SE

ds practical1.py - C:/Users/Abhishek/Desktop/ds practical1.py (3.7.4)

File Edit Format Run Options Window Help

```
print("Name: Abhishek Sharma")
print("Roll-no: 1741")
print("Class : FYCS-A")
a=[11,32,44,67]
b=0
print(a)
search=int(input("Enter number to be searched : "))
for i in range(len(a)):
    if a[i]==search:
        print("Number found at : ",i+1)
        b=1
        break
if(b==0):
    print("Number not found")
```

PRACTICAL-1 (SE) 33

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

Theory:- The process of identifying or finding a particular record is called as searching. There are two types of search, they are as follows:

a) SORTED a) Linear search
b) UNSORTED b) Binary search

Here we will look on the unsorted linear search, also known as the sequential search, is a process that checks every elements in the list sequentially until the desired element is found. When the elements to be searched are not specificially arranged in ascending or descending order. They are arranged in random variable manners. That is what it called as the unsorted linear search. The data is entered in an random manner. User needs to specify the elements to be searched in the list. Check the condition that whether entered numbers matches if it, matches then display the location pulse increment (+) as data is stored in floating point.

If all the elements are checked one by one and element not find ~~the~~ ~~whether~~ it then prompt message number not found.


```
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/Abhishek/Desktop/ds practical1.py ==========
Name: Abhishek Sharma
Roll-no: 1711
Class : FYCS-A
[11, 32, 44, 67]
Enter number to be searched : 11
Number found at :  1
>>>
```

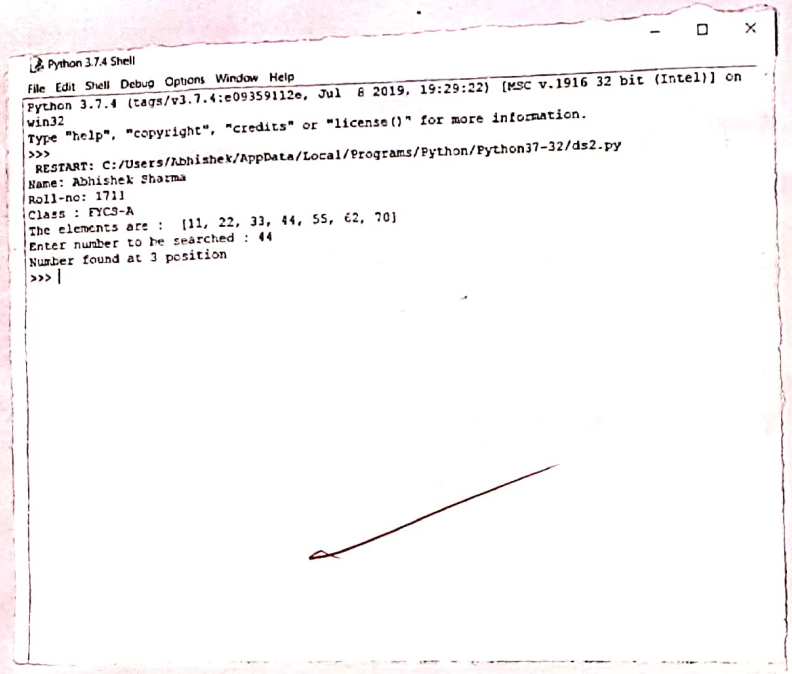
Aim:- To search a number from the list using linear sorted method.

THEORY: SEARCHING and sorting are the different modes or types of data-structures.

SORTING:- To tasically sort the inputed data in ascending or descending order.

SEARCHING - To search the elements and to display the same.

In searching that too is linear sorted search. The data is arrange in ascending to ascending order. That i saw what means by searching through sorted that is how will arranged data.



Sorted Linear search:-

→ The user is supposed to enter data in sorted ~~manner~~ manner.
→ User has to given an element, for searching through sorted list.
→ If element is found, display with an updation of value is stored from the location.
→ If data or elements we can check the condition from starting point till the last element of not then without any processing will end say number not in the list.

PRACTICAL-3

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

Theory:- A binary search also known as a half-interval search, is an algorithm used in computer science to locate a specified value (key) within an array. For the search to be binary the array must be sorted in either ascending or descending order. At each step of algorithm a comparison is made and the procedure branches into one of two directions, specifically the key value is compared is made and the middle element of the array. If the key value is less than or greater than the middle element of the array, ~~If the key value is less than or~~ The algorithm knows which half of the array to continue searching in because the array is sorted.

BINARY_SEARCH

Source code:

```
print("Name : Abhishek_Sharma")
print("Rno : 1711")
print("FYBSC-CS(A)")

a=[11,22,45,34,87]
print(a)
search=int(input("Enter number to be searched from the list:"))
l=0
h=len(a)-1
m=int((l+h)/2)
if((search<a[l]) or (search>a[h])):
   print("Number not in RANGE!")
elif(search==a[h]):
   print("number found at location :",h+1)
elif(search==a[l]):
   print("number found at location :",l+1)
else:
   while(l!=h):
      if(search==a[m]):
         print ("Number found at location:",m+1)
         break
      else:
         if(search<a[m]):
            h=m
            m=int((l+h)/2)
         else:
            l=m
            m=int((l+h)/2)
      if(search!=a[m]):
         print("Number not in  given list!")
         break
```

Output:

Case1:
Name : Abhishek_Sharma
Rno : 1711
FYBSC-CS(A)
[11, 22, 45, 34, 87]
Enter number to be searched from the list:11
number found at location : 1

Case2:
Name : Abhishek_Sharma
Rno : 1711
FYBSC-CS(A)
[11, 22, 45, 34, 87]
Enter number to be searched from the list:100
Number not in RANGE!

Case3:
Name : Abhishek_Sharma
Rno : 1711
FYBSC-CS(A)
[11, 22, 45, 34, 87]
Enter number to be searched from the list:46
Number not in given list!

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

PRACTICAL-4

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

Theory:- Sorting is type in which any random data is stored sorted i.e. arranged in ascending or descending order. BUBBLE SORT sometimes referred to 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 sorting 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 compared one element checks if condition fails they only swaps otherwise goes on.

BUBBLE_SORT

Source code:

```
print("Name : Abhishek_Sharma")
print("Rno : 1711")
print("FYBSC-CS(A)")
a=[1,33,24,66,75,87,33]
print("Before BUBBLE SORT elemets list: \n ",a)
for passes in range (len(a)-1):
    for compare in range (len(a)-1-passes):      if(a[compare]>a[compare+1]):
        temp=a[compare]
        a[compare]=a[compare+1]
        a[compare+1]=temp
print("After BUBBLE SORT elemnts list: \n",a)
```

Output:

```
Name : Abhishek_Sharma
Rno : 1711
FYBSC-CS(A)
Before BUBBLE SORT elemets list:
 [1, 33, 24, 66, 75, 87, 33]
After BUBBLE SORT elemnts list:
 [1, 24, 33, 33, 66, 75, 87]
```

Example:-

First pass:-
(5 1 4 2 8) → (1 5 4 2 8) Here algorithm compares first two elements and swaps. since 5>1

(1 5 4 2 8) → (1 4 5 2 8) swap since 5>4
(1 4 5 2 8) → (1 4 2 5 8) swaps since 5>2
~~(1 4 2 5 8) → (1 4 2 5 8) swaps since~~

Now, since these elements are already in order (8>5) algorithm doesnot swap them.

Second pass:-
(1 4 2 5 8) → (1 4 2 5 8)
(1 4 2 5 8) → (1 2 4 5 8) swap ∵ 4>2
(1 2 4 5 8) → (1 2 4 5 8)

Third pass:-
(1 2 4 5 8) It checks and gives the data in sorted order.

PRACTICAL-5

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 the data and pop which removes the data. The order may be LIFO (last in first out) or FIFO (first in last out).

The basic operations performed in the stack are as follows:-

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

ii) 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 under flow.

—CODE—

```
print("NAME : Abhishek 1711")
class Stack:
    global tos
    def __init__(self):
        self.l=[0,0,0,0,0,0,0]
        self.tos=-1
    def push(self,data):
        n=len(self.l)
        if self.tos==n-1:
            print("Stack is full")
        else:
            self.tos=self.tos+1
            self.l[self.tos]=data
    def pop(self):
        if self.tos<0:
            print("Stack is empty")
        else:
            k=self.l[self.tos]
            print("Data : ",k)
            self.tos=self.tos-1
s=Stack()
s.push(21)
s.push(24)
s.push(25)
s.push(26)
s.push(27)
s.push(38)
s.push(78)
s.push(84)
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
```

```
----OUTPUT----
Stack is full
Data :  78

Data :  38
Data :  27
Data :  26
Data :  25
Data :  24
Data :  21
Stack is empty
>>>
```

43

PRACTICAL-6

Aim:- To determine que and add and delete the data.

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 follows the FIFO structure.

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

44

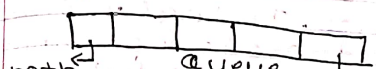
—CODE—

```
print("NAME : Abhishek 1711")
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=self.r+1
        else:
            s=self.r
            self.r=0
            if self.r<self.f:
                self.L[self.r]=data
                self.r=self.r+1
            else:
                self.r=s
                print("Queue is full")
    def remove(self):
        n=len(self.L)
        if self.f<=n-1:
            print("Data removed : ",self.L[self.f])
            self.f=self.f+1
        else:
            s=self.f
            self.f=0
            if self.f<self.r:
                print(self.L[self.f])
                self.f=self.f+1
            else:
                print("Queue is empty")
                self.f=s
Q=Queue()
Q.add(44)
Q.add(55)
Q.add(66)
Q.add(77)
```

```
Q.add(88)
Q.add(99)
Q.remove()
Q.add(44)


—OUTPUT—
NAME : Abhishek
ROLL NO. : 1711
Data added : 44
Data added : 55
Data added : 66
Data added : 77
Data added : 88
Queue is full
Data removed : 44
>>>
```

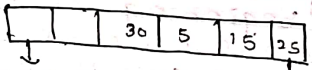

Front is used to get the front data item from a queue

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



both sides.

queue can have ends.



front = 2
Rear = 5

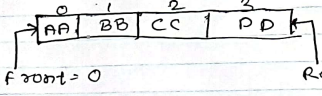
4/2

PRACTICAL- 7

Aim:- To demonstrate the use of circular que in data structu

Theory:- The queue that is implemented using an array suffer from one limitation. In that implementation there is a possibility that que is reported as full, even though it is actually there might be empty at the beginning of the queue. To, overcome this limitation, we can implement 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.

Example :-

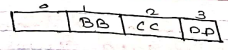


—CODE—

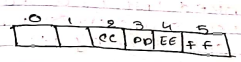
```
print("NAME : Abhishek 1711")
class Queue:
    global r
    global f
    def __init__(self):
        self.r=0
        self.f=0
        self.L=[0,0,0,0,0,0]
    def add(self,data):
        n=len(self.L)
        if self.r<n-1:
            self.L[self.r]=data
            print("Data added : ",data)
            self.r=self.r+1
        else:
            s=self.r
            self.r=0
            if self.r<self.f:
                self.L[self.r]=data
                self.r=self.r+1
            else:
                self.r=s
                print("Queue is full")
    def remove(self):
        n=len(self.L)
        if self.f<=n-1:
            print("Data removed : ",self.L[self.f])
            self.f=self.f+1
        else:
            s=self.f
            self.f=0
            if self.f<self.r:
                print(self.L[self.f])
                self.f=self.f+1
            else:
                print("Queue is empty")
                self.f=s
Q=Queue()
Q.add(44)
Q.add(55)
Q.add(66)
Q.add(77)
```

Q.add(88)
Q.add(99)
Q.remove()
Q.add(44)

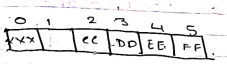
—OUTPUT—
NAME : Abhishek
ROLL NO. : 1711
Data added : 44
Data added : 55
Data added : 66
Data added : 77
Data added : 88
Queue is full
Data removed : 44
>>>



Front = 1 Rear = 3



Front = 2 Rear = 5



Front = 2 Rear = 0

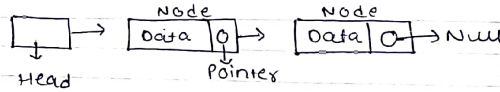
PRACTICAL- 8

Aim:- To demonstrate the use of linked list in data structure.

Theory:- A linked list is a sequence of data structures. Link list is a sequence of list which contains items. Each link contains a connection to the another link.

- LINK:- Each link of a linked list can store a data called as element.
- NEXT:- Each link of 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 First.

Representation:-



LINKED LIST

Source Code :-

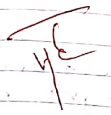
```
print("Abhishek Sharma :- 1711")

class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None
class SLinkedList:
    def __init__(self):
        self.headval = None
    def Inbetween(self,middle_node,newdata):
        if middle_node is None:
            print("The mentioned node is absent")
            return

        NewNode = Node(newdata)
        NewNode.nextval = middle_node.nextval
        middle_node.nextval = NewNode
    def listprint(self):
        printval = self.headval
        while printval is not None:
            print (printval.dataval)
            printval = printval.nextval
list = SLinkedList()
list.headval = Node("Jan")
e2 = Node("Feb")
e3 = Node("March")

list.headval.nextval = e2
e2.nextval = e3
list.Inbetween(list.headval.nextval,"April")
list.listprint()
```

Output :-

```
Abhishek Sharma :- 1711
Jan
Feb
April
```

Types of linked list:-
-> Simple.
-> Doubly.
-> circular.

Basic Operations:-
-> Insertion
-> Deletion
-> Display
-> Search
-> Delete.



CP

PRACTICAL-9

Aim:- TO evaluate post fix expres
using Stack.

Theory:- stack is an ADT and work
on LIFO i.e. push and Pop
operations.

A postfix expression is a colle
-tion of operators and
operands in which the
operators are placed after th
operands.

Steps to be followed:-
i) Read all the symbols one b
one from left to right in
the given postfix expressio
ii) If the reading symbol is opera
then push it on to the stack
iii) If the reading symbol is
operator (+, -, *, /, etc) then perf
two pop operations and store
two popped operands in two
different variables. Then perfor
reading symbol operation
using operand 1 and operand 2
and push result back onto th
stack.

Postfix-Evaluation

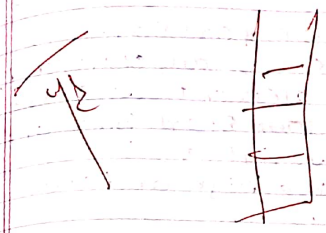
Source-Code : -

```
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="4 55 + 62 23 - *"
r=evaluate(s)
print("The evaluated value is:",r)
print("Abhishek Sharma :- 1711")
```

Output : -

```
The evaluated value is : 2301

Abhishek Sharma :- 1711
```

iv.) Finally perform pop operation and display the popped value as final result.



PRACTICAL-10

Aim:- To sort the element using quicksort.

Algorithm:-

i) Create a function partition with three attributes array, start, end
ii) set the pivot element as start
iii) increment the start by 1 and store it into a variable.
iv) create a variable high and store it into end.
v) use the while loop with appropiate condition if the low is less than or equal to high and array of high is greater than or equal to pivot decrement high by 1.
vi) And if low is less than equal to high and array of low is less than equal to pivot increment low by 1.
vii) use the if conditional statement with appropiate condition and swap the variables values and by
viii) Define a function quick sort and call the function with three attributes.
ix) create an array of unsorted element and print the array

SOURCE-CODE:-

```
def partition(array, start, end):
    pivot = array[start]
    low = start + 1
    high = end
    while True:
        while low <= high and array[high] >= pivot:
            high = high - 1
        while low <= high and array[low] <= pivot:
            low = low + 1
        if low <= high:
            array[low], array[high] = array[high], array[low]
        else:
            break
array[start], array[high] = array[high], array[start]
    return high
def quick_sort(array, start, end):
    if start >= end:
        return
p = partition(array, start, end)
    quick_sort(array, start, p-1)
    quick_sort(array, p+1, end)
array = [29,99,27,41,66,28,44,78,87,19,31,76,58,88,83,97,12,21,44]
quick_sort(array, 0, len(array) - 1)
print("Sorted array is : ",array)
print("Abhishek\n1711")
```

OUTPUT:

```
>>>Sorted array is : [12, 19, 21, 27, 28, 29, 31, 41, 44, 44, 58, 66, 76, 78, 83, 87, 88, 97, 99]
Abhishek
1711
```

Source code:

```
print("Abhishek Sharma \n 1711")
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):
ifself.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,"added on left of",h.data)
break
else:
ifh.r!=None:
                h=h.r
else:
h.r=newnode
print(newnode.data,"added on right of",h.data)
break
defpreorder(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)
defpostorder(self,start):
if start!=None:
self.postorder(start.l)
self.postorder(start.r)
print(start.data)
```

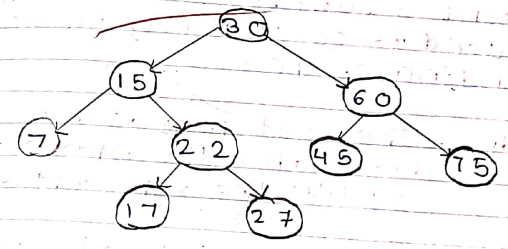

Aim:- Binary Tree and Travesal

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 implement 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.



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

There are 3 ways which we use to travese a tree:-

1) Inorder
2) Preorder
3) Post Order

Inorder:- The left-subtree is visit 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 key values in Ascending order.

Preorder:- The root node is visited first then 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

```
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:

```
Abhisheksharma
1711
80 added on left of 100
70 added on left of 80
85 added on right of 80
10 added on left of 70
78 added on right of 70
60 added on right of 10
88 added on right of 85
15 added on left of 60
12 added on left of 15
preorder
100
80
70
10
60
15
12
78
85
88
inorder
10
12
15
60
70
78
80
85
88
100
postorder
12
15
60
10
78
70
88
85
80
100
```

PRACTICAL-12

Aim:- To demonstrate the use of Merge Sort.

Theory:-

Merge sort is a sorting algorithm based on divide and conquer technique. Its worst-case time complexity being zero (n log n), it is one of the most used algorithm.

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

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

Source code:

```
Name: Abhishek Sharma
def mergeSort(arr,l,r):
    if l<r:
        m=int((l+(r-1))/2)
        mergeSort(arr, l, m)
        mergeSort(arr, (m+1), r)
        merge(arr, l, m, r)
arr = [12, 11, 13, 5, 6, 7,98,66,42,63]
n = len(arr)
print ("Given array is")
print(arr)

mergeSort(arr,0,n-1)
print ("\n Sorted array is")
print(arr)
```

Output:

```
Abhishek Sharma
1711
Given array is
[12, 11, 13, 5, 6, 7, 98, 66, 42, 63]
Sorted array is
[5, 6, 7, 11, 12, 13, 42, 63, 66, 98]
```