## * Practical No. 1

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

Theory :- the process of identifying or a finding a particular ~~record~~ record is called searching. there are two type of search i) linear ii) binary

The linear search futher classified into 2 type i sorted ii unsorted

Here we look on unsorted linear search is also know as sequential search, is a process that check every element in list sequential until the desired no. is ~~not~~ founded when element to be search is not specifically arranged in ascending or descending order. they are ~~not~~ arranged in random manner that is what it call unsorted linnear search

* Unsorted linear search
  - the data is entered in random manner
  - user need to specify the element to be search in entered list
  - check the condition that weather the entered no. is matches or if it matches then display the location
  - if all element are checked one by one and element not found then print message no. not found



Unsorted linear search

```
Print("Aditya 1769")
found=False
a=[21,24,26,6,88,45,58,69,99
search=int(input("enter the number search"))
for i in range(len(a)):
if(search==a[i]):
print("number found at",i)
found=True
break
if(found==False):
print("number dose not exist")
```

Python 3.4.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Aditya 1769
enter the number search45
number found at 5
>>> ========== RESTART ==========
>>>
Aditya 1769
enter the number search22
number dose not exist
>>>
```

BE.

: Linear sorted

```
print("Aditya 1769")
found=False
a=[21,23,47,57,69,77,82,99]
search=int(input("enter the number search"))
if(search<a[0] or search>a[len(a)-1]):
    print("number dose 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 dose not exist")
```

Output:

```
Aditya 1769
enter the number search69
number found at 4
>>> ========== RESTART ==========
>>>
Aditya 1769
enter the number search22
number dose not exist
```



Practical No. 2

Aim:- to search Number from a list using linear sorted method

Theory:- Searching & sorting are different type of data structure

Sorting:- To basically sort the input in ascending or descending manner

searching:- to search element & display the same

Linear Sorted search the data is arranged in ascending to descending or descending to ascending that is all it meant by searching through sorted that is well arranged the data

* Sorted linear search
- the user is suppose to Enter data in sorted manner
- user has to give a element for searching through sorted list
- If element is founded display with updation as value stored from location 0
- If data or element not found print the same
- In sorted ordered list of element we can check the condition that weather the entered number lies from starting point to last element if not then we can say no is not in list.

# Practical No. 3

- Aim:- To search a no. from the given sorted list using binary search
- Theory :- A binary search is also know as half interval search is an algorithm used in computer science to locate a specific value (key) with an array for the search to be binary the array should be sorted in ascending or decending
- At each step of algorithm a comparision is made and procedure branches into one or two direction
- Specifically the key value is compared to middle element of array
- If key value is less than or greater than the middle element, the algorithm know which half of array to continue the search is because array is sorted
- this process is repeated on progressi smaller element of a array until value is located. because each step in algorithm divide the array size in binary search will complete successfully in algorithm time

```
print("Aditya Pathak 1769")
a=[1,12,19,21,24,47,69]
search=int(input("Enter a number to be searched: "))
l=0
r=len(a)-1
while (True):
  m=int(l+r)//2
  if(l>r):
    print("Number not found!")
    break
  if (search==a[m]):
    print("Number found at: ",m+1)
    break
  elif (search<a[m]):
    r=m-1
  else:
    l=m+1
```

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\Binary_Search.py ========
Aditya Pathak 1769
Enter a number to be searched: 21
Number found at:  4
>>>
======== RESTART: C:\Users\Admin\Documents\Practice\Binary_Search.py ========
Aditya Pathak 1769
Enter a number to be searched: 22
Number not found!
>>>
```

```
print("Adityapathak 1769")
class stack:
globaltos
def __init__(self):
self.l=[0,0,0,0,0,0,0]
self.tos=-1
def push(self,data):
    n=len(self.l)
ifself.tos==n-l:
print("stack is full")
else:
self.tos=self.tos+1
self.l[self.tos]=data
def pop(self):
ifself.tos<0:
print("stak is empty")
else:
    k=self.l[self.tos]
print("data=",k)
self.tos=self.tos-1
s=stack()
s.push(11)
s.push(22)
s.push(33)
s.push(44)
s.push(55)
s.push(66)
s.push(77)
s.push(88)
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
```



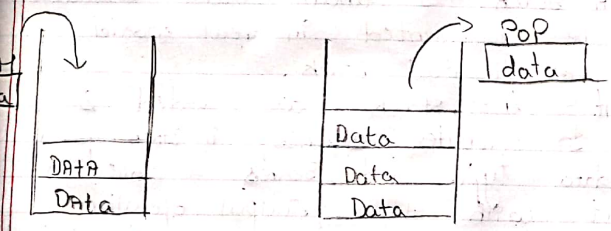
# Practical No. 4

Aim: To demonstrate use of Stack

Theory: A stack is linear data structure that can be represented in real world in form of physical stack.

- The element in stack are added & removed. In computer science, A stack is a abstract data type that serves a collection of element with two principal operation Push, which adds the 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 stack
- PUSH: Adds an item in the if the stack is full then is said to be overflow condition
- POP: Remove an item from stack. The item are popped in reversed order in which they are pushed. If the stacked is empty then it said to be underflow condition

- Peek or top : Return top element of stack
- is empty : Return true if stack is empty else false






```
File  Edit  Shell  Debug  Options
Python 3.8.0 (tags/v3.8.0
Type "help", "copyright",
>>>
==========
Aditya pathak 1769
stack is full
data= 77
data= 66
data= 55
data= 44
data= 33
data= 22
data= 11
stak is empty
>>>
```

```
print("Adityapathak 1769")
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)
ifself.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)
ifself.f<n-1:
print(self.l[self.f])
self.f=self.f+1
else:
print("queue is full")
q=queue()
q.add(33)
q.add(44)
q.add(55)
q.add(66)
q.add(77)
q.add(88)
q.remove()
q.remove()
q.remove()
q.remove()
q.remove()
q.remove()
```

Output :

```
Python 3.8.0 (tags/v3.
tel)] on win32
Type "help", "copyrigh
>>>
====================
Aditya pathak 1769
queue is full
33
44
55
66
77
queue is full
>>>
```



Practical No: 5

Aim: To demonstrate queue add and delete

Theory: Queue is linear data structure where first element is inserted from one end to is another called REAR and deleted from other end is called FRONT

Front point to beginning of queue & REAR is Point to End of queue.

Queue follow the fifo (first in first out) structure

According to fifo structure element inserted first will also removed first

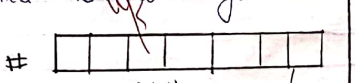
In a queue, one end is always used to insert data (Enqueue) and other use to delete data (dequeue because Enqueue is open at both of its End

Enqueue(): can be termed as add() in queue i.e adding element in queue

Dequeue(): can be termed as delete or remove i.e deleting or remove of element

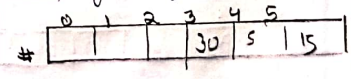
Front is used to get the fronet data item from queue

Rear is used to get last item from queue




open on both side

can have open end




Front = 2
Rear = 5

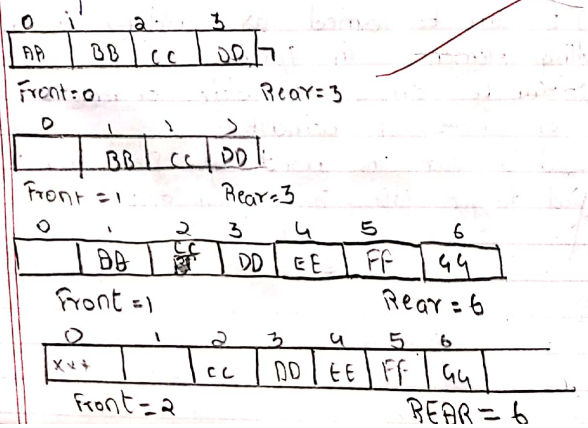
43

## Practical No: 6

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 there is possitively that the queue is reported as full, even though in actuality there might be empty slot at the beginning of queue. To overcome this limitation we can implement queue as Circular queue. In Circular queue we go on adding the Element to the queue & reach End of array. The next element is stored in first slot of array

Example




```
print("Adityapathak 1769")
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)
   ifself.r<n-1:
    self.l[self.r]=data
    print("data added",data)
    self.r=self.r+1
    else:
          s=self.r
     self.r=0
     ifself.r<self.f:
      self.l[self.r]=data
      self.r=self.r+1
      else:
       print("queue is full")
       def remove(self):
           n=len(self.l)
        ifself.f<n-1:
         print("data removed:",self.l[self.f])
         self.f=self.f+1
         else:
              s=self.f
          self.f=0
          ifself.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(30)
            q.add(40)
            q.add(50)
            q.add(60)
            q.add(70)
            q.add(80)
            q.remove()
            q.add(90)
```



output:

```
>>>
========================
Aditya pathak 1769
data added 30
data added 40
data added 50
data added 60
data added 70
queue is full
data removed: 30
data added 90
>>>
```

## Source Code:

```
print("Aditya pathak 1769")
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="8 6 9 * +"
r=evaluate(s)
print("the evaluate value is :",r)
```



## Practical No. 7:

- Aim: To Evalulate Postfix expression using stack
- Theory: Stack is an ADT (adt) work on LIFO (Last in first out) i.e Push & Pop operation.
- A postfix expression is collection of operator and operand in which the operator is place after the operand.
- Step to be followed

i) read all the symbol one by one from left to right given by postfix operation

ii) If reading symbol operation i.e operator (+, *, -, /) then performed two pop operation & store two popped operand in a different variable

iii) The performed reading symbol operation using operand 1 & 2 & push result back into stack

iv) finally perform pop operation & display popped value as final result

S = 8 6 9 * +

Stack

| 8 |
|---|
| 6 ← b |
| 9 ← a |

6*9 = 54 Store again in Stack

| 8 ← b |
|---|
| 54 ← a |

54+8 = 62 //

Store result in Stack

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (In-
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Aditya pathak 1769
the evaluate value is : 62
>>>
```

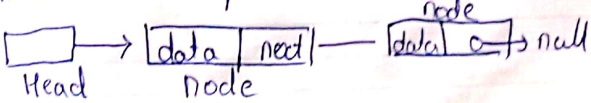
```
print("Aditya pathak 1769")
class node:
    global data
    global next
    def __init__(self,item):
        self.data=item
        self.next=None
class linkedlist:
    global s
    def __init__(self):
        self.s=None
    def addl(self,item):
        newnode=node(item)
        if self.s==None:
            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()
```



# Pratical No. 8

Aim: To demonstrate the use of Linked List in data Structure

- Theory: A Linked List is sequence of data structure Linked List is sequence of link which Contain item each link contain a Connection to another link
- Link - Each link of a Linked List can store a data called an Element
- NEXT - Each link of linked list contain Link to next Link called as NEXT
- Linked List - A linked list contain the Connection link first link is called first
- LINKED list Representation

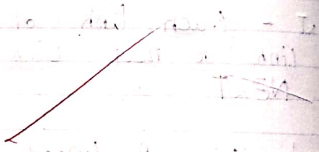


there are three type of linked list

i) single ii) Double iii) Circular

Basic operation

i) Insertion
ii) Deletion
iii) display
iv) Search
v)





Python 3.4.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.3 (v3.4.3:9b73f1c3e
tel)] on win32
Type "copyright", "credits" or
>>> ==========================
>>>
Aditya pathak 1769
20
30
40
50
60
70
80
>>>
```

## Code

```
print("Aditya pathak 1769")
A=[12,5,16,1,23,9,30,4,45]
print(A)
for i in range(len(A)-1):
  for j in range(len(A)-1-i):
    if A[j] > A[j+1]:
      A[j],A[j+1]=A[j+1],A[j]
print("the array in bubble sort",A)
```

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Aditya pathak 1769
[12, 5, 16, 1, 23, 9, 30, 4, 45]
the array in bubble sort [1, 4, 5, 9, 12, 16, 23, 30, 45]
>>>
```



## Practical No. 9

Aim: To sort given random data by using bubble sort

Theory: Sorting is type in which any random data by using bubble sorted i.e ascending or descending order
Bubble sort sometime referred to as sinking sort. Is a simple sorting algorithm reapatly step through the list, compare adjeant Element & swap them if they are corong The pass through list is sorted.
the algorithm which is comparison sort is named for the way smaller or larger Element "bubble" on top of list.
Although the algorithm is simple, it is too slow as it compare one element check if condition fail then only swap otherwise goes on.

First Pass

A = [12, 5, 16, 1, 23]

Now first two element will compare & swap since 12 > 5

(5, 12, 16, 1, 23) → (5, 12, 16, 1, 23) No swap since 16 > 12

(5, 12, 16, 1, 23) → (5, 1, 16, 12, 23) Swap 12 > 1

Second pass

(5, 1, 16, 12, 23) → (1, 5, 16, 12, 23) Swap 5 > 1

Program

```
print("Aditya pathak 1769")
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:
         alist[leftmark],alist[rightmark]=alist[rightmark],alist[leftmark]
      alist[first],alist[rightmark]=alist[rightmark],alist[first]
      return rightmark
```



# Practical No. 10

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

Theory: QuickSort is an efficient Sorting algorithm type of an divide & Conquer algorithm to it pick an element as pivot and partition. the given array around the picked pivot. there are many different version of quick sort that picked pivot in different ways

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

The key process in quick sort is partition. Target of partition is give an array and an element x' of array Put x at its current position as Sorted array and Put smaller array element, before x & Put all greater element (greater than x) after x. All this should be done in linear time.

```
alist=[12,5,16,1,23,9,30,4,45,41,44,47]

print(alist)

quicksort(alist)

print("quicksort",alist)
```

Output

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Aditya pathak 1769
[12, 5, 16, 1, 23, 9, 30, 4, 45, 41, 44, 47]
quicksort [4, 5, 1, 9, 23, 30, 45, 44, 41, 47, 16, 12]
>>>
```

WE

Source code:

```
print("Aditya pathak 1769")
A=[12,5,16,1,23,9,30,4,45]
print(A)
for i in range(len(A)-1):
  for j in range(len(A)-1):
    if A[j] > A[j+1]:
      A[j],A[j+1]=A[j+1],A[j]
print("the array in selection sort",A)
```

Output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================ RESTART ================
>>>
Aditya pathak 1769
[12, 5, 16, 1, 23, 9, 30, 4, 45]
the array in selection sort [1, 4, 5, 9, 12, 16, 23, 30, 45]
>>> ================ RESTART ================
>>>
```



# Practical No. 11

Aim: to demonstrate the Selection sort method

Theory:
The sorting is based on selection hence is know as selection sort. It is also know as placing sorting algorithm
Select the first largest element & place it at end of array. Select the last second largest element & so on

Eg [12, 5, 16, 1, 23, 9, 30, 4, 45]

[5, 12, 16, 1, 23, 9, 30, 4, 45]

find largest & replace with last

[5, 12, 16, 1, 23, 9, 4, 30, 45] & do it same

[5, 12, 16, 1, 9, 4, 23, 30, 45]

[5, 12, 1, 9, 4, 16, 23, 30, 45]

[5, 1, 9, 4, 12, 16, 23, 30, 45]

[5, 1, 4, 9, 12, 14, 16, 23, 30, 45]

[1, 4, 5, 9, 12, 14, 16, 23, 30, 45]

# Practical No. 12

**Aim:** To demonstrate the binary Tree & Traversing it into postorder, Preorder, Inorder

**Theory:**

Binary tree is tree which support maximum of two children for any node within the tree thus any particular node can have either 0, 1 or 2 node this is another identity of binary tree that it is order such one child is identified as left child and another right child

- Leaf node: Nodes which do not have any children
- Internal node: Node which don't leaf node
- Traversing can be defined as process of visiting every node of tree exactly once

* Preorder: i) visit the root node
ii) Traversing the left subtree, the left subtree in turn might have left & right subtree
iii) Traversing the right subtree. the right subtree inturn might have left & right subtree

```
print("Adityapathak 1769")
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:
        ifnewnode.data<h.data:
         ifh.l!=None:
```



```
                h=h.l
          else:
           h.l=newnode
           print(newnode.data,"add on left data",h.data)
           break
          else:
           ifh.r!=None:
                h=h.r
           else:
            h.r=newnode
            print(newnode.data,"add on right data",h.data)
            break
            def preorder(self,start):
             if start!=None:
             print(start.data)
             self.preorder(start.l)
             self.preorder(start.r)
             definorder(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()
t.add(99)
t.add(82)
t.add(87)
t.add(50)
t.add(66)
t.add(41)
t.add(47)
t.add(12)
t.add(30)
print("PREORDER")
t.preorder(t.root)
print("INORDER")
t.inorder(t.root)
print("POSTORDER")
t.postorder(t. root)
```

```
>>>
Aditya patnik 17BD
99 add on left data 82
82 add on right data 87
50 add on left data 82
66 add on right data 50
41 add on left data 50
47 add on right data 41
12 add on left data 41
30 add on right data 12
PREORDER
99
82
50
41
12
30
47
66
87
INORDER
12
30
41
47
50
66
82
87
99
POSTORDER
30
12
47
41
66
50
87
82
99
>>>
```



- Inorder : i) Traverse the left subtree. the right subtree may turn into left & right subtree
  ii) Visit the root node
  iii) Traverse the right subtree. the right subtree may or might turn into left or right subtree
- Post order : i) Traverse the left subtree. The left subtree inturn might have left or right subtree
  ii) Traverse the right subtree. the right subtree in turn might have left or right subtree
  iii) Visit root node

# Practical NO. 13

Aim: Merge sort

Theory: Merge sort use the divide and conquer the technique. In merge sort we divide the list into equal sublist of which are again divide into 2 sublist

- we continue this procedure until there is only one element in individual sublist
- One, we have individual element in sublist, we consider this sublist as sub problem which can be solved
- Since there is only element in sublist it already sorted so now merge sub problem
- Now while merging the subproblem, we compare the 2 sublist & create a combine list by comparing element of sublist. After comparision, we place these element in their correct position in org sublist



```
print("Aditya  pathak 1769")
def mergeSort(a):
  if len(a)>1:
    mid=len(a)//2
    lefthalf=a[:mid]
    righthalf=a[mid:]
    mergeSort(lefthalf)
    mergeSort(righthalf)
    i=j=k=0
    while i<len(lefthalf) and j<len(righthalf):
      if lefthalf[i] < righthalf[j]:
        a[k]=lefthalf[i]
        i=i+1
      else:
        a[k]=righthalf[j]
        j=j+1
      k=k+1
    while i < len(lefthalf):
        a[k]=lefthalf[i]
        i=i+1
        k=k+1
    while j < len(righthalf):
        a[k]=righthalf[j]
        j=j+1
        k=k+1
a=[12,5,16,1,23,9,30,4,45]
print("the unsorted list :",a)
mergeSort(a)
print("the merge sorted list:",a)
```

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Aditya pathak 1769
the unsorted list : [12, 5, 16, 1, 23, 9, 30, 4, 45]
the merge sorted list: [1, 4, 5, 9, 12, 16, 23, 30, 45]
>>>
```