PRACTICAL NO:- 01

Aim :- To search a number from a given list by user using linear unsorted search

Theory :-

The process of identifying or finding a particular record from given data is called searching. The data can be numbers, alphabets etc. However, the process of searching is divided in two types:

(i) Linear Search
(ii) Binary Search

The linear Search is further classified as

(a) Sorted (b) Unsorted

The linear Search is a process which each element of a given list sequentially until the desired element is found. Unsorted refers to the arrangement of data i.e data presente in unsorted list may not be ascending or descending in order.

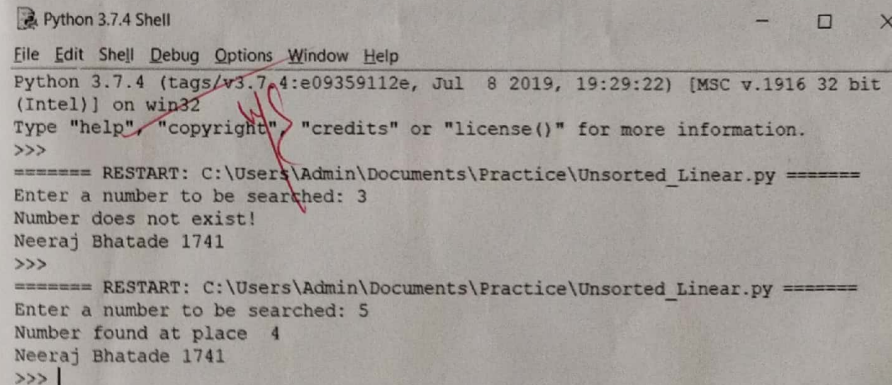
*Unsorted linear Search:

- User needs to provide a number which is to searched from the list.
- This algorithm will compare all the numbers with inputted number one at a time.
- If the match is found, then it will return the position of number in list.
- If all elements are checked and number is not matched, then it will returen a message saying number not found.

PROGRAM :

```
found=False
a=[13,12,28,5,41,69,29,1]
search=int(input("Enter a number to be searched: "))
for i in range(len(a)):
  if (search==a[i]):
print("Number found at place ",i+1)
    found=True
    break
if (found==False):
print("Number does not exist!")
print("Neeraj Bhatade 1741")
```

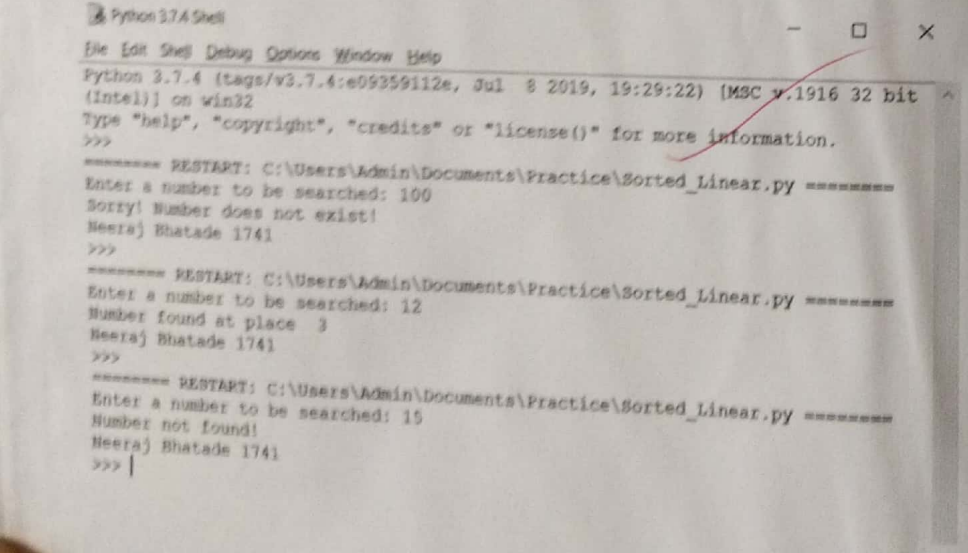
OUTPUT :



PROGRAM:

```
found=False
a=[1,5,12,13,28,29,41,69]
search=int(input("Enter a number to be searched: "))
if (search<a[0] or search>a[len(a)-1]):
print("Sorry! Number does not exist!")
else:
  for i in range(len(a)):
    if (search==a[i]):
print("Number found at place ",i+1)
      found=True
      break
  if (found==False):
print("Number does not exist!")
print("Neeraj Bhatade 1741")
```

OUTPUT:



PRACTICAL NO:- 02

Aim :- To search a number from a given list using the linear sorted method.

Theory :-

The second category in linear search is the sorted linear search method. The term 'Sorting' refers to the process of arranging the given data in either ascending or descending order. Sorting is an effective method which reduces the time required for searching an element in a given list.

Thus, Sorted linear Search is the process where the input provided by the user is being searched in a sorted linear list. Here one can directly check whether the number is present in the list or not i.e if the number is less than first element or greater than last element then the number is not present in the list as those two numbers are the end points from each direction.

* Sorted Linear Search:-

- User needs to provide a input number which will be searched in the list.
- The algorithm will compare the input number with first & last number.
- If it falls in the range, then it will proceed further and compare each element with input number.
- If the match is found then it will return the position of the number in list.

28.

PRACTICAL NO :- 03

Aim :- To search a number from a given sorted list using binary search.

Theory :-

A binary search also known as a half interval search, is an algorithm to locate the specified value in an array of numbers. To use this algorithm over any list, the list should be sorted in either ascending or descending order.

At each step of algorithm a comparision is made and the procedure branches in one of the two direction (left or right). To start this algorithm, first the middle term of the given list is found.

If the input number is equal to the middle term of the list then the position of middle term is returned as the output.

If the input number is greater than the middle term then the second half of the list is selected on which further search will happen. and Now, first term becomes mid+1 and last term remains the same.

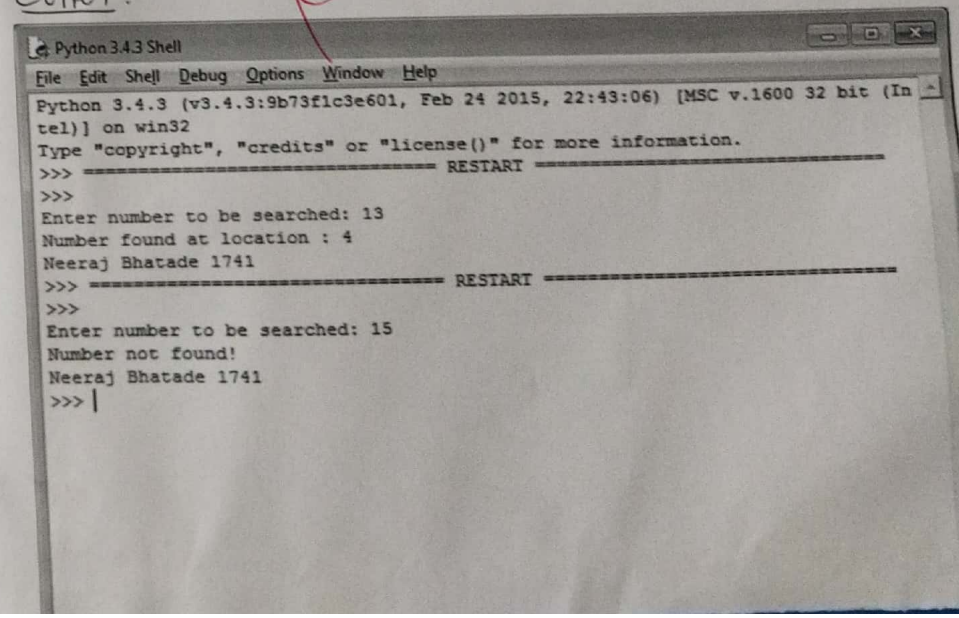
If the input number is less than the middle term then the first half of the list is selected on which further search will happen. Now, last term becomes mid-1 and last term first term remains the same.

Every time a half interval is consider, the first and last elem changes accordingly and new middle term is found.

PROGRAM :

```
a=[1,5,12,13,28,29,41,69]
search=int(input("Enter number to be searched: "))
l=0
r=len(a)-1
while(True):
  m=(l+r)//2
  if(l>r):
    print("Number not found!")
    break
  if(search==a[m]):
    print("Number found at location :",m+1)
    break
  elif(search<a[m]):
    r=m-1
  else:
    l=m+1
print("Neeraj Bhatade 1741")
```

OUTPUT :



PROGRAM:

```
print("Neeraj Bhatade 1741")
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+=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.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)
```


PRACTICAL No:-04

Aim : To demonstrate use of stack.

Theory :

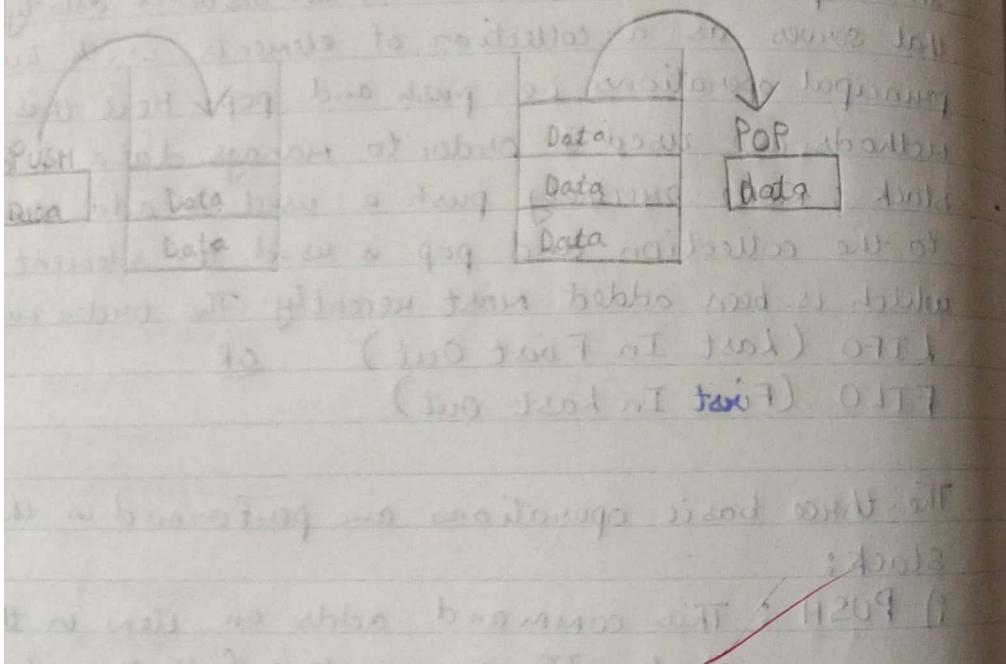
In computer science, a stack is an abstract data type that serves as a collection of elements with two principal operations i.e push and pop. Here this two methods are used in order to manage data in a stack; more precisely push is used to add elements to the collection and pop is used the element which is been added most recently. The order may be LIFO (Last In First Out) OR FILO (First In Last Out)

The three basic operations are performed in the stack:

i) PUSH : This command adds an item in the stack. If the stack is full then it is said to be stack over flow condition.

ii) Pop : Removes an item from the stack. The items are poped in the required order in which they are pushed. If the stack is empty, then it is said to be an Stack underflow condition.

- Top : Returns top element of stack
- isEmpty : Returns true if stack is empty and false if stack is full.




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

OUTPUT :

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/Admin/Documents/Practice/stack.py ==========
Neeraj Bhatade 1741
stack is full!
data:  70
data:  60
data:  50
data:  40
data:  30
data:  20
data:  10
stack is empty!
>>>
```

PROGRAM :

```
print("Neeraj Bhatade 1741")
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+=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+=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)
```


PRACTICAL No :- 05

Aim : To demonstrate Queue add and delete

Theory :

Queue is a linear data structure where the first element is inserted from one end and deleted from the other end. The end from which the element is inserted is called as FRONT and the end from which the data is removed is called REAR. Front points to the begining of the queue.

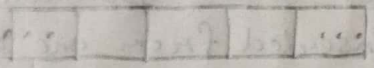
Queue follows the FIFO (First In First Out) structure.

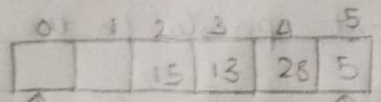
According to its FIFO structure elements inserted first will also be removed first. In a queue one end is always used to insert data and other is used to delete data because queue is is open at both of its ends.

enque() can be termed as add() in queue i.e adding an element in queue.

dequeue() can be termed as delete or remove i.e deleting or removing an element.

Front is used to get the data from the front in a queue
Rear is used to get the last item from the queue






```
Q.remove()
Q.remove()
Q.remove()
Q.remove()
Q.remove()
Q.remove()
```

OUTPUT :

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============ RESTART: C:/Users/Admin/Documents/Practice/Queue.py ============
Neeraj Bhatade 1741
Queue is full!
30
40
50
60
70
Queue is empty!
>>>
```

PROGRAM:

```
print("Neeraj Bhatade 1741")
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+=1
        else:
            s=self.r
            self.r=0
            if self.r<self.f:
                self.l[self.r]=data
                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+=1
        else:
            s=self.f
            self.f=0
```

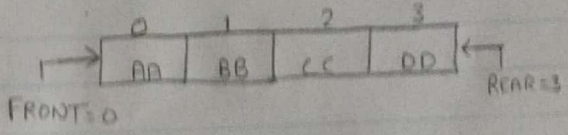

PRACTICAL NO:-06

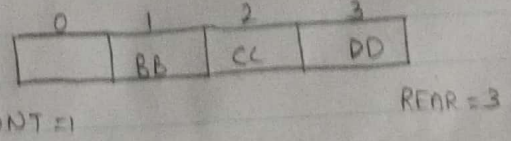
Aim: To demonstrate the use of circular queue in data structure.

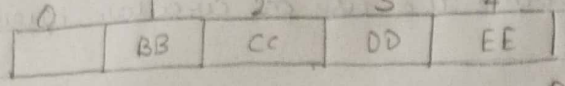
Theory:

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 in actuality there might be empty slots in begining of the queue who some data is either removed or is deleted from the queue. To overcome this limitation we can implement queue as circular queue. In circular queue we go on adding elements to the queue and reach the end of the array. The next element is sorted in first slot of array.

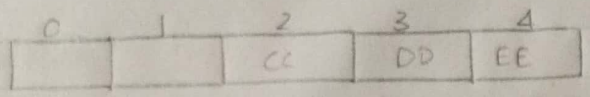
Example:



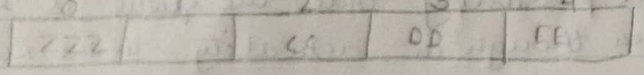




FRONT = 1 REAR = 4



REAR = 4



REAR = 4

```
        if self.f<self.r:
            print(self.l[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 :

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ======== RESTART: C:/Users/Admin/Documents/Practice/Circular_Queue.py ========
Neeraj Bhatade 1741
Data added:  44
Data added:  55
Data added:  66
Data added:  77
Data added:  88
Data added:  99
Data removed:  44
>>>
```

```
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
```


PRACTICAL No:-07

Aim : To demonstrate use of linked list in data structure

Theory :-

A linked list is a sequence of data structures. Linked list is a sequence of links which contains items. This item are as stored in a node which is further divided into two parts. The first part of the node consist of data while the other part of the node contains the location of next node. So in linked list, each link contains a connection to other link.

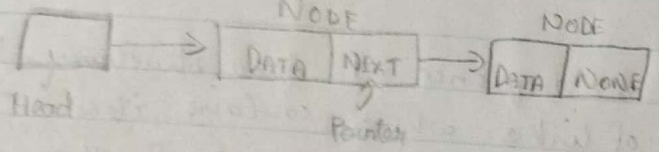
A linked list is of 3 types:
1. Singular linked list
2. Doubly linked list
3. Circular linked list

The singular linked list can store data of type integer, string or any characters with the memory address of their NEXT link.

The Basic Operation that can be performed on linked list are:

1. Insertion
2. Deletion

⑤ Display
④ Search




```
print(head.data)


start=linkedlist()
start.ADDL(50)
start.ADDL(60)
start.ADDL(70)
start.ADDB(40)
start.ADDB(30)
start.ADDB(20)
start.ADDB(10)
start.display()


print("Neeraj Bhatade 1741")
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\Admin\Documents\temp5.py ================
10
20
30
40
50
60
70
Neeraj Bhatade 1741
>>>
```

```
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(a)+int(b))
        elif k[i]=="-":
            a=stack.pop()
            b=stack.pop()
            stack.append(int(a)-int(b))
        elif k[i]=="*":
            a=stack.pop()
            b=stack.pop()
            stack.append(int(a)*int(b))
        else:
            a=stack.pop()
            b=stack.pop()
            stack.append(int(a)/int(b))
    return stack.pop()
s="8 6 9 * +"
r=evaluate(s)
print("The evaluated value is:",r)
print("Neeraj Bhatade 1741")
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Admin\Documents\temp6.py ==========
The evaluated value is: 62
Neeraj Bhatade 1741
>>>
```


PRACTICAL NO:-08

Aim :- To evaluate postfix expression using stack

Theory:

Stack is an ADT and works on LIFO i.e PUSH & POP operation

Steps to be followed:

i) Read all symbols one by one from left to right in given expression

ii) If the reading symbol is operand then push it on to the stack

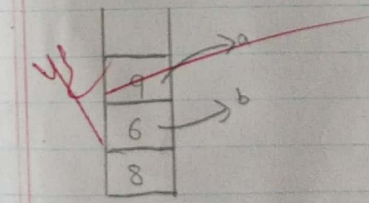
(iii) If symbol is operator then perform Two pop operations and perform the operator action between the popped values.

(iv) Finally! Perform pop to display the result.

Expression:

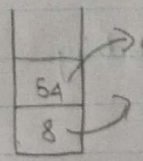
S = 8 6 9 * +

Stack:



b * a = 6 * 9 = 54

Store this value in stack



b + a = 8 + 54 = 62

IA

PRACTICAL No: 09

Aim : To sort given random data by using bubble sort.

Theory:

Sorting is is a process of arranging data in either ascending/descending order. Bubble sort sometimes is a simple sorting algorithm that 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 comparision sort is named for way smaller or larger elements "bubble" to the top of the list.

Although, the algorithm is simple, it is too slow as it compares one element and checks for the condition. If the condition fails then only swapping occurs otherwise the process continues.

Example :

First pass

(2 8 1 3)

As 2<8 no change will occur

8>1 ⇒ (2 1 8 3)

8>3 ⇒ (2 1 3 8)

Second pass

2>1 ⇒ (1 2 3 8)

Now the data has been sorted in the ascending order.

```
a=[13,12,5,28,69,41]
print(a)
for i in range(len(a)-1):
    for j in range(len(a)-1):
        if(a[j]>a[j+1]):
            t=a[j]
            a[j]=a[j+1]
            a[j+1]=t
print("Numbers after Bubble sort: \n",a)
print("Neeraj Bhatade 1741")
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/Admin/Documents/temp7.py ================
[13, 12, 5, 28, 69, 41]
Numbers after Bubble sort:
 [5, 12, 13, 28, 41, 69]
Neeraj Bhatade 1741
>>>
```

```
a=[13,12,5,28,69,41]
print(a)
for i in range(len(a)-1):
  for j in range(len(a)-1):
    if(a[j]>a[i+1]):
      t=a[j]
      a[j]=a[i+1]
      a[i+1]=t
print("Numbers after selection sort: \n",a)
print("Neeraj Bhatade 1741")
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/Admin/Documents/temp7.py ==========
[13, 12, 5, 28, 69, 41]
Numbers after selection sort:
 [5, 12, 13, 28, 41, 69]
Neeraj Bhatade 1741
>>>
```

PRACTICAL NO:-10

Aim: To sort given random data by using selection sort

Theory:

Selection Sort is a simple sorting algorithm. This sorting algorithm is an in-place comparision based algorithm in which the list is divided into two parts, the sorted part at the left end and unsorted part at right end. Initially, the sorted part is empty and unsorted part is entire list.

The smallest element is selected from unsorted array and swapped with the leftmost element, and that element becomes a part of sorted array. This process continues moving unsorted array boundary by one element to the right.

This algorithm is not suitable for large data sets as its average and worst complexities are of $O(n^2)$, where n is number of items.

PRACTICAL NO :- 11

Aim : To sort random data by using quick sort

Theory :

Quick sort is an efficient sorting algorithm. Type of a 'divide and conquer' algorithm. It picks an elements as pivot and partitions the array around picked pivot. There are many different versions of quick sort that pick pivot in different ways.

i) Always pick first element as pivot.
ii) Always pick last element as pivot
iii) Pick a random element as pivot
iv) Pick median as pivot.

The key process in quicksort is partition(). Target of partitions is given array and an element 'x' of array as pivot, Put x at its correct position in sorted array and put all smaller elements before x & greater elements after x.

```
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=[13,12,5,28,69,41,6]
print(alist)
```

```
quickSort(alist)

print("Numbert after quick sort: \n",alist)

print("Neeraj Bhatade 1741")
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/Admin/Documents/temp7.py ==========
[13, 12, 5, 28, 69, 41, 6]
Numbert after quick sort:
 [5, 6, 12, 13, 28, 41, 69]
Neeraj Bhatade 1741
>>>
```

PRACTICAL No :- 12

Aim : To create a Binary Tree and Traverse through it.

THEORY :

Binary Tree is a tree which supports maximum of two children for any node within the tree. Thus any particular node in the tree can have 0 or 1 children.

The first node of Binary tree is also referred to as root as it is the begining of the tree. Further the node are classified into two types :

i) Parent Node
ii) Leaf Node

i) Parent Node : The node which has atleast one child is called as parent node. Even the root is a parent node

ii) Leaf Node : A node which do not have any children is termed as the Leaf node. This node is generally the last node found in the last level of tree.


```
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:
                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(13)
T.add(12)
T.add(28)
T.add(5)
T.add(10)
T.add(8)
T.add(69)
print("preorder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
```


Traversing can be defined as a process of visi every node of tree atleast once. The traversing process can be used to display the data of a specied node i the tree.

- Preorder:

i) Visit the root node.
ii) Traverse the left subtree which may further have left or right subtree.
iii) Traverse right subtree which may further have left/right subt

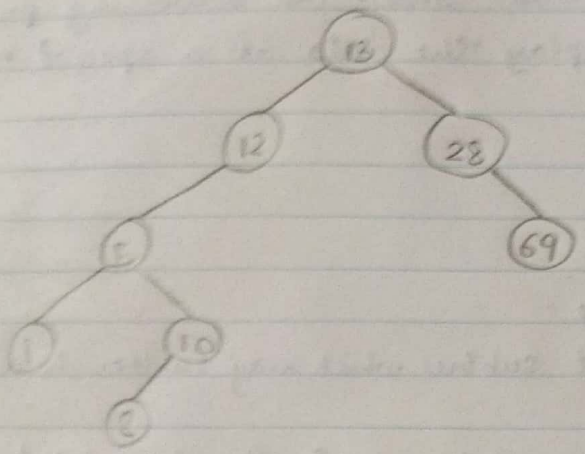
- Inorder:

i) Traverse left subtree which may further have left/right subtr
ii) Visit the root node.
iii) Traverse right subtree which may further have left/right subtre

- Postorder

i) Traverse left subtree which may further have left or right subtree.
ii) Traverse right subtree which may further have left or right subtree.
iii) Visit the root node.

Example :




```
print("postorder")

T.postorder(T.root)

print("Neeraj Bhatade 1741")
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Admin\Documents\templ.py ==========
12 added on left of 13
28 added on right of 13
5 added on left of 12
10 added on right of 5
8 added on left of 10
69 added on right of 28
preorder
13
12
5
10
8
28
69
inorder
5
8
10
12
13
28
69
postorder
5
8
10
12
28
69
13
Neeraj Bhatade 1741
>>>
```

```
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 < len(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 i < len(lefthalf):
      arr[k]=lefthalf[i]
      i=i+1
      k=k+1

    while j < len(righthalf):
      arr[k]=righthalf[j]
```


PRACTICAL No:- 13

Aim: To sort data using Merge sort.

THEORY:

Merge sort uses the divide and conquer technique to sort the data as per the requirement and display it to the user. In merge sort, we divide the list into nearly equal sublist each of which are then again divided into two sublists.

We continue this procedure until there is only one element left in the individual sublist.

Once we have individual elements in the sublists, we consider these sub list as subproblems which can be further solved. Since there is only one element in the sublist, the sublist is already sorted. So now we manage the merge operation on sub problems.

Now while merging the sub problems, we compare the two sublist and create combined list by comparing the elements of the sublist. After comparision, we place the element in their correct position in original sublist.

```
        j=j+1
        k=k+1


arr = [13,12,5,28,11,69,14,41,20]
print("THE UNSORTED LIST : ", arr)
mergeSort(arr)
print("\nTHE MERGE SORTED LIST : ",arr)
print("Neeraj Bhatade 1741")
```

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\Admin\Documents\temp1.py ================
THE UNSORTED LIST :  [13, 12, 5, 28, 11, 69, 14, 41, 20]

THE MERGE SORTED LIST :  [5, 11, 12, 13, 14, 20, 28, 41, 69]
Neeraj Bhatade 1741
>>>
```