CHAPTER 3

Algorithms and Problem Solving

www.ilmkidunya.com

Q. What is a computational problem? Discuss its main components.

Computational Problem

A **computational problem** is a task or challenge that can be solved using a step-by-step procedure. This procedure is known as algorithm. An **algorithm** is a finite set of instructions that can be executed by the computer to solve the problem.

Components of Computational Problem

A computational problem typically consists of three main components:

Input — It refers to the data given to the algorithm. For example, a list of numbers is given to the algorithm to be sorted.

Process — It refers to the steps or rules that are applied to the input to generate the output. For example, an algorithm applies the sorting process to the given list of numbers to sort it.

Output — It is the solution or result produced by the algorithm after processing the input. For example, the list of numbers in ascending order is the output.

Q. What are different classifications of computational problem?

Computational problems can be classified into different categories depending on the nature of the problem and how it needs to be solved.

The main categories are as follows:

1. Decision Problems

Decision problems are the problems where output is **"yes"** or **"no"**. An example of decision problem is as follows:

- **Problem:** Is a given number even?
- **Input:** 8
- **Output:** Yes

2. Search Problems

Search problems are the problems where the goal is to find a solution or item that satisfies certain conditions. An example of search problem is as follows:

- **Problem:** Find the first student in a list who scored more than 90%.
- **Input:** A list of student scores
- **Output:** The name of the first student with a score more than 90%

3. Optimization Problems

Optimization problems are the problems where the goal is to find the best solution from many possible solutions based on certain rules. An example of optimization problem is as follows:

- **Problem:** Find the shortest route to deliver packages to five locations.
- **Input:** List of locations and distances
- **Output:** The route with the shortest total distance

4. Counting Problems

Counting problems are the problems where the goal is to count how many ways certain conditions can be met. An example of counting problem is as follows:

- **Problem:** How many 3-letter passwords can be made using letters A, B and C?
- **Input:** Letters A, B, C
- **Output:** 27 (since $3^3 = 27$ possible combinations)

AAA	AAB	AAC	BAA	BAB	BAC	CAA	CAB	CAC
ABA	ABB	ABC	BBA	BBB	BBC	CBA	CBB	CBC
ACA	ACB	ACC	BCA	BCB	BCC	CCA	CCB	CCC

Figure: 3-letter passwords

Q. Briefly describe the types of problems based on how clearly they are defined. Give examples.

Problems can also be categorized based on how clearly they are defined.

1. Well-defined Problems

A **well-defined problem** is a problem that has clear goals, inputs, processes and outputs. These problems are easy to understand because all aspects of the problem are known and unambiguous. They can be solved using a step-by-step method known as an algorithm. The well-defined problems are more suitable to be solved using a computer because the solution can be clearly programmed and automated.

Example

An example of a well-defined problem is as follows:

- **Problem:** Determine if a number is even.
- **Goal:** Check whether the given number is even.
- **Input:** A number such as 8
- **Process:** Divide the number by 2 and check if the remainder is 0
- **Output:** "Even" if the result is 0, otherwise "Odd"

This is a well-defined problem because all components such as goal, input, process and output are clearly defined.

2. Ill-defined Problems

An **ill-defined problem** is a problem that does not have a clear goal, input, process or expected output. These problems are usually open-ended and vague. It means that these problems have no single correct answer or they can be solved in different ways. They are difficult to solve using computational methods like algorithms.

Example

An example of an ill-defined problem is as follows:

- **Problem:** How can we reduce poverty in Pakistan?

www.ilmkidunya.com

- **Goal:** Reduce poverty which is broad and not clearly defined.
- **Input:** It is not clearly defined and may include economic data or government policies etc.
- **Process:** There is no single method to solve the problem. It may require the economic, social and political reforms which are not defined.
- **Output:** It is difficult to measure and varies depending on the criteria.

Q. Differentiate between well-defined and ill-defined problems.

The difference between well-defined and ill-defined problems is as follows:

Well-defined Problems	Ill-defined Problems
1. A well-defined problem has clear goals.	1. An ill-defined problem does not have clear goals.
2. It includes specific and clearly defined inputs.	2. It has vague, incomplete or varying inputs.
3. It follows a known step-by-step method to solve the problem.	3. It does not provide specific method to solve the problem.
4. It produces definite output.	4. It may produce multiple outputs.
5. It is suitable for computer that solves it using algorithms and programming.	5. It is difficult for computer to solve without additional human guidance.
6. An example includes checking if a number is odd or even.	6. An example includes reducing poverty in a country.

Q. What is an algorithm and how are algorithms used for problem solving? Also discuss "Generate and Test" algorithm.

Algorithm for Problem Solving

An **algorithm** is a step-by-step procedure used to solve a problem or perform a specific task. It is similar to a recipe in cooking where each instruction must be followed in a particular order to achieve the desired result. In computing, algorithms are used to process input data, perform calculations and produce output efficiently and accurately. The understanding of algorithms is important for the following reasons:

- They form the logic behind computer programs and software.
- They help to solve complex problems efficiently.
- They ensure accuracy and consistency in different applications.

Generate and Test Algorithm

The **Generate and Test** algorithm is a basic and powerful method for solving the problems. It generates possible solutions and tests each solution to see if it works. The process continues until a satisfactory solution is found or all possible solutions have been tested. Different heuristics or rules can be applied to reduce the number of generated solutions. It makes the process more efficient.

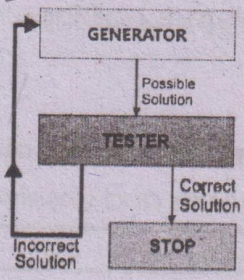
This technique is based on two main steps as follows:

1. Generating the potential solutions of the problem.
2. Testing the solutions to see if they meet the required conditions.

www.ilmkidunya.com

The **Generate and Test** algorithm is useful in the scenarios where:

- The problem space is small and it is feasible to generate and test all possible solutions.
- There is no clear strategy for finding a solution and a search is necessary.



The Generate and Test algorithm is often used in **Artificial Intelligence (AI)** and game-based problem-solving. For example, it is used in the games like chess or puzzle-solving where the computer must try different moves or arrangements to find a winning solution.

Q. What is meant by problem solvability and complexity? Explain the solvable and unsolvable problems with examples.

Problem Solvability and Complexity

Problem solvability refers to whether a problem can be solved by an algorithm. A problem is solvable if a finite step-by-step procedure exists that leads to a solution. **Problem complexity** refers to how efficiently a problem can be solved. It considers the resources such as time and memory required by an algorithm to solve the problem. These concepts are important when designing the algorithms and evaluating their performance.

Solvable vs. Unsolvable Problems

The problems are generally classified into two broad categories named solvable and unsolvable. They are based on whether an algorithm exists that can solve them for all possible inputs.

Solvable Problems

A **solvable problem** is a problem that can be solved by an algorithm in a finite amount of time. These problems have:

- Clearly defined inputs and outputs
- A logical procedure (algorithm) to follow
- A guaranteed result after a certain number of steps

Example

Finding the greatest common divisor (GCD) of two integers is an example of a solvable problem. The Euclidean algorithm is a step-by-step method that efficiently finds the GCD and always completes in a finite number of steps.

Unsolvable Problems

An **unsolvable problem** is a problem for which no algorithm can be created that solves the problem for all possible cases or inputs. These problems do not have a general procedure that can guarantee a solution for every possible input. They often involve situations that are too complex or undefined for a computer to handle.

www.ilmkidunya.com

Example

The **Halting Problem** is a well-known example of unsolvable problem. It is used to determine whether a given computer program will eventually stop running or continue to run forever for a specific input. Alan Turing proved that there is no general algorithm that can correctly solve this problem for all inputs. Therefore, the Halting Problem is considered unsolvable.

www.ilmkidunya.com

Q. Differentiate between solvable and unsolvable problems.

The difference between solvable and unsolvable problems is as follows:

Solvable Problems	Unsolvable Problems
1. A solvable problem is a problem that can be solved by an algorithm in a finite amount of time.	1. An unsolvable problem is a problem for which no algorithm can be created that solves it for all possible inputs.
2. It can be completed in a limited number of steps.	2. It may require infinite steps or cannot be solved at all.
3. It always produces a definite and correct output.	3. It does not guarantee an output for every possible input.
4. These problems can be handled efficiently by computers using algorithms and programs.	4. These problems cannot be solved completely by any computer or program.
5. An example of a solvable problem is finding Greatest Common Divisor of two numbers using Euclid's algorithm.	5. An example of an unsolvable problem is the Halting Problem introduced by Alan Turing.

www.ilmkidunya.com

Q. Explain tractable and intractable problems with examples.

Tractable Problems

A **tractable problem** is a problem that can be solved in a reasonable amount of time even if the input size grows. These problems can be solved using algorithms that run in polynomial time. **Polynomial time** means that the time taken to solve the problem increases at a manageable rate as the input size grows. It makes it possible to solve tractable problems using a computer. Tractable problems are considered efficiently solvable.

Example

Sorting a list of numbers using algorithms such as **Merge Sort** or **Quick Sort** is an example of tractable problem. These algorithms have time complexity of **O(n log n)** where **n** is the number elements in the list. It means that these algorithms can handle large inputs efficiently. The list can be sorted in reasonable amount of time even if it contains a large number of elements.

www.ilmkidunya.com

Intractable Problems

An **intractable problem** is a problem that becomes extremely difficult and time-consuming to solve as the size of the input increases. These problems often require super-polynomial or exponential time. It means that these problems are impractical to solve for large inputs because the time required becomes unmanageable.

Example

The **Travelling Salesman Problem (TSP)** is an example of intractable problem. The task of this problem is to find the shortest possible route to visit a set of cities and return to the starting point. The number of possible routes grows extremely fast as the number of cities increases. It becomes impossible to solve it for a large number of cities.

Q. Describe different classes of problem complexity with examples.

Complexity Classes

The complexity of problems can be classified into different categories based on their solvability and the time required to solve them.

Different categories of complexity classes are as follows:

1. Class P

Class P refers to a category of problems that can be solved efficiently by a computer. It means that the computer can find a solution of these problems quickly even if the size of the problem grows.

Example

A simple example of class P problem is sorting a list of numbers. Suppose the given list of numbers is as follows:

[4, 1, 3, 2, 5]

The goal is to arrange these numbers in ascending order like [1, 2, 3, 4, 5]. The time required to sort the list grows at a manageable rate as the list size increases. For example, sorting a list of 10 numbers will require more time to sort but it remains within a reasonable limit.

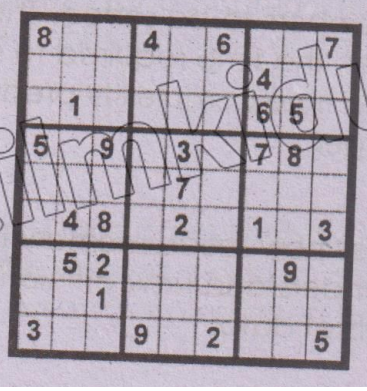
2. Class NP

Class NP refers to a category of problems for which the given solution can be checked by a computer quickly. The proposed solution of these problems can be verified easily but finding that solution can be difficult and time-consuming.

Example

An example of class NP problem is solving a Sudoku puzzle that consists of 9 x 9 grid. It is pre-filled with some numbers. The user needs to fill the blank cells with the numbers using the following rules:

- No duplicate numbers in any row
- No duplicate numbers in any column
- No duplicate numbers in any 3×3 square



www.ilmkidunya.com

3. Class NP-Hard

Class NP-hard refers to a category of problems that are at least as difficult as the hardest problems in **Non-deterministic Polynomial time (NP)**. Solving an NP-hard problem is challenging and no efficient algorithm is available to find a solution.

Example

The **Travelling Salesman Problem (TSP)** is an example of **Class NP-hard** problem. The task of this problem is to find the shortest possible route to visit a set of cities and return to the starting point. The number of possible routes grows extremely fast as the number of cities increases. It becomes impossible to solve it for a large number of cities.

4. NP- Complete

An **NP-Complete problem** is a problem that is both in NP and as hard as the hardest problems in NP. This means that if one NP-Complete problem can be solved efficiently then all NP problems can also be solved efficiently. NP-Complete problems form a subset of NP and are also considered NP-Hard. Solving an NP-Hard problem is considered computationally very difficult and no known algorithm can solve all NP-Hard problems efficiently. Many of these problems do not even have guaranteed solutions that can be verified in polynomial time.

Example

An example of NP-Complete problem is the Knapsack Problem. In this problem, there is a knapsack with a maximum weight capacity and a set of items with a weight and a value. The goal is to determine the most valuable combination of items to put in the knapsack without exceeding its weight capacity.

Q. What is algorithm analysis? Describe time and space complexity with examples.

Algorithm Analysis

Algorithm analysis is the study of how efficiently an algorithm performs in terms of time and space. This analysis is used to predict the algorithm's performance. It is very important for selecting the best algorithm for a particular task.

Time Complexity

Time complexity measures how the runtime of an algorithm increases as the input size grows. It helps to understand the efficiency of an algorithm when it handles large amounts of data.

Example

Suppose there is a list of numbers to be sorted. The task may be quick if the list has only a few numbers. However, the time required to sort the numbers increases as the volume of numbers increases. Time complexity predicts how this runtime will grow as the size of the list becomes larger.

Space Complexity

Space complexity measures how much memory is used by an algorithm as the input size increases. It helps to understand how efficiently an algorithm uses memory when it deals with larger amounts of data.

www.ilmkidunya.com

Example

Suppose an algorithm needs to store a list of numbers. The space complexity tells the amount of memory required as the volume of numbers increases.

Q. Explain the concept of time complexity with the help of Big O notation and suitable examples.

Big O Notation

Big O notation is a mathematical way to describe the time or space complexity of an algorithm. It describes how the runtime or memory usage of an algorithm grows as the input size increases. This notation helps to compare the efficiency of different algorithms based on their scalability. **Scalability** refers to how well an algorithm handles increasing input sizes.

How Big O Notation Works

Big O notation uses symbols to describe how the runtime of an algorithm changes with the size of the input. Some common examples are as follows:

1. O(1) Constant Time

An algorithm has **O(1) time complexity** when its runtime does not change with the size of the input. The **O(1)** means that an algorithm takes the same amount of time to run for any size of the input.

2. O(n) Linear Time

An algorithm has **O(n) time complexity** when its runtime increases directly in proportion to the size of the input. This means that if the input size doubles, the time required also approximately doubles.

Suppose there are **n** students in a college with a unique student ID. A specific student can be found by searching through the list of all **n** students. The time required to find a specific student will depend on the number of students.

3. $O(n^2)$ Quadratic Time

An algorithm has **$O(n^2)$ time complexity** when its runtime grows quadratically as the size of the input increases. The **$O(n^2)$** means that as the input size **n** increases, the runtime grows proportionally to the square of **n**. For example:

- If **n** doubles, the time increases by 4 times.
- If **n** triples, the time increases by 9 times and so on.

Suppose **n** students in a college are participating in a programming competition. The performance of each pair of students needs to be compared to determine the best team. Since each student must be compared with every other student exactly once, the number of required comparisons can be calculated as follows:

$$\frac{n(n-1)}{2}$$

4. $O(\log^n)$ Logarithmic Time

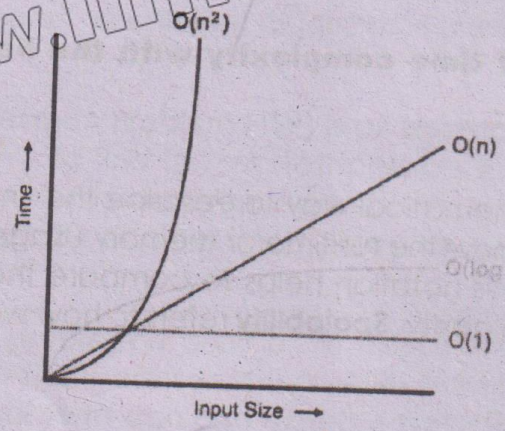
An algorithm has **O(log n) time complexity** when its runtime increases very slowly as the size of the input grows. Suppose a person is thinking a number between 1 and 100 and the other person is guessing the number. He can use the following questions:

- is it greater than 50?

www.ilmkidunya.com

- is it less than 25?
- is it less to 12?

Each question reduces the range and makes it easier to guess the number.



Q. Discuss the differences between time complexity and space complexity. How do they impact the choice of an algorithm for a specific problem?

Time complexity	Space complexity
1. Time complexity refers to how long an algorithm takes to complete as the input size increases.	1. Space complexity refers to how much memory an algorithm uses as the input size increases.
2. It measures number of steps taken by an algorithm to perform a task.	2. It measures amount of memory used by an algorithm to perform a task.
3. It focuses on the speed or runtime efficiency of an algorithm.	3. It focuses on the memory or storage efficiency of an algorithm.
4. It is useful when processing speed is critical such as in real-time systems.	4. It is useful when memory is limited such as in mobile devices.
5. It is expressed in terms of operations relative to input size such as O(n), O(n²), O(log n).	5. It is expressed in terms of memory usage relative to input size such as O(1), O(n)).

Impact on Algorithm Selection

Both time and space complexity are important when choosing an algorithm to solve a problem. The choice depends on the environment in which the algorithm will be applied. An algorithm with low space complexity is preferred if memory is limited such as in mobile devices. An algorithm with low time complexity is more appropriate if speed is crucial such as in gaming or real-time processing.

There are cases where improving time efficiency increases memory usage or saving memory slows down execution. This is known as the **time-space trade-off**. It refers to the practice of balancing between time and space by using more memory to reduce computation time or accepting slower execution in order to save memory. For example, a sorting algorithm called Merge Sort has time complexity O(n log n) and space complexity O(n). It means it is faster but uses more memory.

www.ilmkidunya.com

Q. What is algorithm design technique? List the names of different algorithm design techniques.

Algorithm Design Techniques

Algorithm design is the process of developing step-by-step methods to solve computational problems efficiently. It involves selecting suitable techniques to ensure correctness, efficiency and simplicity.

Different algorithm design techniques are as follows:

1. Divide and Conquer
2. Greedy Algorithms
3. Dynamic Programming
4. Backtracking

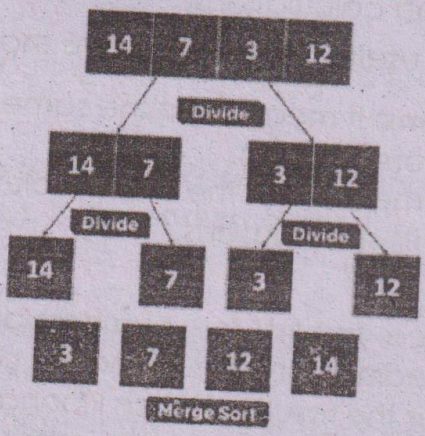
Q. What is Divide and Conquer algorithm design technique? Explain with an example.

Divide and Conquer

Divide and Conquer is a powerful algorithm design technique. It divides a large problem into smaller and more manageable parts. Each smaller part is solved independently. All solutions are then combined to solve the original problem. This approach is very effective for the problems that can be divided into smaller problems. This makes it easier to find a solution step by step.

Example

Merge Sort uses the divide and conquer approach to sort the list. It first divides the list into smaller halves until each part has only one element. It then sorts these parts and merges the sorted parts back together into a single sorted list.



Q. What is Greedy algorithm design technique? Explain with an example.

Greedy Algorithms

A **greedy algorithm** is a problem-solving strategy that builds a solution by always choosing the best available option at each stage. Each choice is locally optimal with the hope that these choices will lead to a globally optimal solution. Greedy approach is often used when a problem has an optimal substructure. It means that if a problem can be divided into smaller parts and solved optimally, then combining those optimal parts gives the best overall solution.

www.ilmkidunya.com

Greedy algorithms are often faster and easier to implement than other techniques. However, they do not always guarantee the optimal solution for every problem. It is important to analyze the problem to ensure that a greedy approach is appropriate.

Example

A classic example of a greedy algorithm is the Coin Change problem. The goal of this problem is to use the fewest number of coins to make the specific amount.

The greedy algorithm works as follows:

1. Pick the largest coin that does not exceed remaining amount.
2. Subtracts that value from the remaining amount.
3. Repeat the process until the remaining amount is zero.

Suppose the desired amount is $27 and the available coins are $25, $10, $5, $1. The greedy algorithm will solve the problem as follows:

1. Pick the largest coin that does not exceed remaining amount: $25
2. Subtracts that value from the remaining amount: $27 – $25 = $2
3. Pick the largest coin that does not exceed remaining amount: $2
4. Subtracts that value from the remaining amount: $2 – $2 = 0
5. The problem is solved using two coins of $25 and $2.

Q. What is dynamic programming? Explain with example.

Dynamic Programming

Dynamic Programming is a method for solving complex problems by breaking them into smaller and simpler subproblems. It solves each subproblem once and stores the results to avoid repeating the calculations.

Dynamic programming is useful for the problems that have:

- **Overlapping subproblems:** It means that the same subproblems appear again and again during the computation.
- **Optimal substructure:** It means that the overall solution of the problem depends on the best solutions of its smaller parts.

Example

In the Fibonacci sequence, the first few numbers are: 0, 1, 1, 2, 3, 5, 8, 13. Each number is the sum of two preceding numbers. Dynamic programming can be applied to Fibonacci sequence. It stores the results of each Fibonacci number as it is computed instead of recalculating it repeatedly. It reduces number of calculations significantly.

Q. What is backtracking in algorithm design? Explain with example.

Backtracking

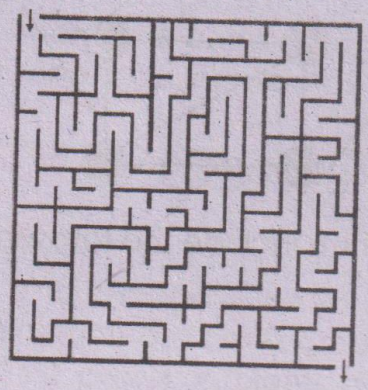
Backtracking is a problem-solving technique that builds a solution step by step. It tries a path but leaves it as soon as it determines that the path cannot lead to a valid solution. It then goes back to try a different path. This process continues until a solution is found or all possible options have been explored.

www.ilmkidunya.com

Example

Maze is an example where backtracking can be applied to solve the problem. The goal is to find a path from the start to the end.

1. Start from the initial position.
2. Try moving in one of the possible directions.
3. Move back if you hit a wall or it has already been visited.
4. Repeat the process until the end point is reached.



Q. What are the common computing algorithms? List some common computing algorithms.

Common Computing Algorithms

Different problems require different types of algorithmic techniques. Different programmers use different algorithms to solve the same problem. Many algorithms are considered fundamental and are commonly used in solving daily life problems. These algorithms are essential tools that help in performing a variety of tasks efficiently and effectively.

Some common computing algorithms are as follows:

1. Sorting Algorithms

Sorting algorithms are techniques used to arrange data in a particular order such as ascending or descending. In **ascending order**, the elements are arranged from the smallest to the largest such as [3, 7, 10, 15]. In **descending order**, the elements are arranged from the largest to the smallest such as [20, 12, 5, 1]. Sorting is a fundamental operation that is often required for other tasks such as searching, data analysis and organizing information efficiently.

Two common sorting algorithms include the following:

- Insertion sort algorithm
- Bubble sort algorithm

2. Searching Algorithms

Searching algorithms are designed to find specific elements or a set of elements within a dataset. They are critical for tasks such as information retrieval, database queries and decision-making processes. Efficient search methods save time and improve program performance.

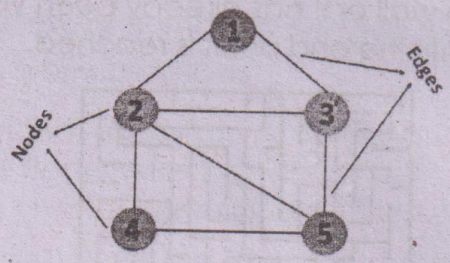
Two common searching algorithms include the following:

- Binary search algorithm
- Linear search algorithm

www.ilmkidunya.com

3. Graph Algorithms

Graph algorithms are used to explore and analyze graphs. **Graph** is a type of data structures that consists of nodes or vertices that connected by edges. The **nodes** refer to the individual items in the data structure. The **edges** refer to the connections or relationships between nodes. The graph algorithms are used for network analysis, route planning and social network analysis.



Two graph traversal algorithms include the following:

- Breadth-First Search (BFS)
- Depth-First Search (DFS)

Q. Describe the working of bubble sort algorithm with an example. Also briefly explain its complexity.

Bubble Sort

Bubble sort is one of the simplest sorting algorithms. It repeatedly visits the list and compares two items at a time. It swaps these the items if they are in the wrong order. This process continues until no swaps are needed and the list is sorted. The algorithm sorts the list using **n-1** passes where **n** represents the number of values in the list.

The bubble sort works as follows:

- Start from the beginning of the list.
- Compare each pair of adjacent elements.
- Swap them if they are in the wrong order.
- Continue this process until no more swaps are needed.

Example

Suppose the following list needs to be sorted in ascending order:

10	30	15	25	5

The bubble sort algorithm will use 4 passes as the number of values in the list is 5. It will work as follows:

Pass 1:

1. The first and second elements are compared. The values are not swapped as 10 is smaller than 30. There is no change in the list.
2. The second and third elements are compared. The values are swapped as 30 is bigger than 15. The list will be as follows:

10	15	30	25	5

3. The third and fourth elements are compared. The values are swapped as 30 is bigger than 25. The list will be as follows:

10	15	25	30	5

www.ilmkidunya.com

4. The fourth and fifth elements are compared. The values are swapped as 30 is bigger than 5. The first pass completes and the last element is finalized as follows:

10	15	25	5	30

Pass 2:

1. The first and second elements are compared. The values are not swapped as 10 is less than 15. There is no change in the list.
2. The second and third elements are compared. The values are not swapped as 15 is less than 25. There is no change in the list.
3. The third and fourth elements are compared. The values are swapped as 25 is bigger than 5. The second pass completes and the last two elements are finalized as follows:

10	15	5	25	30

Pass 3:

1. The first and second elements are compared. The values are not swapped as 10 is less than 15. There is no change in the list.
2. The second and third elements are compared. The values are swapped as 15 is bigger than 5. The third pass completes and the last three elements are finalized as follows:

10	5	15	25	30

Pass 4:

1. The first and second elements are compared. The values are swapped as 10 is bigger than 5. The last pass completes and all elements are finalized as follows:

5	10	15	25	30

Complexity

The time complexity of bubble sort in worst case is $O(n^2)$ that makes it inefficient for large datasets. However, it is easy to understand and implement. It is useful for small datasets.

Q. Describe the working of selection sort algorithm with an example. Also briefly explain its complexity.

Selection Sort

Selection Sort is another simple sorting algorithm. It selects the smallest or largest element from the unsorted part of the list and swaps it with first element of the unsorted part. Selection sort works as follows:

1. Find the minimum or maximum value in the list. The minimum value is used for ascending sort whereas maximum value is used for descending sort.
2. Swap it with the first unsorted element.
3. Move the boundary of the sorted and unsorted sections by one element.
4. Repeat the process for the remaining elements.

Example

Suppose the following list needs to be sorted in ascending order:

56	78	33	81	12

www.ilmkidunya.com

The selection sort algorithm will use 4 passes as the number of values in the list is 5. It will work as follows:

Pass 1:

1. The first pass finds the minimum value from the unsorted list which is 12.
2. It swaps the first element with the minimum value. The list will be as follows:

12	78	33	81	56

Pass 2:

1. The second pass finds the minimum value from the unsorted list which is 33.
2. It swaps the second element with the minimum value. The list will be as follows:

12	33	78	81	56

Pass 3:

1. The third pass finds the minimum value from the unsorted list which is 56.
2. It swaps the third element with the minimum value. The list will be as follows:

12	33	56	81	78

Pass 4:

1. The fourth pass finds the minimum value from the unsorted list which is 78.
2. It swaps the fourth element with the minimum value. The list will be as follows:

12	33	56	78	81

Complexity

The time complexity of selection sort in worst case is $O(n^2)$ that makes it inefficient for large datasets. However, it is easy to understand and implement. It is useful for small datasets.

Q. Differentiate between bubble sort and selection sort.

The difference between bubble sort and selection sort is as follows:

Bubble Sort	Selection Sort
1. Bubble Sort repeatedly compares and swaps adjacent elements if they are in the wrong order.	1. Selection Sort repeatedly finds the smallest or largest element and places it in the correct position.
2. Swapping happens many times in each pass as adjacent elements are compared and exchanged.	2. Swapping happens only once per pass after finding the correct element.
3. Its time complexity is $O(n^2)$ in the worst and average cases.	3. Its time complexity is also $O(n^2)$ in the worst and average cases.
4. It has a space complexity of O(1) as it sorts in place.	4. It also has a space complexity of O(1) as it does not use extra memory.
5. It performs well (O(n)) if list is already sorted and optimized with no swaps.	5. It does not improve and remains $O(n^2)$ even for the sorted data.
6. It is less efficient due to many swaps even when elements are sorted.	6. It is more efficient in terms of the number of swaps performed.

Q. Describe the working of linear search with an example. Briefly explain its complexity.

Linear Search

A **linear search** is a simple method for finding an item in a list. It checks each item one by one until the required item is found. It works on both sorted and unsorted data.

www.ilmkidunya.com

The linear search works as follows:

1. **Start at the Beginning:** Check the first item in the list.
2. **Check Each Item:** Compare the required item with the current item.
3. **Move to the Next:** Move to the next item in the list if the item is not found.
4. **Repeat:** Continue this process until the desired item is found or the list ends.

Example

Suppose the word "Lahore" needs to be searched from the following list:

Islamabad	Karachi	Lahore	Peshawar	Quetta

The linear search will work as follows:

1. Check the first element in the list. The first element is not "Lahore" so move to the next element.
2. The second element is not "Lahore" so move to the next element.
3. The third element is "Lahore" so the required word is found in the list and the search process ends.

Complexity

The time complexity of linear search is **O(n)** where **n** is the number of elements in the list. It is less efficient for large datasets because it may need to check every item.

Q. Describe the working of binary search with an example. Briefly explain its complexity.

Binary Search

Binary search is an efficient algorithm for finding an item in a sorted list. It is only effective on sorted lists. It works by repeatedly dividing the search interval in half. The process continues until the item is found or the interval is empty.

Binary search works as follows:

1. It starts with the middle element of the sorted list.
2. If the middle element is the required value, it returns its position.
3. If the required value is smaller than the middle element, it repeats the search on the left half of the list.
4. If the required value is greater than the middle element, it repeats the search on the right half of the list.
5. The process continues until the required number is found or the list ends.

Example

Suppose the number 3 needs to be searched from the following list:

1	3	5	**7**	9	11	13

The binary search will work as follows:

1. The search will start from middle element 7 which is not the required number.
2. The required number 3 is smaller than 7, so the search will continue to the left half of the list.

1	**3**	5	**7**	9	11	13

3. The middle element of the left half list is 3, which is the required number. The search will be completed successfully.

www.ilmkidunya.com

Complexity

The time complexity of binary search is **O(log n)** that means the search time grows slowly even with large datasets. This makes it much faster than linear search for sorted data.

Q. Discuss the characteristics of search problems and compare the linear search and binary search algorithms.

Characteristics of Search Problems

Search problems involve the task of finding a specific item or value within a given dataset. The process determines whether the required item exists in the data. It also identifies its location or position if the data exists.

Key Characteristics

1. **Input Data**: A dataset such as list or array where the search will be conducted.
2. **Target Value**: The specific item to be searched within the data.
3. **Search Strategy**: The algorithm or method applied to search the value such as linear search or binary search.
4. **Output**: The position of the desired value.

Comparison of Linear with Binary Search

The comparison of linear and binary search is as follows:

Linear Search	Binary Search
1. Linear Search works on both sorted and unsorted datasets.	1. Binary Search only works on sorted datasets.
2. It checks each element one by one from the beginning to the end.	2. It repeatedly divides the search range in half to locate the target.
3. Its time complexity is O(n) that means time grows linearly with large input size.	3. Its time complexity is O(log n) that means time grows slowly with large input sizes.
4. In best case, it can find the required value at the beginning with one comparison.	4. In best case, it can find the required value in the middle with one comparison.
5. It is less efficient for large datasets as it checks each item individually.	5. It is more efficient for large datasets as it reduces the search quickly.
6. It is easy to implement and understand.	6. It is more complex to implement and understand.

Q. Describe Breadth-First Search algorithm with an example. Briefly explain its complexity.

Breadth-First Search (BFS)

Breadth-First Search (BFS) is a graph traversal algorithm that explores all the nodes of a graph level by level. It starts from a given node often called **root**. It uses a queue to manage the nodes to be explored.

www.ilmkidunya.com

Process

The breadth-first search algorithm works as follows:

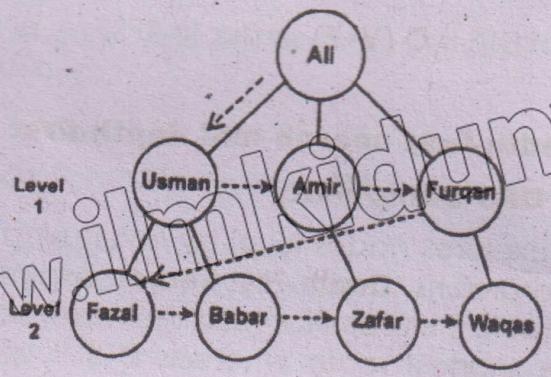
1. It starts from the root node and place it on a queue.
2. It takes the node from front of the queue, process it and places all its unvisited neighbours on the queue.
3. It repeats the process until all nodes are explored and the queue is empty.

Example

In a social network graph, each node represents a user and each edge between two nodes represents a friendship between these users. BFS algorithm can be used to find the shortest path between two users also known as **degree of separation**.

The following process finds the degree of separation between Ali and Babar:

1. Start at the root user Ali.
2. Visit all immediate friends of Ali which are Usman, Amir and Furqan. This is the first-degree connections.
3. Then visit the friends of those friends which are Fazal, Babar, Zafar and Waqas. This is the second-degree connections where the required user Babar is found.



The dotted arrows in the above figure shows the sequence of traversal.

Complexity

The time complexity of BFS is **O(V + E)** where **V** is the number of vertices and **E** is the number of edges. This makes it efficient for exploring large graphs.

Q. Describe Depth-First Search algorithm with an example. Briefly explain its complexity.

Depth-First Search (DFS)

Depth-First Search is a graph traversal algorithm that explores a branch as deep as possible before exploring other branches. It uses a stack to manage the nodes to be explored.

Process

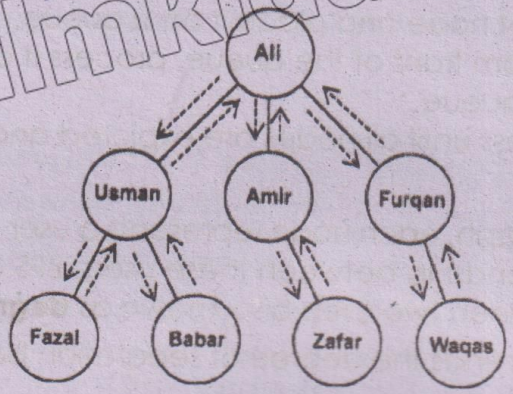
The depth-first search algorithm works as follows:

1. It starts from the root node and push it onto the stack.
2. It takes (pop) the node from the stack, process it and pushes all its unvisited neighbours on the stack.
3. It repeats the process until all nodes are explored and the stack is empty.

www.ilmkidunya.com

Example

The depth-first search algorithm works as follows:



The algorithm starts from the root node **Ali**. It visits **Ali** → **Usman** → **Fazal** → **Babar**. It then backtracks to **Ali**. Next, it visits **Amir** → **Zafar**. It again backtracks to **Ali** and then visits **Furqan** → **Waqas**.

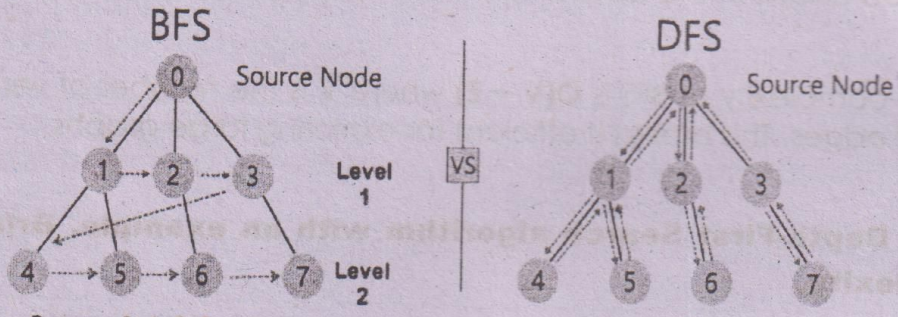
Complexity

The time complexity of DFS is **O (V+E)** similar to BFS.

Q. Briefly compare breadth-first search and depth-first search algorithms.

Comparison of BFS and DFS Complexity

Breadth-First Search explores nodes level by level using a queue. It is ideal for finding the shortest path in graphs. **Depth-First Search** explores as deep as possible along each path using a stack. It is useful for tasks like maze solving and backtracking problems. BFS ensures the shortest route whereas DFS is generally more memory-efficient.



Output: 0, 1, 2, 3, 4, 5, 6, 7

Output: 0, 1, 4, 5, 2, 6, 3, 7

Exercise Solution

Multiple-Choice Questions (MCQs)

1. **The characteristic of a well-defined problem is:**
   a) Ambiguous goals and unclear requirements
   b) Vague processes and inputs
   c) Clear goals, inputs, processes, and outputs
   d) Undefined solutions

www.ilmkidunya.com

2. **Complexity class representing the problems solvable efficiently by a deterministic algorithm is:**
   a) NP b) NP-hard c) NP-complete d) P
3. **The statement that applies to unsolvable problems is:**
   a) They can be solved in polynomial time
   b) They cannot be solved by any algorithm
   c) They are always in NP class
   d) They require exponential time to solve
4. **The meaning of NP in computational complexity is:**
   a) Non-deterministic Polynomial time
   b) Negative Polynomial time
   c) Non-trivial Polynomial time
   d) Numerical Polynomial time
5. **Search algorithm more efficient for large datasets is:**
   a) Bubble Sort b) Merge Sort
   c) Selection Sort d) Quick Sort
6. **A scenario where Dynamic Programming proves most useful is:**
   a) Problems without overlapping subproblems
   b) Problems solved by making local choices
   c) Problems with overlapping subproblems and optimal substructure
   d) Problems divided into independent subproblems
7. **An algorithm that sorts data by stepping through the list and swapping adjacent elements if needed is:**
   a) Selection Sort b) Quick Sort
   c) Bubble Sort d) Merge Sort
8. **Time complexity of Depth-First Search (DFS) in a graph is:**
   a) O(n log n) b) O(V) c) O(V + E) d) O(n)
9. **Best description of time complexity is:**
   a) Amount of memory an algorithm needs
   b) Time taken as a function of input size
   c) Efficiency as input size grows
   d) Upper bound of space requirements
10. **An algorithm with a time complexity of O(n log n) is:**
    a) Bubble Sort b) Binary Search
    c) Merge Sort d) Insertion Sort

Answers

1. c	2. d	3. b	4. a	5. d	6. c
7. c	8. c	9. b	10. c		

Short Questions

Q.1. Differentiate between well-defined and ill-defined problems within the realm of computational problem-solving.

A well-defined problem has clear goals whereas an ill-defined problem does not have a clearly stated goal. A well-defined problem includes specific and clearly defined inputs but an ill-defined problem has vague, incomplete or varying inputs. A well-defined problem follows a known step-by-step method to reach a solution but an ill-defined problem does not provide specific method to solve the problem.

Q.2. Outline the main steps involved in the Generate and Test algorithm.

The main steps involved in Generate and Test algorithm are Generate and Test. The Generate is used to find the possible solutions to a problem. The Test is used to test each solution to see if it satisfies the desired goal or condition.

www.ilmkidunya.com

Q.3. Compare tractable and intractable problems in the context of computational complexity.

Tractable problems can be solved in a reasonable amount of time using efficient algorithms. They typically take polynomial time complexity such as $O(n^2)$ or $O(n \log n)$. Intractable problems require excessive time as input size grows. They often take exponential time complexity such as $O(2^n)$.

Q.4. Summarize the key idea behind Greedy Algorithms.

Greedy algorithms build a solution by always choosing the best available option at each stage. Each choice is locally optimal with the hope that these choices will lead to a globally optimal solution. Greedy approach is often used when a problem has an optimal substructure.

Q.5. Discuss the advantages of using Dynamic Programming.

The advantage of dynamic programming is that it solves each subproblem once and stores the results to avoid repeating the calculations. This saves time, especially in problems with overlapping subproblems such as Fibonacci. It also finds the most efficient or optimal solution.

Q.6. Compare the advantages of Breadth-First Search (BFS) with Depth-First Search (DFS) in graph traversal.

Breadth-First Search explores nodes level by level using a queue. It is ideal for finding the shortest path in graphs. Depth-First Search explores as deep as possible along each path using a stack. It is useful for tasks like maze solving and backtracking problems. BFS ensures the shortest route whereas DFS is generally more memory-efficient.

Q.7. Explain the importance of breaking down a problem into smaller components in algorithmic thinking.

Breaking down a problem into smaller components is a key part of algorithmic thinking. It makes solving complex problems easier by focusing on one part at a time. This approach improves clarity, supports code reuse and simplifies testing. It also helps to identify and fix errors more efficiently.

Q.8. Identify the key factors used to evaluate the performance of an algorithm.

The key factors used to evaluate the performance of an algorithm are time complexity and space complexity. Time complexity measures how fast the algorithm runs while space complexity measures how much memory it uses. Some other factors include scalability, accuracy and behavior in best, average and worst-case scenarios. They help to determine the efficiency of an algorithm and its suitability for different problems.

Long Questions

Q.1. Provide a detailed explanation of why the Halting Problem is considered unsolvable and its implications in computer science.

Halting Problem

The Halting Problem is a well-known example of unsolvable problem. It is used to determine whether a given computer program will eventually stop running or continue to run forever for a specific input. Alan Turing proved that there is no general algorithm that can correctly solve this problem for all program-input pairs. Therefore, the Halting Problem is considered unsolvable.

Alan Turing proved that no computer program can correctly answer the halting question for all other programs. He proved that there is no perfect way to check whether any given program will halt or run forever.

Implications in Computer Science

The key implications of halting program in computer science are as follows:

1. Limits of Computation

The Halting Problem proves that there are fundamental limits to what computers and algorithms can solve. Not every problem can be solved with an algorithm no matter how powerful the hardware is.

www.ilmkidunya.com

2. Program Verification Limitations

It is impossible to design a perfect program checker that can always determine whether another program will halt or run forever.

3. Security and Malware Detection

Security software tries to detect harmful behavior in programs. The Halting Problem shows that it is impossible to predict all future behaviors of a program that means no antivirus or security system can perfectly detect every malicious program in every situation.

Q.2. Discuss the characteristics of search problems and compare the efficiency of Linear Search and Binary Search algorithm. (See chapter for the answer)

Q.3. Discuss the nature of optimization problems and provide examples of their applications in real-world scenarios.

Optimization Problems

Optimization problems involve selecting the best solution from a set of possible choices according to a specific goal. The objective is to optimize a particular value such as cost, profit, time or distance while satisfying a set of constraints. Optimization problems are essential where resources are limited and decisions must be made efficiently to achieve the best outcome.

Examples

1. **Route Planning**: Optimization helps to find the shortest or fastest route for delivery trucks used in GPS navigation systems and logistics. For example, Google Maps uses optimization to suggest the best route.
2. **Scheduling:** It helps in allocating time slots or resources without conflicts to improve productivity. For example, airlines use it to schedule flights and crew. The schools use it to prepare exam timetables.
3. **Resource Allocation:** It is used to distribute limited resources like budget or materials effectively to maximize output or minimize costs. For example, project managers optimize how to allocate team members and funds.
4. **Manufacturing and Production:** It is used to improve production efficiency by reducing waste and minimizing costs while maintaining quality. For example, factories use it to determine the best combination of raw materials and production processes.

Q.4. Explain the process and time complexity of the Bubble Sort algorithm. Compare it with another sorting algorithm of your choice in terms of efficiency. (See chapter for answer)

Q.5. Discuss the differences between time complexity and space complexity. How do they impact the choice of an algorithm for a specific problem? (See chapter for answer)

SHORT QUESTIONS

Q.1. What is a computational problem?

A computational problem is a task or challenge that can be solved using a step-by-step procedure. This procedure is known as algorithm. An algorithm is a finite set of instructions that can be executed by the computer to solve the problem.

Q.2. What are the three key components of a computational problem?

Three key components of a computational probelm are input, process and output. Input refers to the data given to the algorithm. Process refers to the steps or rules that are applied to the input to generate the output. Output is the solution or result produced by the algorithm after processing the input.

Q.3. How are computational problems classified?

Computational problems are classified into decision problems, search problems, optimization problems and counting problems. Each category requires different methods and approaches to solve.

www.ilmkidunya.com

Q.4. What are decision problems?

Decision problems are problems with a yes or no answer. They check whether a given condition is met. For example, "Is a number even?" is a decision problem.

Q.5. What are well-defined problems?

A well-defined problem is a problem that has clear goals, inputs, processes and outputs. These problems are easy to understand because all aspects of the problem are known and unambiguous. They can be solved using a step-by-step method known as an algorithm.

Q.6. What is an ill-defined problem?

An ill-defined problem is a problem that does not have clearly stated goal, input, process or expected output. They are difficult to solve using computational methods like algorithms. For example, "how to reduce poverty" is vague and does not have a single defined solution.

Q.7. How does Google use algorithms in its search engine?

Google uses a complex algorithm called PageRank to rank web pages in search results. It evaluates both the number and quality of links pointing to a page. Pages with more high-quality links are considered more relevant. This helps Google display the most useful results to users.

Q.8. What is a Generate and Test algorithm?

The Generate and Test algorithm is a basic and powerful method for solving the problems. It generates possible solutions and tests each one to see if it works. The process continues until a satisfactory solution is found or all possible solutions have been tested.

Q.9. When is the Generate and Test algorithm most useful?

The Generate and Test algorithm is most useful when the problem space is small and it is feasible to generate and test all possible solutions. It is also useful when there is no clear strategy for finding a solution and a search is necessary.

Q.10. How is the Generate and Test algorithm used in AI?

The Generate and Test algorithm is often used in Artificial Intelligence (AI) and game-based problem-solving. For example, it is used in the games like chess or puzzle-solving where the computer must try different moves or arrangements to find a winning solution.

Q.11. What is meant by problem solvability in computer science?

Problem solvability refers to whether a problem can be solved by an algorithm. A problem is solvable if there exists a finite step-by-step procedure that leads to a solution.

Q.12. How is complexity related to problem-solving in computing?

Problem complexity refers to how efficiently a problem can be solved. It considers the resources such as time and memory required by an algorithm to solve the problem. These concepts are important when designing algorithms and evaluating their performance.

Q.13. What are the characteristics of solvable problems?

The solvable problems have clearly defined inputs and outputs, a logical procedure (algorithm) to follow and a guaranteed result after a certain number of steps.

Q.14. Give an example of a solvable problem.

Finding the greatest common divisor (GCD) of two integers is an example of a solvable problem. The Euclidean algorithm is a step-by-step method that efficiently finds the GCD and always completes in a finite number of steps.

Q.15. What is an unsolvable problem?

An unsolvable problem is a problem for which no algorithm can be created that solves the problem for all possible cases or inputs. These problems do not have a general procedure that can guarantee a solution for every possible input. They often involve situations that are too complex or undefined for a computer to handle.

Q.16. What is a well-known example of an unsolvable problem?

Halting Problem is a well-known example of unsolvable problem. It is used to determine whether a given computer program will eventually stop running or continue to run forever for a specific input. Alan Turing proved that there is no general algorithm that can correctly solve this problem for all program-input pairs. Therefore, the Halting Problem is considered unsolvable.

www.ilmkidunya.com

Q.17. What is meant by computational complexity?

Computational complexity refers to the amount of time and space needed to solve a problem using an algorithm. It helps determine whether a problem can be solved efficiently. Complexity is an important factor after determining if a problem is solvable.

Q.18. What are tractable problems in computer science?

A tractable problem is a problem that can be solved in reasonable amount of time even if the input size grows. These problems can be solved using algorithms that run in polynomial time.

Q.19. How does input size affect intractable problems?

A small increase in input size in intractable problems can lead to a big increase in computation time. These problems become extremely difficult and time-consuming to solve as the size of the input increases.

Q.20. What is the purpose of classifying problems into complexity classes?

Classifying problems into complexity classes helps us understand how difficult a problem is to solve and how much time or resources it may require. It also guides algorithm design and sets realistic expectations for solving large or complex problems.

Q.21. What is Class P in computational complexity?

Class P refers to a category of problems that can be solved efficiently by a computer. It means that the computer can find a solution of these problems quickly even if the size of the problem grows.

Q.22. What is Class NP in computational complexity?

Class NP refers to a category of problems for which the given solution can be checked by a computer quickly. These are problems where verifying a proposed solution is easy but finding that solution can be difficult and time-consuming.

Q.23. Give an example of an NP-Complete problem?

An example of NP-Complete problem is the Knapsack Problem. In this problem, there is a knapsack with a maximum weight capacity and a set of items with a weight and a value. The goal is to determine the most valuable combination of items to put in the knapsack without exceeding its weight capacity.

Q.24. What is algorithm analysis?

Algorithm analysis is the study of how efficiently an algorithm performs in terms of time and space. This analysis is used to predict the algorithm's performance. It is very important for selecting the best algorithm for a particular task.

Q.25. Why is algorithm analysis important?

Algorithm analysis is important because it is used to compare different algorithms based on their performance. It helps to determine how fast and how much memory an algorithm will use especially when the input data grows. This leads to better decision-making in programming.

Q.26. What is time complexity in algorithm analysis?

Time complexity measures how runtime of an algorithm increases as the input size grows. It helps to understand the efficiency of an algorithm when it handles large amounts of data.

Q.27. Give an example to explain time complexity.

Suppose there is a list of numbers to be sorted. The task may be quick if the list has only a few numbers. However, the time required to sort the numbers increases as the volume of numbers increases. Time complexity predicts how this runtime will grow as the size of the list becomes larger.

Q.28. Why is space complexity important?

Space complexity is important as it measures how much memory an algorithm uses as the input size increases. It helps to understand how efficiently an algorithm uses memory when it deals with larger amounts of data.

Q.29. How does space complexity differ from time complexity?

Time complexity refers to how long an algorithm takes to complete as the input size increases whereas space complexity refers to how much memory an algorithm uses as the input size increases.

www.ilmkidunya.com

Q.30. Why is Big O notation useful?

Big O notation is a mathematical way to describe the time or space complexity of an algorithm. It describes how the runtime or memory usage of an algorithm grows as the input size increases. This notation helps compare the efficiency of different algorithms based on their scalability. Scalability refers to how well an algorithm handles increasing input sizes.

Q.31. What does $O(n^2)$ time complexity mean?

$O(n^2)$ time complexity means that as the input size n increases, the runtime grows proportionally to the square of n. For example, if n doubles, the time increases by 4 times. If n triples, the time increases by 9 times and so on. It is less efficient for large datasets.

Q.32. List the names of different algorithm design techniques.

Different algorithm design techniques include Divide and Conquer, Greedy Algorithms, Dynamic Programming and Backtracking.

Q.33. What is the Divide and Conquer technique?

Divide and Conquer is a powerful algorithm design technique. It divides a large problem into smaller and more manageable parts. Each smaller part is solved independently. All solutions are then combined to solve the original problem.

Q.34. What is the key advantage of Divide and Conquer?

The key advantage of Divide and conquer approach is that it is very effective for the problems that can be divided into similar smaller problems. This makes it easier to find a solution step by step.

Q.35. Give an example of Divide and Conquer.

Merge Sort uses the divide and conquer approach to sort the list. It first divides the list into smaller halves until each part has only one element. It then sorts these parts and merges the sorted parts back together into a single sorted list.

Q.36. What is a limitation of greedy algorithms?

A key limitation of greedy algorithms is that they do not always guarantee the optimal solution for every problem. It is important to analyze the problem to ensure that a greedy approach is appropriate.

Q.37. What is Dynamic Programming (DP)?

Dynamic Programming is a method for solving complex problems by breaking them into smaller and simpler subproblems. It solves each subproblem once and stores the results to avoid repeating the calculations.

Q.38. Dynamic programming is useful for which problems?

Dynamic programming is useful for the problems that have overlapping subproblems and optimal substructure. Overlapping subproblems means that the same subproblems appear again and again during the computation. Optimal substructure means that the overall solution of the problem depends on the best solutions of its smaller parts.

Q.39. Give an example where dynamic programming is applied.

The Fibonacci sequence is an example where dynamic programming is applied. Each number in this sequence is the sum of two preceding numbers. Dynamic programming stores the results of each Fibonacci number as it is computed instead of recalculating it repeatedly.

Q.40. What is the main idea behind backtracking?

Backtracking is a problem-solving algorithm that builds a solution step by step. It tries a path but leaves it as soon as it determines that the path cannot lead to a valid solution. It then goes back to try a different path. This process continues until a solution is found or all possible options have been explored.

Q.41. What is an algorithm in computer science?

An algorithm is a step-by-step procedure to solve a problem or perform a task. Algorithms are used in computer programs to process data, perform calculations and make decisions.

Q.42. What are commonly used algorithms?

Commonly used algorithms include sorting algorithms, searching algorithms, and graph traversal algorithms. These are foundational for many computer science applications.

www.ilmkidunya.com

Q.43. What is the use of sorting algorithms? List two sorting algorithms.

Sorting algorithms are techniques used to arrange data in a particular order such as ascending or descending. Sorting is a fundamental operation that is often required for other tasks such as searching, data analysis and organizing information efficiently. Two common sorting algorithms include insertion sort and bubble sort.

Q.44. Why is sorting important before searching?

Sorting improves efficiency of search algorithms like binary search. When data is sorted, searches become faster and more predictable. It reduces the time required to locate an item.

Q.45. What is bubble sort?

Bubble sort is one of the simplest sorting algorithms. It repeatedly visits the list and compares two items at a time. It swaps these the items if they are in the wrong order. This process continues until no swaps are needed and the list is sorted.

Q.46. How does Bubble Sort work?

The bubble sort starts from the beginning of the list. It compares each pair of adjacent elements and swaps them if they are in the wrong order. It continues this process until no more swaps are needed.

Q.47. What is the time complexity of Bubble Sort?

The time complexity of bubble sort in worst case is $O(n^2)$ that makes it inefficient for large datasets. However, it is easy to understand and implement. It is useful for small datasets.

Q.48. What is Selection Sort?

Selection Sort is another simple sorting algorithm. It selects the smallest or largest element from the unsorted part of the list and swaps it with first element of the unsorted part.

Q.49. How does Selection Sort work?

Selection sort finds the minimum or maximum value in the list. The minimum value is used for ascending sort whereas maximum value is used for descending sort. It swaps it with the first unsorted element. It moves the boundary of the sorted and unsorted sections by one element. It repeats the process for the remaining elements.

Q.50. What is the time complexity of Selection Sort?

The time complexity of selection sort in worst case is $O(n^2)$ that makes it inefficient for large datasets. However, it is easy to understand and implement. It is useful for small datasets.

Q.51. What is a key difference between Bubble Sort and Selection Sort?

The key difference is that bubble sort repeatedly compares and swaps adjacent elements if they are in the wrong order. Selection sort repeatedly finds the smallest or largest element and places it in the correct position.

Q.52. What is the use of searching algorithms? List two search algorithms.

Searching algorithms are designed to find specific elements or a set of elements within a dataset. They are critical for tasks such as information retrieval, database queries and decision-making processes. Two search algorithms include linear search and binary search.

Q.53. What is linear search?

A linear search is a simple method for finding an item in a list. It checks each item one by one until the required item is found. It works on both sorted and unsorted data.

Q.54. How does linear search work?

The linear search starts at the beginning and checks the first item in the list. It compares the required item with the current item. It moves to the next item in the list if the item is not found. It continues this process until the desired item is found or the list ends.

Q.55. What is the time complexity of linear search?

The time complexity of linear search is $O(n)$ where n is the number of elements in the list. It is less efficient for large datasets because it may need to check every item.

Q.56. What is binary search?

Binary search is an efficient algorithm for finding an item in a sorted list. It is only effective on sorted lists. It works by repeatedly dividing the search interval in half. The process continues until the item is found or the interval is empty.

www.ilmkidunya.com

Q.57. How does binary search work?

Binary search starts with the middle element of the sorted list. If the middle element is the required value, it returns its position. If the required value is smaller than the middle element, it repeats the search on the left half of the list. If the required value is greater than the middle element, it repeats the search on the right half of the list.

Q.58. What is the time complexity of binary search?

The time complexity of binary search is O(log n) that means the search time grows slowly even with large datasets. This makes it much faster than linear search for sorted data.

Q.59. How does binary search improve efficiency?

Binary search reduces the number of comparisons by eliminating half of the remaining elements at each step. This makes it much more efficient than checking every item especially in large datasets.

Q.60. Why is Binary Search more efficient than Linear Search?

Binary search is faster because it repeatedly divides the list in half that reduces the size of the collection to be searched.

Q.61. Which search method should be used to find an item from an unsorted list of 10 items and why?

The linear search should be used because the list is unsorted. The linear search does not require the list to be sorted.

Q.62. Which search method is more efficient to search an item from a sorted list of 1,000 items?

Binary search is more efficient to search an item from a sorted list of 1,000 items. This is because binary search quickly narrows down the possible location of the item by repeatedly dividing the list in half. It quickly finds the required item from the list.

Q.63. Why linear search is preferred over binary search to search an item from unsorted list?

Linear search is preferred over binary search because the list is unsorted. The binary search cannot be used with an unsorted list.

Q.64. What are graph algorithms?

Graph algorithms are used to explore and analyze graphs. Graph is a type of data structures that consists of nodes or vertices that connected by edges. The nodes refer to the individual items in the data structure. The edges refer to the connections or relationships between nodes.

Q.65. What is Breadth-First Search (BFS)?

Breadth-First Search (BFS) is a graph traversal algorithm that explores all the nodes of a graph level by level. It starts from a given node often called root. It uses a queue to manage the nodes to be explored.

Q.66. How does BFS work?

The BFS starts from the root node and place it on a queue. It takes the node from front of the queue, process it and places all its unvisited neighbors on the queue. It repeats the process until all nodes are explored and the queue is empty.

Q.67. What is the time complexity of BFS?

The time complexity of BFS is O(V + E) where V is the number of vertices and E is the number of edges. This makes it efficient for exploring large graphs.

Q.68. What data structure is used in BFS?

BFS uses a queue data structure to manage the order of node exploration. This allows it to explore nodes in the order they are discovered level by level.

Q.69. What is Depth-First Search (DFS)?

Depth-First Search is a graph traversal algorithm that explores a branch as deep as possible before exploring other branches. It uses a stack to manage the nodes to be explored.

Q.70. How does DFS work?

DFS starts from the root node and push it onto the stack. It takes (pop) the node from the stack, process it and pushes all its unvisited neighbors on the stack. It repeats the process until all nodes are explored and the stack is empty.

www.linkkidunya.com

Q.71. What data structure is used in DFS?

DFS uses a stack to keep track of nodes to visit. This allows it to go deep into the graph before backtracking.

Multiple Choice Questions

1. **Which of the following refers to a challenge that can be solved through a computational process involving an algorithm?**
   a. Computational problem b. Programming language
   c. Binary conversion d. Debugging technique
2. **A step-by-step set of instructions to solve a problem is called a(n):**
   a. Algorithm b. Syntax c. Protocol d. Heuristic
3. **Which of the following is not a core component of a computational problem?**
   a. Input b. Output c. Debugging d. Process
4. **A problem with a "yes" or "no" output is called a:**
   a. Search problem b. Decision problem
   c. Optimization problem d. Counting problem
5. **A problem where the goal is to find the best solution according to specific criteria is classified as a(n):**
   a. Counting problem b. Optimization problem
   c. Search problem d. Decision problem
6. **Which type of problem is characterized by clear goals, inputs and outputs?**
   a. Ill-defined problem b. Abstract problem
   c. Open-ended problem d. Well-defined problem
7. **Determining if a number is even is a type of _____ problem.**
   a. Well-defined b. Search c. Optimization d. Ill-defined
8. **A problem with open-ended goals and no fixed solution is called:**
   a. Well-defined b. Algorithmic
   c. Ill-defined d. Optimization
9. **A problem that asks "How to reduce poverty in Pakistan" can be classified as:**
   a. Well-defined problem b Ill-defined problem
   c. Optimization problem d. Numerical problem
10. **What is the name of the algorithm used by Google search engine to rank web pages?**
    a. PageRank b. SearchMaster
    c. LinkRank d. WebCrawler
11. **The Generate and Test algorithm is most often used in which field?**
    a. Network security b. Artificial Intelligence
    c. Hardware circuit design d. Financial auditing
12. **What is a specific example of an AI application where Generate and Test algorithm is often used?**
    a. Sending emails b. Game playing
    c. File compression d. Virus detection
13. **_____ can be used reduce the number of potential solutions generated in a Generate and Test algorithm.**
    a. Compilers b. Heuristics
    c. Loops d. Syntax rules
14. **In computer science, how are problems formally categorized based on the existence of algorithmic solutions?**
    a. Solvable or Unsolvable b. Linear or Nonlinear
    c. Deterministic or Nondeterministic d. Sequential or Parallel
15. **A solvable problem in computer science must have:**
    a. Finite procedure b. At least 1000 steps
    c. Graphical user interface d. Cloud computing support

www.ilmkidunya.com

16. **Calculating the GCD (Greatest Common Divisor) of two numbers is a(n):**
    a. Ill-defined problem
    b. Solvable problem
    c. Unsolvable problem
    d. Non-computable function
17. **The Euclidean algorithm is used to solve which of the following computational problem?**
    a. Calculating square root
    b. Finding GCD of two numbers
    c. Sorting a list
    d. Finding factorial
18. **Which of the following is a famous example of an unsolvable problem?**
    a. Calculating square root
    b. Finding factorial
    c. Halting Problem
    d. Sorting numbers
19. **Which classic computational problem involves determining whether a program will halt or run infinitely for a given input?**
    a. Sorting problem
    b. Halting problem
    c. Search problem
    d. Optimization problem
20. **Who proved that the Halting Problem is unsolvable?**
    a. Bill Gates
    b. Alan Turing
    c. Charles Babbage
    d. Tim Berners-Lee
21. **A problem is considered computationally tractable if it can be solved in:**
    a. Infinite time
    b. Exponential time
    c. Constant time
    d. Polynomial time
22. **Intractable problems are those that cannot be solved in polynomial time and typically require:**
    a. Constant time
    b. Linear time
    c. Logarithmic time
    d. Super-polynomial time
23. **Which of the following is an example of a tractable problem?**
    a. Halting Problem
    b. Sorting a list using Merge Sort
    c. Traveling Salesman Problem (TSP)
    d. Boolean Satisfiability (SAT)
24. **The time complexity of merge sort is:**
    a. O (n!) b. O(n log n) c. $O(2^n)$ d. $O(n^2)$
25. **Which of the following sorting algorithms has polynomial time complexity in its typical implementation?**
    a. Bogo Sort
    b. Quick Sort
    c. Sleep Sort
    d. Brute-force Sort
26. **How does the time complexity of intractable problems typically scale with input size?**
    a. Exponentially
    b. Linearly
    c. Logarithmically
    d. Constant time
27. **Which of the following is a classic example of an intractable (NP-hard) problem?**
    a. Bubble Sort
    b. Euclidean Algorithm
    c. Travelling Salesman Problem
    d. Linear Search
28. **What happens to the number of possible routes in the Traveling Salesman Problem as the number of cities increases?**
    a. It stays the same
    b. It increases linearly
    c. It grows factorially
    d. It decreases
29. **For problems in class NP, finding a solution is:**
    a. Quick and easy
    b. Randomly determined
    c. Potentially time-consuming
    d. Always impossible
30. **Verifying a proposed solution to an NP problem is:**
    a. Slow
    b. Difficult to implement
    c. Efficient
    d. Theoretically impossible
31. **Which of the following is an example of an NP problem?**
    a. Sudoku Puzzle
    b. Knapsack Problem
    c. Calculating Factorials
    d. Adding integers
32. **NP-Complete problems lie at the intersection of:**
    a. P and NP-Hard
    b. P and NP
    c. NP and NP-Hard
    d. NP and Unsolvable

www.ilmkidunya.com

33. **A classic example of an NP-Complete problem is:**
a. Calculating Factorials b. Merge Sort
c. Knapsack Problem d. Finding GCD

34. **Which visual representation best shows how P, NP, NP-Complete and NP-Hard problems relate to each other?**
a. Tree diagram b. Bar chart
c. Venn diagram d. Flowchart

35. **The primary focus of algorithm analysis is studying:**
a. Input and output formatting b. Hardware performance
c. Time and space complexity d. Code readability

36. **Time complexity helps to understand how the running time of an algorithm changes with:**
a. Input format b. Input size
c. Output size d. Program language

37. **In Big O notation, what does the 'n' typically represent?**
a. Time b. Output size
c. Input size d. Memory usage

38. **Which of the following is NOT a standard Big O notation?**
a. $O(n)$ b. $O(n^2)$ c. $O(\log n)$ d. $O(n-1)!$

39. **Which of these Big O notations represents constant time complexity?**
a. $O(n)$ b. $O(1)$ c. $O(n^2)$ d. $O(\log n)$

40. **Which Big O notation describes the most efficient algorithm in terms of time complexity?**
a. $O(n^2)$ b. $O(2^n)$ c. $O(1)$ d. $O(\log n)$

41. **How is O(1) complexity represented on a runtime vs. input size graph?**
a. A steeply rising line b. A flat horizontal line
c. A parabolic curve d. A fluctuating line

42. **Which Big O notation describes an algorithm whose runtime grows linearly with input size?**
a. $O(n^2)$ b. $O(\log n)$ c. $O(n)$ d. $O(1)$

43. **What is the time complexity of linear search in the worst case?**
a. $O(1)$ b. $O(\log n)$ c. $O(n)$ d. $O(n^2)$

44. **If an algorithm processes each item in a list once, its time complexity is:**
a. $O(1)$ b. $O(\log n)$ c. $O(n)$ d. $O(n^2)$

45. **Which time complexity increases very slowly even as the input size becomes large?**
a. $O(n^2)$ b. $O(\log n)$ c. $O(n)$ d. $O(n!)$

46. **Which Big O notation represents quadratic time complexity?**
a. $O(n^2)$ b. $O(\log n)$ c. $O(n)$ d. $O(1)$

47. **Which time complexity grows the fastest as input size (n) increases?**
a. $O(n^2)$ b. $O(\log n)$ c. $O(n)$ d. $O(1)$

48. **_____ measures the memory required by an algorithm relative to input size.**
a. Time complexity b. Data structure
c. Space complexity d. Code optimization

49. **Which performance scenario does Big O notation primarily describe?**
a. Best-case performance b. Average-case performance
c. Worst-case performance d. Memory usage

50. **Which technique breaks a problem into smaller subproblems, solves each one, and combines their solutions to solve the original problem?**
a. Greedy method b. Divide and Conquer
c. Dynamic Programming d. Backtracking

51. **Which algorithm is a classic example of Divide and Conquer technique?**
a. Linear search b. Bubble sort
c. Merge sort d. Hashing

52. **Which method makes locally optimal choices at each step to reach a global solution?**
a. Divide and Conquer b. Greedy method
c. Dynamic Programming d. Brute Force

www.ilmkidunya.com

53. **Which technique may NOT produce optimal solutions in all cases?**
    a. Divide and Conquer
    b. Greedy
    c. Dynamic Programming
    d. Backtracking
54. **Which approach stores solutions of subproblems to avoid recomputing and improve efficiency?**
    a. Divide and Conquer
    b. Greedy
    c. Dynamic Programming
    d. Backtracking
55. **Which method explores all possible solutions systematically?**
    a. Divide and Conquer
    b. Greedy
    c. Dynamic Programming
    d. Backtracking
56. **Which problem is commonly solved using a greedy approach?**
    a. Binary Search
    b. Coin change problem
    c. Depth-First Search
    d. Merge Sort
57. **Which algorithm technique is used in solving puzzles like Sudoku?**
    a. Divide and Conquer
    b. Greedy
    c. Dynamic Programming
    d. Backtracking
58. **Which of the following is NOT an algorithm design technique?**
    a. Divide and Conquer
    b. Greedy
    c. Dynamic Programming
    d. Compilation
59. **Which sorting algorithm repeatedly compares adjacent elements and swaps them if they are in the wrong order?**
    a. Quick Sort
    b. Bubble Sort
    c. Merge Sort
    d. Insertion Sort
60. **What is the worst-case time complexity of the Bubble Sort algorithm?**
    a. O(n) b. O(log n) c. O(n log n) d. $O(n^2)$
61. **Which sorting algorithm works by repeatedly selecting the smallest unsorted element and swapping it into its correct position?**
    a. Quick Sort
    b. Selection sort
    c. Bubble sort
    d. Insertion sort
62. **What is the worst-case time complexity of selection sort?**
    a. O(n) b. O(log n) c. O(n log n) d. $O(n^2)$
63. **Which of the following sorting algorithms is inefficient for large datasets due to its high time complexity?**
    a. Merge Sort
    b. Quick Sort
    c. Bubble Sort
    d. Heap Sort
64. **Which algorithm is most efficient for sorting large datasets?**
    a. Merge Sort
    b. Bubble Sort
    c. Selection Sort
    d. Both b and c
65. **Which search algorithm examines each element sequentially until it finds the target value or reaches the end of the list?**
    a. Binary Search
    b. Hashing
    c. Linear Search
    d. Merge Sort
66. **Binary search requires the input list to be:**
    a. Unsorted b. Sorted c. Partitioned d. Hashed
67. **The worst-case time complexity of Linear Search is:**
    a. O(log n) b. O(n) c. $O(n^2)$ d. O(1)
68. **What is the time complexity of Binary Search?**
    a. O(log n)
    b. O(n)
    c. $O(n^2)$
    d. O(n log n)
69. **Where does binary searc examining in the list?**
    a. First element
    b. Last element
    c. Middle element
    d. Random element
70. **Which search algorithm is better for unsorted data?**
    a. Linear Search
    b. Binary Search
    c. Both are equally efficient
    d. Neither

www.ilmkidunya.com