1. Which data structure in Python follows the Last In, First Out (LIFO) principle?
   1. Queue
   2. Stack
   3. Linked List
   4. Dictionary Answer: B) Stack
2. What is the time complexity of the quicksort algorithm in the best-case scenario?
   1. O(n)
   2. O (n log n)
   3. O(n^2)
   4. O (log n)

Answer: B) O (n log n)

1. Which search algorithm does not require the list to be sorted?
   1. Binary Search
   2. Linear Search
   3. Depth-First Search
   4. Breadth-First

Search Answer: B) Linear Search

1. Which data structure uses the FIFO (First In, First Out) principle?
   1. Stack
   2. Linked List
   3. Queue
   4. Tree

Answer: C) Queue

1. What is the worst-case time complexity of the insertion operation in a binary search tree (BST)?
   1. O (1)
   2. O (log n)
   3. O(n)
   4. O(n^2)

Answer: B) O (log n)

1. Which sorting algorithm has a time complexity of O(n^2) in the worst-case scenario?
   1. Merge Sort
   2. Quick Sort
   3. Bubble Sort
   4. Insertion Sort

Answer: C) Bubble Sort

1. Which of the following is not a type of tree traversal?
   1. Preorder
   2. Post order
   3. Midorder
   4. Level order

Answer: C) Midorder

1. Which data structure is typically used to implement a LIFO (Last In, First Out) mechanism?
   1. Queue
   2. Heap
   3. Stac Set

Answer: C) Stack

1. Which of the following is not a searching algorithm?
   1. Depth-First Search (DFS)
   2. Binary Search
   3. Dijkstra’s Algorithm
   4. Bubble Sort

Answer: D) Bubble Sort

1. What is the purpose of dynamic programming?
   1. Solving problems by breaking them down into smaller subproblems
   2. Solving problems through recursion only
   3. Solving problems using iterative methods
   4. Solving problems using large datasets

Answer: A) Solving problems by breaking them down into smaller subproblems

1. In a binary tree, what is the maximum number of nodes at level 3?
   1. 4
   2. 8
   3. 16
   4. 32

B:8

5 marks question

1. Write an algorithm to convert an in-fix expression to a post-fix expression
2. Create an empty stack.
3. Scan the infix expression from left to right.
4. If the scanned character is an operand, append it to the postfix expression.
5. If the scanned character is an operator, then:
   * If the stack is empty or the top element of the stack is a left parenthesis, push the operator onto the stack.
   * Otherwise, pop the operator from the stack and append it to the postfix expression. Then push the scanned operator onto the stack.
6. If the scanned character is a right parenthesis, pop all the operators from the stack and append them to the postfix expression until the top element of the stack is a left parenthesis. Then pop the left parenthesis from the stack.
7. Then all the remaining operators left in the stack will be popped out and will be directly append to the postfix expression

Or

What is stack? Write an algorithm/python code to determine the sum of all elements of an array using recursion 5

Stack is a linear data structure and ordered collection of items where the addition of new items and removal of items always take place at the same end i.e., from the top. It follows LIFO last in first out.

Operations of stack

Push (): adds the elements to the stack

Pop (): removes the elements from the stack

peek or top (): gives you the element present in the top

is empty (): to check the stack is empty or not

is full ():to check the stack is full or not.

Just take an example. Let's say there is multiple words in a single sentence. You remove some of the words, and the words stored in the stack one by 1Then you press central Z to bring back those words which you were deleted That words will be removed from the stack one by one from the top

In python stack is not a build in data structure

We can implement stack using modules or list the modules can be collection or queue

Disadvantages of Stack

the insertion and deletion of elements can be performed at only one end

Elements gate inserted first has to wait for the longest time to get pop off

Any object can be added only at the top

Only one element at the top position can be related at a time

If we want to delete only the last element of the stack, then we have to unnecessarily delete the top of the elements to reach to the last element

The removal of items can only be start from the top

def array\_sum\_recursive(arr):

    # Base case: If the array is empty, return 0

    if not arr:

        return 0

    # Recursive case: Sum the current element with the sum of the rest of the array

    return arr[0] + array\_sum\_recursive(arr[1:])

# Example usage:

my\_array = [1.1, 3, 9, 49, 6.9]

result\_sum = array\_sum\_recursive(my\_array)

print("Array:", my\_array)

print("Sum of all elements:", result\_sum)

2.Write a python code to implement queue

Queue = []

def enqueue():

e = input("Enter the element you want to add in the queue which works in the FIFO order: ")

Queue.append(e)

print(e + " is added in the queue ")

def dequeue():

if not Queue:

print("Queue is empty")

else:

el = Queue.pop(0)

print("removed element from the queue is:", el)

def display():

print(Queue)

while True:

choice = int(input("Select the operation you want to perform on the queue: 1. ADDITION 2. REMOVAL 3. SHOW 4. QUIT: "))

if choice == 1:

enqueue()

elif choice == 2:

dequeue()

elif choice == 3:

display()

elif choice == 4:

print("Exiting the program. Goodbye!")

break

else:

print("Invalid input. Please try again.")

Or

Explain the internal working of a hash map

A Hash Map, also known as a Hash Table, is a data structure that maps keys to values using hash function and that allows for efficient insertion, deletion, and retrieval of key-value pairs.

**Hash function** The hash function creates a mapping between key and value, this is done through using hash functions. The **hash function**

receives the input key and returns the index of an element in an array called hash table.

**Collision:** The hashing process generates a small number for a big key, so there is a possibility that two keys could produce the same value. The situation where the newly inserted key maps to an already occupied, and it must be handled using some collision handling technology.

There are different techniques to handle collisions, and two common approaches are separate chaining and open addressing.

* + 1. **Separate Chaining:** make each cell of the hash table point to a linked list that have the same hash function value.
    2. **Quadratic Probing:** Quadratic probing operates by taking the original hash index and adding successive values of an arbitrary quadratic polynomial until an open slot is found.
    3. Linear Probing

In linear probing, the hash table is searched sequentially that starts from the

original location of the hash. If in case the location that we get is already occupied, then we check for the next location.

Algorithm:

1.Calculate the hash key. i.e. key = data % size

2.Check, if hashTable[key] is empty

•store the value directly by hashTable[key] = data

3.If the hash index already has some value then

1. check for next index using key = (key+1) % size

4.Check, if the next index is available hashTable[key] then store the value.

Otherwise try for next index.

5.Do the above process till we find the space.

**Load Factor:** The load factor of the hash table can be defined as the number of

items the hash table contains divided by the size of the hash

table.

1. What do you mean by Binary search tree? Construct a binary search tree using the given data 20 26 200 343 322 444 221 664 343 322

The binary search tree is a binary tree data structure in which each node has at most two children node.

Properties or Characteristics of binary tree

All nodes in the left subtree have values less than the node’s value.

All nodes in the right subtree have values greater than the node’s value.

In a binary search tree, the search operation is performed with O (log n) time complexity. The search operation

In a binary search tree, the insertion operation is performed with O (log n) time complexity.

In a binary search tree, the deletion operation is performed with O (log n) time complexity.

or

Define Heap. Convert a binary tree into a minimum heap

A heap is a tree-based data structure that satisfies the heap property: -

Min heap

The value of the child node is greater than or equal to the value of the parent node

Max heap

The value of the root node must be greatest number among all the nodes in the entire subtree Heap is used to store and manage collection of elements

Here is the algorithm to convert a binary tree into a minimum heap:

1. Create an array of size n, where n is the number of nodes in the given BST.
2. Perform the in-order traversal of the BST and copy the values of the nodes in the array.
3. Now Perform the preorder traversal of the BST. While traversing the BST in a preorder manner, copy the values one by one from the array to the nodes.

15 Marks Question:-

1. (a) Difference between stack and queue. (5)

|  |  |
| --- | --- |
| Stack | Queue |
| Stack follows LIFO (Last in First Out) operation. | Queue follows FIFO (First in First Out) operation. |
| Stack performs push or pop operation for inserting or deleting an element into or from the stack. | Queue Performs Enqueue or dequeue Operation for inserting or deleting an element into or from the queue. |
| Example: - Collection of books, the first book at the top is getting picked first. | Example: People are waiting in a line, and the first person who arrived first will be in the first to get his food served. |
| Stack can be implemented using arrays or linked list. | Queues can be implemented using linked list arrays and circular linked list. |
| Insertion and deletion take place on the one end, that is same end i.e., top of the stack. | Insertion and deletion take place on two different ends. |

* 1. What is divide and conquer algorithm? (2)

The divide and conquer algorithm is recursively breaks down a complex problem into subproblems.  Once the solutions to the sub-problems are obtained, they are combined to give a solution to the original problem.

* 1. What is deque and priority queue? (5)

A deque, short for "double-ended queue," is a data structure that allows elements to be added or removed from both ends where elements are added at one end (rear) and removed from the other end (front).

It is a combination of stack and queue.

A priority queue is type of queue and an abstract data type each element in the priority queue has an associated priority, and elements with higher priority values are retrieved before elements with lower priority values. This means that when elements are removed from the priority queue, they are not necessarily removed in the order they were added; instead, they are removed based on their priority.

* 1. Evaluate the time and space complexity of stack operation in linked list implementation. (3)

### **Time Complexity:**

1. **Push Operation:**
   * Time Complexity: O(1).
2. **Pop Operation:**

Time Complexity: O(1).

The time complexity of stack operations in linked list implementation is O(1) for all operations, including push, pop, and peek. This is because the stack is implemented using a linked list, which allows for constant-time insertions and deletions at the beginning of the list.

**Space Complexity:** space complexity for a stack implemented using a linked list is O(N), where N is the number of elements in the stack. The space complexity is determined by the total number of nodes, and it is proportional to the number of elements in the stack.

OR

Implement the Bubble Sort algorithm in Python. Explain how the algorithm works and discuss its time complexity. Provide a Python code snippet demonstrating the implementation. (15)

Bubble sort simplest sorting algorithm that works by comparing the adjacent elements and swaps them if they are in the wrong order.

Implementation of the bubble sorting algorithm:

1.Begin with the first element and compare the current element with the next element.

2. If they current element is greater than the next element swept them,

3. If the current element is less than the next element, move to the next element and repeat step 1

4. Continue this process for each adjacent elements in the list.

5. Repeat this process for remaining unsorted elements until the entire list is sorted.

The time complexity of bubble sort:

**Best Case:** The best-case scenario occurs when the input array is already sorted.

The best-case time complexity is O(n), where 'n' is the number of elements.

**Average Case:**

* The average-case time complexity of bubble sort is O(n^2).

**Worst Case:**

* The worst-case scenario occurs when the input array is in reverse order
* The worst-case time complexity is O(n^2), where 'n' is the number of elements.

def bubble\_sort(arr):

    n = len(arr)

    # Traverse through all array elements

    for i in range(n):

        # Last i elements are already sorted, so we don't need to check them

        for j in range(0, n-i-1):

            # Swap if the element found is greater than the next element

            if arr[j] > arr[j+1]:

                arr[j], arr[j+1] = arr[j+1], arr[j]

# Example usage

my\_list = [69, 68, 9, 77, 404, 6.9, 99, 420, 104]

print("Original array:", my\_list)

bubble\_sort(my\_list)

print("Sorted array:", my\_list)

1. (a) What is AVL tree? (2)

An AVL tree defined as a self-balancing Binary Search Tree (BST) where the difference

between heights of left and right subtrees for any node cannot be more than one.

The difference between the heights of the left subtree and the right subtree for any node

is known as the balance factor

* 1. Insert the following keys in order to build an AVL tree clearly mention the different rotations used and the balanced factor of each node.

A, Z, B, Y, C, X, D, W, E, V, F. (7)

* 1. The in-order and pre-order traversal of a binary tree is given below: In-order: D, G, B, H, E, A, G, I, C

Pre-order: A, B, D, G, E, H, C, F, I

Find out the post-order traversal of the binary tree. (6) OR

1. What is linked list? (2)

## Linked list is a linear data structure, which is consist of group of nodes in a sequence where each node contains a data element and reference. Of the next node

## type. Singular, double linked list. And circular linked list.

1. How many types of linked list are there? Explain in details. (5)

Single linked list: is a type of linked list which is a chain of nodes in which each node contains data part and references pointing to the next node. In the last node of the single Linked list, the next reference is also assigned to None. It only facilitates forward directional traversal

Double linked list

Double linked list is a type of linked list which is a chain of nodes in which each node contains data part as well as to pointer or references pointing both previous node and the next node at the same time, In the first node of the double linked list the previous reference is assigned to none and in the last node of the double Link list, the next reference is assigned to None as well. It facilitates bidirectional traversal.

Circular linked list

Circular link list is a type of linked list which is a collection of nodes where each node is corrected through link in such a way it forms a circle. In circle linked list the last node of the list points to the first node of the list Instead of containing null value as the next reference this creates a loop in the list allowing them for continuous traversal

There are two types of circular link list single, circular link list and double circular link list

in circular, single linked list. A node contains data part and reference part. Each node point to the next node in the sequence but the last node contains the address of the first node

The traversal start from the first Node and the last node is linked with the first node, which forms a traversing loop

  1066            2042             1099

[69|2046]--><--[68|1099]---><---[77|1066]

    <---------------------------------

In Circular Double link list is a type of circular linked list, where each node contains the data part, previous reference. And next reference each node point to the next node in the sequence. the first node contains the reference of the last node. At the last node contains the reference of the first node, which forms a circle and facilitates even forward and backward traversing.

    -------------------------------------------->

  1066                   2042                 1099

[1099|69|2046]--><--[1066|68|1099]---><---[2042|77|1066]

  <-------------------------------------------------

1. What are the advantages of using linked list? (3)
2. linked list advantages
3. Linked lists are dynamic in nature: LinkedList are not contained of a fixed size
4. Efficient memory utilisation: LinkedList’s are not contained a fixed size. It is dynamic in nature. It can fit to its size accordingly allowing it to use memory more efficiently
5. Efficient insertion and deletion in linked list insertion. deletion operations can be easily implemented
6. Efficient in dynamic operation: Linked list reduces the access time for insertion and deletion and for the other operations
7. Ease in implementation: implementing basic operations like insertion, deletion, peak or top, and is empty, etc. Is simpler in linked list
8. What is header linked list? (1)

A header linked list is a type of linked list that starts with a special node called the header node. it serves as a placeholder or a starting point for the linked list.

1. Write an algorithm to insert a node after the given node in a double linked list. (4)

1. Create a new node (new\_node) with the given data (new\_data).

2. Check if the list is not empty

3. If the list is empty, print an error message and exit.

4. Change the previous node next reference to the reference of the new node

5. Store the previous node reference to the previous reference of the new node

6. Change the next node previous reference to the reference of the new\_node

7. Store the next node reference. To the next reference of the new node.

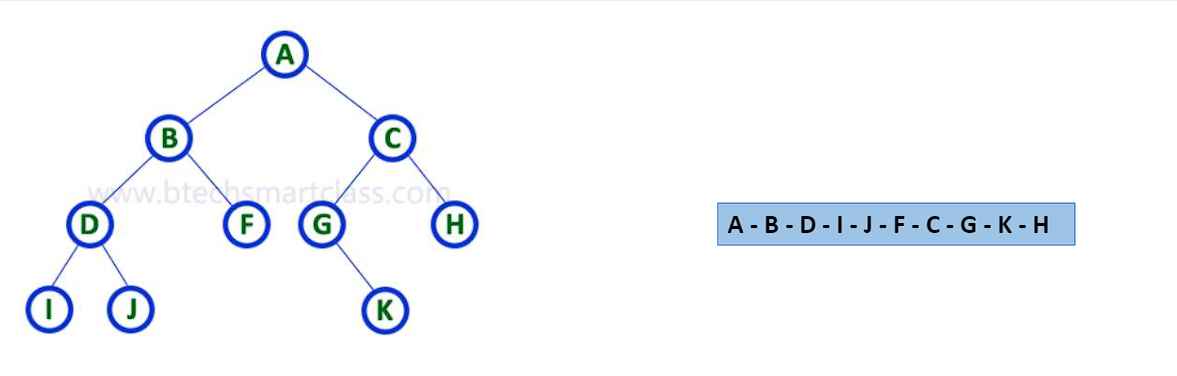
1. (a) What is Binary tree traversal? (2)

Traversing the binary search tree is a process of visiting each node in the tree exactly once in a systematic way. Each node in the binary search tree has at most two children node. And all the nodes in the left subtree have values less than the nodes value. And all the. Nodes in the right subtree has values greater than the nodes value.

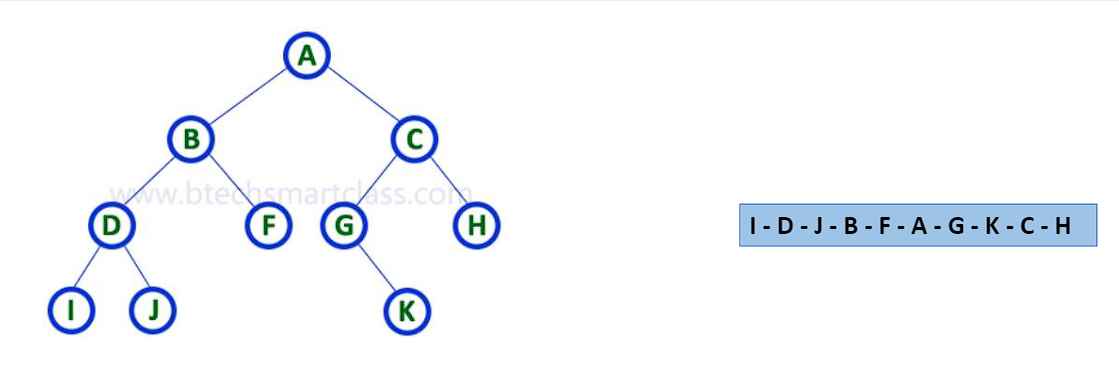
1. What are the three different ways to traverse a binary tree. Briefly explain. (6)

Binary Tree Traversals

1. Pre - Order Traversal:
   1. Visit a root note.
   2. Traverse the left subtree.
   3. Traverse the right subtree



1. In - Order Traversal:
   1. Traverse the left subtree
   2. Visit the route node
   3. Traverse the right subtree.



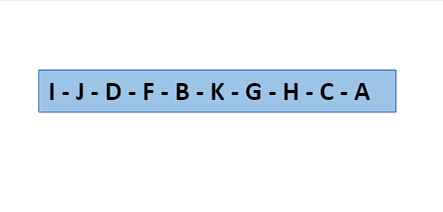
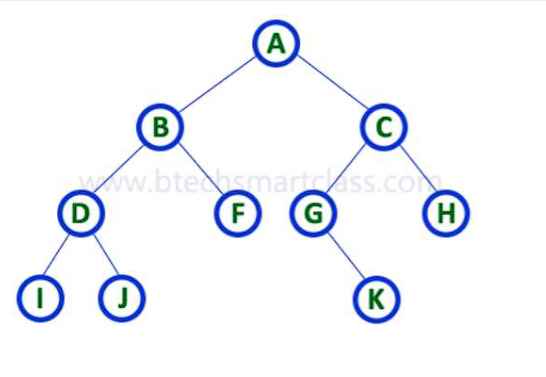
1. Post - Order Traversal:

To traverse in non-empty binary search tree in post order. The following operations are to be performed recursively at each node.

I)Traverse the left subtree.

II) Traverse the right subtree.

III) Visit the root node.



Traversing the binary search tree is a process of visiting each node in the tree exactly once in a systematic way. Each node in the binary search tree has at most two children node. And all the nodes in the left subtree have values less than the nodes value. And all the. Nodes in the right subtree has values greater than the nodes value.

1. How does merge sort work? When does best case, worst case and average case occurs in merge sort write down the time complexity for each case (7)

Merge sort is defined as a sorting algorithm works by dividing array into subarrays and sorting each of the sub array and then merge it together to form a complete sorted array.

Steps of merge sort algorithm:

1. If there is only one element in an array then the array is already sorted.
2. Otherwise, divide the array recursively into two halves until it cannot be divided.
3. Merge the subarrays together in a sorted form.

Time complexity analysis of Merge sort:

best case: when the elements are Already sorted in an ascending order. O (N log N)

worst case: When the elements are sorted in reverse order. O (N log N)

average case: When the elements are in jumbled order. O(N log N)

Or

a) write down a python code and the output to add these given polynomials using array: (5) 5+10x2+ 6x3 and 1+ 2x +3x2

1. def add\_polynomials(poly1, poly2):
2. result = []
3. # Pad the smaller polynomial with zeros to make them of equal length
4. if len(poly1) < len(poly2):
5. poly1 += [0] \* (len(poly2) - len(poly1))
6. else:
7. poly2 += [0] \* (len(poly1) - len(poly2))
8. # Add corresponding coefficients
9. for coeff1, coeff2 in zip(poly1, poly2):
10. result.append(coeff1 + coeff2)
11. return result
12. # Example polynomials
13. poly1 = [0, 5, 0, 10, 6]  # Represents 5^1 + 10x^2 + 6x^3
14. poly2 = [1, 2, 3]         # Represents 1 + 2x^1 + 3x^2
15. # Adding polynomials
16. result = add\_polynomials(poly1, poly2)
17. # Output
18. print("Result of (5^1 + 10x^2 + 6x^3) + (1 + 2x^1 + 3x^2):", result)

b) What is collision in hashing, write down the algorithm of linear probing. (5)

The hashing process**generates a small number for a big key, so there is a possibility that two keys could produce the same value.** The situation where the newly inserted key maps to an already occupied, and it must be handled using some collision handling technology.

**Linear Probing**

In linear probing, the hash table is searched sequentially that starts from the original location of the hash. If in case the location that we get is already occupied, then we check for the next location.

**Algorithm:**

1.Calculate the hash key. i.e. **key = data % size**

2.Check, if **hash Table[key]** is empty

•store the value directly by **hash Table[key] = data**

3.If the hash index already has some value, then

1. check for next index using **key = (key+1) % size**

4.Check, if the next index is available hashTable[key] then store the value.  Otherwise try for next index.

5.Do the above process till we find the space.

c)Write down the algorithm for searching elements in AVL tree (5)

Search Operation in BST

In a binary search tree, the search operation is performed with O(log n) time complexity. The search operation

is performed as follows...

•Step 1 - Read the search element from the user.

•Step 2 - Compare the search element with the value of root node in the tree.

•Step 3 - If both are matched, then display "Given node is found!!!" and terminate the function

•Step 4 - If both are not matched, then check whether search element is smaller or larger than that node value.

•Step 5 - If search element is smaller, then continue the search process in left subtree.

•Step 6- If search element is larger, then continue the search process in right subtree.

•Step 7 - Repeat the same until we find the exact element or until the search element is compared with the leaf node

•Step 8 - If we reach to the node having the value equal to the search value then display "Element is found" and

terminate the function.

•Step 9 - If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.