Computational Thinking and Algorithm Question Bank

Computer Science Class XI (1st Year), Sindh Textbook Board, Unit 2. Short questions, differences, long questions and practical questions with answers, built from the new STBB textbook and its end-of-unit exercise.

STBB Computer Science XI | Unit 2, pages 71 to 90 | Algorithms, sorting, searching | Moderate level

Read this first. This unit is about solving problems step by step. Learn the definitions of algorithm, pseudocode, decomposition, pattern recognition and abstraction, then practise the traces of Bubble Sort, Selection Sort, Linear Search and Binary Search on the book list 8, 4, 1, 9, 3. Examiners like step by step traces, so write each pass clearly. Tags: **BOOK EXERCISE** = taken from the end-of-unit exercise, **LIKELY** = from the book topics.

Short questions

3 to 4 lines each

S1 What is computational thinking?

BOOK TOPIC 2.1 | LIKELY

Computational thinking is a systematic approach to solving problems in the way a computer scientist does. It involves breaking a big problem down into smaller parts, identifying patterns, abstracting away the unnecessary details and developing a step-by-step algorithm. It is used in programming, but it is also useful for solving everyday problems in a logical way.

S2 What is an algorithm?

BOOK TOPIC 2.1 | LIKELY

An algorithm is a step-by-step logical procedure used to solve a problem. It is conceptual in nature and independent of any programming language, which means it can later be written in Python, C++ or any other language. A good algorithm has clear steps, takes some input and gives the correct output in a finite number of steps, such as the steps to make tea or to sort a list.

S3 Why are algorithms important? Give any three reasons.

BOOK TOPIC 2.1.1 | LIKELY

- They give a structured approach that makes complex tasks easier to understand and execute.
- They help in selecting solutions that save time and memory, which is very important for large-scale applications.
- Once developed, an algorithm can be reused for similar problems, and it gives accurate and consistent results by reducing human error.

S4 What is pseudocode?

BOOK TOPIC 2.1 | LIKELY

Pseudocode is a human-readable representation of an algorithm using plain English and structured programming keywords such as IF, FOR and WHILE. It is semi-formal and not executable code, so it cannot be run on a computer. It is used to present how the algorithm will be implemented, in a readable format, before the actual coding is started.

S5 Why is an algorithm called language independent?

LIKELY CONCEPT

An algorithm only describes the logical steps needed to solve a problem and does not use the syntax of any programming language. Because of this the same algorithm can be written in Python, C++, Java or any other language without changing its logic. It is just like a recipe, which can be followed in any kitchen no matter which language it is written in.

S6 What is decomposition? Give a real-life example.

BOOK EXERCISE 3(II) LIKELY

Decomposition is a problem-solving technique in which a complex task or system is broken down into smaller, manageable parts. It makes a big problem easier to understand, develop and solve. Example: when a person learns a new language, he does not learn everything at once but breaks it into subtasks such as learning vocabulary, grammar rules, tenses and sentence building.

S7 Write any three benefits of decomposition.

BOOK TOPIC 2.2.1 LIKELY

- It reduces complexity because we can focus on one small piece at a time.
- It makes learning or work manageable, since each small part is easy to practise in steps.
- It helps in debugging because we can find exactly where the mistake is, and it promotes reusability because a solved part can be applied again to new problems.

S8 What is pattern recognition?

BOOK EXERCISE 3(I) LIKELY

Pattern recognition is the process of identifying regularities, repetitions or rules in data or in a problem. In programming and algorithm design it helps us to discover meaningful relationships among elements, to reduce the overall complexity of the algorithm and to develop generic, reusable solutions. This idea forms the foundation of loops, conditions and data structures.

S9 What three questions do we ask to find a pattern?

BOOK TOPIC 2.2.2 LIKELY

To find a pattern we observe the input data and ask three questions: What is repeating? What changes in a predictable way? Can I generalize this? Answering these questions helps us to find the repeated rule in the data, so that we can write it only once, for example as a loop, instead of writing every step again and again.

S10 In the star triangle (*, , *, **, ***) what pattern do you see?

BOOK TOPIC 2.2.2 LIKELY

The first row shows 1 star, the second row shows 2 stars, the third row shows 3 stars and so on, so each row adds one star to the previous row and row number n has n stars. Because the rule repeats, we do not need to write each row separately. We can write a loop that prints n stars for each row, and it will work for any number of rows.

S11 **What is abstraction?**

BOOK TOPIC 2.2.3 LIKELY

Abstraction means hiding the unnecessary features and focusing only on the essential ones that are needed to solve a problem. In simple words, it reduces the complexity of an algorithm by ignoring what is not important and concentrating on what truly matters. For example, to make tea we need to know the steps, but not the brand or the colour of the kettle.

S12 **Give an example of abstraction using making tea.**

BOOK TOPIC 2.2.3 LIKELY

In making tea the necessary steps are: boil water, add tea leaves or a tea bag, steep for a few minutes, pour into a cup and serve. These five steps are the abstracted process and are true for every type of tea. Details such as the brand of the kettle, the colour or design of the cup, and the place where the tea is made are not important, so they are ignored.

S13 **Which details are inputs or parameters in the making tea example?**

BOOK TOPIC 2.2.3 LIKELY

The details that can change according to the situation are the inputs or parameters of the algorithm. In the tea example these are the ingredients (black tea, green tea, herbal tea, milk tea), the additions (sugar, honey, milk, lemon), the tools (electric kettle, stove, tea pot) and the time (boiling time and steeping time). The main logic of the algorithm remains the same for all of them.

S14 **Define sorting. Name the two common orders.**

BOOK TOPIC 2.3 LIKELY

Sorting is the process of arranging a collection of items or data in a specific order. It is used to organize data so that searching, retrieval and analysis become easier. The two common orders are ascending order (lowest to highest, for example 0 to 9 or A to Z) and descending order (highest to lowest, for example 9 to 0 or Z to A).

S15 **Define searching.**

BOOK TOPIC 2.3 LIKELY

Searching is the process of locating a specific element within a collection of data. Searching algorithms are designed to find the position, the existence or the value of a target element in a list in an efficient and systematic way, for example finding a student's name in the class list. Common searching algorithms are Linear Search and Binary Search.

S16 **Name any four real-life uses of sorting and searching.**

BOOK TOPIC 2.3 LIKELY

- Database queries: records are sorted and searched to get the required data quickly.
- Online shopping filters: products are sorted by price or rating and searched by name.
- Leaderboards in games: scores are sorted from highest to lowest.
- File organization and recommendation systems: files are arranged by name or date, and suggestions are found by searching the data.

S17 What is Bubble Sort? Why is it also called Ripple Sort?

BOOK EXERCISE 1(II) LIKELY

Bubble Sort is an elementary sorting algorithm used to arrange the elements of a list in ascending or descending order. It works by repeatedly comparing two adjacent elements and swapping them if they are in the wrong order, and the process is repeated until the whole list is sorted. It is also called Ripple Sort because the largest value moves ("ripples") towards the end of the list in every pass.

S18 What is Selection Sort?

BOOK TOPIC 2.3.1 LIKELY

Selection Sort is a simple sorting algorithm used to arrange the elements of a list in ascending or descending order. It works by repeatedly selecting the smallest element from the unsorted portion of the list and moving it to its correct position at the front. In this way the sorted part grows by one element after every pass, and only one swap is needed in each pass.

S19 Write any two advantages of Bubble Sort.

BOOK TOPIC 2.3.1 LIKELY

- It is the easiest sorting algorithm to visualize and explain; if you understand swapping two cards in your hand, you understand Bubble Sort.
- It does not need extra memory because it just swaps the elements inside the original array, and it keeps the original order of items with the same value, which is important in some database work.

S20 Write any two disadvantages of Bubble Sort.

BOOK TOPIC 2.3.1 LIKELY

- It uses nested loops, so the time it takes grows very fast with the size of the list; if the number of items is doubled, it takes about four times longer to sort.
- Every swap needs three operations, so it performs a massive number of operations compared with other algorithms, which makes it useless for real-world big data.

S21 What is the main disadvantage of Selection Sort?

BOOK EXERCISE 1(IV) LIKELY

The main disadvantage of Selection Sort is that it must finish every loop even if the list is already sorted, because it always scans the entire remaining list. So it takes the same amount of time whether the data is sorted, reversed or random. It also uses nested loops, so it becomes very slow for large data and is used only for small datasets.

S22 Write two advantages of Selection Sort.

BOOK TOPIC 2.3.1 LIKELY

- In every pass it performs at most one swap, so it is efficient on a system where writing or moving data is very expensive.
- It does not need extra memory, and like Bubble Sort it is easy to code and understand because it is basically just two loops, one to track the sorted boundary and one to find the minimum value.

S23 What is Linear Search?

BOOK TOPIC 2.3.2 LIKELY

Linear search, also known as sequential search, is a simple searching algorithm used to find a specific element within a list. It works by examining each element of the list one by one, from the start, until the target element is found or until all the elements have been checked. It is simple to implement and works on both sorted and unsorted lists.

S24 What is Binary Search?

BOOK EXERCISE 4(IV) LIKELY

Binary search is a highly efficient searching algorithm that works by repeatedly dividing the search interval in half and comparing the middle element with the target. If the target equals the middle element the search is successful; if the target is smaller the search continues in the left half, and if it is larger it continues in the right half. This repeats until the element is found or does not exist.

S25 What is the only limitation of Binary Search?

BOOK TOPIC 2.3.2 LIKELY

The only limitation of Binary Search is that it works only on a sorted list. If the data is scrambled, the logic of comparing with the middle element fails completely and the result will be wrong. So an unsorted list must be sorted first, and sorting takes extra time; if we search only once, a Linear Search may be faster.

S26 Write two advantages and two disadvantages of Linear Search.

BOOK TOPIC 2.3.2 LIKELY

- Advantages: it does not care whether the data is sorted, alphabetical or numerical, so it works on any list; the code is very simple, just one loop and one if statement, and it needs no extra space.
- Disadvantages: for a list of 1 million items it may need 1 million checks if the target is at the end, and its time grows linearly with the size of the list, so doubling the list doubles the search time.

S27 Write two advantages and two disadvantages of Binary Search.

BOOK TOPIC 2.3.2 LIKELY

- Advantages: it cuts the search area in half at every step, so it can find an item in a list of 1 million elements in just about 20 steps; it is the go-to method for big data and stays fast even for 1 billion items.
- Disadvantages: it works only if the data is already in order, and it needs a way to jump to the middle index (mid = (low + high) / 2), so it works best with arrays; logic errors such as an infinite loop are also easier to make.

S28 What do the abbreviations BEG, END, MID and PTR stand for in the algorithms?

BOOK EXERCISE 1(VIII, X) LIKELY

BEG stands for the beginning position of the list or the part being searched, END is the last position and MID is the middle position, which is calculated as INT((BEG + END) / 2). PTR stands for pointer, a variable that keeps track of the current position in an array while the algorithm moves through the data. INT is used to change a decimal result such as 1.5 into an integer.

S29 **Explain what is meant by correctness of an algorithm.**

BOOK EXERCISE 3(III) LIKELY

An algorithm is considered correct if its output matches the expected outcome for the given input. It must solve the problem exactly as defined in the requirements, handle typical inputs gracefully (such as negative numbers, empty values or maximum-capacity data) and be guaranteed to finish its execution without falling into an infinite loop. For example, the even/odd check using MOD 2 is correct because it always gives the right result.

S30 **Why is efficiency important in problem-solving?**

BOOK EXERCISE 3(IV) LIKELY

Efficiency means how fast an algorithm runs and how much memory it uses as the size of the input grows. It is important because an efficient algorithm saves time and memory, scales well with larger inputs and gives better performance in real-world use. For example, a linear search is less efficient than a binary search on a large sorted list because it needs many more steps.

S31 **What do you understand by clarity of an algorithm?**

BOOK EXERCISE 3(V) LIKELY

Clarity, or readability, means that an algorithm is easy to read, understand and modify by other users. For this the steps should be clearly written, the variable names should be meaningful, the logic should have proper indentation and comments, and the flow of control (IF, LOOP and so on) should be easy to follow. A clear algorithm is easier to check for errors and to update later.

S32 **What is a trace table?**

BOOK TOPIC 2.4.1 LIKELY

A trace table is a table that follows an algorithm step by step with sample input values and shows the operation done and the result at each step. It is used to test an algorithm manually and to check that it is correct. For example, the trace table of the even/odd algorithm shows the input number, the division by 2, the remainder, and the final result Even or Odd.

S33 **Write the questions to ask before deciding which algorithm to use.**

BOOK TOPIC 2.5 LIKELY

It can be difficult to decide which algorithm is best for a problem, so before deciding we should ask these questions: What is the type of problem? What is the size of the input? Is the data sorted or not? Do we need speed and memory efficiency? The answers to these questions help us to select the most suitable algorithm, for example Linear Search or Binary Search.

S34 **Which algorithm do you choose in these cases: searching unsorted data, searching sorted data and ordering data?**

BOOK TOPIC 2.5.1 LIKELY

If we are searching in unsorted data we use the Linear Search algorithm, because it works on any list. If we are searching in sorted data we use the Binary Search algorithm because it is much faster. If the data needs ordering we use a Sorting algorithm such as Bubble Sort or Selection Sort. If a problem repeats many times, the Dynamic Programming algorithm is used.

S35 **What does the abbreviation CT stand for, and what does MOD mean?**

BOOK EXERCISE 1(VI, VII) LIKELY

CT stands for Computational Thinking, the problem-solving approach based on decomposition, pattern recognition, abstraction and algorithms. MOD is an operation that finds the remainder after division. For example, 7 MOD 2 = 1 because 7 divided by 2 leaves a remainder of 1. It is used in the even/odd algorithm: if number MOD 2 = 0 then the number is even, otherwise it is odd.

Differences

3 to 5 marks each

D1 **What is the difference between an algorithm and pseudocode?**

BOOK EXERCISE 4(I) LIKELY

Algorithm	Pseudocode
A step-by-step logical procedure to solve a problem	A human-readable representation of an algorithm using plain English and programming keywords
Conceptual and language-independent	Semi-formal and structured, but not executable code
Usually described in plain steps, bullets or numbered points	Written with structured constructs (IF, FOR, WHILE) without programming syntax
Focuses on the logic and steps of the problem	Focuses on how to represent those steps clearly before coding
Anyone can understand it, including non-programmers	Mainly for programmers or students preparing for coding
No fixed format; can be informal	Follows standard pseudocode conventions

D2 **Differentiate between decomposition and abstraction.**

LIKELY CONCEPT

Point	Decomposition	Abstraction
Meaning	Breaking a big problem into smaller parts	Hiding unnecessary details and keeping the essential ones
Goal	Make the problem manageable	Reduce the complexity of the algorithm
Example	Learning a language by parts (vocabulary, grammar, tenses)	Making tea ignoring the brand and colour of the kettle

D3 Differentiate between pattern recognition and decomposition.

LIKELY CONCEPT

Pattern recognition	Decomposition
Finds repeated rules or regularities in data	Splits a task into smaller subtasks
Helps to make generic, reusable solutions	Helps to focus on one piece at a time
Example: each star row adds one star	Example: learning a language in parts

D4 Differentiate between Bubble Sort and Selection Sort.

LIKELY

Point	Bubble Sort	Selection Sort
Working	Compares adjacent elements and swaps if in wrong order	Selects the smallest element and places it at its correct position
Swaps per pass	Many swaps	At most one swap
Other name	Ripple Sort	None
Memory	No extra memory	No extra memory
Disadvantage	Many swaps, slow for big data	Always same time, may change the order of equal items

D5 Differentiate between Linear Search and Binary Search.

BOOK EXERCISE 4(V) LIKELY

Point	Linear Search	Binary Search
Other name	Sequential search	None
Method	Checks each element one by one	Repeatedly divides the list in half
Data must be sorted	No	Yes
Speed	Slow for large lists	Very fast for large lists
Steps for 1 million items	Up to 1 million	About 20
Code	Very simple	Little more complex
Best used for	Unsorted and small data	Sorted and large data

D6 Differentiate between sorting and searching.

LIKELY CONCEPT

Sorting	Searching
Arranges data in a specific order	Finds a specific element in data
Output is the ordered list	Output is the position or found or not found
Examples: Bubble Sort, Selection Sort	Examples: Linear Search, Binary Search

D7 Differentiate between ascending and descending order.

LIKELY CONCEPT

Ascending	Descending
Lowest to highest	Highest to lowest
0 to 9, A to Z, a to z	9 to 0, Z to A, z to a
Example: 1, 3, 4, 8, 9	Example: 9, 8, 4, 3, 1

D8 Differentiate between time efficiency and space efficiency.

BOOK TOPIC 2.4.2 LIKELY

Time efficiency	Space efficiency
How fast the algorithm runs as the input grows	How much memory the algorithm uses
Also called time complexity	Also called space complexity
Example: Binary search is faster than linear search	Example: Bubble sort needs no extra memory

Long questions

5 to 8 marks each

L1 What is abstraction in an algorithm, and why is it used? Explain with the example of making tea.

BOOK EXERCISE 4(II) LIKELY

Abstraction means hiding unnecessary features and focusing only on the essential ones that are needed to solve a problem. In simple terms, it reduces the complexity of an algorithm by ignoring what is not important and concentrating on what truly matters. It is used so that the algorithm stays simple, general and reusable for many similar situations.

WHY ABSTRACTION IS USED

- It removes noise, so the logic of the solution is easy to see.
- It makes the algorithm work for many cases, not only one.
- It saves time because we do not design useless details.

APPLYING ABSTRACTION TO "MAKING TEA": THREE QUESTIONS

We ask three questions: What is necessary? What can change based on situation? What can we ignore?

1. WHAT IS NECESSARY?

These are the minimum steps required, regardless of the type of tea: boil water, add tea leaves or a tea bag, steep for a few minutes, pour into a cup, and serve. This is the abstracted process and the key logic of "making tea" in all cases.

2. WHAT CAN CHANGE BASED ON THE SITUATION?

Details can vary: the ingredients (black tea, green tea, herbal tea, milk tea), the additions (sugar, honey, milk, lemon), the tools (electric kettle, stove, tea pot) and the time (boiling time and steeping time). These are the inputs or parameters of the algorithm.

3. WHAT CAN WE IGNORE?

The brand of the kettle, the colour or design of the cup, where the tea is being made (kitchen or office) and the type of mug (glass or ceramic) are not important for the main logic. These details add noise and are excluded from the abstraction.

In this way abstraction gives one simple algorithm that works for every cup of tea.

L2 Explain decomposition and pattern recognition with examples. Why are they useful in computational thinking?

LIKELY CONCEPT

Computational thinking is a systematic way of solving problems. Decomposition and pattern recognition are two of its main techniques, along with abstraction and algorithms. Both of them make a problem simpler before the solution is written.

DECOMPOSITION

Decomposition is a problem-solving technique in which a complex task or system is broken down into smaller, manageable parts. It is used in computational thinking and software development, and it is equally useful in daily life.

EXAMPLE OF DECOMPOSITION: LEARNING A NEW LANGUAGE

A person does not learn a whole language at once. The task is broken into subtasks: learn vocabulary, understand grammar rules, learn tenses, practise sentence building, listen and imitate native speakers, and get feedback from a teacher.

WHY DECOMPOSITION IS USEFUL

- It reduces complexity because we focus on one piece at a time.
- It makes learning manageable and easy to practise in steps.
- It helps in debugging because we can find exactly where the mistake is.
- It promotes reusability: once you learn how to form a sentence, you can apply it to new words.

PATTERN RECOGNITION

Pattern recognition is the process of identifying regularities, repetitions or rules in data. To find a pattern we observe the input data and ask: what is repeating, what changes in a predictable way, and can I generalize this?

EXAMPLE OF PATTERN RECOGNITION: STAR TRIANGLE

In the triangle of stars (*, **, ***, ****) the first row shows 1 star, the second 2 stars and so on. The pattern is that each row adds one star to the previous row.

WHY PATTERN RECOGNITION IS USEFUL

- It helps to discover meaningful relationships among elements.
- It reduces the overall complexity of the algorithm.
- It helps to develop generic and reusable solutions, and it forms the foundation of loops, conditions and data structures.

CONCLUSION

Decomposition makes a big problem small, and pattern recognition finds the repeated rule inside it, so together they help us to write clear and short algorithms.

L3 Write the steps (algorithm) of the Bubble Sort and trace it on the list 8, 4, 1, 9, 3.

BOOK TOPIC 2.3.1 LIKELY

Bubble Sort is an elementary sorting algorithm used to arrange the elements of a list in ascending or descending order. It works by repeatedly comparing adjacent elements and swapping them if they are in the wrong order. This process is repeated until the whole list is sorted. It is also called Ripple Sort because the largest value moves to the end in every pass.

ALGORITHM (IN WORDS)

- Step 1: Start and set the array DATA and N (the number of items).
- Step 2: Repeat for K = 1 to N-1 (this is the outer loop, one pass for each K).
- Step 3: In each pass set the pointer PTR = 1, and while PTR is less than or equal to N-K: if DATA[PTR] is greater than DATA[PTR+1], swap them using a temporary variable, then increase PTR by 1.
- Step 4: End. The list is now sorted.

TRACE OF THE LIST 8, 4, 1, 9, 3

Pass	Comparisons and swaps	List after pass
1	8>4 swap; 8>1 swap; 8<9 no change; 9>3 swap	4, 1, 8, 3, 9
2	4>1 swap; 4<8 no change; 8>3 swap	1, 4, 3, 8, 9
3	1<4 no change; 4>3 swap	1, 3, 4, 8, 9
4	1<3 no change	1, 3, 4, 8, 9

Sorted list: 1, 3, 4, 8, 9. After every pass the largest remaining value reaches its final place at the right end.

PSEUDOCODE

```
FOR outer = 0 TO N-1
  FOR inner = 0 TO N-2-outer
    IF DATA[inner] > DATA[inner+1] THEN swap them
  END FOR
END FOR
```

ADVANTAGES AND DISADVANTAGES

- Advantages: easy to understand and code; needs no extra memory.
- Disadvantages: nested loops make it slow for big lists, and it makes many swaps.

L4 Write the steps of the Selection Sort algorithm and trace it on the list 8, 4, 1, 9, 3.

BOOK EXERCISE 4(III) LIKELY

Selection Sort is a simple sorting algorithm that arranges a list by repeatedly selecting the smallest element from the unsorted portion and moving it to its correct position at the front. After every pass the sorted part of the list grows by one element.

ALGORITHM (IN WORDS)

- Step 1: Start and set the array DATA and N.
- Step 2: Repeat for K = 1 to N-1: this position K is the place for the next smallest value.
- Step 3: Set PTR = K+1 and compare DATA[K] with every DATA[PTR] up to N. If DATA[K] is greater than DATA[PTR], swap them, so the smallest value comes to position K. Move PTR forward.
- Step 4: End. The list is sorted.

TRACE OF THE LIST 8, 4, 1, 9, 3

Pass	What happens	List after pass
1	Smallest value 1 comes to position 1 (8>4 swap, 4>1 swap, then no change)	1, 8, 4, 9, 3
2	Smallest of the rest is 3, placed at position 2 (8>4 swap, 4>3 swap)	1, 3, 8, 9, 4
3	Smallest of the rest is 4, placed at position 3 (8>9 no, 8>4 swap)	1, 3, 4, 9, 8
4	9>8 swap	1, 3, 4, 8, 9

Sorted list: 1, 3, 4, 8, 9.

ADVANTAGES

- It is easy to code, as it is just two loops: one to track the sorted boundary and one to find the minimum value.
- In each pass it makes at most one final placement, so it is good where moving data is expensive.
- It needs no extra memory.

DISADVANTAGES

- It must finish every loop even if the list is already sorted, so it takes the same time for any data.
- It becomes very slow for large data, so it is used only for small datasets.
- Equal items may change their relative order.

L5 Write the algorithm and pseudocode of Linear Search. Trace it for the list 8, 3, 6, 1, 7, 2, 4, 5 searching 7.

BOOK TOPIC 2.3.2 LIKELY

Linear search, also known as sequential search, is a simple searching algorithm used to find a specific element in a list. It works by examining each element one by one, from the beginning, until the target is found or all the elements have been checked. It works on both sorted and unsorted lists.

ALGORITHM (IN WORDS)

- Step 1: Start. Set DATA (the list), N (the number of items) and FindValue (the item to search).
- Step 2: Set ValueFound = 0, where 0 means not found and 1 means found.
- Step 3: Repeat for PTR = 1 to N: if DATA[PTR] is equal to FindValue, set ValueFound = 1 and stop the loop.
- Step 4: If ValueFound = 1 print the data, otherwise print "Data not found". End.

PSEUDOCODE

```
ValueFound = 0
FOR PTR = 0 TO N-1
  IF DATA[PTR] == FindValue THEN ValueFound = 1, Exit loop
END FOR
IF ValueFound == 1 THEN print DATA[PTR] ELSE print "Data Not Found"
```

TRACE FOR ITEM 7

Step	Element checked	Equal to 7?
1	8	Not found
2	3	Not found
3	6	Not found
4	1	Not found
5	7	Found

The item is found at the 5th position.

ADVANTAGES AND DISADVANTAGES

- Advantages: it works on any list (sorted or scrambled), the code is very simple, and it needs no extra space.
- Disadvantages: if the list has 1 million items and the target is at the end, it makes 1 million checks, and its time grows linearly with the list size.

L6 Write the steps of the Binary Search algorithm. Search the number 3 in the list 1, 2, 3, 4, 5, 6, 7, 8.

BOOK EXERCISE 4(IV) LIKELY

Binary search is a highly efficient searching algorithm that works by repeatedly dividing the search interval in half. It compares the middle element with the target: if they are equal the search is successful, if the target is smaller it continues in the left half, and if it is larger it continues in the right half. The list must be sorted.

STEPS

- Step 1: Start. Take a sorted array. Set BEG = 1 (beginning) and END = last position (here 8).
- Step 2: Find the middle position: MID = INT((BEG + END) / 2). INT changes a decimal result into an integer.
- Step 3: Repeat while BEG is less than or equal to END and DATA[MID] is not equal to the item: if the item is smaller than DATA[MID] set END = MID - 1, otherwise set BEG = MID + 1. Find a new MID.
- Step 4: If DATA[MID] equals the item, the location is MID. Otherwise the item is not found. End.

TRACE: LIST 1 TO 8, SEARCHING 3

```
BEG = 1, END = 8, MID = INT(9/2) = 4. DATA[4] = 4.
3 < 4, so search the left part: END = 3.
BEG = 1, END = 3, MID = 2. DATA[2] = 2.
3 > 2, so search the right part: BEG = 3.
BEG = 3, END = 3, MID = 3. DATA[3] = 3.
```

Answer: Found at position 3 in only 3 steps. For 1 million items Binary Search needs only about 20 steps.

LIMITATION

It works only on a sorted list. If the data is scrambled, the logic fails.

L7 Which searching algorithm is best for large datasets? Explain your answer.

BOOK EXERCISE 4(V) LIKELY

Binary Search is the best searching algorithm for large datasets, provided that the data is sorted. The answer depends on the speed of the algorithm as the number of items grows.

SPEED OF BINARY SEARCH

It cuts the search area in half at every single step. So an item in a list of 1 million elements can be found in about 20 steps, whereas Linear Search may need 1 million checks. Whether the list has 1,000 items or 1 billion items, Binary Search stays very fast.

SPEED OF LINEAR SEARCH

Linear Search checks the elements one by one, so if the target is at the very end the algorithm has to perform all the checks. If the size of the list is doubled, the search time is also doubled.

CONDITIONS FOR BINARY SEARCH

The data must already be in order. If the list is not sorted, it must be sorted first, which takes extra time. If we have to search only once in unsorted data, Linear Search may actually be faster. But for a static, sorted list (like a dictionary or a phone book) that is searched thousands of times, sorting once and using Binary Search saves a massive amount of time.

Linear Search	Binary Search
Unsorted and small data	Sorted and large data
Time grows linearly with data	Search area halves at each step
Very simple code	Needs mid = (low + high) / 2 logic

Conclusion: Linear Search is used for unsorted and small datasets. Binary Search is used for sorted and large datasets that need fast searching.

L8 Explain the evaluation criteria of an algorithm: correctness, efficiency and clarity.

BOOK TOPIC 2.4 LIKELY

There are many ways to solve a problem, so we need criteria to judge how good an algorithm is. The three most important criteria used to assess the quality of an algorithm are correctness, efficiency and clarity.

1. CORRECTNESS

An algorithm is correct if its output matches the expected outcome for the given input. We check whether it solves the problem exactly as defined in the requirements, whether it handles typical inputs gracefully (negative numbers, null values, empty strings or maximum-capacity data) and whether it is guaranteed to finish its execution and avoid an infinite loop. Example: the even/odd check (IF number MOD 2 = 0 THEN display Even, ELSE display Odd) is correct because it always decides properly.

2. EFFICIENCY

Efficiency is measured by the time and the space (memory) that an algorithm uses as the input grows. Time (speed) means how fast it runs and space means how much memory it needs. We ask: does it scale well with

larger inputs, is there a way to make it faster or use less memory, and is it optimized for real-world use? Example: Linear Search is less efficient than Binary Search for large sorted data.

3. CLARITY (READABILITY AND UNDERSTANDABILITY)

An algorithm must be easy to read, easy to understand and easy to modify by other users. For clarity: the steps are clearly written, variable names are meaningful, the logic is shown with indentation and comments, and the flow of control (IF, LOOP and so on) is easy to follow.

CONCLUSION

A good algorithm should be correct first, then efficient, and it should also be clear so that other people can understand and maintain it.

L9 Check whether numbers are even or odd. Write the algorithm and complete the trace table for 3, 6, 9, 12, 15.

BOOK TOPIC 2.4.1 LIKELY

An even number is exactly divisible by 2 (the remainder is 0), and an odd number leaves a remainder of 1. The MOD operation gives the remainder after division, so we can use it to check a number. For example, 6 MOD 2 = 0 (even) and 7 MOD 2 = 1 (odd).

ALGORITHM

```
1. INPUT number
2. IF number MOD 2 = 0 THEN
3.    DISPLAY "Even"
4. ELSE
5.    DISPLAY "Odd"
6. END IF
```

EXPLANATION OF THE STEPS

Step 1 takes the number from the user. Step 2 divides it by 2 and checks whether the remainder is 0. If yes, step 3 displays "Even". Otherwise step 5 displays "Odd". Step 6 closes the decision.

TRACE TABLE

Input	Divide by 2	Remainder is 0?	Result
3	Quotient 1, remainder 1	No	Odd
6	Quotient 3, remainder 0	Yes	Even
9	Quotient 4, remainder 1	No	Odd
12	Quotient 6, remainder 0	Yes	Even
15	Quotient 7, remainder 1	No	Odd

CONCLUSION

The algorithm is correct because it always determines even or odd properly for every input.

L10 How do we decide which algorithm to use? Give a simple decision guide.

BOOK TOPIC 2.5 LIKELY

It can be difficult to decide which algorithm is best for a problem. Before deciding, we should think about the problem and the data. Choosing the wrong algorithm can waste a lot of time and memory.

QUESTIONS TO ASK

- What is the type of problem (searching, sorting or something else)?
- What is the input size (small or large)?
- Is the data sorted or unsorted?
- Do we need speed and memory efficiency?

SIMPLE DECISION GUIDE

Situation	Algorithm to choose
Searching in unsorted data	Linear Search
Searching in sorted data	Binary Search
Data needs ordering	A sorting algorithm (Bubble or Selection Sort)
Problem repeats many times	Dynamic programming

EXAMPLE

Problem: find a number in a sorted list. Is the data sorted? Yes. Is a fast result needed? Yes. So the correct choice is Binary Search.

L11 Explain sorting and searching algorithms in computer science. Why are they important?

BOOK TOPIC 2.3 LIKELY

Sorting and searching are fundamental techniques of computer science and data analysis. Both work on lists of data of the same type, such as a student's name, father's name or age.

SORTING

Sorting is the process of arranging a collection of items or data in a specific order. It may be ascending (lowest to highest, 0 to 9, a to z, A to Z), descending (highest to lowest) or chronological. It is used to organize data so that it becomes easier to use for searching, retrieval and analysis. Common sorting algorithms are Bubble Sort and Selection Sort.

SEARCHING

Searching is the process of locating a specific element within a list of data. Searching algorithms are designed to find the position, the existence or the value of a target element in an efficient and systematic way, for example finding a student's name in the class list. Common searching algorithms are Linear Search and Binary Search.

IMPORTANCE AND REAL-LIFE USES

- Database queries: finding and arranging records.
- Online shopping filters: sorting by price and searching by name.
- Leaderboards: sorting scores in descending order.
- File organization: arranging files by name or date.
- Recommendation systems: searching and ordering suggestions.

Without sorting and searching, working with large amounts of data would be very slow.

L12 Explain the importance of algorithms in problem-solving. How does an algorithm differ from pseudocode?

BOOK EXERCISE 4(I) LIKELY

An algorithm is a step-by-step logical procedure used to solve a problem. It is conceptual in nature and independent of any programming language. Algorithms are vital because they provide clear and efficient solutions. Without algorithms we would have to reinvent the wheel every time we face a challenge.

IMPORTANCE OF ALGORITHMS

- They provide a structured approach to problem-solving, which makes complex tasks easier to understand and execute.
- They help in selecting solutions that save time and memory, which is crucial for large-scale applications.
- Once developed, an algorithm can be reused for similar problems, saving both time and effort.
- They ensure accuracy and consistency, producing correct results and reducing the chance of human error.
- Every computer program is built upon algorithms, and they serve as a blueprint before the actual code is written.

WHAT IS PSEUDOCODE

Pseudocode is a human-readable representation of an algorithm using plain English and structured programming keywords (IF, FOR, WHILE). It is not executable code but shows how the algorithm can be implemented in a readable format before actual coding.

DIFFERENCE BETWEEN ALGORITHM AND PSEUDOCODE

Algorithm	Pseudocode
A step-by-step logical procedure to solve a problem	A human-readable form of an algorithm using plain English and programming keywords
Conceptual and language-independent	Semi-formal and structured, but not executable code
Usually described in plain steps or numbered points	Written with constructs such as IF, FOR and WHILE
Focuses on the logic and steps	Focuses on how to represent the steps clearly before coding
Anyone can understand it, including non-programmers	Mainly for programmers or students preparing for coding
No fixed format; can be informal	Follows standard pseudocode conventions