II	D.S.	54

THAKUR COLLEGE OF SCIENCE & COMMERCE

NAAC Accredited with Grade "A" (3rd Cycle)

THAKUR TRUSTS

ISO 9001 : 2015 Certified

Degree College

Computer Journal

CERTIFICATE

SEMESTER II UID No. _____

Class FyBSC-CS Roll No. 1859 Year 2019-2020

This is to certify that the work entered in this journal is the work of Mst. / Ms. Rishi Rohan _____

who has worked for the year 2019-2020 in the Computer Laboratory.

Teacher In-Charge

Head of Department

Date : _____

Examiner

☆☆ INDEX ☆☆

Sem II

No.	Title	Page No.	Date	Staff Member's Signature
1.	Linear Search to find on the item in the list	14	29/12	mm 2/12/19
2.	Binary search to find an searched no in the list	17	2/12	
3.	Implement of bubble sorting the given list	20	9/12	mm 9/12/19
4.	Implement quick sort to sort the given list	22	16/12	mm 16/12/19
5.	Stack push and pop	25	6/1/20	mm 06/12/19
6	Implementing queue using python	27	20/01/20	mm 20/01/2020
7	Python Libraries	29	20/01/20	

No.	Title	Page No.	Date	Staff Member's Signature
8	Implementation of single link list	31	20-1-20	[illegible] 10/02/2020
9	merge sort	34	27-1-20	[illegible]
10	Sets using python	36	3-2-20	[illegible]
11.	Binary tree	37	10-2-20	[illegible] 10/02/2020

5cm W

610

Practical no 1.

Aim: Implement liner search to find an Item in the list.

Theory:

Liner Search

Linear search is one of the simplest searching algorithm in which largest item is matched with each item in the list

It is worst searching algorithm with worst case time complexity. It is force approched. on the other hand incase of an ordered list instead of sorching the list in seq. A binary search is used which will start examing the middle item.

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

coding

```
print("linear search")
a=[]
n2 int(input("enter a range"))
for s in range(0,n):
    s = int (input("enter a no:")
    a: append (s)
    print(a)

e = int(input("enter a search no"))
for i in range (0.n):
    if (a[i] == e):
        print ("found at position")
        break
else:
    print ("not found")
```

output

```
linear search
enter a range: 5
enter a no: 4
[4]
enter a no: 5
[4,5]
enter a no: 7
[4,5,7,8]
```

```
enter a no = 8
(4,5,6,7,8)
enter a no: 9
[4,5,7,8,9]
enter search no: 8
found at position 3
```

1. Unsorted
Algorithm

Step 1: Creat an empty list and assign to a variable

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

Step 3: use the loop for adding the elements into the list

Step 4: print the new list

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

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

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

Step 8: If the element is found then print

210

Sorted:

algorithm

Step 1: create a empty list and assign to a variable

Step 2: accept the total no of a variable.

Step 3: create for loop for for adding in a element into a list.

Step 4: Sort the list and sort the accept

Step 5: use if statement to give the age. then displays "Element not found"

Step 6: Then use else statement if element is not found.

Step 7: Then use loop end range from the total no of element to be scanned before closing it accept an each no.

Step 8: Use the another loop to print the element is not found of the element which is accept

Step 9: Draw the output.

Scanned with CamScanner

```
coding
print(" linear search")
a[]
n=int[input(" enter a range :"))
for s in range (0,n):
    s=int(input(" enter a no"))]
    a: append (s)
    a: sort()
    print(a)
c = int(input(" enter a search no:"))
for i in range(0,n):
    if(a[i]==c):
        print(" found at position", i)
        break

    else:
        print(" not found")
```

output

linear search
enter a number: 5
enter a number: 4
[4]
enter a no: 8
[4,8]
enter a no: 9
[4,8,9]
enter a no: 8
[4,8,8,9]

enter a number: 6
[4,6,8,8,9]

enter a search no: 9
[found at position: 9]

practical no 2.

Aim: Implement Binary search to find an searched number in the list.

Theory: Binary search is also known as half interval search, log 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 that number which is at 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 fusion search

* Algorithm

Step 1: Create an empty list and assign

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

Step 3: Use for loop, add elements in list using append() method.

Step 4: use sort method to sort the accepted elements and assing it in increasing order list print the list after sorting.

Step 5: use if loop to gives range in which elements is found in given range then display a message "Element not found")

Step 6: Then use else statement if elements is not found in range then satisfy the bellow condition

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

Step 8: Initizal 1000 and last to last element of the list. as array is starting from 0 so it is initized 1 less than the total count

Step 9: use for loop and assing the given range.

Scanned with CamScanner

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

Output.

Binary search.

enter the range : 7

enter a num : 8

[8]

enter a num : 9

['8', '9']

enter a num : 10

['10', '8', '9']

enter a num : 11

['10', '11', '8', '9']

m

Step 10: If element in list and still the element to be searched is not found then. Find the middle element.

Step 11: Else if the item to searched is still less than the middle item then.

Initialized last (lb) = mid(m)-1

Else

Initialized first(1) = mid(m)-1

Step 12: Repeat all you find for the element still the input and output of above algorithm.

Scanned with CamScanner

practical 3.

Aim: Implementation of bubble sort program on the given list

Theory:

Bubble sort is based on the idea of repeatedly comparing pairs of adjacent elements and the swapping them 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 comparing the adjacent elements of a list.

Algorithm:

Step 1: Bubble sort algorithm start by comparing the first two elements of any two elements of 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 the element

Scanned with CamScanner

coding:

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

050

Output

>>> Bubble sort algorithm
Enter num of elements = 5
Enter the number=3
[3]
Enter the number = 8
[3, 8]
Enter the number = 5
[3, 8, 5]
Enter a number = 2
[3, 8, 5, 2]
Enter a number = 1
[3, 8, 5, 2, 1]
Elements after sorting [1, 2, 3, 5, 8]

9/12/24

Scanned with CamScanner

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

Step 4: Again second and third elements are compared and swapped if it is necessary and this process goes on until the last and second last element is compared and swapped.

Step 5: If 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: Display the output of the above algorithm of bubble sort stepwise.

150

Prac 4

Aim: Implement quick sort to sort the list.

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

Algorithm:

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

Step 2: The partion process will happen next. It will find the split point and at the same time move other items to appropriate side of the list either less than or greater than

Scanned with CamScanner

```
def quicksort(alist):
    help(alist, 0, len(alist)-1)
def help(alist, first, last):
    if first < list:
        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 l<=r and dust[l]<=pivot:
            l = l+1
    while alist[r] >= pivot and r>=l:
            r = r-1
    if r<l:
        done = true
    else:
        temp = alist[l]
        alist[i] = alist[r]
        alist[r] = temp
temp = alist[first]
```

```
        alist[first] = alist[r]
        alist[r] = temp
        return r
x = int(input("Enter range for the list"))
alist = []
for b in range(0,x):
    b = int(input("enter elements"))
    alist.append(b)
    n = len(alist)
quicksort(alist)
print(alist)
```

Output:

```
Enter range of the list: 5
enter elements : 10
enter elements: 19
enter elements : 2
enter elements: 80
enter elements: 1
[1, 2, 10, 19, 80]
```


Scanned with CamScanner

Step 3: Partition began by locating the two position markers let's call them leftmark and rightmark at the beigning and end of remaining items in the list. The goal of the partition process is to move items that are wrong with respect to pivot value while also converging on the split point.

Step 4: We began by incrimenting leftmark until be locate a value that is greater than p.v. We then decrement rightmark until we find value that is less than the pivot value. At the point we have discovered two items

Step 5: At the point when rm becomes less the leftmark. We stop. The position of rightmark is now the split point

Step 6: The pivot value can be exchanged with the content of split point and p.v (pivot value) is now in place.

Scanned with CamScanner

Step 7: In addition, all the items to left of split point are less than p.v and all the items to the right of split point are greater than pv. on the two half

Step 8: The quick sort function invokes a function of quick sort helper.

Step 9: quickSort helper, begins with same base as the merge sort.

Step 10: If the len of the list is less than its equal to 1 or it is already sorted.

Step 11: If it is greater than it can be partitioned and return sort.

Step 12: The portion function, implements the described process.

Step 13: Display and store the codings and output of above algorithm.

16/12/19

Scanned with CamScanner

Practical 5

* Aim: Implemention of stack using python list

* Theory: A Stack is a linera data Structure that can be represted in the real world in the form of the physical Stack or a pile. The elements in the Stack are added or removed only from one position. that is at the topmost position thus the Stack works on the LIFO last in first out principle at the elements that was inserted last will be removed first. A Stack can be implemented using array. as well as a list. Stack has 3 basic operations push, pop, peek. The operations of adding and removing of elements is known as push and pop:

* Algo:

1. Create a class stack with instance variable items:
2. def init method, init with self

argument and then initialized to the empty list.

3. Def methods under push and pop under the class Stack.

4. use a statement to give the condition that is len of given list is greater than the range list then print Stack is full.

5. or print Stack is to insert elements.

6. push method is used to insert elements and pop is to remove elements.

7. If the value is less than 1 then return the, Stack cannot delete the elements from stack.

8. First constant check the no of elements zero the second case where to is assigned any value if not another any value then it is sure that stack is empty.

9. assigning value elements is pass method to delete and print the given value

Scanned with CamScanner

10 attach the input and output of algo

```
print("stack")
class stack:
    global tos
    def __init__(self):
        self.l = [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("empt stack")
        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]
```

026

```
print("top
      element",p)
S = Stack()
```

Output-

>>> S.L

[0, 0, 0, 0, 0]

>>> S.push(10)

>>> S.push(20)

>>> S.push(30)

[10, 20, 30, 0, 0]

>>> S.L

[10, 20, 30, 0, 0]

>>> S.pop()

data: 30

>>> S.L

[10, 20, 0, 0]

S.peek()

20

06/11/2021

Scanned with CamScanner

Practical - 6.

r: Implementing a queue using python list.

ory: Queue is a linear data structure which has 2 ref front and rear implementing a queue using python list is the simplest as the python list provide inbuilt functions to perform the specified operation of the queue. If it is based on the principle that is a new element is inserted at rear end element of queue is deleted which is at a front of simple terms. a queue can be described as a DS based in the form of FIFO first in first out--

* Queue(): Creates a new empty queue

* Enqueue(): Insert an element which was at the front

Scanned with CamScanner

* Algo:

S1. Def a class queue and assign global variable then def init() method with self argument in init(), assing or initialise the inf value with the help of self argument.

S2. Def a empty list and def enqueue() method with two argument, assing the len of empty list.

S3. Use if statement that len is equal to zero empty list or display that queue element called and implies by 1

S4. Def dequeue() with self-argument under this use if statement that front is equal to len of list the display queue empty or else from front side and increase it by 1

S5. now call all the queue function or give adding and display the list after the element from the list.

```
class queue:
    global r
    global f
    def __init__(self)
        self.r = 0
        self.f = 0
        self.l = [0,0,0,0]
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")
        self.r = 0
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")
q = queue()
```

RSOSL

O/P

Q.add(10)
elem inserted 10
Q.add(20)
elem inserted 20
Q.add(30)
elem inserted 30
Q.add(40)
Elements inserted 40
Queue is full
Q.remove
20
Element deleted

Prac-7.1

Evaluation of a postfix Exp.

Aim: Program on Evaluation of a given String by using Stack in Python Environment i.e postfix.

theory: The postfix expression is the form of any persons in the priorities of the operators in the program. A given postfix exp can easily be evaluated using Stack. During the exp is always from left to right.

Algo:

S1. def evaluate as function the create an empty stack

S2. Convert the String to a list by using the string method "split"

S3. Calculate the len of string and print it

S4. use for loop to design the range of string then give

conclusion using of if statement.

35. 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.

36. perform the arithmetic operation - push the result back on the stack.

37. The input expression has been completely processed, the result is on the stack. pop the p and return the value.

38. print the result of string after the evaluation of postfix

39. Attached output and input of above algo.

code.


```
def evalRPN(S):
    k = S.split()
    n = len(k)
    Stack = []
    for i in range (n):
        if k[i].isdigit():
            Stack.append(int(k[i]))
        print k[i] == '+':
elif : k[i] == '+':
    a = stack.pop()
    b = stack.pop()
    Stack.append(int(b) + int(a))
    print (Stack)
elif k[i] == '-':
    a = stack.pop()
    b = stack.pop()
    Stack.append(int(b) - int(a))
    print (Stack)
elif k[i] == '*'
    a = stack.pop()
    b = stack.pop()
    stack = append(int(b) * int(b))
    print (Stack)
else:
    a = stack.pop()
    b = stack.pop()
Stack.append(int(b)/int(a))
    print (Stack)
return stack.pop()
S = "86 9 + 2 *"
```

039

x - evvuuu (S)

o/p

[8]

[8,6]

[8,6,9]

[8,15]

[4,0]

RIRO

the current value is 120

22/01/2024

Prac - 8

Aim: Implementing of Single linked list by adding the nodes from last position.

Theory: A linked list is a liner data structure which stores the element is a node in a liner fasion but not necessarily contigrous. The individual element of the link list called a node compress of 2 parts

1. Data
2. Next data stores all the information wot the element of the example, roll no name, address, etc, whereas next refers to the next node. In case of larger list, if we add/remove any element. In case of larger lists, If we add/remove any element from the list, all the elements of the list has to adjust - itself every time we add. It is very laborious task so linked is used to solve this type of problem.

Scanned with CamScanner

Algo-

S1. Traversing of a linked list means using all the nodes in the linked list in order to perform some operations on them.

S2. The entire linked list can be accessed, the first node of the linked list from is is referred by the head pointer of the list.

S3. Thus. the linked list can be traversed using the node which is referred by the head pointer of the linked list.

S4. now, that we know that we can traversal the entire linked list using the node which is referred by the pointer of the linked list

S5. we may lose the referred to the 1st node in our linked list and hence most of our linked list. So in order to avoid making some unwanted changes to the 1st node, we use a temp node the data type of temp node should be node

```
class node
    global data
    global next
    def __init__(self,item)
        self.data = item
        self.next = None
class linkedlist:
    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 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 != None:
            print(head.data)
            head = head.next
        print(head.data)
```

```
def delete (self):
    if self.s == nom
        print ("list is empty")
    else:
        head = self.s
        while True:
            if head.next != nom
                d = head
                head = head.next
            else:
                d = next = nom
                break
```

```
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

10/01/24

now that the current is reffering to the first node, if we want to access 2nd node we can refer the next node of 1st node.

- But the 1st node is refered by current so we can transfer to 2nd node as n=head.
- our concern now is to find info condition for while loop.
- The last node is the link list is refered by the data of the last node the last node of linklist does not have any node the value is the next file of the last node is null.
- we have to see now how to start traversing the linklist.
- Attached the coding or input and output of above also.

Scanned with CamScanner

Practical - 9

Topic = Merge Sort

~~algo:~~

Theory: Like Quick Sort, There is merge sort merge sort uses divide and conquer algo. It divides input array in two halfs, calls itself for the two half and then merge the two sorted half. The merge SORT is used to merge two half. The merge is a key process that assumes that and are sorted. The array is recursed divided in two half the size becomes 1 or the size becomes 1, the sorts & then. prices comes into action and starts merge sort is useful for Sorting due to it is O(nlogn) due
2, Inversion count problem.
3, used in External Sorting

The merge sort is a been used standard data Structure.

Scanned with CamScanner

```
def Sort(arr,l,m,r)
    n1 = l = m+1 + 1
    n2 = r - m
    L = [0] * (n1)
    R = [0] * (n2)
    for i in range(0,n1):
        L[i] = arr[l+i]
    for j in range(0,n2):
        R[j] = arr[m+1+j]
    i = 0
    j = 0
    k = l
    while i<n1 and j<n2
        if L[i] <= R[j]:
            arr[k] = L[i]
            i+=1
        else:
            arr[k] = R[j]
            j+=1
            k+=1
        while j<n2:
            j+=1
            j+=1
def mergesort(arr,l,r):
    if l<r:
        m = int((l+(r-1))/2)
        mergesort(arr,l,m)
        Sort(arr,l,m,r)
```

PEO

```
arr=[12,23,34,56,78,45,86,98,42]
print(arr)
n=len(arr)
mergesort(arr,0,n-1)
print(arr)
```

output

[12, 23, 34, 56, 78, 45, 86, 98, 42]

[12, 23, 56, 56, 42, 45, 78, 86, 98]

Scanned with CamScanner

merge sort is more efficient than quick sort for some types of list if the data to be sort can only be efficiently accessed sequentially and is thus popular where data structures are very common.

032

practical-10

Aim: Implementing sets using Python

Algo: def two empty sets as set1 and set2 now use for statement providing the range of above 2 sets

- Now add is method used for adding the elements according to given range. Then print the sets after adding

- Find the union and intersection of above 2 sets by using &, and() ; (or) method .print

- Display that elements is set3 is not. by using mutually operation.

- Use is disjoint() to check that anything is elements is print is not. If not then display that is mutually exclusive.

- Use clear() to remove clear the set and print the set- clearing the set.

```
print ("Name 7000 1859")
set I = set()
set II = set()
for i in range (8,15):
    set I. add(i)
for i in range(1,12):
    set II add(i)
print ("set 1:", set I)
print ("set 2:", set 2)
print ("\n")
set 3 = set 1 | set 2
print ("Union of set 1 and set 2: set 3, "set 3)
set 4 = set 1 & set 2
print ("intersection of set 1 and set 2: set 4"
                                        set 4)
print ("\n")
If set 3 > set 4:
print ("set 3 is subset of set 4")
elif set 3 < set 4:
    print ("set 3 is subset of set 4")
else:
    print ("set 3 is equal to set 4")
    print ("\n")
set 5 = set 3 - set 4
```

03/01/2022 MT

Scanned with CamScanner

```
print("Elements in set3 and not in
        set4 : set5", set5)

print("\n")

if set4.isdisjoint(set5):
    print("Set4 and set5 are
            mutually exclusive\n")
set.clear()
print("after applying clear, set5 is
                        empty :")
print("set=", set5)
```

Output

ಸಾಲು 7000 1859

set1 : {8, 9, 10, 11, 12, 13, 14}

set2: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

union of set1 and set2 : set3 {1, 2, 3, 4, 5, 6, 7, 8, 9, 10
11, 12, 13, 14}

intersection : set4 {8, 9, 10, 11}

03/01/ set3 is superset of set4

set4 is subset of set3

Elements in set3 and not in set4 : set5 : {1, 2, 3, 4, 5,

set4 and set5 is exclusive 7, 8, 9, 10, 11, 12 13}

after applying clear set5 is empty set: set5 = set()

Practical - 11

Aim: Program based on binary search tree by implementing Inorder, preorder and postorder, traversal.

Theory: Binary tree is a tree which supports maximum of 2 children for any node within the tree, thus any perticular node can have either 0 or 1 or 2 children there is another ordering of left child and order as right-child.

Inorder: i. Traversing the left subtree; the left subtree might have left and right subtree.
ii. visit the root node.
iii. Traversing the right subtree and repeat it.

Procedure: i. visit the root node
ii. Traverse the left subtree. The left subtree might have left and right subtree
iii. Traverse the right subtree

algo:

1. Def class node and def init() method with 2 arguments. Init the value in this method.

2. Again def a class BST that is binary search tree with init() method with self argument.

3. use if statement for checking the condition Root == None is none then else statement for if node

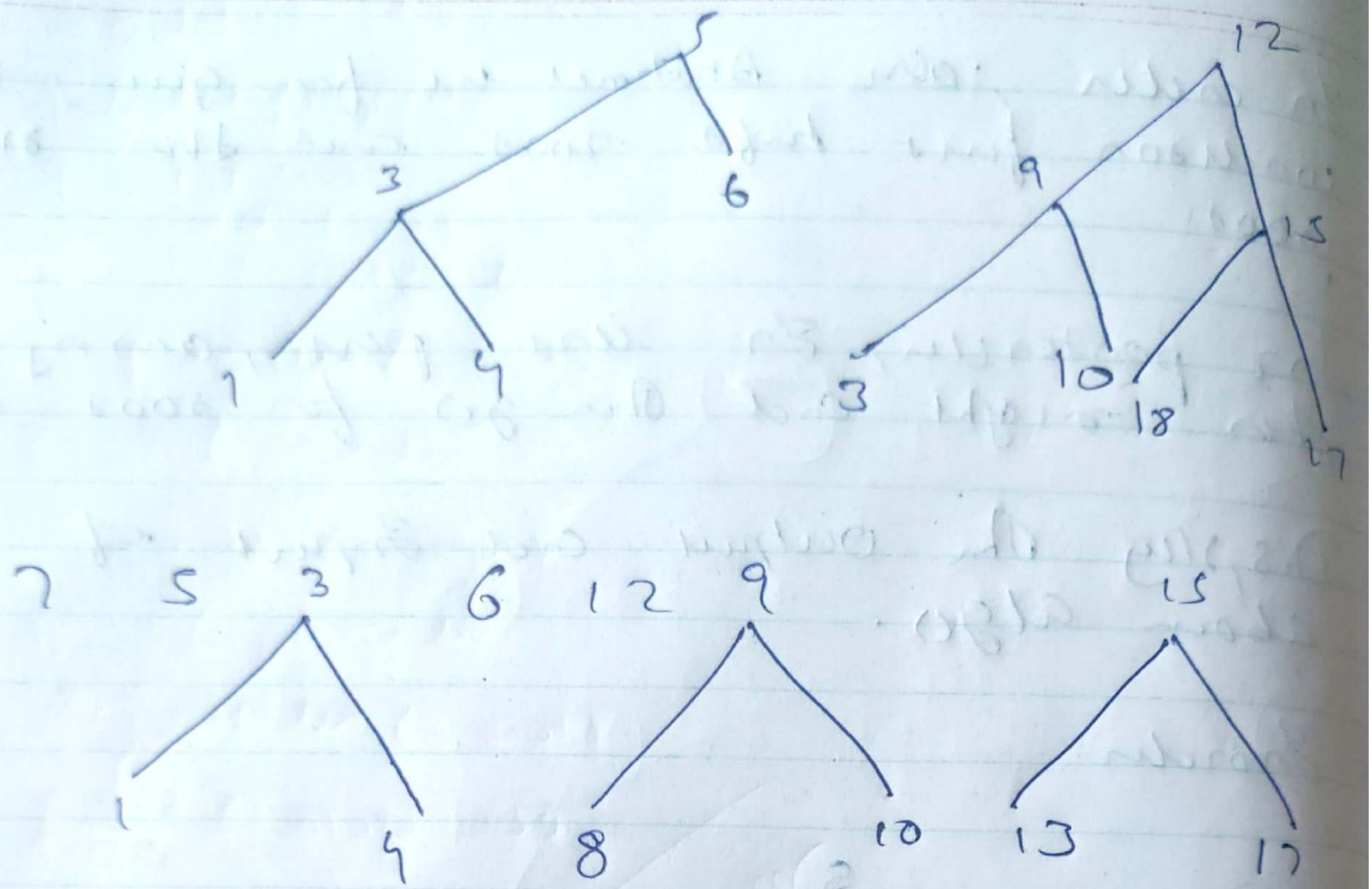
4. use while loop for checking the node less than or greater than that root then loop if it is satisfies.

5. Use if statement with in third-else statements for checking; that node is greater then move then on loop if it is

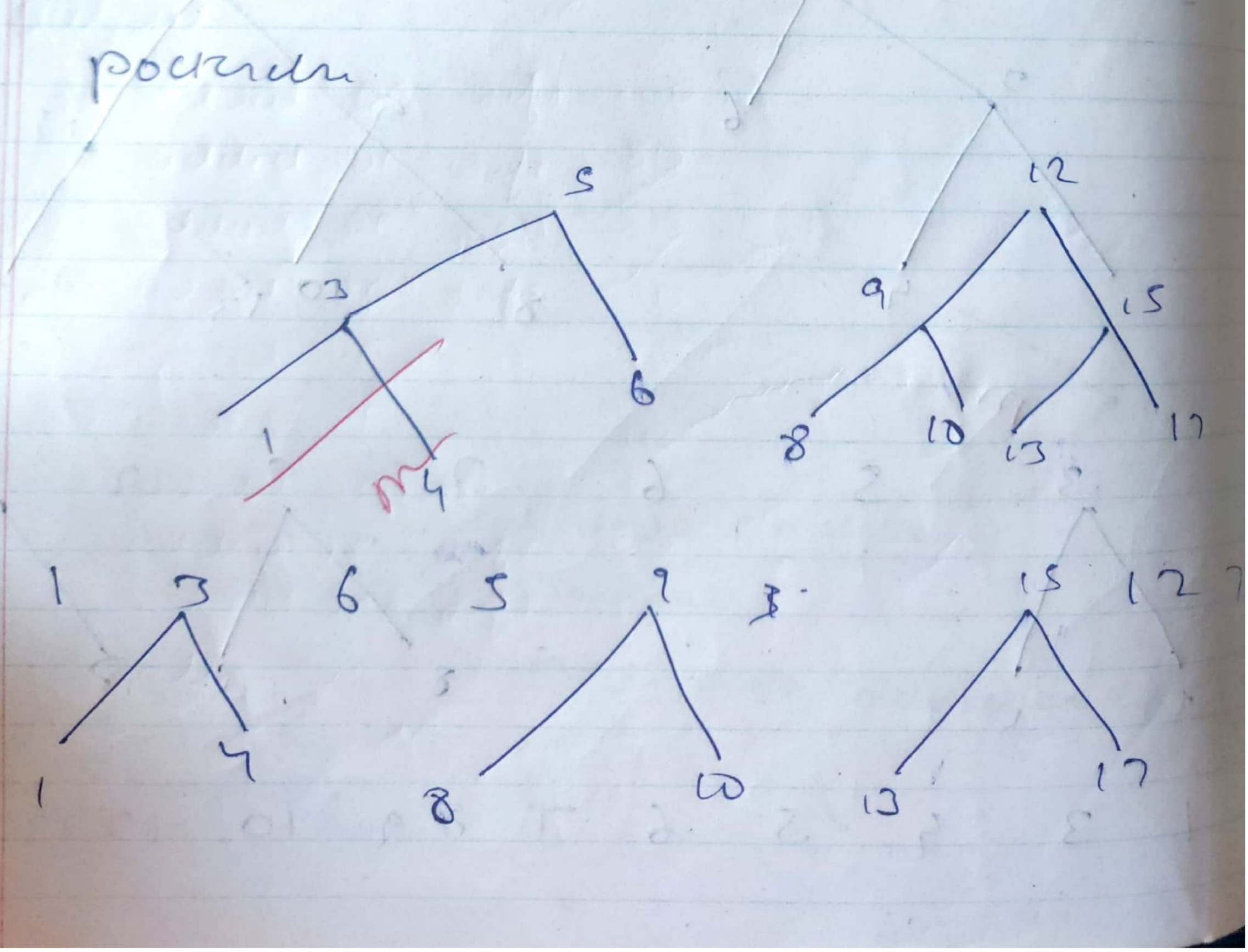
6. After this left subtree and right subtree repeat to cross the method to access the node values to build tree.

Binary tree

```
class node:
    global r
    global l
    global data
    def __init__(self,l):
        self.l = none
        self.data=l
        self.r=none
class tree:
    global root
    def __init__(self):
        self.root = none
    def add(self,val)
        if self.root=none
        self.root=nodeval
    else:
        newnode=node(val)
        h=self.root
        while true:
            if newnode.data<h.data:
                If h.l!=newnode
                print(newnode.data,"added on left of",h.d
            break
        else:
            If h.r!=none
                h=h.r
```



рассчитать



Scanned with CamScanner

```
else:
    h = r = new node
    print(new node.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(start.l)
    self.inorder(start.r)
    print(start.data)
T = Tree()
T.add(100)(7)
T.add(80)(5)
T.add(20)(12)
T.add(85)(3)
T.add(10)(6)
T.add(60)(9)
```

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

morden

7

5

3

1

4

6

12

9

8

10

15

13

17

Scanned with CamScanner

Jaarlijn

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

postcode

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

mm
10/02/2016

Scanned with CamScanner

The Shell Sort

Aim: To perform Shell Sort

Theory: The Shell sort is sometimes called "diminishing increment sort" improves the insertion by breaking the original list into a smaller number bubble list, each of this sorted are using an insertion sort The unique way that the sublist are chooses in the key to the Shell list Instead of breaking list into sub list of contiguous items the Shell sort uses an increment, sometimes called a gap to create a sublist by choosing all items that are items apart

Scanned with CamScanner

```
def Shell Sort (alist):
    Sublist count = len(alist)//2
    while sublist > 0:
        for start position in a range (sublist count
            gap Insertion Sort (alist, start posn, sublist)

    Print(" After increment of size", sublistcount)
                    " the list is ", alist)

    Sublist count = sublist //2

def gap Insertion Sort (alist ; start ; gap)
    for i in range (start + gap, len(alist)
                    , gap)

    current value = alist[i]
    position = i

    while position >= gap and alist [position
                                    -gap]

    alist [position) = alist [position - gap)
    position = position - gap

alist = [54, 26, 93, 17, 77, 31, 44, 55, 20]
Shell Sort (alist)
print (alist)
```

Output

After increment of size 4 the list is
[20, 26, 44, 17, 54, 31, 93, 55, 77]

After increment of size 2 the list is
[20, 17, 44, 26, 54, 31, 77, 55, 93]

After increment of size 1 the list is
[17, 20, 26, 31, 44, 54, 55, 77, 93]

Scanned with CamScanner