1. **What is a data structure? Explain its importance.**

A data structure is a way of organizing and storing data in a computer so that it can be accessed and modified efficiently. The importance of data structures lies in their ability to manage large amounts of data efficiently for uses such as large databases and internet indexing services. They allow for efficient data search and retrieval, which is essential for developing efficient software applications.

2. **Differentiate between linear and non-linear data structures.**

- **Linear Data Structures:** Elements are arranged in a sequential manner, such as arrays, linked lists, stacks, and queues. Each element has a single successor and predecessor, except the first and last elements.

- **Non-linear Data Structures:** Elements are not in sequence, such as trees and graphs. Each element can have multiple successors or predecessors, allowing hierarchical or interconnected relationships.

3. **What are the primary operations that can be performed on a stack?**

- **Push:** Add an element to the top of the stack.
- **Pop:** Remove the top element of the stack.
- **Peek/Top:** View the top element without removing it.
- **isEmpty:** Check if the stack is empty.

4. **Describe the difference between a stack and a queue.**

- **Stack:** Follows Last-In-First-Out (LIFO) principle. Elements are added and removed from the same end.

- **Queue:** Follows First-In-First-Out (FIFO) principle. Elements are added from the rear and removed from the front.

5. **How does a circular queue differ from a regular queue?**

- **Regular Queue:** Has a fixed size and becomes full when the rear reaches the last position.

- **Circular Queue:** The last position is connected to the first position, making it a circular buffer, allowing efficient utilization of space by reusing vacated spaces.

6. **Explain the concept of a linked list. How is it different from an array?**

A linked list is a collection of nodes where each node contains data and a reference (or link) to the next node in the sequence. Unlike arrays, linked lists do not have a fixed size and elements can be easily inserted or removed without shifting elements, offering dynamic memory allocation.

7. **Difference between array and linked list**

- **Array:** Fixed size, continuous memory allocation, easy access via indices, insertion/deletion is expensive.

- **Linked List:** Dynamic size, non-continuous memory allocation, access requires traversal, insertion/deletion is efficient.

8. **What are the main operations of a queue? Describe their time complexities.**

- **Enqueue:** Add an element to the rear of the queue (O(1)).
- **Dequeue:** Remove an element from the front of the queue (O(1)).
- **Peek/Front:** Access the front element without removing it (O(1)).
- **isEmpty:** Check if the queue is empty (O(1)).

9. **What are the main operations of a stack?**

The primary operations of a stack are push, pop, peek/top, and isEmpty, as explained in question 3.

10. **Describe the algorithm for evaluating postfix expressions using a stack.**

1. Initialize an empty stack.
2. Scan the postfix expression from left to right.
3. For each element:
   - If it is an operand, push it onto the stack.
   - If it is an operator, pop the required number of operands from the stack, apply the operator, and push the result back onto the stack.
4. The final result is on the top of the stack.

11. **Explain the concept of a singly linked list and provide an example.**

A singly linked list is a collection of nodes where each node contains data and a pointer to the next node in the sequence. Example:

```
Head -> [Data|Next] -> [Data|Next] -> [Data|Next] -> NULL
```

```

12. **How does a doubly linked list differ from a singly linked list?**

A doubly linked list has nodes with two pointers: one pointing to the next node and another pointing to the previous node, allowing traversal in both directions.

13. **What is a circular linked list, and how is it implemented?**

A circular linked list is a linked list where the last node points back to the first node, creating a loop. It can be implemented by setting the `next` pointer of the last node to the head node.

14. **What is linear search, and what is its time complexity?**

Linear search is a method for finding an element in a list by checking each element sequentially until the desired element is found or the list ends. Its time complexity is O(n), where n is the number of elements in the list.

15. **Explain binary search. How does it improve search efficiency compared to linear search?**

Binary search is a method for finding an element in a sorted list by repeatedly dividing the search interval in half. It improves search efficiency by reducing the search space logarithmically, with a time complexity of O(log n), compared to linear search's O(n).

16. **How does selection sort work?**
```

Selection sort works by repeatedly finding the minimum element from the unsorted portion of the list and swapping it with the first unsorted element. This process is repeated until the entire list is sorted.

17. **Classify data structures with diagram.**

Data structures can be classified into two main types:

- **Linear Data Structures:** Arrays, Linked Lists, Stacks, Queues.
- **Non-linear Data Structures:** Trees, Graphs.

18. **Interpret Big O complexity chart.**

The Big O complexity chart is used to describe the performance or complexity of an algorithm in terms of time or space. Common complexities include:

- O(1): Constant time
- O(log n): Logarithmic time
- O(n): Linear time
- O(n log n): Linearithmic time
- O(n^2): Quadratic time
- O(2^n): Exponential time

19. **Discuss Time complexity**

Time complexity is a measure of the amount of time an algorithm takes to complete as a function of the length of the input. It provides an upper bound on the growth of the runtime as the input size increases, helping to predict performance and compare algorithms.

20. **Describe sparse matrix. Find the address of A [2][1] if base address is 1024 for an integer array A[5][4] in row-major order and word size is 2 bytes.**

A sparse matrix is a matrix in which most of the elements are zero. To find the address of `A[2][1]`:

Address = Base Address + ((Row Index * Number of Columns) + Column Index) * Element Size

Address = 1024 + ((2 * 4) + 1) * 2 = 1042

21. **Given a two dimensional array A1(1:8, 7:14) stored in row-major order with base address 100 and size of each element is 4 bytes, find address of the element A1(4, 12).**

Address = Base Address + ((Row Index - Start Row) * Number of Columns + (Column Index - Start Column)) * Element Size

Address = 100 + ((4 - 1) * 8 + (12 - 7)) * 4 = 100 + (24 + 5) * 4 = 216

22. **Define dynamic memory allocation?**

Dynamic memory allocation is the process of allocating memory during the runtime of a program, as opposed to static memory allocation, which occurs at

compile time. It allows programs to use memory more efficiently by allocating and freeing memory as needed.

23. **Define referential structure?**

A referential structure is a data structure that consists of references or pointers to other data structures. Examples include linked lists, trees, and graphs.

24. **Array is a heterogeneous data type. (True/False). Justify your answer.**

False. An array is a homogeneous data type, meaning it stores elements of the same type.

25. **A m*n matrix which contains very few non-zero elements. A matrix contains more number of ZERO values than NON-ZERO values. Such matrix is known as ?**

Such a matrix is known as a sparse matrix.

26. **Convert infix to Postfix and Prefix**

1. **(A + B) / C – D * E**
   - Postfix: `AB+C/DE*-`
   - Prefix: `- / + A B C * D E`

2. **P ^ Q ^ R + S / T**
   - Postfix: `PQR^^ST/+`

- Prefix: `+ ^^PQR / ST`

3. **A * B – (C / D + (E – F)) ^ G**
   - Postfix: `AB*CD/EF-+G^-`
   - Prefix: `-*AB^+ / CD - EF G`

27. **List applications of stack and Convert 2 * 3 / (2-1) + 5 * 3 infix expression into postfix format. Showing stacks status after every step in tabular form and evaluates that postfix notation.**

Applications of Stack:

- Expression evaluation and conversion (infix to postfix/prefix)
- Syntax parsing
- Backtracking (e.g., navigating back in a

browser)

- Function call management (call stack in programming languages)

Infix to Postfix Conversion:

Infix: `2 * 3 / (2 - 1) + 5 * 3`

Postfix: `23*21-/53*+`

Step	Stack	Output
1	*	2

| 2 | / | 23* |

| 3 | (| 23* |

| 4 | (| 23*2 |

| 5 | - | 23*21 |

| 6 | | 23*21- |

| 7 | + | 23*21-/|

| 8 | * | 23*21-/5|

| 9 | | 23*21-/53*+|

Postfix Evaluation:

Evaluate `23*21-/53*+`:

1. Push 2, 3: `2 3`
2. *: `6`
3. Push 2, 1: `6 2 1`
4. -: `6 1`
5. /: `6`
6. Push 5, 3: `6 5 3`
7. *: `6 15`
8. +: `21`

28. **Transform the following expression to postfix and evaluate postfix expression by assuming A=1,B=2,C=3,D=4,E=6,F=6,G=1,I=3 and J=3. INFIX- A + B – C * D / E + F * G / (I + J)**

Infix to Postfix Conversion:

Infix: `A + B – C * D / E + F * G / (I + J)`

Postfix: `AB+CD*E/-FG*IJ+/+`

Postfix Evaluation with Given Values:

`A=1,B=2,C=3,D=4,E=6,F=6,G=1,I=3,J=3`

- Postfix: `12+34*6/-61*33+/+`
- Evaluation: `3 12 24 / 2.4 6 1 3 3 3 1.8 3 + 3.6 + 2.4 = 3`

29. **Differentiate between LIFO and FIFO access mechanism.**

- **LIFO (Last In, First Out):** The last element added is the first to be removed. Used in stacks.
- **FIFO (First In, First Out):** The first element added is the first to be removed. Used in queues.

30. **How linked list is better compared to stack, queue, and array? Explain with concept of dynamic memory allocation.**

Linked lists provide dynamic memory allocation, allowing efficient memory usage by allocating memory as needed and freeing it when not required. This makes linked lists more flexible than arrays, which have fixed sizes. Linked lists can efficiently implement stacks and queues by dynamically adjusting their sizes.

31. **In which type of scenario, linear queue (simple queue) is better than circular queue?**

A linear queue is better when the total number of elements processed is less than the queue's capacity, avoiding the complexity of circular implementation. It is also simpler when there is no need for reusing spaces vacated by dequeued elements.

32. **After evaluation of 3,5,4,*,+ result is ?**

- Postfix Expression: `354*+`
- Evaluation: `5*4 = 20`, `3 + 20 = 23`

33. **What will be the value of Front and Rear pointers when Queue is empty?**

- **Front Pointer:** Points to the position before the first element (or -1 in some implementations).
- **Rear Pointer:** Points to the same position as the Front or -1.

34. **Apply selection sort algorithm on following input. 12, 29, 25, 8, 32, 17, 40. Explain step by step.**

Selection Sort Steps:

Initial Array: `12, 29, 25, 8, 32, 17, 40`

- Step 1: Find the minimum element `8`, swap with `12`

- Result: `8, 29, 25, 12, 32, 17, 40`

- Step 2: Find the minimum element `12`, swap with `29`
  - Result: `8, 12, 25, 29, 32, 17, 40`

- Step 3: Find the minimum element `17`, swap with `25`
  - Result: `8, 12, 17, 29, 32, 25, 40`

- Step 4: Find the minimum element `25`, swap with `29`
  - Result: `8, 12, 17, 25, 32, 29, 40`

- Step 5: Find the minimum element `29`, swap with `32`
  - Result: `8, 12, 17, 25, 29, 32, 40`

- Step 6: Already sorted

Sorted Array: `8, 12, 17, 25, 29, 32, 40`

35. **Write an algorithm for bubble sort. Apply it on random 8 input data.**

Bubble Sort Algorithm:

1. Start from the first element, compare the current element with the next element.
2. If the current element is greater than the next element, swap them.
3. Move to the next element and repeat step 1 until the end of the list.
4. Repeat steps 1-3 for the number of elements minus one times.

Example Application on Random Data:

Input: `64, 34, 25, 12, 22, 11, 90`

- Pass 1: `34 25 12 22 11 64 90`
- Pass 2: `25 12 22 11 34 64 90`
- Pass 3: `12 22 11 25 34 64 90`
- Pass 4: `12 11 22 25 34 64 90`
- Pass 5: `11 12 22 25 34 64 90`

36. **Write Merge Sort algorithm. Apply the algorithm to the following elements: 10,5,28,7,39,310,55,15,1**

Merge Sort Algorithm:

1. Divide the unsorted list into n sublists, each containing one element.
2. Repeatedly merge sublists to produce new sorted sublists until there is only one sublist remaining.

Example Application on Given Data:

Input: `10,5,28,7,39,310,55,15,1`

- Divide: `[10], [5], [28], [7], [39], [310], [55], [15], [1]`
- Merge Step 1: `[5,10], [7,28], [39,310], [15,55], [1]`
- Merge Step 2: `[5,7,10,28], [15,39,55,310], [1]`

- Merge Step 3: `[5,7,10,15,28,39,55,310], [1]`
- Final Merge: `[1,5,7,10,15,28,39,55,310]`

If you have any specific questions or need further elaboration on any of the topics, feel free to ask!