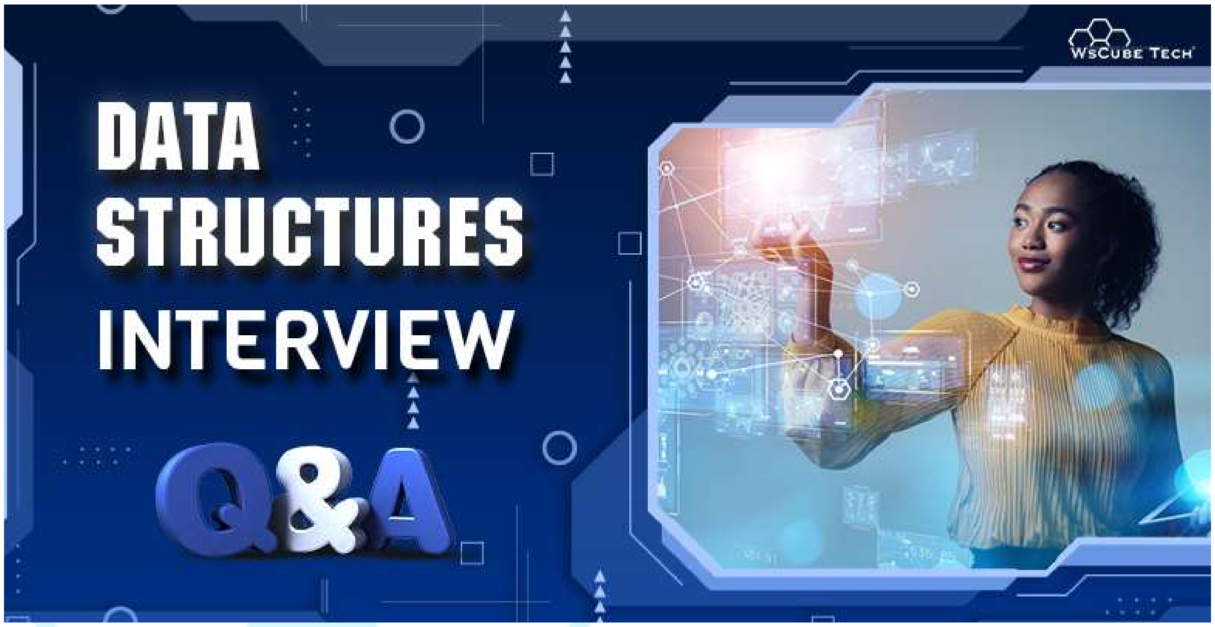
# Top 50 Data Structures Interview Questions And Answers PDF



## Data Structure Interview Questions for Freshers

### 1. What is a data structure?

A data structure is a mechanical way to organise, align, and manipulate data as per requirements. It is not restricted to putting data in a table but deals with different datasets and how well they are aligned.

The aim is to ensure that data can be organised and accessed efficiently. Data organisation determines how a program performs. Moreover, data dependency and relationships between two or more datasets play a crucial role in data structures.

While designing code, we need to pay utmost attention to how data is structured because incorrectly structured or inefficiently stored data can hamper the overall performance of the code.

### 2. What are the applications of data structures?

Data structures are applied across multiple industries and domains as algorithms are the primary requirements in every business. Here are the major applications of data structures:

- Numerical and statistical analysis

- Compiler design
- Artificial intelligence
- Database management
- Graphical processing
- Lexical analysis
- Blockchain
- Decision making
- Operating system design
- Simulation

## 3. Describe the types of data structures.

**Data structures are divided into two categories:**

- Linear data structure
- Non-linear data structure

### Linear Data Structure

It includes data elements that are arranged sequentially or linearly. Also, each element is connected to its previous or next adjacent element.

#### Types of linear data structure

- **Static Data Structure:** It has a fixed memory size, and it's easier to access data elements. An example includes Array.
- **Dynamic Data Structure:** Size is not fixed and can be updated randomly. Examples include Queue and Stack.

#### Common linear data structures

- **Array Data Structure:** All the elements or values are of the same type.
- **Stack Data Structure:** It follows LIFO (Last in, first out) order to perform operations, so the last element to be stored will be the first to be retrieved.
- **Queue Data Structure:** It follows FIFO (first in, first out) order for operations, so the item stored first will be the first to be accessed.
- **Linked List Data Structure:** Elements in this data structure are connected through a series of nodes, each containing data items and the address of the next node.

**Non-linear Data Structure**

Here, data elements are not arranged in a sequence, and all elements can't be traversed in a single run. Common non-linear data structures are:

- **Tree Data Structure:** A hierarchical data structure where elements are placed in a tree-like structure. It contains a central node, structural nodes, and sub-nodes connected via edges. Tree-based data structures are Binary Tree, Binary Search Tree, B- Tree, B+ Tree, AVL Tree, and Red-Black Tree.
- **Graph Data Structure:** Consists of vertices and edges, and a pair of nodes is connected through edges. It is used to solve complex problems.

## 4. Explain the difference between a File Structure and a Storage Structure.

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.

| **File Structure** | **Storage Structure** |
|---|---|
| Represent data in auxiliary memory, such as disk and pen drive. | Data is stored in the computer memory, i.e., RAM and disk. |
| Data is intact until deleted manually. | Once the function using the data gets executed, it is deleted from the memory. |
| Standard storage policies. | Customized storage policies. |
| Compatibility with external apps is low. | Highly compatible with external apps. |

It is one of the most asked data structure interview question for freshers.

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

A stack data structure represents the state of an application at a specific point in time. It consists of a series of elements arranged in a linear structure like a real physical stack or pile where you add or removes items from the top.

This linear data structure follows a particular order, LIFO (Last In First Out) and FILO (First In Last Out), to perform operations.

Elements are arranged in a sequence, so the item added last will be restricted first. A real-life example is a stack of clothes where the cloth placed on the top, or the last to be added, will be the first to be picked.

**Applications of a stack data structure are:**

- Temporary storage during recursive operations
- Reversing a string
- Redo and Undo operations in doc editors
- Parenthesis matching
- Syntax parsing
- Postfix, prefix, and infix expressions
- Backtracking

## 6. What operations can be performed on a stack?

Common operations that can be performed in a stack data structure are:

### push():

It inserts a new element at the top of the stack. However, before we insert a value, it is important to verify if TOP=MAX–1. If the stack is filled, no more insertion is possible, and you can see an OVERFLOW message if y try to put an element in the already-filled stack.

### pop():

Its basic function is to remove an element from the top of the stack. Before you retrieve an item, verify if TOP=NULL. If the stack is empty, no further deletions are allowed, and you will get an UNDERFLOW message if you try to remove an element from an empty stack.

**peek():**

It returns a value of the top-most element in the stack but doesn't remove it from the data elements collection.

**isFull():**

To check if the stack has reached its maximum capacity of data insertion.

**isEmpty():**

Checks if the stack is empty or not.

**size():**

To find the number of elements in a stack.

## 7. Explain what multidimensional arrays is.

A multidimensional array spans across more than one dimension, which means it has more than one index variable for each storage point. It has multiple layers and is primarily used where data can't be stored or represented in one dimension.

Basic multidimensional arrays are 2D and 3D arrays. It's technically an array of different arrays. A 2D array, also known as a matrix or a table, consists of rows and columns.

## 8. What is a binary search tree?

It is a type of data structure that stores data elements efficiently. Here, each node stores a key and a value. The former is used to access an item, while the latter is used to find if the item is present or not.

A key is constant and defines the dataset (e.g., colour, gender, price), while a value is variable and belong to any of the dataset (e.g., red, female/male, 100).

**A binary search tree has a specific order to place elements and is known for three qualities:**

- The values of elements of nodes in the left sub-tree are less or equal to that of the root nodes.
- Moreover, the values of the right root nodes are higher or equal to the root node.
- Both the left and right sub-tree are two individual binary search trees at all points in time.

**Major applications of binary search trees are:**

- For indexing and multi-level indexing
- For organising and sorting data
- For implementing different search algorithms

## 9. What is a linked list data structure?

A linked list data structure is a sequence of elements that are not stored in adjacent memory locations. The elements in this data structure use pointers to form a chain. Think of it as a series of linked nodes (each element or node representing a separate object) connected by paths or links. Every node has two items- a data field and a reference to the next node.

Each link is an entry into the linked list, and every entry point, also known as the head, points to the next node. In an empty list, the head is a null reference, with the last node consisting of a reference to the null.

The order to add nodes to the list is based on the order in which they are created. A linked list data structure doesn't have a fixed number of nodes, and the list can grow or shrink as per demand.

**Some of the key applications of a linked list data structure are as follows:**

- Dealing with the unknown number of elements
- Implementing queue, stack, graphs, and binary trees data structures
- Accessing elements randomly is not required
- Round-robin scheduling for an operating system
- Real-time computing when time predictability is important
- Forward and backward operation in the browser
- Interesting items in the middle of the list
- Dynamic management for Operating System memory

## 10. Are linked lists linear or non-linear data structures?

Linked lists can be linear and non-linear data structures based on their usage. If it's used for storage, it's non-linear, but if it's used for access or retrieval strategies, it's considered a linear data structure.

## 11. What are the pros of a linked list over an array? In which scenarios do we use a linked list and when array?

The benefits of a linked list over an array are:

### a) Dynamic Data Structure

The linked list is a dynamic data structure, so there is no need for an initial size. It grows and shrinks at runtime through memory allocation and deallocation.

In an array, size is limited because the number of elements is stored statistically in the memory.

### b) Insertion and Deletion

Node insertion and deletion are easier as you just have to update the address present in the next pointer.

However, it is expensive in an array because you need to create room for new elements by shifting existing elements.

### c) Implementation

Implementing stack and queue data structures using a linked list is easier.

With an array, implementing data structures is complex.

### d) Memory Usage

A linked list can be decreased or increased based on the demand of the program, and memory is allocated only when needed, so there is no memory wastage.

An array cause more memory wastage. For example, if we store five elements in an array when the size is 10, then the remaining space will be wasted.

**A linked list is used when:**

- A large number of operations are added or removed.
- You want to insert an item in the middle of a list, like implementing a priority queue.
- You already know the upper limit of a number of elements.

**Some common cases when an array is preferred over a linked list:**

- You need to randomly access or index data elements.
- You need to iterate through all elements quickly.
- You already know the number of elements in an array, so you can allocate the correct amount of memory.

- In case memory is a major concern, arrays use less memory than linked lists because each element in an array is data, whereas each node in a linked list requires data and one or more pointers to other elements.

In a nutshell, it comes to time, space, and ease of implementation while deciding between a linked list and an array.

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

The doubly-linked list is a double-ended complex linked list with two links in a node. One of them connects to the next adjacent node in the sequence, while the second one connects to the previous node. This enables traversal in both directions. Some of the examples of a doubly-linked list are:

- Browser cache with back-forward visited pages
- Song playlist with next and previous buttons
- Undo and redo functionality

## 13. What is FIFO?

FIFO stands for First in, First out order, representing the way and order in which data is accessed. The data element stored first in the list is the first entity to be removed from the data structure.

## 14. What is the working of LIFO?

LIFO stands for Last in, First out order and directly corresponds to how data is accessed and modified. The data element stored last in the stack is the first one to be worked on at any point in time. If the first one doesn't need to be accessed, you need to retrieve the data after that element.

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

### a) Row-Major Order

Rows of a 2D array are stored contiguously. The first row is stored in the memory, then the second row of the array, then the third, and so on until the final row.

### b) Column-Major Order

All the columns are stored in the same order. In the beginning, the first column is stored in the memory, then the second column, and so on, until the last column is saved in the memory.

So, these are the common data structure interview questions for freshers.

# Data Structures and Algorithms (DSA) Interview Questions

## 16. What is an algorithm?

An algorithm shows a step-by-step process to solve a problem or manipulate data. It is a set of instructions to be executed in a specific order to get the desired results.

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

A problem can be resolved in multiple ways with different algorithms. Using algorithm analysis, we can estimate the required resources to solve a particular computation problem. We can determine the amount of time and space resources needed.

The time complexity determines the amount of time required for an algorithm to run as a function of input length, and the space complexity determines the amount of space or memory consumed by an algorithm.

## 18. What are different sorting algorithms? Which is the fastest of them?

**Different sorting algorithms are:**

- Quicksort
- Bubble sort
- Radix sort
- Balloon sort
- Merge sort

Among all the options available, putting one on the podium for enhanced and fastest performance is neither fair nor possible. Each sorting algorithm serves a specific purpose, and its performance varies according to the data, the way it's stored, and the reaction once the algorithm processes the data.

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

Asymptotic analysis of an algorithm is a technique to determine its run-time performance according to the mathematical units.

It is used to find the best case (Omega Notation, Ω), worst case (Big Oh Notation, O), and average case (Theta Notation, θ) performances of an algorithm. It is not a deep learning method but an essential tool for programmers to analyse the efficiency of an algorithm.

## 20. What is a queue data structure?

A queue data structure supports systematic operations, i.e., elements are accessed and manipulated in a specific order. It follows FIFO (First in, First out) method, so elements are added to the queue one after the other from the rear end and are removed or processed on the front end.

It is similar to the literal queues in the real world. Unlike stack, it is open at both ends, with one end being used to insert elements and another one to remove them. Queues are commonly used when users want to store items for a long period.

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

A priority queue is a type of abstract data resembling a normal queue, but here, each element is assigned a priority value. The order of elements in the priority queue determines their priority.

So, the elements with higher priority are processed first. In case two or more elements have the same priority, they are processed in the order they appear in the queue.

**Some real-life applications of the priority queue are:**

- Huffman code for data compression
- Graph algorithms like Prim's Minimum spanning tree
- Robotics to plan and execute tasks based on priorities.

Be prepared for this type of interview questions in data structures, as these are often asked to both freshers and experienced professionals.

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

A queue is used to manage a group of elements in FIFO (First in, First out) order. A few of its applications are:

- Call management in call centres
- Buffers in MP3 players and CD players
- Handling interruptions in real-time systems
- In graphs for a breadth-first search algorithm

- Waiting lists or serving requests for a single shared resource like CPU, image uploads, printer, etc.
- Operating system: CPU scheduling, job scheduling operations, and Disk scheduling.
- Manage a playlist in media players, for example adding or removing a song
- Asynchronous transfer of data

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

Different operations that can be performed on queues are:

- **dequeue()** removes an element from the queue. If the queue is empty, you get UNDERFLOW notification.
- **enqueue()** adds an element to the queue. If the queue is full, it will be an overflow condition.
- **isEmpty** is to check if the queue is empty or not.
- **init()** initialises the queue.
- **front** is used to find the value of the first element without removing it.
- **rear** returns the last or rear element from the queue.

## 24. What is a Deque?

A deque is a double-ended queue. It is like an array of items, but rather than adding or removing elements from only one end, a deque allows elements to be inserted at either end.

This feature makes deque suitable for various tasks, such as scheduling, keeping track of inventory, and handling a large amount of data. Moreover, it can be used as a stack and queue and performs better than stacks and linked lists.

## 25. What are the different types of deque? What are its applications?

There are two main types of deque:

### a) Input Restricted Deque

Here, insertion is performed at one end, but deletion is performed at both ends.

### b) Output Restricted Deque

Insertion is performed at both ends while deletion is performed at one end.

**Some real-life applications of a deque data structure are:**

- To store web browser history
- As it supports all data structure operations, it can be used as a stack and queue.
- Operating system job scheduling algorithm

## Interview Questions on Data Structures For Experienced Professionals

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

Some major advantages of heap over stack are as follows:

- A heap is more flexible than a stack.
- With a heap, it is possible to allocate or de-allocate memory space dynamically.
- The heap is used to store data elements in Java, while the stack is used for variables.
- When you use recursion, heap memory size is more, but in the case of stack memory, it fills up quickly.
- Objects in a heap are visible to all threads, whereas variables are stored as private memory in the stack and are visible to only the owner.

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

Push and pop are the two key data structure activities that stand for Pushing and Popping. These operations signify the way data is stored and retrieved in a stack.

**Push-** It denotes that a user is adding an element to the stack.

- It requires the name of the stack and the value of the entity.
- In case the stack is filled but you still use the ‘Push’ command, you’ll get an ‘overflow’ error, indicating that the stack can no longer accommodate another element.

**Pop-** It is an activity to retrieve or pull elements from the stack.

- It only needs the name of the stack.
- When you try to pull an element from an empty stack, you see an ‘underflow’ error.

When we talk about adding or removing an element, it refers to the top-most available data.

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

Merge sort is based on the divide and conquer method. Here, the data is divided and sorted to attain the end goal. The adjacent data elements are merged and sorted in each iteration to create bigger sorted lists, which are combined recursively until there is a single sorted list.

## 29. How is a variable stored in memory when using data structures?

A variable is stored according to the required amount of memory. The steps to store a variable in memory are as follows:

- It starts by assigning the amount of memory that is needed.
- It is stored as per the data structure.
- Dynamic allocation ensures enhanced efficiency, and one can access units based on their real-time requirements.

## 30. What is dynamic memory management?

In the dynamic memory management technique, storage units are allocated according to the requirements. Dynamic memory allocation allows storing each data structure either separately or together to form entities known as composites. You can work on these entities whenever required.

## 31. What is a postfix expression?

Post-fix expression consists of operators and operands, and every operator precedes the operands. With the concept of operator precedence, it eliminates the need for using parentheses or sub-expressions.

## 32. What is a binary tree?

A binary tree is used to organise data, making data retrieval and manipulation efficient. This data structure has two nodes, left and right, called leaves and nodes. Leaves represent data, while nodes represent relationships between leaves.

Every node has two children, also known as siblings, with each child having one parent. The parent node is closest to the root tree. If a node is deleted from the tree, it gets deleted from its child and parent. According to programming language, a binary tree is an extension to the linked list.

## 33. What are the applications of binary trees?

**Common applications of a binary tree are:**

- In data compression methods in the Huffman coding tree
- Display hierarchical data
- Search for a specific element
- Put priority queues into action
- Implement file systems where every individual node represents a directory or file.
- Implement decision trees, a machine learning algorithm for classification and regression analysis.
- In editing apps of spreadsheets and Microsoft Excel
- Encoding and decoding operations
- Implement efficient sorting algorithms

## 34. What is the AVL tree data structure?

An AVL tree, named after its inventors Adelson, Velskii, and Landi, is the highest-balancing binary search tree. It compares the heights of the left and right subtrees of nodes and ensures that the difference is less than or equal to one. The difference is called the Balance Factor. It is used for a large set of data with continuous pruning through insertion and deletion.

## 35. List the operations, rotations, and applications of an AVL tree.

An AVL tree is used to perform the following operations:

### a) Insertion

Although insertion in an AVL tree is the same as in a binary search tree, it may cause a violation of the property. Hence, the tree must be balanced, for which rotations can be used.

### b) Deletion

It is also the same as in the binary search tree. However, deletion can lead to disruption in the tree's balance, so rotations are used to balance it.

To balance an AVL tree, it performs the following rotations:

### c) Left Rotation

When inserting a node into the right subtree of the right subtree cause the tree to be unbalanced, users need to perform a single left rotation.

### d) Right Rotation

If the tree is unbalanced because of insertion in the left subtree of the left subtree, it requires a single right rotation.

### e) Left-Right Rotation

It is the extended version of the single rotation and is also known as double rotation. When node insertion is into the right subtree of the left subtree, a left rotation is followed by a right rotation to balance the tree.

### f) Right-Left Rotation

When node insertion is done into the left subtree of the right subtree, a right rotation is followed by a left rotation.

**Here are a few real-life applications of an AVL tree:**

- In-memory sets and dictionaries
- Database applications with fewer insertions and deletions but frequent data lookups.
- Applications that need enhanced searching.

## 36. What is the difference between the B tree and the B+ tree?

### a) B Tree

It refers to a self-balancing m-way tree, where 'm' signifies the tree's order. It is an extension of the binary search tree where nodes can have more than two children and one key. The data is sorted in a B tree with lower values placed on the left subtree while higher values are placed on the right subtree.

**b) B+ Tree**

It refers to an advanced self-balanced tree where every path from the root to the leaf is of the same length. This shows that all the paths occurred at the same level. So, when all leaf nodes appear on the second level, no other lead node can appear on the third level.

## 37. What are the applications for B-trees?

Before discussing the real-life applications of B-tress, let's have a look at some of its properties:

**The following are the key properties of a B-tree data structure:**

- All the leaves are at the same height.
- Minimum degree 't' refers to a B-tree, and its value is determined by the disk block size.
- Each node must have at least t-1 keys. The root is the only exception to this, which must contain at least one key.
- In B-tree, node insertion happens at Leaf Node.
- The number of children of a node= number of keys in the node plus one.
- Unlike the binary search tree, which grows and shrinks from downward, B-tree grows and shrinks from the root.
- Nodes, including root, can't have more than 2*t-1 keys.
- Keys are sorted in ascending order. A child between k1 and k2 consists of all keys ranging from k1 to k2.

**The applications of B-tree are:**

- In CAD systems, to sort and search geometric data.
- In the case of a large database, to access data stored on a disc
- Most servers use B-tree
- To achieve multilevel indexing
- To search data in less time
- In areas such as computer networks, natural language processing (NLP), and cryptography.

It is among the top interview questions in data structures.

## 38. What is a graph data structure?

It is a non-linear data structure with nodes or vertices connected by a set of ordered edges. These edges are lines or arcs connecting two nodes in a graph so that data can be accessed and stored based on requirements.

## 39. What are the applications of a graph data structure?

**Real-time applications of a graph data structure are:**

- Social network graphs to track users' data.
- Google Maps for building transportation systems
- Transport grids with stations represented as vertices and routes as edges
- Dijkstra algorithm to determine the smallest path between two or more nodes
- Utility graph or water or power with vertices representing connection points and edges of the wires.
- Operating System, in the Resource Allocation Graph where each process and resource are represented as vertices
- World Wide Web, where web pages are considered as vertices
- Computer science to represent the flow of computation

# Advanced Data Structure Interview Questions

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

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

| | |
|---|---|
| Uses Queue data structure to find the shortest path. | Uses the Stack data structure to find the shortest path. |
| Need to walk through all the nodes on the same level to go to the next level in BFS. | Begins at the root node and then proceeds as far as possible through all the nodes to reach the node with no unvisited nearby nodes. |
| It is slower. | Faster compared to BFS. |
| Performs better if the target is closer to the source. | Performs better if the target is farther from the source. |
| Needs more memory. | Needs less memory. |
| No backtracking is possible. | As it is a recursive algorithm, it employs the backtracking. |
| Based on the FIFO principle (First In First Out). | Based on the LIFO principle (Last In First Out). |

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

The height of a node is equal to the number of edges in the longest path to the leaf from that node. Here, the depth of a leaf node is zero.

### I) List the types of trees.

#### a) General Tree

A generic tree with no constrained hierarchy. Here, nodes can have an endless number of offspring and the rest of the trees are subsets of the tree.

#### b) Binary Tree

It is the most commonly-known tree. Here, each parent has a minimum of two offspring, which are called the left and right youngsters. When a binary tree has certain features and limitations, other trees, such as RBT, AVL, and binary search trees, are used.

#### c) Binary Search Tree

Also known as BST, it is an extension of a binary tree with various optional constraints. The left child value of a node should be less than or equal to the parent value, and the correct child value should be higher than or equal to the parent value.

#### e) Red and Black Tree

It is an auto-balancing tree with red or black nodes. Hence, the name Red-Black Tree. It keeps the forest balanced. The tree might not be balanced, but the searching process takes O(log n) time.

#### f) N-ary Tree

N represents the maximum number of children. In this tree, every node's children are either 0 or N.

## 42. How does bubble sort work?

A bubble sort is a commonly-used sorting method appalled to arrays where adjacent elements are compared to each other. Also, values are exchanged as per the order followed in the arrangement. It is named bubble sort as it involves exchanging elements like bubbles floating in the water and larger entities sinking to the bottom.

## 43. What is a Jagged array?

A jagged array is used to store rows of data elements of varying sizes and dimensions. It enhances efficiency while working with multidimensional arrays.

## 44. How do you find a loop in a linked list?

There are multiple ways to detect a loop in a linked list. One can use the following methods:

- Hashing
- Floyd's cycle-finding algorithm
- Visited nodes method

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

- A binary search is comparatively more efficient than a linear search as it involves fewer comparisons.
- Linear search allows you to eliminate only one element per comparison each time you fail to find the value you want. However, a binary search allows you to eliminate half the data set with each comparison.
- The binary search runs in O(log n) time while the linear search runs in O(n) time. So, the more elements in the search array, the faster the binary search is.

## 46. What is a heap data structure?

Heap is a unique tree-based non-linear data structure with a complete binary tree. It means that all the levels are filled except possibly the last one, which consists of all the elements, and all the nodes are left-justified.

**Broadly, a heap is of two types:**

- **Max-Heap:** The value of the root node must be greatest from all other keys of its children node. The same is recursively true for all sub-trees in that binary tree.
- **Min-Heap:** The value of the key at the root node must be the smallest among all keys of the children node. The same stands recursively true for all other sub-trees in that binary tree.

## 47. Define Red-Black Tree.

A red-black tree is a binary tree invented in 1972 by Rudolf Bayer and was dubbed as "symmetric binary B-tree”. It is a self-balancing binary tree that stores data in two’s complementary binary formats.

Each node is either red or black, and by comparing these colours on a path from root to leaf, the tree ensures that the path is not more than twice as long as others, making it balanced. It can be used to store any kind of data represented as a set of values.

Although it is similar to a binary tree, there is one difference. The read-black tree is faster to access, making it suitable for storing a large amount of data.

## 48. What is a Trie data structure?

Trie stands for ‘Retrieval’ and is also known as the prefix tree or digital tree. It stores a set of strings in the form of a sorted tree. Let’s say strings range from letters ‘a to z’ of the English alphabet, so each trie node can’t have more than 26 points.

Also, every individual node has the same number of pointers as alphabet characters. This data structure can check the work in the dictionary by its prefix. Moreover, it allows you to print all words alphabetically, which is an advantage over hashing. It ensures efficient prefix search or auto-complete.

The key to which each node is connected is based on its position in the Trie data structure. You can add or find strings in O(L) time, where L = Length of a single word. This is a faster way than BST and hashing, considering the way it is implemented. You don’t need to compute a hash function or manage collisions like in open addressing or separate chaining.

## 49. Explain the Segment Tree data structure.

As the name suggests, a segment tree data structure stores segments or intervals. It is a binary tree comprising nodes representing intervals. It is used in case there are multiple changes in array elements or various range queries on an array.

**Its operations include:**

- **Building Tree:** To create a data structure and initialise the segment tree variable.
- **Querying Tree:** To run a range query on an array.
- **Updating the Tree:** To change the tree by updating the value of an array at a certain point over an interval.

**A few real-time applications of segment tree data structure are:**

- Computation geometry
- Pattern recognition and image processing
- Storing segments randomly
- Calculate range max/min, prefix sum/product, range sum/product, etc.
- Static and dynamic RMQ
- Geographic information system