★★ INDEX ★★

No.	Title	Page No.	Date	Staff Member's Signature
10.	del statement		4/1/20	
	II nd Sem.			
1.	Linear Search (sorted and unsorted)	35	25/12/19	
2.	Binary search	39	2/12/19	
3.	Bubble sort	42	9/12/19	
4.	Quick sort	44.	16/12/19	
5.	Stack Implementation.	46	6/1/20	06/01/20
6.	Implementing a queue using python list	49	13/1/20.	
7.	Program on Evaluation of given string by using stack in python environment	51	20/1/20	20/01/20
8.	Implementation of linked list			
9.	Merge sort.			

★★ INDEX ★★

No.	Title	Page No.	Date	Staff Member's Signature
10	Implementation of sets in python			03
11.	Binary Tree		10/2/20	10/01/20

PRACTICAL - 1

AIM: Implement the linear search to find an item in the list.

THEORY:

Linear search

Linear search is one of the impliest searching ~~program~~ algorithm in which targeted item is sequentially matched with each item in the list. It is the loorst searching algorithm with worst case time complexity. It is a force approach. On the other hand incase of an ordered list, instead of searching the list in sequence. A binary search is used which will start by examining the middle term.

Linear search is a technique to compare each and every element with the key element to be found, if both of them matches, the algorithm returns that element found, and its position is also found.

1] Unsorted:
Algorithm:

Step 1: Create an empty list and assign it to a variable

Step 2: Accept the total no. of elements to be inserted into the list from the user. Say 'n'

Step 3: List for loop for adding the element into the list

Step 4: Print the new list

Step 5. Accept an element from the user that to be searched in the list

Step 6: Use for loop in a range from '0' to the total no. of elements to search the elements from the list

Step 7: Use if loop that the elements in the list is equal to the element accepted from the user.

Step 8 If the element is found then print the statement that the element is found along with the elements position.

Step 9 Use another if loop to print that the element is not found, if the element which is accepted from the user is not there in the list

step 10: Draw the output of given algorithm


```
print ("Linear search")
a=[]
n = int (input(" Enter a range"))
for s in range (0,n):
    s = int (input (" Enter each elements no"))
    a.append ()
    print (a)
c= int (input (" Enter a number to be searched"))
for i in range (0,n):
    if (a[i] == c):
        print (" found at position", i)
else: ~~abort~~
    print (" not found")
```

```
Linear Search
Enter a range 5
Enter each elements no 4
[4]
Enter each elements no 11
[4,11]
Enter each elements no 6
[4,11,6]
Enter each elements no 1
[4,11,6,1]
Enter each elements no 12
[4,11,6,1,12]
Enter a number to be searched 7
Element not found
```

```
print (" Linear search")
a=[ ]
n= int (input (" Enter a range")
for s in range (0,n):
    s= int (input (" enter each elements no."))
    a. append ( )
    a. sort ( )
    print (a)
c = int (input (" Enter a no. to be searched:"))
for i in range (0,n):
    if (a[i] == c):
        print (" Element found at position", i)
else:
    print (" not found");
```

```
Linear search
enter the range 6
enter each elements no. 2
[2]
enter each elements no. 100
[2,100]
enter each elements no. 99
[2, 99, 100]
Enter each elements no. 56
[2, 56, 99, 100]
enter each elements no. 34
[2, 34, 56, 99, 100]
enter each elements no. 78
[2, 34, 56, 78, 99, 100]
enter a no. to be searched: 34
Element found at position 1
```


2.] Sorted linear Search:
Algorithm

step1: Create an empty list and assign it to a variable.

step 2: Accept total no. of elements to be inserted into the list from user, say 'n'

step 3: Use For loop for append method ( ) to add the elements in the list

step 4: Use sort( ) method to sort the accepted element and assign in increasing order the list then print the list.

step 5: Use If statement to give the range in which element is found in given range then display " Element not found"

step 6: Then use else statement, if element not found in the range then satisfy the given condition.

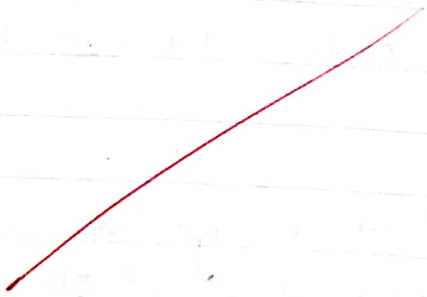
step 7: Use for loop in range from 0 to the total no. of elements to be searched before doing this accept an search no. is from the user using input statement.

step 8: Use if loop the elements in the list is equal to the element accepted from user.

Step 9. If the element is found then print the statement that the element is found along with the element position.

Step 10. Use another if loop to print that the element is not found if the element which is accepted from user is not their in the list.

Step 11 Attach the input and output of above algorithm.



PRACTICAL-2

AIM: Implement Binary search to find an searched no. in the list.

THEORY:

Binary search

Binary searched is also known as half interval search, logarithmic search or binary search. It a search algorithm that finds the position of a target value within a sorted array. If you are looking for the number which is at the end of the list then you need to search entire list in linear search which is the time consuming. This can be avoided by using Binary search.

ALGORITHM:

step1: Create empty list and assign it to a variable.

step2: Using input method, accept the range of given list

step3: Use for loop, add elements in list using append() method.

step4: Use sort() method to sort accepted element and assign it in increasing ordered list print the list after sorting.

Step 5. Use if loop to give the range in which element is found in given range then display a message " Element not found.

Step 6. Then use the else statement, if element is not found in range then satisfy the below condition.

Step 7 Accept an argument and key of the element that element has to be searched.

Step 8. Initialize first to 0 and last to last element of the list, as array is starting from 0 hence it is initialized 1 less than the total count.

Step 9: Use for loop and assign the given range.

Step 10: If statement in the list and still the element to be searched is not found then find the middle element (m).

Step 11: Else if the item to be searched is still less than the middle term then. Initialize last(h) = mid(m)-1

else

Initialize first(f) = mid(m)-1

Step 12: Repeat till you found the element with the input and output of above algorithm.


```
a=[ ]
n = int(input(" Enter the range"))
for b in range (0,n):
    b = int(input(" Enter the number"))
    a. append (b)
    a. sort ( )
    print (a)
s= int (input(" Enter a no. to be searched "))
if ( s<a[0] or s>a[n-1]):
    print (" Element not found").
else:
    f=0
    l=n-1
    for i in range (0,n):
        m = int ((f+l)/2)
        print (m)
        if (s==a[m]):
            print (" Element found at :" , m)
            break
        else:
            if (s<a[m]):
                l=m-1
            else:
                f=m+1
```

9b

output:

```
Enter the range 5
Enter the number 6
[6]
Enter the number 10
[6,10]
Enter the number 16
[6,10,16]
Enter the number 1
[6,10,16,1]
Enter the number 7
[6,10
```

output:

```
Enter the range 5
Enter the number 6
[6]
Enter the number 3
[3,6]
Enter the number 8
[3,6,8]
Enter the number 11
[3,6,8,11]
Enter the number 5
[3,5,6,8,11]
Enter a no. to be searched 6
2
2
element found at: 2
```

PRATICAL-3

BUBBLE SORT

AIM: Implentation of bubble sort program on given list.

THEORY: Bubble sort is based on the idea of repeatdly temporary pair of adjacent element and then swapping their pair if they exist in the wrong order. This is the simplest form of sorting available. In this, we sort the given element in ascending or descending order by ~~unpairing~~ comparing two adjacent elements at a time.

ALGORITHM:

step 1: Bubble sort algorithm starts by comparing the first two elements of an array and swapping if necessary.

step 2: If we want to sort the elements of array ascending order then first element is greater than second, then we need to swap the element.

step 3: If the first element is smaller than second then we do not swap the element.


```
print (" bubble sort")
a=[]
b= int (input (" Enter the no. of elements."))
for s in range (0, b).
    S= int (input (" Enter elements. "))
    a. append (S)
    print (a)
n = len(a)
for i in range (0,n):
    for j in range (n-1).
        if (a[i] < a[j]):
            tmp = a[j]
            a[j] = a[i]
            a[i] = tmp
print
print (" Elements after sorting. ", a)
```

output:

```
bubble sort
Enter the no. of elements - 5
Enter elements: 3
[3]
Enter elements. 8
[3] [8] [3, 8]
Enter elements: 4
[3,8,4]
Enter elements: 10
[3,8,4,10]
Elements after sorting - [3,4,8,10]
```

step 4: Again second and third elements are compared and swapped if it is nescessary and this process go on until last and second last element is compared and swapped.

step 5: There are n elements to be sorted then the process mentioned above should be repeated n-1 times to get the required result.

step 6: Stick the output and input of above algorithm of bubble sort ~~sts~~ step wise.

PRACTICAL-4

Quick Sort.

Aim: Implement Quick sort to sort the given lists.

Theory: The quick sort is a recursive algorithm based on the divide and conquer technique.

Algorithm:

Step 1: Quick sort list selects a value, which is called pivot value, first element serve as our first pivoat value, since we know that first will eventually end up as last in that list.

Step 2: The partition pretests will happen next it will find the split point and at the same time move other items to the appropriate side of list, either less than or greater than pivot value.

Step 3: Partitioning begins by locating two positions markers lets call them leftmark and right mark at the beginning and end of remaining items in the list. The goal of the

code

```
def quick(alist):
    help(alist, 0, len(alist)-1)
def help(alist, first, last):
    if first < last:
        split = part(alist, first, last)
        help(alist, first, split-1)
        help(alist, split+1, last)
def part(alist, first, last):
    pivot = alist[first]
    l = first+1
    r = last
    done = False
    while not done:
        while l<=r and alist[l]<=pivot:
        l = l+1
        while alist[r] >= pivot and r>=l:
            r=r-1
        if r<l:
            done = True
        else:
            t= alist[l]
            alist[l] = alist[r]
            alist[r] < t
        t = alist[first]
        alist[first] = alist[r]
        alist[r] = t
        return r.
X = int(input("Enter range for the list:"))
alist []
```

```
for b in range(0, x):
    b = int(input("Enter the elements:"))
    alist.append(b)
    n = len(alist)
quick(alist)
print(alist)
```

Output:

```
Enter range for the list: 4
enter the elements: 6
enter the elements: 8
enter the elements: 4
enter the elements: 9
[4, 4, 6, 9]
```


partition process is to move items that are wrong side with respect to pivot value while also converging on the split point.

step 4: We begin by incrementing leftmark until we locate a value that is greater than the P.V. We then decrement rightmark until we find value that is less than the pivot value. At this point we have discovered two items that are ~~equal~~ place with respect to eventual split point.

Step 5: At the point where rightmark becomes less than leftmark, we stop. The ~~point~~ position of rightmark is how the split point.

Step 6: The pivot value can be exchanged with the content of split point and p.V (pivot value) is no.

PRACTICAL-5

AIM: Implementation of stack using python list

THEORY: A stack is a linear data structure that can be represented in the real world in the form of a physical stack or a pile. The elements in the stack are added or removed only from one position that the topmost position. Thus the stack works on the LIFO (last in first out) principle as the element that was inserted last will be removed first. A stack can be implemented using array as well as linked list. Stack has three basic operation: push, pop, peek. The operations of adding and removing the elements is known as push & pop

ALGORITHM:

step 1: Create a class stack with instance vaiable items

step 2: Define the init method with self self argument and initialize the initial value and then initalize to an empty list.

```
print("Sushmita Rangawar")
class stack:
    global tos
    def __init__(self):
        self.i=[0,0,0,0,0]
        self.tos=-1
    def push(self,data):
        n=len(self.i)
        if self.tos==n-1:
            print("stack is full")
        else:
            self.tos=self.tos+1
            self.i[self.tos]=data
    def pop(self):
        if self.tos<0:
            print("stack empty")
        else:
            k=self.i[self.tos]
            print("data=",k)
            k=self.i[self.tos]=0
            self.tos=self.tos-1
    def peek():
s=stack()
```


step3: Define Methods push and pop under the class stack.

step4: Use if statement to given the condition that if length of given list is greater than the range of list then print stack is full.

step5: Or else print statement as insert the element into the stack and initialize the value

step6: Push method used to insert the element but pop method used to delete the element from the stack.

step7: If in pop method, value is less than 1 then return the stack is empty or else delete the element from stack at topmost position

step8: Assign the element values in push method to an and print the given value is popped not.

step9: first condition check whether the no. of element are zero with the second case whether top is assigned any value. If top is not assigned any value. then it can be store that stack is empty.

Step 10: Attach the input and output of above algorithm.

output:

Sushmita Rangawar

```
>>> s.i
 [0,0,0,0,0]
>>> s.push(10)
>>> s.push(20)
>>> s.push(30)
>>> s.i
 [10,20,30,0,0]
>>> s.pop()
 data=30
>>> s.i
[10,20,0,0,0]
```

s.peek() → ?

06/01/2021

```
class queue
    global r
    global f
    def __init__(self):
        self.r = 0
        self.f = 0
        self.l = [0,0,0]
    def enqueue(self, data):
        n = len(self.l)
        if self.r<n:
            self.l[self.r] = data
            self.r = self.r+1
            print("Element inserted ...", data)
        else:
            print("Queue is full")
    def dequeue(self)
        n = len(self.l)
        if self.f<n
            print(self.l[self.f])
            self.l[self.f]=0
            print("Element deleted ...")
            self.f = self.f+1
        else:
            print("Queue is empty")
queue()
```


PRACTICAL - 6

Aim: Implementing a queue using python list

Theory: Queue is a linear data structure which has 2 references front and rear. Implementing a queue using python list is the simplest as the python list provides inbuilt functions to perform the specified operations of the queue. It is based on the principal that a new element is inserted after rear and element of queue is deleted which is at front. In simple terms, a queue can be described as a data structure based on first in first out FIFO principle.

Queue() — Creates a new empty queue

Enqueue() — Insert an element at the rear of the queue and similar to that of insertion of linked using tail

Dequeue(): Returns the element which was at the front. The front is moved to the successive element. An dequeue operation cannot remove element if the queue is empty

Algorithm :

Step 1: Define a class Queue and assign global variable then define init() method with self argument in init(), assign or initialize the init value with the help of the self argument.

Step 2: Define a empty list and define enQueue() method with 2 arguments. assign the length of empty list.

Step 3: Use if statement that length is equal to rear then Queue is full or else insert the element in empty list or display that Queue element added successfully and increment by 1

Step 4: Define deQueue() with self argument under this use if statement that front is equal to length of list then display Queue is empty or else give that front is at zero and using that delete the element from front side and increment it by 1

Step 5. Now call the Queue() function and give the element that has to be added in the empty list by using enQueue() and print the list after adding and same for deleting and display the list after deleting the element from the list.

output

>> Q.add(10)
element insert ..10

>> Q.add(20)
element insert ..20

>> Q.add(3)
element insert .. 3

>> Q.add(4)
Queue is full

>> Q.remove
10 element deleted

```
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 = "3 4 5 + *"
r = evaluate(s)
print("The evaluated value is :", r)
```


PRACTICAL- 7

Evaluation of postfix Expression

- **Aim :** Program on Evaluation of given string by using stack in python environment

- **Theory:** The postfix expression is free of any parantheses further we took are priorities of the operator in the program A given postfix expression can easily be evaluated using stacks. Reading the expression is always from left to right in postfix

- **Algorithm :**

step1: Define evalute as function then create a empty stack in python

step2: Convert the string to a list by using the string method split

step3: Calculate the length of string and print it

step4: Use for loop to assign the range of string then give condition using if statement.

steps: Scan the token list from left to right
If token is an operand, convert it from a string to an integer and push the value onto the 'p'

Step 6: If the token is an operator +, /, *, -, ^ it will ~~record~~ need two operands. Pop the 'p' twice. The first pop is second operand and the second pop is the first operand

Step 7: Perform the Arthmatic operation. Push the result back on the 'n'

Step 8: When the input Expression has been Completely processed the result is on the stack. Pop the 'p' and return the value

Step 9: Print the result of string after the evaluation of postfix

Step 10: Attach the output and input of the above algorithm.

Output: The evaluated value is 27
Sushmita

```
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):
```


PRACTICAL-8.

Aim : Implementation of single linked list by adding the nodes from last deposition.

Algorithm:

1) Transversing of linked list means using all the nodes in the linked list in order to perform some operation on them

2) The entire linked list means can be accessed via the first nodes of the linked list the first node of the linked list in term of refered by the head pointer of the linked list.

3) Thus the entire linked list can be transversed using the nodes which is refered by head pointer.

4) Now that we know that we can transverse the entire linked list using the head pointer we should only use it to refer the first node of list only.

5) We should not use the head pointer to transverse entired linked list because the head pointer is Only reference to the 1st node in the linked list, modifying the reference of the head pointer can lead to changes which cannot reverd back.

6) we may lose the reference to the 1st node in our linked list and hence most of our list so in order to avoid making some unwanted changes to the 1st node we will use the temporary node to traverse

7. We will use this temporary node as a copy of the node we are currently traversing since the idea making temporary node a copy of current node the datatype of the temporary node should also be node.

8. But the 1st node is referred by current - so we can transverse to 2nd node as next

9. Similarly we can transverse rest of nodes in the linked list using same method by while loop

10. Our concern now is to find terminating condition for while loop

11. The last node in the linked list is referred to the tail of linked list. Since the last node of linked list does not have any next node, the value is in the next field of the last node.

12. We can refer to the last of linked list self.s none.


```
    head = self.s
while head.next != None
        print (head.data)
        head = head.next
        print (head.data)
s1 = LinkedList()
s1.addL(50)
s1.addL(60)
s1.addL(70)
s1.addL(80)
s1.addB(40)
s1.addB(30)
s1.addB(20)
s1.display()
print("sushmita")
```

output

```
20
30
40
50
60
70
80
sushmita
>>>
```

13. We have to now see how to start transversing the linked list how to identify whether we have reached the last node of linked list or not.

14. Attach the coding or input and output of above algorithm.

PRATICAL-91

Aim: program based on binary search by implementing Inorder, Preorder and Postorder transversal.

~~Algorithm~~

Theory:

1) Binary tree is a tree which supports maximum of 2 children for any node within the Tree. Thus any particular node can have 0 or 1 or 2 children that it is ordered such that one child is identified as left child and other as right child.

→ Inorder : i) Transverse the left subtree, The left subtree intern might have left and right subtree

ii) Visit the root node

iii) Transverse the right subtree and repeat it.

→ Preorder : (i) Visit root node.

ii) Traverse the left subtree. The left subtree intern. might have left and right subtree.

iii) Traverse the right subtree repeat it.

→ Postorder : i) Traverse the left subtree. The left subtree intern might have left and right subtree.

ii) Traverse the right subtree.

iii) Visit the root node.


```
class node:
    global r
    global l
    def __init__(self, l):
        self.l = None
        self.data = l
        self.r = None
class Tree:
    global root
    def __init__(self):
        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, "data on left of", h.data)
                        break
                else:
                    if h.r != None:
                        h = h.r
                    else:
                        h.r = newnode
                        print(newnode.data, " added on right of ",
                        h.data)
                        break
```

```
def preorder (self, start)
if start != None:
    print (start.data)
    self.preorder (start.l)
    self.preorder (start.r)

def inorder (self, start):
    if start != None:
        self.inorder (start.l)
        print (start.data)
        self.inorder (start.r)

def postorder (self, start):
    if start != None:
        self.inorder (self.l)
        self.inorder (self.r)
        print (start.data)
```

```
T= Tree()
T.add(100)
T.add(80)
T.add(20)
T.add(85)
T.add(10)
T.add(78)
T.add(60)
T.add(88)
T.add(15)
T.add(18)
print("preorder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
print("sushmita")
```


Algorithm :

1. Define class node and define init() method with a argument. Initialize the value in this method.

2. Again, Define a class BST that is Binary search tree with init() method with self argument and assign the root is none

3. Define add() method for adding the node define a variable p that p = node(value)

4. Use If statement for checking the condition that root is none the use else statement. For if node is less than the main node then put or arrange that in leftside

5. Use while loop for checking the node is less than or greater than the main node and break the loop if it is not satisfying.

6. Use if statement within that else statement for checking that node is greater than main root then put it into right side.

7. After this, left subtree and right subtree repeat this method to arrange the node according to Binary search Tree.

8. Define Inorder(), Preorder(), and Postorder() with root argument and use if statement that root is none and return that in all.

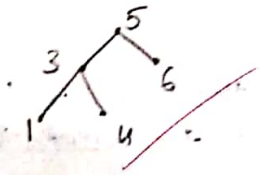
9. Inorder, else statement used for giving that condition first left, root and then right node.

10. For Preorder, we have to give condition in else that first root not, left and then right node.

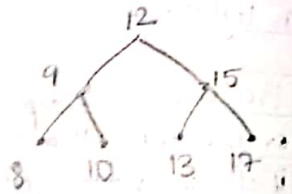
11. For Postorder. In else part, assign left then right and then go for root node.

12. Display the output & input of above algorithm

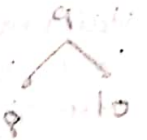
In order : (LVR)



7



5 6 7



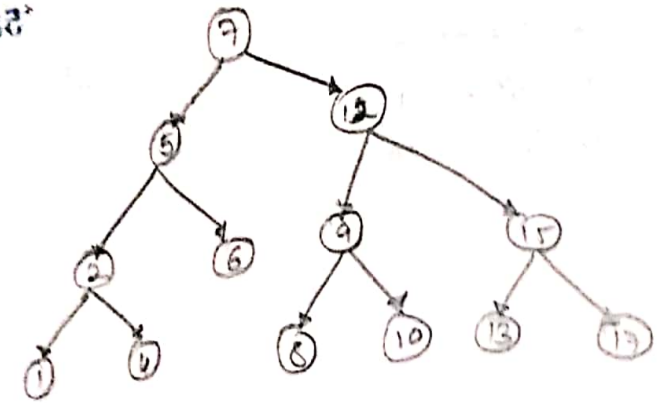
12 15

1 3 4 5 6 7 8 9 10 11 12 13 15 17

output.

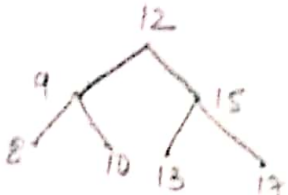
80 added on left of 100
70 added on left of 80
85 added on left of 70
10 added on left of 10
preorder
100
80
70
10
~~15~~
17
85
inorder
10
15
17
70
85
100

Binary search Tree

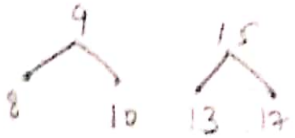


Preorder (VLR)

step1 7

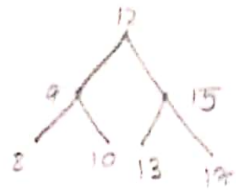


step2 7 5 12



7 5 3 1 4 6 12 9 8 10 15 13 17

Postorder: (LRV)



1 4 3 6 5 8 10 9 13 17 15 12

10/02/2025