reach the end of algorithm.
- f. **Verify Output:** At the end, evaluate the final output and ensure that it matches our expectations based on the given set of inputs.

Example:

Let's take example of Jeroo that we solved in previous topic and apply algorithm tracing.

- a. **Understand the Algorithm:** What we need to do, from where we need to pick flowers and to which location, we need to plant flower.
- b. **Choose test Input:** In the example flower pickup location e.g. (3,0) serves as input
- c. **Initialization:** Initial position of Jeroo and Initial position of flower
- d. **Trace each step:**
 - ⑥ **Track record input:** Jeroo's location, flower's location are the input variables.
 - ⑥ **Processing:** Taking moves, taking turns, picking flower, putting flower are the action that require processing.
 - ⑥ **Update variables:** Jeroo's location and flower's location will be updated after each step movement of jeroo and flower pick / put action respectively.
 - ⑥ **Go by control flow:** follow the algorithm e.g. sequentially.
- e. **Repeat until Completion:** take moves and turns until the jeroo is reached its destination and flower is placed at the designated location.
- f. **Verify output:** verify whether the completion done in above step is same as desired in the problem statement.

Trace Tables: The dry run of an algorithm can be done with a technique known as Trace Tables. These tables show the variables change at each stage in the algorithm. The trace tables generate the output on a given set of input data to test if algorithm is giving expected output. It also helps to check that our logic is correct and find logical errors, if any.

Example: Consider the following the pseudocode and accordingly its trace table.

<u>Pseudocode</u>	
1.	number = 3
2.	PRINT number
3.	FOR i from 1 to 3:
4.	number = number +5
5.	PRINT number
6.	PRINT "?"

<u>Trace Table</u>			
Pseudocode steps	Number	i	Output
1	3		
2			3
3		1	
4	8		
5			8
3		2	
4	13		
5			13
3		3	
4	18		
5			18
6			?

Example:
Consider the same example, we discussed in the chapter above.

<u>Pseudocode</u>	
1.	Start
2.	Set c=1
3.	Read n
4.	m=n*c
5.	print m
6.	c=c+1
7.	Repeat steps 4 to 6 while (c<=10)
8.	End



Teacher's Guide

"CS Unplugged: Computational Thinking Activities"

<https://www.csunplugged.org/>

This resource provides a series of activities designed to familiarize students with computational thinking principles such as problem decomposition, algorithms, and testing. These activities serve as an excellent introduction to the fundamental aspects of creating computational artifacts, all without the requirement of computers.

Table 2.1- Trace table for the given pseudocode

Pseudocode Step	c	Input n	m	Output m
1	-	-	-	-
2	1	-	-	-
3	1	2		
4	1	2	2	
5	1	2	2	2
6	2	2	2	-
7	2	2	2	-
4	2	2	4	-
5	2	2	4	4
6	3	2	4	-
7	3	2	4	-
4	3	2	6	-
5	3	2	6	6
6	4	2	6	-
7	4	2	6	-
4	4	2	8	-
5	4	2	8	8
6	5	2	8	-
7	5	2	8	-
4	5	2	10	
5	5	2	10	10
6	6	2	10	-

(a)

Algorithm Step	c	Input n	m	Output m
7	6	2	10	-
4	6	2	12	-
5	6	2	12	12
6	7	2	12	-
7	7	2	12	-
4	7	2	14	-
5	7	2	14	14
6	8	2	14	-
7	8	2	14	-
4	8	2	16	
5	8	2	16	16
6	9	2	16	-
7	9	2	16	-
4	9	2	18	-
5	9	2	18	18
6	10	2	18	-
7	10	2	18	-
4	10	2	20	-
5	10	2	20	20
6	11	2	20	-
7	11	2	20	-
8	11	2	20	

(b)

b) Algorithms Evaluation Parameters

Algorithms are evaluated using a variety of criteria and methods to assess their performance, efficiency, and correctness for specific tasks. The evaluation process determines whether an algorithm meets its intended objectives. The following are some common ways to evaluate algorithms, however at advanced level there are many other ways also.

Correctness:

Correctness is considered the most fundamental evaluation parameter. An algorithm must produce the desired output for all possible valid inputs. Therefore, various inputs are tried during evaluation of algorithm.

Efficiency:

Efficiency of an algorithm deals with time and space (amount of memory) complexity. The time complexity refers to the amount of time taken by an algorithm to run and space complexity refers to the amount of memory required to run that algorithm.

Clarity:

This refers to how easily the algorithm's logic and steps can be understood by humans involved in

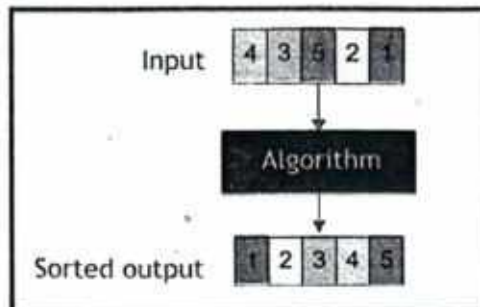
the software development lifecycle.

Reliability:

Reliability refers to the ability to always produce correct and accurate results when the algorithm is given a set of inputs. A reliable algorithm should always perform the required task with a high degree of accuracy and consistency.

2.2 Common Computing Algorithms

Different problems require different types of algorithmic techniques to solve the problems. Similarly, different programmers prefer to use different algorithms to solve the same problem. There are many algorithms that have been pre-developed because they are considered fundamental and commonly used in solving daily life problems. Few of them are discussed in the below sections.



2.2.1 Sorting Algorithms:

The sorting algorithms are used to arrange data in a useful manner (Such as ascending or descending order). Various methods of sorting are used for example Insertion sort algorithms, Bubble sort algorithms, etc.

a) Insertion sort algorithm

In Insertion sort, we compare adjacent elements and sort them if they are not in correct order. While comparison, the smallest element is selected and swapped with the element that is placed at first location. This is the first iteration, and we will continue until all elements are sorted. At the end, we have an array of sorted elements.

1. Working of Insertion sort algorithm

Consider the following array of unsorted elements which need to be sorted in ascending order.

For ease let's use four colors to represent four different things in the given array

- Yellow color for unsorted array.
- Blue color for elements to be compared in the unsorted array.
- Green color for sorted array.
- Red color for elements that are not correct in the sorted part of array.

At the start, all elements are colored yellow

12	31	25	8	32	17
----	----	----	---	----	----

Assuming first element (12) is already sorted.

12	31	25	8	32	17
----	----	----	---	----	----

We pick the second element (31) and compare with the adjacent element with sorted array.

12	31	25	8	32	17
----	----	----	---	----	----

Because 31 is larger than 12. It means that first element is already in correct order.

12	31	25	8	32	17
----	----	----	---	----	----

Now, move to the next element (25) and compare it with adjacent element in the sorted array (25 with 31).

12	31	25	8	32	17
----	----	----	---	----	----

Here, 25 is smaller as compared to 31. It means, 31 is not at its correct position, therefore, swap both the elements 25 with 31.

12	25	31	8	32	17
----	----	----	---	----	----

Along with swapping, we also need to check and compare it with all elements in the sorted array. Till now, the sorted array contains one element (12). So, 25 will be compared with 12. Here, 25 is greater than 12. So, the sorted array remains same.

We now move forward to next element that is 8 and compare it with its adjacent element in the sorted array (31).

12	25	31	8	32	17
----	----	----	---	----	----

Here, 31 is greater than 8, so swap them.

12	25	31	8	32	17
----	----	----	---	----	----

After the above swapping, you will note that elements 25 and 8 are not in its correct order.

12	25	8	31	32	17
----	----	---	----	----	----

So, accordingly swap them.

12	8	25	31	32	17
----	---	----	----	----	----

The elements 12 and 8 are also not in its correct order.

12	8	25	31	32	17
----	---	----	----	----	----

So, accordingly swap them.

8	12	25	31	32	17
---	----	----	----	----	----

Now we move to the next elements that is 32 and compare it with adjacent element in sorted array that is 31.

similarly 32 will be compared with previous element in the sorted array there will be no change because they are already sorted.

8	12	25	31	32	17
---	----	----	----	----	----

No swapping is required because they are already sorted.

8	12	25	31	32	17
---	----	----	----	----	----

Here we will take the next element that is 17 and compare it with its adjacent element in sorted array that is 32.

8	12	25	31	32	17
---	----	----	----	----	----

32 is greater than 17. So, swap them

8	12	25	31	32	17
---	----	----	----	----	----

This swapping will make 31 and 17 unsorted.

8	12	25	31	17	32
---	----	----	----	----	----

So, swap them too.

8	12	25	17	31	32
---	----	----	----	----	----

Now, this will make 25 and 17 unsorted.

8	12	25	17	31	32
---	----	----	----	----	----

So, swapping them again.

8	12	25	17	31	32
---	----	----	----	----	----

Now, the sorting process is complete as the array is in correct order.

ii) Insertion Sort in context of Computational Thinking

Let's solve the Insertion sort by applying the computational thinking properties.

Abstraction:

Organize a collection of items by repeatedly picking up each item and inserting it into its correct position relative to the items already sorted. This will gradually build a sorted collection by inserting items in the right order.

Decomposition:

Sub-Processes:

Pick and Insert: For each item, pick it up and insert it into the sorted collection.

Correct Position Logic: Determine the correct position for each item in the sorted collection.

Pattern Recognition:

Repetition: The process involves repeating the steps for each item.

Building Order: The algorithm focuses on building a sorted collection incrementally.

Ordered Comparison: It employs comparisons to determine the correct position of each item.

Algorithmic Design

1. Start with the second element:

Begin with the second element of the array (assuming the first element is already "sorted").

2. Pick the current element:

Pick the current element and remember it.

3. Compare with sorted elements:

Compare the current element with the elements in the "sorted" part of the array.

4. Shift larger elements to the right:

If the current element is smaller than the elements in the "sorted" part, shift those larger elements to the right to create space.

5. Insert the current element:

Insert the current element into its correct position in the "sorted" part.

6. Repeat for each element:

Repeat these steps for each element in the array, gradually expanding the "sorted" part.

b) Bubble sort algorithm

In Bubble Sort algorithm, we repeatedly compare and swap the adjacent elements if they are not in correct order.

If we need to sort the array in ascending order, then bubble sort algorithm starts by taking first element of the array and compare it with the second element first iteration. Suppose if the first element is greater than the second then it will swap these elements, and then move forward to compare the second element with the third element, and it continues until the largest element reaches the last place in the array.

We need to repeat this process (iteration) until the complete list is sorted.

1. Working of Bubble Sort Algorithm

Consider the following array of elements:

13	32	26	35	10
----	----	----	----	----

First Iteration

Take the first element and compare it with its adjacent element

13	32	26	35	10
----	----	----	----	----

Here, the first element (13) is small than the second element (32), so it is already sorted.

Now, the second element and compare it with next adjacent element (e.g. 32 with 26).

13	32	26	35	10
----	----	----	----	----

Here, 32 is greater than 26. So, we need to swap them both. After the swapping, array will look like this:

13	26	32	35	10
----	----	----	----	----

Now, take third element and compare it with its next adjacent (e.g. 32 with 35). No swapping is required because 35 is greater than 32

13	26	32	35	10
----	----	----	----	----

Now, we will compare 35 with 10.

13	26	32	35	10
----	----	----	----	----

Here, 35 is greater than 10 and need swapping.

The element 35 has reached at the last place of the array. At the end of first iteration, the array will look like this:

13	26	32	10	35
----	----	----	----	----

Now, its time for second iteration.

Second Iteration

In second iteration, we will follow the same process.

13	26	32	10	35
----	----	----	----	----

13	26	32	10	35
----	----	----	----	----

13	26	32	10	35
----	----	----	----	----

Here, 32 is greater than 10. So, perform swapping operation.

13	26	10	32	35
----	----	----	----	----

13	26	10	32	35
----	----	----	----	----

Similarly, next iteration will be done.

Third Iteration

In third iteration, we will follow the same process again.

13	26	10	32	35
----	----	----	----	----

13	26	10	32	35
----	----	----	----	----

Here, 26 is greater than 10. So, perform the swapping again.

13	10	26	32	35
----	----	----	----	----

13	10	26	32	35
----	----	----	----	----

13	10	26	32	35
----	----	----	----	----

Now, move to next iteration.

Fourth Iteration

Again follow the same process the array is completely sorted now.

10	13	26	32	35
----	----	----	----	----

ii. Bubble Sort in context of Computational Thinking

Let's solve the Bubble sort by applying the computational thinking properties.

Abstraction:

Organize a collection of items by repeatedly comparing and swapping adjacent items until the entire collection is sorted. It will perform iterative comparison and swapping to achieve order.

Decomposition:

Sub-Processes:

1. Adjacent Comparison: Compare each pair of adjacent items.
2. Swap Logic: If a pair is out of order, swap them.
3. Iteration: Repeat the process until the collection is sorted.

Pattern Recognition:

Pairwise Comparison: The algorithm focuses on comparing and adjusting adjacent pairs.

Iterative Process: The sorting process is achieved through multiple iterations.

Repeating Patterns: The comparison and swapping steps are repeated until the entire collection is in order.

Algorithmic Design

1. Start from the first element of the array.
2. Compare the first element with the adjacent element.
 - If the first element is greater than the adjacent element, swap them.
 - If the first element is smaller or equal, do nothing and move to the next pair of elements.
3. Move to the next pair of elements and repeat the comparison and swapping process.
 - Continue this process until you reach the end of the array for the first iteration.
4. After the first iteration, the largest element is guaranteed to be at the end of the array.
 - You don't need to consider this element in the subsequent passes since it is already in its correct position.
5. Repeat the above steps for the remaining elements (excluding the ones that are already sorted).
 - In each iteration, the next largest unsorted element will be placed in its correct position.
6. Continue this process until the entire array is sorted.
 - The number of iterations required is one less than the total number of elements in the array.

Both Insertion Sort and Bubble Sort are basic sorting algorithms. The choice of sorting algorithm depends on the specific requirements of your problem. Following are some generic guidelines

that could be helpful while selecting any sorting algorithm. However, there are some advanced sorting algorithms available that you would probably study in your higher class.

Table 2.2- Insertion Vs. Bubble Sort		
When to use	Insertion Sort	Bubble Sort
	Use for a small number of items	Use for a small number of items
	nearly sorted data	nearly sorted data
	Practical applications	Educational purposes Educational purposes

Example

Sorting a Shelf of Books: Imagine you have a shelf of books that you want to arrange in alphabetical order. Compare different methods:

- **Insertion Sort:** Pick up each book one by one. Insert each book into its correct position on the shelf relative to the books you're already holding.
- **Bubble Sort:** Compare each pair of adjacent books from left to right. If a pair is out of order, swap their positions. Continue this process, moving from left to right, until the entire shelf is sorted.

Algorithm Evaluation -> Efficiency: The time it takes and the number of movements represent the efficiency of the sorting algorithm.

2.2.2 Searching Algorithms:

The searching algorithms are used to search desired element that is available in some group of elements. Like sorting, various searching methods are available for example Binary Search algorithms and Linear Search algorithms.

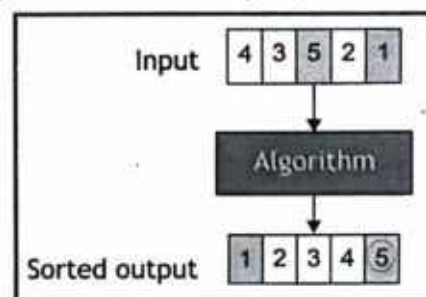
a) Binary Search algorithm

This search method works on sorted lists. It follows the divide and conquer approach in which the sorted list is divided into two parts. The item to be searched is then compared with the middle element of the list. If it matches then, the location of the middle element is returned. Otherwise, we search into one of the divided parts depending upon the result produced in the last comparison.

i. Working of Binary Search

Consider the following sorted array of nine elements and we need to find a number 56.

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69



Divide the array in to two parts, the middle element is 39.

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69



The item to be searched (56) is compared with the middle value (e.g. 39). we came to know that 56 is greater than 39, therefore the item is surely in the right side of the array (because the array is sorted).

Here we get right part of the array.

5	6	7	8
40	51	56	69

It will again be divided into two parts.

5	6	7	8
40	51	56	69



Considering 51 the middle element, we will compare the item (56) with this, because 56 is greater than 51 therefore, we will again take right side of the array.

7	8
56	69

Divide the array again into two parts.

7	8
56	69



Considering 56 as the middle element, we will compare it with the item to be searched (e.g. 56). Now, the required item is found, So we will provide output of the algorithm that is location 7. If the required item is not found the algorithm returns NULL.

ii) Binary Search in context of Computational Thinking

Let's solve the Binary Search by applying the computational thinking properties.

Abstraction:

Search for a specific item in a sorted collection by repeatedly dividing the search space in half until the target item is found or the search space is empty. It efficiently narrow down the search space by half in each iteration.

Decomposition:

Sub-Processes:

1. **Middle Item Check:** Examine the middle item to determine if it matches the target.
2. **Search Space Division:** Divide the search space based on the comparison result.
3. **Recursive Process:** Repeat the process on the narrowed down search space.

Pattern Recognition:

Halving the Search Space: The core concept is repeatedly reducing the search space by half.

Algorithmic Design

// Assume target value is the element that needs to be searched

1. Start with the entire sorted array.
2. Check the middle element of the array.
 - If it's equal to the target value, you're done.
 - If it's greater than the target value, ignore the right half.
 - If it's less than the target value, ignore the left half.
3. Repeat the process on the remaining half.
4. Continue until you find the target value, or the array becomes empty.

b) Linear Search algorithm

This is a simple search algorithm; we go through the complete list and match the elements one-by-one until the required item is found. If the item is found, then the location of that element is returned, otherwise a NULL is returned. This search is also called as sequential search algorithm. However, there is no compulsion that the array should be sorted.

1) Working of Linear Search

Consider the given unsorted array and we need to find location of item 41.

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	56

We will start from first element.

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	56



The required item (41) is compared with the first element. Both don't match, we will go to next element.

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	56



We will again compare the second element (e.g. 40) with required item (e.g. 41), both don't match. So will continue the process.

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	56



0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	56

↑

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	56

↑

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	56

↑

We found the element at location 5 so output of this algorithm would be 5. In these type of algorithms, if the required item is found we provide the output or if item is not found and we reach at the end of array then a NULL is given in output.

Let's solve the Linear search by applying the computational thinking properties.

Abstraction:

Search for a specific item in a collection by checking each item in order until the target item is found or the end of the collection is reached. It sequentially inspects each item until a match is found.

Decomposition:

Sub-Processes:

1. Starting Point: Begin searching from the first item.
2. Item Checking: Inspect each item to see if it matches the target.
3. Stopping Condition: Stop when the target is found or the end of the collection is reached.

Pattern Recognition:

- Sequential Nature: The process involves moving through the collection one item at a time.
- Linear Progression: The search progresses linearly through the collection.
- Termination Condition: The search stops when the target is found or the entire collection is checked.

Algorithmic Design

// Assume target value is the element that needs to be searched

1. Start from the beginning of the array.
2. Check if the current element is equal to the target value.
 - If it is, you've found the target value.
 - If it's not, move to the next element in the sequence.
3. Repeat step 2 until the target value is found or you reach the end of the array.
4. If you reach the end of the array without finding the target value, the target value is not in the array.

The choice of searching algorithm depends on the specific requirements of your problem. However, following are some generic guidelines that could be helpful while selecting any searching algorithm:

Table 2.3- Linear Vs Binary Search		
When to use	Linear Search	Binary Search
	When the list is not sorted	When the list is sorted
	Suitable for a smaller number of elements	Ideal for larger number of elements

Example

Movie Streaming: Choosing a movie to watch from a list:

- Linear Search: Start from the top and scroll down until you find the movie.
- Binary Search: Divide the list in half, quickly narrowing down the options.

Algorithm Evaluation -> Efficiency: The time it takes to find a movie reflects the efficiency of the search algorithm.



Teacher's Guide

"CodeCombat: Learn to Code by Playing a Game"

<https://codecombat.com/>

This resource transforms the learning experience into a game, where students can grasp programming concepts through an entertaining role-playing adventure. The game effectively introduces fundamental coding logic, which can be applied to comprehend search and sort algorithms.

Summary

- **Abstraction:** Identifying essential information and removal of the unnecessary details to solution.
- **Algorithm** is a step-by-step set of instructions that define how a specific problem should be solved.
- **Algorithm Design:** This is actual designing of solution. This involves creating step-by-step plan of the problem solution.
- **Algorithm Evaluation** assess the algorithm performance, efficiency, and correctness for specific tasks.
- **Algorithm Tracing** is used to hand simulate the execution of your algorithm, keep change track of variable values and flow of logic control.
- **Binary Search:** Sorted list is divided into two parts. The item to be searched is then compared with the middle element of the list. If it matches then, the location of the middle element is returned. Otherwise, we search into one of the divided parts depending upon the result produced in the last comparison.
- **Bubble Sort:** we repeatedly compare and swap the adjacent elements if they are not in correct order. Bubble sort algorithm starts by taking first element of the array and compare it with the second element. Suppose if the first element is greater than the second then it will swap these elements, and then move forward to compare the second element with the third element, and it continues until the largest element reaches the last place in the array.
- **Computational Artifacts** refer to human made objects and systems using computational thinking.
- **Computational Thinking** is a thought process that helps us to understand the problem and solve it in a way that computers do.
- **Decomposition:** Breaking down the larger problems into smaller/ manageable ones and working on them one by one. These smaller problems are referred to as sub-problems. This way we simplify the problem and solve it easily.
- **Development** describes the steps needed to convert or manipulate the inputs to produce the outputs.
- **Flowchart** provides a visual representation of flow and logic using defined symbols.
- **Insertion sort,** we compare adjacent elements and sort them if they are not in correct order. While comparison, the smallest element is selected and swapped with the element that is placed at first location.
- **Linear Search:** we go through the complete list and match the elements one-by-one until the required item is found. If the item is found, then the location of that element is

returned, otherwise a NULL is returned. This search is also called as sequential search algorithm. However, there is no compulsion that the array should be sorted.

- Pattern Recognition: Examine the problem for a pattern or similarities between previously solved problems.
- Pseudocode is an intermediate step between the algorithm and actual programming code.
- Searching algorithms are used to search desired element that is available in some group of elements.
- Sorting Algorithms are used to arrange data in a useful manner (Such as ascending or descending order).
- Trace Tables: The dry run of an algorithm can be done with a technique known as Trace Tables.

Exercise



Select the best answer for the following Multiple-Choice Questions (MCQs).

1. A complexity of algorithm depends upon _____
 - a) Time Only
 - b) Space Only
 - c) Both Time and Space
 - d) None of above
2. An algorithm: can be represented through _____
 - a) Flow charts
 - b) Pseudo-codes
 - c) Instructions in common language
 - d) All of the mentioned above
3. There are two algorithms suppose A takes 1.41 milliseconds while B takes 0.9 milliseconds, which one of them is better considering all other things the same?
 - a) A is better than B
 - b) B is better than A
 - c) Both are equally good
 - d) None of the mentioned
4. Considering an array has 10 elements and the searching element is at array index 6. A starting element is present at index zero. How many comparisons are required to search and element using linear search?
 - a) 5
 - b) 6
 - c) 7
 - d) 8
5. In computer science, the computational artifacts could include _____
 - a) Programs
 - b) Simulations
 - c) Videos
 - d) Programs, Simulations and videos
6. In the planning phase of computational artifact development following is included in
 - a) Logic
 - b) Data Structures
 - c) Control Structures
 - d) Logic, Data and Control Structures
7. trace method is used to _____
 - a) Take input
 - b) Hand simulate the execution algorithm.
 - c) Take a print
 - d) Align margins
8. Trace Tables are used to _____
 - a) dry run algorithm
 - b) test if algorithm is giving expected output
 - c) show the variables change
 - d) a, b and c

9. Algorithms are evaluated using _____
- a) Correctness
 - b) Efficiency
 - c) a and b
 - d) None of above



Give short answers to the following Short Response Questions (SRQs).

1. Differentiate between
 - i Clarity vs. Efficiency
 - ii Abstraction vs. Pattern Recognition
 - iii Pseudocode vs. Flowcharts
 - iv Data Structures vs. Control Structures
 - v Algorithm vs. Pseudocode
2. Write a note on working of Bubble Sort.
3. Write names of commonly used computational artifacts made during computational thinking.
4. Where we prefer to use binary search algorithm rather than linear search algorithm?
5. What are the advantages of using flowcharts?



Give long answers to the following Extended Response Questions (ERQs).

1. Determine the properties involved in computational thinking.
2. Sketch an algorithm that inputs length in inches and prints it in centimetres.
3. Implement an algorithm to print multiplication table of a number in reverse order.
4. Examine the uses of Flowcharts.
5. A newly developed Algorithm needs to be tested. Argue about the reasons.



Activity 1: Complete the given trace table using the following algorithm snippet.

1. Start
2. $x = 2$
3. $total = 0$
4. REPEAT Step 5 Three TIMES
5. $total = total + x$
6. Print (total)
- End

Line No.	x	total	output



Activity 2: Insertion sort algorithm

Sort the following array using insertion sort

10	4	5	8	6	9	2
----	---	---	---	---	---	---



Activity 3: Bubble sort algorithm

Sort the following array using bubble sort

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



Activity 4: Binary Search algorithm

Consider the following sorted array of seven elements and we need to find a number 9.

2	4	5	6	8	9	10
---	---	---	---	---	---	----



Activity 5: Linear Search algorithm

Consider the following array of eight elements and we need to find a number 9.

58	25	39	78	12	9	79	80
----	----	----	----	----	---	----	----