Practical No.1

Source Code:

unsorted.py - C:\Python34\unsorted.py (3.4.3)

File Edit Format Run Options Window Help

```
print("Deepak \n1724")
a=[1,45,7,24,27,18,25,93,21]
print(a)
search=int(input("Enter a number to be searched:"))
for i in range(len(a)):
    if(search==a[i]):
        print("Number found at location:",i)
        break
else:
    print("Enter a valid number to be searched")
```

Output:

Python 3.4.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Deepak
1724
[1, 45, 7, 24, 27, 18, 25, 93, 21]
Enter a number to be searched:24
Number found at location: 3
>>> ========== RESTART ==========
>>>
Deepak
1724
[1, 45, 7, 24, 27, 18, 25, 93, 21]
Enter a number to be searched:99
Enter a valid number to be searched
>>>
```

PRACTICAL - 01

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

Theory:-

Linear search is used to search an elements from an array.

Linear search is of two types:-

a) Unsorted linear search

b) Sorted linear search.

They are also called as unordered and ordered array.

- Unsorted linear search:-

The elements are arranged in random manner. It searches each element in the array until the user gets the desired element.

The data is entered in random manner in the array which is assigned to the variable. The user enters the desired element he/she wants. The compiler then ~~the~~ check each element of the array that matches the user's desired element. If the number is matched, the compiler prints the position of that element in the array. If the number is not found, the compiler then prints an appropriate statement to enter a appropriate element.

PRACTICAL - 02

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

Theory:-

In this type of searching method, the elements of the list are arranged in a proper manner i.e either they are in ascending order or descending order.

The array is filled with number arranged in ascending or descending order. This helps user to choose a number between the range. As the user inputs a desired value to be searched, the compiler check each element of the array / list. If the desired number is present in the list, the compiler prints the location of that desired number. Else a appropriate message is sent to user to input a valid number.

Practical No.2

Source Code:

sorted.py - C:\Python34\sorted.py (3.4.3)

File Edit Format Run Options Window Help

```
print("Deepak \n1724")
a=[1,7,18,21,24,25,27,45,93]
print(a)
k=0
search=int(input("Enter a number to be searched from the list:"))
if((search>=a[0]) or (search<a[len(a)-1])):
    for i in range(len(a)):
        if(search==a[i]):
            print("Number found at location:",i)
            k=1
            break
if(k==0):
    print("Number does not exists")
```

Output:

Python 3.4.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Deepak
1724
[1, 7, 18, 21, 24, 25, 27, 45, 93]
Enter a number to be searched from the list:24
Number found at location: 4
>>> ========== RESTART ==========
>>>
Deepak
1724
[1, 7, 18, 21, 24, 25, 27, 45, 93]
Enter a number to be searched from the list:23
Number does not exists
>>>
```

```
**************************SOURCE CODE**********************************
print("Name: Deepak Maurya")
print("Roll No.: 1724")
a=[1,11,21,24,25,26,27,29,45]
print(a)
search=int(input("Enter the number to be search : "))
l=0
h=len(a)-1
m=(l+h)//2
if (search<a[l]) or (search>a[h]):
    print("Number does not exist!")
elif search==a[l]:
    print("Number found at location ",l)
elif search==a[h]:
    print("Number found at location ",h)
else:
    while (l!=h):
        if search==a[m]:
            print("Number found at location ",m)
            break
        else:
            if(search<a[m]):
                h=m
                m=(l+h)//2
            else:
                l=m
                m=(l+h)//2
```

```
**********************************OUTPUT*****************************
*******
>>> ========== RESTART ==========
>>>
Name: Deepak Maurya
Roll No.: 1724
[1, 11, 21, 24, 25, 26, 27, 29, 45]
Enter the number to be search : 24
Number found at location  3
>>> ========== RESTART ==========
>>>
Name: Deepak Maurya
Roll No.: 1724
[1, 11, 21, 24, 25, 26, 27, 29, 45]
Enter the number to be search : 99
Number does not exist!
```

PRACTICAL NO. 3

Aim :- To search a number using binary search.

Theory :-

A binary search also known as a half-internal search, is an algorithm used in computer science to allocate a specified value within an array. For the search to be linear, the array must be sorted in either ascending or descending order.

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

Specially 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 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.

4/12/19 169

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 referred is as sinking sort

It is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in wrong order.

The pass through the list is repeated until the list is sorted. The algorithm which is a comparision 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 is compare one elements checks if condition ~~false~~ fails then only swaps otherwise goes on.

```
**************************SOURCE
CODE************************************

print("Name: Deepak Maurya")
print("Roll No.:1724")

a=[29,45,21,27,25,24]
print("Elements of the list before sorting is:",a)

for i in range(len(a)-1):
     for j in range(len(a)-1-i):
          if (a[j]>a[j+1]):
               temp=a[j]
               a[j]=a[j+1]
               a[j+1]=temp
print("Elements of the list after sorting:",a)


**********************************OUTPUT*****************************
*******

>>> ================================ RESTART
================================
>>>
Name: Deepak Maurya
Roll No.:1724
Elements of the list before sorting is: [29, 45, 21, 27, 25, 24]
Elements of the list after sorting: [21, 24, 25, 27, 29, 45]
>>>
```

**********INPUT**********

```
print("Name: Deepak Maurya")
print("Roll No.:1724")
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("Satck is 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()
```

**********OUTPUT**********

```
Name: Deepak Maurya
Roll No.:1724
Stack is full
data= 70
data= 60
data= 50
data= 40
data= 30
data= 20
```

PRACTICAL-05

033

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 elements to the collection and pop, which removes the most recently added element that was not yet removed.

• PUSH :- Adds an item in the stack if the stack is full then it is said to be over flow condition.

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

PRACTICAL - 06

Aim :- To demonstrate queue add and delete.

Theory:- Queue is a linear data structure where the first element is inserted from one end called REAR and delete 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 which, elements inserted first will be removed first.

In a queue, one end is always used to insert data and the other end is used to delete data because queue is open at both of its ends.

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

```
*****************INPUT****************
print("Name: Deepak Maurya")
print("Roll No.:1724")
class queue:
    global r
    global f
    def __init__(self):
        self.l=[0,0,0,0,0,0]
        self.r=0
        self.f=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])
        else:
            print("Queue is empty")
s=queue()
s.add(30)
s.add(40)
s.add(50)
s.add(50)
s.add(60)
s.add(70)
s.add(80)
s.remove()
s.remove()
s.remove()
s.remove()
s.remove()
s.remove()
```

```
*****************OUTPUT****************
Name: Deepak Maurya
Roll No.:1724
Queue is full
Queue is full
30
30
30
30
30
30
```

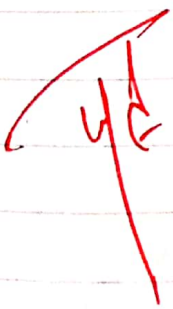
```
**********INPUT**********
print("Name: Deepak Maurya")
print("Roll No.:1724")
class queue:
    global r
    global f
    def __init__(self):
        self.l=[0,0,0,0,0,0]
        self.r=0
        self.f=0
    def add(self,data):
        n=len(self.l)
        if self.r<=n-1:
            self.l[self.r]=data
            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
s=queue()
s.add(44)
s.add(55)
s.add(66)
s.add(77)
s.add(88)
s.add(99)
s.remove()
s.add(66)
```

```
**********OUTPUT*********
Name: Deepak Maurya
Roll No.:1724
Data Added: 44
Data Added: 55
Data Added: 66
Data Added: 77
Data Added: 88
Data Added: 99
Data removed: 44
```

PRACTICAL- 07 035

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 posibility that the queue is reported as full, even ~~things~~ though in actually there might be empty state 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 stored in the first slot of the array.



PRACTICAL-08

Aim :- To demonstrate the use of linked list

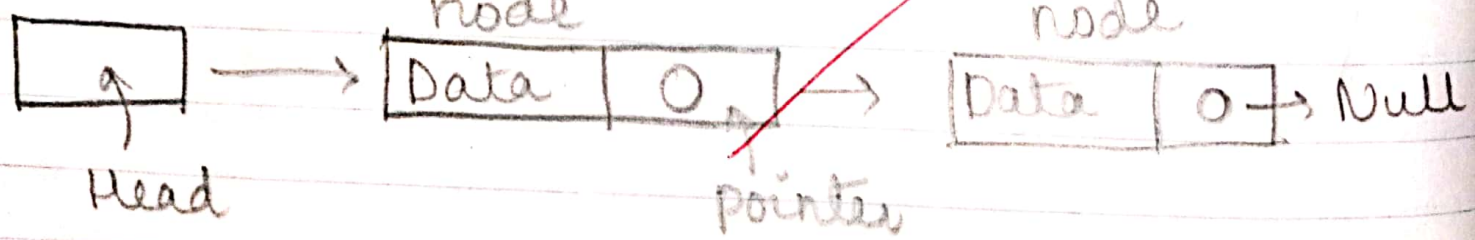
Theory :- A linked list is a sequence of data structure. Linked list is a sequence of links contains a connection to another link.

- Link :- Each link of a linked list can store a data called an element.
- NEXT :- Each link of a linked list contains a lin to the next link called NEXT.
- LINKED LIST :- A linked list contains the connectio link to the first link called first.

Types of linked list :-

i) Simple
ii) Doubly
iii) Circular

Linked list representation :-



```
print("Name: Deepak Maurya \nRoll No.:1724")
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 add1(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 add2(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.add1(24)
start.add1(25)
start.add1(27)
start.add1(21)
start.add2(45)
start.add2(18)
start.add2(7)
start.add2(93)
start.display()
```

OUTPUT

```
Name: Deepak Maurya
Roll No.:1724
93
7
18
45
24
25
27
21
```

```
print("Name: Deepak Maurya \nRoll No.:1724")
def eval(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(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='9 7 1 * +'
print("The given evaluation expression is:",s)
r=eval(s)
print("The evaluated value is:",r)
```

OUTPUT

```
Name: Deepak Maurya
Roll No.:1724
The given evaluation expression is: 9 7 1 * +
The evaluated value is: 16
```

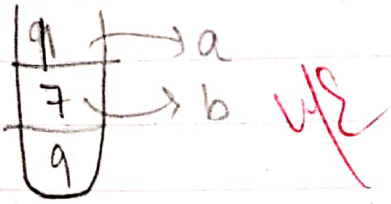
PRACTICAL-09

Aim:- To evaluate postfix expression using stack.

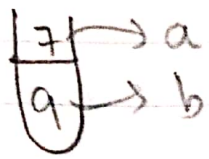
Theory:- Stack is an ADT and works on LIFO i.e PUSH and POP operation.
A postfix expression is a collection of operators and operands in which the operator is placed after the operands.

Steps to be followed:-
i) Read all the symbols one by one from left to right in the given postfix expression.
ii) If the reading symbol is operand then push it on to the stack.
iii) If the reading symbol is operator then perform (first ^ two) pop operation and store the two popped operands in two different variables. Then perform reading symbol operation using operand 1 and operand 2 and push result back on the stack.
iv) Finally, Perform a pop operation and display the popped value as final result.

Eg:- S = 9 7 1 * +



b * a = 7 * 1 = 7



b + a = 7 + 9 = 16

569

PRACTICAL-10

Aim:- To evaluate quick sort Method.

Theory:- Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value say pivot value, based on which the partition is made and another array holds values greater than the pivot value.

Quicksort partitions an array and then calls itself recursively twice to sort the two resulting subarrays. This algorithm is quite efficient for large-sized data sets as its average and worst-case complexity are O and image respectively.

Quicksort is an divide and conquer algorithm. It is an efficient sorting algorithm Developed by British computer scientist Tony Hoare in 1959 and published in 1961.

Input:-

```
print("Name: Deepak Maurya\nRoll No.: 1724")
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=[1,9,45,24,25,27,26,21,93]
print("The number before sorting is:",alist)
quicksort(alist)
print("The number after sorting is:",alist)
```

Output:-

```
Name: Deepak Maurya
Roll No.: 1724
The number before sorting is: [1, 9, 45, 24, 25, 27, 26, 21, 93]
The number after sorting is: [1, 9, 21, 24, 25, 26, 27, 45, 93]
```

INPUT

```
print("Name: Deepak Maurya \nRoll No.: 1724")
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:
            nn=Node(val)
            h=self.root
            while True:
                if nn.data<h.data:
                    if h.l!=None:
                        h=h.l
                    else:
                        h.l=nn
                        print(nn.data,"addedon left of",h.data)
                        break
                else:
                    if h.r!=None:
                        h=h.r
                    else:
                        h.r=nn
                        print(nn.data,"data on right of",h.data)
                        break
    def preorder(self,start):
        if start!=None:
            print(start.data)
            self.preorder(start.l)
            self.preorder(start.r)
    def inorder(self,start):
        if start!=None:
            self.inorder(start.l)
            print(start.data)
            self.inorder(start.r)
    def postorder(self,start):
        if start!=None:
            self.postorder(start.l)
            self.postorder(start.r)
            print(start.data)
t=Tree()
t.add(100)
t.add(80)
t.add(70)
t.add(85)
t.add(10)
t.add(78)
t.add(60)
t.add(88)
t.add(15)
t.add(12)
print("preorder")
t.preorder(t.root)
print("inorder")
t.inorder(t.root)
print("postorder")
t.postorder(t.root)
```

OUTPUT

```
Name: Deepak Maurya
Roll No.: 1724
80 addedon left of 100
70 addedon left of 80
85 data on right of 80
10 addedon left of 70
78 data on right of 70
60 data on right of 78
88 data on right of 85
15 addedon left of 60
12 addedon 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
12
15
60
10
78
70
88
85
80
100
```

PRACTICAL-12

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.

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

Traversal is a process to visit all the node of a tree and may print their value too There are 3 ways to traverse a tree:-

i) Inorder:- The left subtree is visited 1st then the root & later the right subtree. We should always remember that every node may represent a subtree itself. Output produced is ~~stored~~ sorted values in ascending order.

ii) Pre-order:- The root node is visited 1st then the left subtree and finally the right subtree.

iii) Post-order:- The root node is visited too, left subtree, then the right subtree and finally root node

PRACTICAL-13

Aim :- Merge Sort

Theory :-

Mergesort is a sorting technique based on divide and conquer technique with worst case time complexity being, it one of the most respected algorithm.

Merge sort first divides the array into equal halves and then combines them in a sorted manner.

It divides inputs array into two halfs, calls itself for the two halfs 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 merges the two sorted sub-array into one.

SOURCE CODE:

```
#MergeSort
print("Name:Deepak Maurya \nRoll No.: 1724")
def sort(arr,l,m,r):
  n1=m-l+1
  n2=r-m
  L=[0]*(n1)
  R=[0]*(n2)
  for i in range(0,n1):
      L[i]=arr[l+i]
  for j in range(0,n2):
      R[j]=arr[m+1+j]
  i=0
  j=0
  k=l
  while i<n1 and j<n2:
    if L[i]<=R[j]:
      arr[k]=L[i]
      i+=1
    else:
      arr[k]=R[j]
      j+=1
    k+=1
  while i<n1:
    arr[k]=L[i]
    i+=1
    k+=1
  while j<n2:
    arr[k]=R[j]
    j+=1
    k+=1
def mergesort(arr,l,r):
  if l<r:
    m=int((l+(r-1))/2)
    mergesort(arr,l,m)
    mergesort(arr,m+1,r)
    sort(arr,l,m,r)
arr=[24,27,38,9,7,18,45,1,93]
print("Before Mergesort\n",arr)
n=len(arr)
mergesort(arr,0,n-1)
print("After Mergesort\n",arr)
```

OUTPUT:

```
Name: Deepak Maurya
Roll No.: 1724
Before Mergesort
 [24,27,38,9,7,18,45,1,93]
After Mergesort
 [1,7,9,18,24,27,38,45,93]
```