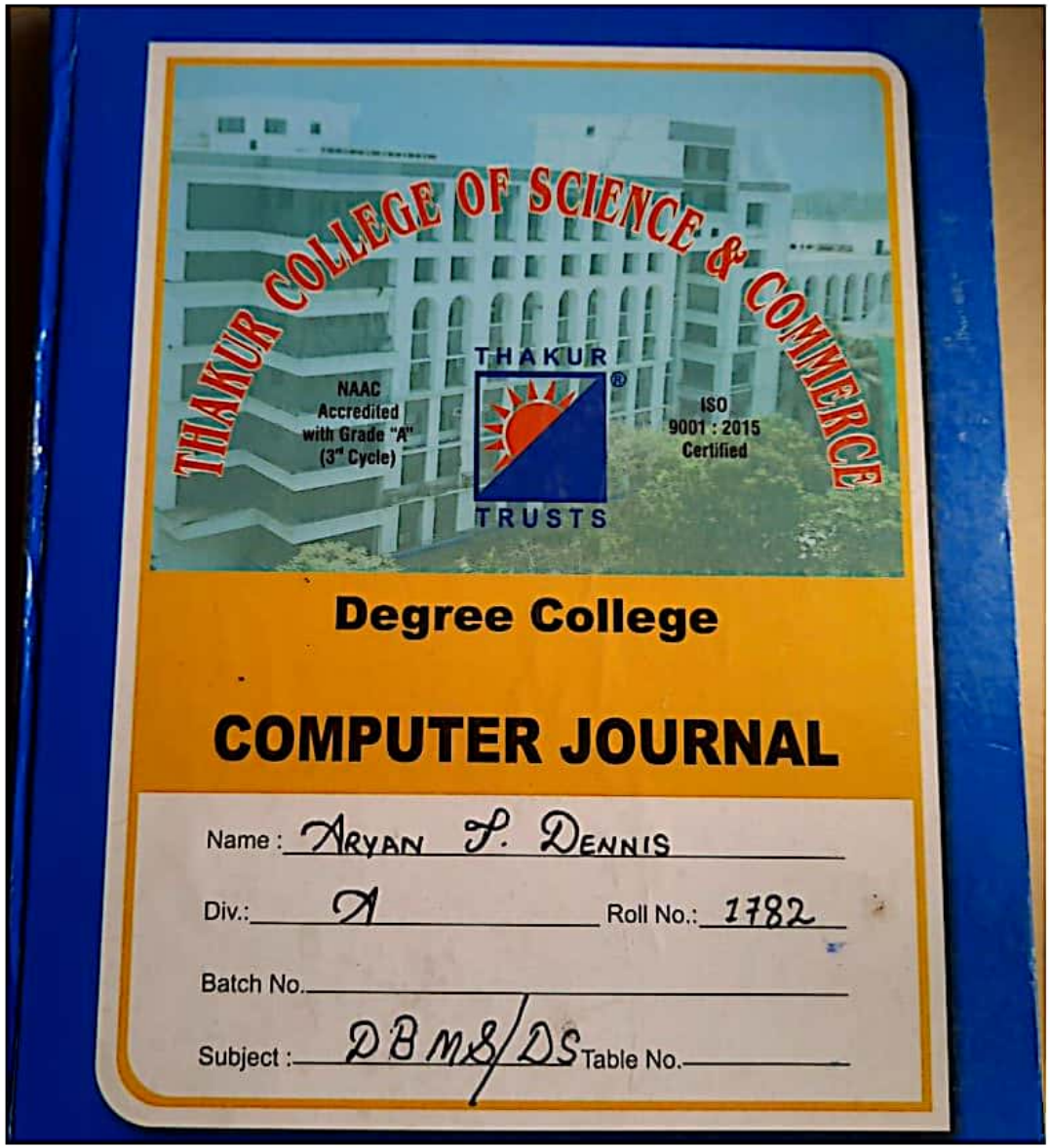
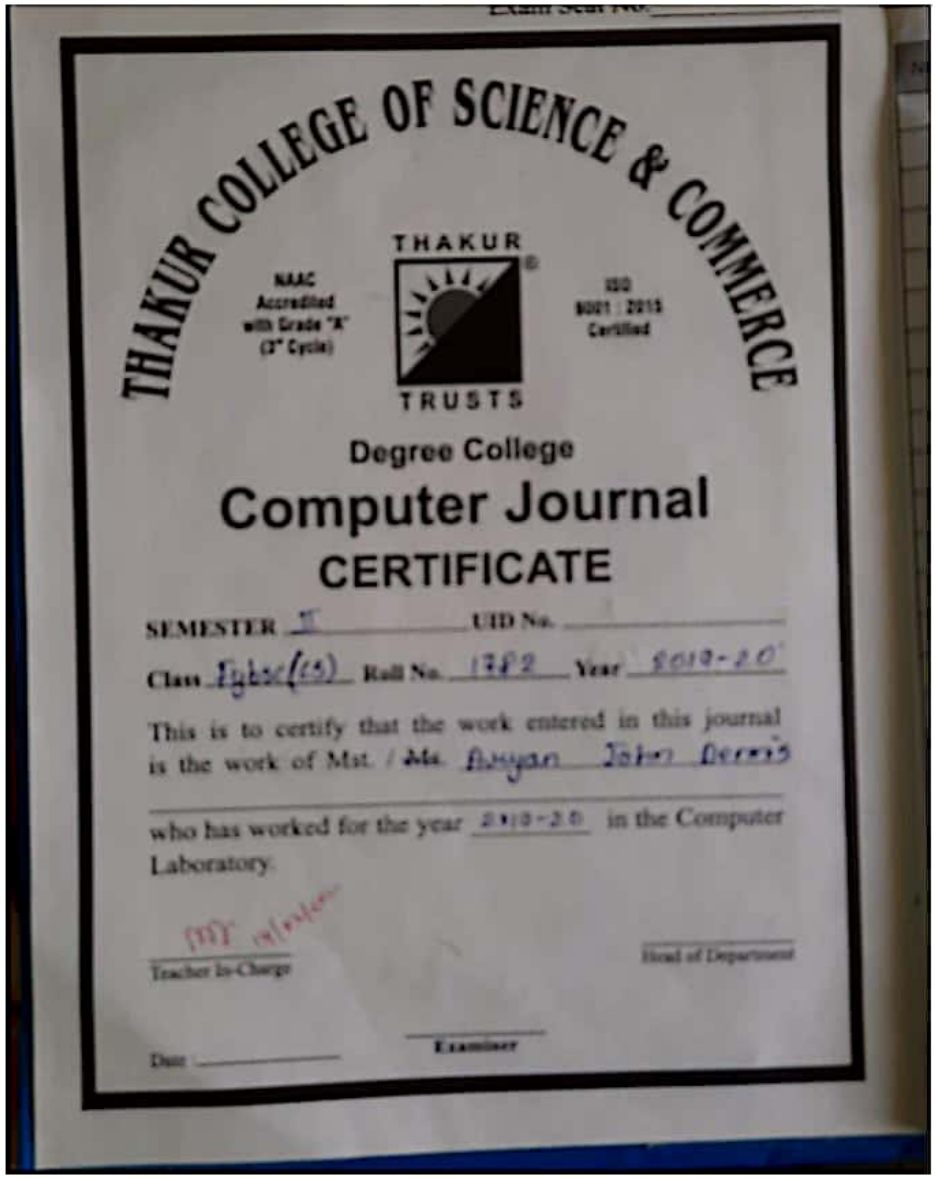




THAKUR COLLEGE OF SCIENCE & COMMERCE

NAAC Accredited with Grade "A" (3rd Cycle)

THAKUR TRUSTS

ISO 9001 : 2015 Certified

Degree College

Computer Journal

CERTIFICATE

SEMESTER I UID No. _____

Class Sybsc(IT) Roll No. 1782 Year 2019-20

This is to certify that the work entered in this journal is the work of Mst. / Ms. Aayan John Dennis

who has worked for the year 2019-20 in the Computer Laboratory.

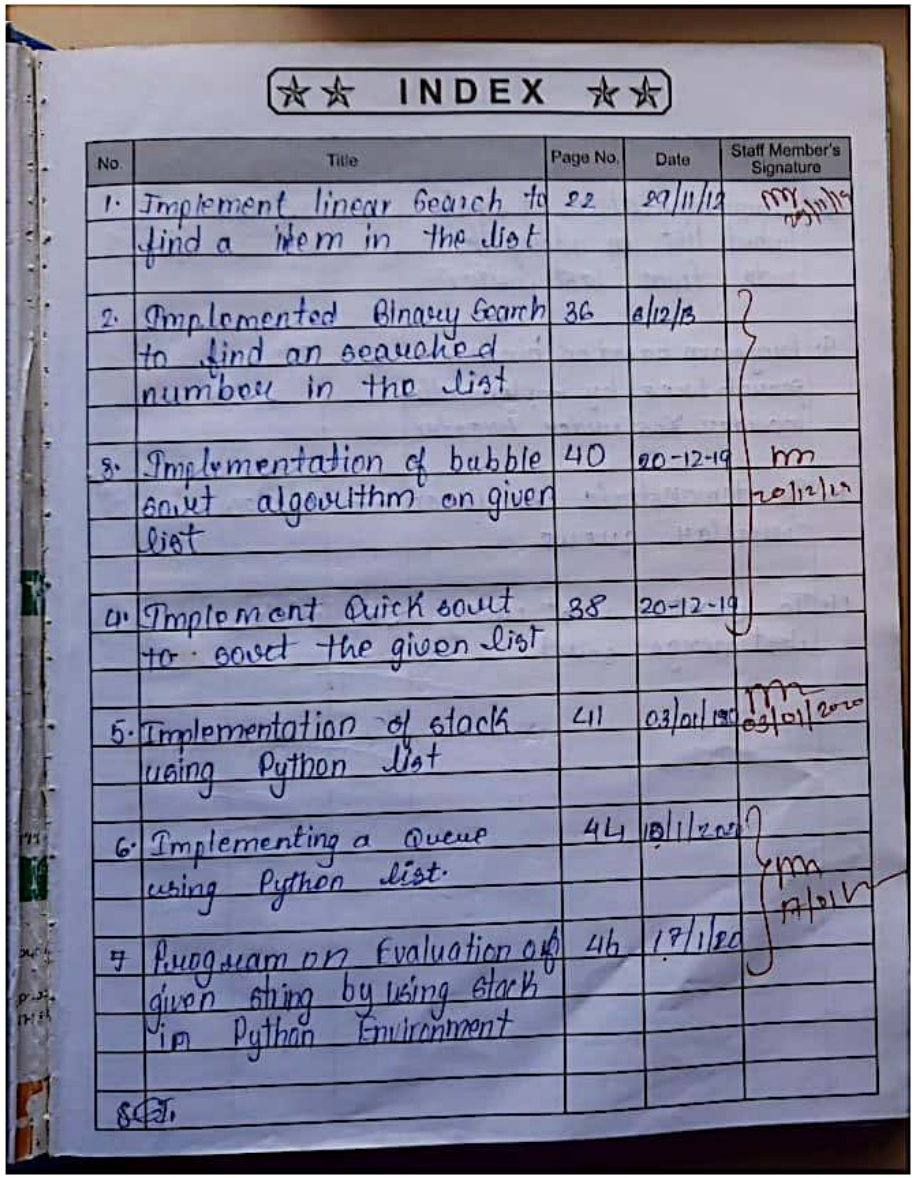
IIT Mumbai

Teacher In-Charge

Head of Department

Date _____

Examiner

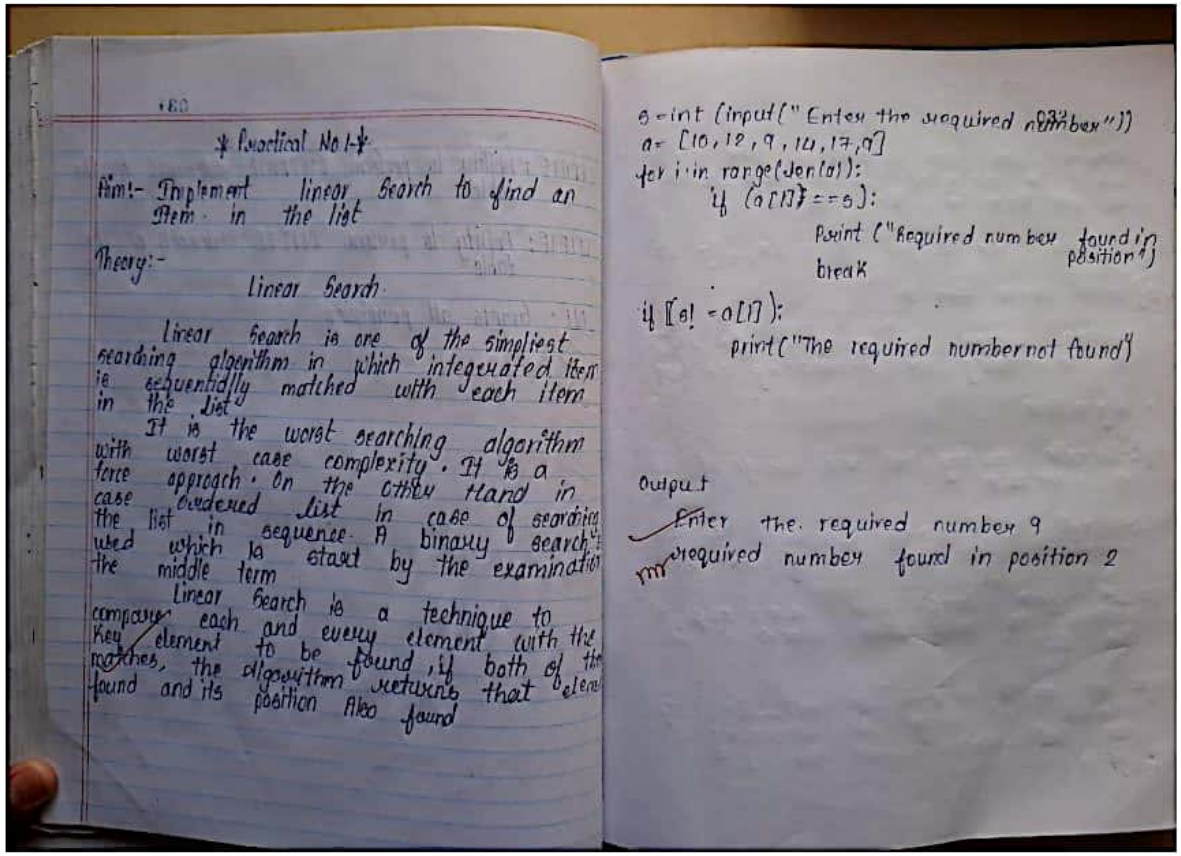


☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
1.	Implement linear Search to find a item in the list	22	29/11/19	m 29/11/19
2.	Implemented Binary Search to find an searched number in the list	36	6/12/19	
3.	Implementation of bubble sort algorithm on given list	40	20-12-19	m 20/12/19
4.	Implement Quick sort to sort the given list	38	20-12-19	
5.	Implementation of stack using Python list	41	03/01/2020	m 03/01/2020
6.	Implementing a Queue using Python list.	44	10/1/2020	m 17/01/20
7	Program on Evaluation of given string by using stack in Python Environment	46	17/1/20	
8.				

☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
8.	Implementation of single linked list by adding the node from last position	48	24/01/2020	mr 24/01/20
9.	Program based on binary search tree by implementing inorder, Postorder, Preorder	51	7/2/20	mr 14/02/2020
10.	To demonstrate the use of circular queue	57	14/2/20	
11.	To sort a list using ~~last~~ merge sort.	56	21/2/20	



* Practical No 1 *

Aim:- Implement linear Search to find an item in the list

Theory:-

Linear Search

Linear Search is one of the simpliest searching algorithm in which integerated item is sequentially matched with each item in the list

It is the worst searching algorithm with worst case complexity. It is a force approach. On the other hand in case ordered list in case of searching the list in sequence. A binary search is used which is start by the examination the middle term

Linear Search is a technique to compare each and every element with the key element to be found, if both of the matches, the algorithm returns that element found and its position also found

```
s=int (input(" Enter the required number"))
a= [10,12,9,14,17,9]
for i in range(len(a)):
    if (a[i]==s):
        Print ("Required number found in position")
        break
if [s! =a[i]):
    print("The required number not found")
```

output

Enter the required number 9

required number found in position 2

1] Unsorted:-
Algorithm:

Step1:- Create an empty list and assign it to a variable

Step2: Accept the total no. of elements to be interset into the list from the user say 'n'.

Step3: Use for loop for adding the elements into the list.

Step4: Print the new list

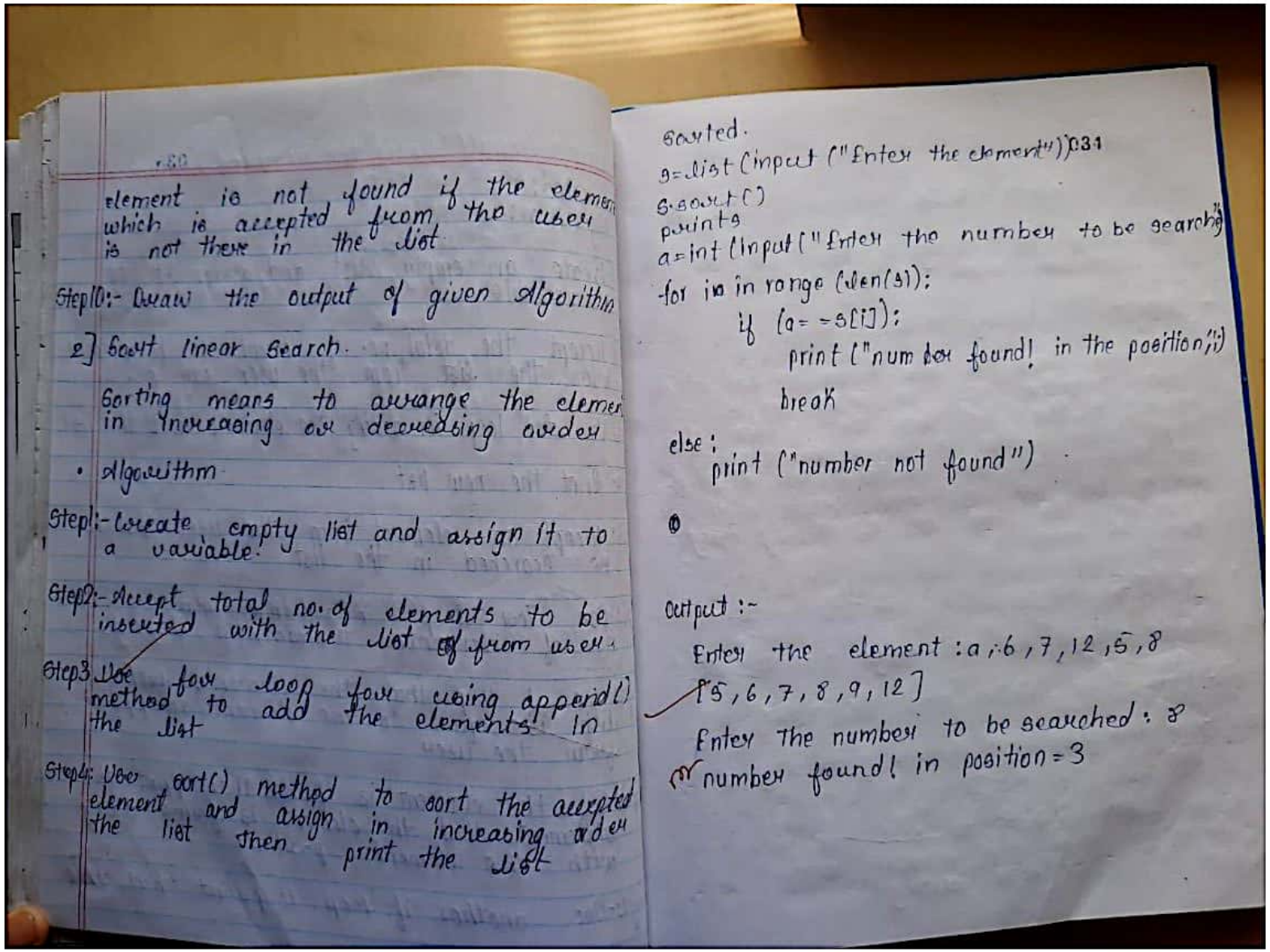
Step5: Accept an element from the user that to be searched in the list.

Step6: Use four loop in a range from '0' to the no. of elements to search the elements

Step7:- Use if loop that the elements from the list is equal to the element accepted from the user

Step8:- If the element is found then print the statement that the element is found along with the elements position

Step9:- Use another if loop to print that the



element is not found if the element which is accepted from the user is not there in the list

Step0:- Draw the output of given Algorithm

2] Sort linear Search.

Sorting means to arrange the elements in increasing or decreasing order

• Algorithm

Step1:- Create empty list and assign it to a variable.

Step2:- Accept total no. of elements to be inserted with the list of from user.

Step3 Use for loop for using append() method to add the elements in the list

Step4: Use sort() method to sort the accepted element and assign in increasing order the list then print the list

sorted.

```
a=list(input("Enter the element"))#31
a.sort()
print a
a=int(input("Enter the number to be search"))
for i in range(len(a)):
    if(a==a[i]):
        print("number found! in the position",i)
        break
else:
    print("number not found")
```

Output :-

Enter the element : a,6,7,12,5,8

[5,6,7,8,9,12]

Enter the number to be searched : 8

number found! in position = 3

Practical No. 2

- Aim:- Implemented Binary Search to find an searched number in the list

- Theory:- Binary Search

Binary Search is also known as half interval search, logarithm search or binary chop is a search algorithm that finds the position of a target value within a sorted ~~value~~ array. If we are looking at the number which is at the end of the list in linear search, which is time consuming. This can be avoided by using Binary Search

- Algorithm:-

Step 1:- Create a Empty list and assign a variable to it

Step 2:- Using input Method. accept the range of given list

Step 3:- Use for loop, add elements in list using append() Method.

Step 4:- Use sort() method to sort the accepted element and assign it in increasing ordered list print the list after sorting


```
a= list (input ("enter list: "))
a=sort()
c= len(a)
s= int(input ("enter search no:"))
if (s>a [c-1] or s<a[0]):
    print ("not in a list")
else:
    first, last =0, c-1
    for i in rang (0,c):
        m=int((first+last)/2)
        print ("found at", m)
        if s==a[m]:
            print ("number found")
            break
        else:
            if s<a[m]:
                last= m-1
            else:
                first =m+1
```

Output

```
Enter search no 1, 2, 3, 4, 5
[1, 2, 3, 4, 5]
Enter no:- 4
Number found at 3
```

Step5:- Use if loop to give the range in which element is found in given range, then display a message "Element not found"

Step6:- Then use else statement, if statement is not found in range then satisfy the below condition

Step7:- Accept an arguement & key of the element that element has to be searched.

Step8:- Initialize first to zero and last to element of the list as array is starting from 0 hence it is initialize 1 less than the total count

Step9:- Use for loop & assign the given range.

Step10:- If standard in list and still the element to be searched is not found then find the middle element (m)

Step11:- Else if the item to be searched is still less than the middle term then
Initialize last(h) = mid(m) - 1
Else
Initialize first(l) = mid(m) - 1

Step12:- Repeat till you found the element stick the input & output of above algorithm

Step5: Use If statement to give the range in which element to be sedrched before doing this accept on search number fiom user using Input statement.

Step6: Then use else statement if statement is not found in range then satisfy the given condition

Step7:- Use for loop in range from 0 to the table no. of elements to be searched ~~after~~ before doing this accept on search no from user using input statement.

mr
29/11/19

TE0

* PRACTICAL No. 4

Aim:- Implement Quick Sort to sort the given list

Theory:- The quick sort is given by Recursive Algorithm then based on Divide and Conquer technique

Algorithm:-

Step1:- Quick sort first select a value which is called pivot value, first element serve as our first pivot value, since we know that first will eventually end up as last in that last

Step2:- The partition process will happen next. It will find the split point and at the same time move other items to the co-orporate side of the list, either less than or greater than pivot value

Step3:- Partitioning begins by locating two position markers - lets call them leftmark & rightmark at the same time and end of remaining items in the list. The goal of partition process is to move items that on the wrong sides with respect to the pivot value. while also converging on the split value.

```
def quick(alist):
    help(alist,0,len(alist)-1)
def help(alist,first,last):
    if first<last:
        split=part(alist,first,last)
        help(alist,first,split-1)
        help(alist,split+1,last)
def part(alist,first,last):
    print(alist[first])
    l=first+1
    r=last
    done=false
    while not done:
        while l<=r and alist[l]<=pivot:
            l=l+1
        while alist[r]>=pivot and r>=l:
            r=r-1
        if r<l:
            done=True
        else:
            t=alist[l]
            alist[l]=alist[r]
            alist[r]=t
    t=alist[first]
    alist[first]=alist[r]
    alist[r]=t
    return=t
x=input("Enter range")
alist[]
```

038

```
for each b in range (0,n):
        b=input ("Enter element")
        alist.append (b)
        n = list (alist)
quick(alist)
print (alist)
```

Output:

```
Enter range of list 3
enter element 4
enter element 3
enter element 2
enter element 1
enter element 8
[1, 2, 3, 4, 8]
```


Step4:- We begin by incrementing leftmark until we locate a value that is greater than the pv. We then decrement rightmark until we find value that is less than the P.V. At this point we have discovered two items that are out of place with respect to eventual split point.

Step5:- At the point where rightmark becomes less then leftmark, we stop. The position of right mark is now the split point.

Step6:- The pivot value can be exchange with the contents of split point and pv is now in place.

Step7:- In addition all the items in left of split point are less than PV & all items in right are greater than PV. The list can be divided at split point and quick sort can be invoked recursively.

Step8:- The quicksort function invokes a recursive function.

Step9:- quick sort begin with same base as the merge sort.

Step10:- If length of the line is less than or equal to 1 it is already sorted.

Step11:- If it is greater that it can be partitioned.

Step12:- The partition function implements describe the process earlier.

Step13:-

Step5:- Use if loop to give the range in which element is found in given range then display a message "Element not found"

Step6: Then use else statement, if statement is not found in range then satisfy the below condition

Step7: Accept an arguement & key of the element that element has to be searched.

Step8:- Initialize first to zero and last to element of the list as array is starting from 0 hence it is initialize 1 less than the total count

Step9:- Use for loop & assign the given range.

Step10:- If standard in list and still the element to be searched is not found then find the middle element (m)

Step11:- Else if the item to be searched is still less than the middle term then
Initialize last(h) = mid(m) - 1
Else
Initialize first(e) = mid(m) - 1

Step12:- Repeat till you found the element stick the input & output of above algorithm

PRACTICAL No. 6

- Aim :- Implementation of stacks using Python List
- Theory : A stack is a linear data structure that can be represented in the linear world in the form of a physical stack or a public. The element in the stack are added are removed only from one position i.e., on the topmost position. Thus the stacks works on the LIFO [Last IN First Out] principle as the element that was inserted last will be removed first A stack can be implemented using array as well as linked list stack has three basic operation : push, pop, ~~Returns~~ pick topmost element ~~of stack~~. The operations of adding and removing the elements is known as Push & POP

- Algorithm :-

① Create a class stack with instance variable item

② define the 'init method with self arguement and initialize the initial value and then initialize the empty list.

(2) Define methods push and pop under the class stack

(4) Use If statement to give the condition that if length of given list is the greater t the range of list then print stack is ful

(5) Or else print statement as insert the elem -nt into stack and initialize the values.

(6) Push method used to insert the element but POP method used to delete the eleme from the stack

(7) If pop method value is less than 1 ther return the stack is empty or else dele the element from stack at topmost positi

(8) first condition check whether ther no. of elements are zero while the secon case whether top is assigned any value. If top is assigned any value then can be sure that stack is empty

(9) Assign the element value in push method add and print the given value is poped fr

(10) Attach the input and Output of above algori

Code:-

```
print("Aryan Dennis")
class stack:
global global tos
    def init (self):
        self.l=[0,0,0,0,0]
        self.tos=-1
    def push (self,data):
        n=len(self.l)
        if self.tos==n-1:
            print("stack is full")
        else:
            self.tos=self.tos+1
        self.l[self.tos]=data
    def pop(self):
        if self.tos<0:
            print("stack is empty")
        else:
            k=self.l[self.tos]
            print("Data=",k)
        self.l[self.tos]=0
        self.tos=self.tos-1
s=stack()
def pick (self):
    if self.tos<0:
        print("stack empty")
    else:
        s=self.l[self.tos]
        print("data=",s)
s=stack()
```

Output:-

Avyan Dennis
Data = 50

```
>>> s.d
[10,20,30,40,50]
>>> s.pop()
>>> s.pop()
>>> s.d
[10,20,30,0,0]
>>> s.pop()
>>> s.pop()
>>> s.pop()
>>> s.d
[0,0,0,0,0]
>>> s.push(10)
>>> s.push(20)
>>> s.push(30)
>>> s.push(40)
>>> s.push(50)
[10,20,30,40,50]
```

03/01/2020

PRACTICAL NO5
BUBBLE SORT

Aim:- Implementation of Bubble sort program on given list

Theory:- Bubblesort is based on idea of repeatedly comparing pairs of adjacent elements and then swapping their position if they exist in wrong order. This is the simpliest foum of sorting available. In this we sort the given elements in ascending and descending oreder by comparing two adjacent elements at a time.

*Algorithm

1. Bubble sort Algorithm start by comparing the first two element of an array and swapping necessary.
2. If we want to sort the elements of array in ascending order then first element is greater then second.
3. If the element is smaller than second then we do not swap the element.

```
a = list(input("Enter list"))
for i in range(0, len(a)-1):
    for j in range(0, len(a)-1):
        for a[j] > a[j+1]:
            a[j], a[j+1] = a[j+1], a[j].

print(a)
```

Output :-

Enter list: 69,75,30,90,2,11
[2, 11, 30, 65, 75, 90]

20/12/19

040

PRACTICAL NO. 6

Aim:- Implementing a Queue using python list

Theory:- Queue is a linear data structure which has 2 references front and rear implementation a queue using python list is the simpliest as the python list provides inbuilt function to perform the specified operations of the Queue. It is based on the principal that the new element is inserted after rear and element at queue can be described a data structure based on first out FIFO principle.

- Queue CI; Creates a new empty queue
- Enqueue (); Inserts an element at the rear of the queue and similar to insertion linked using tail
- Dequeue(); Returns the element which was at the front the front is removed to the successive element. A dequeue operation cannot remove the element if the queue is empty


```
class Queue:
    global if
    global r
    global a
    def a
    def __int__(self);
        self.f = 0
        self.r = 0
        self.a = [0,0,0,0,0]
    def enqueue(self,value):
        self.n = len(self.a)
        if(self.r == self.a):
            print("Queue is full")
        else:
            self.a[self.r] = value
            self.rt = 1
            print("insert value", value)
    def deQueue(self):
        if(self.f == len(self.a)):
            print("Queue")
        else:
            value = self.a[self.f]
            self.a[self.f] = 0
            print("Queue element deleted", value)
            self.ft = 1
b = Queue()
```

Output:

```
>>> b.enqueue(4)
('Queue element inserted', 4)
>>> b.endure(5)
  ("Queue element inserted", 5)
>>> b.enqueue(6)
  ("Queue element inserted", 6)
>>> b.enqueue(7)
  ("Queue element inserted", 7)
>>> b.enqueue(8)
  ("Queue element inserted", 8)
>>> b.enqueue(9)
  ("Queue element is full", 9)
>>> print(b.a)
  [4, 5, 6, 7, 8]
>>> b.dequeue(a)
  ("Queue element deleted", 4)
>>> b.dequeue()
  ("Queue element deleted", 5)
>>> b.dequeue()
  ("Queue element deleted", 6)
```


Algorithm

① Define a class queue and assign a class global global variables then define init() method with if arguement init assign or iniatize the initial value with the help of self arguement

② Define a empty set and define enqueue() method with 2 arguement assign the length of empty list

③ Define a empty use if statement that length is equal to length then queue is full or else to the element in empty list or display that Queue element added successfully and increement by 1.

④ Define deQueue() with arguement under these use if else statement that front is equal to length of is to then display Queue is Empty or else give that front is at zero and using to delete the element front front side and increment it by 1

⑤ Now call the queue() function give the element that has to be added in the empty list by using enqueue()

Practical No. 7
Resolution of Post fix

- Aim: Program on Evaluation of given string by using stack in Python Environment i.e. Postfix

- Theory :- The postfix expression is free of any paranthese. Further we took care of the priority of the operation in the program. A given postfix expression can easily be evaluated using stacks. Reading the expression is always from left to right in postfix

- Algorithm

Step1: define evaluate as function. Then create a empty stack in Python

Step2:- Convert the string to a list by using the string method split

Step3:- Calculate the length of string and print it

Step4:- Use for loop to assign the range of string then give condition using if statement

Step5:- Scan the token list from left to right. If token is an operand, convert it from

```
def evaluate(s):
    k = s.split()
    n = len(k)
    stack = []
    for i in range(n):
        if k[i].isdigit():
            stack.append(int(k[i]))
        elif k[i] == '+':
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) + int(a))
        elif k[i] == '-':
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) - int(a))
        elif k[i] == '*':
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) * int(a))
        elif k[i] == '/':
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) / int(a))
    return stack.pop()
s = "4 6 9 * +"
r = evaluate(s)
print("the evaluate value is:", r)
```

print("Aryan Dennis")

Output:-

The evaluate value is: 58
Aryan Dennis

a string to an integer and push the value onto the p

Step6: If the token is an operator *, /, +, -, it will need two operand. Pop the 'p' twice. The first pop is second operand and the second pop is the first operand

Step7: Perform the Arithmetic operation, Push the result back on the 'n'

Step8: When the input expression has been completely processed, the result is on the stack. Pop the 'p' and return the value

Step9: Print the result of string after the evaluation of postfix

Step10: Attach output and input of above Algorithm

17/01/24

PRACTICAL No. 8
Linked List.

- Aim: Implementation of single linked list by adding the node from last position.

- Theory:- A linked list is a linear data structure which stores the elements in a node in a linear fashion but not necessarily contiguous. The individual element of linked list called a node. Node comprises of 2 parts.
① Data ② Next. Data stores all the information w.r.t the element for example roll no, name, address, etc whereas next refers to the next node. In case of longer list, if we add/remove any element of the list then to adjust itself everytime we add it is very tedious task so linked list is used to solving this type of problem.

- Algorithm

Step1:- Transversing of linked list means using all the nodes in the linked list in order to perform some operation on them.

Step2:- The entire linked list means can be accessed via the first node of the linked list. The first node of the linked list in turn is referred by the

```
class node:
    global data
    global next
    def __init__(self,item):
        self.data=item
        self.next=None
class linked list:
    global e
    def __init__(self):
        self.s = None
    def addL(self,item):
        newnode =node(item)
        if self.s==None:
            self.s = newnode
        else:
            head=self.s
            while head.next!=None
            head=head.next
            head.next=newnode
    def addB(self,item):
        newnode = node(item):-
        ifself.s == None:
            self.s = newnode
        elef:-
            newned.next=self.s
            self.s= newnode
    def display(self):
        head =self.s.
        while head.next!) = None:
            print(head!.data)
```

Output:-

```
>>>start.addL(60)
>>>start.addL(60)
>>>start.addl(70)
>>>start.addL(80)
>>>start.addB(40)
>>>start.addB(30)
>>>start.addB(20)
>>>start.display()
>>> 20
    30
    40
    50
    60
    70
    80
>>>print("Aryan Dennis")
Aryan Dennis
```


Head pointer of linked list.

Step3:- Thus the entire linked list can be trapsverse the using the node which is referred by the head pointer of the linked list.

Step4:- Now that we know that we can transverse the entire linked list using the head pointer, we should only use it to refer the first node only

Step5:- we should not use the head pointer to transver -se entire linked list because the head pointer is our only reference to the 1st node in the linked list

Step6:- we may loose the reference to 1st node in our linked list and hence most of our linked list so in order to avoid making some unwanted changes to the 1st node we will use temporary node. to transverse the entire linked list

Step7:- we will use temporary node as a copy of the node we are currently transversing. Since we are making temporary node a copy of current node the datatype of temporary node should also be node

Step8: Now that current is referring to the first node if we want to access 2nd node of list we can refer it as a the next node of 1st node

Step9:- But 1st node is referenced by current we can traverse the 2nd node as h=h.next

Step10:- Similarly we can traverse next etc in the linked list using same method by while loop

Step11:- Our concern is to find terminating condition by the while loop.

Step12:- The last node in the linked list is referenced by the tail of linked list. Since the last node of linked list does not have any next node

Step13:- So we can write that to the last node of linked list self.next = None

Step14: We have to see now how to start traversing the linked list2 how to insert the last node of linked list

```
class node:
    def __init__(self,value):
        self.left=None
        self.val=value
        self.right=None
class BST:
    def __init__(self):
        self.root=None
    def add(self,value):
        p=node(value)
        if self.root==None:
            self.root=p
            print("Root is added successfully",p.value)
        else:
            h=self.root
            while root:
                if p.val<h.val:
                    if h.left==None:
                        h.left=p
                        print(p.value,"Node is added to the left side successfully",h.val)
                        break
                    else:
                        h=h.left
                else:
                    if h.right==None:
                        h.right=p
                        print(p.val,"Node is added to the right side successfully")
```

051

Practical No. 9

Aim:- Program based on Binary Search Tree by implementing, Inorder, Preorder, Postorder.

Theory:- Binary Tree is a tree which supports maximum number of two children for any node within tree. Thus any particular node can have 0 or 1 or 2 children there is another identity of binary tree, identified as left child and other as right child.

Inorder: i) Traverse the last subtree. The left subtree in turn might have left and right subtree
ii) Visit the root node
iii) Traverse the right subtree and repeat it

Preorder:- i) Visit the root node
ii) Traverse the left subtree. The left subtree might have left and right subtree
iii) Traverse the right trees and repeat it.

Postorder: i) Traverse the left subtree. The left subtree inturn might have left and right subtree
ii) Traverse the left subtree.
iii) Visit the node.

• Algorithm

i) Define class node and define init() method with 2 arguement. Initialize the value in this method

Step2: Again define a class BST that is binary search tree with init() method with self arguement and assign the root is None.

Step3: Define add() method for adding the node Define a variable p that p=node(value)

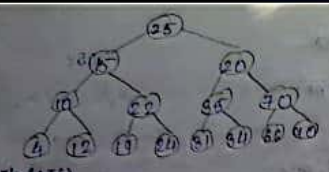
Step4: Use if statement for checking the condition that root is none then use else statement for if node is less than the main node then put or arrange that in leftside

Step5: Use while loop for checking the node less than or greater than the main node and breadth the loop if it is not satisfying

Step6: Use if statement within that else statement for checking that node is greater than main root then put it on rightside

Step7: After this leftsubtree and rightsubtree repeat this method to arrange the node accordingly to binary search tree

```
break
else:
    h= h.right
def Inorder(root):
    If root == None:
        return
    else:
        Inorder(root.left)
        print(root.val)
        Inorder(root.right)
def Preorder(root):
    if root == None:
        return
    else:
        print(root.val)
        Preorder(root.left)
        Preorder(root.right)
def Postorder(root):
    if root==None:
        return
    else:
        Postorder(root.left)
        Postorder(root.right)
        print(root.val)
```




```
>>>t=BST()
>>>t.add(15)
Root added successfully @25
>>>t.add(15)
15 node is added to rightside successfully at
>>>t.add(20)
20 node is added to right side successfully at @25
>>> t.add(10)
10 node is added to rightside at @25
>>>t.add(22)
22 node is added to leftside successfully at
>>>t.add(24)
24 node is added to rightside successfully at
>>>t.add(18)
18 node is added to leftside successfully at
>>>t.add(12)
12 node is added to leftside successfully at 25
>>>t.add(21)
21 Node is added to rightside successfully at 2
>>>t.add(25)
25 Node is added to leftside successfully at
```


Step6:- Define Inorder, Preorder() and Postorder() with root. arguement and use if statement that root is none and return that in all.

Step7:- In order else statement used for giving that condition first left root and then right node

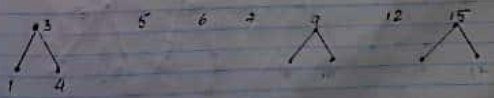
Step10:- for preorder. We have to give condition in else that first root, left and then go for root node

Step12:- Display the output and input of above algorithm

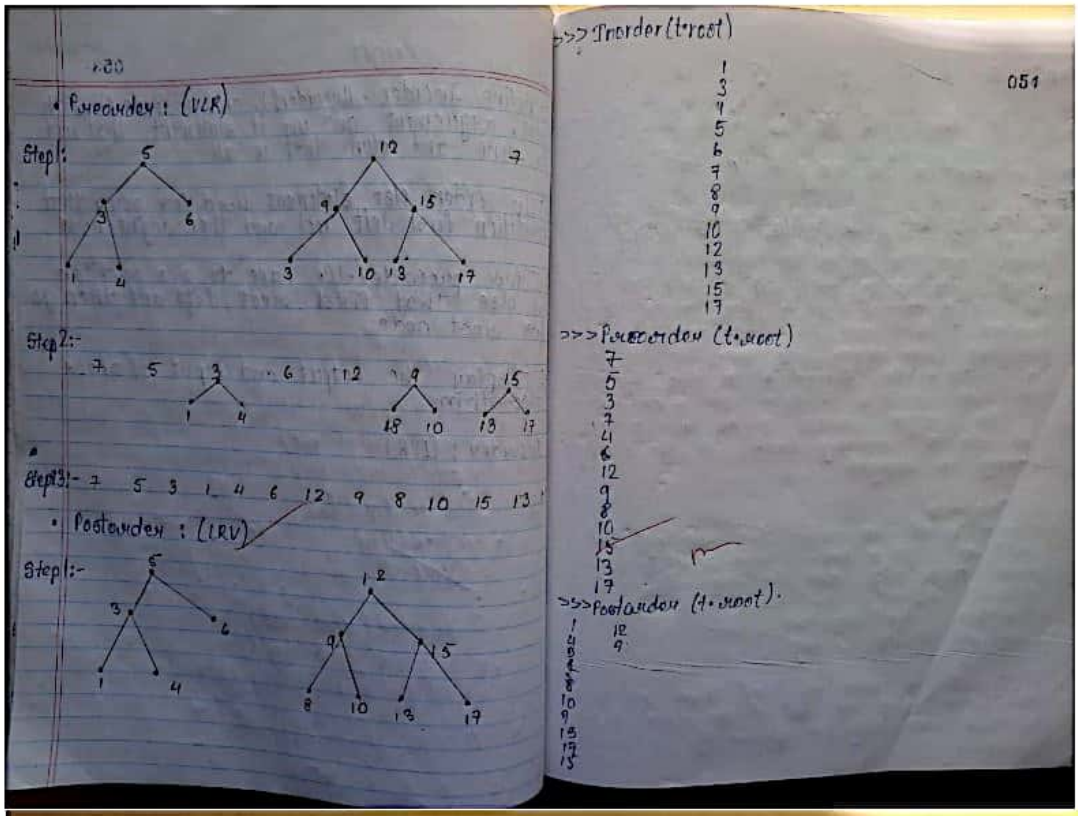
• Inorder : (LVR)

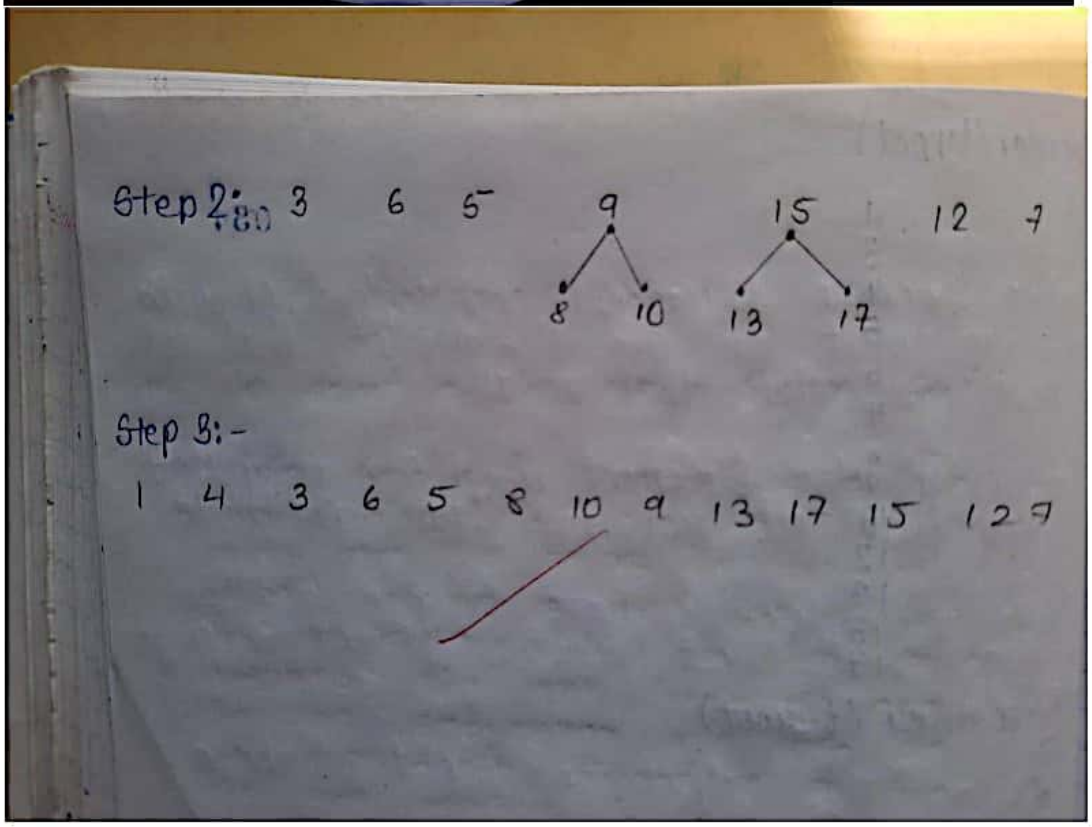
Step1:-

Step2:-



Step3:- 1 3 4 5 6 7 8 9 10 12 13 15 17





PRACTICAL No- 11

Aim:- To sort a list using Merge Sort

Theory:- Like Quicksort, Mergesort is, a divide and conquer algorithm. It divides input array in halves and then merges the two halves. The merge() function is used for merging two halves. The merge (arr, l, m, r) is key process that assumes that arr[l.....m] and merges the two sub-array into one. The array is recursively divided in to two halves the size becomes 1. Once the size becomes 1, the merge process comes into action and starts merging back till the complete array is merged.

Applications
1. Merge sort is useful for sorting linked list in O(nlogn) time. Merge sort accesses data sequentially and the need of random access is low.
2. Inversion count problem
3. Used in External Sorting

Mergesort is more efficient than quicksort for some types of list if the data to be sorted can only be efficiently accessed sequentially.

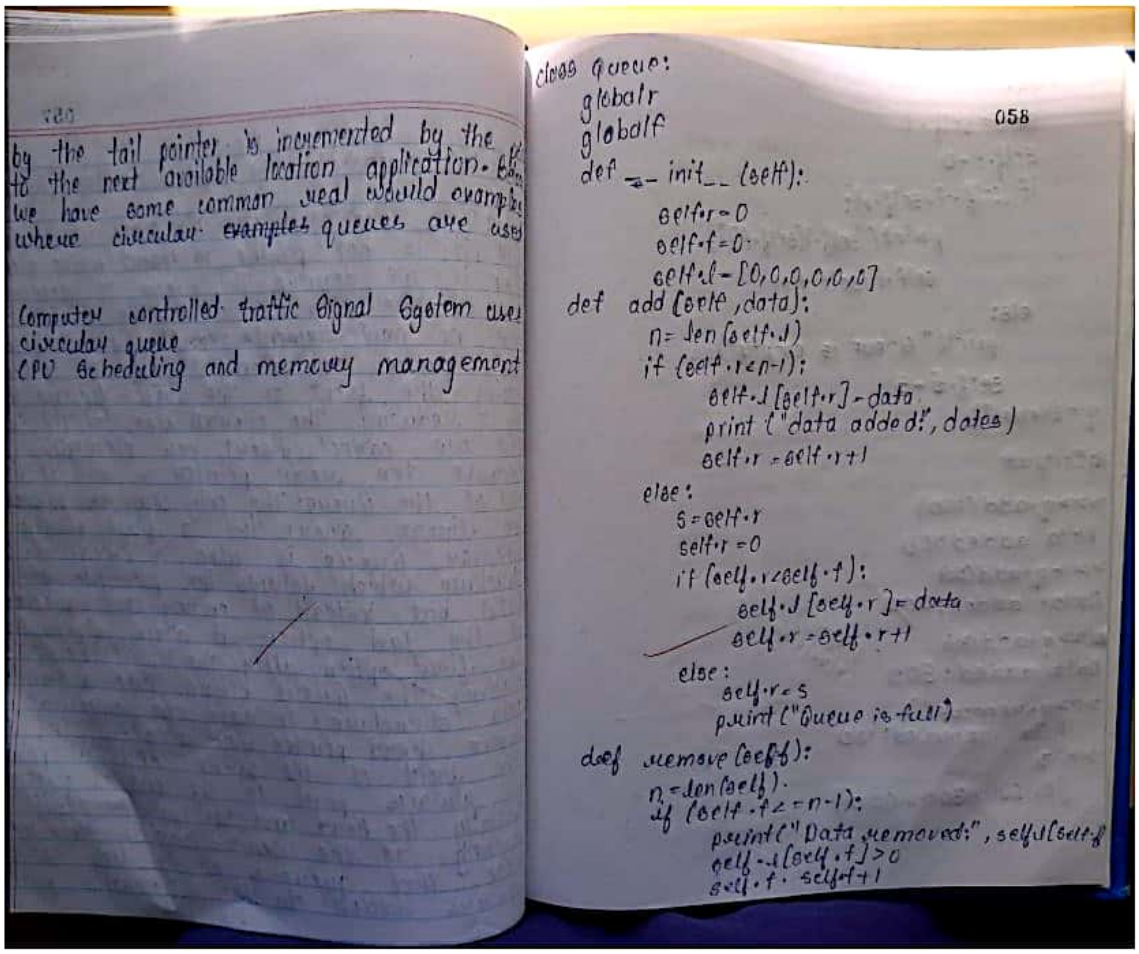
```
def mergeSort(arr):
    if len(arr)>1:
        mid=len(arr)//2
        leftHalf=arr[:mid]
        rightHalf=arr[mid:]
        mergeSort(leftHalf)
        mergeSort(rightHalf)
        i=j=k=0
        while(i<len(leftHalf) and j<(rightHalf)):
            if leftHalf[i]<rightHalf[j]:
                arr[k]=leftHalf[i]
                i=i+1
            else:
                arr[k]=rightHalf[j]
                j=j+1
            k=k+1
        while(j<len(leftHalf)):
            arr[k]=rightHalf[j]
            j=j+1
            k=k+1
arr=[27,89,70,55,62,99,45,12,10]
print("RANDOM LIST",arr)
mergeSort(arr)
print("\n MergeSorted List:",arr)
```

Output :-
RANDOM LIST : [27, 89, 70, 55, 62, 99, 45, 12, 10]
MERGESORTED LIST : [10, 12, 27, 45, 62, 70, 89, 99]

PRACTICAL No.10.

Aim: To demonstrate the use of circular queue.

Theory: In a linear queue once the queue is completely full, it is not possible to insert more elements even if we dequeue the queue to remove some of the elements until the queue is reset, no new elements can be inserted. When we dequeue any element we are actually moving the front of the queue forward, thereby reducing the overall size of the queue. And we cannot insert new elements because the rear pointer is still at the end of the queue. The only way is to reset the linear queue for a fresh start. Circular Queue is also a linear data structure which follows the principle of FIFO but instead of ending the queue at the last option it again starts from the first option after the last, hence making the queue behave like a circular data structure. In case of a circular queue head pointer will always point to the front of the queue and tail pointer will always point to the end of queue. Initially, the head and tail pointer will be pointing to the same location this would mean that queue is empty. New data is always added to the location pointed.

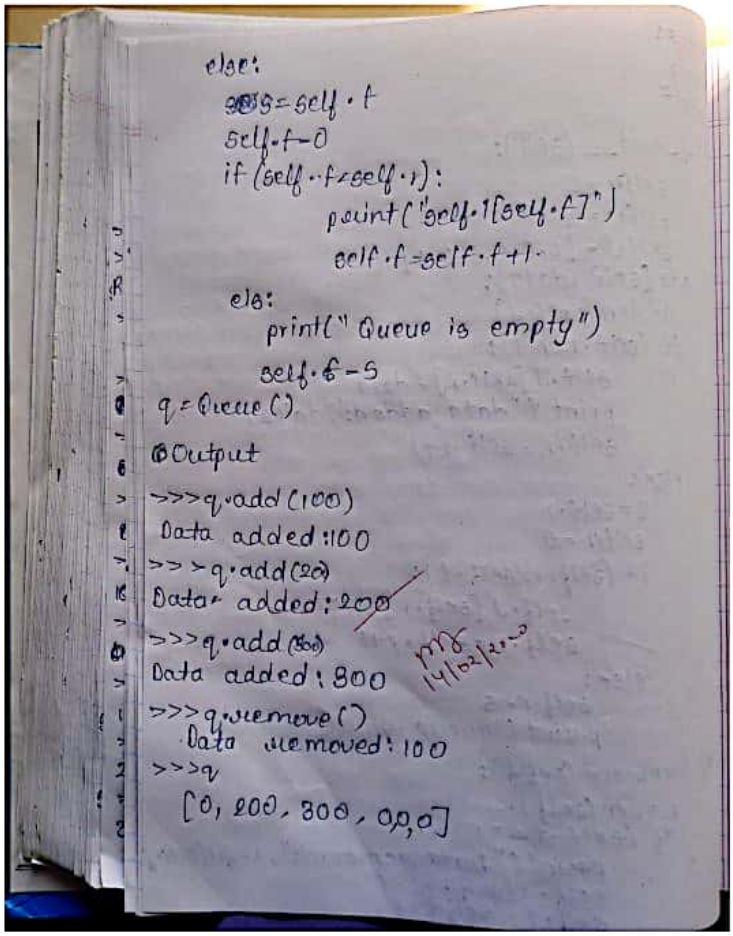


by the tail pointer is incremented by the
to the next available location application.
we have some common real would example
where circular examples queues are use

Computer controlled traffic Signal System uses
circular queue
CPU Scheduling and memory management

058

```
class Queue:
    global r
    global f
    def __init__(self):
        self.r = 0
        self.f = 0
        self.l = [0,0,0,0,0,0]
    def add(self,data):
        n = len(self.l)
        if (self.r <= n-1):
            self.l[self.r] = data
            print("data added:", data)
            self.r = self.r+1
        else:
            s = self.r
            self.r = 0
            if (self.r < self.f):
                self.l[self.r] = data
                self.r = self.r+1
            else:
                self.r = s
                print("Queue is full")
    def remove(self):
        n = len(self)
        if (self.f <= n-1):
            print("Data removed:", self.l[self.f]
            self.l[self.f] = 0
            self.f = self.f+1
```



```
        else:
            s=self.f
            self.f=0
            if (self.f<self.r):
                print("self.l[self.f]")
                self.f=self.f+1
            else:
                print("Queue is empty")
                self.f=s
q=Queue()
Output
>>>q.add(100)
Data added:100
>>>q.add(200)
Data added:200
>>>q.add(300)
Data added:300
>>>q.remove()
  Data removed:100
>>>q
  [0,200,300,0,0]
```