

2

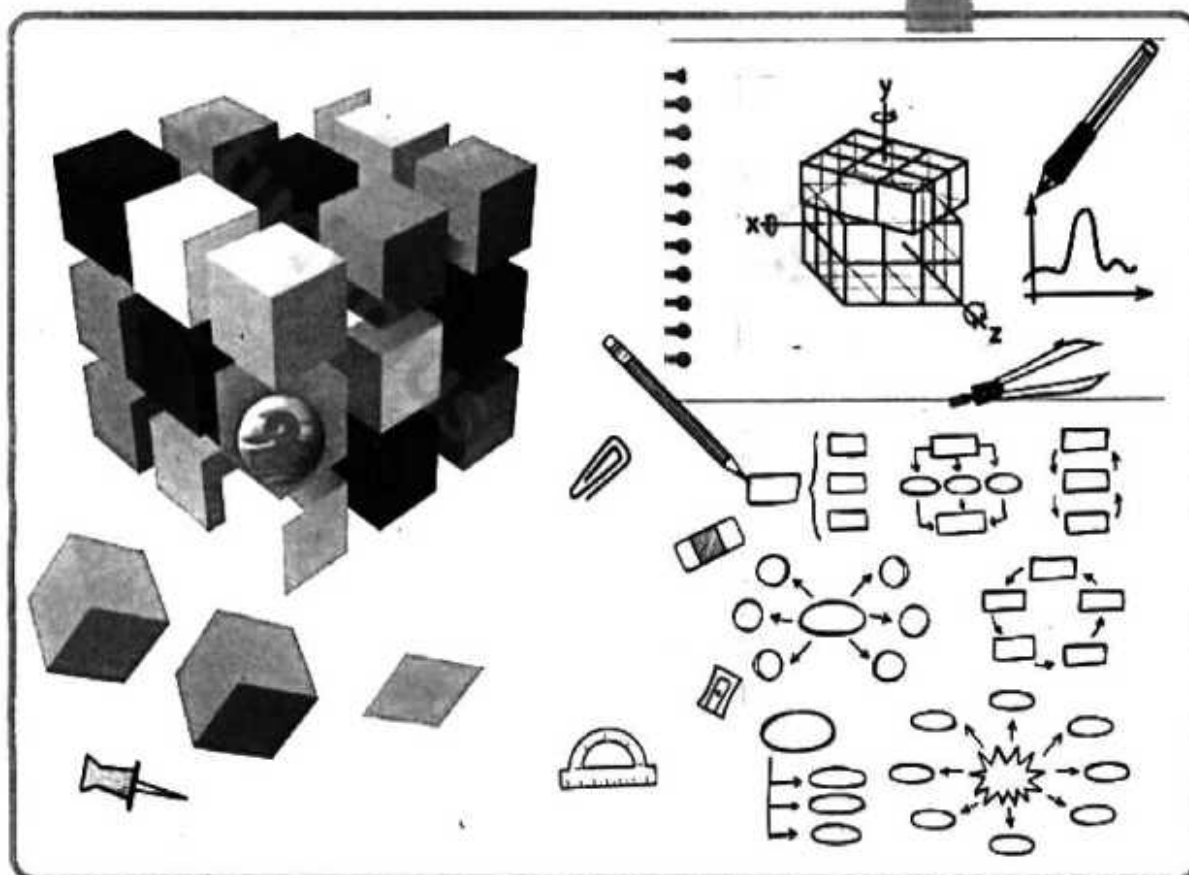
Computational Thinking & Algorithms



Learning Outcomes

At the end of this unit students will be able to:

- identify common algorithms used to develop software, store, search, or sort information.
- develop and apply abstractions to create generalized, modular solutions.



Introduction

Computational Thinking (CT) is a way of solving problems by using ideas from computer science. It helps you break down big problems into smaller parts and come up with step-by-step solutions that a computer or a person can follow.

CT isn't just about coding; it's a basic skill useful in any area. It involves using mental tools and strategies to think more clearly, logically, and creatively.

Computational thinking simplifies difficult issues into doable stages. It teaches you to think through many strategies and create logical answers, or algorithms. Critical thinking encourages innovation and efficient use of technology to solve issues in any industry.

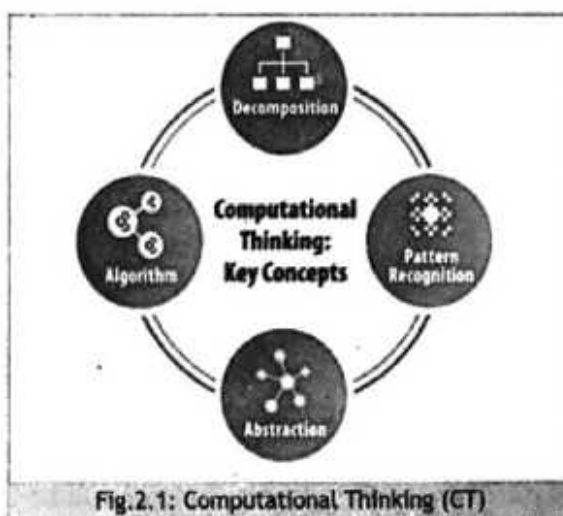


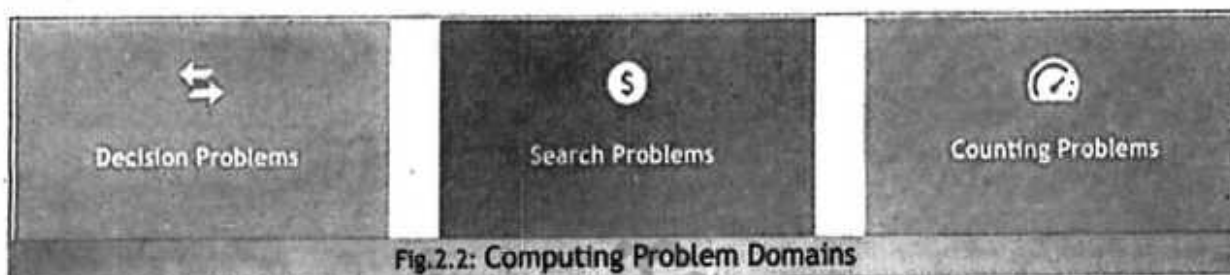
Fig.2.1: Computational Thinking (CT)

2.1 Computing Problems

A computing problem in computer science is one that is solved step-by-step through computation. Any kind of calculation, including arithmetical and logical ones, can be included. These problems often involve an input that is well-defined and an output that needs to satisfy a few desired properties. Some examples include:

- Finding the largest number in a list.
- Checking if a number is even or odd.
- Simulating a scientific phenomenon.

Some of the computing problem domains are as follows:



Decision Problems

In decision problems normally 'yes-or-no' response is found in a choice. For example, given a number n , "is n even?"

However, some decision problems may require more steps to solve. For example:

- Given a number n , "is n prime?" requires more work than just checking number's parity.
- Given a string, is its length greater than 5 characters?
- Given an integer n , is n an even number?



Fig.2.3: Decision Problems

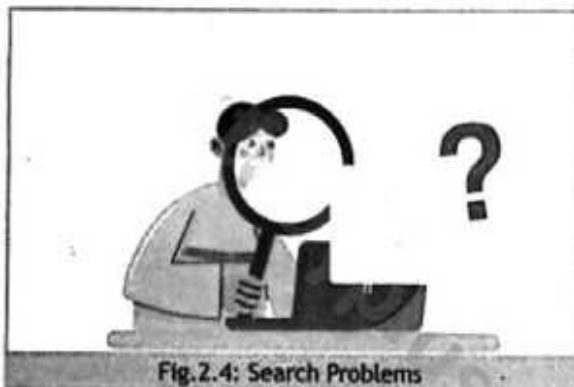


Fig.2.4: Search Problems

Search Problems

In search problems, the solution is made up by having a set of values that satisfy a certain criteria. For example:

- we might wish to calculate a route on a map from one site to another (e.g. from Islamabad to Lahore)

Counting Problems

In counting problems, the number of possible solutions to a search problem is the answer of that counting problem. For the above-mentioned search problem "we might wish to calculate a route on a map from one site to another." The answer of counting problem would be two, supposing we have two roads from Islamabad to Lahore e.g. G.T. Road and Motorway.



Fig.2.5: Counting Problems

2.2 Basics of Counting Problems

The foundation of counting problems is the basic method of counting while considering the options available to select appropriate items for a given selection.

Tree-based structural approaches are the most effective for solving Counting Principle problems. Counting Principle Problems could easily be understood by the use of tree representation, where the available options are considered as tree branches.

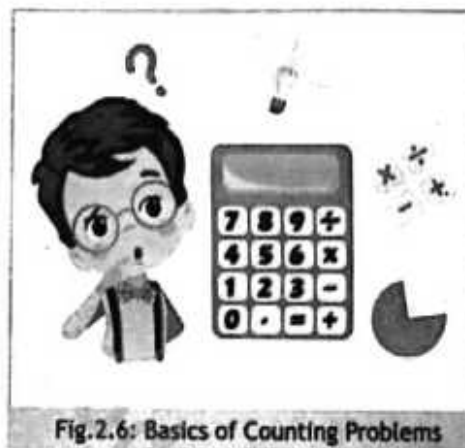


Fig.2.6: Basics of Counting Problems

2.3 Basic Counting Principles

Addition

The addition principle of counting is a basic idea for counting the total number of possibilities that can occur when dealing with many events or options that are mutually exclusive. The events that are mutually exclusive are those that cannot occur simultaneously. This rule is used to get the total number of ways that an event can occur. It is typically used when you have to make a choice between multiple options.

The Addition principle is written as:

Total Possibilities = Number of Possibilities for Event A +
Number of Possibilities for Event B + ... + Number of Possibilities for Event N

For example, a student wants to buy a new computer. He has to decide among 3 desktop computers and 4 laptop computers.

Total number of computer options = Total options in desktop computers + Total options in laptop computers

So in this case

Total number of computer options = $3+4 = 7$

Multiplication

The Multiplication Principle is used when we are dealing with many events, and we have to select one option from each event.

For example: Bilal has 3 Apples, 3 Bananas and 3 bunch of Cherries. In how many ways, he can make fruit basket by putting fruit of one kind.

(A1 B1 C1) (A1 B1 C2) (A1 B1 C3) (A1 B2 C1) (A1 B2 C2) (A1 B2 C3) (A1 B3 C1) (A1 B3 C2) (A1 B3 C3)
(A2 B1 C1) (A2 B1 C2) (A2 B1 C3) (A2 B2 C1) (A2 B2 C2) (A2 B2 C3) (A2 B3 C1) (A2 B3 C2) (A2 B3 C3)
(A3 B1 C1) (A3 B1 C2) (A3 B1 C3) (A3 B2 C1) (A3 B2 C2) (A3 B2 C3) (A3 B3 C1) (A3 B3 C2) (A3 B3 C3)

Using Counting Principle Problems, the total number of ways of choosing this pairing would be:

- Options available for Apples (A) = 3
- Options available for Bananas (B) = 3
- Options available for Bunch of Cherries (C) = 3

Total no. of ways: $3 \times 3 \times 3 = 27$



Fig.2.7: Addition



Fig.2.8: Multiplication

Permutation

Many different kinds of problems can be resolved using the Multiplication Principle. Putting objects in order is one kind of challenge. We queue up for pictures, arrange letters into words and digits into numbers, decorate rooms, and more. A permutation is a way to arrange items.

Drawing line segments for each choice can be useful in solving permutation problems. This allows us to multiply by figuring out how many of each option there are.

Assume we have four paintings and three places on the wall to display. We wish to determine how many different ways there are to arrange three of them on a wall. To symbolize the three locations on the wall, we can draw three lines.

_____ x _____ x _____

From these three locations, for the first location have 4 paintings, so we will write 4 on the first location.

4 x _____ x _____

When we have placed the first painting, for 2nd location, we have three choices, so we will write 3 on the second location.

4 x 3 x _____

Similarly, for the 3rd location, we have left with 2 choices, so we will write 2 on the third location.

4 x 3 x 2

So multiplying, these values gives us the answer of 24, means there are 24 permutations possible for the paintings.

Combination

In the above theories and their examples, we have looked at problems where we have to arrange objects in a certain order. There are numerous situations where we wish to pick a few items at random from a collection of things, without regard to the order. The situation, when we are choosing objects and the sequence is irrelevant, this is called combination counting principle.

Consider the above example of paintings that we solved in permutation. So without considering the paintings' order, if we want to determine how many options there are to select three of the four paintings.

Combinations = number of permutations divided by the number of ways to order 3 paintings.

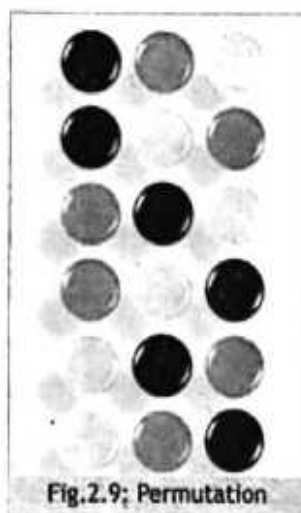


Fig.2.9: Permutation



Fig.2.10: Combination

There are $3! = 3 \cdot 2 \cdot 1 = 6$ ways to order 3 paintings, Hence, there are $24/6 = 4$ combinations that defines the no of ways to select 3 out of the 4 paintings.

Pigeonhole Principle

If there is a flock of 10 pigeons and we have 09 pigeonholes, means one of the pigeonholes must have more than 1 pigeon.

The principle is stated as:

There are n boxes (n is a positive integer) and $n + 1$ objects are to be placed into these n boxes, then at least one box contains two or more objects.

Example

Suppose you have 13 apples and 12 baskets. According to the Pigeonhole Principle, at least one basket will contain more than one apple.



Fig.2.11: Pigeonhole Principle

Inclusion and Exclusion

Recall, the addition principle that states if a task can be performed in one of the n ways or one of the m ways (with no common of two $n+m$), then there are total $n+m$ ways to the task.

But if there are some ways in n and m to do the task, then according to the inclusion-exclusion principle, we must sum the number of ways to complete a task one way and the number of ways to complete it another, and then subtract the number of ways that are shared by both sets of ways in order to count only unique ways of completing the work.

Example:

How many binary strings of length eight have two bits at the end, "00," or begin with a "1" bit?

There are $2^6 = 64$ possibilities to fill in 6 bits if the string ends in '00'.

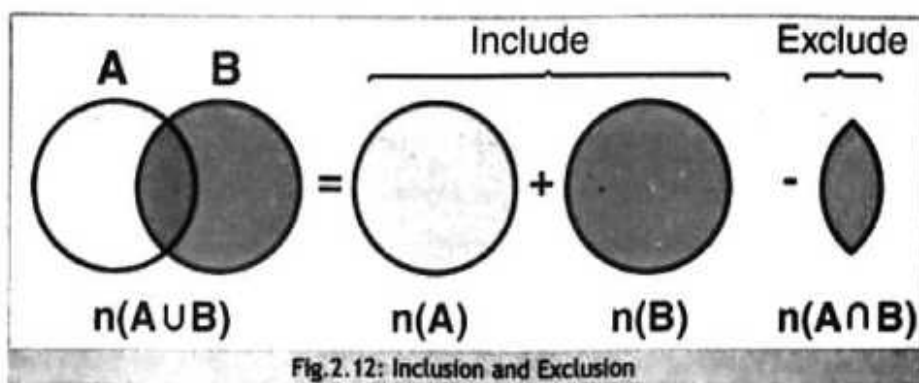
Similarly, if the string begins with '1', then there are seven bits left in the string and there are $2^7 = 128$ possibilities to fill it.

If add both of the aforementioned sets and consider it the whole solution it will be incorrect according to Inclusion and Exclusion principle of counting because there are strings that both begin with '1' and end in '00', and since they meet both requirements, they are counted twice.

In order to obtain an accurate count, we must subtract these strings.

There are $2^5 = 32$ possibilities to fill the five characters in strings that begin with '1' and end with '00', accordingly, the inclusion-exclusion principle gives us:

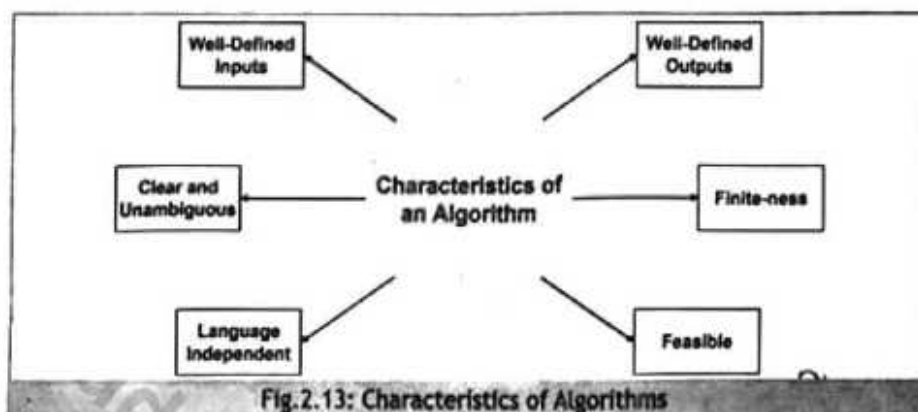
total String = $128 + 64 - 32 = 160$.



2.4 Algorithms and their Characteristics

An algorithm is a well-defined, step-by-step procedure or set of rules for solving a specific problem or performing a task.

Computer science relies heavily on algorithms, which are used for processing data, automating reasoning activities, and doing calculations. They can be expressed in a variety of ways, including computer languages, flowcharts, pseudocode, and natural language.



Properties of Algorithms

- **Input:** An algorithm can have one or more externally supplied values as inputs.
- **Output:** The outcome of the calculations made by an algorithm is at least one output.
- **Definiteness:** Every algorithmic step needs to be clearly and accurately stated.
- **Finiteness:** There must always be a finite number of steps in an algorithm before it ends.
- **Effectiveness:** An algorithm's operations must be simple enough for a human to execute them with paper and pencil in a reasonable amount of time.
- **Generality:** An algorithm should be able to handle a group of problems rather than simply one particular one. It needs to offer a broad solution for several related problems.

Role of Algorithms in Computational Problem-Solving

In computational problem-solving process, the algorithms are considered essential, because they provide a set of instructions to follow before actually writing the code in a programming language. The following points highlight their importance:

-
- Algorithms are designed to be efficient in terms of time and space.
 - Algorithms automate repetitive and tedious tasks
 - Algorithms help in analyzing and breaking down complex problems into manageable steps.
 - Once an algorithm is designed and tested, it can be reused for similar problems.
 - Algorithms provide precise and accurate solutions
 - Algorithms are considered foundation of programming and software development.

Algorithm Applications

The algorithms can be applied to solve various problems such as

- **Sorting and Searching:** For organizing and retrieving data efficiently, algorithms like quicksort, mergesort, and binary search are used.
- **Graph Algorithms:** For finding the shortest paths in navigation applications, Dijkstra algorithm could be used.
- **Cryptography:** TO provide privacy and security in online transactions, such algorithms be used to encrypt and decrypt data.
- **Machine Learning:** To learn from data and make predictions, machine learning algorithms like decision trees, neural networks can be used.
- **Image Processing:** To enhance image quality or compress the image to reduce size or even to analyze images, Image processing algorithms could provide help. These have very critical role in medical imaging, computer vision, and multimedia applications.

2.5 Logical Reasoning

Logical reasoning is the core part of algorithm development. Logical reasoning fits into computational thinking in the following ways:

- **Problem Decomposition:** Logical reasoning is needed to guarantee that each component is treated systematically when breaking large problems down into smaller, more manageable portions.
- **Pattern Recognition:** Logical sequences and linkages are frequently recognized in order to identify patterns in data or processes.
- **Abstraction:** Logical thinking is needed to discern between pertinent and extraneous information in order to simplify complicated problems by concentrating on the most important features.
- **Algorithm Design:** To guarantee the accuracy and effectiveness of the process, logical thinking is a crucial component of creating step-by-step solutions, or algorithms, to issues.

Boolean Logic

In Boolean, two statements or expressions are evaluated in a way that can only have one of two potential values as a result—either true or false. Boolean logic uses logical operator like AND, OR, NOT, etc.

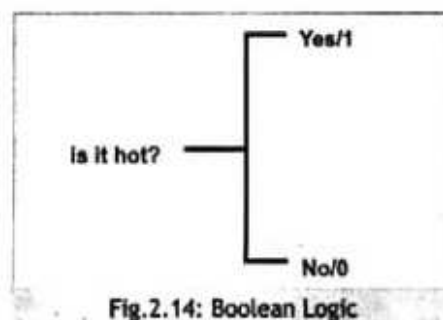


Fig.2.14: Boolean Logic

The above diagram is evaluating an expression regarding today's weather "Is it hot", so it can have either answer 'Yes it is hot - represented with 1' or 'No it is not hot - represented with 0'. Following are few of the most widely used operators with their Venn diagram and truth table.

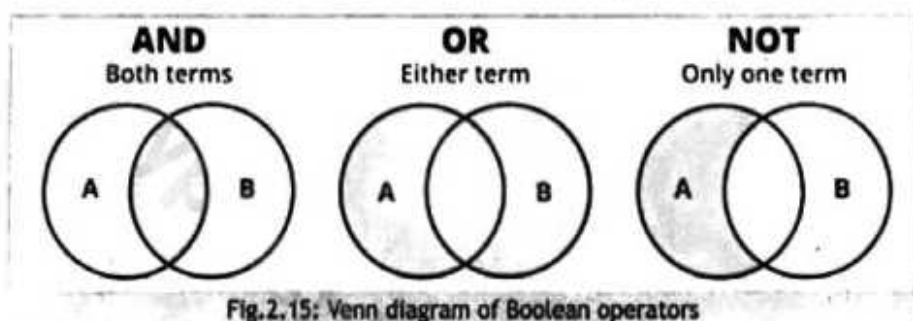


Fig.2.15: Venn diagram of Boolean operators

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Table 2.1: Truth Table of Boolean operators

Logical Reasoning Types

The logical reasoning questions of wither verbal or non-verbal, the concepts and issues in verbal logical reasoning questions are stated verbally. To attempt such questions, first the provided language or paragraph must be read, comprehend it, and select the appropriate response from the list of possibilities. However, in non-verbal logical reasoning questions, the concepts and issues are presented using figures, pictures, or diagrams.

Example

Among three colors (red, blue, and green), Aiza, Uzair, and Luqman each have different favorite

colors, Use the clues below to determine each person's favorite color.

- »» Aiza does not like green
- »» Uzair's favorite color is blue
- »» Luqman does not like red

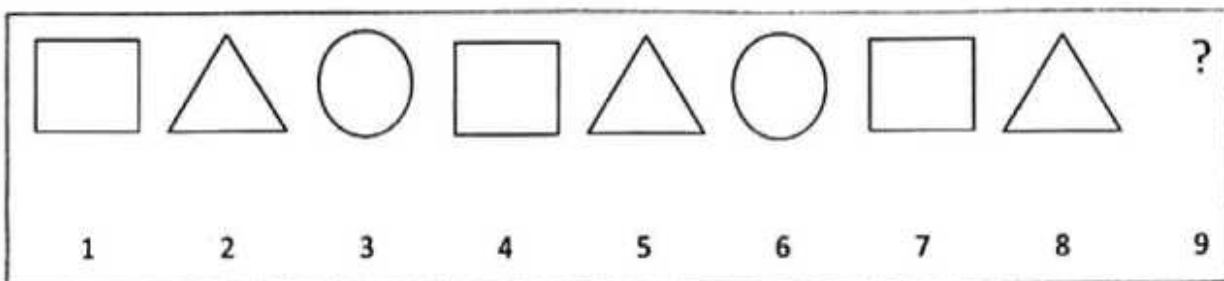
From second Clue, it is clear that Uzair's favorite color is BLUE, therefore other two persons can't have blue as their favorite color.

From first clue, we came to know that Aiza does not like green and blue color is already the favorite of Uzair, so left option is only RED

Now the only left over option is green, this also does not contradict the third clue. So GREEN is the final option for Luqman.

Example

Look at the sequence of shapes below and determine which shape comes next.



To solve this, observe the pattern in the sequence, We have a a repeating pattern of Square, Triangle, Circle. This pattern repeats every three shapes.

Following the pattern:

- After Square (1), Triangle (2), Circle (3)
- It repeats with Square (4), Triangle (5), Circle (6)
- Again, Square (7), Triangle (8)

The next shape in the sequence, after Triangle, should be a Circle. So, the shape that comes next is a Circle.

2.6 Standard Algorithms

In computational thinking, standard algorithms are fundamental methods used to solve common problems. Understanding these algorithms helps to develop problem-solving skills and provides a foundation for more complex programming tasks. Following are some standard algorithms:

Sorting Algorithms

These algorithms are used to arrange data in a specific order (ascending or descending)

Bubble Sort: Repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.

Selection Sort: Divides the list into a sorted and an unsorted part, and repeatedly selects the smallest (or largest) element from the unsorted part to move to the sorted part.

Merge Sort: Divides the list into halves, recursively sorts each half, and then merges the sorted halves.

Insertion Sort: An in-place sorting algorithm that builds the final sorted array one element at a time, shifting elements greater than the current element to the right.

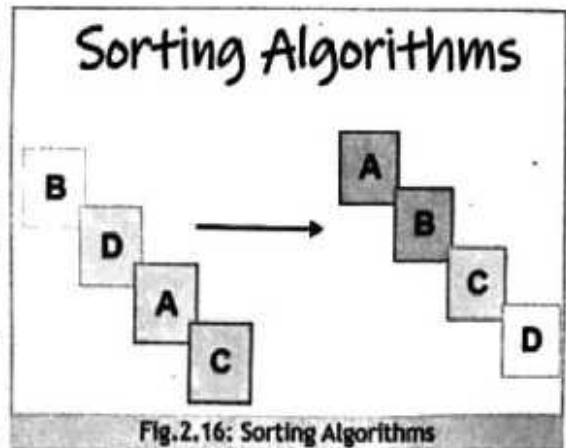


Fig.2.16: Sorting Algorithms

Searching Algorithms

Linear Search: A searching algorithm that sequentially checks each element in a list until the target element is found or the end of the list is reached.

Binary Search: A searching algorithm that works on sorted arrays by repeatedly dividing the search interval in half. It efficiently finds the position of a target value within the array.

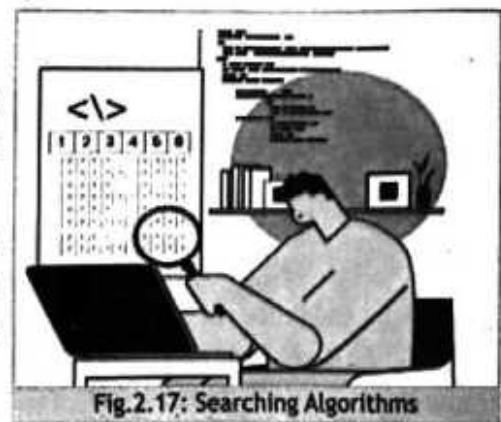


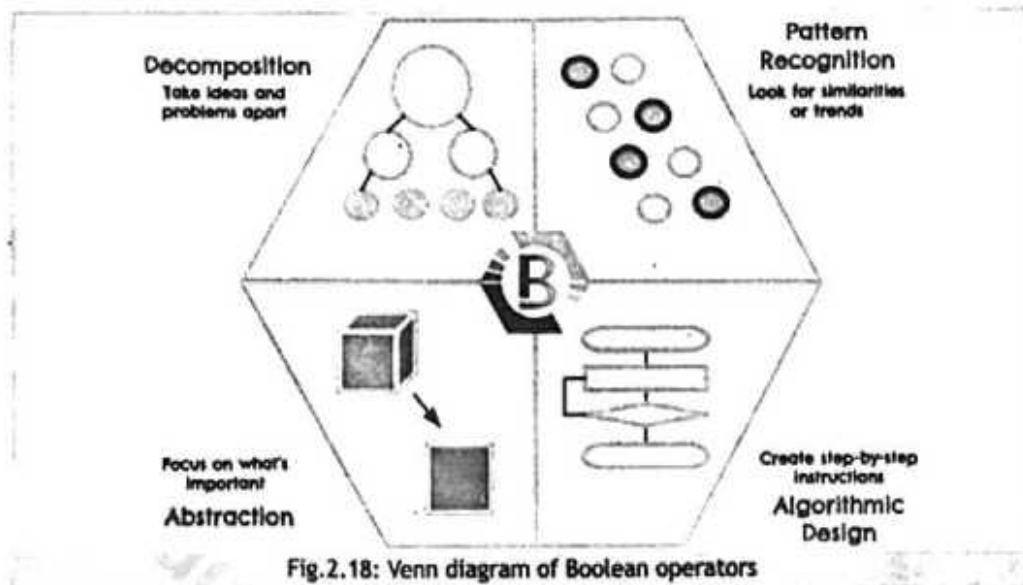
Fig.2.17: Searching Algorithms

2.7 Steps in algorithm to solve computational problems

Core Concepts of Computational Thinking

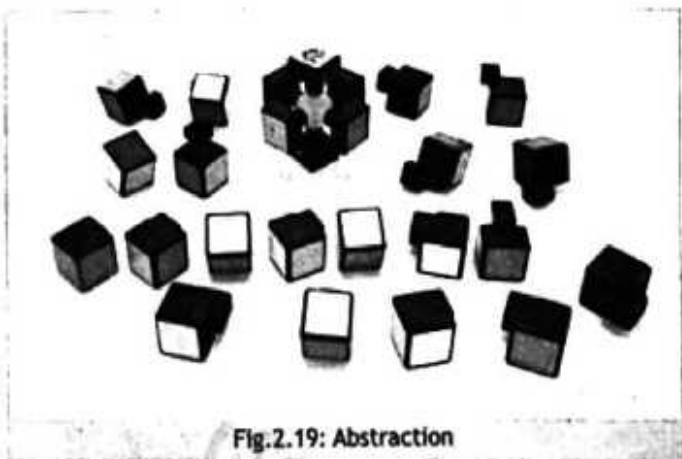
These core concepts collectively help in designing systems and solving problems efficiently and effectively in various fields. These concepts are:

- **Decomposition:** The problem that need solution could be a complex problem, that may not be solved all at once. Therefore, in decomposition, the complex problems are broken down into sub problems and these sub problems are solved one at a time. By this process, it becomes understandable and simple.
- **Pattern Recognition:** We solve different problems in daily life, but when we analyse, many problems have similar pattern. Identification of these similarities and differences in data or problem helps you to make a general solution.
- **Abstraction:** Focusing on the important information only and ignoring irrelevant details. Abstraction helps in reducing complexity and makes it easier to develop solutions.
- **Algorithms:** Algorithms are the step-by-step solution of a problem. Each step defines some rule to follow that can solve the problem. For automation and efficiency of a system Algorithms are considered essential part of problem-solving.



2.8 Abstraction

While solving problems, specifically complex problems, the concentration should be on core idea and unnecessary components should not be given more attention. By doing this, we are making our problems simple to solve and easy to understand.



- **Maps:** For example, when we are using maps application to reach to some destination, we view the map abstractly, focusing on roads and landmarks and eliminate the minor details like specific plants, trees or houses.
- **Symbols:** We being a smartphone user, considers the icons as applications and does not care about the complex code and procedures that are used to build these applications.
- **Models:** For example, when we plan to construct a house, we only consider architectural models that represents simple depictions of structures. By doing this we can easily communicate design without going into little details..

Let us consider some examples of Abstraction in the filed of Computing

- **Programming Variables:** In our programming chapter, we have dealt with data, and it is abstractly represented by using different variables. Rather than, working with constants we use variables that make our solution generic and easier.
- **Functions:** To make the programming job easier, we are given many built-in functions. These functions are identified using some names and the programmer only call that function by their names, rather than going into detail of activities/ procedures that this function contain. At

advanced level, the programmers also develop their own functions to be used at some later stage in their programming jobs.

- **Object-Oriented Programming (OOP):** In Object Oriented Programming, actual entities are represented with objects. By doing this we conceal the complex details of their implementation.

Example

Steps to Make a Cup of Tea:

1. Boil water.
2. Put tea bag into to the cup.
3. Fill the cup with boiled water.
4. After some time, remove tea bag from the cup.
5. Add sugar (optional).
6. Add Milk (optional).
7. Stir for sometime, Tea is ready.

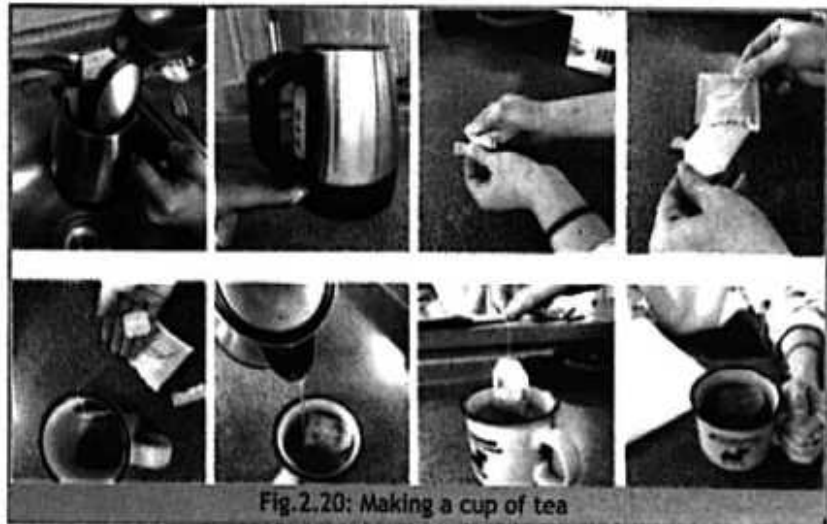


Fig.2.20: Making a cup of tea

2.9 Creating Generalized Solutions

The problems could be specific, but the best solution needs to be generic. The specific solution could only handle that particular situation that is being faced at any particular time. However, the generic solution could be any time to handle similar problems. This way, we can enhance efficiency, scalability and reusability of our solutions.

Steps to Create Generalized Solutions

1. Identify the Core Problem:

- To understand the core problem, first you need to understand specific cases of problems. Among these specific cases of problems, identify the essential components that are common across different cases.

2. Extract Common Patterns:

- Identify Repeated patterns, In problems, there are some elements that occur periodically, identify such elements. Bifurcate the elements whether they are specific to the examples or could be applied more broadly.

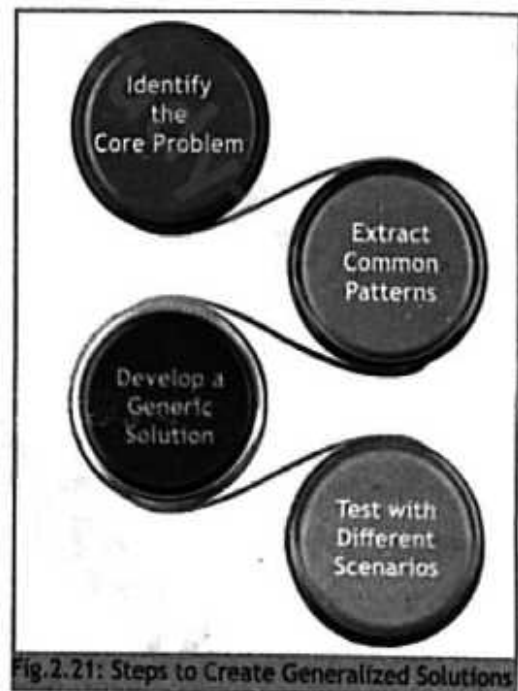


Fig.2.21: Steps to Create Generalized Solutions

3. Develop a Generic Solution:

- » Create such a solution that addresses both core and common elements of the problem. It is better to use variables instead of constants, this will help you testing various inputs or conditions.

Test with Different Scenarios:

- » Test the solution with a wide range of inputs to ensure validity of solution in different scenarios. If needed, modify your solution based on test results.

Example: Sorting Algorithms

- » **Specific Case:** In sorting algorithms, the Bubble Sort is a specific solution that works by swapping adjacent elements and this swapping is performed repeatedly, until we get a sorted list of items.
- » **Generalized Solution:** This sorting concept, however, could be generalized, as we have various algorithms for this purpose e.g. Quick Sort, Merge Sort, and Heap Sort. Each of the above algorithms use different strategies to sort the list of items, however, end result of each algorithm is same.

Example: Calculating Area of shapes

- » **Specific Case:** For calculating the rectangle area, we use the formula $\text{length} \times \text{width}$.
- » **Generalized Solution:** There are various shapes, whose area need to be calculated. Therefore, a generalized solution should be created to calculate the area of different shapes (rectangle, circle, triangle). Because, each shape has its own formula, therefore, a function can be created that can handle different shapes along with their parameters.

2.10 Modular Design

Modular design is a fundamental concept in software engineering and computational thinking, focused on breaking down a system into smaller, manageable, and interchangeable parts, known as modules.

Example: In programming, functions or methods are modules that perform specific tasks. In Python, a function to calculate the area of a rectangle:

Example: OOP uses classes and objects to implement modular design.

Example: In web development, modular design can be applied through components and services.

Steps to Implement Modular Design

1. **Define Modules:** While solving complex problems, we divide the problem into sub problems. For each of these sub problems, there will always be core problem that needs to be handled.

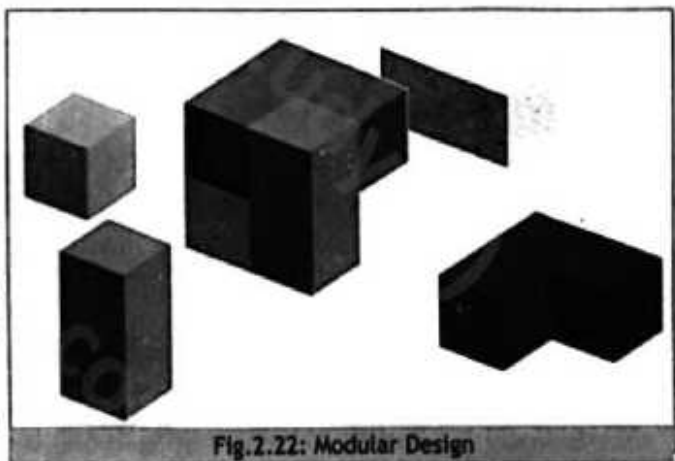


Fig.2.22: Modular Design

For solution of each such core problem, we may define modules.

2. **Design Interfaces:** Each of the above modules needs to interact with each other. To handle this, connectivity of modules, interfaces for modules could make the job easy. This may include input and output specifications for different interfaces.
3. **Implement Modules:** The modules (that are in fact solution of sub problems) needs to be implemented according to their defined algorithmic steps and interfaces.
4. **Integrate Modules:** To come-up with a complete solution of the whole complex problem, combine the modules of the complete system. However, it needs to be ensured that they interact with each other correctly through their interfaces.
5. **Test Modules:** In testing, both type of testing is necessary, the separate module testing and the complete system testing.

2.11 Algorithm Dry Run

The algorithm dry run means manual checking the execution of an algorithm by providing it specific set of inputs. This helps not only helps to verify algorithm's correctness but also to understand algorithm's behavior on certain inputs and conditions. By doing this, we are also able to identify logical errors, if any.

Steps of Dry Run

1. **Choose an Input:** To test an algorithm, the first step is the selection of specific set of input values.
2. **Simulate Execution:** to test algorithm working, each step of the algorithm is manually performed by using the chosen inputs. However, it also needs to keep track of variables states, that may change on performing specific operation.
3. **Observe Output:** After performing all the required steps of an algorithm, its final output is observed.
4. **Compare with Expected Result:** The final output of the algorithm is crosschecked with the expected result. If it matches then it means algorithm is correct, otherwise we review each step to identify if some thing have gone wrong.

Example: Performing dry run of algorithm that finds largest number from a a list of numbers.

Algorithm:

- i. Take a variable max and initialize with the first element of the list.
- ii. Iterate through all the remaining elements of the list.
- iii. Compare each element with the variable max.
- iv. If the compared element is greater than the variable max, update max variable with the compared element.
- v. Now max variable contains largest number return it.

Example Input List: [3, 1, 4, 1, 5, 9, 2, 6]

Dry Run:

1. Initialization:

››› max = 3 (First element in the list)

Iteration (take element other than first in each iteration):

››› Compare 1 with 3: 1 is less than 3, so max will not change.

››› Compare 4 with 3: 4 is greater than 3, update value of max to 4.

››› Compare 1 with 4: 1 is less than 4, so max will not change.

››› Compare 5 with 4: 5 is greater than 4, update value of max to 5.

››› Compare 9 with 5: 9 is greater than 5, update value of max to 9.

››› Compare 2 with 9: 2 is less than 9, so max will not change.

››› Compare 6 with 9: 6 is less than 9, so max will not change.

Result:

››› max contains 9, return this value.

If you visually see your input list, 9 is the largest number and our dry run of the algorithm have also correctly identified 9 as the maximum number. This shows that the algorithm steps are working as intended. The dry runs are the powerful tool that can be used to validate the algorithm logic before writing the code.

Trace Table

A trace table are also used to verify the algorithm correctness. While tracing the algorithm, the trace table records the values of variables and their respective change at each of the algorithm step. The final values of the variables are considered final output of the algorithm.

Step	Current Element	Max (Initial)	Updated Max	Comment
0	3	3	3	Initialize max with first element
1	1	3	3	$1 \leq 3$, so max remains 3
2	4	3	4	$4 > 3$, so update max to 4
3	1	4	4	$1 \leq 4$, so max remains 4
4	5	4	5	$5 > 4$, so update max to 5
5	9	5	9	$9 > 5$, so update max to 9
6	2	9	9	$2 \leq 9$, so max remains 9
7	6	9	9	$6 \leq 9$, so max remains 9
End	-	-	9	Final result

Table 2.2: Trace Table

In the above table at

- Step 0: Initialize max to the first element (3).
- Step 1: Compare 1 with 3. Since 1 is not greater than 3, max remains 3.
- Step 2: Compare 4 with 3. Since 4 is greater than 3, update max to 4.
- Step 3: Compare 1 with 4. Since 1 is not greater than 4, max remains 4.
- Step 4: Compare 5 with 4. Since 5 is greater than 4, update max to 5.
- Step 5: Compare 9 with 5. Since 9 is greater than 5, update max to 9.
- Step 6: Compare 2 with 9. Since 2 is not greater than 9, max remains 9.
- Step 7: Compare 6 with 9. Since 6 is not greater than 9, max remains 9.
- End: The final maximum value is 9.

This trace table shows how the algorithm processes each element of the list and how the max value is updated accordingly.

2.12 Errors

There are two of errors and understanding these errors helps in developing accurate and effective problem-solving strategies.

- **Syntax Errors:** Mistakes related to the structure of an expression or statement, similar to grammatical errors in writing.
- **Logical Errors:** Mistakes in reasoning or planning that lead to incorrect conclusions or results, akin to flawed problem-solving approaches.

Consider the example of Search Engine Query, where search engine returns irrelevant results due to a both logical and syntax error in the query.

Query : "Best smartphones for college students under 500"

Here, the user meant to exclude used phones but gets results including them, it has the logical error

so refine query will be

"Best new smartphones for college students under 500."

Still the user is getting irrelevant result due to syntax error of not writing the currency e.g PKR

So, the refine query will be

"Best new smartphones for college students under PKR 500."

Unit Summary

- **Computational Thinking** is a way of solving problems by using ideas from computer science.
- **Computing problem** in computer science is one that is solved step-by-step through computation.
- In **decision problems** normally 'yes-or-no' response is found in a choice
- In **search problems**, the solution is made up by having a set of values that satisfy a certain criteria.
- **Addition principle of counting** is a basic idea for counting the total number of possibilities that can occur when dealing with many events or options that are mutually exclusive.
- The **Multiplication Principle** is used when we are dealing with many events, and we have to select one option from each event.
- A **permutation** is an arrangement of all the members of a set into a specific sequence or order.
- A **combination** is a selection of items from a set where the order of selection does not matter.
- The **Pigeonhole Principle** states that if more items are put into fewer containers than there are items, at least one container must contain more than one item.
- The **Principle of Inclusion and Exclusion** calculates the size of the union of overlapping sets by adding the sizes of the sets and then subtracting the sizes of their intersections.
- An **algorithm** is a well-defined, step-by-step procedure or set of rules for solving a specific problem or performing a task.
- **Sorting** is the process of arranging items in a specific order.
- **Searching** is the process of finding a particular item within a set.
- **Logical reasoning** is the basis for developing and evaluating algorithms, it is an essential part of computational thinking
- In **Boolean**, two statements or expressions are evaluated in a way that can only have one of two potential values as a result –either true or false.
- **Bubble Sort**: Repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.
- **Selection Sort**: Divides the list into a sorted and an unsorted part, and repeatedly selects the smallest (or largest) element from the unsorted part to move to the sorted part.
- **Merge Sort**: Divides the list into halves, recursively sorts each half, and then merges the sorted halves.

Exercise

Q1. Select the best answer for the following MCQs.

1. Identify the activity that is based on personal preferences rather than algorithmic logic.
 - a) Sorting numbers in a list.
 - b) Deciding what to eat for dinner.
 - c) Determining if a number is a prime number.
 - d) Calculating the square root of a number.
2. Counting Principle Problems could easily be understood by the use of
 - a) Graph representation
 - b) Tree representation
 - c) Clock representation
 - d) Map representation
3. In how many ways can you choose 2 out of 5 different books to take on a trip? (Order does not matter)
 - a) 10
 - b) 20
 - c) 15
 - d) 5
4. A committee of 3 members is to be selected from a group of 8 people. How many different committees can be formed?
 - a) 56
 - b) 84
 - c) 28
 - d) 120
5. You have 5 different shirts and 4 different pants. How many different outfits can you make by choosing one shirt and one pair of pants?
 - a) 9
 - b) 20
 - c) 15
 - d) 10
6. In how many ways can you select 3 different fruits from a basket of 7 different fruits? (Order does not matter)
 - a) 35
 - b) 21
 - c) 56
 - d) 84
7. The Pigeonhole Principle states that if more items are placed into containers than the number of containers, then:
 - a) Some containers must be empty
 - b) At least one container must hold more than one item
 - c) All containers will be full
 - d) Each item will be placed in a different container

8. In a set of 50 students, 30 have completed a math project, 25 have completed a science project, and 15 have completed both. How many students have completed at least one project?
- a) 35 b) 40 c) 45 d) 50
9. Which property of an algorithm ensures that it produces at least one output after the calculations?
- a) Input b) Output c) Definiteness d) Finiteness
10. Which property ensures that an algorithm will eventually end after a finite number of steps?
- a) Input b) Finiteness c) Effectiveness d) Generality

Q2. Write answers of the following short response questions.

1. Consider 3 boys and 3 girls want to team up as pair for the 14th August related student performance at the school. Find the total no. of ways these pairs could be formed using Counting Principle Problems.
2. Suppose you have selected black pant to attend school function and have three options of shirt to choose. Find the total no. of ways you can dress up using Counting Principle Problems.
3. You are sitting at a study table doing homework. The penholder on the table contains 2 blue pens, 1 black pen, and 2 lead pencils. How many options do you have for writing?
4. For a race on sports day activities, a group of 20 students are given a race lane at random. There are six students selected for the first race. How many different orders of the six lanes for students may be chosen?

Q3. Write answers of the following extended response questions.

1. Ejaz, Javeria, Liaqat, and Zainab are standing in a line. Here are the clues to determine their positions:
 - a) Javeria is standing directly behind Ejaz.
 - b) Zainab is not next to Liaqat.
 - c) Liaqat is not at the end of the line.
 - d) Zainab is standing in the first position.

In which order are they standing?

2. Provide students with a detailed map of their school. Ask them to create an abstract version, highlighting only the main buildings and pathways.

3. Give students a problem that can be solved with a simple program. Encourage them to write functions to handle different parts of the problem, abstracting complex operations into manageable pieces.

4. Given the following web form inputs, identify any syntax errors and correct them

Form Inputs:

➤ Name: Amjad Ali ➤ Email: amjadali@gmail ➤ Date: 01/01/2024

5. You have developed a web application that takes the budget head and provides total. Analyze the following budget and identify any logical errors in the total amount.

Budget Example:

- Rent: \$1000
- Utilities: \$150
- Groceries: \$200
- Total: $\$1000 + \$150 + \$200 = \$1500$



Activity-1

Provide students with a detailed map of their school. Ask them to create an abstract version, highlighting only the main buildings and pathways.



Activity-2

Give students a problem that can be solved with a simple program. Encourage them to write functions to handle different parts of the problem, abstracting complex operations into manageable pieces.