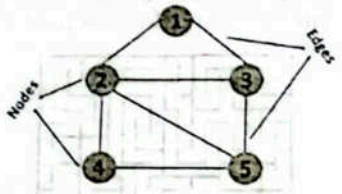
3. Graph Algorithms

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

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 $\mathbf{O(n^2)}$ 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

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.

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.

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.

Process

The breadth-first search algorithm works as follows:

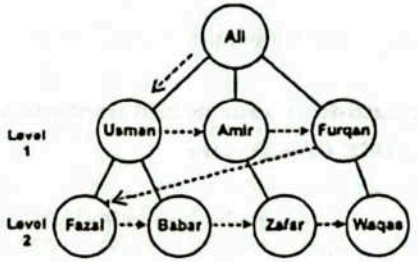
1. It starts from the root node and place it on a queue.
2. It takes the node from front of the queue, process it and places all its unvisited neighbours on the queue.
3. It repeats the process until all nodes are explored and the queue is empty.

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.

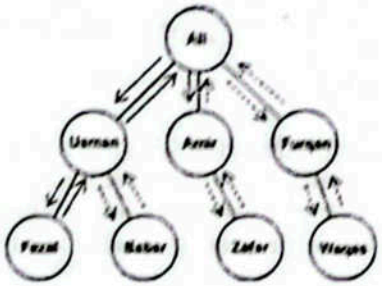
Process

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.

Example

The depth-first search algorithm works as follows:



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

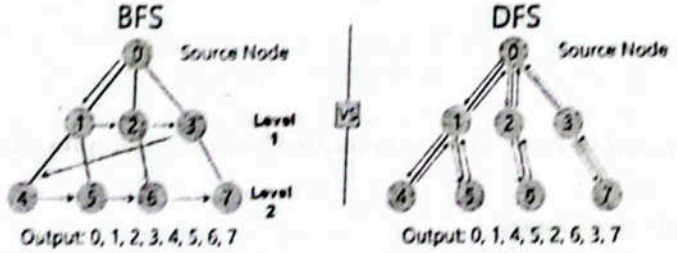
Complexity

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.



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

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.

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.

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.

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.

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.

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.

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.

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.

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

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 (nl) 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

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(nl)
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

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 search begin 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

71. **_____ search algorithm halves the search space at each step.**
a. Linear b. Binary c. Bubble d. Insertion

72. **In Binary search, if the target is less than the middle element, the algorithm:**
a. Stops b. Searches the left half
c. Searches the right half d. Restarts

73. **Which task is NOT typically solved using graph algorithms?**
a. Route planning b. Social network analysis
c. Text formatting d. Network exploration

74. **Which traversal algorithm explores all neighbors at current depth before going deeper?**
a. Depth-First Search b. Quick Sort
c. Breadth-First Search d. Merge Sort

75. **What data structure is essential for implementing BFS?**
a Queue b. Stack c. Heap d. Array

76. **Which graph traversal algorithm is used to find the shortest path in unweighted graph?**
a DFS b. BFS c. Both d. Neither

77. **Finding mutual friends in a social network is best done with:**
a DFS b. BFS
c. Binary Search d. Linear Search

78. **In DFS, when a dead end is reached, the algorithm:**
a. Restarts b. Backtracks
c. Jumps to a random node d. Terminates

79. **What data structure is used by Depth-First Search (DFS)?**
a. Queue b. Stack
c. Heap d. Priority Queue

80. **Which graph traversal method is most commonly used to solve maze/puzzle problems?**
a. DFS b. BFS
c. A* Search d. Dijkstra's Algorithm

81. **What is the time complexity of BFS on a graph with V vertices and E edges?**
a. $O(V^2)$ b. $O(E^2)$ c. O(V + E) d. O(V × E)

82. **Which algorithm is optimal for exploring graphs where solutions are near the root?**
a. Depth-First Search (DFS) b. Breadth-First Search (BFS)
c. Greedy Best-First Search d. Bidirectional Search

83. **Which graph traversal algorithm is most space-efficient for exploring deep graphs with long branches?"**
a. Depth-First Search (DFS) b. Breadth-First Search (BFS)
c. Uniform-Cost Search d. Depth-Limited Search

Answers

1. a	2. a	3. c	4. b	5. b	6. d
7. a	8. c	9. b	10. a	11. b	12. b
13. b	14. a	15. a	16. b	17. b	18. c
19. b	20. b	21. d	22. d	23. b	24. b
25. b	26. a	27. c	28. c	29. c	30. c
31. a	32. c	33. c	34. c	35. c	36. b
37. c	38. d	39. b	40. c	41. b	42. c
43. c	44. c	45. b	46. a	47. a	48. a
49. c	50. b	51. c	52. b	53. b	54. c
55. d	56. b	57. d	58. d	59. b	60. d
61. b	62. d	63. c	64. a	65. c	66. b
67. b	68. a	69. c	70. a	71. b	72. b
73. c	74. c	75. a	76. b	77. b	78. b
79. b	80. a	81. c	82. b	83. a	

CHAPTER 4

Computational Structures

Q. What is a list? Describe its properties and applications.

List

A **list** is a data structure in Python that can store multiple values. The list is **mutable** that means its elements can be created, accessed, modified or removed easily. Each item stored in a list is called an **element**. Lists can store elements of different data types including integers, floats, strings etc. Each element in the list can be accessed with reference to its position in the list. This position is called **Index**. Each element in the list has a unique index. The index of first element is 0 and the index of last element is length -1. The value of the index is written in brackets along with the name of the list.

The following example shows a list **Num** with three elements:

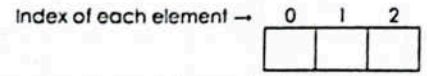


Figure: A list **Num** with three elements

List Properties

List has the following properties:

1. Dynamic Size

A list in Python can change its size. The new items can be added to the list and existing items can be removed easily. The size of the list automatically changes.

2. Index-Based Access

Every item in a list has a unique index that indicates its position in the list. The index is used to access specific items in the list easily.

3. Ordered Collection

The order in which items are added to a list is preserved. It means that the first item added to the list will stay in that position unless it is changed.

Applications of Lists

Lists are one of the most commonly used data structures in Python due to their flexibility and ease of use. Some important applications of lists are described below:

1. Data Storage and Manipulation

Lists are commonly used to store and manage collections of data like records, entries or values. The data in the lists can be inserted, deleted and accessed easily.

2. Stack and Queue Implementations

Lists can be used to implement the data structures known as stack and queue.

Stack — It is a data structure that follows the **Last In, First Out (LIFO)** principle. This means the last element added to stack is the first one to be removed.

Queue — It is a data structure that follows the **First In, First Out (FIFO)** principle. This means the first element added to queue is the first one to be removed.

Q. Describe some common operations of a list with examples.

Some common operations of a list include insertion, deletion and searching.

1. Insertion

The insertion operation is used to add a new item to a list. An item can be inserted at different positions in the list. The **insert()** method in Python is used to insert an item at a specific index in a list.

Syntax

The syntax of **insert()** method is as follows:

```
list_name.insert(index, element);
```

list_name — It indicates the name of the list.
index — It indicates the index where the element will be inserted in the list.

Example

```
country = ["Turkey", "Iran", "Oman"]
country.insert(0, "Pakistan")
print(country)
```

Output: ['Pakistan', 'Turkey', 'Iran', 'Oman']

2. Deletion

The deletion operation is used to remove an item from a list. The deletion of an item can be done using the value or index of the item.

a. Removing by Value

The **remove()** method is used to delete an item from the list by value. It removes the first occurrence of the specified value from the list.

Syntax

The syntax of **remove()** method is as follows:

```
list_name.remove(val)
```

list_name — It indicates the name of the list.
val — It indicates the value of the list element to be removed.

Example

```
fruit = ["apple", "banana", "cherry", "potato", "kiwi"]
fruit.remove("potato")
print(fruit)
```

Output: ['apple', 'banana', 'cherry', 'kiwi']

b. Removing by Index

The **pop()** method is used to delete an item from the list by index. The function deletes the last item from the list if the index is not given.

Syntax

The syntax of **pop()** method is as follows:

```
list_name.pop(ind)
```

list_name It indicates the name of the list.
ind It indicates the index of the value to be removed. It is optional.

Example

```
even = [2, 4, 6, 7, 8, 10]
even.pop(3)
print(even)
```

Output: [2, 4, 6, 8, 10]

3. Searching

The searching operation is used to find an item in a list. The **in** operator can be used to check whether a specific value exists in the list. It returns **True** if the value exists and returns **False** if the value does not exist. It is case-sensitive and only checks for exact matches.

Syntax

The syntax of **in** operator is as follows:

```
value in list_name
```

value It is the item to be searched in the list.
list_name It indicates the name of the list.

Example

```
fruits = ["apple", "banana", "cherry"]
if("mango" in fruits):
  print("Mango is available")
else:
  print("Sorry, mango is not available")
```

Output: Sorry, mango is not available

PROGRAM 4.1 **Write a program that inputs a number from the user and searches the number in a list.**

```
mylist = [10, 20, 30, 40, 50]
n = int(input("Enter a number of search: "))
if(n not in mylist):
  print(n, "is not in the list.")
else:
  s = mylist.index(n)
  print("The number ", n, "is at index", s)
```

Output:
Enter a number to search: 30
The number 30 is at index 2

Q. What is a stack? Discuss its basic operations.

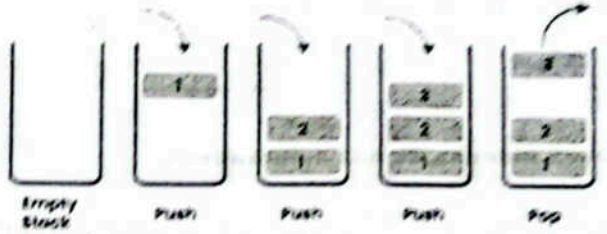
Stack

A **stack** is a data structure that follows the **Last In, First Out (LIFO)** principle. This means that the last element added to stack is the first one to be removed. It can be considered as a container where both insertion and deletion of elements occur at the top end.

Stack Operations

Two basic operations in a stack are as follows:

- **Push:** This operation is used to add an item to the top of the stack.
- **Pop:** This operation is used to remove an item from the top of the stack.



Q. Explain the operations on stack with real life example and python code.

Stack in Python

A **stack** can be created in Python using a list. The basic stack operations can be implemented using **append()** and **pop()** method.

The Push Operation

The push operation of stack can be implemented in Python using **append()** method. It adds an item at the top of the stack.

Syntax

The syntax of this method is as follows:

```
list_name.append(val)
```

list_name It indicates the name of the list.
val It indicates the value that will be appended to the list.

The Pop Operation

The pop operation of stack can be implemented in Python using **pop()** method. It deletes the item from the top of the stack.

Syntax

The syntax of this method is as follows:

```
list_name.pop()
```

list_name It indicates the name of the list.

Real-life Example

A simple real-world example of a stack is a pile of books. The last book placed on the top of the pile is the first one to be removed. The books at the bottom cannot be taken without first removing the books on the top. This approach is called **Last In First Out (LIFO)**.

Python Code for Book Stack

```
books = [ ]
print("Initial stack of books:", books)
print("Adding books to stack...")
books.append("Computer")
```

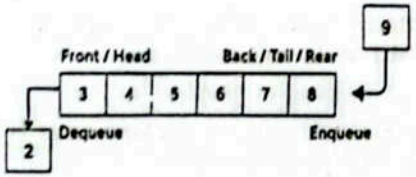
```
print("Stack after adding one book: ", books)
books.append("Python")
print("Stack after adding two books: ", books)
print("Deleting top book... ")
top = books.pop()
print("Removed book:", top)
print("Stack after removing book:", books)
```

Output:
```
Initial stack of books: []
Adding books to stack...
Stack after adding one book: ['Computer']
Stack after adding two books:
['Computer', 'Python']
Deleting the top book...
Removed book: Python
Stack after removing book: ['Computer']
```

Q. What is a queue? Discuss its basic operations.

Queue

A **queue** is a data structure that follows the **First In First Out (FIFO)** principle. This means that it stores new item at the end of all previously entered items. Similarly, the item that is stored at first position is removed first. It can be considered as a line in front of a bank or a ticket counter. The first person in the line is served first.



Queue Operations

Two basic operations in a queue are as follows:

- **Enqueue:** This operation is used to add an item to the end of the queue.
- **Dequeue:** This operation is used to remove an item from the front of the queue.

Some other operations include the following:

- Checking if the queue is empty or full.
- Retrieving the element at the front of the queue without removing it.
- Determining the size of the queue.

Q. How can a queue be created in Python? Give an example.

Queue in Python

A **queue** can be created in Python using built-in **Queue** class of **queue** module. It provides various methods to create a queue and perform different operations on it. The **Queue** class must be imported in Python program in order to create a queue.

The following line of imports **Queue** class from Python's built-in **queue** module:

```
from queue import Queue
```

Methods of queue Module

Some important methods in queue module are as follows:

1. The Queue() Method

The **Queue()** method creates an object of **Queue()** class. The syntax of this function is as follows:


```
obj Queue();
```

obj It indicates the name of the object to be created.

2. The put() Method

The **put()** method adds an item to the queue. The syntax of this method is as follows:

```
obj.put(item);
```

obj It indicates the name of the Queue object.
item It indicates the item to be added to the queue.

3. The get() Method

The **get()** method retrieves an item from the queue. The syntax of this method is as follows:

```
obj.get(item);
```

obj It indicates the name of the Queue object.
item It indicates the item to be added to the queue.

Example

```
from queue import Queue
q = Queue()
q.put("Ali")
q.put("Usman")
q.put("Furqan")
front = q.queue[0]
print(front, "is at the front of the queue.")
removed = q.get()
print(removed, "is removed from the queue.")
q.put("Sara")
updated = list(q.queue)
print(updated)
```

Output:
```
Ali is at the front of the queue.
Ali is removed from the queue.
['Usman', 'Furqan', 'Sara']
```

Explanation

```
from queue import Queue
```
This line imports **Queue** class.

```
q = Queue()
```
This line creates a new empty queue object **q**.

```
q.put("Ali")
q.put("Usman")
q.put("Furqan")
```
These lines add three people to the queue.

```
front = q.queue[0]
```
This line accesses the first item in the queue ("Ali") without removing it.

```
print(front, "is at the front of the queue.")
```
This line displays the first item in the queue ("Ali").

```
removed = q.get()
```
This line removes the first item in the queue ("Ali") and stores it in variable **removed**.

```
print(removed, "is removed from the queue.")
```
This line displays the value of **removed** variable.

```
q.put("Sara")
```

This line adds "Sara" to the end of the queue.

updated = list(q.queue)

This line stores the whole queue in **updated** variable.

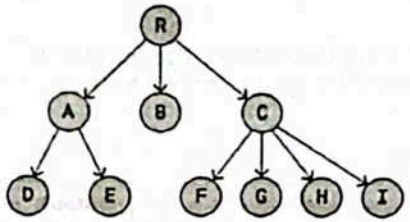
print(updated)

This line adds displays **updated** variable.

Q. What is a tree? Explain with an example.

Tree

A **tree** is a data structure that organizes data in a hierarchical way. It consists of nodes and edges where each node stores data and is connected to other nodes by edges. It forms a branching structure like a tree.



Example

A tree is similar to a family tree where the oldest ancestor represents root node. It is the starting point of the hierarchy. Each individual in a tree may have descendants that form subsequent levels of the hierarchy. It is a non-linear structure that represents parent-child relationships.

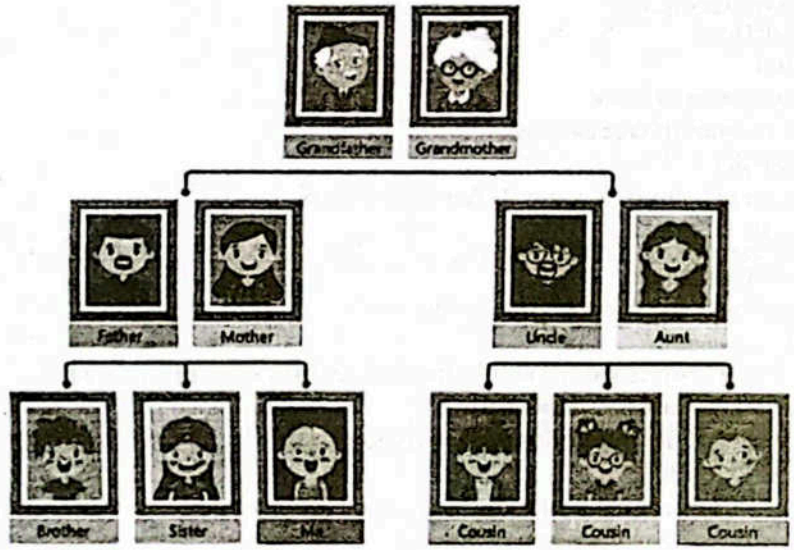


Figure: Family tree

Q. Describe the properties and applications of trees.

Properties of Trees

Some important properties of trees are as follows:

1. Root Node

The **root** is the top most node in a tree. It acts as the entry point to the entire structure. It is similar to the main folder in a computer where all other folders and files are contained.

2. Edges and Nodes

Nodes are the individual elements in the tree. The nodes are connected by lines called **edges.** A node without any child nodes is called a **leaf** similar to a file in a folder that does not contain any other files.

3. Height

The height of a tree is the longest path from the root node down to the deepest leaf node. It indicates the height or depth of the tree.

4. Balanced Trees

A tree is considered balanced if the branches on the left and right sides are nearly the same height.

Applications of Trees

Some important applications of trees are as follows:

1. File Systems

Pre-order tree traversal is useful for creating backups of file systems. It is a method of traversing a tree where each node is visited before its children. It ensures that directories are backed up before their contents.

2. File System Deletion

Post-order traversal is a tree traversal method where the child nodes are visited before the parent node. It is used in file systems to ensure that files and directories are deleted in the correct order. It first deletes all sub directories and files before deleting the parent directory.

3. Hierarchical Data Representation

Trees are used in representing data with a clear hierarchical relationship such as organizational charts and family trees.

4. Decision Making

Trees such as decision trees are used in algorithms to make decisions based on various conditions and outcomes.

Q. What is a graph? Explain with an examples.

Graph

A **graph** is a data structure that consists of a set of **vertices** or **nodes** connected by **edges.** Graphs are used to represent networks of connections where each connection is a relationship between two vertices. These vertices can represent real-world entities such as cities, people or concepts. The edges represent the relationships, connections or pathways between them.

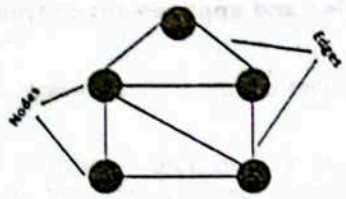


Figure: Graph

A graph does not have a single root node and it does not follow a hierarchical structure. It can have multiple connections between nodes.

Example 1

Suppose the user is mapping out all cities in Pakistan and the roads that connect them. Each city is a vertex and each road between two cities is an edge. This forms a graph because cities can be connected in multiple ways and there is no single starting or ending point.

Example 2

In a social network, each user is a node and a friendship or connection between users is an edge. A user can be connected to many other users that forms a graph. There is no single starting or ending point.

Q. Describe different characteristics and properties of graphs.

Characteristics of Graphs

Some important characteristics of graphs are as follows:

1. Vertices (Nodes)

These are the individual points or entities in a graph. For example, each user in a social network graph is represented by a vertex.

2. Edges (Links)

These are the connections between vertices. For example, each road in a transport system graph is an edge that connects two cities.

Properties of Graphs

Some important properties of graphs are as follows:

1. Degree

This is the number of edges connected to a vertex. For example, the degree of vertex will be 3 if a city is connected to three other cities.

2. Weight

In some graphs, edges have weights that represent the values like distances or costs. For example, if a road between two cities is 50 kilometers long, its edge may have a weight of 50.

3. Direction

Edges can be either directed or undirected. Directed edges have a one-way connection such as a one-way street. The undirected edges represent a two-way connection.

Q. Describe different types of graphs.

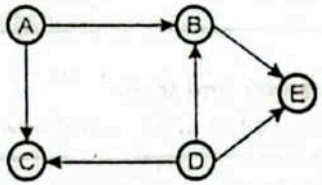
Graphs can be classified into many types based on their structure and properties. The main types of graphs are directed, undirected and weighted.

1. Directed Graphs

A **directed graph** is a type of graph where each edge has a direction. It means that it goes from one vertex to another vertex in a specific way.

Example

The following graph is a directed graph. The arrows show the direction of link between two vertices.

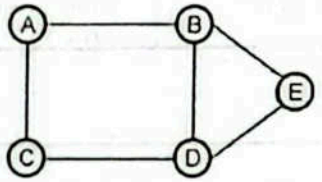


2. Undirected Graphs

An **undirected graph** is a type of graph in which the edges have no direction. This means that if there is a connection between two vertices, you can travel in both directions.

Example

The following graph is an undirected graph. Suppose the user needs to go from A to D. There is no restriction on the direction.

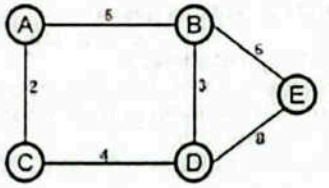


3. Weighted Graphs

A **weighted graph** is a type of graph in which each edge has an associated numerical value called a **weight** or **cost**. It represents the distance, time or cost required to travel from one vertex to another.

Example

The following example shows a graph for the map of a city. Each road on the map has a different distance or travel time. This information helps the user to determine the shortest or quickest route between two points.



Q. Differentiate between directed and undirected graph.

The difference between directed and undirected graph is as follows:

Directed Graph	Undirected Graph
1. The edges in directed graph have directions from one vertex to another.	1. The edges in undirected graph have no directions.
2. It allows one-way movement from one node to another.	2. It allows two-way movement from one node to another.
3. It is less flexible than undirected graph.	3. It is more flexible than directed graph.
4. The vertices are connected using arrows in directed graph.	4. The vertices are connected using lines in undirected graph.

Q. Differentiate between graph and tree.

The difference between graph and tree is as follows:

Tree	Graph
1. A tree is a hierarchical structure.	1. A graph is a network-like structure.
2. It has a single root node.	2. It does not have any root node.
3. It has exactly one path between any two nodes.	3. It can have multiple paths between nodes.
4. It is usually directed from parent to child node.	4. It can be directed or undirected.
5. It cannot have cycles or loops.	5. It can have cycles or loops.
6. It is used to represent file systems, organizational charts and family trees.	6. It is used to represent social networks, maps, web links and transport systems.

Exercise Solution

Multiple-Choice Questions (MCQs)

1. **The function used to add an item at the end of a list in Python:**
   a) insert( ) b) append( ) c) remove( ) d) pop( )
2. **The purpose of the "in" keyword used with a Python list:**
   a) Adds an item to the list b) Removes an item from the list
   c) Checks if an item exists in the list d) Returns the length of the list
3. **An operation that removes an item from the top of the stack:**
   a) Push b) Pop c) Peek d) Add
4. **The operation used to add an item to a queue:**
   a) Dequeue b) Peek c) Enqueue d) Remove
5. **True statement about the height of a tree:**
   a) Number of edges from the root to the deepest node
   b) Number of nodes from the root to the deepest node
   c) Number of children of the root node
   d) Always equal to the number of nodes in the tree

6. **A scenario where a graph data structure is most suitable:**
   a) Managing a to-do list
   b) Modeling a line of customers in a store
   c) Representing connections in a social network
   d) All of the above

Answers

1. b	2. c	3. b	4. c	5. a	6. c

Short Questions

Q.1. Explain how the insert function works in python list. Provide an example.

The insert() function in Python is used to add an item at a specific index in a list. The syntax of this function is list_name.insert(index, element). An example of insert() function is as follows:

```
country = ["Turkey", "Iran", "Oman"]
country.insert(0, "Pakistan")
print(country)
```

Q.2. Explain the potential issues which could arise when two variables reference the same list in a program? Provide an example.

When two variables reference the same list, they both point to the same memory location. This means that changes made through one variable will affect the other. It will often occur unintentionally.

```
a = [1, 2, 3]
b = a
b.append(4)
print("a", a)  # Output: a: [1, 2, 3, 4]
print("b", b)  # Output: b: [1, 2, 3, 4]
```

Q.3. Define a stack and explain the Last In, First Out (LIFO) principle.

Stack is a data structure that follows the **Last In, First Out (LIFO)** principle. This means the last element added to stack is the first one to be removed.

Q.4. Differentiate between the Enqueue and Dequeue operations of queue.

Enqueue and Dequeue are the two main operations of a queue. Enqueue adds an element to the rear of the queue while Dequeue removes an element from the front. Enqueue increases the queue size whereas Dequeue decreases it.

Q.5. Name two basic operations performed on stack.

Two basic operations performed on stack are push and pop. Push operation is used to add an item to the top of the stack. Pop operation is used to remove an item from the top of the stack.

Long Questions

Q.1. Discuss the dynamic size property of lists in Python. How does this property make lists more flexible?

The lists in Python have a dynamic size. It means that they can grow or shrink automatically as elements are added or removed. The user does not need to declare a fixed size when creating a list. The dynamic size property makes the lists highly flexible for the following reasons:

1. **No Predefined Size:** The lists can be defined without any predefined size. It gives flexibility to add any number of elements to the list
2. **Automatic Memory Management:** Python internally resizes the list's storage as needed. It handles memory allocation for the user.
3. **Ease of Use:** Lists support operations like append(), insert(), pop() and remove(). These operations allow dynamic manipulation of content at runtime.

4. **Suitable for Varying Data:** It is ideal for situations where the number of items is not known in advance such as reading user input.

Q.2. Explain the operations on stack with real life example and python code. (See chapter)

Q.3. Write, a simple program to implement a queue (insertion and deletion). (See chapter)

Q.4. Define Tree and explain its properties. (See chapter)

Q.5. What is graph? Explain differences between directed and undirected graph. (See chapter)

SHORT QUESTIONS

Q.1. What are the properties of a list in Python?

The properties of a list in Pythonn include dynamic size, index-based access and ordered collection. Dynamic size means that a list can change its size. Index-based access means that every item in a list has a unique index that indicates its position in the list. Ordered collection means that the order in which items are added to a list is preserved.

Q.2. What does dynamic size mean in the context of lists?

It means that a list in Python can change its size. The new items can be added to the list and existing items can be removed easily. The size of the list automatically changes.

Q.3. List some common operations that you can perform on lists in Python.

Some common operations that you can perform on lists in Python include insertion, deletion, searching and sorting.

Q.4. How do you remove an item from a list in Python?

You can remove an item from a list in Python using remove() or pop() function. The remove() function deletes an item from the list by value. It removes the first occurrence of the specified value from the list. The pop() function is used to delete an item from the list by index. The function deletes the last item from the list if the index is not given.

Q.5. How can you remove an item from a list by value?

You can remove an item from a list by value using remove() function. It removes the first occurrence of the specified value from the list. For example, fruit.remove("potato") removes "potato" from fruit list.

Q.6. How can you remove an item from a list by index?

You can remove an item from a list by index using pop() function. It deletes the value at the specified index of the list. The function deletes the last item from the list if the index is not given. For example, even.pop(3) removes the item at index 3 from even list.

Q.7. How do you search for an item in a list in Python?

The in operator is used to search a value in the list. It returns True if the value exists and returns False if the value does not exist.

Q.8. What is a stack in data structures?

A stack is a data structure that follows the Last In, First Out (LIFO) principle. This means that the last element added to stack is the first one to be removed. It can be considered as a container where both insertion and deletion of elements occur at the top end.

Q.9. Name two basic operations in a stack.

Two basic operations in a stack are push and pop. The push operation is used to add an item to the top of the stack. The pop operation is used to remove the top item.

Q.10. How do you implement a stack using a list in Python?

You can implement a stack using a list in Python by using the append() method to add items to the stack and the pop() method to remove items from the stack.

Q.11. Can you access items in the middle of a stack?

No, you cannot access items in the middle of a stack directly. You can only access the top item of the stack.

Q.12. What is a queue in data structures?

A queue is a data structure that follows the First In First Out (FIFO) principle. This means that it stores new item at the end of all previously entered items. Similarly, the item that is stored at first position is removed first. It can be considered as a line in front of a bank or a ticket counter, the first person in the line is served first.

Q.13. Write two basic operations in a queue.

Two basic operations in a queue are enqueue and dequeue. The enqueue operation is used to add an item to the end of the queue. The dequeue operation is used to remove an item from the front of the queue.

Q.14. How do you implement a queue using the queue module in Python?

You can implement a queue using queue module in Python by creating a Queue object and using its put() and get() methods to add and remove items.

Q.15. How do you add items to a queue in Python?

You add items to a queue in Python using put() method. For example, q.put("Ahmed") will add "Ahmed" to the queue named q.

Q.16. How do you remove items from a queue in Python?

You remove an item from a queue in Python using get() method. For example, q.get() removes the first added item from the queue named q.

Q.17. What is the difference between a stack and a queue?

The stack data structure follows Last In, First Out (LIFO) principle. This means that the last element added to stack is the first one to be removed. The queue follows the First In First Out (FIFO) principle. This means that the first element added to queue is the first one to be removed.

Q.18. What is a tree in data structure?

A tree is a data structure that organizes data in a hierarchical way. It consists of nodes and edges where each node stores data and is connected to other nodes by edges. It forms a branching structure like a tree.

Q.19. What is the root node in a tree?

The root is the top most node in a tree. It acts as the entry point to the entire structure. It is similar to the main folder in a computer where all other folders and files are contained.

Q.20. What are nodes and edges in a tree?

Nodes are the individual elements in the tree. The nodes are connected by lines called edges. A node without any child nodes is called a leaf similar to a file in a folder that does not contain any other files.

Q.21. How do you determine the height of a tree?

The height of a tree is determined by finding the longest path from the root node down to the deepest leaf node.

Q.22. What is a balanced tree?

A tree is a balanced if the branches on the left and right sides are nearly the same height.

Q.23. What is pre-order tree traversal?

Pre-order tree traversal is useful for creating backups of file systems. It is a method of traversing a tree where each node is visited before its children. It ensures that directories are backed up before their contents.

Q.24. What is post-order tree traversal?

Post-order traversal is a tree traversal method where the child nodes are visited before the parent node. It is used in file systems to ensure that files and directories are deleted in the correct order. It first deletes all sub directories and files before deleting the parent directory.

Q.25. What is a graph?

A graph is a data structure that consists of a set of vertices or nodes connected by edges. Graphs are used to represent networks of connections where each connection is a relationship between two vertices.

Q.26. Give a real-world example of a graph.

Suppose the user is mapping out all cities in Pakistan and the roads that connect them. Each city is a vertex and each road between two cities is an edge. This forms a graph because cities can be connected in multiple ways and there is no single starting point.

Q.27. Can graphs have multiple paths between two vertices?

Yes, graphs allow multiple paths between two vertices. It is useful for modeling transport routes or communication channels.

Q.28. What is meant by the degree of a vertex?

The degree of a vertex is the number of edges connected to it. For example, the degree of vertex will be 3 if a city is connected to three other cities.

Q.29. What is the difference between directed and undirected edges?

Directed edges have a one-way connection that means the relationship or pathway is only in one direction. Undirected edges represent a two-way connection.

Q.30. How do graphs differ from trees?

Tree is a hierarchical structure whereas graph is a network-like structure. Tree has a single root node but graph does not have any root node. Tree has exactly one path between any two nodes but graph can have multiple paths between nodes.

Multiple Choice Questions

1. **A data structure used to store multiple items in a sequence is called:**
   a. Dictionary b. Loop c. List d. Function
2. **Each piece of data in a list is known as a:**
   a. Variable b. Element c. Operator d. Function
3. **Lists are created using:**
   a. Curly braces {} b. Parentheses ()
   c. Angle brackets <> d. Square brackets []
4. **How are elements separated in a Python list?**
   a. Semicolons b. Commas c. Periods d. Colons
5. **Every item in a list has a position, called an _____.**
   a. Index b. Value c. Label d. Node
6. **What is the index of the first element in a list?**
   a. 0 b. 1 c. -1 d. First
7. **Which of the following are properties of a list in Python?**
   a. Dynamic Size b. Index-Based Access
   c. Ordered Collection d. All
8. **Which property allows Python lists to increase or decrease in size during execution?**
   a. Static Size b. Indexed Storage
   c. Dynamic Size d. Constant Size
9. **In Python, what type of collection is a list?**
   a. Random b Unordered c. Ordered d. None
10. **Which method is used to insert an element at a specific position in a Python list?**
    a. add() b. insert() c. append() d. put()
11. **Which of the following can be used to insert 5 into the third position (index 2) in list1?**
    a. list1.insert(3, 5) b. list1.insert(2, 5)
    c. list1.add(3, 5) d. list1.append(3, 5)
12. **Which method removes an item from the list by value?**
    a. pop() b. delete() c. remove() d. discard()
13. **Which of the following removes "Buy snacks" from the party_list?**
    a. party_list.delete("Buy snacks") b. party_list.remove("Buy snacks")
    c. party_list.pop("Buy snacks") d. party_list.clear("Buy snacks")

14. **Which method is used to remove an item by its index?**
    a. remove() b. pop() c. delete() d. clear()
15. **Which of the following removes the first occurrence of an item from a list?**
    a. append() b. pop() c. remove() d. clear()
16. **Which operator is used to check if an item exists in a Python list?**
    a. find b. in c. has d. exist
17. **Which types of elements can a Python list contain?**
    a. Only integers b. Only strings
    c. Only characters d. Any type of data
18. **Which data structure can be implemented using a Python list to follow Last In First Out (LIFO) order?**
    a. Queue b. Dictionary c. Stack d. Tuple
19. **Which data structure can be implemented using a list in First In First Out (FIFO) order?**
    a. Set b. Queue c. Stack d. Matrix
20. **In a stack data structure, from which end are elements added and removed?**
    a. Bottom b. Middle c. Top d. Random
21. **Which operation adds an item to the top of the stack?**
    a. insert() b. pop() c. push d. delete()
22. **Which operation removes an item from the top of the stack?**
    a. append() b. remove() c. push d. pop
23. **Which Python list method is used to implement the "push" operation in a stack?**
    a. insert() b. append() c. pop() d. extend()
24. **Which Python list method is used to implement the "pop" operation in a stack?**
    a. delete() b. insert() c. pop() d. remove()
25. **What does the following code do? stack_of_books = []**
    a. Creates a queue b Creates an array
    c. Creates an empty stack d. Deletes a list
26. **A line of customers at a bank is an example of which data structure?**
    a. Stack b. Queue c. Tree d. Array
27. **In a queue data structure, new elements are always added to the:**
    a. Front b. Middle c. Rear d. Any
28. **Which operation adds an item to the end (rear) of a queue?**
    a. dequeue b. push c. enqueuer d. pop
29. **The operation to remove an element from the front of a queue is called:**
    a. pop b. Dequeue c. remove d. delete
30. **Which Python module is commonly used to implement a queue?**
    a. stack b. collections c. queue d. list
31. **Which method is used to add (enqueue) an item to a queue in Python?**
    a. add() b. push() c. put() d. append()
32. **Which method is used to remove an item from a queue in Python?**
    a. remove() b. get() c. delete() d. pop()
33. **A tree is a _____ data structure.**
    a. Linear b. Hierarchical c. Sequential d. Random
34. **The topmost node in a tree is called:**
    a. Leaf node b. Parent node
    c. Root node d. Branch node
35. **The individual elements in a tree are called:**
    a. Leaves b. Edges c. Nodes d. Roots
36. **The lines that directly connect nodes in a tree are called:**
    a. Path b. Edge c. Link d. Branch
37. **In tree data structures, nodes without any children are called:**
    a. Root nodes b. Internal nodes
    c. Leaf nodes d. Edge nodes
38. **The number of edges from root node to the deepest leaf is called the _____ of tree.**
    a. Height b. Depth c. Distance d. Width

39. **Which tree traversal is most suitable for creating file system backups?**
a. In-order b. Pre-order
c. Post-order d. Level-order

40. **The correct traversal for safely deleting a directory and its contents is:**
a. In-order b. Pre-order
c. Post-order d. Breadth-first

41. **Which data structure uses tree-based branching to evaluate conditions and make decisions?**
a. Stack b. Queue
c. Decision trees d. Circular list

42. **A _____ is a data structure that consists of a set of vertices (or nodes) connected by edges.**
a. Graph b. Tree c. Array d. Stack

43. **Which of the following is a data structure that consists of a set of vertices (or nodes) connected by edges?**
a. Tree b. Array c. Graph d. Stack

44. **Another name for a vertex in a graph is called:**
a. Leaf b. Node c. Root d. Point

45. **What are the individual points or entities in a graph called?**
a. Edge b. Path c. Vertices d. Degrees

46. **A connection or relationship between two vertices in a graph is called a:**
a. Node b. Edge c. Root d. Path

47. **Which structure is best for modeling a network of friends connected with each others?**
a. Stack b. Tree c. Graph d. Queue

48. **Which of the following is NOT typically represented using a general graph structure?**
a. Social networks b. Web page links.
c. Family trees d. transport systems

49. **In a graph representing a social network, what do vertices typically represent?**
a. Friendships b. People c. Messages d. Groups

50. **In a transport system graph, what represents the roads between cities?**
a. Vertices b. Lists c. Edges d. Trees

51. **Which of the following data structures allows multiple paths between nodes and permits cycles?**
a. Tree b. Graph c. Stack d. Queue

52. **In a tree data structure, how many distinct paths exist between any two nodes?**
a. Multiple paths b. No path
c. Exactly one d. Random paths

53. **Which data structure is most suitable for representing hierarchical relationships like family trees or organizational charts?**
a. Stack b. Queue c. Tree d. Graph

54. **The number of edges connected to a vertex in a graph is called its:**
a. Weight b. Degree c. Path d. Cycle

55. **If a city is connected to three other cities in a graph, its vertex degree is:**
a. 1 b. 2 c. 3 d. 4

56. **Which of the following is NOT a type of graph?**
a. Undirected b. Weighted c. Directed d. Linear

57. **What type of edge has a one-way connection in a graph?**
a. Undirected edge b. Weighted edge
c. Directed edge d. Circular edge

58. **If an edge allows travel only from node A to node B, it is called:**
a. Undirected b. Weighted
c. Reversible d. Directed

59. **What type of edge has a two-way connection in a graph?**
a. Undirected b. Weighted
c. Directed d. Circular

60. **Which type of graph has edges that do not indicate a specific direction?**
a. Directed graph b. Weighted graph
c. Undirected graph d. Cyclic graph

61. **A road map where edges show travel time is an example of _____ graph.**
a. Unweighted b. Weighted c. Complete d. Bipartite

Answers

1. c	2. b	3. d	4. b	5. a	6. a
7. d	8. c	9. c	10. b	11. b	12. c
13. b	14. b	15. b	16. b	17. d	18. c
19. b	20. c	21. c	22. d	23. b	24. c
25. c	26. b	27. c	28. c	29. b	30. c
31. c	32. b	33. b	34. c	35. c	36. b
37. c	38. a	39. b	40. c	41. c	42. a
43. c	44. b	45. c	46. b	47. c	48. c
49. b	50. c	51. b	52. c	53. c	54. b
55. c	56. d	57. c	58. d	59. a	60. c
61. b					

CHAPTER 5

Data Analytics

Q. What is data analytics and statistics?

Data Analytics

Data analytics is the process of examining and analyzing data to find useful information, patterns or trends that can help in decision-making.

Statistics

Statistics is a branch of mathematics that is used to understand, analyze and interpret data. Statistics is important because it can summarize large sets of data in a simple way that makes it easier to draw useful conclusions and make decisions. It is widely used in various fields such as education, business, healthcare and economics etc.

Q. What are measures of central tendency? Briefly describe the three main measures of central tendency with examples.

Measures of Central Tendency

Measures of central tendency are statistical tools used to find the central or typical value in a dataset. The three main measures are **mean, median** and **mode.**

1. Mean (Average)

Mean is the average of all the values in a dataset. It is calculated by adding all the data values and then dividing by the number of values.

Example

Suppose the scores of five students in a test are 41, 12, 30, 25 and 9. The mean score is calculated by adding all the scores and then dividing by 5.

Data set: 41 12 30 25 9
Mean: (41 + 12 + 30 + 25 + 9) / 5 = 23.4

2. Mode

Mode is the value that appears most frequently in a dataset. It helps to identify the most common or repeated value in a dataset. There can be more than one mode if multiple values appear with the same highest frequency.

Example 1: Single Mode

Suppose the scores of five students are 50, 60, 70, 70 and 90. The number **70** appears twice whereas all other numbers appear only once. Therefore, the mode is **70** as it is the most frequent value.

Example 2: Multiple Modes

Suppose the scores of six students are 50, 60, 70, 70, 60 and 90. Both **60** and **70** appear twice which is the highest frequency in this dataset. Therefore, this dataset has two modes 60 and 70.

3. Median

Median is the middle value when the numbers are arranged in order. The median is the exact middle value if there is an odd number of values. The median is the average of the two middle values if there is an even number of values.

Example 1

Suppose the scores of five students are 50, 60, 70, 80 and 90. These scores are already in ascending order. The number of scores is odd with the middle value 70. Therefore, the median score is 70.

Example 2

Suppose the scores of four students are 50, 60, 70 and 80. These scores are already in ascending order. The number of scores is even. Therefore, the median score is the average of two middle values of 60 and 70 as (60 + 70) / 2 = 65.

Q. What are measures of dispersion? List two commonly used measures of dispersion.

Measures of Dispersion

Measures of dispersion describe how spread out or scattered the data values are in a dataset. They help to understand the degree of variation or inconsistency among the data points. These measures are important for identifying how much individual values differ from the mean (average).

Two commonly used measures of dispersion are as follows:

- Variance
- Standard deviation

Q. What is variance. Explain how variance is calculated and interpreted

Variance

Variance is a statistical measure that shows how much the values in a dataset differ from the mean (average). A higher variance means the values are more spread out from the mean. A lower variance means the values are closer to the mean. Variance is calculated using the following formula:

$$Variance(\sigma^2) = \frac{\sum_{i=1}^{N}(X_i - \mu)^2}{N}$$

Where

- σ^2 is the symbol for variance
- X_i indicates each individual value in the dataset
- μ is the mean (average) of the dataset
- $\sum$ is the sum of all squared differences
- N indicates the total number of values in the dataset