MCQs

1. If a is an array then a[-1] points to its
   1. First element b) Last element c) Element at the 1th index d) length

b) Last element

1. If a is an array then a.remove(1) will remove
   1. Element at the 1th index b) first occurrence of 1 c) all the 1 d) first element from the array

b) first occurrence of 1

1. a = arr.array('i', [1, 2, 3,9, 4, 7,5, 6,810])

print(a[3:8])

This Python code will print

* 1. 3 and 8 b) elements from 3rd to 7th index c) element on the 3rd index and 8th index d) all of the above

b) elements from 3rd to 7th index

1. using double hashing we stored three keys 27, 42, and 692 In a hash table if the first hash function is h1(key) = key%7, the second hash function is h2(key) = 1 + (key % 5) and the size of the hash table is 7 then the index number where a new key 72 will store
   1. 2 b) 3 c) 4 d)5

a)2

1. Time complexity of insertion sort is
   1. O(n2) b) O(logn) c) O(nlogn) d)O(n)

a)O(n2)

1. In which linked list the last node points to the head of the linked list
   1. Single linked list b) double linked list c) circular linked list d) none of the above

c) circular linked list

1. Recursion is a process of
   1. A function calling itself b) a function calling another function c) a function getting called multiple times d) a predefined function which does not get called

a) A function calling itself

1. Which data structure is used for implementing recursion?
   1. Stack b) Queue c) List d) Array
   2. Stack
2. What is the value of the postfix expression 6 5 2 4 + – \*?
   1. 74 b) -18 c) -6 d) 40

c) -6

1. Which data structure is used in cryptography
   1. Hash b) tree c) linked list d) array

a) Hash

## Suppose a circular queue of capacity (n – 1) elements is implemented with an array of n elements. Assume that the insertion and deletion operation are carried out using REAR and FRONT as array index variables, respectively. Initially, REAR = FRONT = 0.

The conditions to detect queue full and queue empty are

* 1. Full: (REAR+1) mod n == FRONT, empty: REAR == FRONT
  2. Full: (REAR+1) mod n == FRONT, empty: (FRONT+1) mod n == REAR
  3. Full: REAR == FRONT, empty: (REAR+1) mod n == FRONT
  4. Full: (FRONT+1) mod n == REAR, empty: REAR == FRONT

# 5 Marks Question

1. Write down a python code and the output to add these given polynomials using array: 5+10x2+ 6x3 and 1+ 2x +3x2
2. def add\_polynomials(poly1, poly2):
3. result = []
4. # Pad the smaller polynomial with zeros to make them of equal length
5. if len(poly1) < len(poly2):
6. poly1 += [0] \* (len(poly2) - len(poly1))
7. else:
8. poly2 += [0] \* (len(poly1) - len(poly2))
9. # Add corresponding coefficients
10. for coeff1, coeff2 in zip(poly1, poly2):
11. result.append(coeff1 + coeff2)
12. return result
13. # Example polynomials
14. poly1 = [0, 5, 0, 10, 6]  # Represents 5^1 + 10x^2 + 6x^3
15. poly2 = [1, 2, 3]         # Represents 1 + 2x^1 + 3x^2
16. # Adding polynomials
17. result = add\_polynomials(poly1, poly2)
18. # Output
19. print("Result of (5^1 + 10x^2 + 6x^3) + (1 + 2x^1 + 3x^2):", result)

or

What is collision in hashing, write down the algorithm of linear probing.

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.

## How many types of linked list are there? Explain in details.5

## Linked 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.

## Advantages

## 1.Linked list are dynamic in nature: Link lists are not constrained of fixed size, like array.

## 2. Efficient memory utilization: linked list are not constrained or fixed size. It is dynamic in nature. It can fit into its size, accordingly, allowing them to use the memory more efficiently.

## 3. Sufficient insertion and deletion: In Linked list instruction and deletion operation can be easily implemented.

## 4. Cache issues: Linked list may not utilize the cache efficiently as the nodes are scattered in different memory locations lead to slower the access time.

## Applications of linked list:

## Dynamic Memory Allocation:

## Implementation of data structure: serve as a fundamental structure for implementing graphs, queues and stack.

## Singular Linked list: Single linked list is type of linked list which is a chain of nodes in which each node contains the data part and reference of the next node. The reference of the last node is assigned a NULL value. It facilitates traversal in only forward direction.

## Double linked list: Double link list is a type of linked list which is a chain of nodes in which each note contains data part and 2 two references pointing to both the previous node and the next node at the same time, if it is a first node in the double link list, the previous link is assigned to none. And if it is the last node of the double link list, then the next reference is assigned to none as well.

## Advantages of double linked list:

## Bidirectional traversal: Double link list allows traversal in both the directions, forward and backward. Providing more flexibility in various operations.

## Efficient insertions and deletions: Adding or removing nodes at both ends of the Double linked list is more efficient.

## Versatility: Suitable for Those scenarios were forward and backward traversal and manipulation of elements are frequently required.

## Disadvantages or limit or limitations or demerits of Double linked list: -

## Increased memory usage: Double link list requires extra memory for the backward pointers leading to higher memory usage compared to single linked list.

## Complexity: The extra pointers make the implementation and maintenance of double linked list more complex than single linked list.

## Higher storage cost: In double link list each node requires more space to store two pointers, making it higher cost.

## Potential for inconsistency: The bidirectional nature of doubling list make it potentially inconsistent, requiring careful management to manage and maintain data.

## Circular linked list is a type of linked list Which is a collection of nodes where each node is connected through link in such a way it forms a circle. In. circular linked list, the last node of the list points to the first node instead of containing a null reference this creates a loop in the list, allowing them for continuous traversal.

## There are two types of circular linked list:

## Single circular linked list: In circular single link list. A node contains data part and reference word. Each node points to the next node in this sequence. But the last node contains the address of the first node The traversal starts from the first node, and the last node is linked with the 1st node, which forms a traversing loop.

## Double circular linked list: Double circular linked list is a type of the circular linked list where each node contains the data part, previous reference and next reference. Each node points to the next node in the sequence. And first node contains the reference of the last node and the last node contains the reference of the 1st node, which forms a circle and facilitates even forward and backward traversal.

## Advantages of circular linked list:

## Traversal convenience: Traversing circular linked list is more convenient as the last node contains the reference of the first node.

## Efficient operations: Since circular linked list Do not have a fixed rent operations like insertion and deletion can be more efficient.

## Time effective: It saves time when we have to go to the first node from the last node.

## Memory utilization: Circular linked list make better use of memory as the last node points to the first node, avoiding unused memory.

or

Write down the algorithm for searching elements in AVL tree

Search Operation in AVL Tree

In an AVL tree, the search operation is performed with O(log n) time complexity. The search operation in the

AVL tree is similar to the search operation in a Binary search tree. We use the following steps to search an

element in AVL tree...

•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.

1. 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 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 2 different ends |

Or

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.

h, g, i, e, d, b, f, c, a

# 15 Marks Question

1. a) What is shell sort? Write down shell sort algorithm to sort this given array [35, 33, 42, 10, 14, 19, 27, 44] (10)

Shell sort is highly efficient sorting algorithm based on insertion sort algorithm.

Space complexity of shell sort algorithm.

Best case: O(n)

Worst case: O(n)

shell sort algorithm.

1.Initialize h as an interval with N/2 where N is total number of elements in an Array and divided by two in every iteration until it becomes 1.

2. Divide the array into sub lists of equal intervals.

3. Apply insertions for algorithm to sort the sub lists.

4. Repeat 2nd and 3rd step until the array becomes sorted.

0 1 2 3 4 5 6 7

[**35**, 33, 42, 10, **14**, 19, 27, 44] 8/2 = 4

0 1 2 3 4 5 6 7

[**14**, 33, **42**, 10, **35**, 19, **27**, 44] 4/2 = 2

0 1 2 3 4 5 6 7

[**14, 33, 27, 10, 35, 19, 42, 44**] 2/2 = 0

Check 33 is less than 14 no don’t swap

[**14, 33, 27, 10, 35, 19, 42, 44**]

Check 27 is less than 33 yes swap

[**14, 27, 33, 10, 35, 19, 42, 44**]

Check 10 is less than 33 yes swap yes swap then again check 10 is less than 27 yes swap then again check 10 is less than 14 yes swap

[10,**14, 27, 33, 35, 19, 42, 44**]

Check 35 is less than 33 No don’t swap

[10,**14, 27, 33, 35, 19, 42, 44**]

Check 19 is less than 33 yes then again check 19 is less than 33 yes then again check 19 is less than 27 yes swap

[10,**14, 19,27, 33, 35, 42, 44**]

Check 42 is less than 35 No keep it remain

[10,**14, 19,27, 33, 35, 42, 44**]

Check if 44 is less than 35 No keep it same

[10,**14, 19,27, 33, 35, 42, 44**]

1. write down the insertion sort algorithm and its time complexity. (5)
2. Assume that the first element is sorted.
3. Take the next element and store it as a key.
4. Now compare the key with all the elements in the sorted array.
5. If the element in the sorted array is smaller than the key, then move to the next element. And if the element in the sorted array is greater than the key, then shift the greater elements to the right.
6. Insert the value
7. Repeat until the array is sorted.

Time complexity of insertion sort

best case O(N)

worst case order(N^2).

Average case=order(N^2)

Or

* 1. Write Python code to insert a node at the end of a double linked list (7)

1. class Node:
2. def \_\_init\_\_(self, data):
3. self.data = data
4. self.nref = None  # next reference
5. self.pref = None  # previous reference
6. class Double\_ll:
7. def \_\_init\_\_(self):
8. self.head = None
9. def print\_ll\_FORWARD(self):  # Traversal Operation - FORWARD
10. print()
11. if self.head is None:
12. print("Linked List is Empty!!")
13. else:
14. n = self.head
15. while n is not None:
16. print(n.data,"--->",end="")
17. n = n.nref  # Move to the next node
18. def print\_ll\_Backward(self):
19. print()
20. if self.head is None:
21. print("Linked List is Empty!")
22. else:
23. n = self.head
24. while n.nref is not None:  # Move forward to reach the last node
25. n = n.nref

            while n is not None:  # Backward

1. print(n.data,"---> ", end="")
2. n = n.pref

    def add\_end(self, data):

        new\_node = Node(data)  # new\_node 4269[|86|]

        if self.head is None:

            self.head = new\_node

        else:

            n = self.head  # Temporary variable for traversal

            while n.nref is not None:  # Move forward to reach the last node

                n = n.nref

            n.nref = new\_node  # Update the last node's next reference

            new\_node.pref = n  # Set the previous reference for the new node

list = Double\_ll()

list.add\_end(100)

list.print\_ll\_FORWARD()

list.print\_ll\_Backward()

* 1. Write Python code to insert a node in between two nodes of a single linked list

1. class Node:
2. def \_\_init\_\_(self, data):
3. self.data = data
4. self.ref = None
5. class LinkedList:
6. def \_\_init\_\_(self):
7. self.head = None
8. def print\_linkedlist(self):
9. if self.head is None:
10. print("Linked list is empty")
11. else:
12. while self.head is not None:
13. print(self.head.data,"-->",end="")
14. self.head = self.head.ref

    def add\_begin(self, data):

        new\_node = Node(data)

        new\_node.ref = self.head

        self.head = new\_node

1. def add\_after(self,data,x):
2. n = self.head
3. while n is not None:
4. if x==n.data:
5. break
6. else:
7. n=n.ref
8. if x is None:
9. print("node is not present in the linked list")
11. else:
12. new\_node=Node(data)
13. new\_node.ref=n.ref
14. n.ref=new\_node
15. LL1 = LinkedList()
16. LL1.add\_begin(10)
17. LL1.add\_begin(20)
18. LL1.add\_after(200,10)
19. LL1.print\_linkedlist()

(8)

1. a) Write Python program to convert infix expression 3+4\*(5-6/7) into prefix expression (10)

## Write an algorithm to convert an in-fix expression to a post-fix expression (5)

To convert an infix expression to a postfix expression, we can use the following algorithm:

1. Create an empty stack and an empty output string.
2. For each character in the infix expression, do the following:
   * If the character is an operand, add it to the output string.
   * If the character is an operator, pop operators from the stack and add them to the output string until an operator with lower precedence is at the top of the stack. Then push the current operator onto the stack.
   * If the character is a left parenthesis, push it onto the stack.
   * If the character is a right parenthesis, pop operators from the stack and add them to the output string until a left parenthesis is at the top of the stack. Then pop the left parenthesis from the stack.
3. After processing all characters in the infix expression, pop any remaining operators from the stack and add them to the output string.
4. The resulting output string is the postfix expression.

Or

a. Explain the working principle of Hash Map (5)

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: The** idea is to make each cell of the hash table point to a linked list of records 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: Member of Elements The hash table divided by the size of the hash table.

b) what is algorithm? Write down the characteristics of an algorithm (5)

An algorithm is a well-defined sequence of instructions that are designed to solve a specific problem where each step indicates an intermediate work.

**Input and output**: Algorithms take inputs, which are the initial data provided to the algorithm, and produce outputs, which are the result generated by the algorithm after processing all the inputs.

**Definiteness:** Each step of the algorithm must be precisely defined.

**Finiteness:** The algorithm must terminate after a finite number of steps. It should not go into an infinite loop.

**Correctness:** An algorithm should produce the correct output for all valid inputs.

**Modularity:** Algorithms can be designed with modularity. This promotes code readability and maintainability.

**Generality**: An algorithm must be designed to solve a specific problem.

1. Define Heap. Convert a binary tree into a minimum heap (5)

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

Min heap

The element of each node is greater than or equal to the element at its parent. The minimum value element is at the root.

Max heap

The value of a node must be greatest among the node values in its entire subtree. and It is used to store and manage a 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.

a) Write a python code to merge two 3d arrays (5)

1. import pprint
2. def three(a,b,c):
3. list1= [[[[0]for col1 in range(a)]for col2 in range (b)] for row in range (c)]
4. list2= [[[[2]for col1 in range(a)]for col2 in range (b)] for row in range (c)]
5. list=list1+list2
6. return (list)
7. col1=2
8. col2=3
9. row=2
10. pprint.pprint(three(col1,col2,row))
11. output:
12. [[[[0], [0]], [[0], [0]], [[0], [0]]],
13. [[[0], [0]], [[0], [0]], [[0], [0]]],
14. [[[2], [2]], [[2], [2]], [[2], [2]]],
15. [[[2], [2]], [[2], [2]], [[2], [2]]]]
    1. how does merge sort work? When does best case, worst case and average case occurs in merge sort (5)

Merge sort is defined as a sorting algorithm works by dividing array into two 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 a descending order, O(N Log N

average case when the elements are in jumbled order O (N Log N) not in properly ascending order and not in properly descending order.

0 1 2 3 4 5 6

[69][68][77][104][99][420][9]

| |

[69][68][77] [104][99][420][9]

| | | |

[69] [68][77] [104|99] [420][9]

| | | |

[69] [69][77] [99|104] [9][420]

🡪[68|69|77]🡨 🡪[9|99|104|420]🡨

🡪[9][68][79][77][99][104][420]🡨

Advantages of merge sort:

* 1. Stability: Merge sort is a stable sorting algorithm.
  2. **Parallelizable: Merge sort is a Parallelizable algorithm.**
  3. **Time-Complexity: Merge sort has time complexity of O(N log N) which is well suited for both small and large datasets.**

**Disadvantages of Merge sort: -**

**a) Space Complexity: Merge sort requires additional space to store the merged sub arrays during the sorting process.**

**b) Not in place: Merge sort is a Not-in-place sorting algorithm which requires additional space to store the sorted data values/elements.**

**c) Not optimal for small datasets: Merge sort has a higher time complexity so it is not optimal for smaller datasets.**

* 1. write python program to find out the sum of all elements in an array using recursion

1. def array\_sum\_recursive(arr):
2. # Base case: If the array is empty, return 0
3. if not arr:
4. return 0
5. # Recursive case: Sum the current element with the sum of the rest of the array
6. return arr[0] + array\_sum\_recursive(arr[1:])
7. # Example usage:
8. my\_array = [1.1, 3, 9, 49, 6.9]
9. result\_sum = array\_sum\_recursive(my\_array)
10. print("Array:", my\_array)
11. print("Sum of all elements:", result\_sum)
12. OR
    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 (5)

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 nodes value.

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

( 20 )

| |

(26)

|

(200)

|

(343)

| |

(322) (444)

| |

(221) (664)

|

(343)

|

(322)

Operations of binary search tree:

1)Search: Search operation is used to find whether a given node is present in the binary search tree or not. The searching process begins from the root node. first it will check whether binary search tree is empty or not, if the binary search tree is empty then the given values is not present in the binary search tree and if it is not empty then it will compare the value of the root node with the given node. If it is equal that means the given value is present in the binary search tree and if it is not equal then check the given value is less than the root node if yes then you need to search left subtree and if the given value is greater than the root node then you need to search in right subtree.

2) Insertion: Insertion operation is used to add a new node with the given value at the correct position in the binary search tree.

To insert the new node with the given value first we need to check whether binary search tree is empty. If binary search tree is empty then add a new node in the binary search tree it will become the root node of the binary search tree. If binary search tree is not empty then we need to compare the value of the new node and the value of the root node, If it is equal then that means we are adding a duplicate value, if it is not equal and if the new node is greater than the root node then we need to go to the right side and add the node in the binary search tree and if the new node is less than the root node then we need to go to the left subtree and add the new node in the binary search tree.

Algorithm:

* 1. Check BST is empty: If yes: insert the new node: if no: compare the new node with the given value with the root node’s value.
  2. Root node’s value < given: if yes: then go to the right subtree and insert the new node: if no: then go to the right subtree and insert the new node.
  3. END

3)Deletion: - Delete operation deletes a node with the given value from the binary search tree.

It will delete a given node if it is present in the binary search tree. If it is not present in the binary search tree it will print a message that the given node does not get found in the binary search tree.

To delete a node from a binary search tree first we need to check whether binary search tree is empty or not, if binary search tree is empty, we can’t delete any node from binary search tree. If binary search tree is not empty then we need to compare the given value with the node’s value in binary search tree. If the given value is equal to the node’s value, then delete the node from the binary search tree.

* 1. Traversal: Traversing the binary search tree is a process of visiting each node in the tree exactly once in a systematic way.

Pre order traversal in binary tree:

* + 1. Visit the root node
    2. Traverse the left subtree
    3. Traverse the right subtree

In order traversal in binary search tree:

1. Traverse the left subtree
2. Visit the root node
3. Traverse the right subtree

Post order traversal in binary search tree:

To traverse a non-empty binary search tree in post order, the following operations performed recursively at each node

1. Traverse the left subtree
2. Traverse the right subtree
3. Visit the root node

* 1. write down a python code to implement circular queue (10)