Practical -1

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 the list. It is worst searching algorithm with worst case time complexity. It is a force approach on the other hand in case 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 matches, the algorithm returns that element found and its position is also found.

Unsorted :

Algorithm:

Create an empty list and assign to a variable.

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

Scanned with CamScanner

Use for loop for adding the elements into th list.

Print the new list.

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

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

Use if loop that the elements in the list equal to the element accepted from user

Draw the output of the given algorithm

Scanned with CamScanner

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

OUTPUT:

```
Enter the range: 3
Enter a number: 1
[1]
Enter a number: 2
[1,2]
Enter a number: 3
[1,2,3]
Enter a number to search: 2
Number found at this position: 1.
```

Scanned with CamScanner

2) Seated linear search

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

Algo:-

create empty list and assign it is a variable

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

Use for loop for using append() method to add the element in the list.

Use sort() method to sort the accepted element and assign is increasing order the list then print the list.

Use if statement to give the range in which element is found in given range the display "Element not found".

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

for loop in range from 0 to the total no. elements to be searched before doing this pt and search number from user using statement

Scanned with CamScanner

```
a = []
n = int(input("Enter the range:"))
for i in range(0,n):
    b = int(input("Enter a numbers:"))
    a.append(b)
a.sort()
print(a)
c = int(input("Enter a number to search:"))
for i in range(0,n):
    if(a[i]==c):
        print("Number found at",i+1)
        break
else:
    print("Number not found")
```

Input

```
Enter the range: 3
Enter a numbers: 1
[1]
Enter a numbers: 2
[1, 3]
Enter a numbers: 2
[1, 2, 3]
Enter a number to search: 4
Number not found
```

Scanned with CamScanner

8

take the input and output of above

go.

Scanned with CamScanner

Practical – 2.

Aim:- Implement binary search to find a search no. in the list.

Theory:- BINARY SEARCH

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

Algorithm:-
Create empty list and assign it to a variab
Using input method, accept the range of gi
list.
Use for loop, add elements in list using append
method.
Use sort() method to sort the accepted eleme
assign it in increasing ordered list print th
list after sorting.

Scanned with CamScanner

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

Scanned with CamScanner

5) Use if loop to give the range in which element is found in given range, then disp a message "element not found".

6) Then use else statement, if statement is found in range then satisfy the below condition

7) Accept an argument and beg of the element the list as away is starting from 0 he it is initialize (less than total coun

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

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 (l) = mid (m) - 1.

2) Repeat till you found the element check the i/p and o/p of above algorithm

Scanned with CamScanner

output

>>> enter the range :3
>>> enter the number : 2.
[2]
>>> enter the number : 1
[1, 2].
>>> enter the number : 4
[1, 2, 4].
>>> enter the number do search : 3
>>> element not found

Scanned with CamScanner

Practical-3

Bubble sort.

Aim: Implementation of bubble sort program on given list.

Theory: Bubblesort is based on the idea of repeatedly comparing pairs of adjacent elements and then swapping their position if they exist in the wrong order. This is the simplest form of sorting available. In this we sort the given elements in ascending or descending order by comparing two adjacent elements at a time.

Algorithm:

Bubble sort algorithm start by comparing the first two element of an array and swapping if necessary.

If we want to sort the elements of array in ascending order then first element is greater than second then i we need to swap the element.

If the element is smaller than second then we. Again second third elements are compared and swapped if it is necessary and this process go on until last and second last element is compared and swapped.

Scanned with CamScanner

```
print("Bubble sort algorithm")
a = []
b = int(input("Enter no. of element"))
for s in range(0,b):
    s = int(input("enter the element"))
    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]:
            temp = a[j]
            a[j] = a[i]
            a[i] = temp
print("elements after sorting are:",a)
```

output:

```
>>> Bubble sort algorithm
Enter number of elements: 3
enter the element: 8
[8]
Enter the element: 9
[8,9]
Enter the element: 1
[8,9,1]
Elements after sorting are: [1,8,9]
```

Scanned with CamScanner

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

Write the output and input of above algorithm of bubble sort stepwise.

Scanned with CamScanner

Practical — 4.

Quick Sort

Aim: Implement Quick sort the given list.

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

Algorithm:

Step1: Quick sort first selects a value, which is called pivot value, first element serve as our first pivot value. Since we know that first will eventually end up as last in that list.

Step2: The partition process will happen next. It will find the split point and at the same time move our items to the appropriate side of the list, either less than or greater than pivot value.

Scanned with CamScanner

```
:P
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 list[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
r = int(input("Enter range for the list"))
alist[]
```

Scanned with CamScanner

step5: At the point where rightmark becomes less than leftmark we stop. The position of rightmark is now the split point.

step6: The pivot value can be exchanged with the contents of split point and piv is now in proper position.

step8: The quicksort function invokes a recursive function, quicksorthelper.

step9: Quick sort helper begins with same base as the merge sort.

step10: The partition function implements the process described earlier.

Scanned with CamScanner

```
for b in range(0,k)
b = int(input("enter the elements"))
    list = append
    n. len(a list)
quicksort(a list)
print(a list)
```

output:

```
Enter range for the list: 4.
Enter the element : 6
Enter the element : 8
Enter the element : 4.
Enter the element : 9
[4, 4, 6, 9].
```

Scanned with CamScanner

Practical - 5.

Aim: Implementation of stacks using python list

Theory: A stack is a linear data structure that can be represented in the real world in the form of a stack or a pile. The elements in a stack are added or removed only from one end or position. i.e, the topmost position. Thus, the stack works on the LIFO principle as the element that was inserted last will be removed first. A stack can be implemented using array as well as linked list.

Algorithm:

1. Create a stack with instance variable items.
2. Define init methods with self argument and initialize the initial value and the initialize to an empty l
3. Define methods push and pop under the class stack.
4. Use if statement to give the function that if length given list is greater than the range of list the print stack is full
5. Or else print statement as insert the element into the Stack and initialize the values.
6. Push method used to insert the element but pop is used to delete the element from the stack.

Scanned with CamScanner

abhishek

```
>>> s.push(20)
>>> s.l
[20,0,0,0,0]
>>> s.pop()
Data= 20
>>> s.l
[0,0,0,0,0]
>>> s.push(10)
>>> s.push(20)
>>> s.push(30)
>>> s.push(40)
>>> s.push(50)
>>> s.l
[10, 20, 30, 40, 50]
```

Scanned with CamScanner

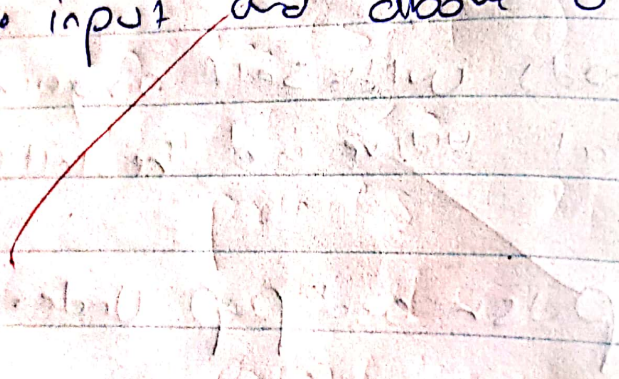
4

Push method used to insert the element by
pop method used to delete the element
stack

If in pop method, value is less than 1
return the stack is empty or else delete
the element from stack or top most
position.

First and checks whether the no. of elements
zero.

Attach the input and above algorithm



Scanned with CamScanner

```
print "abhishek"
class stack:
    gbbal tos
    def __init__(self):
        self.l = [0,0,0,0,0,0,0,0]
        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.
    def peek [self]
        if self.tos < 0:
            print ("stack empty")
            else:
                p = self.l(self.tos)
                print ("top element = ", p)
S = stack().
```

Scanned with CamScanner

Practical -6.

m: Implementing a queue using Python list

ory: Queue is a linear data structure which has 2 ferences front and rear. Implementing a queue using thon list is the simplest as the Python list provide uilt functions to perform the specified operations of queue. It specifies operations of the queue. It based on the principal that a new element is inse to rear and element of queue is deleted which is front. In simple terms, a queue can be scribed as a data structure based on first i st out principal.

ue()': Creates a new empty queue.
ueue()': Insert an element at the rear of queue, and similar to that of insert(). o ed using tail.

ueue()': Returns the element which was at t. The front is moved to successive element. ueue operation cannot remove element if t is empty.

Scanned with CamScanner

```
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()
```

Scanned with CamScanner

Algorithm:

step1: Define a class queue and assign global ... the define init() method with self arg... In init(), assign or initialize the init ... with the help of the self argument.

step2: Define a empty list and define enqueue() ... with 2 arguments assign the length of empty

step3: Use if Statement that length is equal to ... queue is full or else insert the element ... or display that queue element added successfully ... increment by 1.

step5: Now call the enqueue() 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.

Scanned with CamScanner

Output :

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

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

>> Q - ddd (3)
elements insert ... 3

>> Q. dddd (4)
queue is full

>> Q : remove
10 element deleted

Scanned with CamScanner

Practical - 7

Evaluation of postfix expression.

Aim: Program on evaluation of given string by using stack in

Theory: The postfix expression is free of any parenthesis further the logic one priorities of the operation in the program. A given postfix expression can easily be evaluated using stacks.

Algorithm:

Step1: Define evaluate as function then create an empty stack in python.

Step2: Convert the string to a list by using the string method split.

Scanned with CamScanner

```
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].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))
        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
print("abhishek")
```

Scanned with CamScanner

1.A

Perform the arithmatic operation. Push the result back on the stack.

5. After the input expression has been completely processed the result is on the stack.

Print, the result of storing after the evaluation of postfix.

Attach the output and input of the above algo.

Scanned with CamScanner

output: The evaluated value is 2.
Abhishek

Practical - 8.

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

Algorithm:

Transversing of linked list means all the nodes in the linked list is in order to perform same operation on them.

The entire linked list means can be accessed using the first nodes of the linked list the first node of the linked list in term of returned by the pointer of the linked list.

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

We should not use the head pointer to transverse

```
# code:
    class node:
        global data
        global next
        def __init__(self, item):
            self.data = item
            self.next = None
    class linked list:
        global S
        def __init__(self):
            self.S = NONE
                self.S = newnode
            else:
                head = self.S
                while head.next != None
                    head = head.next
                    head.next = newnode
        def addB(self, item):
            newnode = node(item)
            if self.S = NONE:
                self.S = newnode
            else:
                newnode.next = self.S
                self.S = newnode
    def display(self):
        head = self.S
        while head.next != None:
            print(head.data)
            head = head.next
    def delete(self):
        if self.S = NONE:
            print("List is empty")
        else:
            head = self.S
            while True:
                if head.next != None
else:
    d.next = None
```

Scanned with CamScanner

9) Similarly the rest of list of node can be traversed using while loop.

11) Attach the coding of i/p or o/p of the attached algorithm

Scanned with CamScanner

```
S= LinkedList()
S.add(50)
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
```

Scanned with CamScanner

Practical - 9.

Aim:- Implementation of merge sort.

Theory:- Like quicksort, merge sort is an divide and conquer algorithm. It divides i/p array int halves call itself for the two halves and then merges the 2 sorted halves. The merge is fin. is used for merging two halves. The merge (arr, l, m, r) is key process that assures that arr [l-m] and arr [m+1-r] are sorted and merges the two sorted sub-arrays into on once the size becomes 1, the merge process comes into action and starts merging array bcoic till the complete arrray is merged.

Applications:

1) Merge cost is useful for sorting linked lis

In-

Merge sort access data sequentially and the ne of random access is low.

Scanned with CamScanner

Code:

```
def mergesort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        lefthalf = arr[:mid]
        righthalf = arr[mid:]
        mergesort(lefthalf)
        mergesort(righthalf)
        i = j = k = 0
        while i < len(lefthalf) and j < len
            if lefthalf[i] < righthalf[j]
                arr[k] = lefthalf[i]
                i = i + 1
            else:
                arr[k] = righthalf[j]
                j = j + 1
            k = k + 1
        while i < len(righthalf):
            arr[k] = righthalf[j]
            j = j + 1
            k = k + 1
```

Scanned with CamScanner

inversion count problem

3) Used in external sorting.

Merge sort is more efficient than quick for some types of lists if the data to be sorted can only be efficiently accessed sequentially as it is thus it is popular in sequentially accessed data structure co... common.

Scanned with CamScanner

```
arr=[27,81,70,55,62,99,43,14,10]
print("random list:",arr)
mergesort(arr)
print("In MERGE SORTED LIST:",arr]
```

output:

Random list [77,81,70,55,62,99,43,14,10]

MERGESORTED [10,14,27,43,55,62,81,99]

Scanned with CamScanner

Practical-10

Aim: Implementation of sets using Python.

Algorithm:

1) Define 2 empty sets 0 and 1 and set2 now use for statement providing the range of above 2 sets.
2) Now, add() method used for addition the elements accordingly to given range then print the after addition.
3) Find the union and intersection of above 2 sets by using &(and), |(or) method. Print the sets of union and intersection of set3.
4) Use if statement to find out the subset and superset of set 3 and set 4. Display the above set.
5) Display that element in set 3, is not in set4 using mathematical operation
6) Use disjoint() to check that anyone is common element is present or not. If not then display that it is mutually exclusive

Scanned with CamScanner

```
# code:
print ("Abhishek 1843")
set1 = set()
set2 = set()
for i in range (5,15)
    set1.add (i)
for i in range (1,12)
    set2.add (i)
print ("set1:", set1)
print ("set2:", set2)
print ("\n")
set3 = set1 / set2
print ("Union of set1 and set2: Set3", set3)
set4 = set1 and set2
print ("Intersection of set1 and set2: set4", set4)
print ("\n")
if set3 > set4:
    print ("set3 is support of set4")
elif set3 < set4:
    print ("set3 is same as set4")
else:
    print ("set3 is same as set4")
if set4 < set3:
    print ("set4 is subset of set3")
    print ("\n")
set5 = set3 - set5
print ("Elements is set3 and not in set4: set5")
print ("\n")
```

Scanned with CamScanner

```
if set 4 is disjoint(set 5)
print("set 4 and set 5 are returning
                                    exclusive \n")
set 5. clear()
print("after applying clear set 5 is empty
                    set:")
print("set 5", set 5)
```

output.

>>> ~~so~~ Abhishek 1&43

set 1 : { 8, 9, 10, 11, 12, 13, 14}

set 2 : { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

Union of set 1 and set 2 : sets {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}

Intersection of set 1 and set 2 : set 4 {8, 9, 10, 11}

set 3 is suppor of set 4

set 4 is sub set of set 3

Scanned with CamScanner

Practical - 11

Aim: Program based on binary search by implementing in order, preorder and postorder traversal method.

Theory:

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 such that if it is ordered such that one child is identified as left child and other as right child.

Order: i) Transverse the left subtree, the left subtree itself might have left and right subtrees.

Visit the root node

Traverse the left subtree. The left subtree itself have left and right subtree.

Transverse the right subtree repeat it.

Post order: i) Transverse the left subtree. The left subtree itself might have left and right subtrees.

Traverse the right subtree

and the root node.

Scanned with CamScanner

```
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(new node.data, "added
                                on right of",
                                h.data)
                        break
```

Scanned with CamScanner

Algorithm

Define class node and define . init () method with 2 argument. Initialize the value is this method

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

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

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 left side

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

if statement within that else statement checking that node is greater than new root then put it into right side.

Scanned with CamScanner

```
def preorder (self, start):
    if start != None:
        print (start.data)
        self.preorder (start.l)
        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(10)
T = add(85)
T = add(10)
T = add(78)
T = add(60)
= add(8)
=add(8)
=add(15)
add(12)
int("preorder")
preorder(T.root)
("inorder")
inorder(T.root)
("
```

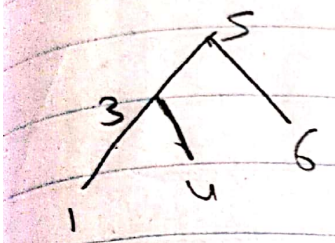
Scanned with CamScanner

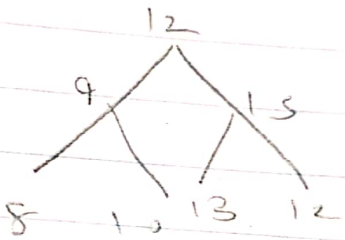
Inorder, the statement used for giving that condition first left, root and then right node.

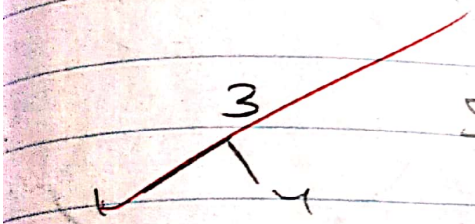
For postorder. In else part, assign left then right and then go for root node.

Display the output and input of above algorithm.

In order: (LVR).







1 3 4 5 6 7 8

Scanned with CamScanner

2.

Output

80 added on left of 100

70 added on left of 80

65 added on left of 70

10 added on left of 15

preorder

100

80

76

10

15

17

85

inorder

10

15

17

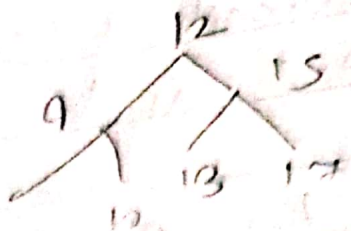
70

65

100

Scanned with CamScanner

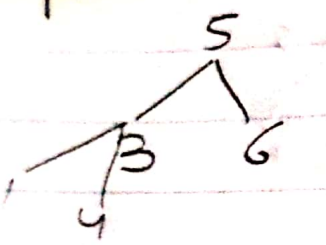
Preorder (VLR)

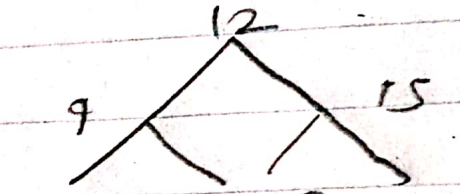


5 3 6 12 9 15 1 4 8 10 13 17

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

postorder (LRV)





3 6 3 9 15 12

Scanned with CamScanner