Practical: 01

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 2 types of search:-

1. Linear search
2. Binary search

The linear search is further classified as:

→ Sorted and unsorted

UNSORTED LINEAR SEARCH:-

Linear search 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. That is why it is called unsorted linear search.

1. The data is entered in random manner.

```
found=False
a=[1,12,23,34,45,56,67,78,89,90]
search=int(input("enter the number to be searched"))
for i in range(len(a)):
  if(search==[i]):
    print("number is found at place",i+1)
    found=True
    break
if(found==False):
  print("number doesnt exist")
print("anamika")
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Anamika/AppData/Local/Programs/Python/Python38-32/unsort no.py
enter the number to be searched 12
number doesnt exist
anamika
>>>
```

2 User needs to specify the element to be searched.

3 Check the condition whether the entered number matches, if the entered number matches, then display the location plus increment by 1 as data is stored from location zero

4. If all elements are checked on by one and element not found then prompt msg no not found.

Practical: 02

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

Theory: SEARCHING and SORTING are different modes of data structure.

SORTING: To basically sort the input data in ascending or it in the descending order.

SEARCHING: TO search and diplay the desired element.

LINEAR SORTED SEARCH:- The data is arranged in ascending to descending or descending to ascending order. That is all what is meant by searching. through sorted that is the well arangid data.

1. The work is supposed to enter the data in sorted manner.
2. User has to give an element for searching through sorted list.
3. If element is found display with an updown do value is stored from location zero.

```
found=False
a=[11,22,33,44,55,66,77,88,99]
print("anamika")
search=int(input("enter the number search"))
if(search<a[0] or search>a[len(a)-1]):
  print("number does not exist")
else:
   for i in range(len(a)):
    if(search==a[i]):
     print("number found at",i)
     found=True
     break
if(found==False):
  print("number does not exist")
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
  RESTART: C:/Users/Anamika/AppData/Local/Programs/Python/Python38-32/sort n0.py
anamika
enter the number search 33
number found at 2

  RESTART: C:/Users/Anamika/AppData/Local/Programs/Python/Python38-32/sort n0.py
anamika
enter the number search 23
number does not exist
>>>
```

4. If data or element not found print the same.

5. 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.

Practical : 03

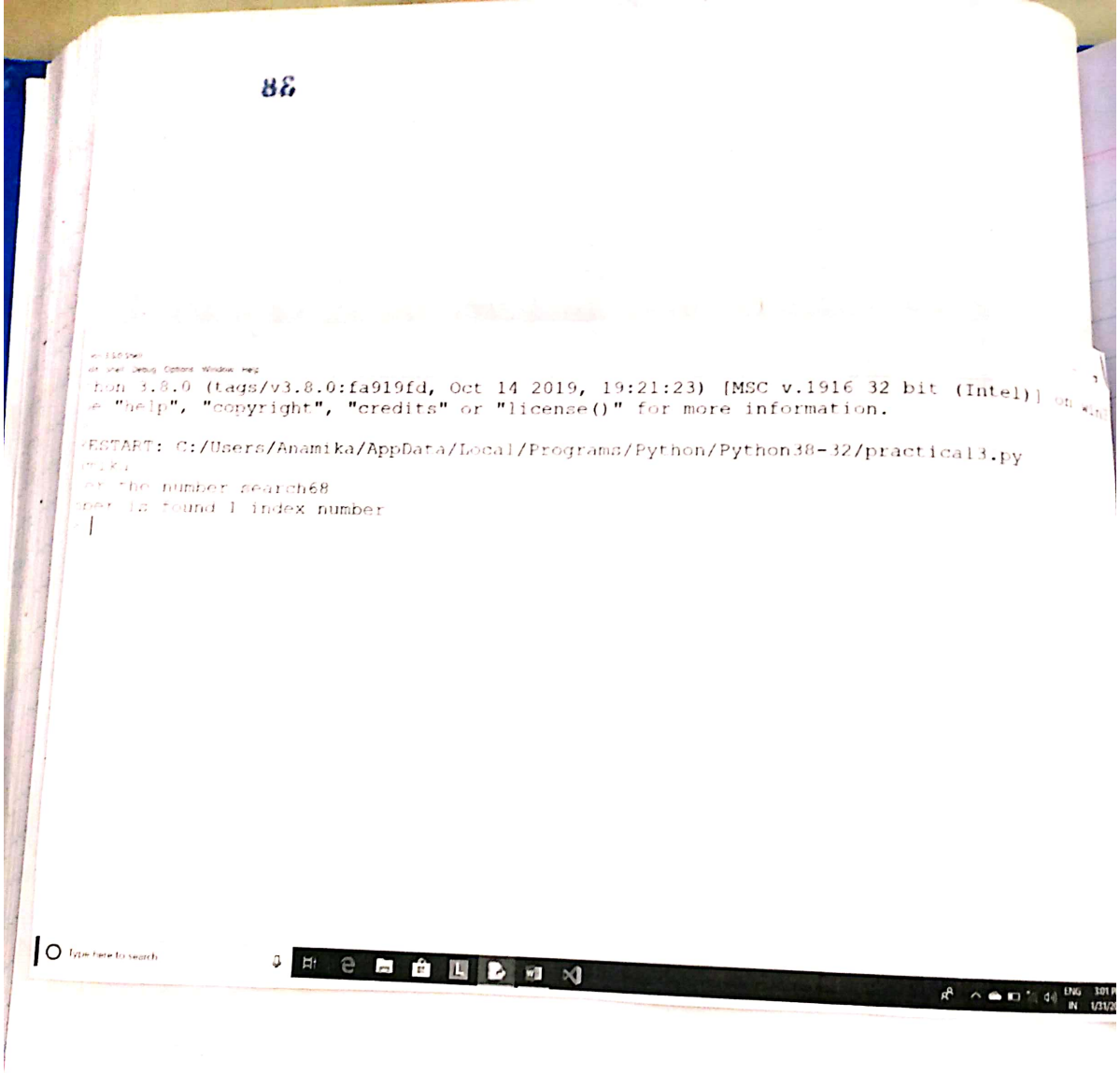
Aim : To search a number from the given sorted list using binary search.

Theory : A binary search also known as a half internal search, is an algorithm used in computer science to locate a specified value (key) within a 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 directions.

specifically, the key value is compared to the middle element of the array. If the key value is less than or greater than from this middle element, the algorithm knows which half of the array is sorted.

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

```
print("anamika")
search=int(input("enter the number search"))
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 is found",m,"index number")
break
else:
if(search<a[m]):
r=m-1
else:
r=m+1
```



Because each step in the algorithm divides the array size in half and perform binary search.

Practical : 04

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 principal operations push, which adds an element to the collection and pop, which removes the most recently added element that was not yet removed. The order maybe LIFO or FILO

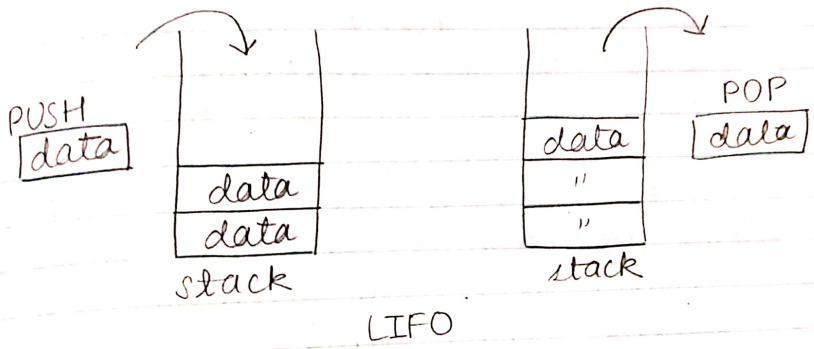
Three basic operations are performed in the stack :

1. PUSH - adds an item in the if the stack is full then it is said to be over flow condition
2. 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 is said to be an underflow condition.

```
print("anamika yadav")
class stack:
    global tos
    def __init__(self):
        self.l=[0,0,0,0,0,0,0,0]
        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()
```

3 Peek or Top : Returns top element of stack

4 is empty : Returns true if stalk is empty else false.



LIFO

Practical: 05

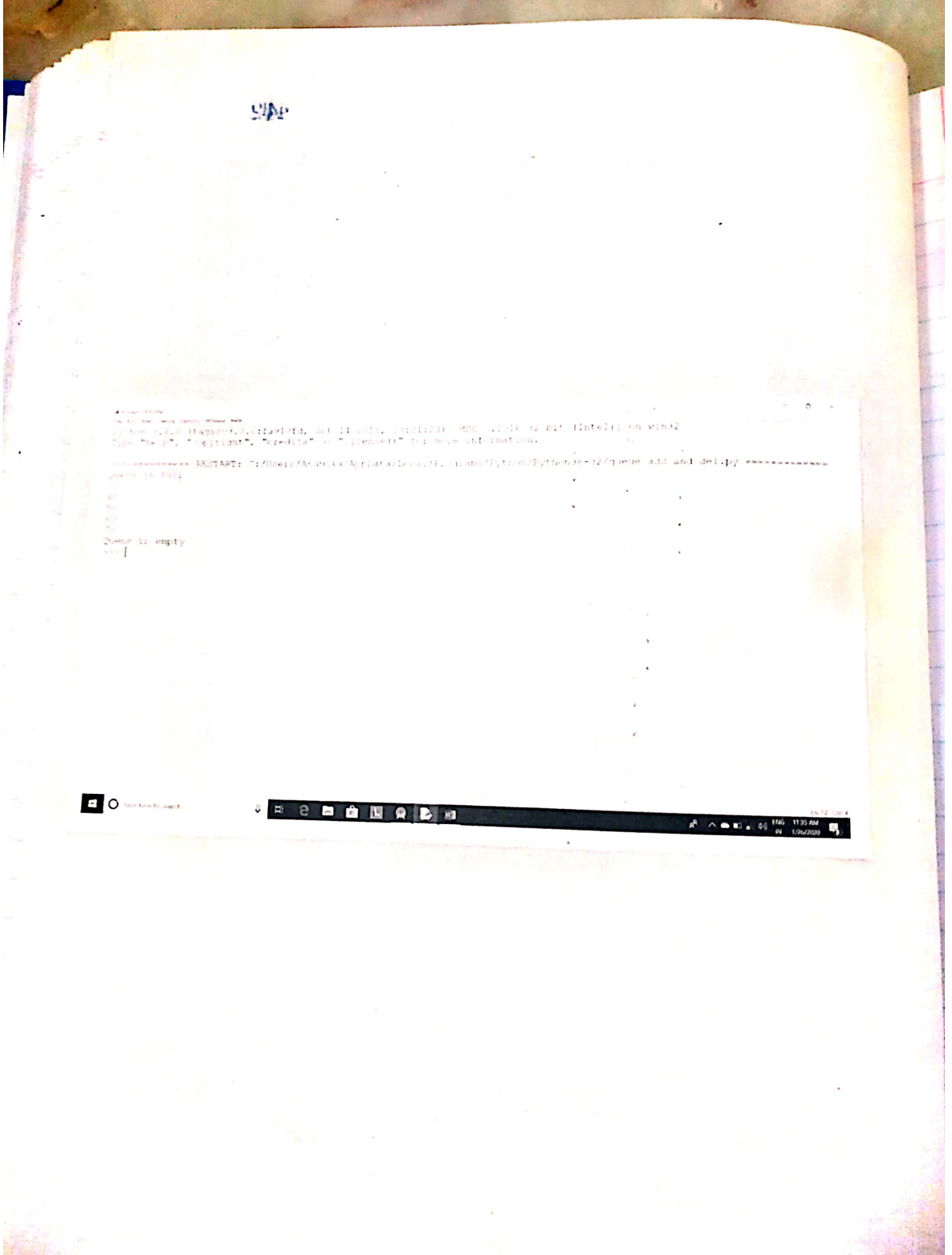
Aim: To demonstrate Queue add and delete

Theory: Queue is a linear data structure where the first element is inserted from one end called REAR and deleted from the other end called as FRONT. Front points to the beginning of the queue and Rear points to the end of the queue. Queue follows the FIFO structure. According to its FIFO structure, element inserted first will also be removed first. In a queue, one end is always used to 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 an element in queue.

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

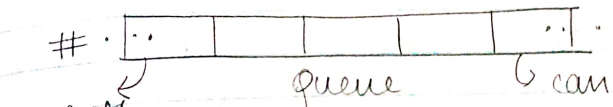
```
        global r
        global f
    def __init__(self):
        self.r=0
        self.f=0
    self.l=[0,0,0,0,0,0]
    def add(self,data):
        n=len(self.l)
        if self.r<n-1:
    self.l[self.r]=data
        self.r=self.r+1
            else:
    print("Queue is full")
    def remove(self):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

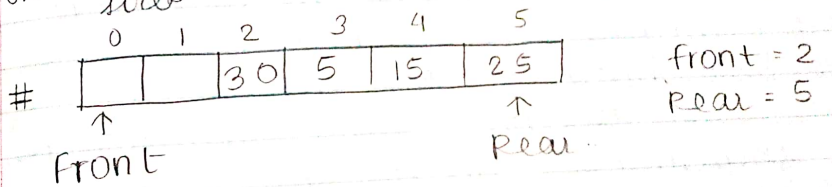


Scanned with CamScanner

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

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





Practical : 06

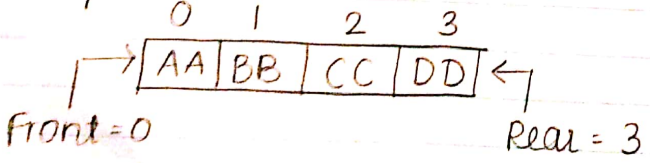
Aim : To demonstrate the use of circular queue in data structure

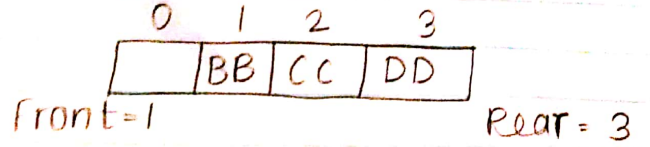
Theory: The queue that we implement using an array suffer from one limitation. In that implementation there is a possibility that the queue is reported as full, even though in actuality there might be empty slots at the beginning of the queue.

To overcome this limitation we can implement queus as circular queue.

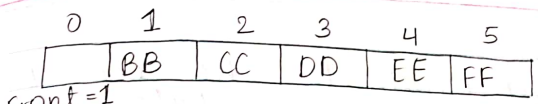
In circular queue we go on adding the element to the queue and reach the end of the array. The next element is sorted in the first slot of the array.

Example :

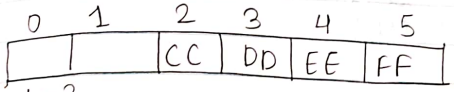




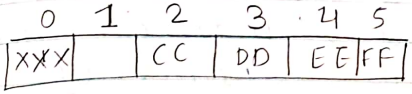
```
global data
global next
def __init__(self,item):
self.data=item
self.next=None
class linkedlist:
global s
def __init__(self):
self.s=None
def addL(self,item):
newnode=node(item)
if self.s==None:
self.s=newnode
else:
head=self.s
while head.next!=None:
head=head.next
head.next=newnode
def addB(self,item):
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)
```



Front = 1 Rear = 5



Front = 2 Rear = 5



Front = 2 Rear = 0

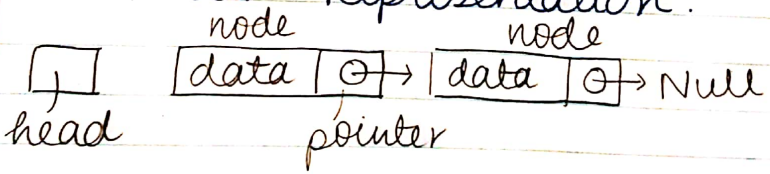
Practical: 07

Aim: To demonstrate the 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 such as. Each link contains a connection to another link.

- LINK - Each link of a linked list can store a data called an element
- NEXT - Each link of a linked list contains a link to the next link called NEXT.
- LINKED - LIST - A linked list contains the connection link to the first link called linked list

LINKED LIST Representation:



```
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.addB(40)
start.addB(30)
start.addB(20)
start.display()
print("anamika yadav")
```

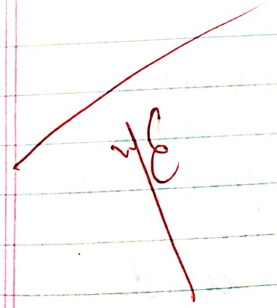
```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
- RESTART: C:/Users/Anamika/AppData/Local/Programs/Python/Python38-32/linked list.py
20
30
40
50
60
70
80
anamika yadav
>>>
```

Types of linked list:-

- single
- Doubly
- Circular

Basic Operations:

Insertion
deletion
search
display
delete



TA

Practical: 08

Aim: To evaluate postfix expression using stack.

Theory:- stack is an ADT and works on LIFO i.e PUSH and POP operations.

A postfix expression is a collection of operators and operands in which the operator is placed after the operands.

Steps to be 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 ~~operand~~ operator (+, -, *, /, etc) then perform two pop operations and store the two popped operands in two different variables (operand1 and operand2) then perform reading symbol operation using operand 2 and operand 2 and push result back onto the stack.
4. Finally perform a pop operation and display the popped value as final result

```
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="2 3 4 * +"
r=evaluate(s)
print("the evaluated value is:",r)

print("anamika yadav")
```

```
Debug Options Window Help
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win
Type "help", "copyright", "credits" or "license()" for more information.

>>> ================ RESTART: C:/Users/Anamika/AppData/Local/Programs/Python/Python38-32/#postfix evaluation.py
 evaluated value is: 14
anamika yadav
>>>
```

Types of linked list:

- simple
- Doubly
- circular.

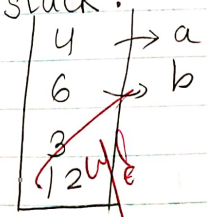
Basic Operations

Insertion
Deletion
Display
search
delete

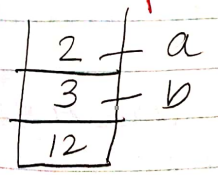
value of postfix expression:

S = 12 3 6 4 ⊖ + *

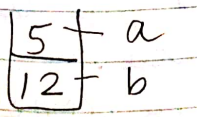
stack:



b-a = 6-4 = 2 # store again in stack



b+a = 3+2 = 5 # store result in stack



b*a = 12*5 = 60

Practical - 09

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

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

1. Always pick 1st element as pivot.
2. Always pick last element as pivot
3. Pick a random element as pivot
4. Pick median as pivot

The key process in quick sort is partition(). Target of partitions is given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

```
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=[98,87,76,65,54,43,32,21,11]
quicksort(alist)
print(alist)
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)]
Type "help", "copyright", "credits" or "license()" for more information.
>>> RESTART: C:/Users/Anamika/AppData/Local/Programs/Python/Python38-32/quicksort.py
```

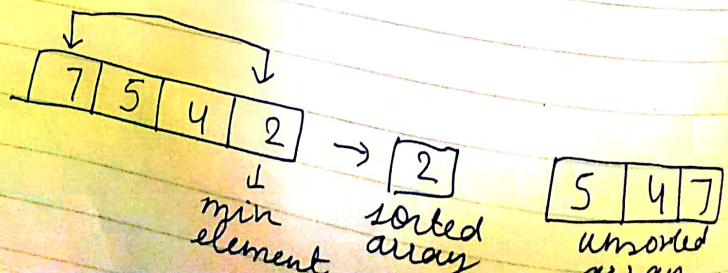
Practical - 10

Aim: To sort a given number from the list using selection sort

Selection sort: selection sort is a simple sorting algorithm. This sorting algorithm is an in-plane comparison based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially the sorted part is empty and the unsorted part is the entire list. The smallest element is selected from the unsorted array and swapped with the left most element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

Example:-

step 1



```
A=[9,7,5,3,4,6,8]
print(A)
for i in range (len(A)-1):
    for j in range (len(A)-1-i):
        if (A[j]>A[j+1]):
            t=A[j]
            A[j]=A[j+1]
            A[j+1]=t
print(A)
print("anamika")
```

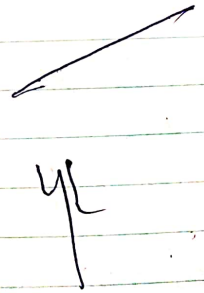
```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bi
t (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Anamika/AppData/Local/Programs/Python/Python38-32/anami
kal.py
[9, 7, 5, 3, 4, 6, 8]
[3, 4, 5, 6, 7, 8, 9]
anamika
>>>
```

step 2 : [2] [5|4|7]

[2|4] [5|7]

step 3 : [2|4] [5|7] [2|4|5] [7]

step 4 : [2|4|5] [7] → [2|4|5|7]



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 referred to as sinking sort. Is a simple sorting algorithm that repeatedly steps through the lists. 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 way smaller or larger elements "bubble" to the top of the list. Although the algorithm is simple, it is too slow as if compares one element checks if condition fails then only swaps otherwise goes on

```
A=[1,3,5,7,9,8,6,4,2]
print(A)
for i in range (len(A)-1):
    for j in range (len(A)-1):
        if (A[i]>A[j+1]):
            t=A[i]
            A[i]=A[j+1]
            A[j+1]=t
print(A)
print("ANAMIKA")
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bi
t (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Anamika/AppData/Local/Programs/Python/Python38-32/anami
ka22.py
[-, 3, 5, 7, 9, 8, 6, 4, 2]
[-, 9, 8, 7, 6, 5, 4, 2, 3]
ANAMIKA
>>>
```

example :
First pass

(51428) → (15428) here algorithm compare the first two elements and swap
∴ 5 > 1
(15428) → (14528) swap since 5 > 4
(14528) → (14258) swap since 5 > 2
(14258) → (14258) swap now since these elements are already in order (8 > 5) algorithm does not swap them.

Second pass
(14258) → (14258)
(14258) → (12458) swap sin 4 > 2
(12458) → (12458)

Third pass
(12458) it checks and gives the data in sorted order.

Practical-12

Aim: To sort given random data using merge sort

Theory: Mergesort is a divide and conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. It is an efficient, general-purpose, comparison based sorting algorithm. Most implementations produce a stable sort, which means that the order of equal elements is the same in input and output. It has a worst case time complexity being $O(n \log n)$ is one of the most respected algorithm.

→ 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

```
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
        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)
print("ANAMIKA YADAV")
arr=[99,98,87,76,65,54,43,32,21,11]
print(arr)
n=len(arr)
mergesort(arr,0,n-1)
print(arr)
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/Anamika/AppData/Local/Programs/Python/Python38-32/last practical.py ==========
ANAMIKA YADAV
[55, 45, 57, 76, 65, 54, 63, 32, 21, 11]
[11, 65, 98, 99, 87, 76, 43, 54, 32, 22]
>>>
```

merges the two sorted sub-array into one.

Practical - 13

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 nodes.

→ A binary tree is an important class of a tree data structure in which a node can have at most two children.

~~diagramatic representation of Binary search 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:
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)
print("ANAMIKA YADAV")
t=Tree()
t.add(100)
t.add(90)
t.add(80)
t.add(70)
t.add(30)
t.add(40)
t.add(50)
t.add(60)
t.add(10)
t.add(20)
print("preorder")
t.preorder(t.root)
print("inorder")
t.inorder(t.root)
print("postorder")
t.postorder(t.root)
```

Traversal is a process and visits all the nodes of a tree and may print their values too.

There are 3 ways which we use to traverse a tree

- Inorder
- Preorder
- Postorder

Inorder - The left subtree is visited first then the root and later the right sub tree we should always remember that every node may represent a subtree itself. Output produced is sorted by values in Ascending order.

Pre-order - The root node is visited first 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.