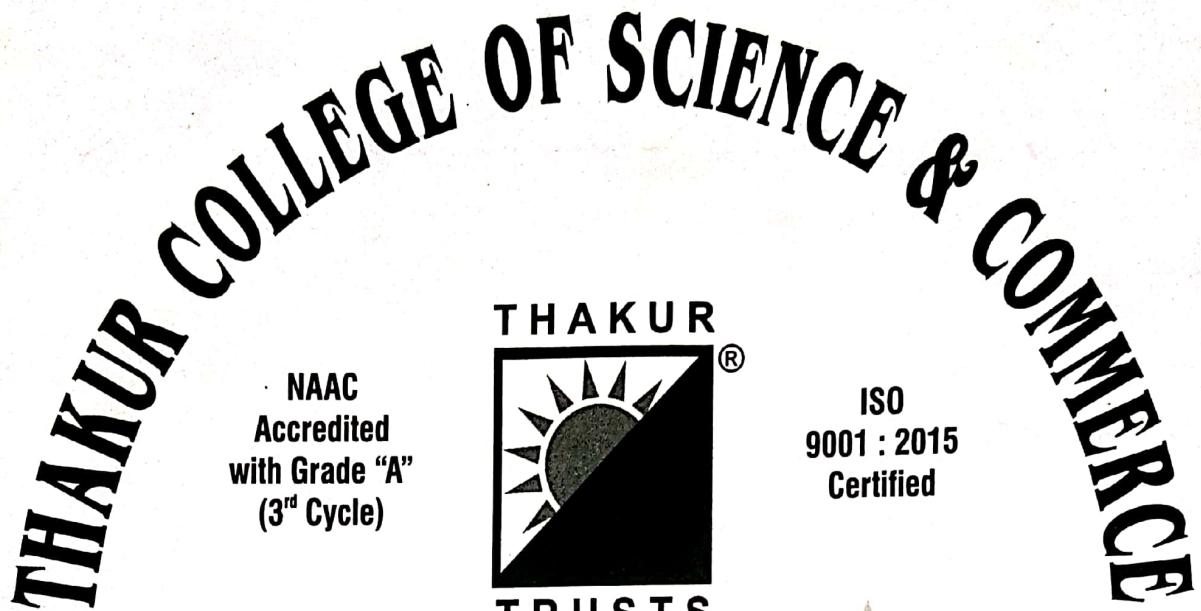


PERFORMANCE

Term	Remarks	Staff Member's Signature
I	✓.6	✓.6
II		

Exam Seat No. _____



Degree College
Computer Journal
CERTIFICATE

SEMESTER II UID No. _____

Class FYBSC CS Roll No. 1738 Year 2019-2020

This is to certify that the work entered in this journal
is the work of Mst. / Ms. Anukai Suraj

Shivkumar
who has worked for the year 2019-2020 in the Computer
Laboratory.

Teacher In-Charge

Head of Department

Date : _____

Examiner

INDEX

No.	Title	Page No.	Date	Staff Member's Signature
G				
01	To Search a number from list using sorted Linear sorted.	28		WJ
02	To Search a number from list using Linear Unsorted.	30		WJ
03	To Search a number using binary search	32		WJ
04	To sort random number using Bubbler sort	33		WJ
05	To demonstrate Use of Stack	35		WJ
06	To demonstrate Queue add delete	37		WJ
07	To demonstrate Use of circular Queue is a data structure	38		WJ
	Jaspal	88	imported 24/01/20	

★ ★ INDEX ★ ★

PRACTICAL - 01

AIM:- To Search a number from the list using linear 'Sorted' method

THEORY:- SEARCHING and SORTING are different modes or types of data-Structure

* **SORTING** :- To basically sort the inputed data in the ascending or descending manner.

* **SEARCHING** :- To Search elements elements and to display the same.

* **LINEAR SORTING**. In searching that too in linear sorted search the data is arranged in ascending to descending order, that is all what it meant by searching through 'sorted' that is well arranged data

```
File Edit Format Run Options Window Help
print ("Suraj Anakal \n CS 3B \n Batch A")
#LINEAR SORTED
```

```
j=0
a=[2,5,24,30,33,54,60]
print(a)
search=int(input("Enter number to be searched:"))
if ((search>a[0]) or (search<a[6])):
    print("Number does not exist")
else:
    for i in range (len(a)):
        if(search==a[i]):
            print("number found at:",i)
            j=1
            break
    if(j==0):
        print("Number not found")
```

```
File Edit Format Run Options Window Help
Python 3.7.1 (v3.7.1:3cefcd12, Mar 25 2019, 21:26:53) [MSC V.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=====
Suraj Anakal
CS 3B
Batch A
[2, 5, 24, 30, 33, 54, 60]
Enter number to be searched:38
number found at: 4
>>>
=====
RESTART: C:\Users\DELL\Desktop\sorted.py
=====
Suraj Anakal
CS 3B
Batch A
[2, 5, 24, 30, 33, 54, 60]
Enter number to be searched:99
Number does not exist
Number not found
>>>
```

PRACTICAL - 01

AIM: To Search a number from the list using linear sorted method

THEORY: SEARCHING and SORTING are different modes or types of data-structure

* **SORTING:** To basically SORT the inputed data in the ascending or descending manner.

* **SEARCHING:** To Search elements and to display the same.

* **LINEAR SORTED:** In searching that too in linear sorted search the data is arranged in ascending to descencing order, that is all what it meant by searching through 'sorted' that is well arranged data

```
File Edit Format Run Options Window Help
print ("Suraj Anakal \n CS 38\n Batch A")
#LINEAR SORTED
```

```
j=0
a=[2,5,24,30,38,54,60]
print(a)
search=int(input("Enter number to be searched:"))
if ((search==a[0]) or (search==a[6])):
    print("Number does not exist")
else:
    for i in range (len(a)):
        if(search==a[i]):
            print("Number found at:",i)
            j=1
            break
    if(j==0):
        print("Number not found")
```

```
File Edit Shell Debug Options Window Help
Python 3.7.3 (v3.7.3:efactfed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

```
==== RESTART: C:\Users\DELL\Desktop\sorted.py ====
Suraj Anakal
CS 38
Batch A
[2, 5, 24, 30, 38, 54, 60]
Enter number to be searched:38
Number found at: 4
>>>
==== RESTART: C:\Users\DELL\Desktop\sorted.py ====
Suraj Anakal
CS 38
Batch A
[2, 5, 24, 30, 38, 54, 60]
Enter number to be searched:99
Number does not exist
Number not found
>>>
```

Sorted Linear Search:

- The user is supposed to enter data in sorted manner
- User has to give an element for searching through sorted list
- If element is found display with an update as value is stored from location '0'
- If data or element not found print the same
- In sorted order list of elements we can check the condition that whether the entered number lies from starting point till the last element if not then without any processing we can say number not in the list

Q6

85

PRACTICAL-02

AIM: To search a number from the list using linear search.

THEORY: The process of identifying or finding a particular record is called searching.

There are two types of search

i) Linear Search

ii) Binary Search,

The linear search is further classified as Sorted and Unsorted.

- Here we will look on the Unsorted Linear Search
- It is also known as Sequential Search, is a process that checks every element in the list sequentially until the desired element is found when the elements to be searched are not specifically arranged in ascending or descending order.

They are arranged in random manner, that is what it calls Unsorted Linear Search

86

```
File Edit Insert Run Options Window Help
print ("Suraj Anakal in CS 1738 in Batch A")
LINEAR UNSEARCHED
j=0
l=[23, 9, 29, 6, 10, 15]
print(l)
search=int(input("Enter number to be searched"))
for i in range (len(l)):
    if (l[i]==search):
        print ("Number found at",i)
        j+=1
        break
if (j==0):
    print("Number not found")
```

```
File Edit Shell Debug Options Window Help
Python 3.7.3 (v3.7.3:efacd612, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Inte
l) on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\DELL\Desktop\LINEAR UNSEARCHED.py =====
Suraj Anakal
CS 1738
Batch A
[23, 9, 29, 6, 10, 15]
Enter number to be searched23
Number found at 0
>>>
===== RESTART: C:\Users\DELL\Desktop\LINEAR UNSEARCHED.py =====
Suraj Anakal
CS 1738
Batch A
[23, 9, 29, 6, 10, 15]
Enter number to be searched03
Number not found
>>>
```

Unsorted Linear Search:

- The data is entered in random manner
- The user needs to specify the elements to be searched in the entered list.
- Check the condition that whether the entered number matches if it matches then display the location plus increment 1 as data is stored from location zero.
- If all element are checked one by one and element not found then prompt that number is not found.

Wt

PRACTICAL - 03

AIM:- To search a number from the given sorted list using binary search

THEORY:- A linear search is also known as a half-interval search is an algorithm used in the computer science to locate a specified value (key) within an array. For the search to binary, the array must be binary and be sorted in either ascending or descending order.

At each step of the algorithm of comparison is made and the procedure branches into one of two directions.

Specifically, the key value is compared to the middle elements of the array. If the key value is less than or greater than this middle elements, the algorithm knows which half of the array to continue searching in because the array is sorted.

This process is repeated on progressively smaller segments of array until the value is found.

Because each step in the algorithm divides the array size in half, a binary search will complete successfully in logarithmic time

```
#SOURCE CODE:
print("FYCS CS ANAKAL SURAJ 1738")
a=[3,4,21,23,25,29,64]
print(a)
search=int(input("Enter Number To Be Searched:"))
l=0
h=len(a)-1
m=int((l+h)/2)
if((search<=a[l]) or (search>=a[h])):
    print("Does Not Exists")
elif(search==a[l]):
    print("Number Found At:",l)
elif(search==a[h]):
    print("Number Found At:",h)
else:
    while(l!=h):
        if(search==a[m]):
            print("Number Found At:",m)
            break
        else:
            if(search<a[m]):
                h=m
                m=int((l+h)/2)
            else:
                l=m
                m=int((l+h)/2)
    if(search!=a[m]):
        print("NO Not Found")
        break
```

OUTPUT:

```
>>>
FYCS CS ANAKAL SURAJ 1738
[3, 4, 21, 23, 25, 29, 64]
Enter Number To Be Searched:23
Number Found At: 3
```

```
>>>
FYCS CS ANAKAL SURAJ 1738
[3, 4, 21, 23, 25, 29, 64]
Enter Number To Be Searched:60
NO Not Found
```

```
>>>
FYCS CS ANAKAL SURAJ 1738
[3, 4, 21, 23, 25, 29, 64]
Enter Number To Be Searched:99
Does Not Exists
```

PRACTICAL - 04

AIM: To Sort given random data by using bubble sort.

THEORY:- Sorting is the type in which any random data is sorted, i.e. arranged in ascending or descending order.

- Bubble Sort Sometimes referred to sinking sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in wrong order.
- The pass through the list is repeated until the list is sorted.
- The algorithm which is a comparison sort is named for the heavy smaller or larger elements "bubble" to the top of the list.
- Although the algorithm is simple, it is too slow as it compare one element to check if condition fails then only swaps otherwise it goes on.

Example:-

$(6\ 1\ 5\ 2\ 4) \rightarrow (1\ 6\ 5\ 2\ 4)$, here algorithm compares the first two elements and get exchange with each i.e $6 > 1$

$(1\ 6\ 5\ 2\ 4) \rightarrow (1\ 5\ 6\ 2\ 4)$ swap $6 > 5$

$(1\ 5\ 6\ 2\ 4) \rightarrow (1\ 5\ 2\ 6\ 4)$ swap $6 > 2$

$(1\ 5\ 2\ 6\ 4) \rightarrow (1\ 5\ 2\ 6\ 4)$ these elements are already

in Order ($a > s$) algorithm does not swap them

Second Pass :-

$(15269) \rightarrow (15269)$
 $(15269) \rightarrow (12569)$ Swap 5 > 2
 $(12569) \rightarrow (12569)$

Third pass

(12569) It checks and gives the data in sorted order.

#SOURCE CODE:

```
print("FYBSC CS ANAKAL SURAJ 1738")
a=[9,45,55,23,34,39]
for p in range(len(a)-1):
    for c in range(len(a)-1-p):
        if(a[c]>a[c+1]):
            t=a[c]
            a[c]=a[c+1]
            a[c+1]=t
print(a)
```

OUT PUT

>>>
FYBSC CS ANAKAL SURAJ 1738
[9, 23, 34, 39, 45, 55]

SOURCE CODE:

```

## Stack ##

print("ANAKAL SURAJ")
class stack:
    global tos
    def __init__(self):
        self.l=[0,0,0,0,0,0]
        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 empty")
        else:
            k=self.l[self.tos]
            print("data=",k)
            self.tos=self.tos-1
s=stack()
s.push(10)
s.push(20)
s.push(30)
s.push(40)
s.push(50)
s.push(60)
s.push(70)
s.push(80)

s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()

```

PRACTICAL - 05

AIM:- To demonstrate the use of Stack

Theory:- In computer science, a Stack is an abstract data type that serves as a collection of elements with two principle operation push, which adds an elements to the collection and pop recently added element that was not yet removed. The order may be LIFO or FILO (Last In First Out)(First In Last Out). The three basic operations are performed in the Stack.

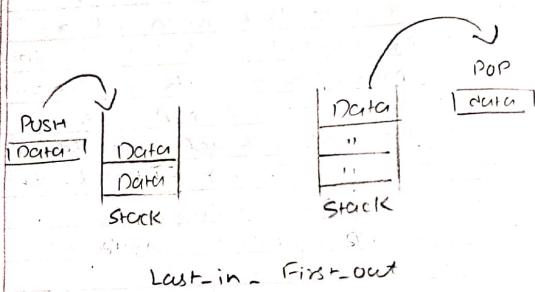
* **PUSH:-** It adds an item in the if then the Stack is full then it is said to be over flow condition.

* **POP:-** Removes an item from the Stack, the items are popped in the reversed order in which they are pushed. If the Stack is empty, then it's said to be underflow condition.

* **Peek or TOP:-** Returns top elements of the Stack.

38

* Is Empty: Returns true if stack is empty else false.



36

OUTPUT:

ANAKAL SURAJ
stack is full
data= 70
data= 60
data= 50
data= 40
data= 30
data= 20
data= 10
stack empty

SOURCE CODE:

```

## Queue add and Delete ##

class Queue:
    global r
    global f
    def __init__(self):
        self.r=0
        self.f=0
        self.l=[0,0,0,0,0]
    def add(self,data):
        n=len(self.l)
        if self.r<n-1:
            self.l[self.r]=data
            self.r=self.r+1
        else:
            print("Queue is full")
    def remove(self):
        n=len(self.l)
        if self.f<n-1:
            print(self.l[self.f])
            self.f=self.f+1
        else:
            print("Queue is empty")
Q=Queue()
Q.add(30)
Q.add(40)
Q.add(50)
Q.add(60)
Q.add(70)
Q.add(80)
Q.remove()
Q.remove()
Q.remove()
Q.remove()
Q.remove()
Q.remove()

```

OUTPUT:

```

Queue is full
30
40
50
60
70
Queue is empty

```

PRACTICAL-06

AIM:- To demonstrate Queue Add And delete

THEORY:-

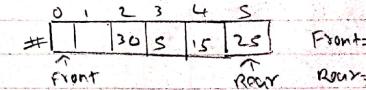
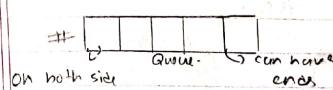
Queue is a linear data structure where the first element is inserted from one end is called REAR and deleted from the outer end is called FRONT.

Front points to the beginning of the queue follows the FIFO (First in First Out) structure. According to its FIFO structure element inserted first will also be removed first.

In a queue, one end is always used to insert data (enqueue) and the other is used to delete data (dequeue) because queue is open at both of its end, Enqueue() can be termed as add() in queue i.e. adding a element in a queue. Dequeue() can be termed as delete or remove i.e. deleting or removing the elements.

Front is used to get the front data item from a queue.

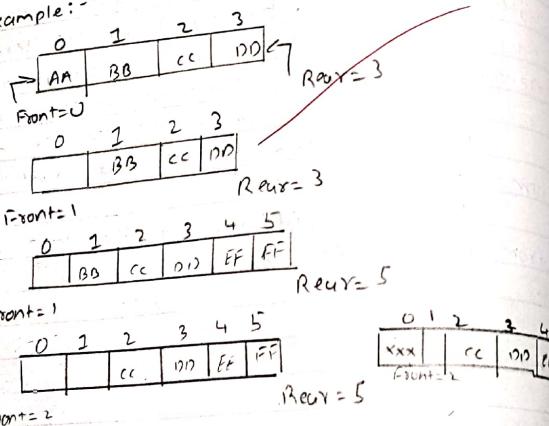
Rear is used to get the last item from a queue.



Aim:- To demonstrate the use of circular queue is data structure.

Theory:- The queue that we implement using an array suffer from one limitation. In that implementation there is a possibility that the queue is reported as full, even though is actually there might be empty slots at the beginning of the queue. In circular queue as circular queue, then in that we keep on adding the element to the queue and reach the end of the array. The next element is stored in the first slot of the array.

Example:-



SOURCE CODE:

```
#(circular queue)
class Queue:
    global r
    global f
    def __init__(self):
        self.r=0
        self.f=0
        self.l=[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.l)
        if self.f==n-1:
            print("data removed:",self.l[self.f])
            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()
Q.add(44)
Q.add(55)
Q.add(66)
Q.add(77)
Q.add(88)
Q.add(99)
Q.remove()
Q.add(66)
```

OUTPUT :

```
data added: 44
data added: 55
data added: 66
data added: 77
data added: 88
data added: 99
data removed: 44
```

PRACTICAL - 08

Aim :- To Demonstrate the use of Linked list in data structure.

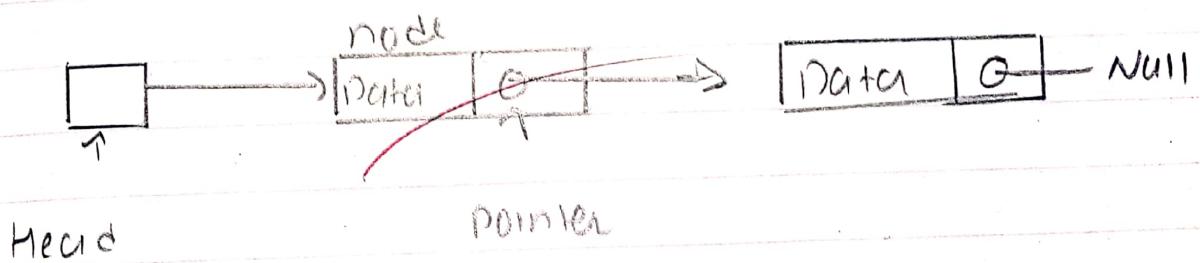
Theory :- A linked list is a sequence of the data structure. Linked list is a sequence of links which contains items. Each link contains a connection to another link.

- Link :- Each link of linked list can store a data called an element.

- NEXT :- Each link of a linked list can store a link to the next link called NEXT.

- LINKED :- A linked list contains link to the first link called first.

Representation of Linked List:



88

* TYPES OF Linked List

- Simple.
- Doubly
- Circular

* Basic Operator

- Insertion
- Deletion
- Display
- Search
- Delete

40

```
##LINKLIST##
print("ANAKAL SURAJ 1738")
class node:
    global data
    global next
    def __init__(self,item):
        self.data=item
        self.next=None
class linkedlist:
    global s
    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)
        if self.s==None:
            self.s=newnode
        else:
            newnode.next=self.s
            self.s=newnode
    def display(self):
        head=self.s
        while head.next!=None:
            print(head.data)
            head=head.next
        print(head.data)
start=linkedlist()
start.addl(50)
start.addl(60)
start.addl(70)
start.addl(80)
start.addB(40)
start.addB(30)
start.addB(20)
start.display()
```

```
OUTPUT:
ANAKAL SURAJ 1738
20
30
40
50
60
70
80
```

```

##Postfix##
print("ANAKAL SURAJ 1738")
def evaluate(s):
    k=s.split()
    n=len(k)
    stack=[]
    a=[]
    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))
        else:
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)/int(a))
    return stack.pop()
s="2 3 9 * +"
r=evaluate(s)
print("The evaluated value is:",r)

```

OUTPUT:

ANAKAL SURAJ 1738
The evaluated value is: 29

PRACTICAL-09AIM :- To Evaluate Postfix Expression Using Stack.

Theory: Stack is an ADT and work on LIFO (Last-In-First-Out) i.e Push and Pop operations. A postfix expression is a collection of operators and operands in which the operators is placed after the operands.

* Steps to be followed :

- Read all symbols one by one from left to right in the given Postfix expression.
- If the reading symbol is operand then push it on to the stack.
- If the reading symbol is operator i.e (+, -, *, /, // etc) then perform two operations and store the two popped operands in two different variables (operand 1 and operand 2). Then perform reading symbol operation using operand 1 and operand 2 and push result back on to the stack.
- Finally, Perform a pop operation and display the popped value as final result.

14

Date: 10/10/2023

Value of Postfix Expression

$$S = 12 \ 3 \ 6 \ 4 \ - \ + \ \star$$

Stack :

4
6
3
12

$$b - a = 6 - 4 = 2 \ // \text{ Store again in stack}$$

2
3
12

$$b + a = 3 + 2 = 5 \ // \text{ Store result in stack}$$

5
12

~~$$b * a = 12 * 5 = 60$$~~

WTF

PRACTICAL-10

Aim: To evaluate i.e. to sort the given data in Quick Sort.

Theory:-

- Quick Sort is an efficient sorting algorithm, type of divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.
- There are many different version of quick sort that pick pivot in different types.
 - a) Always pick the first element as pivot point
 - b) Always pick the last element as pivot
 - c) Pick a random element as pivot
 - d) Pick median as the pivot point
- The key process in quicksort is partition(). Target of partition is given an array and an element X of array as pivot, put X at its current position in sorted array and put all smaller element before X and put all the greater element (greater than X) after X .
- All this should be done in the linear times.

```
##QUICK SORT##  

print("ANAKAL SURAJ 1738")  

print("The Quick sort is:")  

def quickSort(alist):  

    quickSortHelper(alist,0,len(alist)-1)  

def quickSortHelper(alist,first,last):  

    if first<last:  

        splitpoint=partition(alist,first,last)  

        quickSortHelper(alist,first,splitpoint-1)  

        quickSortHelper(alist,splitpoint+1,last)  

def partition(alist,first,last):  

    pivotvalue=alist[first]  

    leftmark=first+1  

    rightmark=last  

    done=False  

    while not done:  

        while leftmark<rightmark and alist[leftmark]<=pivotvalue:  

            leftmark=leftmark+1  

        while alist[rightmark]>=pivotvalue and rightmark>=leftmark:  

            rightmark=rightmark-1  

        if rightmark<leftmark:  

            done=True  

        else:  

            temp=alist[leftmark]  

            alist[leftmark]=alist[rightmark]  

            alist[rightmark]=temp  

            temp=alist[first]  

            alist[first]=alist[rightmark]  

            alist[rightmark]=temp  

            return rightmark  

    alist=[9,36,3,23,45,67,87,99,30]  

    print(alist)  

===== RESTART: C:\Users\DELL\Documents\Quick Sort.py ======  

ANAKAL SURAJ 1738  

The Quick sort is:  

[3, 9, 23, 30, 36, 45, 67, 87, 99]  

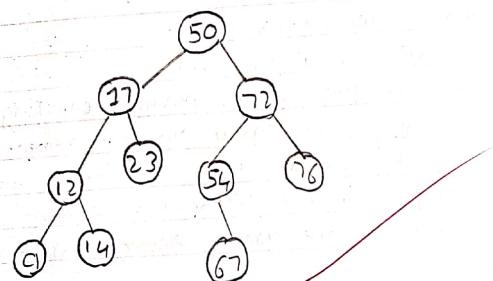
>>>
```

PRACTICAL 17

Aim:- Binary Tree and Traversal

Theory :- A Binary tree is a special type of tree in which every node or vertex has either no child or one child node or two child node.
A binary tree is an important class of data structure in which a node can have at most two children.

Diagrammatic Representation of Binary Search Tree



Traversal is a process to visit all the nodes of tree and may print their value to :
The are 3 ways which we use to traverse tree

SOURCE CODE:

```
#Binary Tree and Traversal  
print("ANAKAL SURAJ 1738")  
class Node:  
    global r  
    global l  
    global data  
    def __init__(self,l):  
        self.l=None  
        self.data=l  
        self.r=None  
class Tree:  
    global root  
    def __init__(self):  
        self.root=None  
    def add(self,val):  
        if self.root==None:  
            self.root=Node(val)  
        else:  
            newnode=Node(val)  
            h=self.root  
            while True:  
                if newnode.data<h.data:  
                    if h.l==None:  
                        h.l=newnode  
                    else:  
                        h=h.l
```

```

h.l=newnode
print(newnode.data,"Added On Left Of",h.data)
break

else:
    if h.r!=None:
        h=h.r
    else:
        h.r=newnode
        print(newnode.data,"Added On Right Of",h.data)
        break

def preorder(self,start):
    if start!=None:
        print(start.data)
        self.preorder(start.l)
        self.preorder(start.r)

def inorder(self,start):
    if start!=None:
        self.inorder(start.l)
        print(start.data)
        self.inorder(start.r)

def postorder(self,start):
    if start!=None:
        self.postorder(start.l)
        self.postorder(start.r)
        print(start.data)

T=Tree()

```

80	
70	
10	
60	
15	
12	
78	
85	
88	
Inorder:	
10	
12	
15	
60	
70	
78	
80	
85	
88	
100	
Postorder:	
12	70
15	88
60	85
10	80
78	100

T W

```
T.add(100)
T.add(80)
T.add(70)
T.add(85)
T.add(10)
T.add(78)
T.add(60)
T.add(88)
T.add(15)
T.add(12)
print("Preorder:")
T.preorder(T.root)
print("Inorder:")
T.inorder(T.root)
print("Postorder:")
T.postorder(T.root)
```

OUTPUT:

>>>

ANAKAL SURAJ 1738

80 Added On Left Of 100

70 Added On Left Of 80807

12 Added On Left Of 15

Preorder:

100

INORDER

PREFORDER

POSTORDER

In-Order :- The left-Subtree is visited 1st then the root and later the right subtree. We should always remember that every node may represent a subtree itself. Output produced is sorted key values in ASCENDING ORDER.

PREF-ORDER : The root node is visited 1st then the left subtree and finally the right subtree

POST-Order : The root node is visited last left subtree, then the right subtree and finally root node.



P.D.

PRACTICAL - 22

Aim: Merge SORT

Theory:-
Merge Sort is a sorting technique based on divide and conquer technique with the worst-case time complexity being $O(n \log n)$, it is one of the most respected algorithms. It first divides the array into equal halves and then combines them in sorted manner.

Merge sort divide input array in two halves, itself for two and then merge the two sorted halves. The merge() function is used for merging 2 halves. The merge($curr, l, m, r$) is a key process that assumes that arr[$l \dots m$] and arr[$m+1 \dots r$] are sorted and merges the two sorted sub-arrays into one.

SOURCE CODE:

```
#MergeSort
print("ANAKAL SURAJ 1738")
def sort(arr,l,m,r):
    n1=m-l+1
    n2=r-m
    L=[0]*(n1)
    R=[0]*(n2)
    for i in range(0,n1):
        L[i]=arr[l+i]
    for j in range(0,n2):
        R[j]=arr[m+1+j]
    i=0
    j=0
    k=l
    while i<n1 and j<n2:
        if L[i]<=R[j]:
            arr[k]=L[i]
            i+=1
            j+=1
            k+=1
        else:
            arr[k]=R[j]
            j+=1
            k+=1
    while i<n1:
        arr[k]=L[i]
        i+=1
        k+=1
    while j<n2:
        arr[k]=R[j]
        j+=1
        k+=1
def mergesort(arr,l,r):
    if l<r:
        m=int((l+r-1)/2)
        mergesort(arr,l,m)
        mergesort(arr,m+1,r)
        sort(arr,l,m,r)
        arr=[12,23,34,56,78,45,86,98,42]
        print("Before Mergesort\n",arr)
        n=len(arr)
        mergesort(arr,0,n-1)
        print("After Mergesort\n",arr)
```

ab

OUTPUT:

>>>

ANAKAL SURAJ 1738

Before Mergesort

[12, 23, 34, 56, 78, 45, 86, 98, 42]

After Mergesort

[12, 23, 34, 56, 42, 45, 78, 86, 98]

42