### **1. What are Data Structures?**

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.

### **2. Why Create Data Structures?**

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.

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

Following are some real-time applications of data structures:

Decision Making

* Genetics
* Image Processing
* Blockchain
* Numerical and Statistical Analysis
* Compiler Design
* Database Design and many more

**You can download a PDF version of Data Structure Interview Questions.**

**[IMG_257Download PDF](javascript:void(0))**

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

* 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.

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

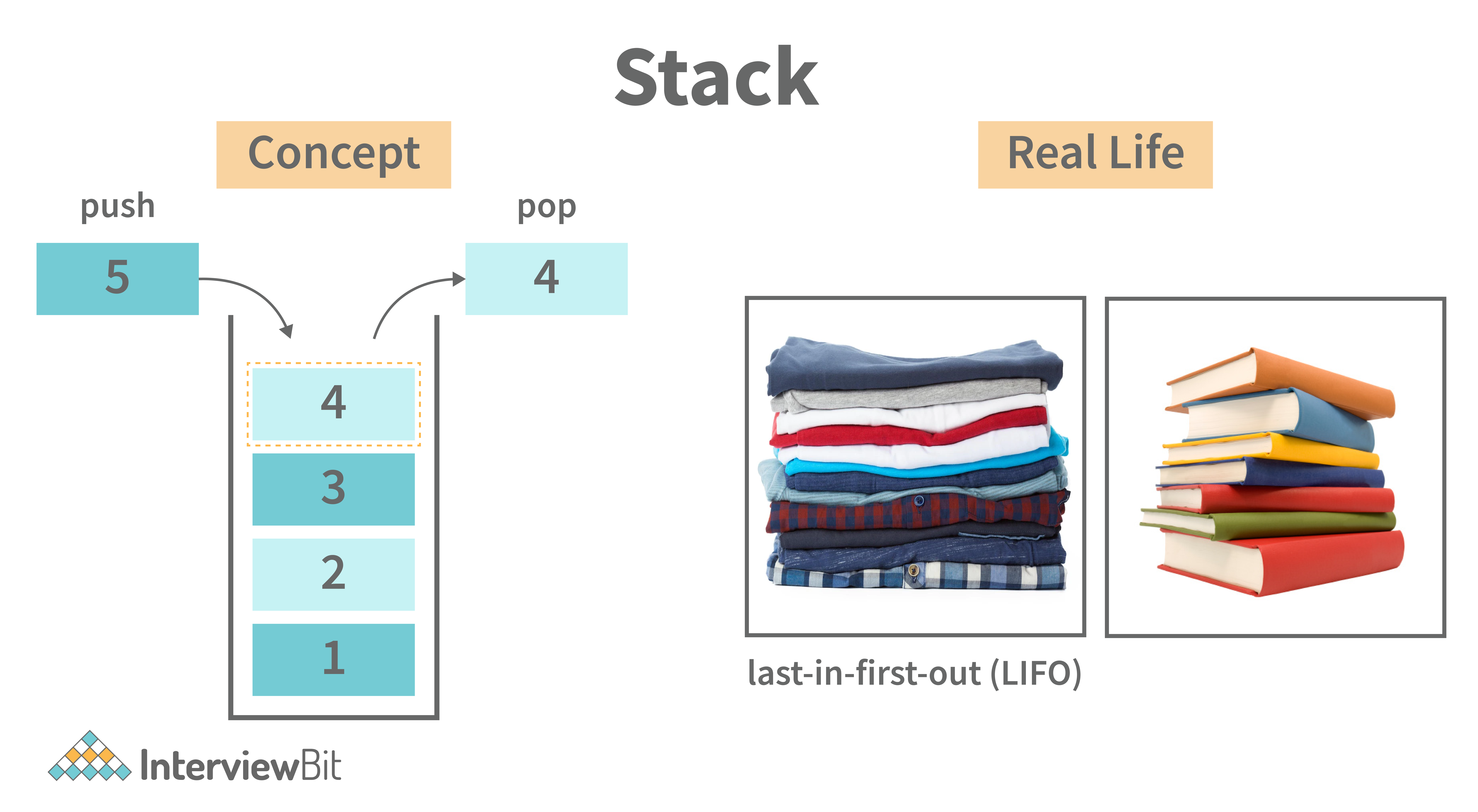
* ****File Structure:**** Representation of data into secondary or auxiliary memory say any device such as a hard disk or pen drives that stores data which remains intact until manually deleted is known as a file structure representation.
* ****Storage Structure****: In this type, data is stored in the 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.

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

* ****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.

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

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 

Following are some applications for stack data structure:

* It acts as temporary storage during recursive operations
* Redo and Undo operations in doc editors
* Reversing a string
* Parenthesis matching
* Postfix to Infix Expressions
* Function calls order

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

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.

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

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.



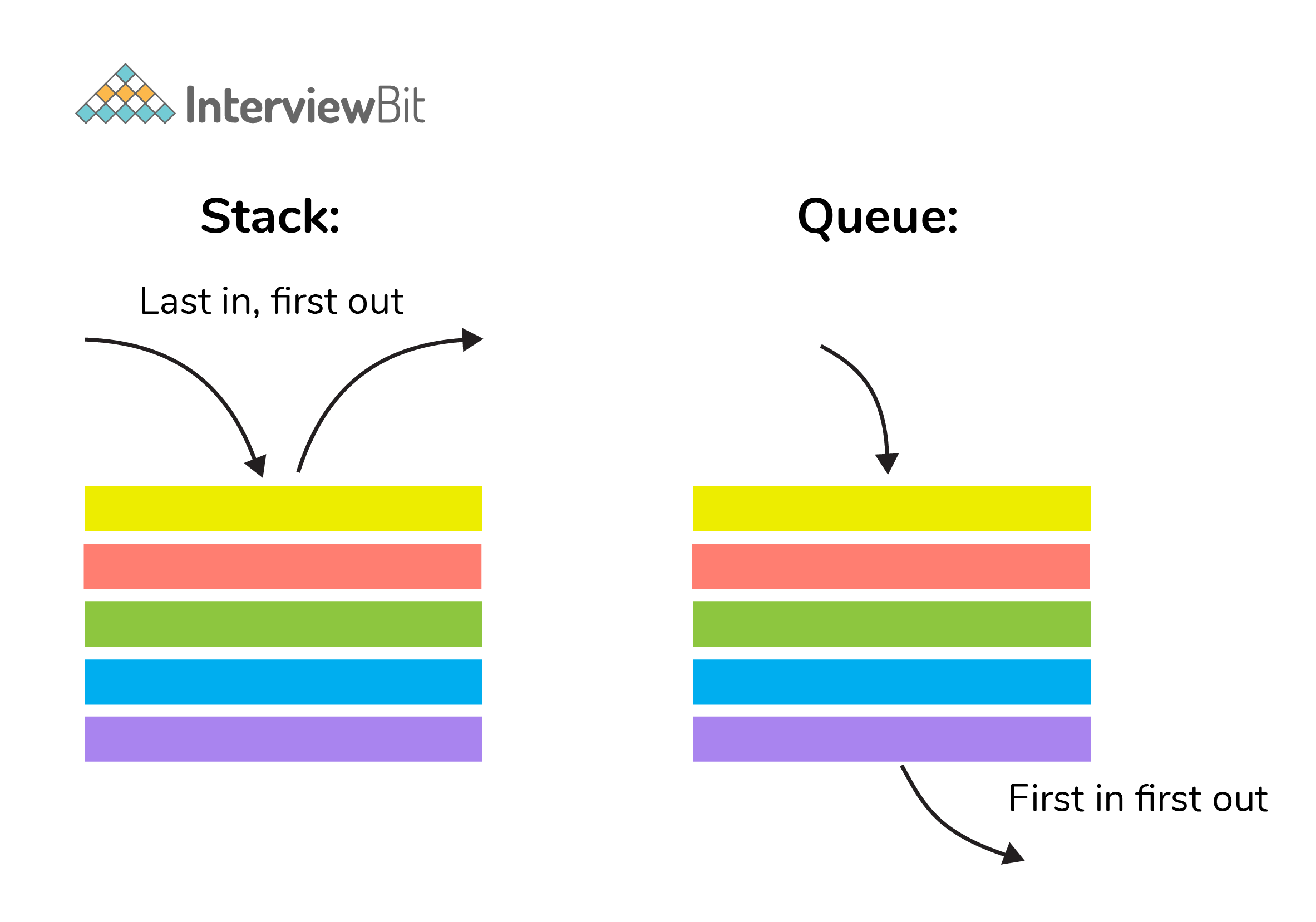
Following are some applications of queue data structure:

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

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

* ****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.

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



| **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() |
| Used in solving recursion problems | Used in solving sequential processing problems |

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

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.

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

* 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.

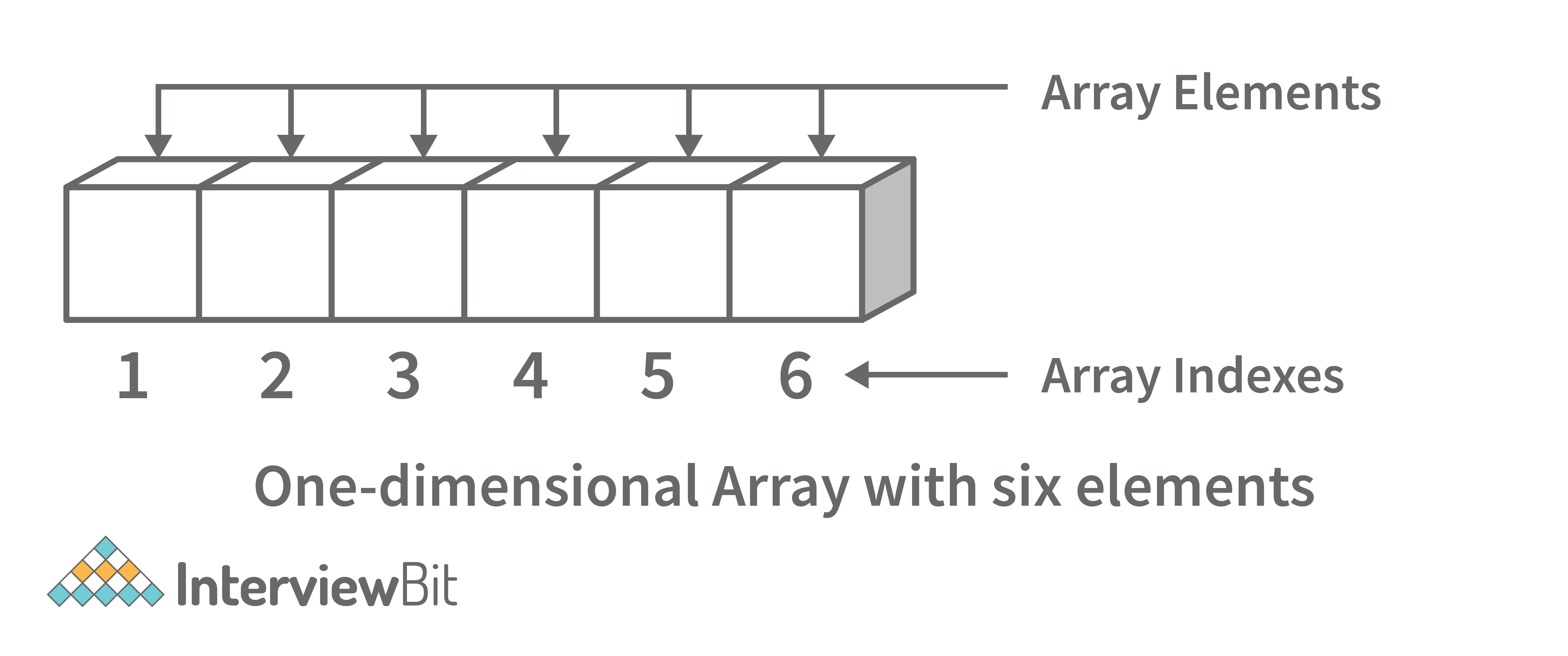
Step2: Dequeue the last item of q1, the dequeued item is stored in result variable.

Step3: Swap the names of q1 and q2 (for getting updated data after dequeue)

Step4: Return the result.

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

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.

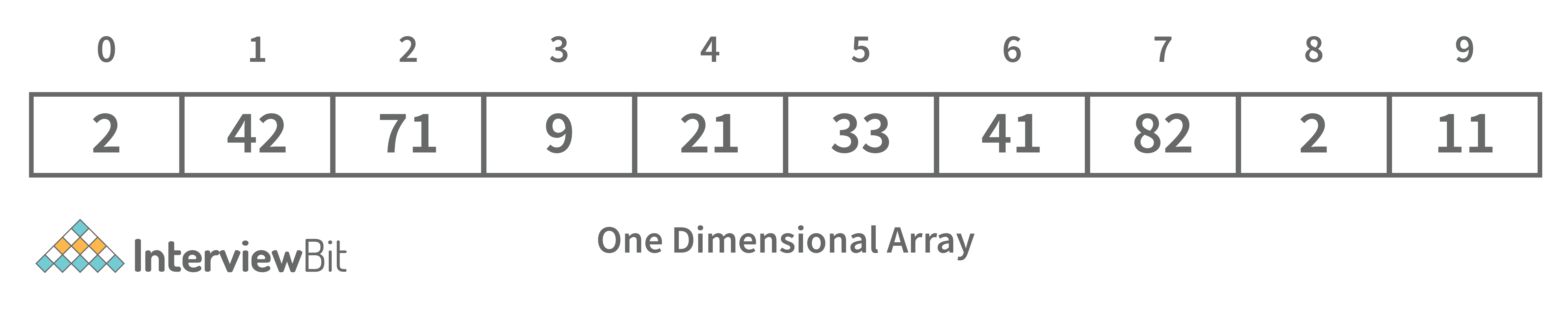


Array data structures are commonly used in databases and other computer systems to store large amounts of data efficiently. They are also useful for storing information that is frequently accessed, such as large amounts of text or images.

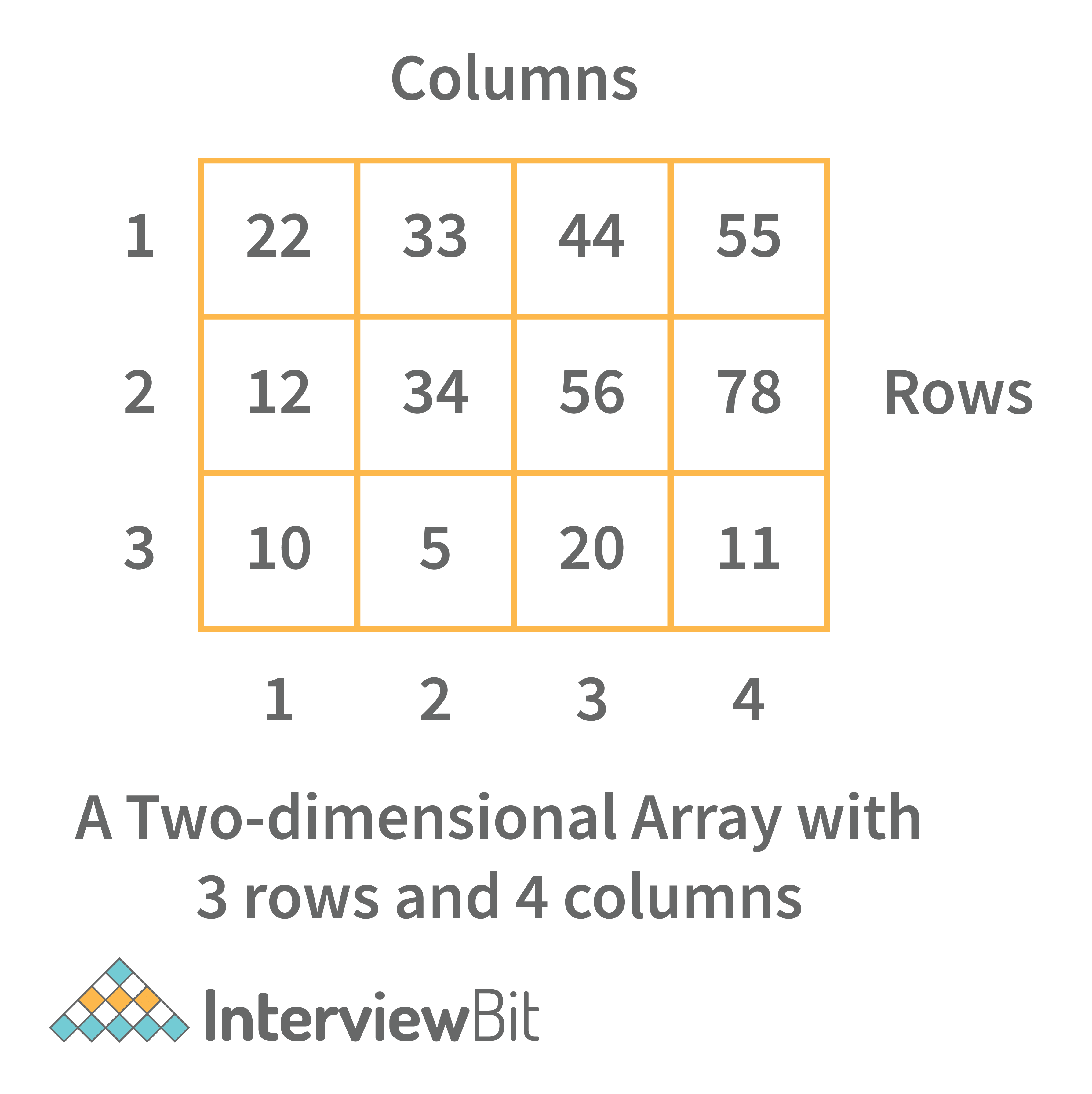
### **15. Elaborate on different types of array data structure**

There are several different types of arrays:

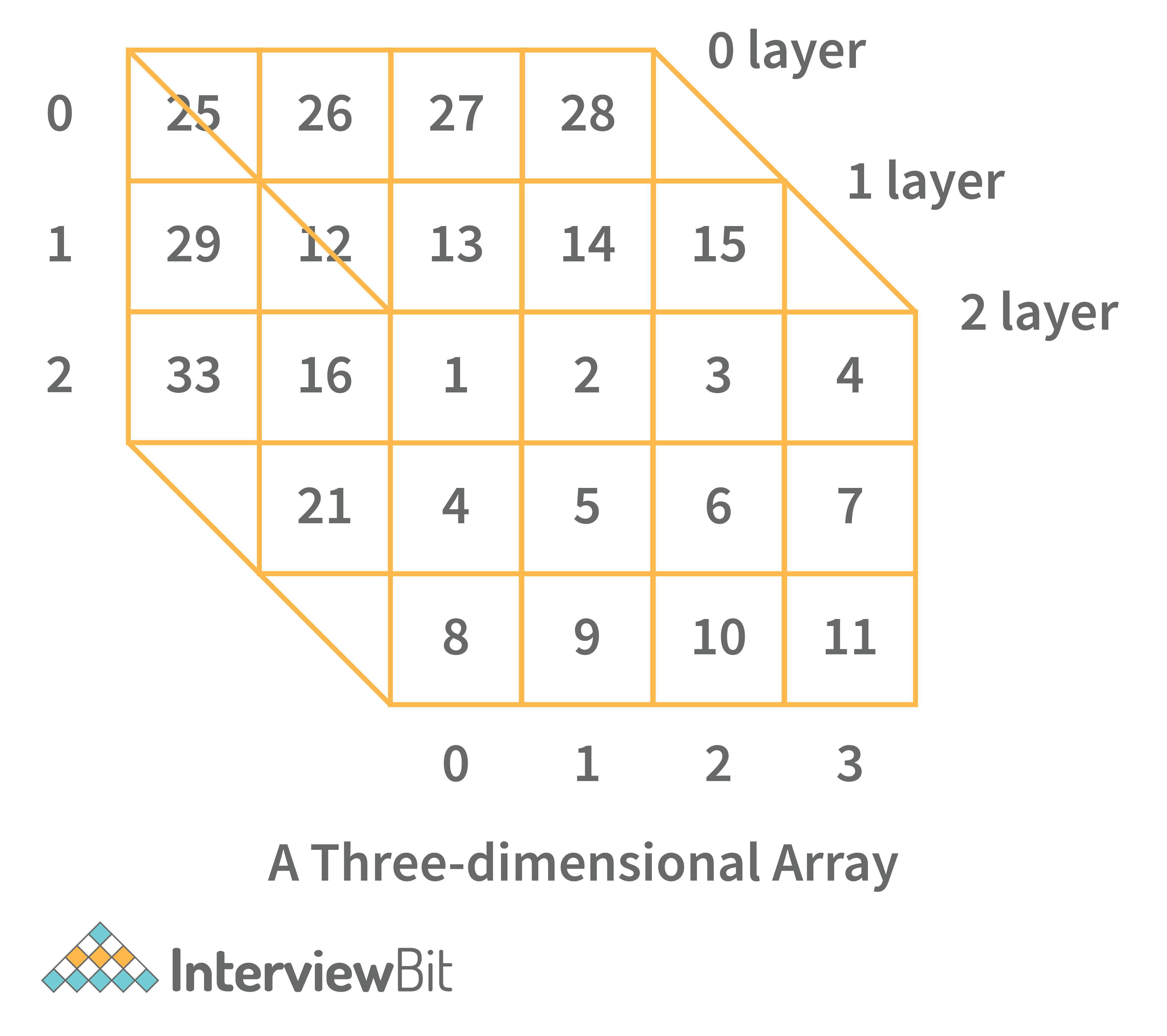
* ****One-dimensional array:**** A one-dimensional array stores its elements in contiguous memory locations, accessing them using a single index value. It is a linear data structure holding all the elements in a sequence.



* ****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.

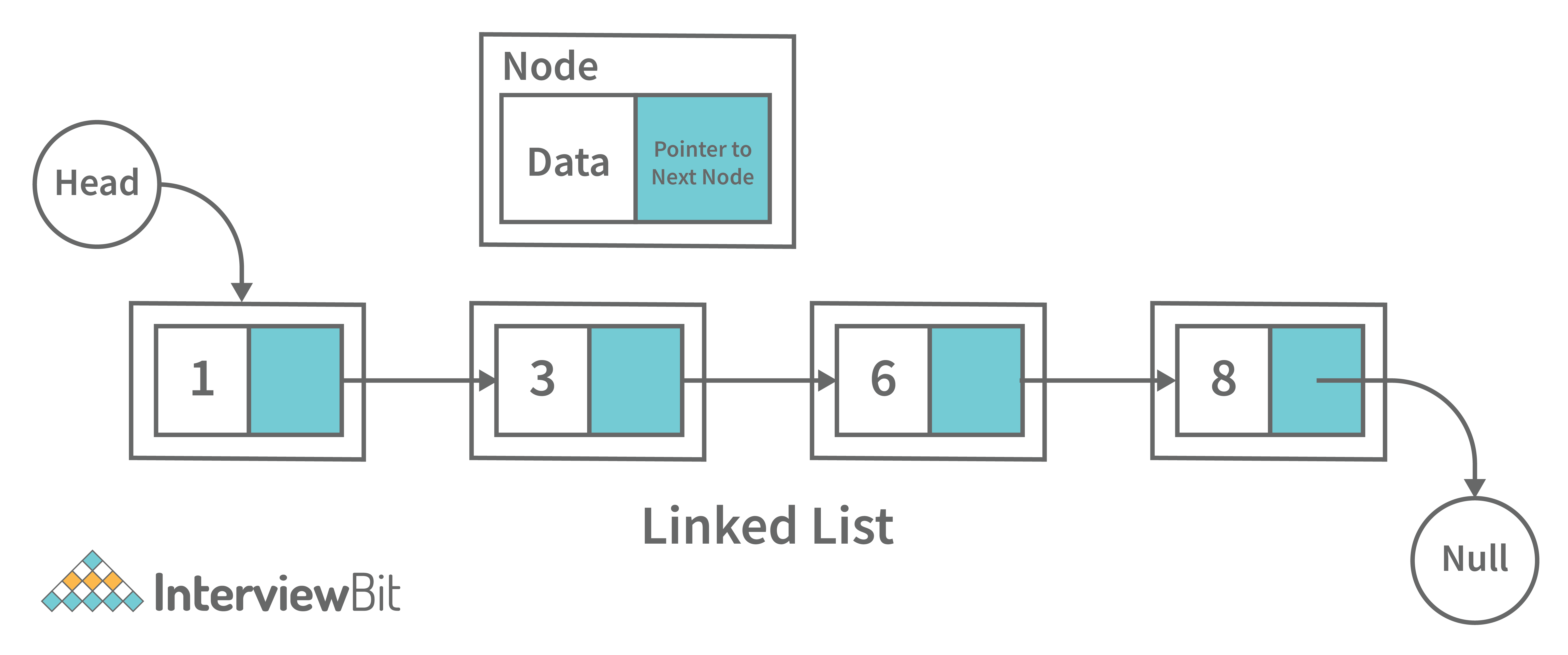


* ****Three-dimensional array:**** A three-dimensional array is a grid that has rows, columns, and depth as a third dimension. It comprises a cube with rows, columns, and depth as a third dimension. The three-dimensional array has three subscripts for a position in a particular row, column, and depth.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



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

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.



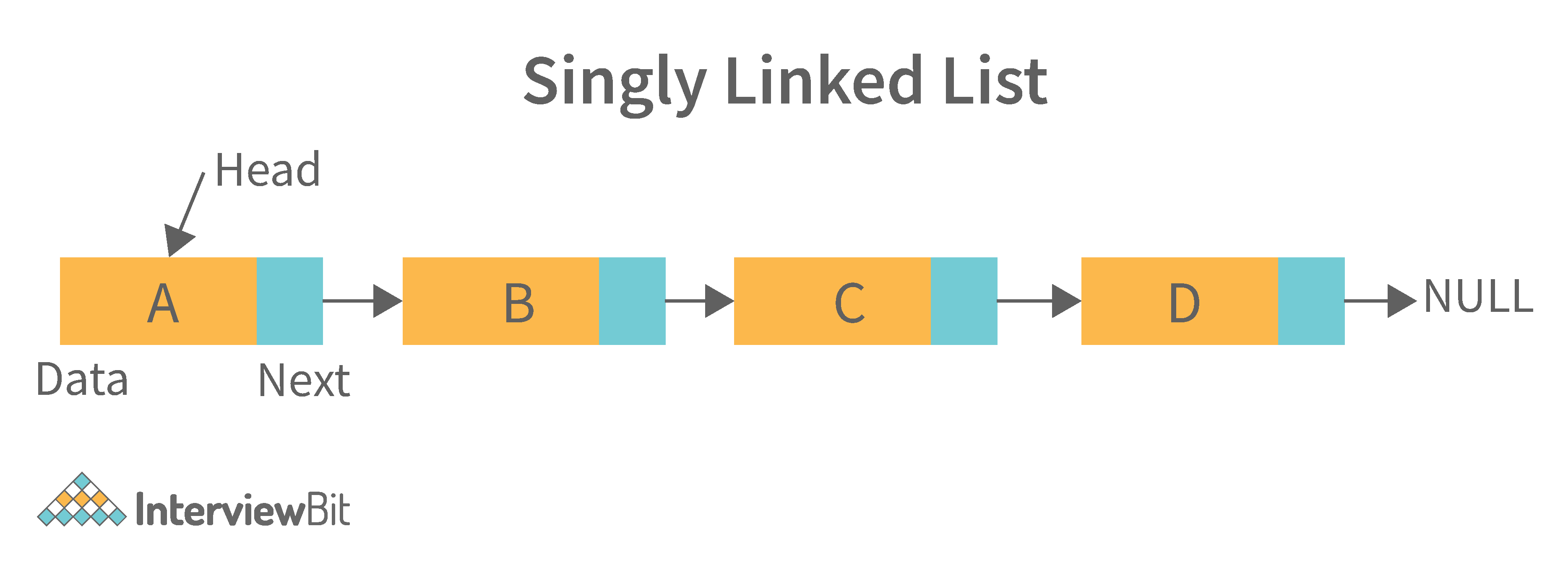
Following are some applications of linked list data structure:

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

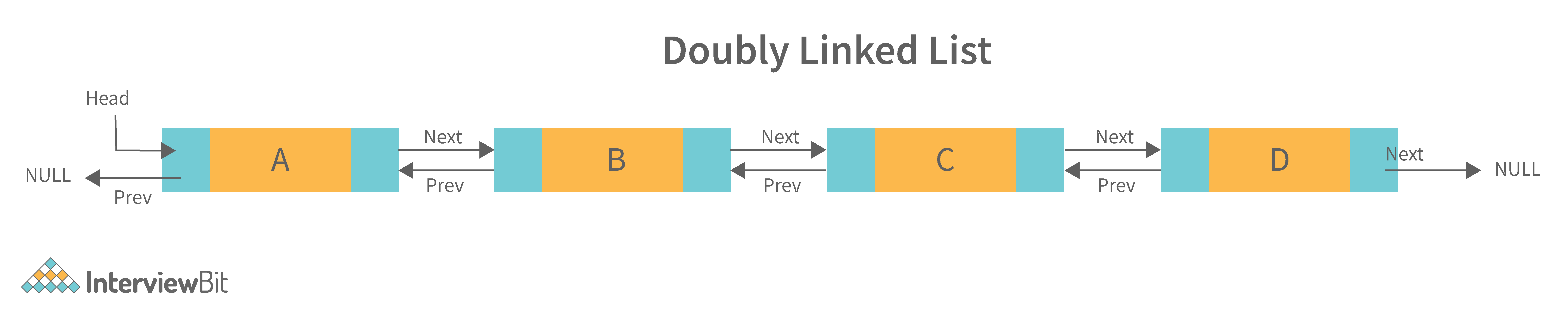
### **17. Elaborate on different types of Linked List data structures?**

Following are different types of linked lists:

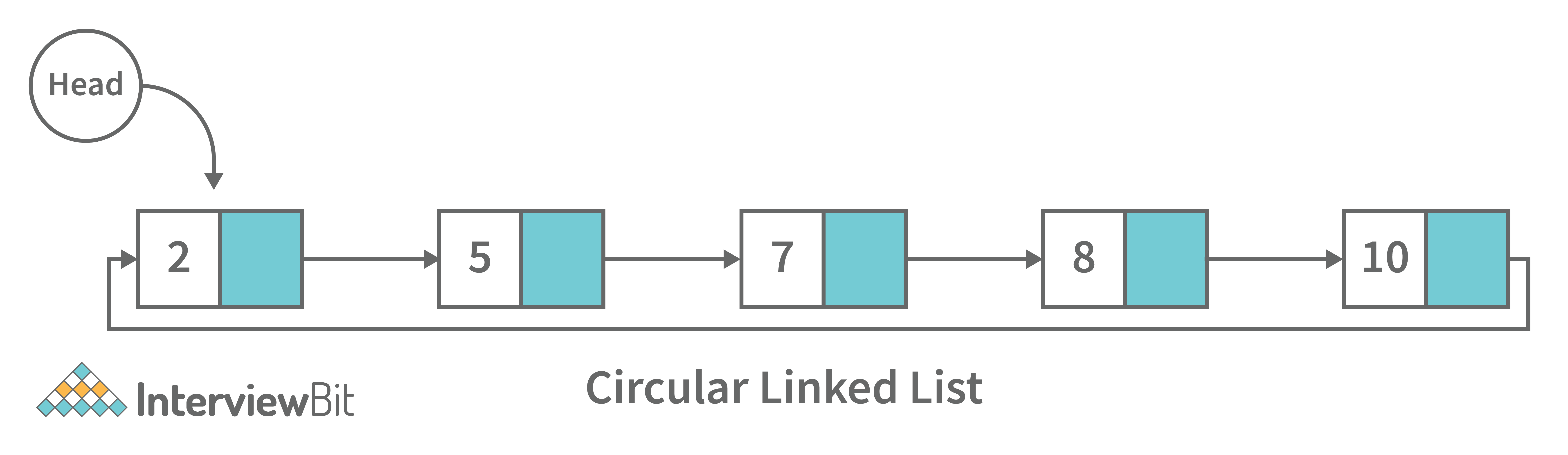
****1. Singly Linked List:****A singly linked list is a data structure that is used to store multiple items. The items are linked together using the key. The key is used to identify the item and is usually a unique identifier. In a singly linked list, each item is stored in a separate node. The node 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.



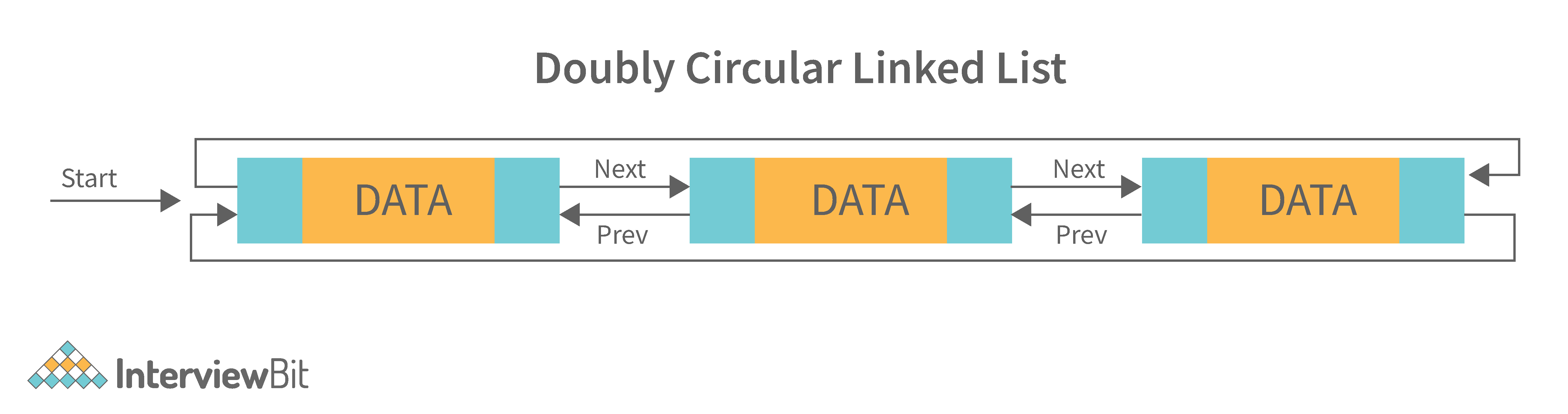
****2. Doubly Linked List:****A doubly linked list is a data structure that allows for two-way data access such that each node in the list points to the next node in the list and also points back to its previous node. In a doubly linked list, each node can be accessed by its address, and the contents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



****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.



****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.



****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.

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

| **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 fixed and cannot be changed during runtime. | The memory size of a linked list is allocated 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 an array. |
| Operations like insertion and deletion take longer time in an array. | Operations like insertion and deletion are faster in the linked list. |

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

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, Ο) performance of an algorithm.

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

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.

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

* 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.

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

* 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.

### **23. What is the time complexity of basic operations get() and put() in HashMap class?**

The time complexity is O(1) ****assuming**** that the hash function used in the hash map distributes elements uniformly among the buckets.

## **Data Structure Interview Questions for Experienced**

### **24. What is binary tree data structure? What are the applications for binary trees?**

A binary tree is a data structure that is used to organize data in a way that allows for efficient retrieval and manipulation. It is a data structure that uses two nodes, called leaves and nodes, to represent the data. The leaves represent the data and the nodes represent the relationships between the leaves. Each node has two children, called siblings, and each child has one parent. The parent is the node that is closest to the root of the tree. When a node is deleted from the tree, it is deleted from both its child and its parent.

Following are some applications for binary tree data structure:

* It's widely used in computer networks for storing routing table information.
* Decision Trees.
* Expression Evaluation.
* Database indices.

### **25. What is binary search tree data structure? What are the applications for binary search trees?**

A binary search tree is a data structure that stores items in sorted order. In a binary search tree, each node stores a key and a value. The key is used to access the item and the value is used to determine whether the item is present or not. The key can be any type of value such as an integer, floating point number, character string, or even a combination of these types. The value can be any type of items such as an integer, floating point number, character string, or even a combination of these types. When a node is added to the tree, its key is used to access the item stored at that node. When a node is removed from the tree, its key is used to access the item stored at that node.

A binary search tree is a special type of binary tree that has a specific order of elements in it. It has three basic qualities:

* All elements in the left subtree of a node should have a value less than or equal to the parent node's value, and
* All elements in the right subtree of a node should have a value greater than or equal to the parent node's value.
* Both the left and right subtrees must be binary search trees too.

Following are some applications for binary tree data structure:

* It is used for indexing and multi-level indexing.
* It is used for implementing various search algorithms.
* It is helpful in organizing a sorted stream of data.

### **26. What are tree traversals?**

Tree traversal is the process of visiting all the nodes of a tree. Since the root (head) is the first node and all nodes are connected via edges (or links) we always start with that node. There are three ways which we use to traverse a tree −

****1. Inorder Traversal:****

* Algorithm:
  + Step 1. Traverse the left subtree, i.e., call Inorder(root.left)
  + Step 2. Visit the root.
  + Step 3. Traverse the right subtree, i.e., call Inorder(root.right)
* Inorder traversal in Java:

// Print inorder traversal of given tree.

void printInorderTraversal(Node root)

{

if (root == null)

return;

//first traverse to the left subtree

printInorderTraversal(root.left);

//then print the data of node

System.out.print(root.data + " ");

//then traverse to the right subtree

printInorderTraversal(root.right);

}

* Uses: In binary search trees (BST), inorder traversal gives nodes in ascending order.

****2. Preorder Traversal:****

* Algorithm:
  + Step 1. Visit the root.
  + Step 2. Traverse the left subtree, i.e., call Preorder(root.left)
  + Step 3. Traverse the right subtree, i.e., call Preorder(root.right)
* Preorder traversal in Java:

// Print preorder traversal of given tree.

void printPreorderTraversal(Node root)

{

if (root == null)

return;

//first print the data of node

System.out.print(root.data + " ");

//then traverse to the left subtree

printPreorderTraversal(root.left);

//then traverse to the right subtree

printPreorderTraversal(root.right);

}

* Uses:
  + Preorder traversal is commonly used to create a copy of the tree.
  + It is also used to get prefix expression of an expression tree.

****3. Postorder Traversal:****

* Algorithm:
  + Step 1. Traverse the left subtree, i.e., call Postorder(root.left)
  + Step 2. Traverse the right subtree, i.e., call Postorder(root.right)
  + Step 3. Visit the root.
* Postorder traversal in Java:

// Print postorder traversal of given tree.

void printPostorderTraversal(Node root)

{

if (root == null)

return;

//first traverse to the left subtree

printPostorderTraversal(root.left);

//then traverse to the right subtree

printPostorderTraversal(root.right);

//then print the data of node

System.out.print(root.data + " ");

}

* Uses:
  + Postorder traversal is commonly used to delete the tree.
  + It is also useful to get the postfix expression of an expression tree.

Consider the following tree as an example, then:

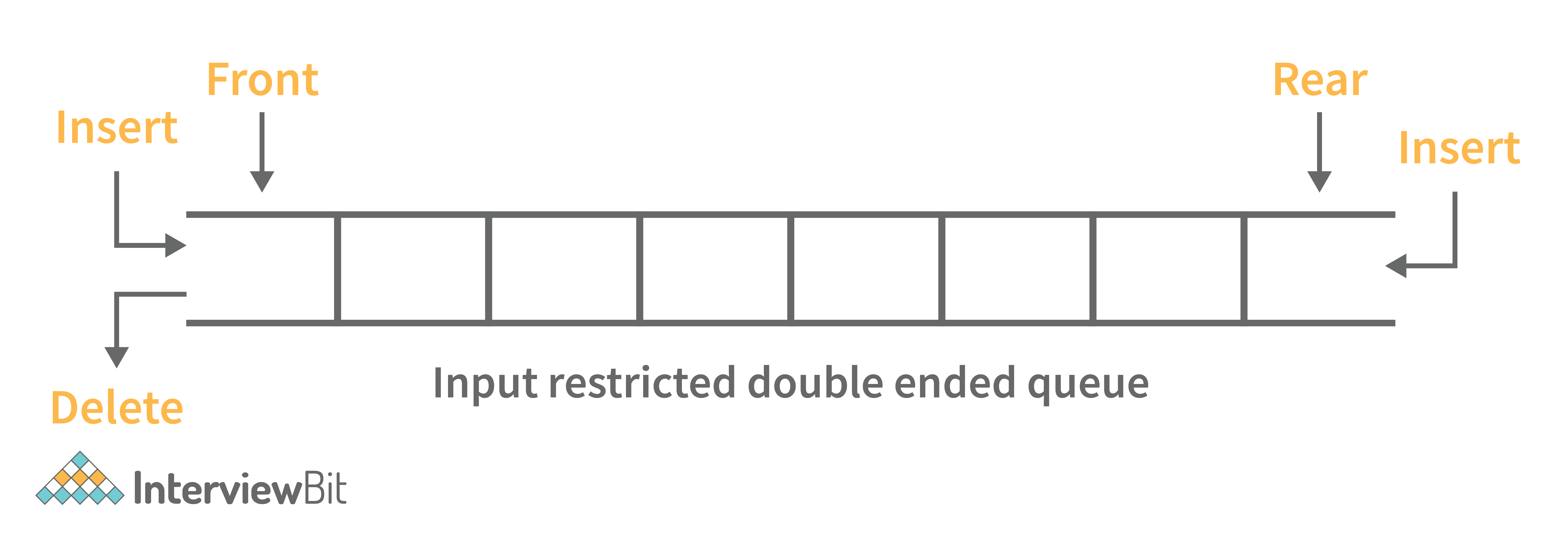
* Inorder Traversal => Left, Root, Right : [4, 2, 5, 1, 3]
* Preorder Traversal => Root, Left, Right : [1, 2, 4, 5, 3]
* Postorder Traversal => Left, Right, Root : [4, 5, 2, 3, 1]

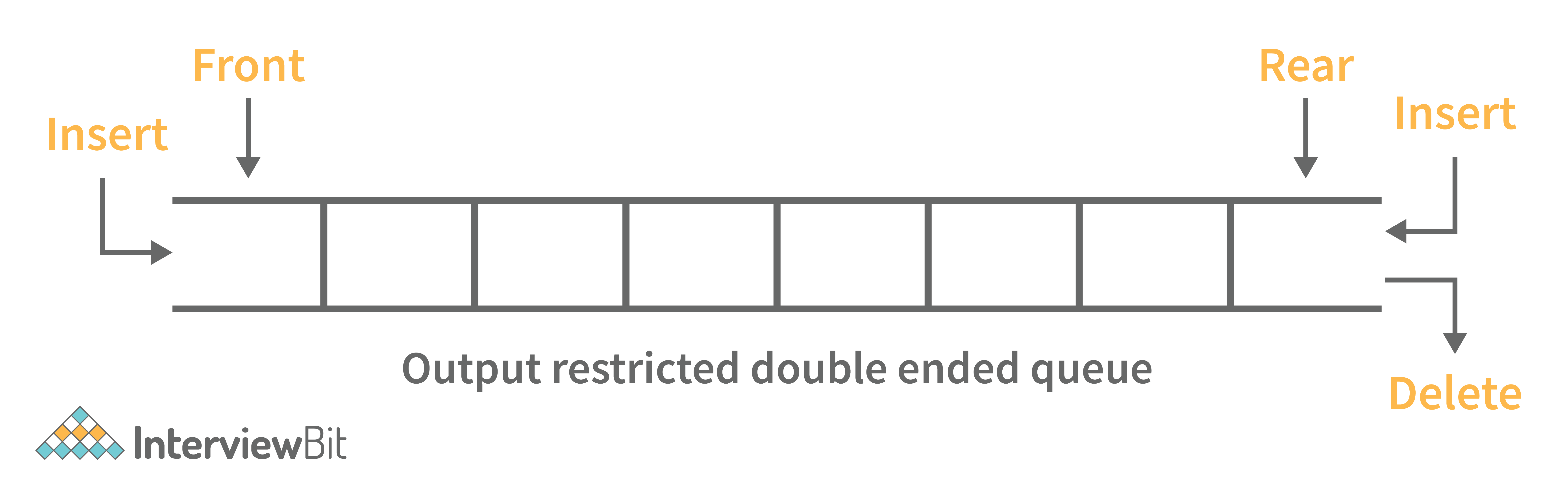
### **27. What is a deque data structure and its types? What are the applications for deque?**

A deque can be thought of as an array of items, but with one important difference: Instead of pushing and popping items off the end to make room, deques are designed to allow items to be inserted at either end. This property makes deques well-suited for performing tasks such as keeping track of inventory, scheduling tasks, or handling large amounts of data.

There are two types of deque:

* ****Input Restricted Deque:**** Insertion operations are performed at only one end while deletion is performed at both ends in the input restricted queue.



* ****Output Restricted Deque:****Deletion operations are performed at only one end while insertion is performed at both ends in the output restricted que

Following are some real-time applications for deque data structure:

* It can be used as both stack and queue, as it supports all the operations for both data structures.
* Web browser’s history can be stored in a deque.
* Operating systems job scheduling algorithm.

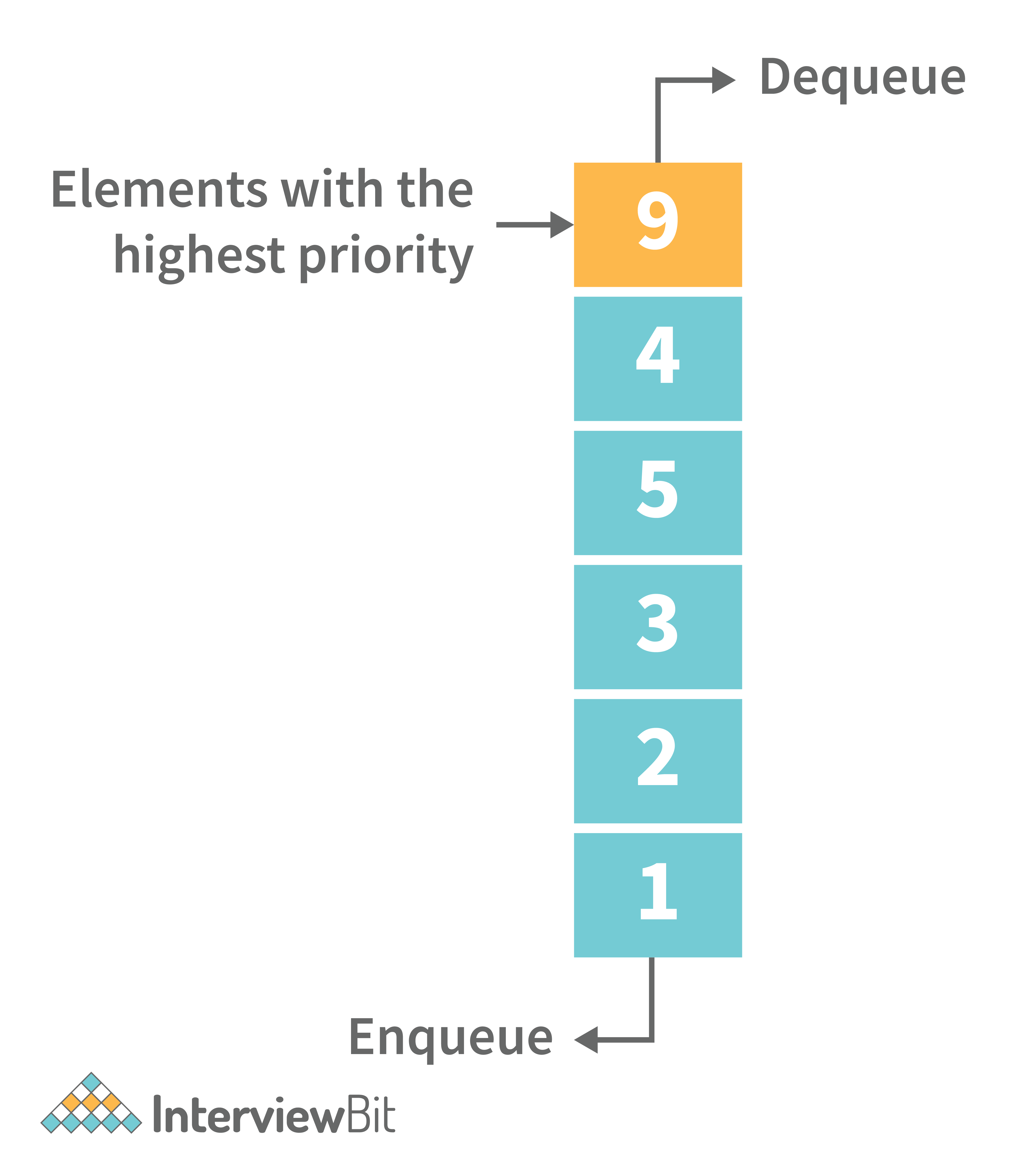
### **28. What are some key operations performed on the Deque data structure?**

Following are the key operations available deque:

* *insertFront()*: This adds an element to the front of the Deque.
* *insertLast()*: This adds an element to the rear of the Deque.
* *deleteFront()*: This deletes an element from the front of the Deque.
* *deleteLast()*:This deletes an element from the front of the Deque.
* *getFront()*: This gets an element from the front of the Deque.
* *getRear()*: This gets an element from the rear of the Deque.
* *isEmpty()*: This checks whether Deque is empty or not.
* *isFull()*: This checks whether Deque is full or not.

### **29. What is a priority queue? What are the applications for priority queue?**

Priority Queue is an abstract data type that is similar to a queue in that each element is assigned a priority value. The order in which elements in a priority queue are served is determined by their priority (i.e., the order in which they are removed). If the elements have the same priority, they are served in the order they appear in the queue.



Following are some real-time applications for priority queue:

* Used in graph algorithms like Dijkstra, Prim’s Minimum spanning tree etc.
* Huffman code for data compression
* Finding Kth Largest/Smallest element

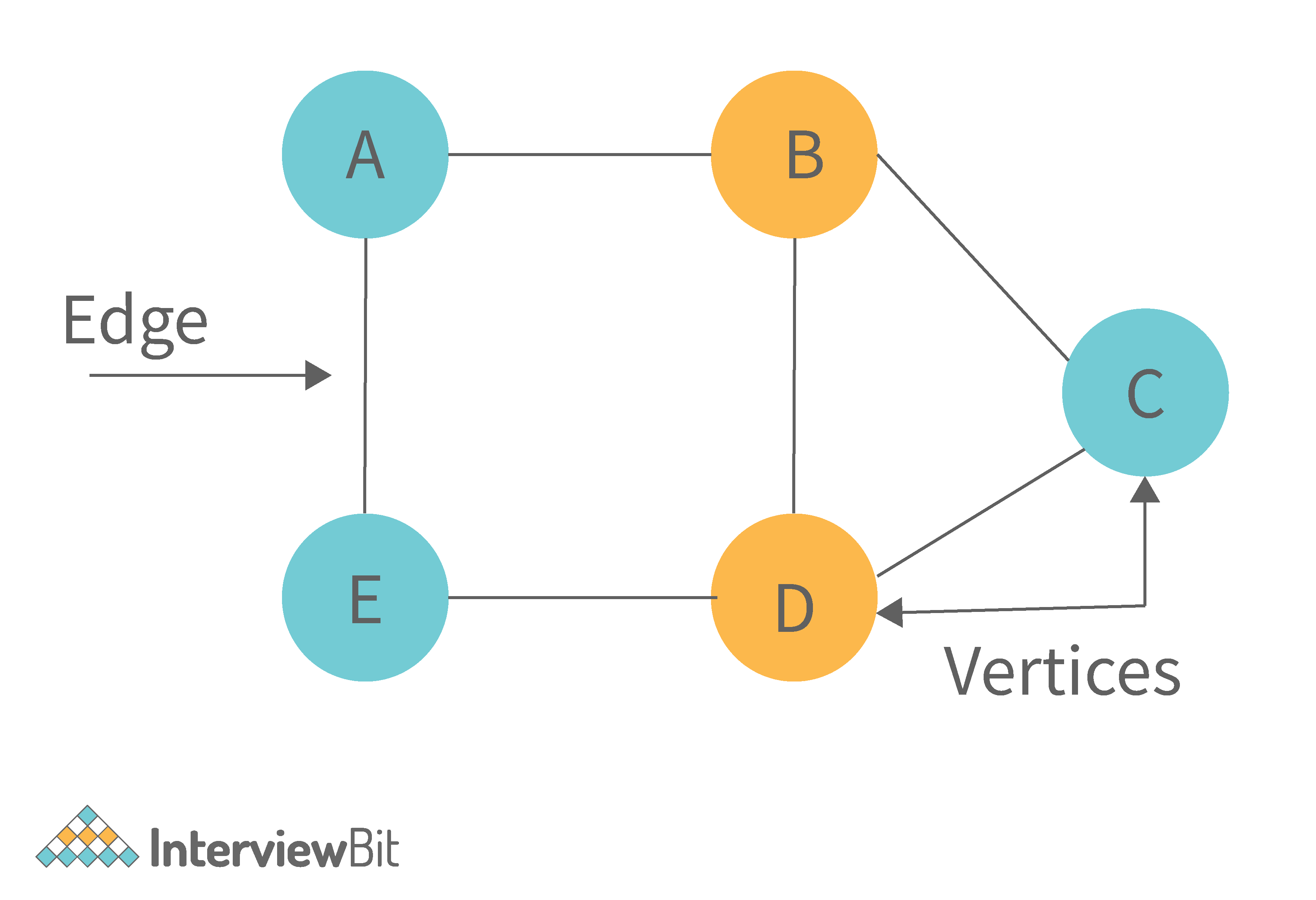
### **30. Compare different implementations of priority queue**

The following table contains an asymptotic analysis of different implementations of a priority queue:

| **Operations** | **peek** | **insert** | **delete** |
| --- | --- | --- | --- |
| **Linked List** | O(1) | O(n) | O(1) |
| **Binary Heap** | O(1) | O(log n) | O(log n) |
| **Binary Search Tree** | O(1) | O(log n) | O(log n) |

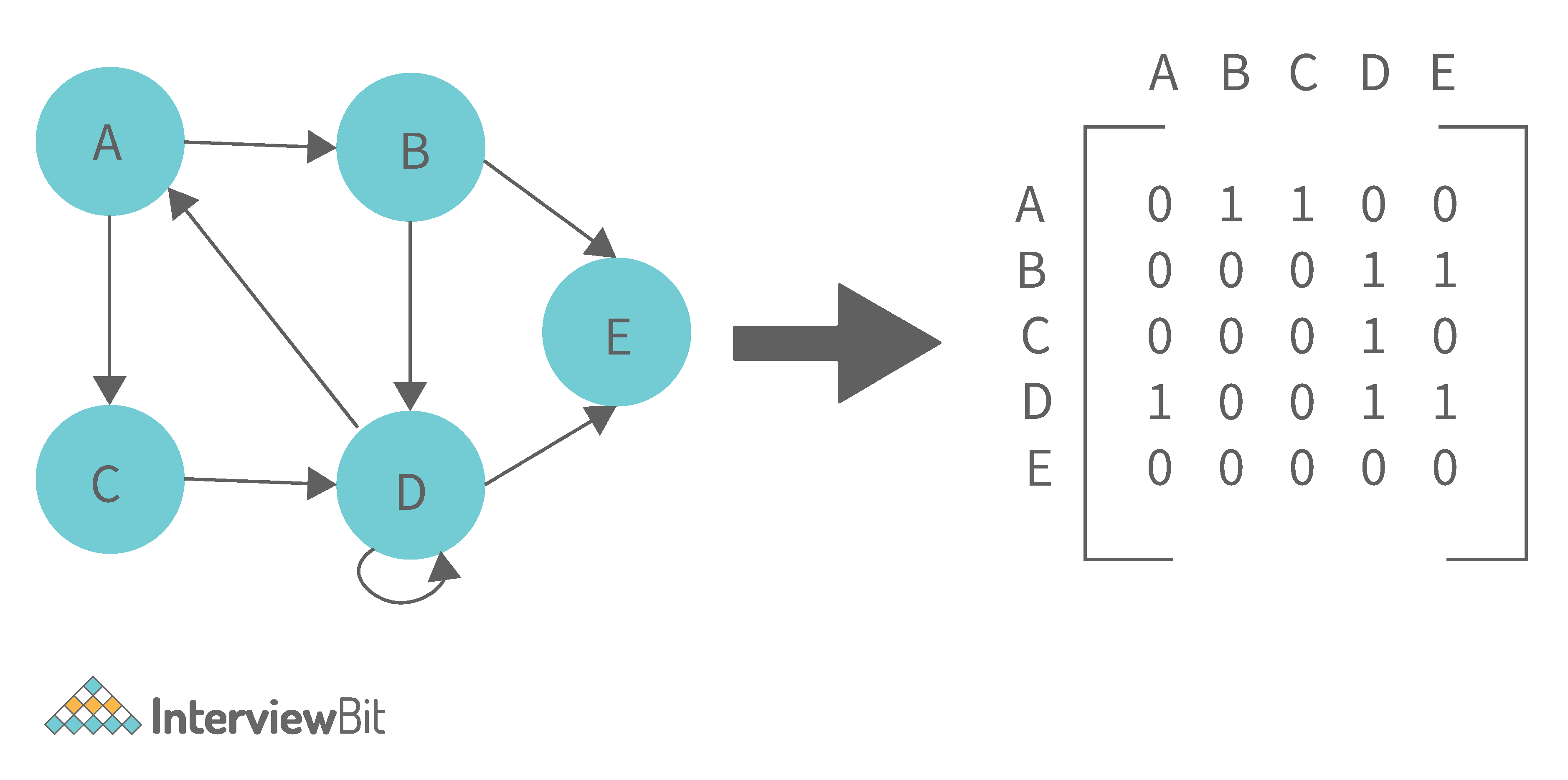
### **31. What is graph data structure and its representations? What are the applications for graphs?**

A graph is a type of non-linear data structure made up of nodes and edges. The nodes are also known as vertices, and edges are lines or arcs that connect any two nodes in the graph.



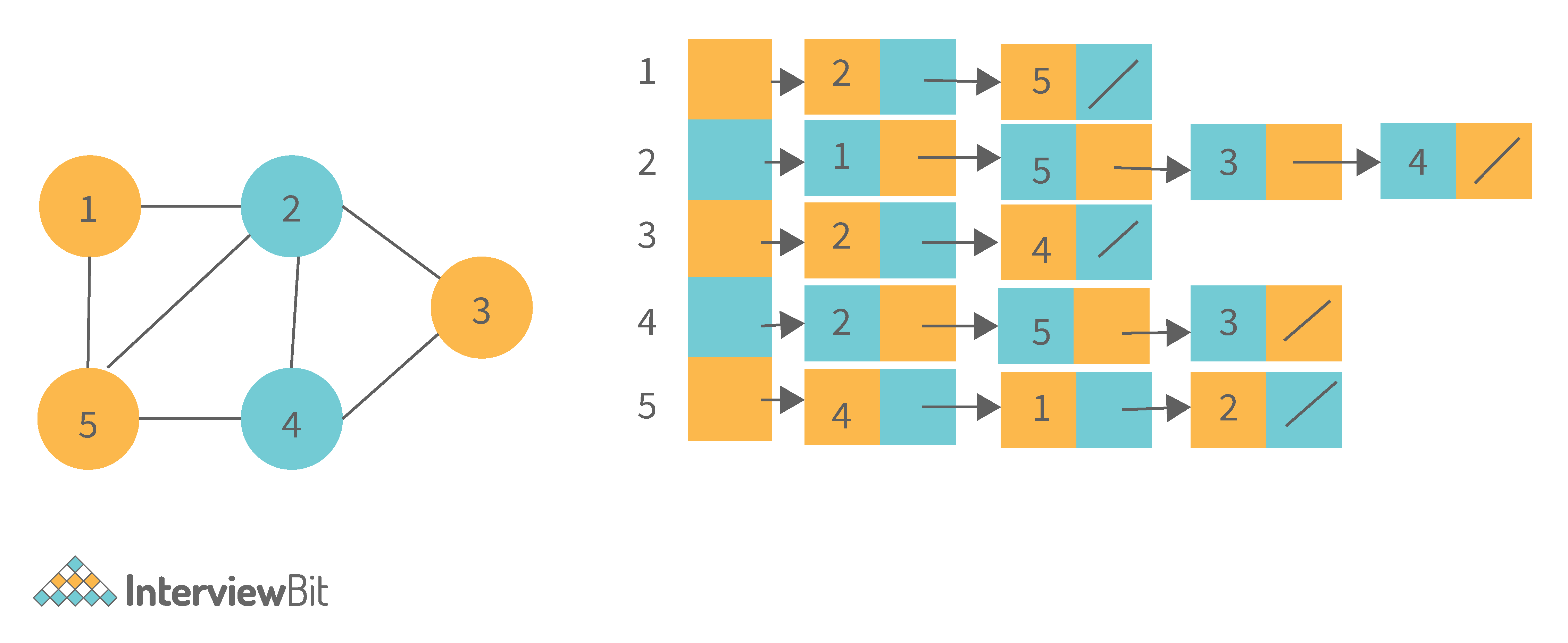
The following are the two most common graph representations:

****1. Adjacency Matrix****: Adjacency Matrix is a two-dimensional array with the dimensions V x V, where V is the number of vertices in a graph. Representation is simpler to implement and adhere to. It takes O(1) time to remove an edge. Queries such as whether there is an edge from vertex 'u' to vertex 'v' are efficient and can be completed in O(1).



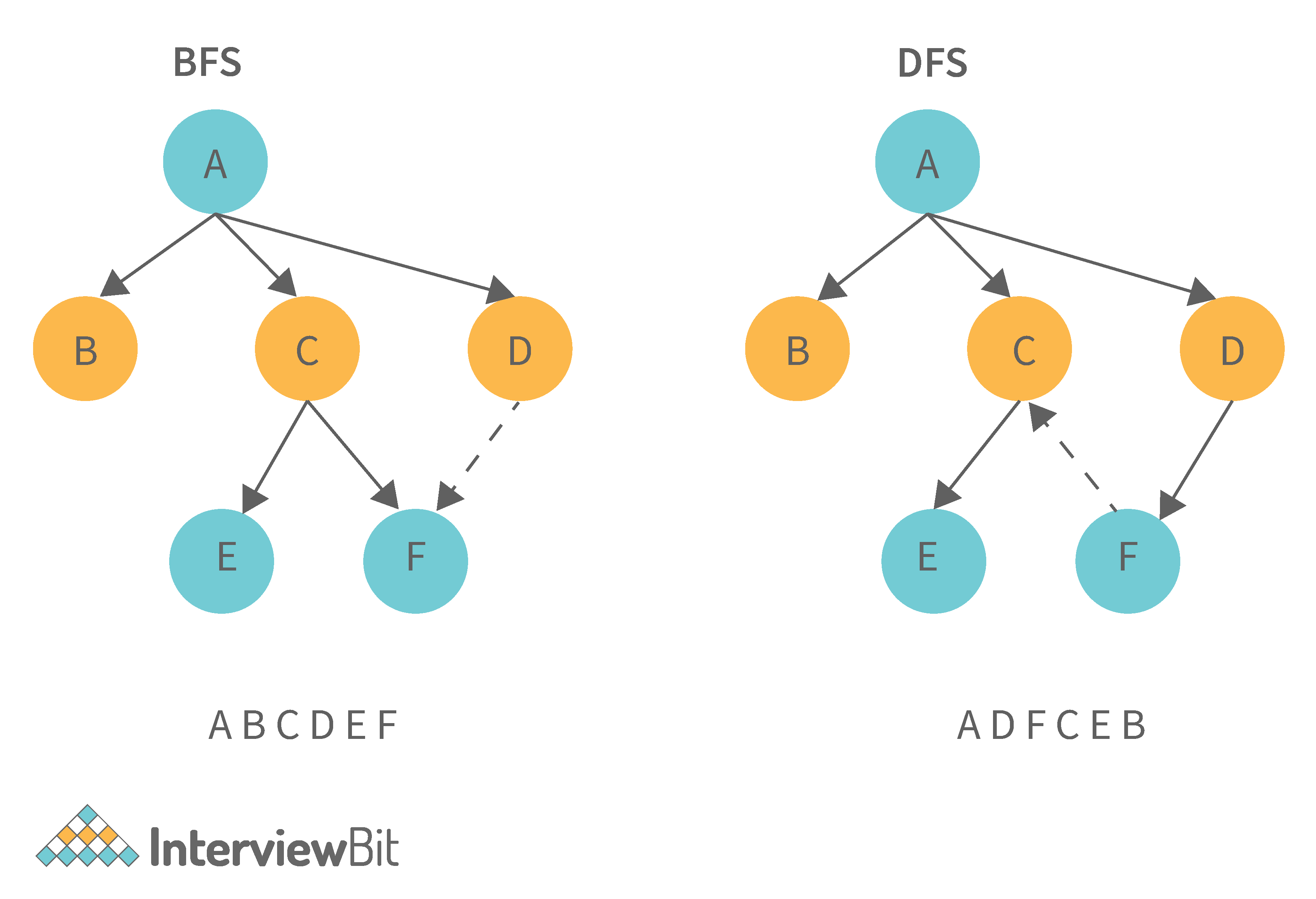
One of the cons of this representation is that even if the graph is sparse (has fewer edges), it takes up the same amount of space. Adding a vertex takes O(V^2). It also takes O(V) time to compute all of a vertex's neighbours, which is not very efficient.

****2. Adjacency List:**** In this method, each Node holds a list of Nodes that are directly connected to that vertex. Each node at the end of the list is connected with null values to indicate that it is the last node in the list. This saves space O(|V|+|E|). In the worst-case scenario, a graph can have C(V, 2) edges, consuming O(V^2) space. It is simpler to add a vertex. It takes the least amount of time to compute all of a vertex's neighbours.



One of the cons of this representation is that ​queries such as "is there an edge from vertex u to vertex v?" are inefficient and take O (V) in the worst case.

### **32. What is the difference between the Breadth First Search (BFS) and Depth First Search (DFS)?**



| **Breadth First Search (BFS)** | **Depth First Search (DFS)** |
| --- | --- |
| It stands for “Breadth First Search” | It stands for “Depth First Search” |
| BFS (Breadth First Search) finds the shortest path using the Queue data structure. | DFS (Depth First Search) finds the shortest path using the Stack data structure. |
| We walk through all nodes on the same level before passing to the next level in BFS. | DFS begins at the root node and proceeds as far as possible through the nodes until we reach the node with no unvisited nearby nodes. |
| When compared to DFS, BFS is slower. | When compared to BFS, DFS is faster. |
| BFS performs better when the target is close to the source. | DFS performs better when the target is far from the source. |
| BFS necessitates more memory. | DFS necessitates less memory. |
| Nodes that have been traversed multiple times are removed from the queue. | When there are no more nodes to visit, the visited nodes are added to the stack and then removed. |
| Backtracking is not an option in BFS. | The DFS algorithm is a recursive algorithm that employs the concept of backtracking. |
| It is based on the FIFO principle (First In First Out). | It is based on the LIFO principle (Last In First Out). |

### **33. What is AVL tree data structure, its operations, and its rotations? What are the applications for AVL trees?**

AVL trees are height balancing binary search trees named after their inventors Adelson, Velski, and Landis. The AVL tree compares the heights of the left and right subtrees and ensures that the difference is less than one. This distinction is known as the Balance Factor.

****BalanceFactor = height(left-subtree) − height(right-subtree)****

We can perform the following two operations on AVL tree:

* ****Insertion:**** Insertion in an AVL tree is done in the same way that it is done in a binary search tree. However, it may cause a violation in the AVL tree property, requiring the tree to be balanced. Rotations can be used to balance the tree.
* ****Deletion:**** Deletion can also be performed in the same manner as in a binary search tree. Because deletion can disrupt the tree's balance, various types of rotations are used to rebalance it.

An AVL tree can balance itself by performing the four rotations listed below:

* ****Left rotation:**** When a node is inserted into the right subtree of the right subtree and the tree becomes unbalanced, we perform a single left rotation.
* ****Right rotation:**** If a node is inserted in the left subtree of the left subtree, the AVL tree may become unbalanced. The tree then requires right rotation.
* ****Left-Right rotation:**** The RR rotation is performed first on the subtree, followed by the LL rotation on the entire tree.
* ****Right-Left rotation:**** The LL rotation is performed first on the subtree, followed by the RR rotation on the entire tree.

Following are some real-time applications for AVL tree data structure:

* AVL trees are typically used for in-memory sets and dictionaries.
* AVL trees are also widely used in database applications where there are fewer insertions and deletions but frequent data lookups are required.
* Apart from database applications, it is used in applications that require improved searching.

### **34. What is a B-tree data structure? What are the applications for B-trees?**

The B Tree is a type of m-way tree that is commonly used for disc access. A B-Tree with order m can only have m-1 keys and m children. One of the primary reasons for using a B tree is its ability to store a large number of keys in a single node as well as large key values while keeping the tree's height relatively small.

A B-tree of order 4 is shown below in the image:

Following are the key properties of a B-tree data structure:

* All of the leaves are at the same height.
* The term minimum degree 't' describes a B-Tree. The value of t is determined by the size of the disc block.
* Except for root, every node must have at least t-1 keys. The root must contain at least one key.
* All nodes (including root) can have no more than 2\*t - 1 keys.
* The number of children of a node is equal to its key count plus one.
* A node's keys are sorted in ascending order. The child of two keys k1 and k2 contains all keys between k1 and k2.
* In contrast to Binary Search Tree, B-Tree grows and shrinks from the root.

Following are real-time applications of a B-Tree data structure:

* It is used to access data stored on discs in large databases.
* Using a B tree, you can search for data in a data set in significantly less time.
* The indexing feature allows for multilevel indexing.
* The B-tree approach is also used by the majority of servers.

### **35. Define Segment Tree data structure and its applications.**

A segment Tree is a binary tree that is used to store intervals or segments. The Segment Tree is made up of nodes that represent intervals. Segment Tree is used when there are multiple range queries on an array and changes to array elements.

The segment tree of array A[7] will look like this:

Following are key operations performed on the Segment tree data structure:

* Building Tree: In this step, we create the structure and initialize the segment tree variable.
* Updating the Tree: In this step, we change the tree by updating the array value at a point or over an interval.
* Querying Tree: This operation can be used to run a range query on the array.

Following are real-time applications for Segment Tree:

* Used to efficiently list all pairs of intersecting rectangles from a list of rectangles in the plane.
* The segment tree has become popular for use in pattern recognition and image processing.
* Finding range sum/product, range max/min, prefix sum/product, etc
* Computational geometry
* Geographic information systems
* Static and Dynamic RMQ (Range Minimum Query)
* Storing segments in an arbitrary manner

### **36. Define Trie data structure and its applications**

The word "Trie" is an abbreviation for "retrieval." Trie is a data structure that stores a set of strings as a sorted tree. Each node has the same number of pointers as the number of alphabet characters. It can look up a word in the dictionary by using its prefix. Assuming that all strings are formed from the letters 'a' to 'z' in the English alphabet, each trie node can have a maximum of 26 points.

Trie is also referred to as the digital tree or the prefix tree. The key to which a node is connected is determined by its position in the Trie. Trie allows us to insert and find strings in O(L) time, where L is the length of a single word. This is clearly faster than BST. Because of how it is implemented, this is also faster than Hashing. There is no need to compute a hash function. There is no need to handle collisions (like we do in open addressing and separate chaining)

Another benefit of Trie is that we can easily print all words in alphabetical order, which is not easy with hashing. Trie can also perform prefix search (or auto-complete) efficiently.

The main disadvantage of tries is that they require a large amount of memory to store the strings. We have an excessive number of node pointers for each node

Following are some real-time applications for Trie data structure:

* Auto-Complete and Search for Search Engines
* Genome Analysis
* Data Analytics
* Browser History
* Spell Checker

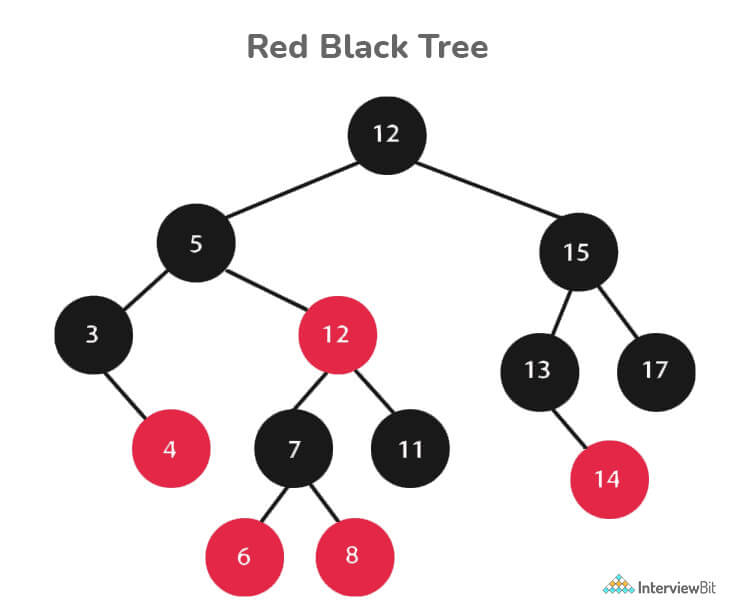
### **37. Define Red-Black Tree and its applications**

Red Black Trees are a type of self-balancing binary search tree. Rudolf Bayer invented it in 1972 and dubbed it "symmetric binary B-trees."

A red-black tree is a Binary tree in which each node has a colour attribute, either red or black. By comparing the node colours on any simple path from the root to a leaf, red-black trees ensure that no path is more than twice as long as any other, ensuring that the tree is generally balanced.

Red-black trees are similar to binary trees in that they both store their data in two's complementary binary formats. However, red-black trees have one important advantage over binary trees: they are faster to access. Because red-black trees are so fast to access, they are often used to store large amounts of data.

Red-black trees can be used to store any type of data that can be represented as a set of values.



Every Red-Black Tree Obeys the Following Rules:

* Every node is either red or black.
* The tree's root is always black.
* There are no two red nodes that are adjacent.
* There is the same number of black nodes on every path from a node to any of its descendant's NULL nodes.
* All of the leaf nodes are black.

Following are some real-time applications for the Red-Black Tree data structure:

* The majority of self-balancing BST library functions in C++ or Java use Red-Black Trees.
* It is used to implement Linux CPU Scheduling.
* It is also used to reduce time complexity in the K-mean clustering algorithm in machine learning.
* MySQL also employs the Red-Black tree for table indexes in order to reduce searching and insertion time.

### **38. Which data structures are used for implementing LRU cache?**

LRU cache or Least Recently Used cache allows quick identification of an element that hasn’t been put to use for the longest time by organizing items in order of use. In order to achieve this, two data structures are used:

* ****Queue**** – This is implemented using a doubly-linked list. The maximum size of the queue is determined by the cache size, i.e by the total number of available frames. The least recently used pages will be near the front end of the queue whereas the most recently used pages will be towards the rear end of the queue.
* ****Hashmap**** – Hashmap stores the page number as the key along with the address of the corresponding queue node as the value.

### **39. What is a heap data structure?**

Heap is a special tree-based non-linear data structure in which the tree is a complete binary tree. A binary tree is said to be complete if all levels are completely filled except possibly the last level and the last level has all elements as left as possible. Heaps are of two types:

* ****Max-Heap:****
  + In a Max-Heap the data element present at the root node must be the greatest among all the data elements present in the tree.
  + This property should be recursively true for all sub-trees of that binary tree.
* ****Min-Heap:****
  + In a Min-Heap the data element present at the root node must be the smallest (or minimum) among all the data elements present in the tree.
  + This property should be recursively true for all sub-trees of that binary tree.

## **Data Structure Coding Interview Questions**

### **40. Write a program to remove duplicates from a sorted array in place?**

* ****Input****: {1, 1, 1, 2, 3, 3, 6, 6, 7}
* ****Output****: {1, 2, 3, 6, 7}
* ****Explanation****: The given input has only 1,2,3,6, and 7 as unique elements, hence the output only lists them out.

#**include** <bits/stdc++.h>**using** **namespace** std;

**class** **Solution**{**public**:

//function that takes an array and its size as arguments

**int** **removeDuplicates**(**int** a[],**int** n){

**int** index=0;

**for**(**int** i=1;i<n;i++) {

**if**(a[i]!=a[index]) { //change index

index++; //swap next line

a[index]=a[i];

}

}

**return** index+1;

}

};

**int** **main**(){

**int** T;

//taking the number of test cases from user

cin>>T;

//running the loop for all test cases

**while**(T--)

{

**int** N;

//taking size input from user

cin>>N;

**int** a[N];

//taking array input from user

**for**(**int** i=0;i<N;i++)

{

cin>>a[i];

}

Solution ob;

//calling the removeDuplicates in the Solution class

**int** n = ob.removeDuplicates(a,N);

//printing the array after removing duplicates

**for**(**int** i=0;i<n;i++)

cout<<a[i]<<" ";

cout<<endl;

}

}

* ****Time Complexity****: O(n)
* ****Space Complexity:**** O(1)

### **41. Write a function for zigzag traversal in a binary tree**

* ****Input:****
* ****Output:****[1, 3, 2, 4, 5, 6, 8, 7]
* ****Explanation:****Zigzag Traversal first iterates the given level of the tree from left to right and then the next level as the right to the level.

// Tree Node

struct Node {

int data;

Node\* left;

Node\* right;

};

//Function to store the zigzag order traversal of a tree in a list.

vector <int> zigZagTraversal(Node\* root)

{

//creating two stacks for level traversals in both order

stack<Node\*> st1;

stack<Node\*> st2;

//vector to store the zigzag traversal

vector<int> result;

//Initialize the first stack with the root element

st1.push(root);

//Iterate until either of the stack is not empty

while(!st1.empty() || !st2.empty()){

//iterate until the first stack is not empty

while(!st1.empty()){

Node\* temp=st1.top();

st1.pop();

result.push\_back(temp->data);

if(temp->left)

st2.push(temp->left);

if(temp->right)

st2.push(temp->right);

}

//Iterate until the second stack is not empty

while(!st2.empty()){

Node\* temp=st2.top();

st2.pop();

result.push\_back(temp->data);

if(temp->right)

st1.push(temp->right);

if(temp->left)

st1.push(temp->left);

}

}

return result;

}

* ****Time Complexity****: O(n)
* ****Space Complexity:**** O(n)

### **42. Write a function to sort a linked list of 0s, 1s and 2s**

* ****Input:**** 0->1->0->2->1->0->2->1
* ****Output:****0->0->0->1->1->1->2->2
* ****Explanation****: All 0’s will come first then 1s and then 2s. This can be done in O(n) time by counting the occurrences of all three and rearranging them in the linked list.

//structure of the linked list

struct Node {

int data;

Node \*left;

Node \*right;

}

//function take the head of the linked list as a parameter

void sortList(Node \*head)

{

//if linked list is empty then return back

if(head==NULL)

return;

else

{

Node \*temp=head;

Node \*temp1=head;

//to store count of 0s, 1s, and 2s

int count0=0,count1=0,count2=0;

//calculating the count of 0s, 1s, and 2s

while(temp!=NULL)

{

if(temp->data==0)

count0++;

else if(temp->data==1)

count1++;

else

count2++;

temp=temp->next;

}

//iterating over count of 0s and filling the linked list

while(count0!=0)

{

temp1->data=0;

temp1=temp1->next;

count0--;

}

//iterating over count of 1s and filling the linked list

while(count1!=0)

{

temp1->data=1;

temp1=temp1->next;

count1--;

}

//iterating over count of 2s and filling the linked list

while(count2!=0)

{

temp1->data=2;

temp1=temp1->next;

count2--;

}

}

}

* ****Time Complexity****: O(n)
* ****Space Complexity:**** O(1)

### **43. Write a function to detect cycle in an undirected graph**

* *****Input:*****n = 4, e = 4 , 0 1, 1 2, 2 3, 3 1
* *****Output:*****Yes
* ****Explanation****: The graph is represented as follows in adjacency list representation:  
  0->1  
  1->2  
  2->3  
  3->1

From the above representation, we can see that there exists a cycle: 1→2→3→1

//function to run dfs for a given node in the graph

int dfs(int v,vector<int> adj[],vector<int> &visited,vector<int> &rec,int i,int parent){

int ans=0;

visited[i]=1;

rec[i]=1;

for(auto x : adj[i]){

if(x!=parent) {

if(rec[x])

return 1;

ans=dfs(v,adj,visited,rec,x,i);

if(ans)

return 1;

}

}

rec[i]=0;

return 0;

}

// Function to detect cycle in an undirected graph.

// it takes adjacency list representation as an argument

bool isCycle(int v, vector<int> adj[]) {

vector<int> visited(v,0),rec(v,0);

int ans=0;

for(int i=0;i<v;i++){

if(visited[i]==0)

ans=dfs(v,adj,visited,rec,i,-1);

if(ans)

return 1;

}

return 0;

}

* ****Time Complexity:****O(V+E)
* ****Space Complexity:****O(V)

### **44. Write a function to convert an infix expression to postfix expression**

* ****Input:****a+b\*(c^d)
* ****Output:****abcd^\*+

int prec(char c)

{

if (c == '^')

return 3;

else if (c == '/' || c == '\*')

return 2;

else if (c == '+' || c == '-')

return 1;

else

return -1;

}

public:

// Function to convert an infix expression to a postfix expression.

string infixToPostfix(string s) {

stack<char> st; // For stack operations, we are using C++ built in stack

string result;

for (int i = 0; i < s.length(); i++) {

char c = s[i];

// If the scanned character is

// an operand, add it to the output string.

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')

|| (c >= '0' && c <= '9'))

result += c;

// If the scanned character is an

// '(', push it to the stack.

else if (c == '(')

st.push('(');

// If the scanned character is an ')',

// pop and to output string from the stack

// until an '(' is encountered.

else if (c == ')') {

while (st.top() != '(') {

result += st.top();

st.pop();

}

st.pop();

}

// If an operator is scanned

else {

while (!st.empty()

&& prec(s[i]) <= prec(st.top())) {

if (c == '^' && st.top() == '^')

break;

else {

result += st.top();

st.pop();

}

}

st.push(c);

}

}

// Pop all the remaining elements from the stack

while (!st.empty()) {

result += st.top();

st.pop();

}

return result;

}

* ****Time Complexity:**** O(n)
* ****Space Complexity:**** O(n)

### **45. Write a function to find the maximum for each and every contiguous subarray of size k.**

* ****Input:****N = 9, K = 3 arr[] = {1, 2, 3, 1, 4, 5, 2, 3, 6}
* ****Output:**** {3, 3, 4, 5, 5, 5, 6}
* ****Explanation****: In the first subarray of size 3: {1,2,3}, the value 3 is maximum, similarly for all such subarrays for size 3.

//function to find maximum in each subarray using sliding window approach

vector<int> max\_of\_subarrays(vector<int> arr, int n, int k){

int i=0,j=0;

deque<int> dq;

dq.push\_front(i++);

while(i<k)

{

while(!dq.empty()&&arr[dq.back()]<=arr[i])

dq.pop\_back();

dq.push\_back(i++);

}

vector<int> ans;

while(i<n)

{

ans.push\_back(arr[dq.front()]);

while(!dq.empty()&&j>=dq.front())

{

dq.pop\_front();

}

j++;

while(!dq.empty()&&arr[dq.back()]<=arr[i])

dq.pop\_back();

dq.push\_back(i++);

}

ans.push\_back(arr[dq.front()]);

return ans;

}

* ****Time Complexity:**** O(n)
* ****Space Complexity:**** O(k)

### **46. Write a function to merge two sorted binary search tree**

****Input:****

First BST

       7

    /     \

   5       9

Second BST

    4

  /   \

3       12

****Output:****3 4 5 6 7 9 12

//Function to return a list of integers denoting the node

//values of both the BST in a sorted order.

void inorder(Node\*root,vector<int>&v){

if(root==NULL)

return;

inorder(root->left,v);

v.push\_back(root->data);

inorder(root->right,v);

}

vector<int> merge(vector<int>v1,vector<int>v2){

vector<int>v;

int n1=v1.size(),n2=v2.size(),i=0,j=0;

while(i<n1&&j<n2){

if(v1[i]>v2[j]){

v.push\_back(v2[j]);

j++;

}

else{

v.push\_back(v1[i]);

i++;

}

}

while(i<n1){

v.push\_back(v1[i]);

i++;

}

while(j<n2){

v.push\_back(v2[j]);

j++;

}

return v;

}

vector<int> merge(Node \*root1, Node \*root2)

{

vector<int>v1,v2;

inorder(root1,v1);

inorder(root2,v2);

return merge(v1,v2);

}

* ****Time Complexity:**** *O(m+n)*
* ****Space Complexity:**** *O(height of the first tree + height of the second tree)*

### **47. Write a function to print all unique rows of the given matrix.**

****Input:****

 {{1, 1, 1, 0, 0},

      {0, 1, 0, 0, 1},

      {1, 0, 1, 1, 0},

      {0, 1, 0, 0, 1},

      {1, 1, 1, 0, 0}}

****Output:****

    {{1, 1, 1, 0, 0},

     {0, 1, 0, 0, 1},

     {1, 0, 1, 1, 0}}

vector<vector<int>> uniqueRow(int M[MAX][MAX],int row,int col)

{

set<vector<int>> st;

vector<vector<int>> v;

for(int i=0; i<row; i++) {

vector<int> v1;

for(int j=0; j<col; j++) {

v1.push\_back(M[i][j]);

}

if(st.count(v1) == 0) {

v.push\_back(v1);

st.insert(v1);

}

}

return v;

}

* ****Time Complexity:**** O( ROW x COL )
* ****Space Complexity:**** O( ROW )

### **48. Write a function to find number of subarrays with product less than K**

* ****Input:****arr = [1, 6, 2, 3, 2, 1], k = 12
* ****Output:****11

int numSubarrayProductLessThanK(vector<int>& nums, int k) {

int ans=0;

int pdt=1;

int left=0,right=0;

while(right<=nums.size()-1){

pdt\*=nums[right];

while(pdt>=k and left<nums.size()){

pdt/=nums[left];

left++;

}

if(right-left>=0)

ans+=right-left+1;//since on adding a new element new subarrays formed is r-i+1;

right++;

}

return ans;

}

* ****Time Complexity:**** O(n)
* ****Space Complexity:**** O(1)

### **49. Find the subsequence of length 3 with the highest product from a sequence of non-negative integers, with the elements in increasing order.**

* ****Input:****n = 8 arr[ ] = {6, 7, 10, 1, 2, 3, 11, 12}
* ****Output:**** {10, 11, 12}

The three increasing elements of the given arrays are 10, 11, and 12, which form a three-size subsequence with the highest product.

vector<int> maxProductSubsequence(int \*a , int n)

{

set<int> s;

long long largestOnLeft[n];

for(int i=0;i<n;i++)

{

s.insert(a[i]);

auto it=s.lower\_bound(a[i]);

if(it==s.begin())

{

largestOnLeft[i]=-1;

continue;

}

it--;

largestOnLeft[i]=\*it;

}

int m=0;

long long p=INT\_MIN;

vector<int> result(3);

result[0]=-1;

for(int i=n-1;i>=0;i--)

{

if(a[i]>=m){

m=a[i];}

else

{

if(largestOnLeft[i] !=-1)

{

if(largestOnLeft[i]\*a[i]\*m >p)

{

p=largestOnLeft[i]\*a[i]\*m;

result[0]=largestOnLeft[i];

result[1]=a[i];

result[2]=m;

}

}

}

}

return v;

}

* ****Time Complexity:**** O(nlog(n))
* ****Space Complexity:****O(n)

### **50. Write a function to implement Quicksort on Doubly Linked List**

* ****Input:****8<->10<->1<->7<->6
* ****Output:****1<->6<->7<->8<->10

class Solution{

public:

Node\* partition(Node \*l, Node \*h){

//Your code goes here

Node\*temp = h;

Node\*tt = l;

Node\*first = l;

while(tt != h){

if(tt->data <= temp->data){

swap(first->data, tt->data);

first = first->next;

}

tt = tt -> next;

}

swap(first-> data, h->data);

return first;

}

};

void \_quickSort(struct Node\* l, struct Node \*h)

{

if (h != NULL && l != h && l != h->next)

{

Solution ob;

struct Node \*p = ob.partition(l, h);

\_quickSort(l, p->prev);

\_quickSort(p->next, h);

}

}

void quickSort(struct Node \*head)

{

struct Node \*h = lastNode(head);

\_quickSort(head, h);

}

* ****Time Complexity:****O(n^2) in the worst case when the list is already sorted. O(nlog(n)) in the best and average case.
* ****Space Complexity:**** O(n)

### **51. Write a function to connect nodes at the same level of a binary tree**

****Input:****  100

                        /     \

                   13      15

                   /  \         \

              14    1        20

****Output:****  100-> NULL

                        /      \

                  13   ->  15   -> NULL

                  /      \           \

               14  ->  1   -> 20 -> NULL

class Solution

{

public:

//Function to connect nodes at the same level.

void connect(Node \*p)

{

map<int,vector<Node \*> > m;

queue<Node \*> q;

queue<int> l;

q.push(p);

l.push(0);

while(!q.empty())

{

Node \*temp=q.front();

int level=l.front();

q.pop();

l.pop();

m[level].push\_back(temp);

if(temp->left!=NULL)

{

q.push(temp->left);

l.push(level+1);

}

if(temp->right!=NULL)

{

q.push(temp->right);

l.push(level+1);

}

}

for(map<int,vector<Node \*> > ::iterator it=m.begin();it!=m.end();it++)

{

vector<Node \*> temp1=it->second;

for(int i=0;i<temp1.size()-1;i++)

{

temp1[i]->nextRight=temp1[i+1];

}

temp1[temp1.size()-1]->nextRight=NULL;

}

}

};

* ****Time Complexity:****O(n)
* ****Space Complexity:****O(n)

### **52. Write a function to find number of structurally unique binary trees are possible**

****Input:****N = 3

****Output:****5 for N = 3, there are 5 possible BSTs:

1           3      3        2 1

   \         /      /          /  \   \

     3     2      1      1    3     2

   /      /          \                     \

2      1         2                       3

class Solution

{

public:

//function to calculate binomial coefficient C(n,k)

long long int binomialCoefficient(long long int n, long long int k)

{

long long int res = 1;

if (k > n - k)

k = n - k;

for (long long int i = 0; i < k; ++i)

{

res \*= (n - i);

res /= (i + 1);

}

return res;

}

//function to calculate Nth Catalan Number

long long int catalanNumber(long long in n)

{

// Calculate value of 2nCn

long long int C = binomialCoefficient(2\*n, n);

// return 2nCn/(n+1)

return C/(n+1);

}

//Function to return the total number of possible unique BST.

long long int numOfUniqueBinarySearchTrees(int n)

{

// find nth catalan number

long long int countOfUniqueBinarySearchTrees = catalanNumber(n);

// return nth catalan number

return countOfUniqueBinarySearchTrees;

}

};

* ****Time Complexity:****O(n)
* ****Space Complexity:****O(1)

### **53. Implement LRU(Least Recently Used) Cache**

class LRUCache

{

private:

class node\_t {

public:

int key;

int value;

node\_t \* next;

node\_t \* prev;

};

int cap;

node\_t head;

unordered\_map<int, node\_t\*> tbl;

void remove\_node(node\_t \* node) {

node->next->prev = node->prev;

node->prev->next = node->next;

}

void add\_node(node\_t \* node) {

node->next = head.next;

node->prev = &head;

head.next = node;

node->next->prev = node;

}

public:

//Constructor for initializing the cache capacity with the given value.

LRUCache(int cap): cap(cap)

{

// code here

head.prev = &head;

head.next = &head;

}

//Function to return value corresponding to the key.

int get(int key)

{

// your code here

unordered\_map<int, node\_t\*>::iterator it = tbl.find(key);

if(it==tbl.end())

return -1;

remove\_node(it->second);

add\_node(it->second);

return it->second->value;

}

//Function for storing key-value pair.

void set(int key, int value)

{

// your code here

unordered\_map<int, node\_t\*>::iterator it = tbl.find(key);

if(it!=tbl.end())

{

remove\_node(it->second);

add\_node(it->second);

it->second->value = value;

}

else {

node\_t \* node = new node\_t;

node->key = key;

node->value = value;

add\_node(node);

tbl[key] = node;

if(tbl.size()>cap) {

auto \* old\_node = head.prev;

tbl.erase(old\_node->key);

remove\_node(old\_node);

delete old\_node;

}

}

}

};

* ****Time Complexity:****O(1) to get an element
* ****Space Complexity:****O(n)

### **54. Write a function to determine whether duplicate elements in a given array are within a given distance of each other.**

* ****Input:****arr[] = {1, 2, 3, 4, 2, 1, 2} range=3
* ****Output:****True

class Solution {

public:

bool checkDuplicatesWithinRange(vector<int> arr, int range)

{

// Creating an empty hashset

unordered\_set<int> myset;

// Traversing the input array

for (int i = 0; i < arr.size(); i++)

{

// If already present in hashset, then we found a duplicate within range distance

if (myset.find(arr[i]) != myset.end())

return true;

// Add this item to hashset

myset.insert(arr[i]);

// Remove the range+1 distant item from the hashset

if (i >= range)

myset.erase(arr[i-range]);

}

return false;

}

};

* ****Time Complexity:****O(n)
* ****Space Complexity:****O(n)

### **55. Write a recursive function to calculate the height of a binary tree in Java.**

* Consider that every node of a tree represents a class called Node as given below:

public class Node{

int data;

Node left;

Node right;

}

* Then the height of the binary tree can be found as follows:

int heightOfBinaryTree(Node node)

{

if (node == null)

return 0; // If node is null then height is 0 for that node.

else

{

// compute the height of each subtree

int leftHeight = heightOfBinaryTree(node.left);

int rightHeight = heightOfBinaryTree(node.right);

//use the larger among the left and right height and plus 1 (for the root)

return Math.max(leftHeight, rightHeight) + 1;

}

}

### **56. Write Java code to count number of nodes in a binary tree**

**int** **countNodes**(Node root){

**int** count = 1; //Root itself should be counted

**if** (root ==**null**)

**return** 0;

**else**

{

count += countNodes(root.left);

count += countNodes(root.right);

**return** count;

}

}

### **57. Print Left view of any binary trees.**

* The main idea to solve this problem is to traverse the tree in pre order manner and pass the level information along with it. If the level is visited for the first time, then we store the information of the current node and the current level in the hashmap. Basically, we are getting the left view by noting the first node of every level.
* At the end of traversal, we can get the solution by just traversing the map.
* Consider the following tree as example for finding the left view:
* Left view of a binary tree in Java:

import java.util.HashMap;

//to store a Binary Tree node

class Node

{

int data;

Node left = null, right = null;

Node(int data) {

this.data = data;

}

}

public class InterviewBit

{

// traverse nodes in pre-order way

public static void leftViewUtil(Node root, int level, HashMap<Integer, Integer> map)

{

if (root == null) {

return;

}

// if you are visiting the level for the first time

// insert the current node and level info to the map

if (!map.containsKey(level)) {

map.put(level, root.data);

}

leftViewUtil(root.left, level + 1, map);

leftViewUtil(root.right, level + 1, map);

}

// to print left view of binary tree

public static void leftView(Node root)

{

// create an empty HashMap to store first node of each level

HashMap<Integer, Integer> map = new HashMap<>();

// traverse the tree and find out the first nodes of each level

leftViewUtil(root, 1, map);

// iterate through the HashMap and print the left view

for (int i = 0; i <map.size(); i++) {

System.out.print(map.get(i) + " ");

}

}

public static void main(String[] args)

{

Node root = new Node(4);

root.left = new Node(2);

root.right = new Node(6);

root.left.left = new Node(1);

root.left.left = new Node(3);

root.right.left = new Node(5);

root.right.right = new Node(7);

root.right.left.left = new Node(9);

leftView(root);

}

}

### **58. Given an m x n 2D grid map of '1’s which represents land and '0’s that represents water return the number of islands (surrounded by water and formed by connecting adjacent lands in 2 directions - vertically or horizontally).**

### ****Assume that the boundary cases - which are all four edges of the grid are surrounded by water.****

### **Constraints are:**

### **m == grid.length n == grid[i].length 1 <= m, n <= 300 grid[i][j] can only be ‘0’ or ‘1’.**

### **Example:**

### **Input: grid = [ [“1” , “1” , “1” , “0” , “0”], [“1” , “1” , “0” , “0” , “0”], [“0” , “0” , “1” , “0” , “1”], [“0” , “0” , “0” , “1” , “1”] ]**

### **Output: 3**

class InterviewBit {

public int numberOfIslands(char[][] grid) {

if(grid==null || grid.length==0||grid[0].length==0)

return 0;

int m = grid.length;

int n = grid[0].length;

int count=0;

for(int i=0; i<m; i++){

for(int j=0; j<n; j++){

if(grid[i][j]=='1'){

count++;

mergeIslands(grid, i, j);

}

}

}

return count;

}

public void mergeIslands(char[][] grid, int i, int j){

int m=grid.length;

int n=grid[0].length;

if(i<0||i>=m||j<0||j>=n||grid[i][j]!='1')

return;

grid[i][j]='X';

mergeIslands(grid, i-1, j);

mergeIslands(grid, i+1, j);

mergeIslands(grid, i, j-1);

mergeIslands(grid, i, j+1);

}

}

### **59. What is topological sorting in a graph?**

* Topological sorting is a linear ordering of vertices such that for every directed edge ij, vertex i comes before j in the ordering.
* Topological sorting is only possible for Directed Acyclic Graph (DAG).
* Applications:
  1. jobs scheduling from the given dependencies among jobs.
  2. ordering of formula cell evaluation in spreadsheets
  3. ordering of compilation tasks to be performed in make files,
  4. data serialization
  5. resolving symbol dependencies in linkers.
* Topological Sort Code in Java:

// V - total vertices

// visited - boolean array to keep track of visited nodes

// graph - adjacency list.

// Main Topological Sort Function.

void topologicalSort()

{

Stack<Integer> stack = new Stack<Integer>();

// Mark all the vertices as not visited

boolean visited[] = new boolean[V];

for (int j = 0; j < V; j++){

visited[j] = false;

}

// Call the util function starting from all vertices one by one

for (int i = 0; i < V; i++)

if (visited[i] == false)

topologicalSortUtil(i, visited, stack);

// Print contents of stack -> result of topological sort

while (stack.empty() == false)

System.out.print(stack.pop() + " ");

}

// A helper function used by topologicalSort

void topologicalSortUtil(int v, boolean visited[],

Stack<Integer> stack)

{

// Mark the current node as visited.

visited[v] = true;

Integer i;

// Recur for all the vertices adjacent to the current vertex

Iterator<Integer> it = graph.get(v).iterator();

while (it.hasNext()) {

i = it.next();

if (!visited[i])

topologicalSortUtil(i, visited, stack);

}

// Push current vertex to stack that saves result

stack.push(new Integer(v));

}

### 1) What is Data Structure? Explain.

The data structure is a way that specifies how to organize and manipulate the data. It also defines the relationship between them. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc. Data Structures are the central part of many computer science algorithms as they enable the programmers to handle the data in an efficient way

### 2) Describe the types of Data Structures?

Data Structures are mainly classified into two types:

****Linear Data Structure:**** A data structure is called linear if all of its elements are arranged in the sequential order. In linear data structures, the elements are stored in a non-hierarchical way where each item has the successors and predecessors except the first and last element.

****Non-Linear Data Structure:**** The Non-linear data structure does not form a sequence i.e. each item or element is connected with two or more other items in a non-linear arrangement. The data elements are not arranged in the sequential structure.

### 3) List the area of applications of Data Structure.

Data structures are applied extensively in the following areas of computer science:

* Compiler Design,
* Operating System,
* Database Management System,
* Statistical analysis package,
* Numerical Analysis,
* Graphics,
* Artificial Intelligence,
* Simulation

### 4) What is the difference between file structure and storage structure?

Difference between file structure and storage structure:

The main difference between file structure and storage structure is based on memory area that is being accessed.

****Storage structure:**** It is the representation of the data structure in the computer memory.

****File structure:**** It is the representation of the storage structure in the auxiliary memory.

### 5) List the data structures which are used in RDBMS, Network Data Modal, and Hierarchical Data Model.

* RDBMS uses Array data structure
* Network data model uses Graph
* Hierarchal data model uses Trees

### 6) Which data structure is used to perform recursion?

Stack data structure is used in recursion due to its last in first out nature. Operating system maintains the stack in order to save the iteration variables at each function call

### 7) What is a Stack?

Stack is an ordered list in which, insertion and deletion can be performed only at one end that is called the top. It is a recursive data structure having pointer to its top element. The stack is sometimes called as Last-In-First-Out (LIFO) list i.e. the element which is inserted first in the stack will be deleted last from the stack.

### 8) List the area of applications where stack data structure can be used?

* Expression evaluation
* Backtracking
* Memory Management
* Function calling and return

### 9) What are the operations that can be performed on a stack?

* Push Operations
* Pop Operations
* Peek Operations

### 10) Write the stack overflow condition.

Overflow occurs when ****top = Maxsize -1****

### 11) What is the difference between PUSH and POP?

PUSH and POP operations specify how data is stored and retrieved in a stack.

****PUSH:**** PUSH specifies that data is being "inserted" into the stack.

****POP:**** POP specifies data retrieval. It means that data is being deleted from the stack.

### 12) Write the steps involved in the insertion and deletion of an element in the stack.

****Push:****

* Increment the variable top so that it can refer to the next memory allocation
* Copy the item to the at the array index value equal to the top
* Repeat step 1 and 2 until stack overflows

****Pop:****

* Store the topmost element into the an another variable
* Decrement the value of the top
* Return the topmost element

### 13) What is a postfix expression?

An expression in which operators follow the operands is known as postfix expression. The main benefit of this form is that there is no need to group sub-expressions in parentheses or to consider operator precedence.

The expression "a + b" will be represented as "ab+" in postfix notation.

### 14)Write the postfix form of the expression: (A + B) \* (C - D)

AB+CD-\*

### )15) Which notations are used in Evaluation of Arithmetic Expressions using prefix and postfix forms?

Polish and Reverse Polish notations.

### 16)What is an array?

Arrays are defined as the collection of similar types of data items stored at contiguous memory locations. It is the simplest data structure in which each data element can be randomly accessed by using its index number.

### 17) How to reference all the elements in a one-dimension array?

It can be done by using an indexed loop such that the counter runs from 0 to the array size minus one. In this manner, you can reference all the elements in sequence by using the loop counter as the array subscript.

### 18) What is a multidimensional array?

The multidimensional array can be defined as the array of arrays in which, the data is stored in tabular form consists of rows and columns. 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

### 19) How are the elements of a 2D array are stored in the memory?

There are two techniques by using which, the elements of a 2D array can be stored in the memory.

* ****Row-Major Order:**** In row-major ordering, all the rows of the 2D array are stored into the memory contiguously. First, the 1st row of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last row.
* ****Column-Major Order:**** In column-major ordering, all the columns of the 2D array are stored into the memory contiguously. first, the 1st column of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last column of the array.

### 20) Calculate the address of a random element present in a 2D array, given base address as BA.

****Row-Major Order:**** If array is declared as a[m][n] where m is the number of rows while n is the number of columns, then address of an element a[i][j] of the array stored in row major order is calculated as,

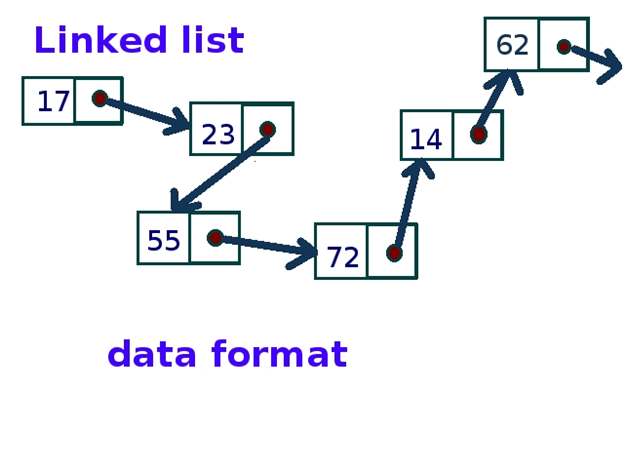
****Address(a[i][j]) = B. A. + (i \* n + j) \* size****

****Column-Major Order:**** If array is declared as a[m][n] where m is the number of rows while n is the number of columns, then address of an element a[i][j] of the array stored in column major order is calculated as

****Address(a[i][j]) = ((j\*m)+i)\*Size + BA****.

### 21) Define Linked List Data structure.

Linked List is the collection of randomly stored data objects called nodes. In Linked List, each node is linked to its adjacent node through a pointer. A node contains two fields, i.e. Data Field and Link Field.



### 22) Are linked lists considered linear or non-linear data structures?

A linked list is considered both linear and non-linear data structure depending upon the situation.

* On the basis of data storage, it is considered as a non-linear data structure.
* On the basis of the access strategy, it is considered as a linear data-structure.

### 23) What are the advantages of Linked List over an array?

* The size of a linked list can be incremented at runtime which is impossible in the case of the array.
* The List is not required to be contiguously present in the main memory, if the contiguous space is not available, the nodes can be stored anywhere in the memory connected through the links.
* The List is dynamically stored in the main memory and grows as per the program demand while the array is statically stored in the main memory, size of which must be declared at compile time.
* The number of elements in the linked list are limited to the available memory space while the number of elements in the array is limited to the size of an array.

### 24) Write the syntax in C to create a node in the singly linked list.

1. struct node
2. {
3. **int** data;
4. struct node \*next;
5. };
6. struct node \*head, \*ptr;
7. ptr = (struct node \*)malloc(sizeof(struct node));

### 25) If you are using C language to implement the heterogeneous linked list, what pointer type should be used?

The heterogeneous linked list contains different data types, so it is not possible to use ordinary pointers for this. For this purpose, you have to use a generic pointer type like void pointer because the void pointer is capable of storing a pointer to any type.

### 26) What is doubly linked list?

The doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. In a doubly linked list, a node consists of three parts:

* node data
* pointer to the next node in sequence (next pointer)
* pointer to the previous node (previous pointer).

### 27) Write the C program to insert a node in circular singly list at the beginning.

1. #include<stdio.h>
2. #include<stdlib.h>
3. **void** beg\_insert(**int**);
4. struct node
5. {
6. **int** data;
7. struct node \*next;
8. };
9. struct node \*head;
10. **void** main ()
11. {
12. **int** choice,item;
13. **do**
14. {
15. printf("\nEnter the item which you want to insert?\n");
16. scanf("%d",&item);
17. beg\_insert(item);
18. printf("\nPress 0 to insert more ?\n");
19. scanf("%d",&choice);
20. }**while**(choice == 0);
21. }
22. **void** beg\_insert(**int** item)
23. {
25. struct node \*ptr = (struct node \*)malloc(sizeof(struct node));
26. struct node \*temp;
27. **if**(ptr == NULL)
28. {
29. printf("\nOVERFLOW");
30. }
31. **else**
32. {
33. ptr -> data = item;
34. **if**(head == NULL)
35. {
36. head = ptr;
37. ptr -> next = head;
38. }
39. **else**
40. {
41. temp = head;
42. **while**(temp->next != head)
43. temp = temp->next;
44. ptr->next = head;
45. temp -> next = ptr;
46. head = ptr;
47. }
48. printf("\nNode Inserted\n");
49. }
51. }

### 28) Define the queue data structure.

A queue can be defined as an ordered list which enables insert operations to be performed at one end called REAR and delete operations to be performed at another end called FRONT.

### 29) List some applications of queue data structure.

The Applications of the queue is given as follows:

* Queues are widely used as waiting lists for a single shared resource like a printer, disk, CPU.
* Queues are used in the asynchronous transfer of data (where data is not being transferred at the same rate between two processes) for eg. pipes, file IO, sockets.
* Queues are used as buffers in most of the applications like MP3 media player, CD player, etc.
* Queues are used to maintain the playlist in media players to add and remove the songs from the play-list.
* Queues are used in operating systems for handling interrupts.

### 30) What are the drawbacks of array implementation of Queue?

* ****Memory Wastage:**** The space of the array, which is used to store queue elements, can never be reused to store the elements of that queue because the elements can only be inserted at front end and the value of front might be so high so that, all the space before that, can never be filled.
* ****Array Size:**** There might be situations in which, we may need to extend the queue to insert more elements if we use an array to implement queue, It will almost be impossible to extend the array size, therefore deciding the correct array size is always a problem in array implementation of queue.

### 31) What are the scenarios in which an element can be inserted into the circular queue?

* If (rear + 1)%maxsize = front, the queue is full. In that case, overflow occurs and therefore, insertion can not be performed in the queue.
* If rear != max - 1, the rear will be incremented to the mod(maxsize) and the new value will be inserted at the rear end of the queue.
* If front != 0 and rear = max - 1, it means that queue is not full therefore, set the value of rear to 0 and insert the new element there.

### 32) What is a dequeue?

Dequeue (also known as double-ended queue) can be defined as an ordered set of elements in which the insertion and deletion can be performed at both the ends, i.e. front and rear.

### 33) What is the minimum number of queues that can be used to implement a priority queue?

Two queues are needed. One queue is used to store the data elements, and another is used for storing priorities.

### 34) Define the tree data structure.

The Tree is a recursive data structure containing the set of one or more data nodes where one node is designated as the root of the tree while the remaining nodes are called as the children of the root. The nodes other than the root node are partitioned into the nonempty sets where each one of them is to be called sub-tree.

### 35) List the types of tree.

There are six types of tree given as follows.

* General Tree
* Forests
* Binary Tree
* Binary Search Tree
* Expression Tree
* Tournament Tree

### 36) What are Binary trees?

A binary Tree is a special type of generic tree in which, each node can have at most two children. Binary tree is generally partitioned into three disjoint subsets, i.e. the root of the node, left sub-tree and Right binary sub-tree.

### 37) Write the C code to perform in-order traversal on a binary tree.

1. **void** in-order(struct treenode \*tree)
2. {
3. **if**(tree != NULL)
4. {
5. in-order(tree→ left);
6. printf("%d",tree→ root);
7. in-order(tree→ right);
8. }
9. }

### 38) What is the maximum number of nodes in a binary tree of height k?

2k+1-1 where k >= 1

### 39) Which data structure suits the most in the tree construction?

Queue data structure

### 40) Which data structure suits the most in the tree construction?

Queue data structure

### 41) Write the recursive C function to count the number of nodes present in a binary tree.

1. **int** count (struct node\* t)
2. {
3. **if**(t)
4. {
5. **int** l, r;
6. l = count(t->left);
7. r=count(t->right);
8. **return** (1+l+r);
9. }
10. **else**
11. {
12. **return** 0;
13. }
14. }

### 42) Write a recursive C function to calculate the height of a binary tree.

1. **int** countHeight(struct node\* t)
2. {
3. **int** l,r;
4. **if**(!t)
5. **return** 0;
6. **if**((!(t->left)) && (!(t->right)))
7. **return** 0;
8. l=countHeight(t->left);
9. r=countHeight(t->right);
10. **return** (1+((l>r)?l:r));
11. }

### 43) How can AVL Tree be useful in all the operations as compared to Binary search tree?

AVL tree controls the height of the binary search tree by not letting it be skewed. The time taken for all operations in a binary search tree of height h is O(h). However, it can be extended to O(n) if the BST becomes skewed (i.e. worst case). By limiting this height to log n, AVL tree imposes an upper bound on each operation to be O(log n) where n is the number of nodes.

### 44) State the properties of B Tree.

A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.

* Every node in a B-Tree contains at most m children.
* Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
* The root nodes must have at least 2 nodes.
* All leaf nodes must be at the same level.

### 45) What are the differences between B tree and B+ tree?

|  |  |  |
| --- | --- | --- |
| **SN** | **B Tree** | **B+ Tree** |
| 1 | Search keys cannot repeatedly be stored. | Redundant search keys can be present. |
| 2 | Data can be stored in leaf nodes as well as internal nodes | Data can only be stored on the leaf nodes. |
| 3 | Searching for some data is a slower process since data can be found on internal nodes as well as on the leaf nodes. | Searching is comparatively faster as data can only be found on the leaf nodes. |
| 4 | Deletion of internal nodes is so complicated and time-consuming. | Deletion will never be a complexed process since element will always be deleted from the leaf nodes. |
| 5 | Leaf nodes cannot be linked together. | Leaf nodes are linked together to make the search operations more efficient. |

### 46) List some applications of Tree-data structure?

Applications of Tree- data structure:

* The manipulation of Arithmetic expression,
* Symbol Table construction,
* Syntax analysis
* Hierarchal data model

### 47) Define the graph data structure?

A graph G can be defined as an ordered set G(V, E) where V(G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having parent-child relations.

### 48) Differentiate among cycle, path, and circuit?

* ****Path:**** A Path is the sequence of adjacent vertices connected by the edges with no restrictions.
* ****Cycle:**** A Cycle can be defined as the closed path where the initial vertex is identical to the end vertex. Any vertex in the path can not be visited twice
* ****Circuit:**** A Circuit can be defined as the closed path where the intial vertex is identical to the end vertex. Any vertex may be repeated.

### 49) Mention the data structures which are used in graph implementation.

For the graph implementation, following data structures are used.

* In sequential representation, Adjacency matrix is used.
* In Linked representation, Adjacency list is used.

### 50) Which data structures are used in BFS and DFS algorithm?

* In BFS algorithm, Queue data structure is used.
* In DFS algorithm, Stack data structure is used.

### 51) What are the applications of Graph data structure?

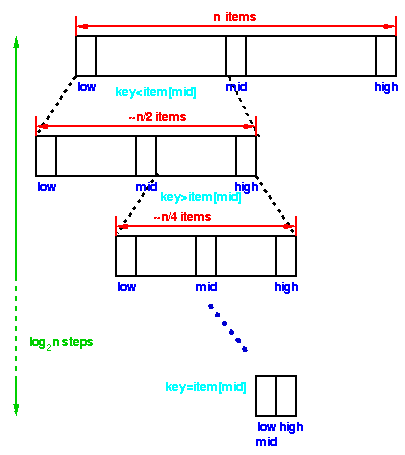
The graph has the following applications:

* Graphs are used in circuit networks where points of connection are drawn as vertices and component wires become the edges of the graph.
* Graphs are used in transport networks where stations are drawn as vertices and routes become the edges of the graph.
* Graphs are used in maps that draw cities/states/regions as vertices and adjacency relations as edges.
* Graphs are used in program flow analysis where procedures or modules are treated as vertices and calls to these procedures are drawn as edges of the graph.

### 54) In what scenario, Binary Search can be used?

Binary Search algorithm is used to search an already sorted list. The algorithm follows divide and conqer approach

****Example:****



### 52) What are the advantages of Binary search over linear search?

There are relatively less number of comparisons in binary search than that in linear search. In average case, linear search takes O(n) time to search a list of n elements while Binary search takes O(log n) time to search a list of n elements.

### 53) What are the advantages of Selecetion Sort?

* It is simple and easy to implement.
* It can be used for small data sets.
* It is 60 per cent more efficient than bubble sort.

### 55) List Some Applications of Multilinked Structures?

* Sparse matrix,
* Index generation.

### 56) What is the difference between NULL and VOID?

* Null is actually a value, whereas Void is a data type identifier.
* A null variable simply indicates an empty value, whereas void is used to identify pointers as having no initial size.

### 1. What is a Data Structure?

The [Data Structure](https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-data-structure" \o "Data Structure" \t "https://www.simplilearn.com/_blank) is the way data is organized (stored) and manipulated for retrieval and access. It also defines the way different sets of data relate to one another, establishing relationships and forming [algorithms.](https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-an-algorithm" \o "algorithms." \t "https://www.simplilearn.com/_blank)

### 2. Describe the types of Data Structures?

The following are the types of data structures:

1. **Lists:** A collection of related things linked to the previous or/and following data items.
2. **Arrays**: A collection of values that are all the same.
3. **Records**: A collection of fields, each of which contains data from a single data type.
4. **Trees**: A data structure that organizes data in a hierarchical framework. This form of data structure follows the ordered order of data item insertion, deletion, and modification.
5. **Tables**: The data is saved in the form of rows and columns. These are comparable to records in that the outcome or alteration of data is mirrored across the whole table.

### 3. What is a Linear Data Structure? Name a few examples.

A data structure is linear if all its elements or data items are arranged in a sequence or a linear order. The elements are stored in a non-hierarchical way so that each item has successors and predecessors except the first and last element in the list.

Examples of linear data structures are Arrays, Stack, Strings, Queue, and Linked List.

### 4. What are some applications of Data Structures?

In terms of data structure interview questions, this is one of the most frequently asked question.

Numerical analysis, operating system, AI, compiler design, [database management](https://www.simplilearn.com/what-is-database-management-article" \o "database management" \t "https://www.simplilearn.com/_blank), graphics, [statistical analysis](https://www.simplilearn.com/what-is-statistical-analysis-article" \o "statistical analysis" \t "https://www.simplilearn.com/_blank), and simulation.

### 5. What is the difference between file structure and storage structure?

The difference lies in the memory area accessed. Storage structure refers to the data structure in the memory of the computer system, whereas file structure represents the storage structure in the auxiliary memory.

### 6. What is a multidimensional array?

A multidimensional array is a multidimensional array with more than one dimension. It is an array of arrays or an array with numerous layers. The 2D array, or two-dimensional array, is the most basic multidimensional array. As you'll see in the code, it's technically an array of arrays. A 2D array is also referred to as a matrix or a table with rows and columns. Declaring a multidimensional array is the same as saying a one-dimensional array. We need to notify C that we have two dimensions for a two-dimensional array.

### 7. How are the elements of a 2D array stored in the memory?

1. **Row-Major Order:** -In row-major ordering, all of the rows of a 2D array are stored in memory in a contiguous manner.

First, the first row of the array is entirely stored in memory, followed by the second row of the array, and so on until the final row.

1. **Column-Major Order**: In column-major ordering, all of the columns of a 2D array are stored in memory in the same order. The first column of the array is entirely saved in memory, followed by the second row of the array, and so on until the last column of the array is wholly recorded in memory.

### 8. What is a linked list Data Structure?

This is one of the most frequently asked data structure interview questions where the interviewer expects you to give a thorough answer. Try to explain as much as possible rather than finishing your answer in a sentence!

It’s a linear Data Structure or a sequence of data objects where elements are not stored in adjacent memory locations. The elements are linked using pointers to form a chain. Each element is a separate object, called a node.  Each node has two items: a data field and a reference to the next node. The entry point in a linked list is called the head. Where the list is empty, the head is a null reference and the last node has a reference to null.

A [linked list](https://www.simplilearn.com/tutorials/data-structure-tutorial/linked-list-in-data-structure" \o "linked list" \t "https://www.simplilearn.com/_blank) is a dynamic data structure, where the number of nodes is not fixed, and the list has the ability to grow and shrink on demand.

It is applied in cases where:

* We deal with an unknown number of objects or don’t know how many items are in the list
* We need constant-time insertions/deletions from the list, as in real-time computing where time predictability is critical
* Random access to any elements is not needed
* The algorithm requires a data structure where objects need to be stored irrespective of their physical address in memory
* We need to insert items in the middle of the list as in a priority queue

Some implementations are stacks and queues, graphs, directory of names, dynamic memory allocation, and performing arithmetic operations on long integers.

### 9. Are linked lists considered linear or non-linear Data Structures?

Linked lists are considered both linear and non-linear data structures depending upon the application they are used for. When used for access strategies, it is considered as a linear data-structure. When used for data storage, it is considered a non-linear data structure.

### 10. What are the advantages of a linked list over an array? In which scenarios do we use Linked List and when Array?

This is another frequently asked data structure interview questions! Advantages of a linked list over an array are:

#### 1. Insertion and Deletion

Insertion and deletion of nodes is an easier process, as we only update the address present in the next pointer of a node. It’s expensive to do the same in an array as the room has to be created for the new elements and existing elements must be shifted.

#### 2. Dynamic Data Structure

As a linked list is a dynamic data structure, there is no need to give an initial size as it can grow and shrink at runtime by allocating and deallocating memory. However, the size is limited in an array as the number of elements is statically stored in the main memory.

#### 3. No Wastage of Memory

As the size of a linked list can increase or decrease depending on the demands of the program, and memory is allocated only when required, there is no memory wasted. In the case of an array, there is memory wastage. For instance, if we declare an array of size 10 and store only five elements in it, then the space for five elements is wasted.

#### 4. Implementation

Data structures like [stack and queues](https://www.simplilearn.com/tutorials/data-structure-tutorial/stacks-and-queues" \o "stack and queues" \t "https://www.simplilearn.com/_blank) are more easily implemented using a linked list than an array.

Some scenarios where we use linked list over array are:

* When we know the upper limit on the number of elements in advance
* When there are a large number of add or remove operations
* When there are no large number of random access to elements
* When we want to insert items in the middle of the list, such as when implementing a [priority queue](https://www.simplilearn.com/tutorials/data-structure-tutorial/priority-queue-in-data-structure" \o "priority queue" \t "https://www.simplilearn.com/_blank)

Some scenarios in which we use array over the linked list are:

* When we need to index or randomly access elements
* When we know the number of elements in the array beforehand, so we can allocate the correct amount of memory
* When we need speed when iterating through all the elements in the sequence
* When memory is a concern; filled arrays use less memory than linked lists, as each element in the array is the data but each linked list node requires the data as well as one or more pointers to the other elements in the linked list

In summary, we consider the requirements of space, time, and ease of implementation to decide whether to use a linked list or array.

### 11. What is a doubly-linked list? Give some examples.

It is a complex type (double-ended LL) of a linked list in which a node has two links, one that connects to the next node in the sequence and another that connects to the previous node. This allows traversal across the data elements in both directions.

Examples include:

* A music playlist with next and previous navigation buttons
* The browser cache with BACK-FORWARD visited pages
* The undo and redo functionality on a browser, where you can reverse the node to get to the previous page

### 12. How do you reference all of the elements in a one-dimension array?

Using an indexed loop, we may access all of the elements in a one-dimensional array. The counter counts down from 0 to the maximum array size, n, minus one. The loop counter is used as the array subscript to refer to all items of the one-dimensional array in succession.

### 13. What are dynamic Data Structures? Name a few.

They are collections of data in memory that expand and contract to grow or shrink in size as a program runs. This enables the programmer to control exactly how much memory is to be utilized.

Examples are the dynamic array, linked list, stack, queue, and heap.

Enroll today for the [Java Certification Training Course](https://www.simplilearn.com/mobile-and-software-development/java-javaee-soa-development-training?source=GhPreviewCTAText" \l "course-curriculum" \o "Java Certification Training Course" \t "https://www.simplilearn.com/_blank) to learn all about arrays, loops, operators, and more. Check out the course curriculum now!

### 14. What is an algorithm?

An algorithm is a step by step method of solving a problem or manipulating data. It defines a set of instructions to be executed in a certain order to get the desired output.

### 15. Why do we need to do an algorithm analysis?

A problem can be solved in more than one way using several solution algorithms. Algorithm analysis provides an estimation of the required resources of an algorithm to solve a specific computational problem. The amount of time and space resources required to execute is also determined.

The time complexity of an algorithm quantifies the amount of time taken for an algorithm to run as a function of the length of the input. The space complexity quantifies the amount of space or memory taken by an algorithm, to run as a function of the length of the input.

### 16. What is a stack?

A [stack](https://www.simplilearn.com/tutorials/data-structure-tutorial/stacks-and-queues" \o "stack" \t "https://www.simplilearn.com/_blank) is an abstract data type that specifies a linear data structure, as in a real physical stack or piles where you can only take the top item off the stack in order to remove things. Thus, insertion (push) and deletion (pop) of items take place only at one end called top of the stack, with a particular order: LIFO (Last In First Out) or FILO (First In Last Out).

### 17. Where are stacks used?

* Expression, evaluation, or conversion of evaluating prefix, postfix, and infix expressions
* Syntax parsing
* String reversal
* Parenthesis checking
* Backtracking

### 18. What are the operations that can be performed on a stack?

A stack is a linear data structure that operates on the same concept, in that components in a stack are added and deleted only from one end, referred to as the TOP. As a result, a stack is known as a LIFO (Last-In-First-Out) data structure because the piece that was put last is the first to be removed.

**A stack may perform three fundamental operations:**

1. **PUSH**: The push action inserts a new element into the stack. The new feature is placed at the top of the stack. However, before inserting the value, we must first verify if TOP=MAX–1, since if so, the stack is filled, and no more insertions are possible. An OVERFLOW message is printed if an attempt is made to put a value into an existing stack.
2. **POP:** The pop operation is performed to remove the stack's topmost element. However, before removing the value, we must first verify if TOP=NULL, since if it is, the stack is empty, and no further deletions are permitted. An UNDERFLOW notice is produced if an attempt is made to erase a value from a stack that is already empty.
3. **PEEK:** A peek action returns the value of the stack's topmost element without removing it from the stack. On the other hand, the Peek operation first checks if the stack is empty, i.e., if TOP = NULL, then an appropriate message is written. Otherwise, a value is returned.

## Data Structure Interview Questions for Experienced

### 19. What is a postfix expression?

A postfix expression is made up of operators and operands, with the operator coming after the operands. That is, in a postfix expression, the operator comes after the operands. Likewise, what is the proper postfix form? The correct postfix phrase is A B + C \*.

### 20. What is a queue Data Structure?

In this type of data structure interview questions, you can also discuss your experience and situations using queue. A queue is an abstract data type that specifies a linear data structure or an ordered list,  using the First In First Out (FIFO) operation to access elements. Insert operations can be performed only at one end called REAR and delete operations can be performed only at the other end called FRONT.

### 21. List some applications of queue Data Structure.

To prioritize jobs as in the following scenarios:

* As waiting lists for a single shared resource in a printer, CPU, call center systems, or image uploads; where the first one entered is the first to be processed
* In the asynchronous transfer of data; or example pipes, file IO, and sockets
* As buffers in applications like MP3 media players and CD players
* To maintain the playlist in media players (to add or remove the songs)

### 22. What is a Dequeue?

It is a double-ended queue, or a data structure, where the elements can be inserted or deleted at both ends (FRONT and REAR).

### 23. What operations can be performed on queues?

* enqueue() adds an element to the end of the queue
* dequeue() removes an element from the front of the queue
* init() is used for initializing the queue
* isEmpty tests for whether or not the queue is empty
* The front is used to get the value of the first data item but does not remove it
* The rear is used to get the last item from a queue

### 24. What are the advantages of the heap over a stack?

In this data structure interview questions, try giving various advantages, along with examples, if possible. It will show the interviewer your domain expertise. Generally, both[heap and stack](https://www.simplilearn.com/tutorials/data-structure-tutorial/stacks-vs-heap" \o "heap and stack" \t "https://www.simplilearn.com/_blank) are part of memory and used in Java for different needs:

* Heap is more flexible than the stack because memory space can be dynamically allocated and de-allocated as needed
* Heap memory is used to [store objects in Java,](https://www.simplilearn.com/tutorials/java-tutorial/java-classes-and-objects" \o "store objects in Java," \t "https://www.simplilearn.com/_blank) whereas stack memory is used to store local variables and function call
* Objects created in the heap are visible to all threads, whereas variables stored in stacks are only visible to the owner as private memory
* When using recursion, the size of heap memory is more whereas it quickly fill-ups stack memory

### 25. Where can stack Data Structure be used?

* Expression evaluation
* Backtracking
* Memory management
* Function calling and return

### 26. What is the difference between a PUSH and a POP?

In terms of data structure interview questions, this is one of the most frequently asked question.

The acronyms stand for Pushing and Popping operations performed on a stack. These are ways data is stored and retrieved.

* PUSH is used to add an item to a stack, while POP is used to remove an item.
* PUSH takes two arguments, the name of the stack to add the data to and the value of the entry to be added. POP only needs the name of the stack.
* When the stack is filled and another PUSH command is issued, you get a stack overflow error, which means that the stack can no longer accommodate the last PUSH. In POP, a stack underflow error occurs when you’re trying to POP an already empty stack.

### 27. Which sorting algorithm is considered the fastest? Why?

A single sorting algorithm can’t be considered best, as each algorithm is designed for a particular data structure and data set. However, the [QuickSort](https://www.simplilearn.com/tutorials/c-tutorial/quick-sort-in-c" \o "QuickSort" \t "https://www.simplilearn.com/_blank) algorithm is generally considered the fastest because it has the best performance for most inputs.

Its advantages over other sorting algorithms include the following:

* Cache-efficient: It linearly scans and linearly partitions the input. This means we can make the most of every cache load.
* Can skip some swaps: As QuickSort is slightly sensitive to input that is in the right order, it can skip some swaps.
* Efficient even in worst-case input sets, as the order is generally random.
* Easy adaption to already- or mostly-sorted inputs.
* When speed takes priority over stability.

### 28. What is the merge sort? How does it work?

[Merge sort](https://www.simplilearn.com/tutorials/data-structure-tutorial/merge-sort-algorithm" \o "Merge sort" \t "https://www.simplilearn.com/_blank) is a divide-and-conquer algorithm for sorting the data. It works by merging and sorting adjacent data to create bigger sorted lists, which are then merged recursively to form even bigger sorted lists until you have one single sorted list.

### 29. How does the Selection sort work?

This is one of the most frequently asked data structure interview questions. Selection sort works by repeatedly picking the smallest number in ascending order from the list and placing it at the beginning. This process is repeated moving toward the end of the list or sorted subarray.

Scan all items and find the smallest. Switch over the position as the first item. Repeat the selection sort on the remaining N-1 items. We always iterate forward (i from 0 to N-1) and swap with the smallest element (always i).

Time complexity: best case O(n2); worst O(n2)

Space complexity: worst O(1)

### 30. What is an asymptotic analysis of an algorithm?

Asymptotic analysis is the technique of determining an algorithm's running time in mathematical units to determine the program's limits, also known as "run-time performance." The purpose is to identify the best case, worst case, and average-case times for completing a particular activity. While not a deep learning training technique, Asymptotic analysis is an essential diagnostic tool for programmers to analyze an algorithm's efficiency rather than its correctness.

### 31. What are asymptotic notations?

Asymptotic Notation represents an algorithm's running time - how long an algorithm takes with a given input, n. Big O, large Theta (), and big Omega () are the three distinct notations. When the running time is the same in all circumstances, big- is used, big-O for the worst-case running time, and big- for the best case running time.

### 32. What are some examples of divide and conquer algorithms?

Quicksort is the name of a sorting algorithm. The method selects a pivot element and rearranges the array elements so that all items less than the pivot chosen element go to the left side of the pivot and all elements more significant than the pivot element move to the right side.

Merge Sort is a sorting algorithm as well. The algorithm divides the array into two halves, sorts them recursively, and then combines the two sorted halves. The goal of points that are closest together is to identify the nearest pair of points in an x-y plane collection of points. The issue may be solved in O(n2) time by computing the distances between each pair of locations and comparing them to determine the shortest distance.

### 33. Define the [graph Data Structure](https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure" \o "graph Data Structure" \t "https://www.simplilearn.com/_blank)?

It is a type of non-linear data structure that consists of vertices or nodes connected by edges or arcs to enable storage or retrieval of data. Edges may be directed or undirected.

### 34. What are the applications of graph Data Structure?

* Transport grids where stations are represented as vertices and routes as the edges of the graph
* Utility graphs of power or water, where vertices are connection points and edge the wires or pipes connecting them
* Social network graphs to determine the flow of information and hotspots (edges and vertices)
* Neural networks where vertices represent neurons and edge the synapses between them

### 35. List the types of trees?

Data structure interview questions like this are very common and frequently asked

* The General Tree

A tree is referred to as a generic tree if its hierarchy is not constrained. In the General Tree, each node can have an endless number of offspring, and all other trees are subsets of the tree.

* The Binary Tree

The [binary tree](https://www.simplilearn.com/tutorials/data-structure-tutorial/binary-tree-in-data-structures" \o "binary tree" \t "https://www.simplilearn.com/_blank) is a type of tree in which each parent has at least two offspring. The children are referred to as the left and right youngsters. This tree is more popular than most others. When specific limitations and features are given to a Binary tree, various trees such as AVL tree, BST (Binary Search Tree), RBT tree, and so on are also utilized.

* Tree of Binary Search

Binary Search Tree (BST) is a binary tree extension that includes numerous optional constraints. In BST, a node's left child value should be less than or equal to the parent value, while the correct child value should always be higher than or equal to the parent's value.

* The AVL Tree

The AVL tree is a self-balancing binary search tree. The term AVL is given in honor of the inventors Adelson-Velshi and Landis. This was the first tree to achieve dynamic equilibrium. Each node in the AVL tree is assigned a balancing factor based on whether the tree is balanced or not. The node kids have a maximum height of one AVL vine.

* Red and Black Tree

Red-black trees are another type of auto-balancing tree. The red-black term is derived from the qualities of the red-black tree, which has either red or black painted on each node. It helps to keep the forest in balance. Even though this tree is not perfectly balanced, the searching process takes just O (log n) time.

* The N-ary Tree

In this sort of tree with a node, N is the maximum number of children. A binary tree is a two-year tree since each binary tree node has no more than two offspring. A full N-ary tree is one in which the children of each node are either 0 or N.

### 36. What are Binary trees?

A binary tree is a tree data structure made up of nodes, each of which has two offspring, known as the left and right nodes. The tree begins with a single node called the root.

Each node in the tree carries the following information:

Data

A pointing device indicates the left kid.

An arrow pointing to the correct child

### 37. What are the differences between the B tree and the B+ tree?

The B tree is a self-balancing m-way tree, with m defining the tree's order. Depending on the number of m, Btree is an extension of the Binary Search tree in which a node can have more than one key and more than two children. The data is provided in the B tree in a sorted manner, with lower values on the left subtree and higher values on the right subtree.

The B+ tree is an advanced self-balanced tree since every path from the tree's root to its leaf is the same length. The fact that all leaf nodes are the same length indicates that they all occur at the same level. Specific leaf nodes can’t appear at the third level, while others appear at the second level.

### 38. What are the advantages of binary search over a linear search?

In a sorted list:

* A binary search is more efficient than a linear search because we perform fewer comparisons. With linear search, we can only eliminate one element per comparison each time we fail to find the value we are looking for, but with the binary search, we eliminate half the set with each comparison.
* Binary search runs in O(log n) time compared to linear search’s O(n) time. This means that the more of the elements present in the search array, the faster is binary search compared to a linear search.

### 39. What is an AVL tree?

An [AVL (Adelson, Velskii, and Landi) tree](https://www.simplilearn.com/tutorials/data-structure-tutorial/avl-tree-in-data-structure" \o "AVL (Adelson, Velskii, and Landi) tree" \t "https://www.simplilearn.com/_blank) is a height balancing binary search tree in which the difference of heights of the left and right subtrees of any node is less than or equal to one. This controls the height of the binary search tree by not letting it get skewed. This is used when working with a large data set, with continual pruning through insertion and deletion of data.

### 40. Differentiate NULL and VOID

* Null is a value, whereas Void is a data type identifier
* Null indicates an empty value for a variable, whereas void indicates pointers that have no initial size
* Null means it never existed; Void means it existed but is not in effect

### 41. Do dynamic memory allocations help in managing data? How?

Dynamic memory allocation stores simple structured data types at runtime. It has the ability to combine separately allocated structured blocks to form composite structures that expand and contract as needed, thus helping manage data of data blocks of arbitrary size, in arbitrary order.

Simplilearn's PG Program in Data Science in partnership with Purdue University and in collaboration with IBM, is ranked #1 Post Graduate in Data Science program by ET. If you wish to ace data science, this program is just the one for you!

### 42. Name the ways to determine whether a linked list has a loop.

* Using hashing
* Using the visited nodes method (with or without modifying the basic linked list data structure)
* Floyd’s cycle-finding algorithm

### 43. List some applications of multilinked structures?

* Sparse matrix
* Index generation

### 44. Explain the jagged array.

It is an array whose elements themselves are arrays and may be of different dimensions and sizes.

### 45. Explain the max heap Data Structure.

It is a type of heap data structure where the value of the root node is greater than or equal to either of its child nodes.

### 46. How do you find the height of a node in a tree?

The height of the node equals the number of edges in the longest path to the leaf from the node, where the depth of a leaf node is 0.

### **. Explain what is data structure?**

The data structure is nothing but an entity where the data is perfectly aligned and can be manipulated as per the requirement. When we deal with data structure it is not just about one table of data but it is about different data sets and how well they are aligned with each other. Overall, it helps the data to be organized.

### **2. What are the basic operations performed on various data structures?**

The basic operations performed on data structures are as follows:

* ****Insertion**** - Adds a new data element in the data structure.
* ****Deletion**** - Removes a data element in a data structure.
* ****Searching**** - Involves searching for a specified data element in a data structure.
* ****Traversal**** - Processing all data elements present in it.
* ****Merging**** - Combines two similar data structures to form a new data structure of the same type.
* ****Sorting**** - Arranging data elements of a data structure in a specified order.

### **3. Explain what is a linked list?**

A linked list is nothing but a sequence of nodes. With this sequence, each node is connected to the following node. It forms a chain of data storage.

### **4. Explain the process of how do you reference all the elements in the one-dimension array in detail?**

To reference all the elements in the one-dimension array, we have to use an indexed loop. With the help of this, it executes from “0” to array size minus one. By following this process the loop counter will be able to refer to all the elements.

### **5. List out the areas where the data structure is applied?**

The data structure is a vital aspect while handling data. The following are specific areas where the data structure is applied:

* Numerical analysis
* Operating systems
* A.I.
* Database management
* Statistical analysis

The above are few areas where the data structure is applied and not limited to.

### **6. What is infix, prefix, and postfix in data structure?**

The way to write arithmetic expressions is known as notation. There are three types of notations used in an arithmetic expression, i.e., without changing the essence or output of expression. These notations are:

* ****Prefix (Polish) Notation**** - In this, the operator is prefixed to operands, i.e. ahead of operands.
* ****Infix Notation**** - In this, operators are used in between operands.
* ****Postfix (Reverse-Polish) Notation**** - In this, the operator is postfixed to the operands, i.e., after the operands.

The following table briefly tries to show the difference in all three notations −

|  |  |  |
| --- | --- | --- |
| ****Infix Notation**** | ****Prefix Notation**** | ****Postfix Notation**** |
| x + y | + x y | x y + |
| (x + y) ∗ z | ∗ + x y z | x y + z ∗ |
| x ∗ (y + z) | ∗ x + y z | x y z + ∗ |
| x / y + z / w | + / x y / z w | x y / z w / + |
| (x + y) ∗ (z + w) | ∗ + x y + z w | x y + z w + ∗ |
| ((x + y) ∗ z) - w | - ∗ + x y z w | x y + z ∗ w - |

## **Data Structures Interview Questions and Answers for Freshers**

### **7. Explain the terminology LIFO?**

LIFO stands for ****Last In First Out****.

This process describes how the data is accessed, stored, and then retrieved. So the latest data that is stored in the database can be extracted first.

**Subscribe to explore the latest tech updates, career transformation tips, and much more.**

### **8. Explain what is a binary tree?**

 It is one type of data structure that has two nodes, has left node and a right node. In a programming language, binary trees are considered to be an extension to the linked list.

### **9. Define what is a stack?**

The stack is considered as a data structure where the top layer element can be accessed. The data is stored in the stack and every time when data is stored, it pushes the data downwards which enables the users to access the latest data from the top layers.

### **10. Explain what is multidimensional arrays?**

Multidimensional arrays use multiple indexes in order to store data in the database. In a few scenarios, data cannot be stored using a single dimension index, in these scenarios multidimensional arrays are useful.

### **11. Explain whether a linked list is considered as a linear or non-linear data structure?**

This is purely determined on the requirement basis, a linked list can be considered as a linear data structure or a non-linear data structure. For example: If the linked list is used on storage, then the linked list is considered as a nonlinear data structure.

If linked lists are used against access strategies then they are considered as a linear data structure.

### **12. Explain how does dynamic memory allocation will help you in managing data?**

A dynamic memory allocation will help you effectively manage your data by allocating structured blocks to have composite structures that can be flexible, i.e. it can expand and can contract based on the need.

Also, they are capable of storing simple structured data types.

### **13. What is FIFO?**

FIFO in data terminology stands as “****First in, First Out****”.

This process defines or depicts how the data is stored inserted and accessed in a queue. Within this process, the data that is inserted at the beginning of the queue will only be extracted or accessed first.

### **14. Explain what is merge sort and how it is useful?**

A merge sort is nothing but a process where the data is divided and sorted to reach the end goal. Within this process, the adjacent elements are merged and sorted to create bigger elements. These sorted elements are gathered again and made the even bigger list. This process is continuous and repetitive until and unless they have nailed it down to a single sorted list.

### **15. List out all the advantages of a linked list?**

The important aspect or advantage of a linked list is that it is the perfect data structure where the data can be modified very easily. Also, it doesn’t matter how many elements are available on the linked list.

### **16. Explain the main difference between PUSH and a POP?**

The two main activities, i.e. Pushing and Popping applies the way how data is stored and retrieved in an entity. So if you check in detail, a Push is nothing but a process where data is added to the stack.  On the contrary, a Pop is an activity where data is retrieved from the stack. When we discuss data retrieval it only considers the topmost available data.

### **17. Can you explain with an example how does a variable declaration activity will consume or affect the memory allocation?**

The amount of space or memory is occupied or allocated depends upon the data type of the variables that are declared. So let’s explain the same by considering an example:  Let’s say the variable is declared as an integer type then 32 bits of memory storage is allocated for that particular variable.

So based on the data type of the variable, the memory space will be allocated.

|  |
| --- |
| Explore **[Core Java Programming Tutorial](https://mindmajix.com/java-tutorial" \o "Core Java Programming Tutorial" \t "https://mindmajix.com/_blank)** For More Information. |

### **18. Define the advantages and disadvantages of the heap compared to a stack?**

The advantages of the heap compared to a stack are listed below:

1. Heap is more flexible when compared to a stack
2. Memory space of the heap can actually be allocated and de-allocated as per the need.

On the contrary, the disadvantages of the heap compared to a stack is listed below:

1. The memory of the heap is slower when compared to the memory of the stack

### **19. Explain how new data can be inserted into the tree?**

The following are the steps that you need to follow to insert the data into the tree:

1. First of all, check whether the data is unique or not ( i.e. check whether the data that you are going to insert doesn’t already exist in the tree).
2. Then check if the tree is empty. If the tree is empty then all you need to do is just insert a new item into the root.  If the key is smaller than that of a root’s key then insert that data into the root’s left subtree or otherwise, insert the data into the right side of the root’s subtree.

### **20. Can you tell me the minimum number of nodes that a binary tree can have?**

A binary tree is allowed or can have a minimum of zero nodes. Further, a binary tree can also have 1 or 2 nodes.

### **21. Explain a little bit about the dynamic data structure?**

The nature of the dynamic data structure is different compared to the standard data structures, the word dynamic data structures means that the data structure is flexible in nature. As per the need, the data structure can be expanded and contracted. Thus it helps the users to manipulate the data without worrying too much about the data structure flexibility.

### **22. Define what is an array?**

While referring to array the data is stored and utilized based on the index and this number actually co-relates to the element number in the data sequence. So thus making it flexible to access data in any order. Within programming language, an array is considered as a variable having a certain number of indexed elements.

### **23. Can you tell me the minimum number of queues that are needed to implement a priority queue?**

The minimum number of queues that are needed is two. Out of which, one queue is intended for sorting priorities and the other queue is meant for the actual storage of data.

### **24. List out all different sorting algorithms that are available and state which sorting algorithm is considered as the fastest?**

The list of all sorting algorithms are below:

1. Quicksort
2. Bubble sort
3. Balloon sort
4. Radix sort
5. Merge sort

Out of the above sorting options, none of the sorting algorithms can be tagged as the fastest algorithm, because each of these sorting algorithms is defined for a specific purpose. So based on the data structure and data sets available the sorting algorithms are used.

### **25. Explain what is a dequeue?**

A de queue is nothing but a double-ended queue. Within this structure, the elements can be inserted or deleted from both sides.

### **26. Explain the process of how a selection sort works?**

A selection sort is a process where it picks up the smallest number from the entire data setlist and places it at the beginning. The same process is continued where the second position is already filled in. The same process is continued all the way until the list is completed. The selection sort is defined as a simple sort algorithm when compared to others.

## **Data Structures Interview Questions and Answers for Experienced**

### **27. Explain what is a graph?**

A graph is nothing but a type of data structure that has a set of ordered pairs. In turn, these pairs are again acknowledged as edges or arcs. These are used to connect different nodes where the data can be accessed and stored based on the needs.

### **28. Is it possible to implement a stack using a queue?**

Yes, you can implement a stack using two queues. Any data structure to act like a stack should have a push() method to add data on top and a pop() method to remove the top data.

### **29. How would you implement a queue using a stack?**

Using two stacks, you can implement a queue. The purpose is to complete the queue's en queue operation so that the initially entered element always ends up at the top of the stack.

* First, to enqueue an item into the queue, migrate all the elements from the beginning stack to the second stack, push the item into the stack, and send all elements back to the first stack.
* To dequeue an item from the queue, return the top item from the first stack.

### **30. Where is the LRU cache used in data structure?**

In data structures, you use LRU (Least Recently Used) cache to organize items in order of use, enabling you to quickly find out which item hasn't been used for a long time.

### **31. Which Data Structure is used to implement LRU cache?**

To implement the LRU cache, you should use two data structures: a hashmap and a doubly linked list.

A hashmap helps with the lookup of cached keys, and a doubly-linked list helps maintain the eviction order.

### **32. What is the difference between an array and a linked list?**

Array and Linked list are two ways of organizing the data in memory. The below table lists the various differences between the array and linked lists:

|  |  |
| --- | --- |
| ****Array**** | ****Linked List**** |
| An array is a sequence of elements of a similar data type. | A Linked list is a set of objects known as a node, where it internally consists of two parts, i.e., data and address. |
| It can be accessed irregularly using the array index. | Linked lists support random access. Only supports sequential access. |
| Array elements store in contiguous locations in memory. | New elements can be stored anywhere, and a reference is created for the new element using pointers. |
| In arrays, memory allocation is done during compile time. | In linked lists, memory allocation is done during runtime. |
| Array size must be defined at the time of declaration/initialization. | Linked list size grows when new elements are inserted or deleted. |

### **33. What are the advantages of Linked Lists over Array?**

The following are the advantages of Linked Lists over Arrays:

* ****Dynamic Data Structure**** - A Linked list is a dynamic data structure so that it can grow at runtime by deallocating and allocating memory. There is no need to present the linked list initial size.
* ****Insertion and Deletion Operations**** - Insertion and Deletion operations are easier in linked lists. You need to have to update the address present in the next pointer of a node.
* ****Implementation****- Data structures like queue, tree, and stack are easily implemented using the Linked list.
* ****No memory wastage**** - Efficient memory utilization can be achieved in the linked lists.

### **34. What are the applications of a stack in a data structure?**

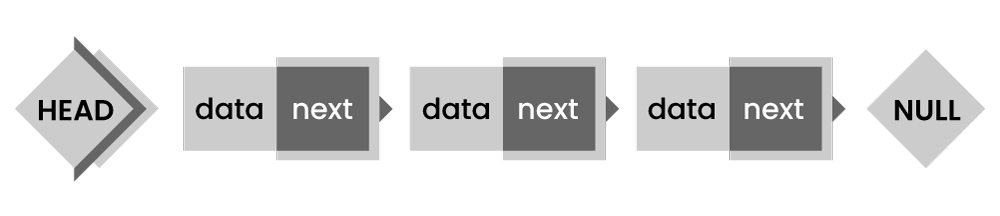
Following are some of the essential applications in a data structure:

* Expression evaluation
* Backtracking
* Function calling and return
* Memory management
* Checking parenthesis matching in an expression

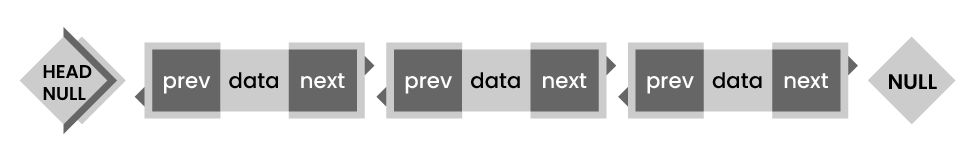
### **35. What are the different types of Linked List?**

The following are the different types of Linked Lists.

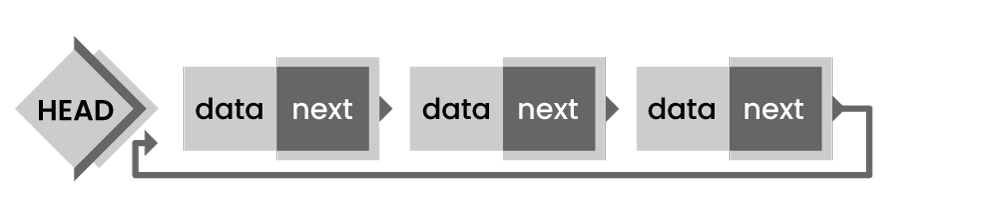
* ****Singly Linked List****- This is the most common type, and each node has data and a pointer to the next node.



* ****Doubly Linked List**** - In this type, the pointer is added to the previous node, either in a forward or backward direction.



* ****Circular Linked List**** - In this type, the last element is linked to the first element.



### **36. What is the difference between storage structure and file structure?**

The main difference between storage structure and file structure depends on the memory area that is accessed.

* ****Storage structure:**** It's a data structure representation in computer memory.
* ****File structure:**** It's a storage structure representation in the auxiliary memory.

### **37. What operations can be performed on a stack?**

Mainly the following operations are performed on a stack:

* ****Push operation:**** To add an item to the stack. If the stack is complete, then it is in an overflow condition.
* ****Pop operation:****It is used to remove an item from the stack. If it's an empty stack, then it is in underflow condition.
* ****isEmpty operation:**** If the stack is empty returns true, else false.
* ****Peek or Top operation:**** This returns the top element of the stack.

### **38. Name some applications of data structures?**

* Operating system
* Artificial Intelligence
* Statistical analysis
* Database management
* Compiler design
* Graphics
* Simulation
* Numerical analysis

### **39. Explain about Linked List Data Structure.**

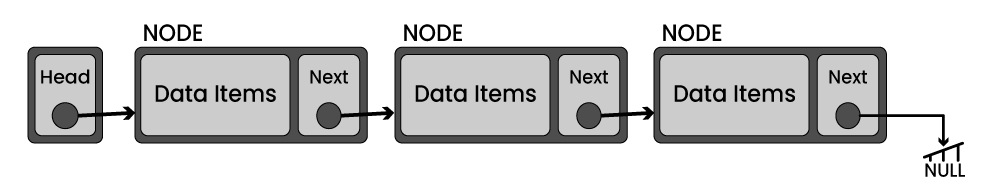
A linked list is a series of data structures connected via links. In simple words, it's a sequence of links that contain items. After the array, the linked list is the second most used data structure. The essential terms to understand the linked list are:

****Link**** - In a linked list, each link stores data called an element.

****Next****- In a linked list, each link is connected to the following link called next.

****LinkedList****-  It contains the connection link to the first link called first.

The below diagram depicts how nodes are connected in the Linked List:



Basic operations supported by a linked list:

* ****Insertion****- Inserts an element at the list beginning.
* ****Deletion****- Deletes an element at the list beginning.
* ****Display****- Displays the complete list.
* ****Search****- Searches an element using the given key.
* ****Delete****- Deletes an element using the given key.

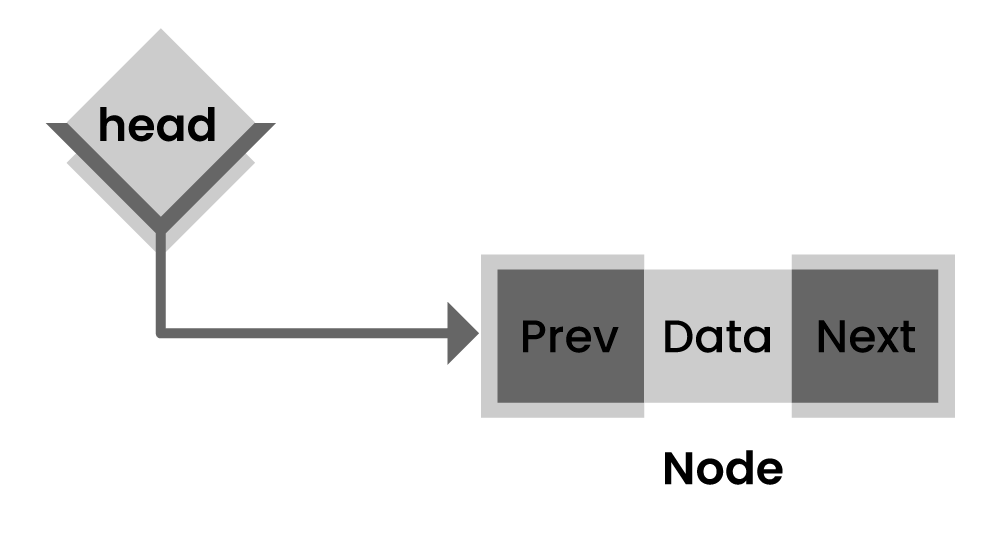
### **40. Are linked lists considered non-linear or linear data structures?**

It depends on where you plan to use Linked lists. You can consider a linked list for both non-linear and linear data structures. When used for data storage, it is regarded as a non-linear data structure. When used for access strategies, it is considered a linear data structure.

### **41. What is a doubly-linked list used for?**

A doubly linked list is one of the complex types of the linked list, where a node contains a pointer to the previous and the next node in the sequence. It consists of three parts: node data, a pointer to the next node in sequence (next pointer), a pointer to the previous node (previous pointer)

The below diagram depicts the working of doubly linked list:



Some of the real-time applications where doubly-linked lists used are navigation systems and browsers (when both back and front navigation is required).

|  |
| --- |
| →  Explore Frequently Asked **[Core Java Interview Questions](https://mindmajix.com/core-java-interview-questions" \o "Core Java Interview Questions" \t "https://mindmajix.com/_blank)** |

### **42. What is a queue in data structure?**

A queue is a linear data structure that supports a specific order in which operations are performed. The order is FIFO (First in First Out) methodology, i.e., data items stored first will be accessed first. Unlike stack, the queue is open at both ends, and one end is always used to insert data and another one to remove data.

The basic operations associated with queues -

* ****Dequeue**** - To remove an item
* ****Enqueue**** - To insert an item
* ****isempty()**** − Confirms whether the queue is empty.
* ****isfull()**** − Confirms whether the queue is full.
* ****peek()**** − Gets the element at the front of the queue without removing it.

### **43. List a few queue data structure applications.**

As the name suggests, the queue is used whenever you need to manage a group of objects in the order FIFO. A few of the queue data structure applications are listed below:

* Serving requests on a single shared resource, like CPU task scheduling, printer, etc.
* Handling interruptions in real-time systems.
* Buffers in apps like CD player and MP3 media players
* In maintaining a playlist in media players, like adding or removing songs.

### **44. What is the difference between stack and heap?**

Both stack and heap are used for memory needs. The stack is primarily used to save the method execution order and local variables, and always follow the LIFO order.

Whereas heap is used for dynamic allocation and deallocation of memory blocks. It stores objects in Java. Memory allocated to the heap lives until one of the following events occurs:

* Memory free
* Program terminated

The size of heap memory is more when using recursion when compared with the stack, as it quickly fill-ups stack memory.

### **45. Name a few graph data structure applications.**

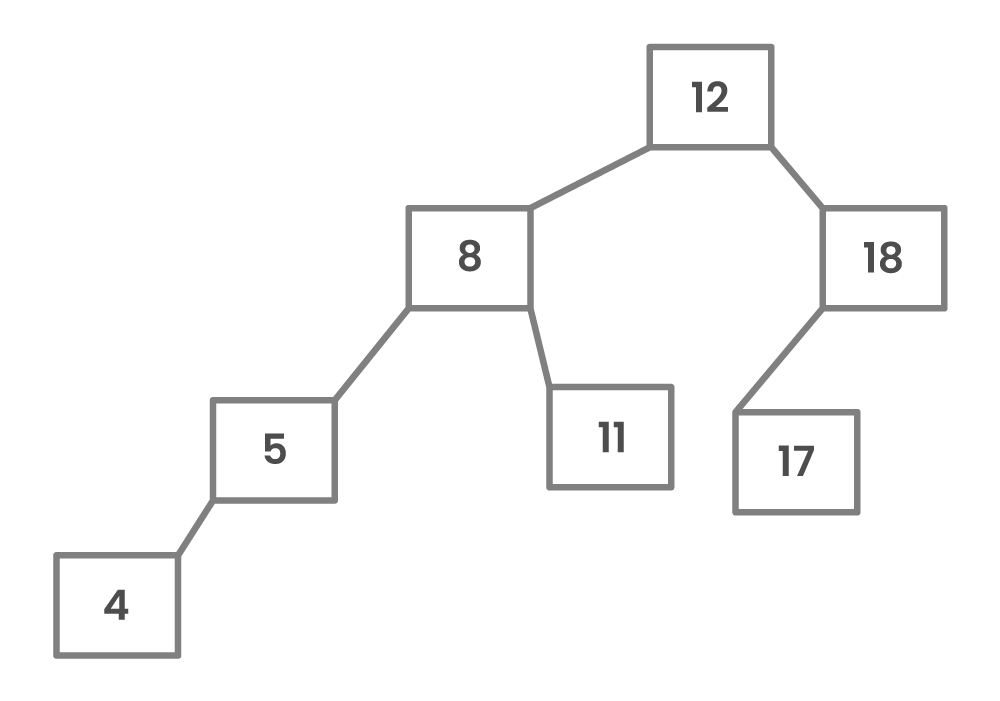
Applications of graph data structures in real-time are:

* Social graphs
* Path optimization algorithms
* Recommendation engines
* Scientific computations

### **46. What is an AVL tree?**

An AVL (Adelson, Velskii, and Landi) is a self-balancing binary search tree where the variation of heights of the right and left subtrees of any node is not more than one.

An example of an AVL tree:



A tree is balanced if the balance factor of each node is between -1 to 1. Or else, the tree is unbalanced and needs to be balanced.

### **47. How do you detect a loop in a linked list?**

* Using Floyd's cycle-finding Algorithm
* Using hashing
* Using the visited nodes method

### **48. What is a Jagged array?**

Jagged arrays are a particular type of arrays used for storing rows of data of varying lengths to improve efficiency when working with multidimensional arrays.

### **49. What is the max heap in the data structure?**

A max heap in a data structure is a complete binary tree where each internal node's value is greater than or equal to that node's children's values.

### **50. How to find the height of a node in a tree?**

You can find the height of a binary tree using a recursive Depth-First Search (DFS) algorithm, as shown below:

* ****Base case:**** If there is no node, return 0.
* ****Else:**** If there are 1 or 2 children, return the peak of the height of the left and right subtrees, plus 1 to account for the current node.

### **51. List the types of trees.**

* General Tree
* Binary Tree
* Forests
* Expression Tree
* Binary Search Tree
* Tournament Tree

### **52. Which data structures do you use in DFS and BFS algorithms?**

* In the DFS algorithm, you use the Stack data structure.
* In the BFS algorithm, you use the Queue data structure.