# TOP 50

# DSA

## INTERVIEW QUESTION

Created by- TOPPERWORLD

## Q 1. What are Data Structures?

**Ans:** A data structure is a mechanical or logical way that data is organized within a program. The organization of data is what determines how a program performs.

There are many types of data structures, each with its own uses. When designing code, we need to pay particular attention to the way data is structured.

If data isn't stored efficiently or correctly structured, then the overall performance of the code will be reduced.

## Q 2. Why Create Data Structures?

**Ans :** Data structures serve a number of important functions in a program.

They ensure that each line of code performs its function correctly and efficiently, they help the programmer identify and fix problems with his/her code, and they help to create a clear and organized code base.

## Q 3. What are some applications of Data structures?

**Ans:** Different applications of an array are as follows:

- An array is used in solving matrix problems.
- Database records are also implemented by an array.
- It helps in implementing a sorting algorithm.
- It is also used to implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.
- An array can be used for CPU scheduling.
- Can be applied as a lookup table in computers.

- Arrays can be used in speech processing where every speech signal is an array.
- The screen of the computer is also displayed by an array. Here we use a multidimensional array.
- The array is used in many management systems like a library, students, parliament, etc.
- The array is used in the online ticket booking system. Contacts on a cell phone are displayed by this array.
- In games like online chess, where the player can store his past moves as well as current moves. It indicates a hint of position.
- To save images in a specific dimension in the android Like 360*1200

## Q 4. Explain the process behind storing a variable in memory.

**Ans :** A variable is stored in memory based on the amount of memory that is needed. Following are the steps followed to store a variable:

- The required amount of memory is assigned first.
- Then, it is stored based on the data structure being used.
- Using concepts like dynamic allocation ensures high efficiency and that the storage units can be accessed based on requirements in real-time.

## Q 5. Can you explain the difference between file structure and storage structure?

## Ans :

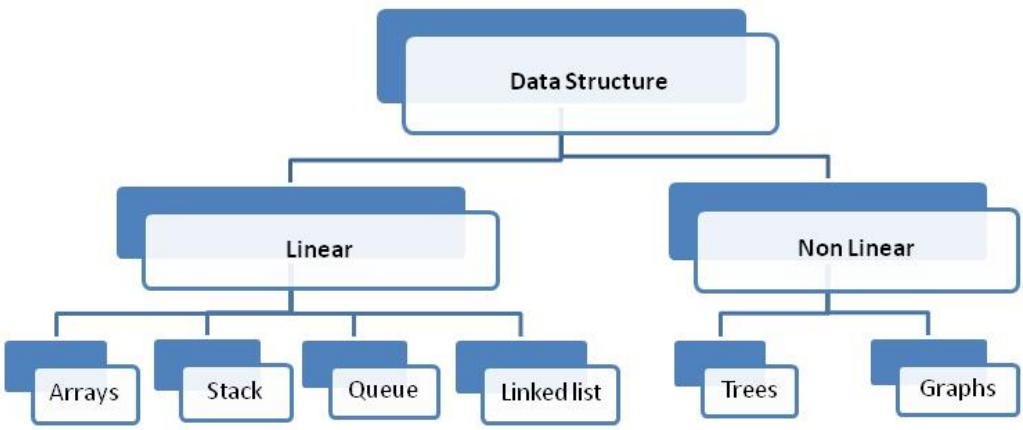
- **File Structure:** Representation of data into secondary or auxiliary memory say any device such as a hard disk or pen drives that stores data which remains intact until manually deleted is known as a file structure representation.
- **Storage Structure:** In this type, data is stored in the main memory i.e RAM, and is deleted once the function that uses this data gets completely executed.

The difference is that the storage structure has data stored in the memory of the computer system, whereas the file structure has the data stored in the auxiliary memory.

## Q 6. Describe the types of Data Structures?

## Ans :

- **Linear Data Structure:** A data structure that includes data elements arranged sequentially or linearly, where each element is connected to its previous and next nearest elements, is referred to as a linear data structure. Arrays and linked lists are two examples of linear data structures.

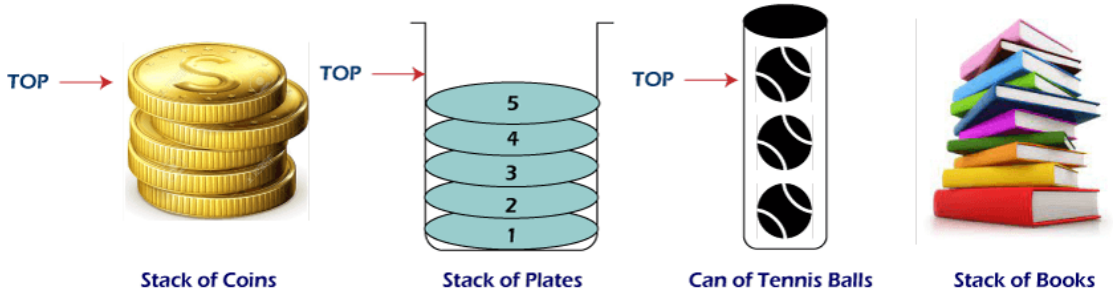


➢ **Non-Linear Data Structure:** Non-linear data structures are data structures in which data elements are not arranged linearly or sequentially. We cannot walk through all elements in one pass in a non-linear data structure, as in a linear data structure. Trees and graphs are two examples of non-linear data structures.

## Q 7. What is a stack data structure? What are the applications of stack?

## Ans :

- A stack is a data structure that is used to represent the state of an application at a particular point in time.
- The stack consists of a series of items that are added to the top of the stack and then removed from the top.
- It is a linear data structure that follows a particular order in which operations are performed.
- LIFO (Last In First Out) or FILO (First In Last Out) are two possible orders.
- A stack consists of a sequence of items.
- The element that's added last will come out first, a real-life example might be a stack of clothes on top of each other. When we remove the cloth that was previously on top, we can say that the cloth that was added last comes out first.




**Following are some applications for stack data structure:**

1) It acts as temporary storage during recursive operations
2) Redo and Undo operations in doc editors

3) Reversing a string
4) Parenthesis matching
5) Postfix to Infix Expressions
6) Function calls order

## Q 8. What are different operations available in stack data structure?

**Ans :** Some of the main operations provided in the stack data structure are:

- **push:** This adds an item to the top of the stack. The overflow condition occurs if the stack is full.
- **pop:** This removes the top item of the stack. Underflow condition occurs if the stack is empty.
- **top:** This returns the top item from the stack.
- **isEmpty:** This returns true if the stack is empty else false.
- **size:** This returns the size of the stack.

## Q 9. What is a queue data structure? What are the applications of queue?

**Ans :**

- A queue is a linear data structure that allows users to store items in a list in a systematic manner.
- The items are added to the queue at the rear end until they are full, at which point they are removed from the queue from the front.
- Queues are commonly used in situations where the users want to hold items for a long period of time, such as during a checkout process.

- A good example of a queue is any queue of customers for a resource where the first consumer is served first.


Following are some applications of queue data structure:

1) Breadth-first search algorithm in graphs
2) Operating system: job scheduling operations, Disk scheduling, CPU scheduling etc.
3) Call management in call centres

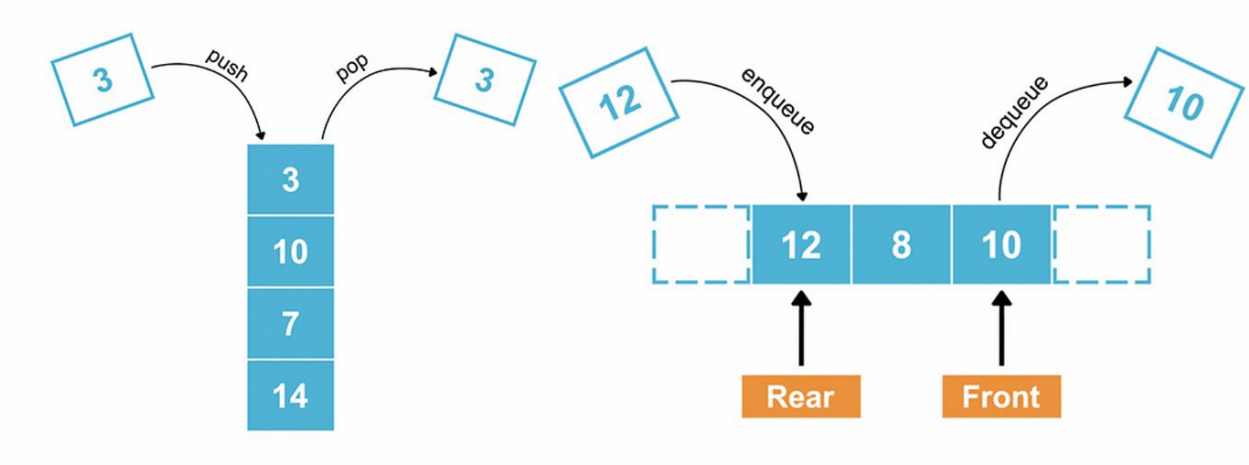
## Q 10. What are different operations available in queue data structure?

## Ans :

- **enqueue:** This adds an element to the rear end of the queue. Overflow conditions occur if the queue is full.
- **dequeue:** This removes an element from the front end of the queue. Underflow conditions occur if the queue is empty.
- **isEmpty:** This returns true if the queue is empty or else false.
- **rear:** This returns the rear end element without removing it.
- **front:** This returns the front-end element without removing it.
- **size:** This returns the size of the queue.

## Q 11. Differentiate between stack and queue data structure.

## Ans :




| Stack | Queue |
|---|---|
| Stack is a linear data structure where data is added and removed from the top. | Queue is a linear data structure where data is ended at the rear end and removed from the front. |
| Stack is based on LIFO(Last In First Out) principle | Queue is based on FIFO(First In First Out) principle |
| Insertion operation in Stack is known as push. | Insertion operation in Queue is known as eneque. |
| Delete operation in Stack is known as pop. | Delete operation in Queue is known as dequeue. |
| Only one pointer is available for both addition and deletion: top() | Two pointers are available for addition and deletion: front() and rear() |

| Stack | Queue |
|---|---|
| | |
| Used in solving recursion problems | Used in solving sequential processing problems |

## Q 12. How to implement a queue using stack?

**Ans :** A queue can be implemented using **two stacks**. Let q be the queue andstack1 and stack2 be the 2 stacks for implementing q.

We know that stack supports push, pop, and peek operations and using these operations, we need to emulate the operations of the queue - enqueue and dequeue.

Hence, queue q can be implemented in two methods (Both the methods use auxillary space complexity of O(n)):

### 1. By making enqueue operation costly:

- Here, the oldest element is always at the top of stack1 which ensures dequeue operation occurs in O(1) time complexity.
- To place the element at top of stack1, stack2 is used.

### Pseudocode:

**Enqueue:** Here time complexity will be O(n)

```
enqueue(q, data):
While stack1 is not empty:
    Push everything from stack1 to stack2.
    Push data to stack1
```

Push everything back to stack1.

**Dequeue:** Here time complexity will be O(1)

```
deQueue(q):
If stack1 is empty then error  else
Pop an item from stack1 and return it
```

## 2. By making the dequeue operation costly:

- Here, for enqueue operation, the new element is pushed at the top of stack1. Here, the enqueue operation time complexity is O(1).
- In dequeue, if stack2 is empty, all elements from stack1 are moved to stack2 and top of stack2 is the result. Basically, reversing the list by pushing to a stack and returning the first enqueued element. This operation of pushing all elements to a new stack takes O(n) complexity.

### Pseudocode:

**Enqueue:** Time complexity: O(1)

```
enqueue(q, data):
Push data to stack1
```

**Dequeue:** Time complexity: O(n)

```
dequeue(q):
If both stacks are empty then raise error.
If stack2 is empty:
While stack1 is not empty:
push everything from stack1 to stack2.
Pop the element from stack2 and return it.
```

## Q 13. How do you implement stack using queues?

## Ans :

- A stack can be implemented using two queues. We know that a queue supports enqueue and dequeue operations. Using these operations, we need to develop push, pop operations.
- Let stack be 's' and queues used to implement be 'q1' and 'q2'. Then, stack 's' can be implemented in two ways:

### 1. By making push operation costly:

- This method ensures that the newly entered element is always at the front of 'q1' so that pop operation just dequeues from 'q1'.
- 'q2' is used as auxillary queue to put every new element in front of 'q1' while ensuring pop happens in O(1) complexity.

**Pseudocode:**

Push element to stack s: Here push takes O(n) time complexity.

```
push(s, data):
  Enqueue data to q2
  Dequeue elements one by one from q1 and enqueue to q2.
  Swap the names of q1 and q2
```

Pop element from stack s: Takes O(1) time complexity.

```
pop(s):
dequeue from q1 and return it.
```

### 2. By making pop operation costly:

- In push operation, the element is enqueued to q1.

- In pop operation, all the elements from q1 except the last remaining element, are pushed to q2 if it is empty. That last element remaining of q1 is dequeued and returned.

**Pseudocode:**

Push element to stack s: Here push takes O(1) time complexity.

```
push(s,data):
Enqueue data to q1
```

Pop element from stack s: Takes O(n) time complexity.

```
pop(s):
Step1: Dequeue every elements except the last element from q1 and enqueue to q2.
Step2: Dequeue the last item of q1, the dequeued item is stored in result variable.
Step3: Swap the names of q1 and q2 (for getting updated data after dequeue)
Step4: Return the result.
```

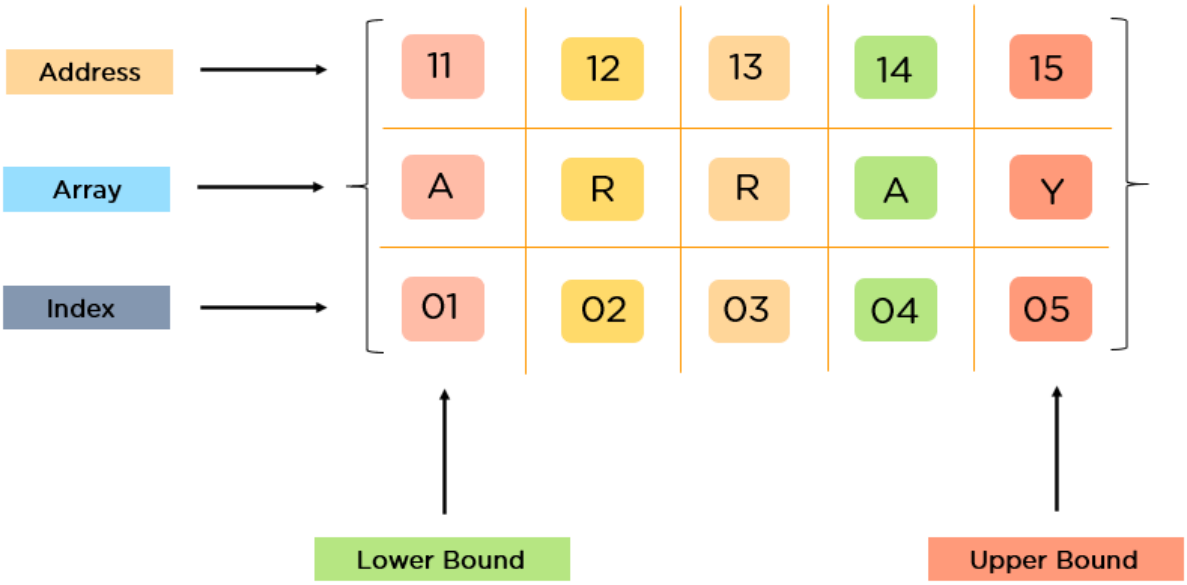
## Q 14. What is array data structure? What are the applications of arrays?

**Ans :** An array data structure is a data structure that is used to store data in a way that is efficient and easy to access.

It is similar to a list in that it stores data in a sequence. However, an array data structure differs from a list in that it can hold much more data than a list can.

An array data structure is created by combining several arrays together.

Each array is then given a unique identifier, and each array ' s data is stored in the order in which they are created.



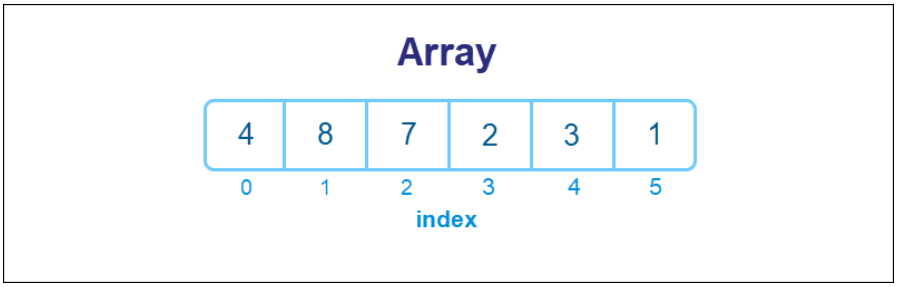

Array data structures are commonly used in databases and other computer systems to store large amounts of data efficiently.

They are also useful for storing information that is frequently accessed, such as large amounts of text or images.

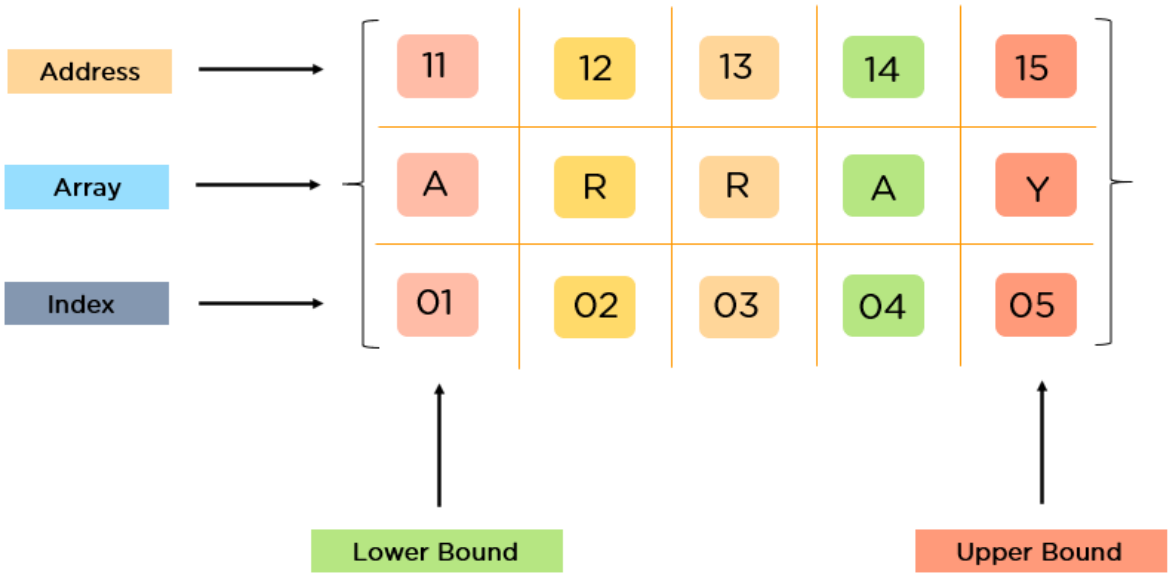
## Q 15. Elaborate on different types of array data structure

**Ans :** There are several different types of arrays:

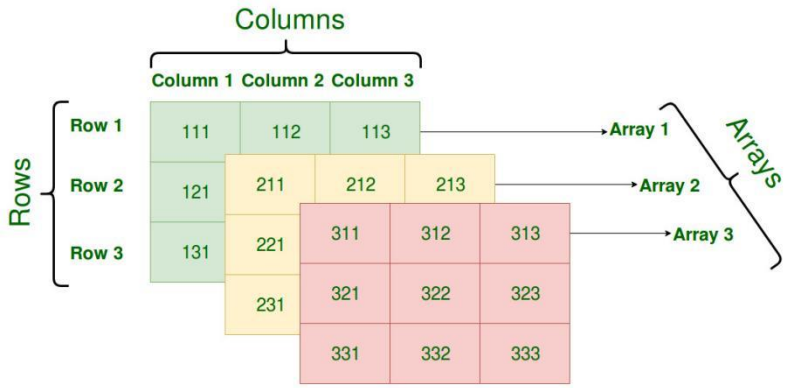
- **One-dimensional array:** A one-dimensional array stores its elements in contiguous memory locations, accessing them using a single index value. It is a linear data structure holding all the elements in a sequence.



➢ **Two-dimensional array:** A two-dimensional array is a tabular array that includes rows and columns and stores data. An M × N two-dimensional array is created by grouping M rows and N columns into N columns and rows.




➢ **Three-dimensional array:** A three-dimensional array is a grid that has rows, columns, and depth as a third dimension. It comprises a cube with rows, columns, and depth as a third dimension. The three-dimensional array has three subscripts for a position in a particular row, column, and depth. Depth (dimension or layer) is the first index, row index is the second index, and column index is the third index.

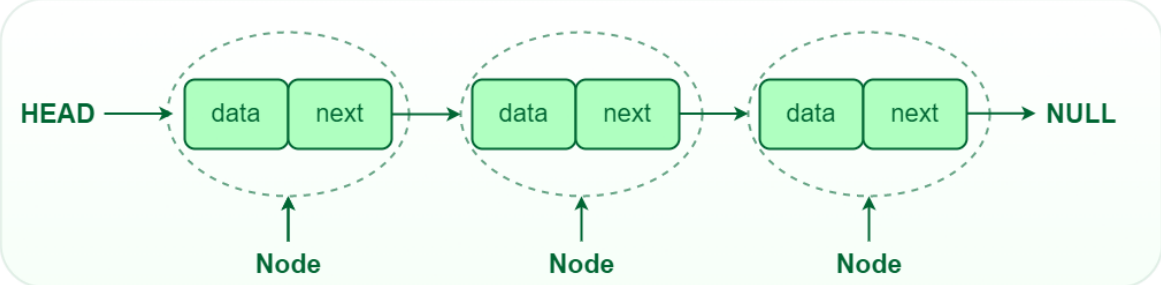


## Q 16. What is a linked list data structure? What are the applications for the Linked list?

**Ans :** A linked list can be thought of as a series of linked nodes (or items) that are connected by links (or paths).

Each link represents an entry into the linked list, and each entry points to the next node in the sequence.

The order in which nodes are added to the list is determined by the order in which they are created.




Following are some applications of linked list data structure:

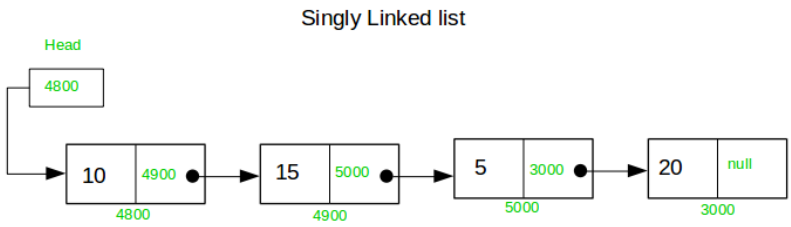
1) Stack, Queue, binary trees, and graphs are implemented using linked lists.
2) Dynamic management for Operating System memory.
3) Round robin scheduling for operating system tasks.
4) Forward and backward operation in the browser.

## Q 17. Elaborate on different types of Linked List data structures?

**Ans :** Following are different types of linked lists:

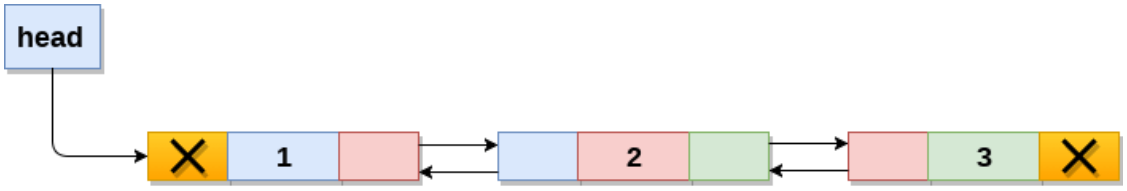
**1. Singly Linked List:** A singly linked list is a data structure that is used to store multiple items.

- The items are linked together using the key. The key is used to identify the item and is usually a unique identifier.
- In a singly linked list, each item is stored in a separate node. The node can be a single object or it can be a collection of objects.
- When an item is added to the list, the node is updated and the new item is added to the end of the list.
- When an item is removed from the list, the node that contains the removed item is deleted and its place is taken by another node.
- The key of a singly linked list can be any type of data structure that can be used to identify an object.
- For example, it could be an integer, a string, or even another singly linked list. Singly-linked lists are useful for storing many different types of data.
- For example, they are commonly used to store lists of items such as grocery lists or patient records.
- They are also useful for storing data that is time sensitive such as stock market prices or flight schedules.



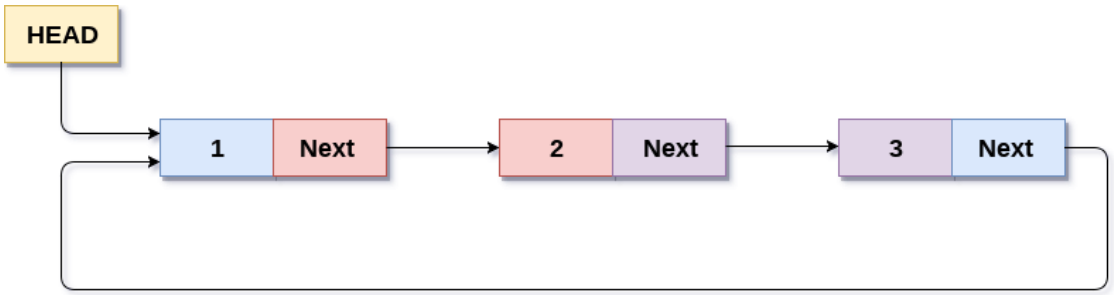

**2. Doubly Linked List:** A doubly linked list is a data structure that allows for two-way data access such that each node in the list points to the next node in the list and also points back to its previous node.

- In a doubly linked list, each node can be accessed by its address, and the contents of the node can be accessed by its index.
- It's ideal for applications that need to access large amounts of data in a fast manner.
- A disadvantage of a doubly linked list is that it is more difficult to maintain than a single-linked list.
- In addition, it is more difficult to add and remove nodes than in a single-linked list.



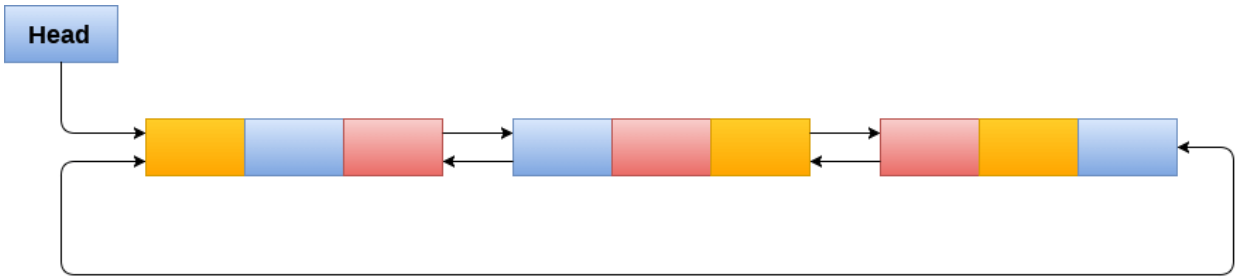

Doubly Linked List

**3. Circular Linked List:** A circular linked list is a unidirectional linked list where each node points to its next node and the last node points back to the first node, which makes it circular.




Circular Singly Linked List

**4. Doubly Circular Linked List:** A doubly circular linked list is a linked list where each node points to its next node and its previous node and the last node points back to the first node and first node' s previous points to the last node.




Circular Doubly Linked List

**5. Header List:** A list that contains the header node at the beginning of the list, is called the header-linked list. This is helpful in calculating some repetitive operations like the number of elements in the list etc.

## Q 18. Difference between Array and Linked List.

## Ans :

| Arrays | Linked Lists |
|---|---|
| An array is a collection of data elements of the same type. | A linked list is a collection of entities known as nodes. The node is divided into two sections: data and address. |
| It keeps the data elements in a single memory. | It stores elements at random, or anywhere in the memory. |
| The memory size of an array is | The memory size of a linked list is allocated |

| Arrays | Linked Lists |
|---|---|
| fixed and cannot be changed during runtime. | during runtime. |
| An array's elements are not dependent on one another. | Linked List elements are dependent on one another. |
| It is easier and faster to access an element in an array. | In the linked list, it takes time to access an element. |
| Memory utilization is ineffective in the case of an array. | Memory utilization is effective in the case of linked lists. |
| Operations like insertion and deletion take longer time in an array. | Operations like insertion and deletion are faster in the linked list. |

## Q 19. What is an asymptotic analysis of an algorithm?

**Ans :** Asymptotic analysis of an algorithm defines the run-time performance as per its mathematical boundations.

Asymptotic analysis helps us articulate the best case(Omega Notation, Ω), average case(Theta Notation, θ), and worst case(Big Oh Notation, O) performance of an algorithm.

## Q 20. What is hashmap in data structure?

**Ans :** Hashmap is a data structure that uses an implementation of a hash table data structure which allows access to data in constant time (O(1)) complexity if you have the key.

## Q 21. What is the requirement for an object to be used as key or value in HashMap?

**Ans :**

- The key or value object that gets used in the hashmap must implement equals() and hashcode() method.
- The hash code is used when inserting the key object into the map and the equals method is used when trying to retrieve a value from the map.

## Q 22. How does HashMap handle collisions in Java?

**Ans :**

- The java.util.HashMap class in Java uses the approach of chaining to handle collisions. In chaining, if the new values with the same key are attempted to be pushed, then these values are stored in a linked list stored in a bucket of the key as a chain along with the existing value.
- In the worst-case scenario, it can happen that all keys might have the same hashcode, which will result in the hash table turning into a linked list. In this case, searching a value will take O(n) complexity as opposed to O(1) time due to the nature of the linked list. Hence, care has to be taken while selecting hashing algorithm.

## Q 23. What is an algorithm? What is the need for an algorithm?

**Ans :** An algorithm is a well-defined computational procedure that takes some values or the set of values, as an input and produces a set of values or some values, as an output.

**Need for Algorithm**

- The algorithm provides the basic idea of the problem and an approach to solve it. Some reasons to use an algorithm are as follows.The algorithm improves the efficiency of an existing technique.
- To compare the performance of the algorithm with respect to other techniques.
- The algorithm gives a strong description of requirements and goal of the problems to the designer.
- The algorithm provides a reasonable understanding of the flow of the program.
- The algorithm measures the performance of the methods in different cases (Best cases, worst cases, average cases).
- The algorithm identifies the resources (input/output, memory) cycles required by the algorithm.
- With the help of an algorithm, we can measure and analyze the complexity time and space of the problems.
- The algorithm also reduces the cost of design.

## Q 24. What is the Complexity of Algorithm?

**Ans :** The complexity of the algorithm is a way to classify how efficient an algorithm is compared to alternative ones.

Its focus is on how execution time increases with the data set to be processed. The computational complexity of the algorithm is important in computing.

It is very suitable to classify algorithm based on the relative amount of time or relative amount of space they required and specify the growth of time/ space requirement as a function of input size.

### Time complexity

Time complexity is a Running time of a program as a function of the size of the input.

### Space complexity

Space complexity analyzes the algorithms, based on how much space an algorithm needs to complete its task. Space complexity analysis was critical in the early days of computing (when storage space on the computer was limited).

Nowadays, the problem of space rarely occurs because space on the computer is broadly enough.

We achieve the following types of analysis for complexity

1) **Worst-case: f(n)**

It is defined by the maximum number of steps taken on any instance of size n.

2) **Best-case: f(n)**

It is defined by the minimum number of steps taken on any instance of size n.

3) **Average-case: f(n)**

It is defined by the average number of steps taken on any instance of size n.

## Q 25. Write an algorithm to reverse a string. For example, if my string is "uhsnamiH" then my result will be "Himanshu".

**Ans :** Algorithm to reverse a string.

**Step1:** start

**Step2:** Take two variable i and j

**Step3:** do length (string)-1, to set J at last position

**Step4:** do string [0], to set i on the first character.

**Step5:** string [i] is interchanged with string[j]

**Step6:** Increment i by 1

**Step7:** Increment j by 1

**Step8:** if i>j then go to step3

**Step9:** Stop

## Q 26 . Write an algorithm to insert a node in a sorted linked list.

**Ans :** Algorithm to insert a node in a sorted linked list.

**Case1:**

Check if the linked list is empty then set the node as head and return it.

1) New_node-> Next= head;
2) Head=New_node

**Case2:**

Insert the new node in middle

1)While( P!= insert position)
2) {
3) P= p-> Next;

4) }
5) Store_next=p->Next;
6) P->Next= New_node;
7) New_Node->Next = Store_next;

**Case3:**

Insert a node at the end

1) While (P->next!= **null**)
2) {
3) P= P->Next;
4) }
5) P->Next = New_Node;
6) New_Node->Next = **null**;

## Q 27. What are the Asymptotic Notations?

**Ans :** Asymptotic analysis is used to measure the efficiency of an algorithm that doesn't depend on machine-specific constants and prevents the algorithm from comparing the time taking algorithm.

Asymptotic notation is a mathematical tool that is used to represent the time complexity of algorithms for asymptotic analysis.

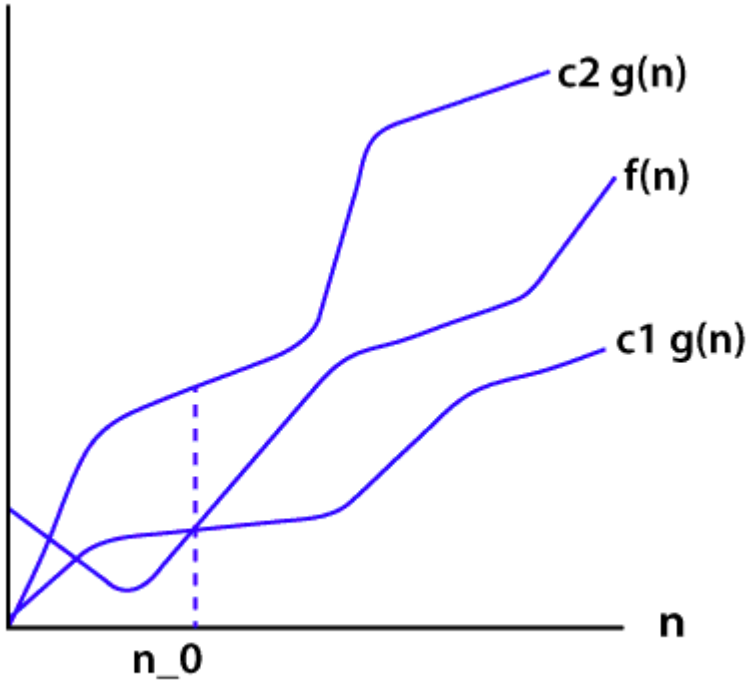
The three most used asymptotic notation is as follows.

### ➢ θ Notation

θ Notation defines the exact asymptotic behavior.

To define a behavior, it bounds functions from above and below.

A convenient way to get Theta notation of an expression is to drop low order terms and ignore leading constants.



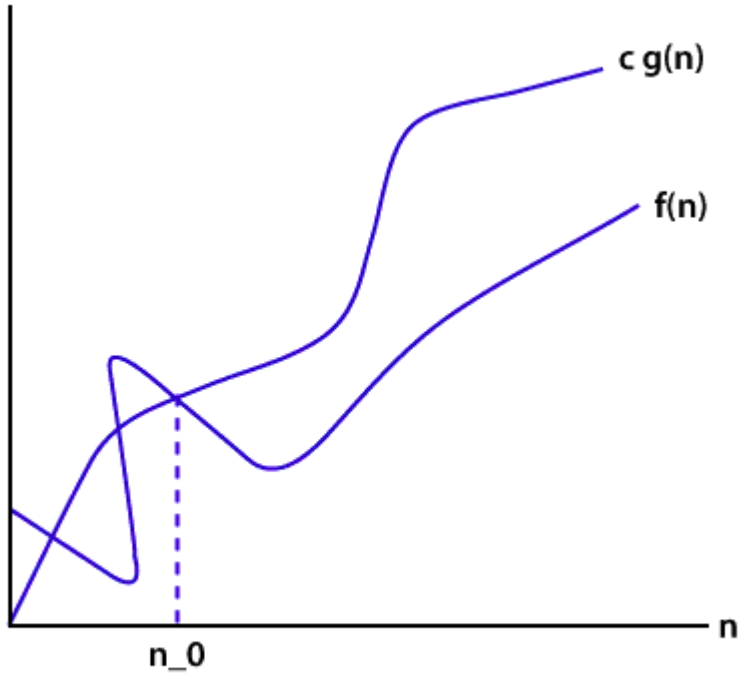

f(n)= theta(g(n))

## ➢ Big O Notation

The Big O notation bounds a function from above, it defines an upper bound of an algorithm.

Let's consider the case of insertion sort; it takes linear time in the best case and quadratic time in the worst case.

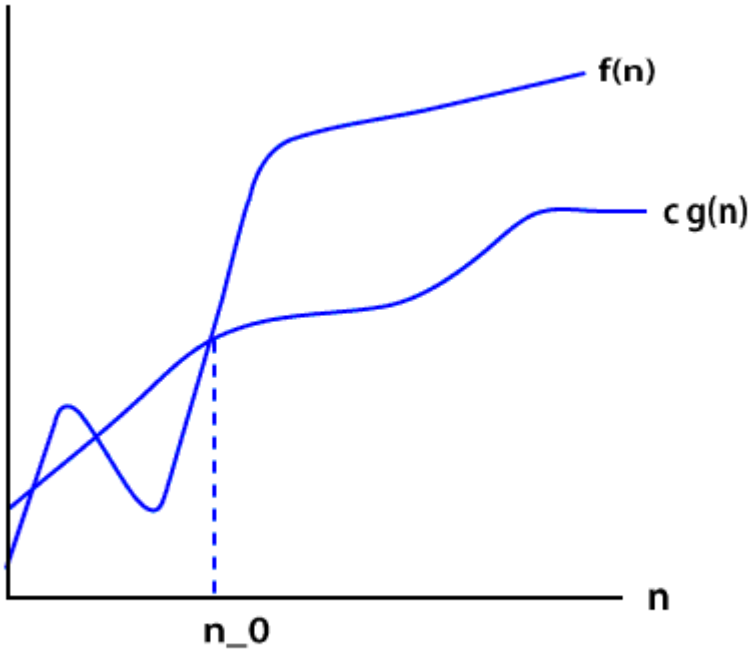
The time complexity of insertion sort is $O(n^2)$. It is useful when we only have upper bound on time complexity of an algorithm.




f(n)=O(g(n))

## ➢ Ω Notation

Just like Big O notation provides an asymptotic upper bound, the **Ω Notation** provides an asymptotic lower bound on a function. It is useful when we have lower bound on time complexity of an algorithm.




f(n)= Omega(g(n))

## Q 28 . Explain the Bubble sort algorithm?

**Ans :** Bubble sort is the simplest sorting algorithm among all sorting algorithm. It repeatedly works by swapping the adjacent elements if they are in the wrong order.

e.g.

(72538) we have this array for sorting.

**Pass1:**

(**72**538) -> (27538) swap 7 and 2.
(2**75**38) -> (25738) swap 7 and 5.
(25**73**8) -> (25378) swap 7 and 3.
(2537**8**) -> (25378) algorithm does not swap 7 and 8 because 7<8.

**Pass2:**

(25378) -> (25378) algorithm does not swap 2 and 5 because 2<5.
(25378) -> (23578) swap 3 and 5.
(23578) -> (23578) algorithm does not swap 5 and 7 because 5<7.
(23578) -> (23578) algorithm does not swap 7 and 8 because 7<8.

Here, the sorted element is (23578).

## Q 29. How to swap two integers without swapping the temporary variable in Java?

**Ans :** It's a very commonly asked trick question. There are many ways to solve this problem.

- But the necessary condition is we have to solve it without swapping the temporary variable.
- If we think about integer overflow and consider its solution, then it creates an excellent impression in the eye of interviewers.
- Suppose we have two integers I and j, the value of i=7 and j=8 then how will you swap them without using a third variable. This is a journal problem.
- We need to do this using Java programming constructs. We can swap numbers by performing some mathematical operations like addition, subtraction, multiplication, and division. But maybe it will create the problem of integer overflow.

### Using addition and subtraction

1) a= a + b;
2) b=a - b; // this will act like (a+b)-b, now b is equal to a.
3) a=a - b; // (a+b)-a, now, a is equal to b.

It is a nice trick. But in this trick, the integer will overflow if the addition is more than the maximum value of int primitive as defined by

Integer.MAX_VALUE and if subtraction is less than minimum value i.e., Integer.MIN_VALUE.

### Using XOR trick

Another solution to swap two integers without using a third variable (temp variable) is widely recognized as the best solution, as it will also work in a language which doesn't handle integer overflow like Java example C, C++. Java supports several bitwise operators. One of them is XOR (denoted by ^).

1) x=x^y;
2) y=x^y;
3) x=x^y;

## Q 30. What is a Hash Table? How can we use this structure to find all anagrams in a dictionary?

## Ans :

- A Hash table is a data structure for storing values to keys of arbitrary type.
- The Hash table consists of an index into an array by using a Hash function. Indexes are used to store the elements.
- We assign each possible element to a bucket by using a hash function.
- Multiple keys can be assigned to the same bucket, so all the key and value pairs are stored in lists within their respective buckets.
- Right hashing function has a great impact on performance.
- To find all anagrams in a dictionary, we have to group all words that contain the same set of letters in them.
- So, if we map words to strings representing their sorted letters, then we could group words into lists by using their sorted letters as a key.

```
FUNCTION find_anagrams(words)
word_groups = HashTable<String, List>
FOR word IN words
 word_groups.get_or_default(sort(word), []).push(word)
 END FOR
anagrams = List
FOR key, value IN word_groups
anagrams.push(value)
END FOR
RETURN anagrams
```

The hash table contains lists mapped to strings. For each word, we add it to the list at the suitable key, or create a new list and add it to it.

## Q 31. What is Divide and Conquer algorithms?

**Ans :** Divide and Conquer is not an algorithm; it's a pattern for the algorithm. It is designed in a way as to take dispute on a huge input, break the input into minor pieces, and decide the problem for each of the small pieces.

Now merge all of the piecewise solutions into a global solution. This strategy is called divide and conquer.



**Divide and conquer uses the following steps to make a dispute on an algorithm.**

- **Divide:** In this section, the algorithm divides the original problem into a set of subproblems.
- **Conquer:** In this section, the algorithm solves every subproblem individually.
- **Combine:** In this section, the algorithm puts together the solutions of the subproblems to get the solution to the whole problem.

## Q 32. Explain the BFS algorithm?

**Ans :** BFS (Breadth First Search) is a graph traversal algorithm.

It starts traversing the graph from the root node and explores all the neighboring nodes.

It selects the nearest node and visits all the unexplored nodes.

The algorithm follows the same procedure for each of the closest nodes until it reaches the goal state.

**Algorithm**

**Step1:** Set status=1 (ready state)

**Step2:** Queue the starting node A and set its status=2, i.e. (waiting state)

**Step3:** Repeat steps 4 and 5 until the queue is empty.

**Step4:** Dequeue a node N and process it and set its status=3, i.e. (processed state)

**Step5:** Queue all the neighbors of N that are in the ready state (status=1) and set their status =2 (waiting state)
[Stop Loop]

**Step6:** Exit

## Q 33. What is Dijkstra's shortest path algorithm?

**Ans :** Dijkstra's algorithm is an algorithm for finding the shortest path from a starting node to the target node in a weighted graph.

The algorithm makes a tree of shortest paths from the starting vertex and source vertex to all other nodes in the graph.

Suppose you want to go from home to office in the shortest possible way. You know some roads are heavily congested and challenging to use this, means these edges have a large weight. In Dijkstra's algorithm, the shortest path tree found by the algorithm will try to avoid edges with larger weights

## Q 34. Give some examples of Divide and Conquer algorithm?

**Ans :** Some problems that use Divide and conquer algorithm to find their solution are listed below.

1) Merge Sort
2) Quick Sort
3) Binary Search
4) Strassen's Matrix Multiplication
5) Closest pair (points)

## Q 35. What are Greedy algorithms? Give some example of it?

**Ans :**

- A greedy algorithm is an algorithmic strategy which is made for the best optimal choice at each sub stage with the goal of this, eventually leading to a globally optimum solution.
- This means that the algorithm chooses the best solution at the moment without regard for consequences.

- In other words, an algorithm that always takes the best immediate, or local, solution while finding an answer.
- Greedy algorithms find the overall, ideal solution for some idealistic problems, but may discover less-than-ideal solutions for some instances of other problems.

Below is a list of algorithms that finds their solution with the use of the Greedy algorithm.

- Travelling Salesman Problem
- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm
- Dijkstra's Minimal Spanning Tree Algorithm
- Graph - Map Coloring
- Graph - Vertex Cover
- Knapsack Problem
- Job Scheduling Problem

## Q 36 . What is a linear search?

**Ans :** Linear search is used on a group of items. It relies on the technique of traversing a list from start to end by visiting properties of all the elements that are found on the way.

For example, suppose an array of with some integer elements. You should find and print the position of all the elements with their value. Here, the linear search acts in a flow like matching each element from the beginning of the list to the end of the list with the integer, and if the condition is `True then printing the position of the element.'

### Implementing Linear Search

Below steps are required to implement the linear search.

**Step1:** Traverse the array using **for loop**.

**Step2:** In every iteration, compare the target value with the current value of the array

**Step3:** If the values match, return the current index of the array

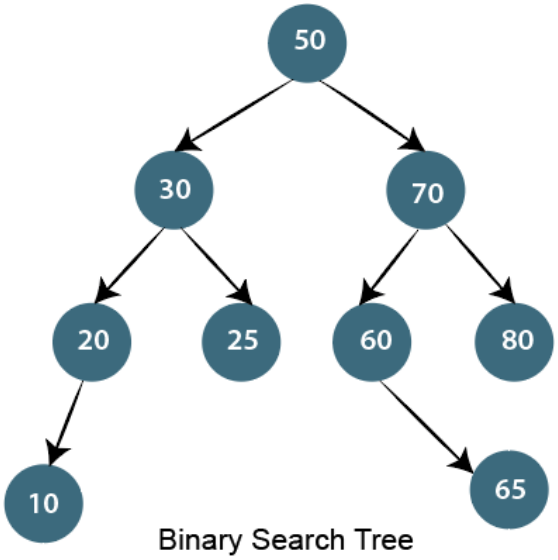
**Step4:** If the values do not match, shift on to the next array element.

**Step5:** If no match is found, return -1

## Q 37 . What is a Binary Search Tree?

**Ans :** The binary search tree is a special type of data structure which has the following properties.

- Nodes which are less than root will be in the left subtree.
- Nodes which are greater than root (i.e., contains more value) will be right subtree.
- A binary search tree should not have duplicate nodes.
- Both sides subtree (i.e., left and right) also should be a binary search tree.



Binary Search Tree

## Q 38. Write an algorithm to insert a node in the Binary search tree?

**Ans :** Insert node operation is a smooth operation. You need to compare it with the root node and traverse left (if smaller) or right (if greater) according to the value of the node to be inserted.

**Algorithm:**

- Make the root node as the current node
- If the node to be inserted < root
    - If it has left child, then traverse left
    - If it does not have left child, insert node here
- If the node to be inserted > root
    - If it has the right child, traverse right
    - If it does not have the right child, insert node here.

## Q 39 . How to count leaf nodes of the binary tree?

**Ans :**

**Algorithm-**

Steps for counting the number of leaf nodes are:

- If the node is null (contains null values) then return 0.
- If encountered leaf node. Left is null and node Right is null then return 1.
- Recursively calculate the number of leaf nodes using

No. of leaf nodes= no of leaf nodes in left subtree + number of leaf nodes in the right subtree.

## Q 40 . How to find all possible words in a board of characters (Boggle game)?

**Ans :** In the given dictionary, a process to do a lookup in the dictionary and an M x N board where every cell has a single character.

Identify all possible words that can be formed by order of adjacent characters.

Consider that we can move to any of the available 8 adjacent characters, but a word should not have multiple instances of the same cell.

### Example:

```
dictionary[] = {"Java", "Point","Quiz"};
Array[][]   = {{'J', 'T', 'P',},
{'U', 'A', 'A'},
{'Q', 'S', 'V'}};
isWord(str): returns true if str is present in dictionary
  else false.
```

### Output:

```
Following words of the dictionary are present
JAVA
```

## Q 41. Write an algorithm to insert a node in a link list?

## Ans :

**Algorithm**

- Check If the Linked list does not have any value then make the node as head and return it
- Check if the value of the node to be inserted is less than the value of the head node, then insert the node at the start and make it head.
- In a loop, find the appropriate node after which the input node is to be inserted. To find the just node start from the head, keep forwarding until you reach a node whose value is greater than the input node. The node just before is the appropriate node.
- Insert the node after the proper node found in step 3.

## Q 42 . How to delete a node in a given link list? Write an algorithm and a program?

**Ans :** Write a function to delete a given node from a Singly Linked List. The function must follow the following constraints:

- ✓ The function must accept a pointer to the start node as the first argument and node to be deleted as the second argument, i.e., a pointer to head node is not global.
- ✓ The function should not return a pointer to the head node.
- ✓ The function should not accept pointer to pointer to head node.

We may assume that the Linked List never becomes empty.

**C program for deleting a node in Linked List:** We will handle the case when the first node to be deleted then we copy the data of the next node to head and delete the next node. In other cases when a deleted node is not the head node can be handled generally by finding the previous node.

## Example:

```
#include <stdio.h>
#include <stdlib.h>
struct Node { int data; struct Node* next; };
void deleteNode(struct Node** head, int key) {
    struct Node *temp = *head, *prev = NULL;
    if (temp && temp->data == key) *head = temp->next, free(temp);
    else while (temp && temp->data != key) prev = temp, temp = temp->next;
    if (!temp) printf("Key not found\n");
    else if (prev) prev->next = temp->next, free(temp);
    else *head = temp->next, free(temp);
}
void printList(struct Node* head) {
    while (head) printf("%d -> ", head->data), head = head->next;
    printf("NULL\n");              }
int main() {
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 1, head->next = NULL;
    head->next = (struct Node*)malloc(sizeof(struct Node));
    head->next->data = 2, head->next->next = NULL;
    printf("Original List:\n"), printList(head);
    int key = 2;
    deleteNode(&head, key);
    printf("List after deleting %d:\n", key), printList(head);
    return 0;
}
```

## Q 43. Write a c program to merge a link list into another at an alternate position?

**Ans :** We have two linked lists, insert nodes of the second list into the first list at substitute positions of the first list.

### Example

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int data;
   struct Node* next;
};

struct Node* mergeAtAlternate(struct Node* list1, struct Node* list2) {
   struct Node *merged = NULL, *temp1 = list1, *temp2 = list2;
   while (temp1 || temp2) {
      if (temp1) merged = (struct Node*)malloc(sizeof(struct Node)), merged->data = temp1->data, merged->next = merged;
      if (temp2) merged = (struct Node*)malloc(sizeof(struct Node)), merged->data = temp2->data, merged->next = merged;
      temp1 = temp1 ? temp1->next : temp1;
      temp2 = temp2 ? temp2->next : temp2;
   }
```

```
return merged;
}

void printList(struct Node* head) {
    while (head) printf("%d -> ", head->data), head = head->next;
printf("NULL\n");
}
int main() {
    struct Node *list1 = NULL, *list2 = NULL;
    for (int i = 1; i <= 5; i += 2) list1 = (struct Node*)malloc(sizeof(struct Node)), list1->data = i, list1->next = list1;
    for (int i = 2; i <= 6; i += 2) list2 = (struct Node*)malloc(sizeof(struct Node)), list2->data = i, list2->next = list2;
  printf("List 1: "), printList(list1);
    printf("List 2: "), printList(list2);
    struct Node* mergedList = mergeAtAlternate(list1, list2);
    printf("Merged List: "), printList(merg  edList);
    return 0;
}
```

## Q 44. Explain how the encryption algorithm works?

**Ans :** Encryption is the technique of converting plaintext into a secret code format it is also called as "Ciphertext."

To convert the text, the algorithm uses a string of bits called as "keys" for calculations.

The larger the key, the higher the number of potential patterns for Encryption.

Most of the algorithm use codes fixed blocks of input that have a length of about 64 to 128 bits, while some uses stream method for encryption.

## Q 45 . What Are The Criteria Of Algorithm Analysis?

**Ans :** An algorithm is generally analyzed by two factors.

1) Time complexity
2) Space complexity

**Time complexity** deals with the quantification of the amount of time taken by a set of code or algorithm to process or run as a function of the amount of input. In other words, the time complexity is efficiency or how long a program function takes to process a given input.

**Space complexity** is the amount of memory used by the algorithm to execute and produce the result.

## Q 46 . What are the differences between stack and Queue?

**Ans :** Stack and Queue both are non-primitive data structure used for storing data elements and are based on some real-world equivalent.

Let's have a look at key differences based on the following parameters.

### ❖ Working principle

The significant difference between stack and queue is that stack uses LIFO (Last in First Out) method to access and add data elements whereas Queue uses FIFO (First in first out) method to obtain data member.

### ❖ Structure

In Stack, the same end is used to store and delete elements, but in Queue, one end is used for insertion, i.e., rear end and another end is used for deletion of elements.

### ❖ Number of pointers used

Stack uses one pointer whereas Queue uses two pointers (in the simple case).

### ❖ Operations performed

Stack operates as Push and pop while Queue operates as Enqueue and dequeuer.

### ❖ Variants

Stack does not have variants while Queue has variants like a circular queue, Priority queue, doubly ended Queue.

### ❖ Implementation

The stack is simpler while Queue is comparatively complex.

## Q 47. What is the difference between the Singly Linked List and Doubly Linked List data structure?

**Ans :** This is a traditional interview question on the data structure. The major difference between the singly linked list and the doubly linked list is the ability to traverse.

You cannot traverse back in a singly linked list because in it a node only points towards the next node and there is no pointer to the previous node.

On the other hand, the doubly linked list allows you to navigate in both directions in any linked list because it maintains two pointers towards the next and previous node.

## Q 48 . Which Sorting Algorithm Is Considered the Fastest?

**Ans :** Quicksort is generally considered the fastest sorting algorithm. It has a best case time complexity of O(n log n) and its time complexity in the worst case is O(n^2). However, it is considered the fastest sorting algorithm because it has an average case time complexity of O(n logn), which is faster than other algorithms.

## Q 49 . How Do You Differentiate Between a Tree and Graph Data Structure?

**Ans :** The following are the key differences between trees and graphs:

- Trees have a root node from which all other nodes originate. Graphs don ' t have root nodes.
- The vertices in a graph can have bidirectional connections. There is only a single path between two vertices in a tree.
- The nodes in a graph can connect to themselves, which is not possible in a tree.
- Trees are a hierarchical data structure whereas graphs are flat networks.

## Q 50 . Can You Differentiate Between a B Tree and B+ Tree?

**Ans :** The following are the differences between B trees and B+ trees:

- Both the internal and leaf nodes in a B tree have pointers. Only the leaf nodes in a B+ tree have pointers.
- Searching a B+ tree is faster because the keys are always in the leaf nodes. The keys of a B tree are not necessarily in the leaf node and searching takes more time as a result.
- B trees don’ t maintain a duplicate of the keys in the tree. B+ trees maintain duplicates of keys.
- The lead nodes of B+ trees are stored in the form of structural linked lists. This is not the case for B trees.

# ABOUT US

**At TopperWorld, we are on a mission to empower college students with the knowledge, tools, and resources they need to succeed in their academic journey and beyond.**

### ➢ Our Vision

- **Our vision is to create a world where every college student can easily access high-quality educational content, connect with peers, and achieve their academic goals.**
- **We believe that education should be accessible, affordable, and engaging, and that's exactly what we strive to offer through our platform.**

## ➢ Unleash Your Potential

- In an ever-evolving world, the pursuit of knowledge is essential. ToppperWorld serves as your virtual campus, where you can explore a diverse array of online resources tailored to your specific college curriculum.
- Whether you're studying science, arts, engineering, or any other discipline, we've got you covered.
- Our platform hosts a vast library of e-books, quizzes, and interactive study tools to ensure you have the best resources at your fingertips.

## ➢ The ToppperWorld Community

- Education is not just about textbooks and lectures; it's also about forming connections and growing together.
- ToppperWorld encourages you to engage with your fellow students, ask questions, and share your knowledge.
- We believe that collaborative learning is the key to academic success.

## ➢ Start Your Journey with ToppperWorld

- Your journey to becoming a top-performing college student begins with ToppperWorld.
- Join us today and experience a world of endless learning possibilities.
- Together, we'll help you reach your full academic potential and pave the way for a brighter future.
- Join us on this exciting journey, and let's make academic success a reality for every college student.

# "UNLOCK YOUR POTENTIAL"

With- TOPPERWORLD

Explore More


topperworld.in

DSA TUTORIAL

C TUTORIAL

C++ TUTORIAL

JAVA TUTORIAL

PYTHON TUTORIAL

Follow Us On

E-mail

topperworld.in@gmail.com