☆☆ INDEX ☆☆

DATA STRUCTURES

No.	Title	Page No.	Date	Staff Member's Signature
1)	To search a number from the list using unsorted method	39	27/11/19	
2)	To search a number from the list using Sorted method	41	27/11/19	
3)	To search a number from the given sorted list using binary search	43	4/11	
4)	To sort given random data by using bubble sort	44		
5)	To demonstrate the use of Stacks.	46		
6)	To demonstrate Queue add and delete.	47		
7)	To demonstrate the use of circular Queue	49		

☆☆ INDEX ☆☆

PRACTICAL-1

AIM :- To Search a number from the list using linear unsorted

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

There are two type of Search

- Linear Search
- Binary Search

The linear Search is further classified as

- SORTED
- UNSORTED.

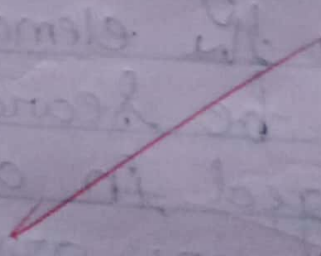
Here we will look on the UNSORTED Linear Search. It is also known as Sequential Search is a process that checks every element in the list Sequentially until the desired element is found when the element is found when the element to be Searched are not specifically ~~unfit~~ arranged in ascending or decending order. They are arranged in random manner That is what it called Unsorted linear Search.

Unsorted linear Search :-

- The data is entered in random manner.
- User need to Specify the element to be

Searched in the entered list.

- check the condition that whether the entered number matches if it matches then display the location plus matches then display then ~~display~~ location plus matches then ~~display~~ 1 as data is stored from location zero
- If all element are checked one by one and element not found then prompt message number not found



Input :-

```
j=0
a=[2,23,3,56,29,74,16,82,6]
search=int(input("Enter number to be search: "))
print("Adarsh Pandey \n 1739")
for i in range (len(a)):
    if(search==a[i]):
        print("No. found at  ",i,"place")
        j=1
        break
else:
    print("No. Not found")
```

Out put :-

```
================== RESTART: D:/ds practical/practical 1.py
Enter number to be search: 56
Adarsh Pandey
 1739
No. found at  3 place
>>>
================== RESTART: D:/ds practical/practical 1.py
Enter number to be search: 100
Adarsh Pandey
 1739
No. Not found
```

PRACTICAL-2

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

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

SORTING - It basically SORT the inputted data in ascending or decending manner.

SEARCHING - It search element and to display the same

In searching that two in LINEAR SORTING search,

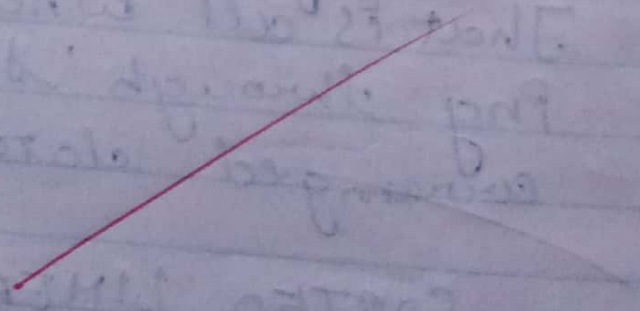
It

The data is arranged in ascending to decending or decending to ascending That is all what it meant by searching through 'Sorted' that is well arranged data.

SORTED LINEAR Search :

- The user is supposed to enter data in Sorted search.
- User has the given element for searching through Sorted list

- If element is found display with an updation as value is sorted from location '0'.
- If data or element are not found print the same
- In sorted order list of element we can check the condition that whether the entred number lies from starting point till the last element if not then without any processing we can say number not in the list.



Input :-


```
j=0
a=[2,23,24,25,26,28,30,32,43,54,76]
search=int(input("Enter number to be search: "))
print("Adarsh Pandey \n 1739")
if(search<a[0]) or (search>a[len(a)-1]):
   print("No. do not exist")
for i in range (len(a)):
   if(search==a[i]):
      print("No. found at  ",i,"place")
      j=1
      break
   else:
      print("No. Not found")
```

Output :-

```
========== RESTART: D:/ds practical/practical 1.py
Enter number to be search: 30
Adarsh Pandey
 1739
No. found at   6 place
>>>
========== RESTART: D:/ds practical/practical 1.py
Enter number to be search: 27
Adarsh Pandey
 1739
No. Not found
>>>
========== RESTART: D:/ds practical/practical 1.py
Enter number to be search: 100
Adarsh Pandey
 1739
No. do not exist
```

Source code:

```
print("Adarsh pandey \n1739")
a=[4,6,8,11,15,16,25,65]
print(a)
search=int(input("Enter number to be searched from the list:"))
l=0
h=len(a)-1
m=int((l+h)/2)
if((search<a[l]) or (search>a[h])):
  print("Number not in RANGE!")
elif(search==a[h]):
  print("number found at location :",h+1)
elif(search==a[l]):
  print("number found at location :",l+1)
else:
  while(l!=h):
    if(search==a[m]):
      print ("Number found at location:",m+1)
      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("Number not in  given list!")
      break
```

Output:

Case1:
Adarsh pandey
1739
[4,6,8,11,15,16,25,65]
Enter number to be searched from the list:16
Number found at location: 6
Case2:
Adarsh pandey
1739
[4,6,8,11,15,16,25,65]
Enter number to be searched from the list:100
Number not in RANGE!
Case3:
Adarsh pandey
1739
[4,6,8,11,15,16,25,65]
Enter number to be searched from the list:1
Number not in given list!

PRACTICAL-03

Aim : To Search a number from the given Sorted list using Binary Search

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

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

Specifically, the key value is compared to the middle element of the array.

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

This process is repeated on progressively smaller segments of the array until the value is located Because each step in the algorithm divides the array size in half, a binary search will complete successfully in logarithmic time.

PRACTICAL-04.

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

Theory:- SORTING is type in which any random data is Sorted i.e arranged in ascending or descending order.
BUBBLE Sort Sometimes refferred is as Sinking Sort.
Is a Simple Sorting algorithm that repeatedly Steps through the lists, compare adjacent element and Swape them if they are in wrong order
They pass through the lise is repeated until the lise is sorted. The algorithm which is a comparision Sort is named for the way Smaller or larger element "bubble" to the top of the list.
Although the algorithm is Simple, It too Slow as compares one element check if condition fails than only Swape otherwise goes on.

Example:-
First pass:
(15428) → (15428) Here algorithm compare the first two element and Swape.
5 > 1
(15428) → (14528) Swap Since 5 > 4
(14528) → (14258) Swap Since 5 > 2
(14258) → (14258) Swap Since these element

Source code:

```
print("Adarsh pandey \n1739")
a=[20,24,16,28,22]
print("Before BUBBLE SORT elemets list: \n ",a)
for passes in range (len(a)-1):
    for compare in range ((len(a)-1)-passes):
        if(a[compare]>a[compare+1]):
            temp=a[compare]
            a[compare]=a[compare+1]
            a[compare+1]=temp
print("After BUBBLE SORT elemnts list: \n",a)
```

Output:

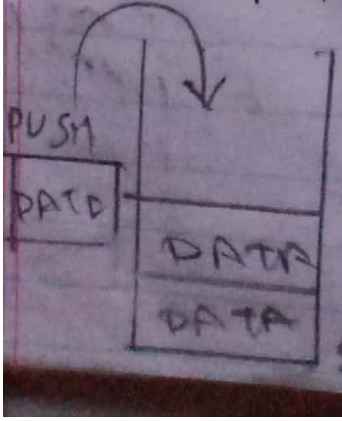
```
Adarsh pandey
1739
Before BUBBLE SORT elemets list:
 [20,24,16,28,22]
After BUBBLE SORT elemnts list:
 [16,20,22,24,28]
```

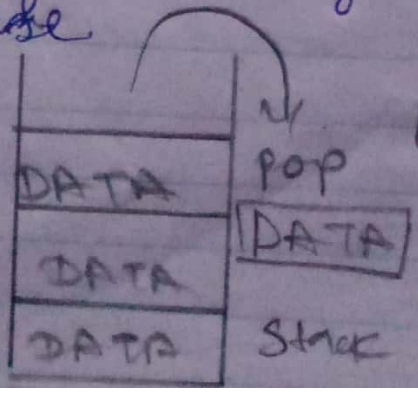
PRACTICAL-05

AIMS- To demonstrate are use of Stack.

Theory :- In computer Science, a Stack is an abstract data type that serves as a collection of elements with two principal operation push, which adds a element to the collection and pop, which removes the most recently added element that was not yet removed. The order may be LIFO (Last in first out) or FILO (First in Last Out). Three Basic Operation are performed in the Stack

- PUSH :- Add an item in the Stack. If the Stack is full it is said to be over flow condition.
- POP :- Removes an item from the Stack the item are popped in the removed ~~oder~~ ordered in which they are push If the Stack is empty, then it is said to be an under flow condition
- Peek or TOP :- Return top element of Stack
- Empty :- Return true if Stack is empty else false





Last in - first out

= Stack ##
URCE CODE:

```
print("Adarsh kumar pandey\n 1739")
class stack:
        global tos
        def __init__(self):
             self.l=[0,0,0,0,0,0,0]
             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()
```

42

OUTPUT:

```
Adarsh kumar pandey
1739
Stack is full
data=70
data=60
data=50
data=40
data=30
data=20
data=20
Stack is empty
```

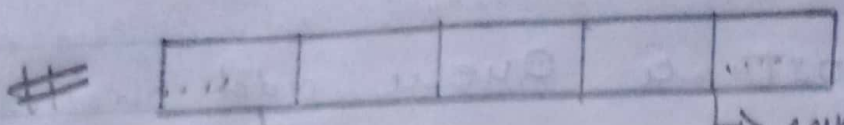
```
## Queue add and Delete ##
SOURCE CODE
print("Adarsh kumar pandey\n 1739")
class Queue:
    global r
    global f
    def __init__(self):
        self.r=0
        self.f=0
        self.l=[0,0,0,0,0,0]
    def add(self,data):
        n=len(self.l)
        if self.r<n-1:
            self.l[self.r]=data
            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:
Adarsh kumar pandey
 1739
Queue is full
Queue is full
30
40
50
60
Queue is empty
Queue is empty
```

PRACTICAL: 06.

AIM :- To demostrate queue add and delete

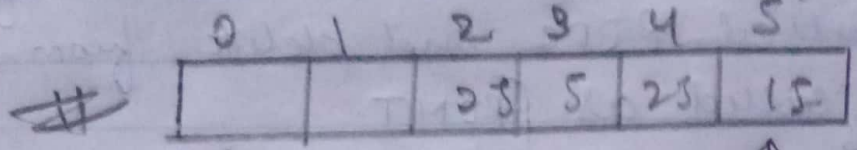
Theory :- Queue is a linear data structure where the first element is inserted from one end called REAR and delete from the other end called as FRONT.
Front points in the beginning of the queue and Rear point to the end of the queue
Queue follows the FIFO (First in- First out) structure.
According to 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 ends.
enqueue() can be termed as add() in queue i.e adding a element in queue.
dequeue() can be termed as delete or Removes i.e deleting or removing of element.
Front is used to get the front data from a queue.
Rear is used to get the last item a queue.



can have end.

on both side

Queue.



Front = 2
REAR = 5

#(circular queue)
SOURCE CODE:

```
print("Adarsh kumar pandey\n 1739")
class Queue:
print()
    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.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)
```

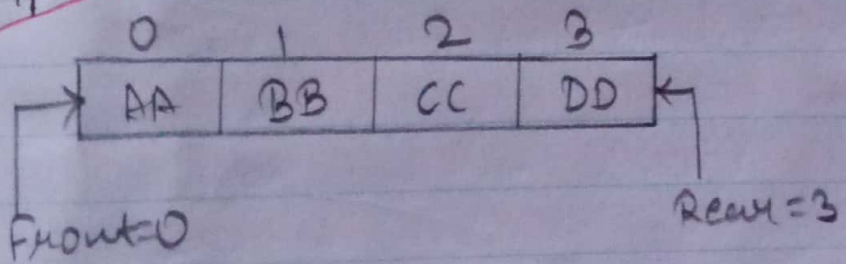
OUTPUT:

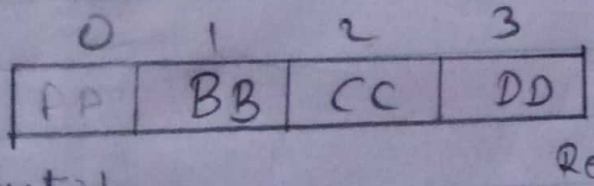
```
Adarsh kumar pandey
 1739
data added:44
data added:55
data added:66
data added:77
```

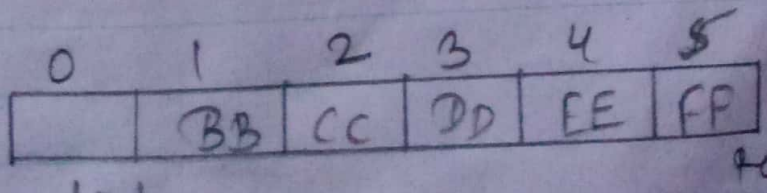
Aim :- To demonstrate the use of circular Queue in data Structure.

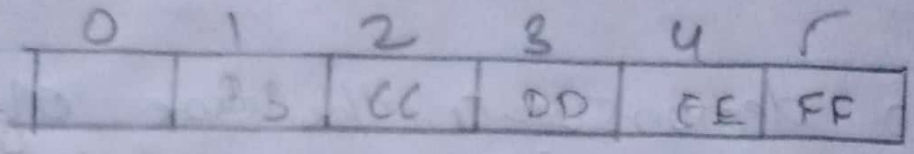
Theory :- The queue that use implement using an array suffer from one limitation. In that implementation there is a posibility that the queue is reported as full, even though in actually there might be empty slote at the beginning of the queue. To overcome this limitation we can implement queue as circular queue In circular queue we go on adding the element to the queue and reach the end of the array. The next element is sorted in the first slot of the ~~array~~ array.

Example :-



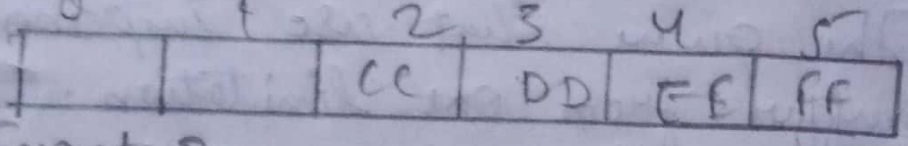






Front = 2

Rear = 5



Front = 2

Rear = 5

```
data added:88
data added:99
data removed:44
```

SOURCE CODE:

```
print("Adarsh kumar Pandey")
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:

```
Adarsh kumar Pandey
20
30
```

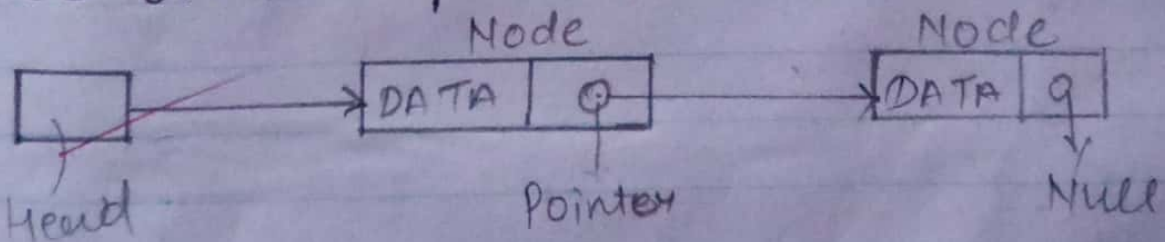
PRACTICAL:-08

Aim :- To demonstrate the use of Linked list in data Structure.

Theory :- A linked list is a sequence of data structure linked list is a sequence of links which contains item 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 contain a link to the next link called NEXT.
- LINKED LIST :- A Linked list contain the connection link to the first link called First

LINKED LIST Representation :-



TYPES OF LINKED LIST :-

- Simple
- Doubly
- Circular

Basic Operations

- Insertion
- Deletion
- Display
- Search
- Delete

40
50
60
70
80

SOURCE CODE:

```
print("Adarsh kumar Pandey")
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))
        else:
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)/int(a))
    return stack.pop()
s="13 4 6 5 - + *"
r=evaluate(s)
print("The evaluated value is:",r)
```

OUTPUT:

```
Adarsh kumar Pandey
The evaluated value is: 65
```

PRACTICAL : 09.

AIM :- To evaluate postfix expression using Stack.

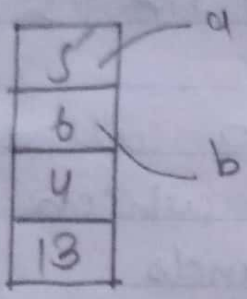
Theory :- Stack is an (ADT) and work on LIFO (Last_in first_out) i.e is PUSH and POP Operations.
A postfix expression is a calledtion of operator and operands in which operator is placed after the operands

Step to the followed :
1. Read all the Symbols one by one from left to right in the given postfix expression.
2. If the reading Symbol is operand then push it on to the Stack
3. If the reading Symbol is operator (+, -, *, /, etc) then perform two poped operand in two popped operand in two different variable (operand 1 & operand 2). Then perform reading Symbols operation using operand 1 operand 2 and push result back on to the Stack.
4. Finally! Perform a POP operation and Display the popped value as final result.

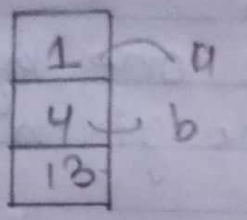
value of postfix expression:

S = 13 4 6 5 - + *

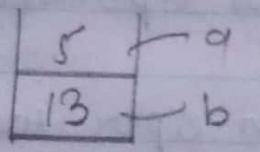
Stack:-



b - a = 6 - 5 = 1 // Store again in stack



b + a = 5 // Store result in stack



a * b = 65 //

practical no:10 QUICK SORT

```
print("Name:Adarsh kumar pandey/nRollno:1739")

# SOURCE CODE:
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=[42,54,45,67,89,66,55,80,100]
quickSort(alist)
print(alist)
```

OUTPUT:

```
Name:Adarsh kumar pandey
Rollno:1739

[42, 45, 54, 55, 66, 89, 67, 80, 100]
```

PRACTICAL :- 10

AIM :- To evaluate i.e to Sort the given data in Quick Sort.

Theory :- Quicksort is an efficient Sorting algorithm type of a Divide & Conquer algorithm. It picks an element at pivot and partition the given array around the picked pivot. There are many different versions of quick Sort that pick pivot in different ways :-

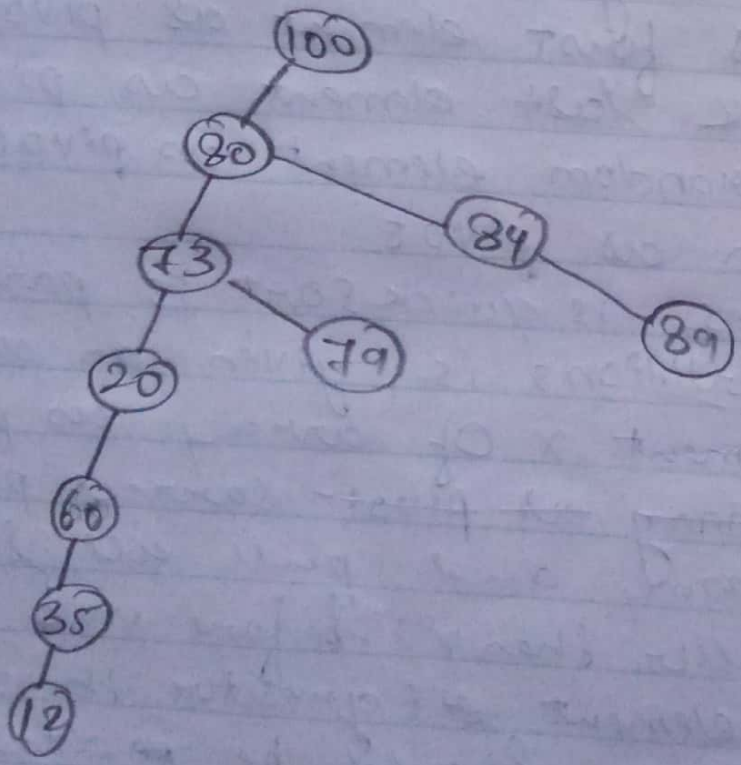
1) Always picks first element as pivot.
2) Always pick last element as pivot
3) Pick a random element as pivot.
4) Pick button as pivot.

The key process is quicksort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x of ~~array as pivot~~ correct position in Sorted array and put all Smaller element (Smaller than x) before x & put all greater element (greater than x) after x. All this Should be done in linear time.

Practical :- 11

AIM :- Binary tree and transversal.

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 a tree data structure in which a node can have as most two children.



Diagramatic Representation of BINARY SEARCH TREE.

```
print("Name:Ashish Kumar Pandey \nRoll No.:1739")
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=h.l
                                else:
                                        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:
```

32.

```
                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.inorder(start.l)
                self.inorder(start.r)
                print(start.data)
T=Tree()
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")
```

Traversal is a process to visit all the node of a tree and many perform their value too.

There are three way which use to traverse a tree.

→ INORDER
→ PREORDER
→ POSTORDER

INORDER :- The left-Subtree is ~~visited~~ 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 value is ASCENDING ORDER

PRE-ORDER :- The root node is visited 1st then the left Subtree and finally the right Subtree.

POST-ORDER :- The root note is visited last left Subtree, then the right Subtree and finally root node.

```
T.postorder(T.root)
```

```
Output:
Name:Adarsh Kumar Pandey
Roll No.:1739
80 added on left of 100
70 added on left of 80
85 added on right of 80
10 added on left of 70
78 added on right of 70
60 added on right of 10
88 added on right of 85
15 added on left of 60
12 added on left of 15
Preorder
100
80
70
10
60
15
12
78
85
88
Inorder
10
12
15
60
70
78
80
85
88
100
Postorder
10
12
15
60
70
78
80
85
88
100
```

```
#Merge Sort
print("Name:Adarsh Kumar Pandey \nRoll No.:1739")
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+=l
            j+=l
            k+=l
        else:
            arr[k]=R[j]
            j+=1
            k+=1
    while i<n1:
        arr[k]=L[i]
        i+=1
        k+=1
    while j<n2:
```

Practical: 12

AIM:- Merge Sort.

Theory :- Merge Sort is a Sorting technique based on divide and conquer technique with worst-case time complexity being $O(n \log n)$, it one of the most respected algorithm.

- Merge Sort first divides the array in to equal halves and then combines them in a sorted manner.

It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves the merge() function is used for merging 2 halves the merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merge the two sorted sub-array into one

```
                    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]
n=len(arr)
mergesort(arr,0,n-1)
print(arr)
```

Output:

Name:Adarsh Kumar Pandey

Roll No.:1739

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