PRACTICAL -01.

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

Theory :- Linear Search.

Linear search is one of the simplest searching algorithm in which targeted item is sequentially watched with each item in list.

It is worst 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 middle items.

Linear search is a technique to compare each & every element with the key element, to be found, if both watches, the algorithm returns that elements found & its position is also found.

i] Unsorted :-

Algorithm :-

(1) Create an empty list & assign it to a variable

(2) Accept the total no. of elements to be inserted

into the list from the user say 'x'.

③ Use for looping for adding the elements the list.

④ Print the new list.

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

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

⑦ Use if loop that the elements in the list is equal to the element from u

⑧ If the element is found then print the statement with the element posit

⑨ Draw the output of given algorithm

```
input ("Linear Search")
a = []
n = int(input("Enter the range"))
for s in range(0, x):
    s = int(input("Enter the no. ="))
    a.append(s)
    print(a).
c = int(input("Enter a search no="))
for i in range(0,n):
    if (a[i] == c):
        print("found at position =", i)
    else:
        print("not found")
```

Output :-

```
>>> Linear Search
    Enter the range =3
    Enter the number =2
    [2]
    Enter the number =1
    [1,2]
```

Code :-

```
print("Linear Search")
a = []
n = int(input("Enter a range="))
for s in range(0,n):
    s=int(input("Enter a number="))
    a.append(s)
    a.sort()
    print(a)
c = int(input("Enter a search number="))
for i in range(0,n):
    if(a[i]==c):
        print("found at position=",i)
        break
else:
    print("not found")
```

6 Sorted :-

Sorting means to arrange the element in the increasing / decreasing order.

* Algorithm :-

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

Step 2 : Accept total number of elements to be inserted into the list from user.

Step 3 :- Use for loop for using append method() to add element in the list.

Step 4 :- Use the sort method to sort the accepted elements and assign in increasing order.

Step 5 :- Use if statement to give a range in which elements needs to be found.

Step 6 :- The use else statement, if statement is not found in range then satisfy the condition.

Step 7 : Use for loop in range from 0 to the total no. of elements to be searched.

Step 8 : Attach the input and output above Algorithm.

Output :-

>>> Linear Search.

Enter a range = 3

Enter a number = 2.

[2]

Enter a number = 1

[1,2]

Enter a search number = 2

found in position = 1

```
a = []
n = int(input("Enter a range="))
for b in range(0,n):
    b=int(input("Enter a number"))
    a.append(b)
    a.sort()
    print(a)

s= int(input("Enter a number"))
if (s<a[0] or s>a[n-1]):
    print("Element not found")
else:
    f=0
    l=n-1
    for in range(0,n):
        m = int((f+l)/2)
        print(m)
        if (s==a[m]):
            print("Element found at=",m)
            break.
    else:
        f = m+1
```

Practical - 02.

Aim :- Implement Binary Search to find a searched number in list.

Theory :-

Binary Search.

Binary Search is also known as half-interval search, logarithmic search or binary chop is a search algo that finds the position of a target value within a sorted array.

Algorithm :-

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

Step 2 :- Using input method, accept the range of given list.

Step 3 :- Use for loops, add elements in list using the append method.

Step 4 :- Use sort() method to sort accept element & assign it. Print the list.

Step 5 : Use if loop to give the range in which element is found in given range.

Step 6 :- Use else statement, if statement is found.

Step 7 :- Accept an augument & key of element that element has to be se

Step 8 :- Initialize first to 0 and last t element of list.

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

Step 10 :- Repeat until you found element.

Output :

>>> Enter a range = 4.
Enter a number = 2.
[2]
Enter a number = 1
[1, 2]
Enter a number = 4.
[1, 2, 4]

Enter a search number = 2
Element found at = 1

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

Practical - 03.

Aim : Implementation of Bubble sort programme of given list.

Theory :

Bubble sort.

In this sort, we sort the given element in ascending or descending order by comparing two adjacents 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 in ascending order then first element is greater than second, then we need to swap element.

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

Step 4 :- Again second and third elements a compaud. and swapped if it necesa and this process go on untill last and second last element is compare and swapped.

Step 5 :- If there are n elements to be sorted then used (n-1) time to get required output / result.

Step 6 : Stick the input and output

Output:

Bubble sort
Enter no. of elements =4
Enter elements :6
[6]
Enter elements :5
[6,5]
Enter elements :8
[6,5,8]
Elements after sorting :[2,5,6,8]

Practical - 04.

Aim :- Implement quick sort to sort the given list.

Theory :- The given sort is a recursion algo based on the divide and conquer techique.

Algorithm :-

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

Step 2 :- The partition process will happen next. It will find the split point and at same time move other items to appropriate side of list.

Step 3 :- Partitioning begins by locating two position numbers call them left marks and right marks at the end and beginning of remaining items is last

Practical - 04.

Aim :- Implement quick sort to sort the given list.

Theory :- The given sort is a recursion algo based on the divide and conquer techique.

Algorithm :-

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

Step 2 :- The partition process will happen next. It will find the split point and at same time move other items to appropriate side of list.

Step 3 :- Partitioning begins by locating two position numbers call them left marks and right marks at the end and beginning of remaining items is last

Step 4 : We began by incrementing left mark until we locate a value that is greater than pivious. At this point we have discovered two items that are out of place with respect to eventual split point.

Step 5 :- At the point when right mark becomes less ~~that~~ than left mark, the we stop.

Step 6 :- The pivot value will be exchanged with the content of split point.

Step 7 :- In addition of all items to left of split point are less than p.v and all items to right.

```
def quick sort (a list):
    help ( a list , 0 , len ( a list)-1)
def help (a list , first , last) :
        if first < last :
            split = part ( list , first, last)
            help = (a, list , fir +1 , last)
def part ( a list , first , last) :
    pivot = a list [first]
    l = first +1
    r = last .
    done = false
    while not done :
        while l <=r and list [l] <= pivot :
                l = l+1
        while a list [r] >= pivot and r >=l
                r= r-1
        if r<l :
                done = True .

x = int (input ( " Enter a number = " ))
a list [ ]
for b in range (0, r) :
        b = int ( input ( " Entir a number " ))
        a list .append (b)
        o = len a (list)
```

```
print ("mustafa")
class stack:
    global toss
    def __init__(self):
        self.l=[0,0,0,0,0]
        self.toss=-1
    def push(self,data):
        n=len(self.l)
        if self.toss=n+1
            print("stack is full")
        else:
            self.toss=self.toss+1
            self.l[toss]=data
    def pop(self):
        if self.toss<0:
            print("stack empty")
        else:
            k=self.l[self.toss]
    def peek(self):
        if self.toss<0
        print("stack empty")
        else:
        p=self.l[self.toss]
        print("Top element=",p]
        s=stack()
```

would in the form of a physical stalk or a pile. The element in the stack can be added or removed.

Algorithm :-

Create a class stack with instance variable into.

Define the init method with self augument and initialize the initial value and than initialize empty list.

Define method push and pop under the class stack.

Use if statement to give the condition that if length of given list is greater given range.

5) Use if statement to give the condition that if length of given list is greater than range of list.

6) Push method used to insert the element but pop method used to delete the element.

7) If in pop-method, value is less than return the stack is empty or else delete the element.

8) Attach the input and output by above algorithm.

Output:

Mustafa.

s.push (10)

s.push (20)

s.l

[10, 20, 0, 0, 10]

s.pop()

data = 20

s.l

[10, 0, 0, 0, 0]

s.pack()

Top element = 20.

```
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("Elements inserted...",data)
        else:
            print("Queue is full")
    def enqueue(self):
        n=len(self.l)
        if self.f<n
            print(self.l[self.f])
        else:
            print("Queue is empty")
q=queue()
```

Practical -06.

Aim : Implementing a Queue python list.

Theory :- Queue is a data linear structure which has 2 front and rear implementation a queue using python list is simplest as python list inbuilts # function to perform specified operation of a queue.

Queue () : Creates a new empty queue.

Enqueue () : Insert an element at the rear of queue and similar to that of insertion of list using last.

Dequeue : Returns the element which was at the front. The front is moved to successive element of dequeue operation cannot remove elements of the queue is empty.

Algorithm :-

1) Define a class queue and assign global var than define init method() with self arguement, with the help of self arguement

2) Define an empty list and define enqueue() with 2 arguement.

3) Use if statement that length is equal to max than queue is full insert the element in empty list or display that queue element added succefully by 1.

4) Define enqueue() with self arguement under this use if statement that point is equal to length of list.

5) Now call the queue() and give the element that has to be added in empty list by using enqueue() and print the list.

Output:

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

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

>>> Q.add(30)
element insert = 30

>>> Q.add(40)
queue is full

>>> Q.remove()
10 element deleted.

```
def evaluate():
    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))
```

Practical no. 07.

Aim :- Program of evaluation given string by stack in python environment.

Theory :- The postfix expression is free of any paranthesis further use took care of the priorities of the operators in the program.

Algorithm :-

Define evaluate a function then create an empty stack in python

Convert the string to list by using the string () 'split'.

Calculate the length of string and print.

Use for loop to assign the range of string then give condition use if statement

Scan the token list from left to right. If token is an append. Convert it from string to an integer and push the value.

6) If the token is an operator +, -, /, * it will need two operands. Pop the p is twice. The first pop is second operands and second pop is first operand.

7) Perform the Arithmetic operation. Push the result back on 'S'.

8) Print the ~~string~~ result of the string.

9) Attach the input and output of above algorithm.

```
s = "8 6 9 * + "
r = evaluate (s)
print ( " The evaluate value is ", r)
print ( " AP ")
```

Output :-

The evaluated value is 62.

AP

```
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 add (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 add (self, item) :
    newnode = node (item)
    if self.s = None :
        Self.s = Newnode :
    else :
        newnode.next = self.s
        Self.s = new node
```

Practical No. 8.

Aim : Implementation of single linked list by adding the node from last disposition.

Algorithm :-

1) Transversing of a linked list means visiting all the node in linked list in order to perform some operations on them.

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

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

4) Now that we know that we can transverse the entire linked list using 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 the entire linked list because the head pointer is our reference.

6) We may lose the reference to the 1st node in our linked list and hence most of the one list so in order, we will use temporary function.

7) We will use those temporary node as a copy of node, we are currently transversing since we are making temporary node a copy of current data type of temporary node should also be node.

8) But the 1st node is refered by current so we can transverse to 2nd nodes as h= h.next.

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

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

11) We can refer to last of linked list Self.next = none.

Attach the coding an input & Output of above algorithm.

```
def display (self):
    head = SELF.s
    while head.next! = None:
        print (head.data)
        head = head.next.
    print (head.data)
def delete (Self):
    if self.s = None:
        print ("list is empty")
    else:
        head = Self.s.
        while True:
            if head.next = None.
                d = head
                head = head.next
            else:
                d.next = None
                break.
```

```
S = linked list()
S.add (10)
S.add (80)
S.add (70)
S.add (80)
S.add (40)
S.add (30)
S.add (20)
S.display()
```

Output:

20
30
40
50
60
70
80

```
print("Mustafa")
set1 = set()
set2 = set()
for i in range(8,15):
    set1.add(i)
for i in range(1,12):
    set2.add(i)
print("set1:", set1)
print("set2:", set2)
print("\n")
set3 = set1 | set2
print("Intersection of set1 and set2: set4 *", set4)
print("\n")
if set3 > set4:
    print("set 3 is superset 4")
if set3 < set4:
    print("set4 is Subset of set 3")

Set5 = set3 - set4.
print("\n")

Set5.clear()
print("After applying clear, set 5 is empty set")
print("set 5=", set5)
```

Output :

Practical - 10.

Aim :- Implementation of set using python

Algorithm :-

1) Define two empty set as set 1 and set 2 now use for statement providing the range of above 2 sets.

2) Now add () method used for addition the element according to given range than print -

Find union and intersection of above of 2 set using (and) / (or) () print the set.

Use if statement to find out the subset and supuset of 3 and 4.

Display the element in set 3 is not in 4 using mathematical opration.

Use disjoint () to check that anything is common is present or not. If is mutually exclusive.

#) Use clear to remove or delete the
and print the set after clearing the
element present in set.

Practical - 11.

Aim :- Program based on binary search tree by implementing Inorder, Preorder & Postorder Transversal.

Theory :-
Binary tree is a tree which supports maximum of 2 children for any node within the tree. Thus any particular node can have either 0 or 1 or 2 children. There is another identity of binary tree that is ordered such that a child is identified as left child & other as right child.

1. Inorder :
Traverse the left subtree. The left subtree in turn may have left and right subtrees.
Visit the root node.
Traverse the right subtree and repeat.

Preorder :- Visit the root node.
Traverse the left subtree. The left subtree in return may have left and right subtree.

Postorder :- Traverse the left subtree. The left subtree in return may have left

```
            else:
                h.r = newnode
                print (newnode.data, " added on right of ",
            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)
            self.inorder (start.r)
    def postorder (self, start):
        if start != None:
            self.inorder (start.l)
            self.inorder (start.r)
            print (start.data)
```

```
T = Tree()
T.add (7)
T.add (5)
T.add (12)
T.add (3)
T.add (6)
T.aad (9)
T.add (15)
T.add (1)
T.add (4)
T.add (8)
T.add (10)
T.add (13)
T.add (17)
```

```
print ("preorder")
T.preorder (T.root)
print ("inorder")
T.inorder (T.root)
print ("postorder")
T.postorder (T.root)
```

and right subtree
Traverse the right subtree.
Visit the root node.

Algorithm :-

1) Define class node and define init () method with 2 arguement. Initialize value in the method.

2) Again, define a class BST, with init method () with self arguement and assign root is none.

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

4) Use if statement for checking the condn that root is none than use else statement for if node than main node than arrange.

5) Use if statement within else statement for checking that node is greater than main node than put on arrange.

6) After this left subtree and right subtree repeat this method to arrange the node according to BST.

8) Inorder, Postorder and preorder with recursive arrangement and use of statement that root is none and returns that - is all.

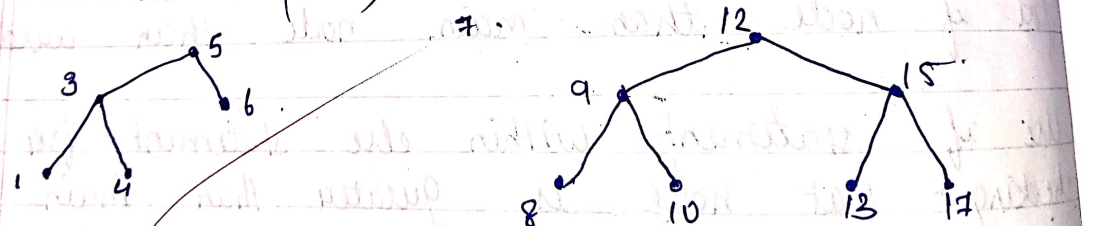
9) In inorder, else statement used for giving that condition first left then right.

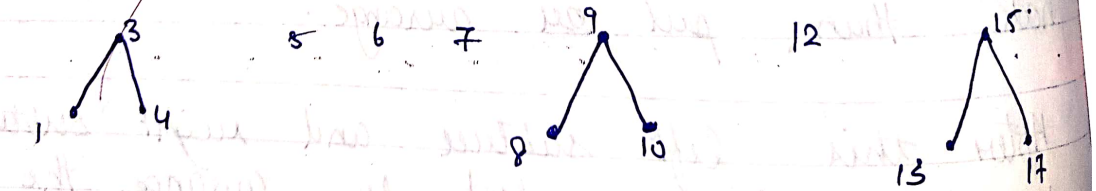
10) For preorder, we have to give condition in else than first root.

11) For postorder, in else, assign left then right and then go for root node.

12) Display the o/p of above algorithm.

Inorder: (LVR)

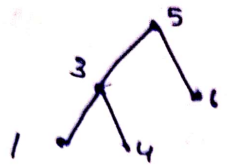
1) 

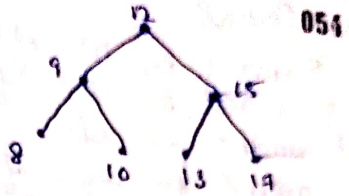
2) 

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

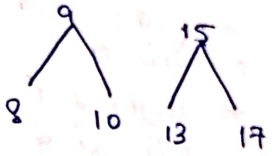
Preorder: (VLR)

1) 7





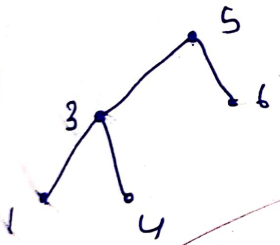
2) 2 5 3 6 12

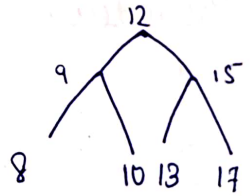


3) 7 5 13 14 6 12 9 8 10 15 13 17.

Postorder: (LRV)

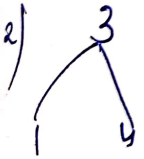
1)

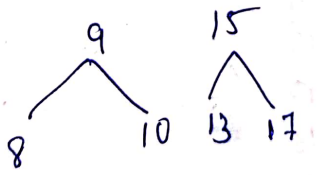




7

2) 3 6 5 9 15 12 17





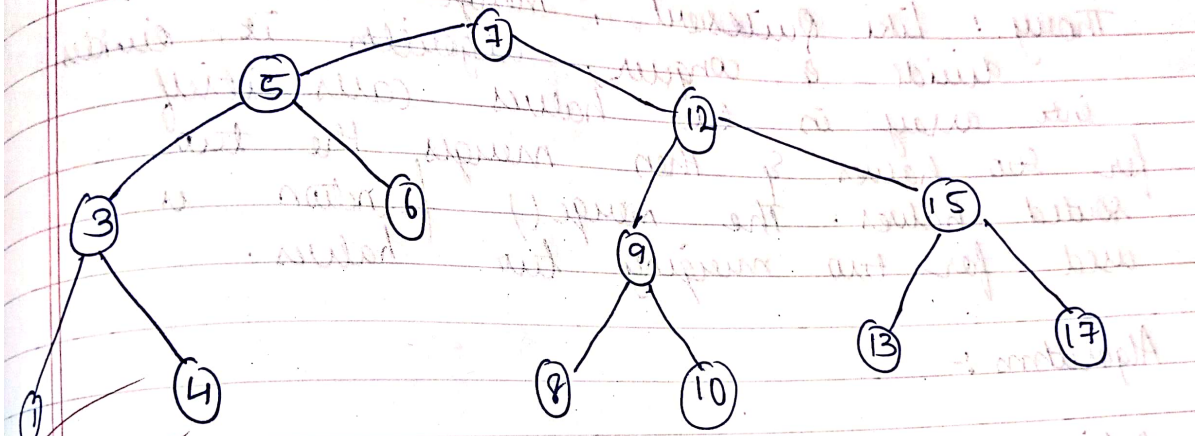
3) 1 4 3 6 5 8 10 9 13 17 15 12 7

Output :-

5 added on left of 7.
12 added on right of 7.
3 added on left of 5.
6 added on right of 5.
9 added on left of 12.
15 added on right of 12.
1 added on left of 3
4 added on right of 3.
8 added on left of 9
10 added on right of 9
13 added on left of 15
17 added on right of 15

Preorder	inorder	Postorder.
7	1	1
5	3	3
3	4	4
1	5	5
4	6	6
6	7	8
12	8	9
9	9	10
8	10	12
10	12	13
15	13	15
13	15	17
17	17	7.

Binary Search Tree :



220

Practical 09

Aim : Implementation of merge sort.

Theory : Like Quicksort, merge sort is a divide & conquer algorithm it divide into array in two halves calls itself for two halves & then merges the two sorted halves. The merge() function is used for two merging two halves.

Algorithms :-

1) Define the sort

2) Stores the starting position of both parts in temporary variables

3) Checks if first part comes to an end or n

4) checks if second part comes to an e or not.

5) Check which parts has smaller elemen

6) Now the real array has element in sor manner including both parts.

```
def [arr, l, m, r]
    n1 = m - l + 1.
    n2 = r - m
    L = [0] * [n1]
    for i in range (0, n1) :
        L[i] = arr [l + i]
    for i in range (0, n2) :
        R[j] = arr [m + 1 + j]
l = 0
j = 0
k = l
while i < n1 & i < n2
    if L[i] <= R[j]
        arr [k] = L[i]
        i += 1.
    else :
        arr [k] = R[j]
        j += 1
        k += 1
    while l < 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) :
```

020

```
if l < r:
    m = int((l + (r-1))/2)
    mergesort(arr, l, m)
    mergesort(arr, m+1, r)
    sort(arr, l, m, r)
print(arr)
n = len(arr)
mergesort(arr, 0, n-1)
print(arr)
```

Output :-

[12 11 13 5 6 7]

[5 6 7 11 12 13].

Defines the correct array in 2 parts.

Sort the 1st part of array.

Sort the 2nd part of array.

Merge both parts by comparing elements of both the parts.

03/03/2024