**FINAL EXAMINATION**

**Q1** :

def reverse(self): #--------------------- Time Complexity O(n)

prev = None

current = self.head

while(current is not None):

next = current.next

current.next = prev

prev = current

current = next

self.head = prev

def insert\_with\_index(self, index, data):

if index > self.size or index < 0:

print("check given", index, "index value and enter again")

return False

if index == 0:

self.head = Node(data, self.head)

else:

current = self.head

for i in range(index - 1):

current = current.next

current.next = Node(data, current.next)

self.size += 1

def add\_tail(self, data): #------------------------- Time Complexity O(n)

self.insert\_with\_index(self.size, data)

**Q2** :

count(root, val): #-------------------------- Time Complexity O(n)

if (!root) return 0

return (root->val == val ? 1 : 0) +

valCount(val, root->left) +

valCount(val, root->right)

**Q3** :

def printAncestors(root, target):

if root == None:

return False

if root.data == target:

return True

if (printAncestors(root.left, target) or

printAncestors(root.right, target)):

print(root.data,end=' ')

return True

return False

**Q4** :

> O(n)

> def binarySearch(arr, l, r, x):

if r >= l:

mid = l + (r - l) // 2

if arr[mid] == x:

return mid

elif arr[mid] > x:

return binarySearch(arr, l, mid-1, x)

else:

return binarySearch(arr, mid + 1, r, x)

else:

return -1

> def printPreorder(root):

if root:

print(root.val),

printPreorder(root.left)

printPreorder(root.right)

**Q5** :

| **Name** | **Best** | **Average** | **Worst** |
| --- | --- | --- | --- |
| Selection Sort | Ω(n^2) | θ(n^2) | O(n^2) |
| Insertion Sort | Ω(n) | θ(n^2) | O(n^2) |
| Quick Sort | Ω(n log(n)) | θ(n log(n)) | O(n^2) |
| Merge Sort | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) |

**Note** :- All functions time run times are calculated under same input length.

**Q6** :

| 0 | 10 |
| --- | --- |
| 1 |  |
| 2 | 12 |
| 3 | 13 |
| 4 | 34 |
| 5 | 25, 45 |
| 6 | 36 |
| 7 |  |
| 8 |  |
| 9 | 9 |
| 10 |  |
| 11 |  |

**Q7** :

10

---------------------------------------------------------

1

10

—-------------------------------------------------------

7

1 10

---------------------------------------------------------

4

1 7

10

----------------------------------------------------------

5

4 7

1 10

------------------------------------------------------------

13

5

4 10

1 7

—----------------------------------------------------------

find(4)

4

1 5

13

10

7

**Q8** :

a)

7

10 12

15 55 33 20

65

--------------------------------------------------------------------------

b)

12

15 20

65 55 33

--------------------------------------------------------------------------

c)

Heapsort can be thought of as an improved selection sort: like selection sort, heapsort divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element from it and inserting it into the sorted region.

**Q9** :

* Double-ended queue
* First-In, First-Out
* PollFirst
* 1
* enqueue()
* Queue
* Font = 1 rear = 5
* \*\*\*\*\*\*
* 0120301
* 80
* 55 40 30 25 11