PERFORMANCE

Term	Remarks	Staff Member's Signature
I	PBMs	
II		

Exam Seat No. _____

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. 1757 Year 2019-2020

This is to certify that the work entered in this journal is the work of Mst. / Ms. Priyanka Kumari Sah

who has worked for the year 2019-20 in the Computer Laboratory.

Teacher In-Charge

Head of Department

Date : _____

Examiner

★★ INDEX ★★

No.	Title	Page No.	Date	Staff Member's Signature
1.	To Search a number from the list. Using linear Unsorted.	33-35		
2.	To Search a number from the list using linear Sorted Method.	35-37		
3.	To Search a number from the given Sorted list using Binary search.	37-38		
	Inputs		08/01/20	
4.	To demonstrate the use of stack	38-40		
5.	To demonstrate Queue add and delete.	41-42		
6.	To demonstrate the use of Circular queue in data attribute.	42-44		

★★ INDEX ★★

No.	Title	Page No.	Date	Staff Member's Signature
7.	To demonstrate the use of LinkedList in data structure.	44-46		
8.	To evaluate Postfix expression using Stack.	46-48		
9.	To Sort a given random data by using Bubble Sort.	48-49		
10.	To Sort a given random data by using Selection Sort.	50-51		
11.	To 'evaluate' i.e to sort the given data in Quick Sort.	52-54		
12.	Binary Tree & Traversal	54-56		
13.	Merge Sort	56-58		

PRACTICAL-01

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

Theory: The process of identifying or finding a particular record is called searching.
There are two types of search
(1) Linear search
(2) Binary search

The Linear search is further classified as
(1) Sorted (2) UNsorted
Here we will look on the Unsorted Linear search.
Linear search also known as sequential search, is a process that checks every element in the list sequentially until the desired element is found.
When the elements ~~is found~~ to be searched are not specifically arranged in ascending or descending in random manner That is what it calls unsorted linear search.

Unsorted linear search,
↳ The data is entered in random manner.

→ User needs to specify the element to be searched in the entered list.
→ Check the condition that whether the entered number matches if it matches then display the location plus increment I as data is sorted from Location zero.

→ If all elements are checked one by one and element not found then prompt message number not found.

ds practical 01 #.py - C:\Python34\ds practical 01 #.py (3.4.2)

File Edit Format Run Options Window Help

```
found=False
a=[3,4,56,24,8]
search=int(input("enter the number to search"))
for i in range(len(a)):
    if(search==a[i]):
        print("number found at",i)
        found=True
        break
if(found==False):
    print("number doesn't found")
```

4/2

Type here to search

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC
D64)] on win32
Type "copyright", "credits" or "license()" for more information
>>> ========== RESTART ==========
>>>
enter the number to search56
number found at 2
>>> ========== RESTART ==========
>>>
enter the number to search2
number doesn't found
>>>
```

28

PRACTICAL-2

AIM: To search a number from the list using linear sorted Method.

Theory: Searching and sorting are different modes or types of data structure.

Sorting - To basically sort the inputed data in ascending or descending manner.

Searching: To search elements and to display the same.

In searching that too in Linear sorted search. The data is arranged in ascending to descending or descending to ascending. That is all what it meant by searching through 'sorted' that is will arranged data.

Sorted Linear search

↳ The User is supposed to enter data in sorted manner.

↳ User has to given an element for searching through sorted list.

↳ If element is found display with an updation as value is sorted from location '0'.

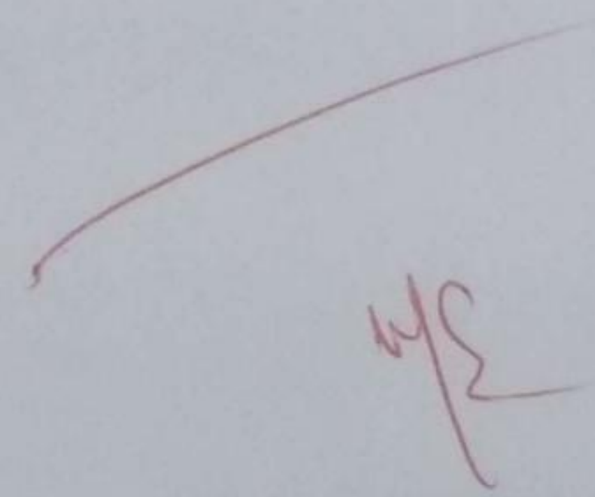
↳ If data or element not found Print the same.

↳ In sorted order list of elements we can check the condition that whether the ~~start~~ entered number lies from starting point till the last element if not then without any Processing we can say number not in the list.

ds practical 2.py - C:/Python34/ds practical 2.py (3.4.3)

File Edit Format Run Options Window Help

```
found=False
a=[45,67,69,50,3,2,54,34]
search=int(input("enter the number to search"))
if(search<a[0] or search>a[6]):
    print("number does not exist")
else:
    for i in range(len(a)):
        if(search==a[i]):
            print("number found at",i+1)
            found=True
            break
    if(found==False):
            print("number is not found")
```



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:
D64)] on win32
Type "copyright", "credits" or "license()" for more in:
>>> ========== RESTART ==========
>>>
enter the number to search54
number found at 7
>>> ========== RESTART ==========
>>>
enter the number to search1
number does not exist
>>>
```

PRACTICAL-3

AIM: To search a number from the given sorted list using binary search.

Theory: A Linear search also known as a half-interval search is an algorithm used in computer science to locate a specified value (key) within an array. For the search to be binary the array must be sorted in either ascending or descending order.
At each step of the algorithm a comparision is made and the Procedure branches into one of two directions. Specifically, The key value is compared to the middle element of the array. If the key value is less than or greater than this middle element, the algorithm known which half of the array to continue searching in because the array is sorted. This Process is repeated on Progressively smaller segments of the array until the value is carried Because each step in the algorith divides the array size in half a binary search will complete successfully in logarithmic time.

ds prac 3.py - C:\Python34\ds prac 3.py (3.4.3)

File Edit Format Run Options Window Help

```
a=[2,4,68,58,34]
print("Priyanka kumari saw")
print("Roll no 1757")
search=int(input("enter a number to be searched:"))
l=0
r=len(a)-1
while(True):
    m=(l+r)//2
    if(l>r):
        print("Number not found")
        break
    if(search==a[m]):
        print("Number is found at",m+1,"index number")
        break
    else:
        if(search<a[m]):
            r=m-1
        else:
            l=m+1
```

Type here to search

Python 3.4.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MS
win32
Type "copyright", "credits" or "license()" for more informati
>>> ========== RESTART ==========
>>>
Priyanka kumari saw
Roll no 1757
enter a number to be searched:4
Number is found at 2 index number
>>> ========== RESTART ==========
>>>
Priyanka kumari saw
Roll no 1757
enter a number to be searched:1
Number not found
>>>
```

Type here to search

Practical :- 4

Aim :- To demonstrate the use of stack

Theory: In computer science, a stack is an abstract data type that serves as a collection of elements with two principal operations push which adds on element to the collection and POP which removes the most recently added element that was not yet removed.

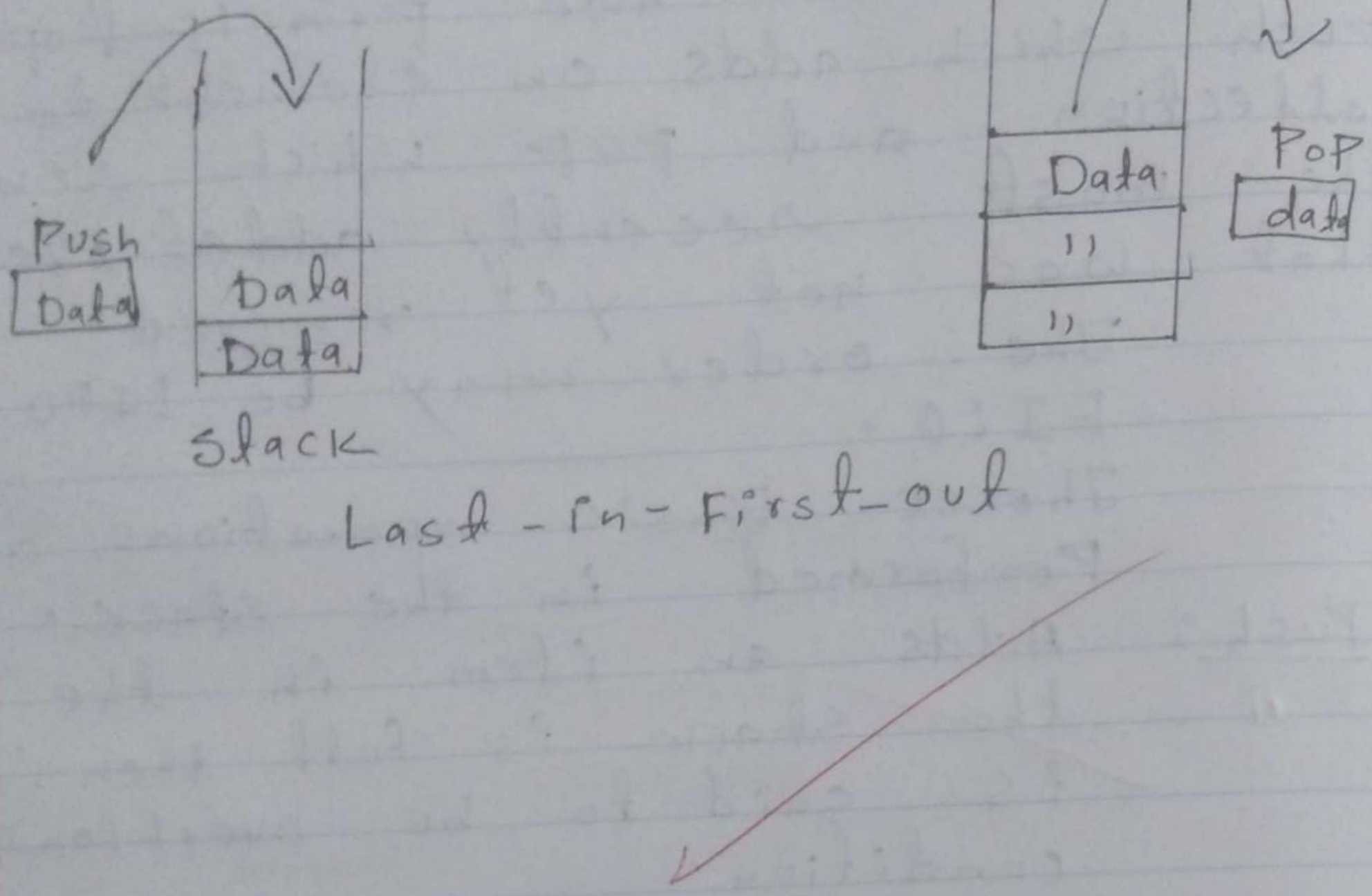
The order may be LIFO or FILO.

There Basic operations are Performed in the stack.

- Push: Adds an item in the if the stack is full then it is said to be overflow condition.

- POP: Removes an item from the stack the items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

- Peek or Top: Return top elements of stack.
- Empty: Returns true if stack is empty else False.



Last - In - First - out

practical:4

```
print("priyanka kumari saw")
class stack:
        global tos
    def __init__(self):
            self.l=[0,0,0,0,0,0,0]
            self.tos=-1
    def push(self,data):
            n=len(self.l)
            if self.tos==n-1:
                print("stack is full")
            else:
                self.tos=self.tos+1
                self.l[self.tos]=data
    def pop(self):
        if self.tos<0:
                print("stack empty")
        else:
            k=self.l[self.tos]
            print("data=",k)
            self.tos=self.tos-1
s=stack()
s.push(10)
s.push(20)
```

```
s.push(30)
s.push(40)
s.push(50)
s.push(60)
s.push(70)
s.push(80)
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
```

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AMD64)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
priyanka kumari saw
stack is full
data= 70
data= 60
data= 50
data= 40
data= 30
data= 20
data= 10
stack empty
>>>
```

Practical-05

Aim: To demonstrate queue add and delete.

Theory:

Queue is a linear data structure ~~inserted~~ where the first element is inserted from oneend called REAR and deleted from the other end called as FRONT.

Front point to the beginning of the queue and Rear points to the end of queue.

Queue follows the FIFO structure. According to its FIFO structure element inserted first will also be removed first.

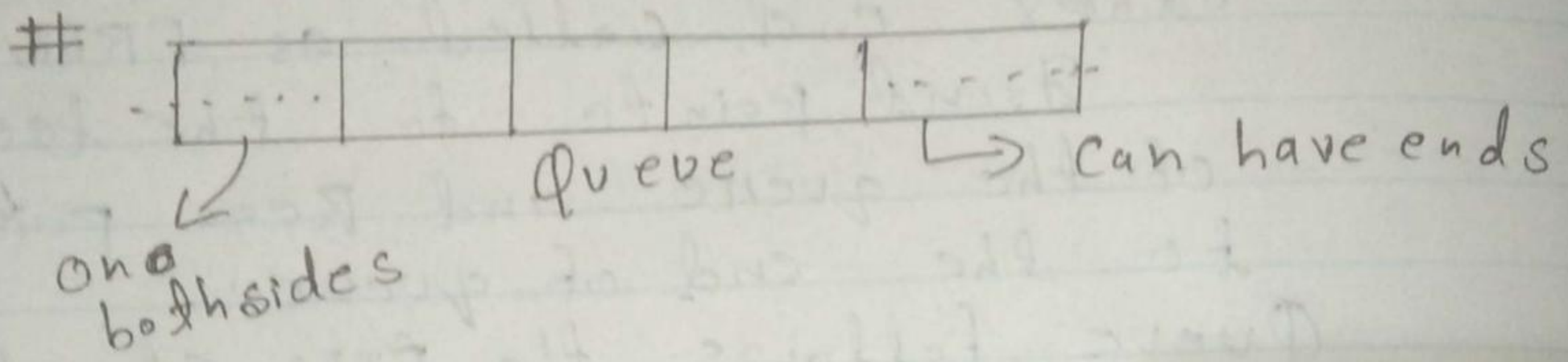
In a queue, one end is always used to insert data (enqueue) and the other is used to delete data (dequeue) because queue is open at both of its ends.

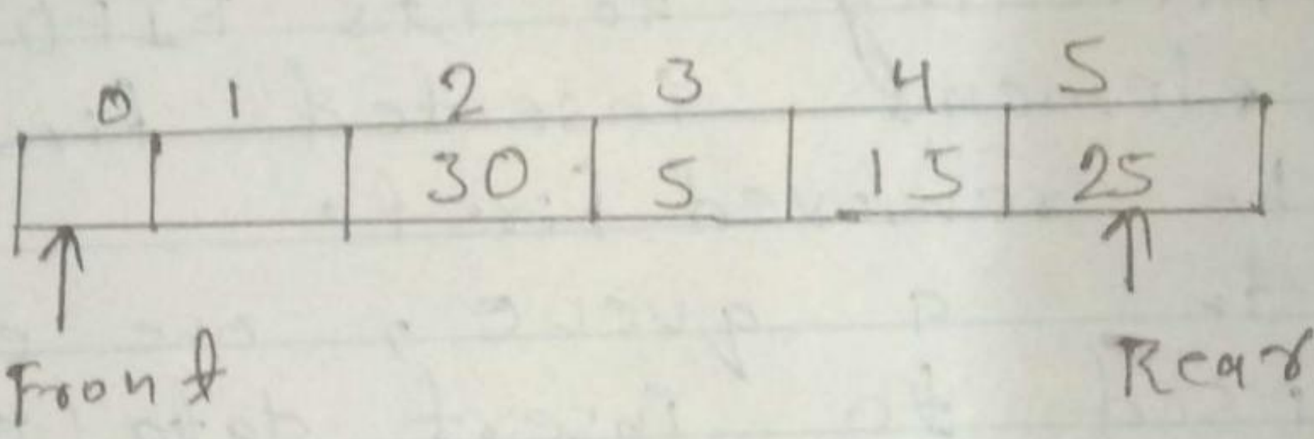
enqueue() can be termed as add() in queue i.e adding a element in queue.

Dequeue() can be termed as delete or Remove. i.e deleting or removing of element.

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

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





```
class Queue:
        global r
        global f
        def __init__(self):
                self.r=0
                self.f=0
                self.l=[0,0,0,0,0,0]
    def add(self,data):
            n=len(self.l)
            if self.r<n-1:
                    self.l[self.r]=data
                    self.r=self.r+1
            else:
                print("Queue is full")
def remove(self):
            n=len(self.l)
            if self.f<n-1:
                    print(self.l[self.f])
                    self.f=self.f+1
            else:
                print("Queue is empty")
Q=Queue()
Q.add(30)
Q.add(40)
Q.add(50)
Q.add(60)
Q.add(70)
Q.add(80)
Q.remove()
Q.remove()
Q.remove()
```

```
Q.remove()
Q.remove()
Q.remove()
```

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Queue is full
30
40
50
60
70
Queue is empty
>>>
```

Practical - 06

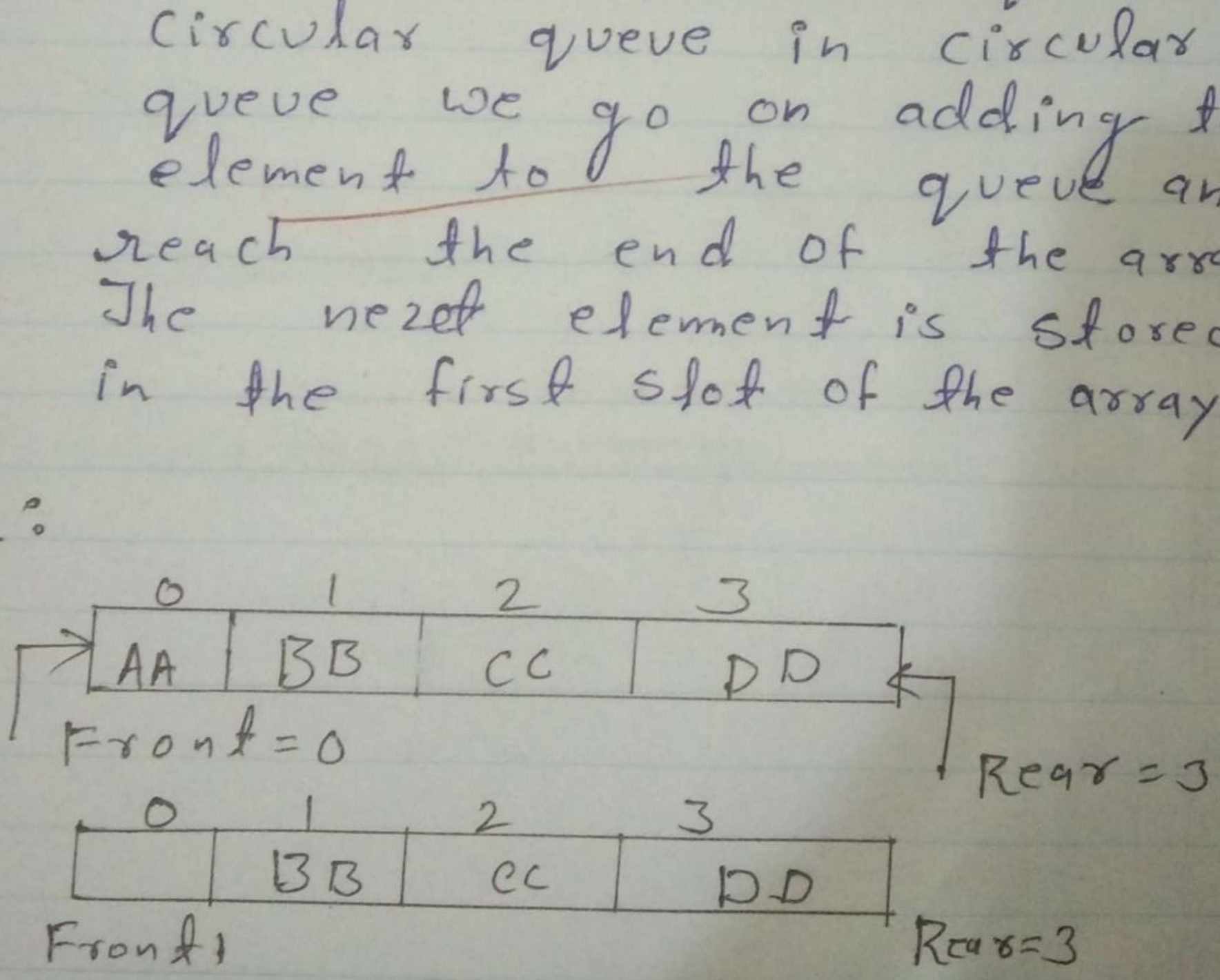
AIM:

To demonstrate the use of circular queue in data-structure

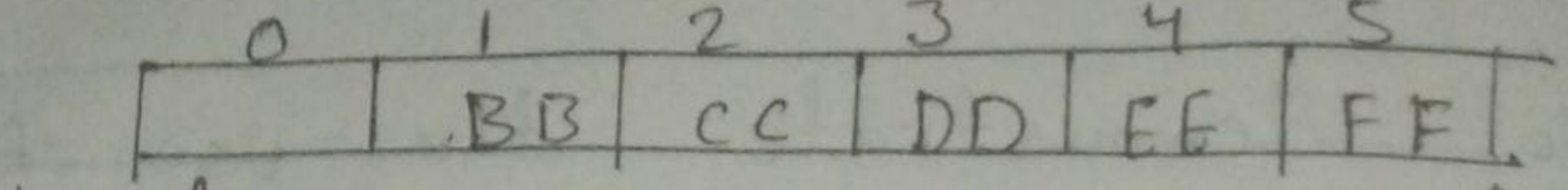
Theory:

The queue that we implement using an array suffer from one limitation. In that implementation there is a possiblity that the queue is reported as full, even though is actually there might be empty slots at the beginning of the queue. To overcome ~~this~~ this limitation we can implement queue as circular queue in circular queue we go on adding the element to the queue and reach the end of the array. The next element is stored in the first slot of the array.

Example:

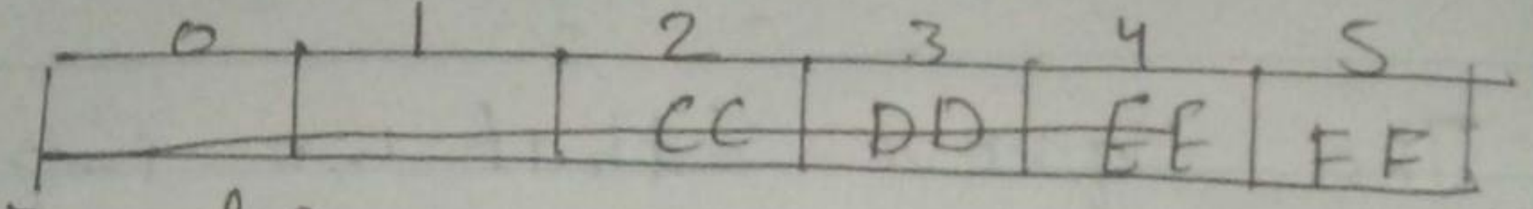


EP



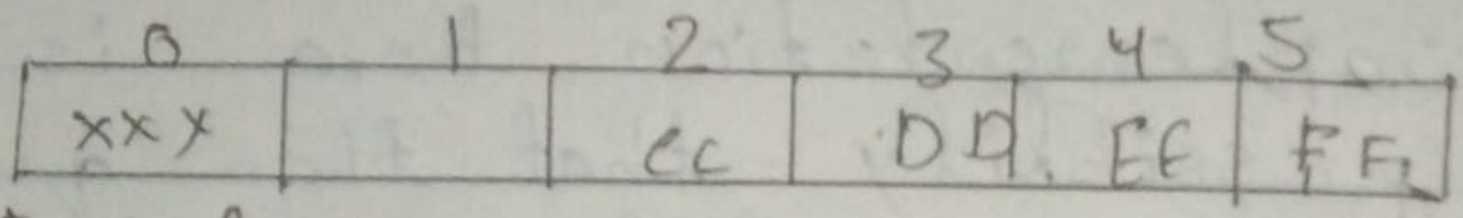
Front = 1

Rear = 5



Front 2

Rear = 5



Front = 2

Rear = 0

```
class Queue:
        global r
        global f
        def __init__(self):
                self.r=0
                self.f=0
                self.l=[0,0,0,0,0,0]
def add(self,data):
        n=len(self.l)
        if self.r<=n-1:
                self.l[self.r]=data
                print("data added:",data)
                self.r=self.r+1
        else:
                s=self.r
                self.r=0
                if self.r<self.f:
                        self.l[self.r]=data
                        self.r=self.r+1
                else:
                        self.r=s
                        print("Queue is full")
def remove(self):
        n=len(self.l)
        if self.f<=n-1:
```

```
                print("data removed :",self.l[self.f])
                self.f=self.f+1
        else:
            s=self.f
            self.f=0
            if self.f<self.r:
                    print(self.l[self.f])
                    self.f=self.f+1
            else:
                print("Queue is empty")
                self.f=s
Q=Queue()
Q.add(44)
Q.add(55)
Q.add(66)
Q.add(77)
Q.add(88)
Q.add(99)
Q.remove()
Q.add(66)
```

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
data added: 44
data added: 55
data added: 66
data added: 77
data added: 88
data added: 99
data removed : 44
>>>
```

Practical-07

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

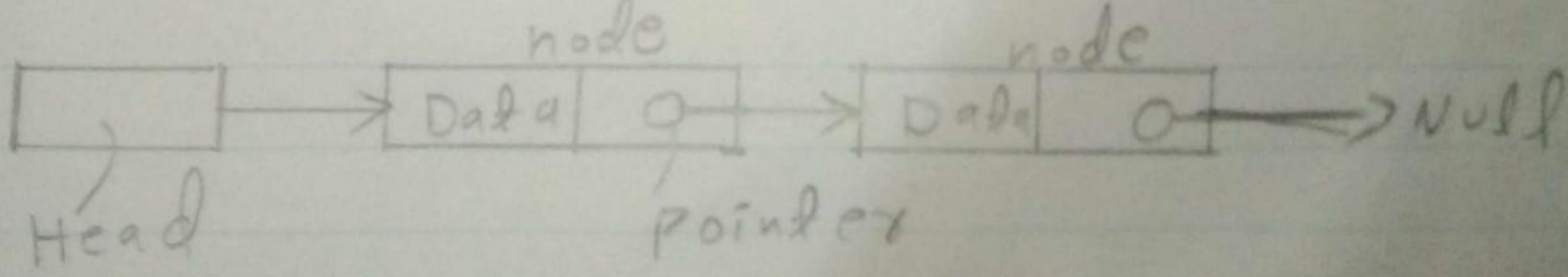
Theory: A Linked list is a sequence of data structures Linked list is a sequence of links which contains items each link contains a connection to another link.

Link: Each link of a linked list can store a data called an element.

NEXT: Each link of a linked list contains a link to the next link called NEXT.

LINKED LIST: A Linked list contains the connection link to the ~~fist~~ first link called FIRST.

Linked List Representation:



Types of Linked List:
↳ Simple
↳ Doubly
↳ Circular

Basic operations:
↳ Insertion
↳ Deletion
↳ Display
↳ Search
↳ Delete

```
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):
                head=self.s
                while head.next!=None:
                        print(head.data)
                        head=head.next
                print(head.data)
start=linkedlist()
start.addL(50)
start.addL(60)
start.addL(70)
start.addL(80)
start.addB(40)
start.addB(30)
start.addB(20)
start.display()
print("priyanka kumari saw")
print("roll no:",1757)
```

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
20
30
40
50
60
70
80
priyanka kumari saw
roll no: 1757
>>>
```

Practical-08

AIM: To evaluate postfix expression using stack.

Theory: Stack is an (ADT) and works on LIFO (Last-in First-out) i.e PUSH & POP operations.
A postfix expression is a collection of operators and operands in which the operator is placed after the operands.

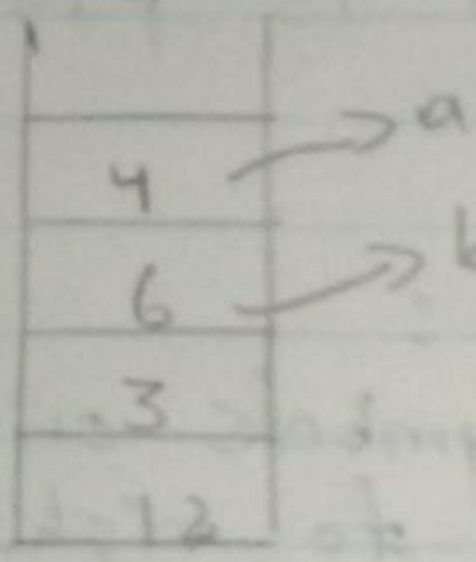
Steps to be followed:
1. Read all the symbols one by one from left to right in the given postfix expression.
2. If the reading symbol is operand then push it on to the stack.
3. If the reading symbol is operator (+, -, *, /, etc) then perform Two pop operations and store the two popped operands in two different variables (operand1 & operand2). Then perform reading symbol operation using operand1 & operand2 and push result back to the stack.

4- Finally! perform a pop operation and display the popped value as final result.

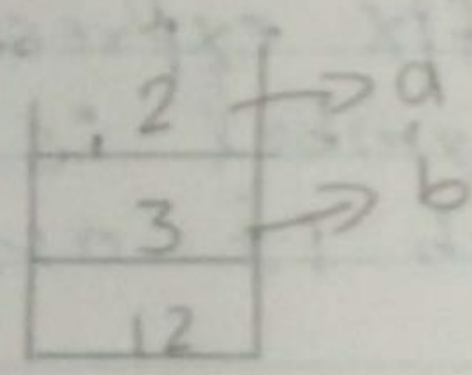
- Value of postfix expression:

S = 12 3 6 4 - + *

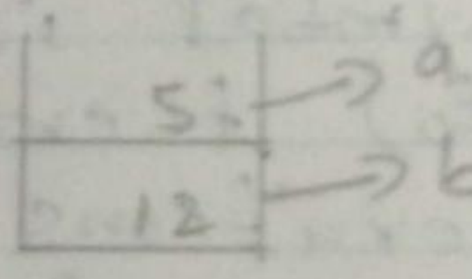
Stack:



b-a = 6-4 = 2 // store again in stack



b+a = 3+2 = 5 // store result in stack



b*a = 12*5 = 60

```
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="8 6 9 * +"
```

```
r=evaluate(s)

print("the evaluated value is:",r)

print("priyanka kumari saw")

print("roll no:",1757)
```

```
Python 3.4.3 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
the evaluated value is: 62
priyanka kumari saw
roll no: 1757
>>>
```

Practical - 09

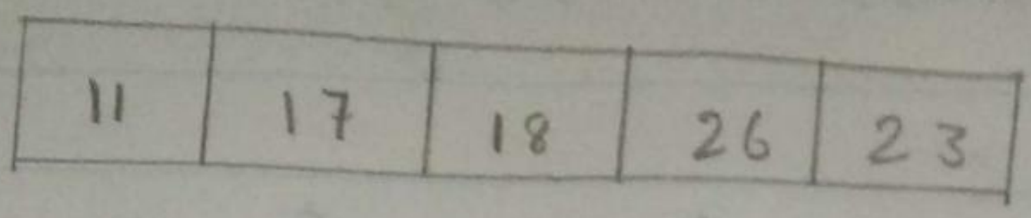
Aim: To Search a given number from the list using Bubble Sort.

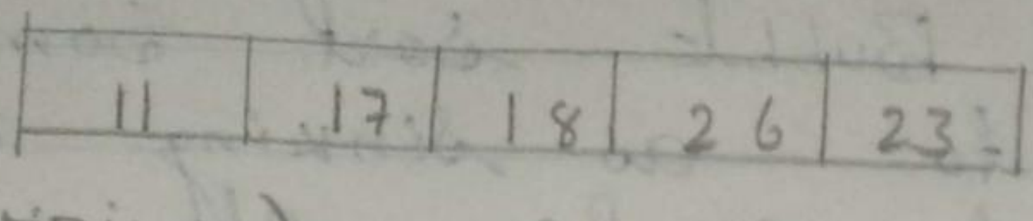
Bubble Sort:

Bubble Sort sometimes referred to as Sinking Sort, is a simple Sorting Algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The Pass through the list is repeated until the list is sorted. The algorithm, which is a comparision Sort, is named for the way smaller or larger elements "bubble" to the top of the list. Bubble sort is a Sorting algorithm that works by repeatedly stepping through lists that need to be sorted, comparing each pair of adjacent items and swapping them if they are in wrong order.

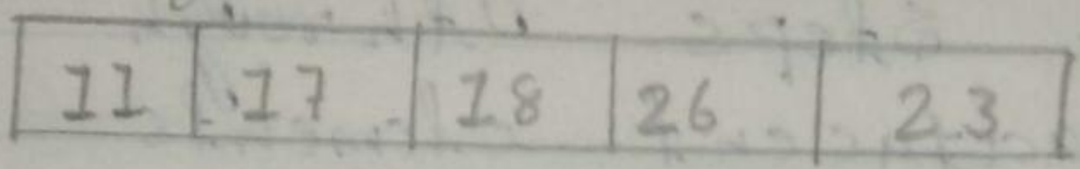
This algorithm is not suitable for ~~Example:~~ - large data sets as its average and worst case complexity are of $O(n^2)$ where n is the number of items.

Example :

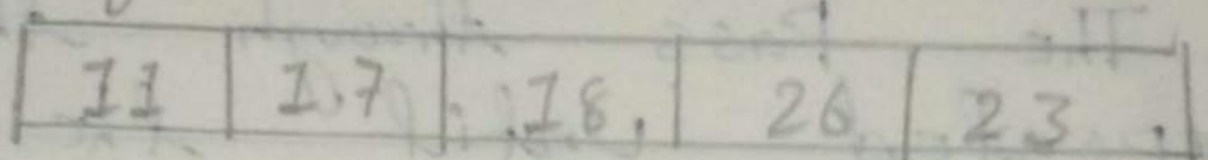


11 < 17  flag = 0

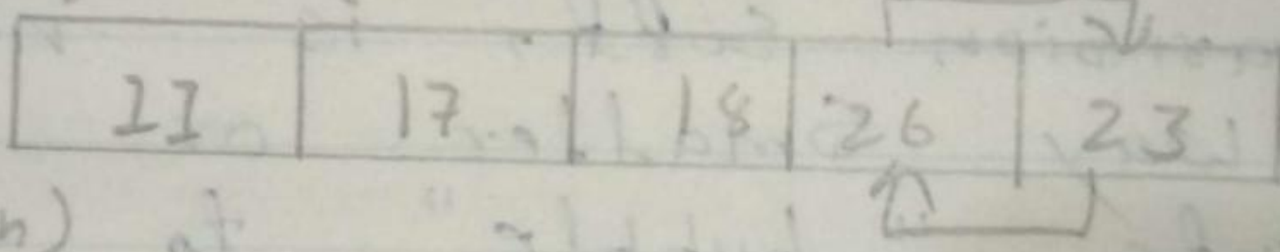
(No Swapping) (flag remains 0)

17 < 18  flag = 0

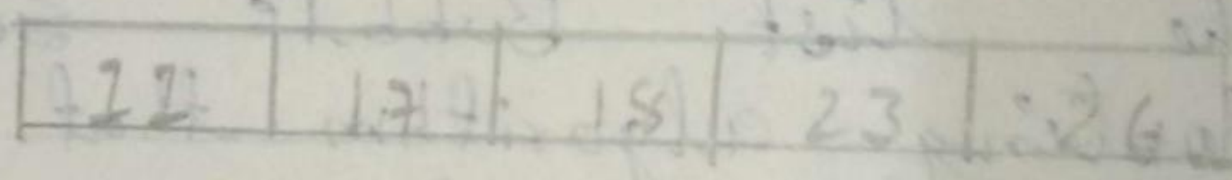
(No Swapping)

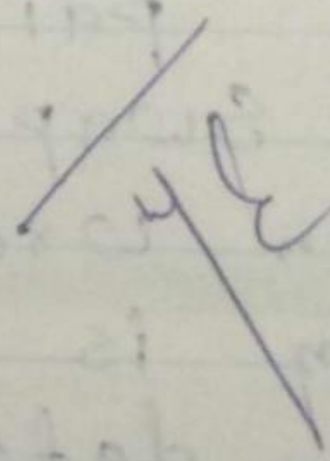
18 < 26  flag = 0

(No Swapping)

26 > 23  flag = 1

(swap then)





```
A=[1,3,5,7,9,8,6,4]
print(A)
for i in range (len(A)-1):
        for j in range (len(A)-1):
                if (A[i]>A[j+1]):
                    t=A[i]
                    A[i]=A[j+1]
                    A[j+1]=t

print(A)
print("priyanka saw ")
print("roll no.",1757)
```

Type here to search

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
[1, 3, 5, 7, 9, 8, 6, 4]
[1, 9, 8, 7, 6, 5, 3, 4]
priyanka saw
roll no. 1757
>>>
```

Type here to search

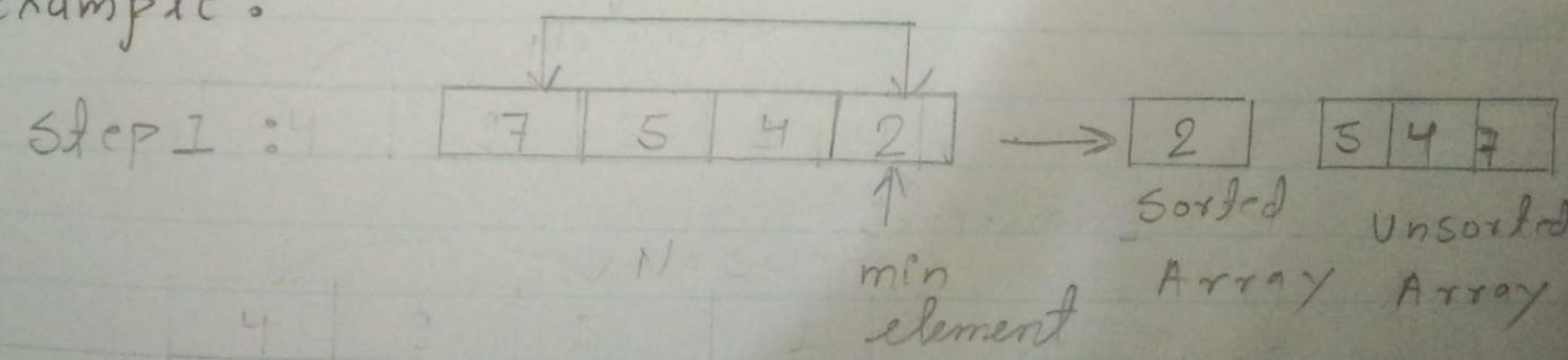
Practical 10

Aim: To search a given number from the list using Selection sort.

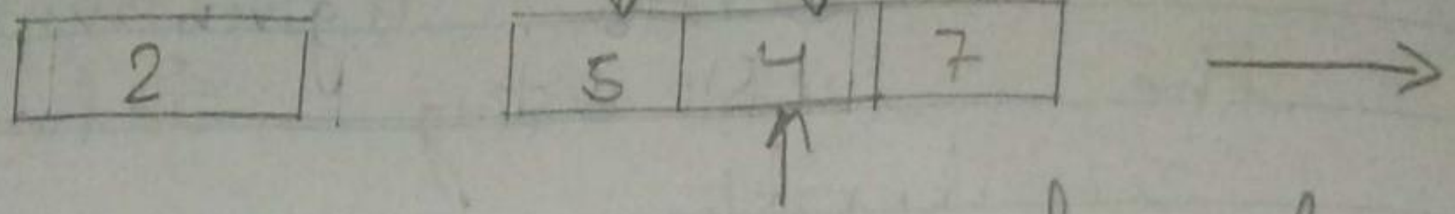
Selection Sort:

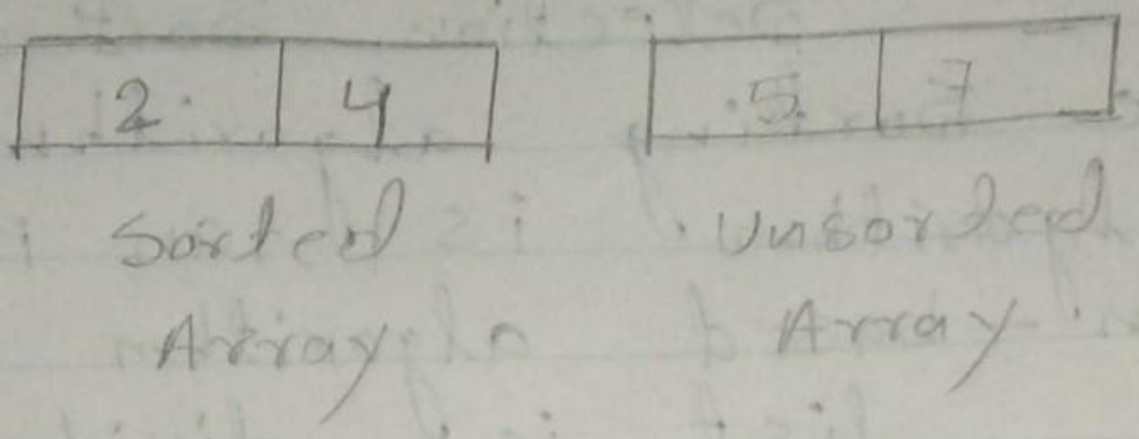
Selection Sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially the sorted part is empty and the unsorted part is the entire list. The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

Example:

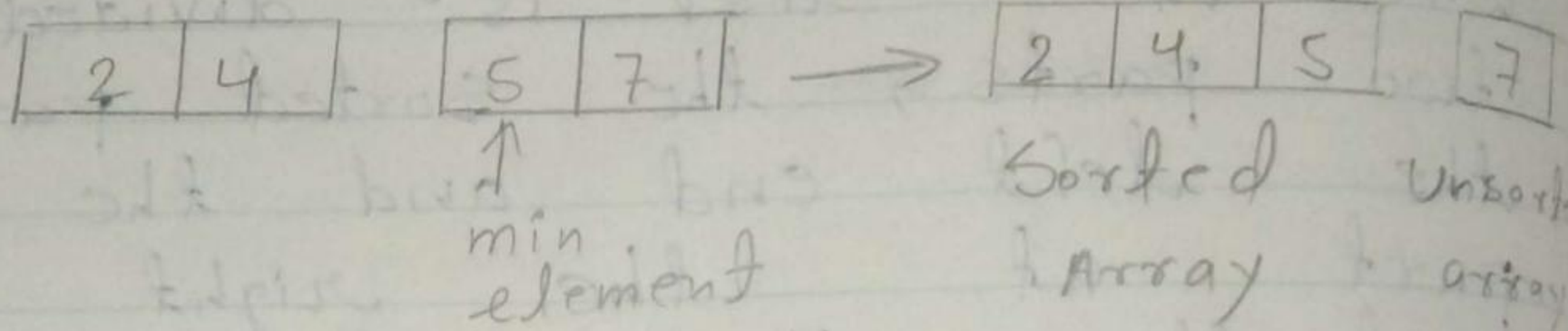


Step 2 :





Step 3 :



Step 4 :

Q7.

```
A=[4,6,8,9,3,2,1,30]
print(A)
for I in range (len(A)-1):
        for J in range (len(A)-1-I):
                if (A[J]>A[J+1]):
                    t=A[J]
                    A[J]=A[J+1]
                    A[J+1]=t
print(A)
print("priyanka saw ")
print("roll no:",1757)
```

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
[4, 6, 8, 9, 3, 2, 1, 30]
[1, 2, 3, 4, 6, 8, 9, 30]
priyanka saw
roll no: 1757
>>>
```

Practical-II

Aim: To 'evaluate' i.e to ~~star~~ sort the given data is Quick Sort.

Theory:

Quicksort is an efficient Sorting algorithm. Type of a Divide & Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quick sort that pick pivot is different ways.

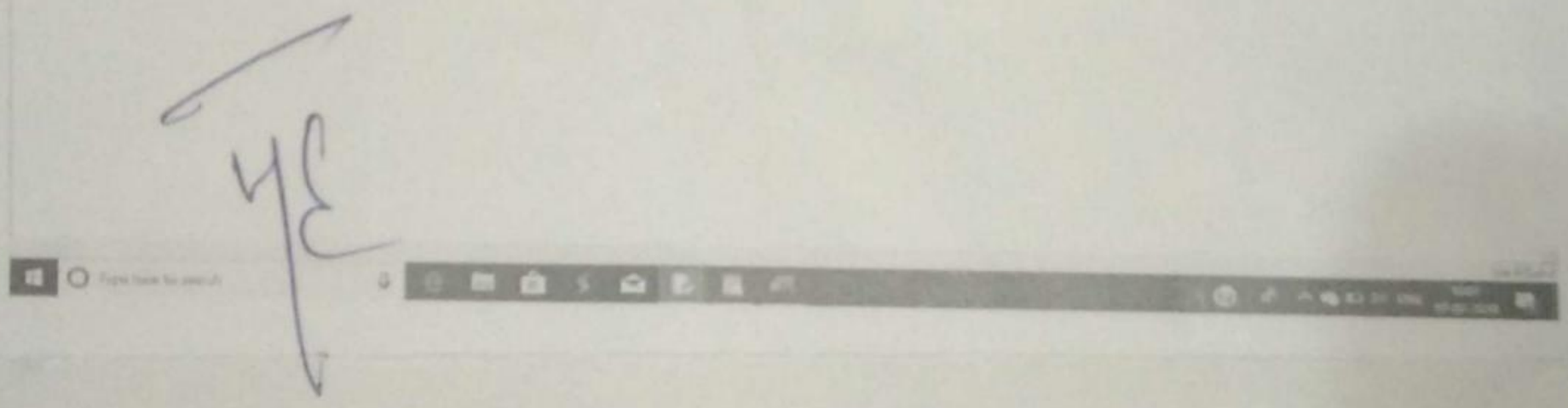
1) Always pick first element as pivot.
2) Always pick last element as pivot.
3) Pick a random element as pivot
4) Pick median as pivot.

The key process in quicksort is partition(). Target of Partitions is given an array and an element x of array as pivot, put x at its Correct position in sorted array and put all smaller elements (smaller than x) before x, & put all greater elements (greater than x) after x. all this should be done linear time.

```
def quicksort(alist):
        quicksortHelper(alist,0,len(alist)-1)
def quicksortHelper(alist,first,last):
    if first<last:
        splitpoint=partition(alist,first,last)
        quicksortHelper(alist,first,splitpoint-1)
        quicksortHelper(alist,splitpoint+1,last)
def partition(alist,first,last):
    pivotvalue=alist[first]
    leftmark=first+1
    rightmark=last
    done=False
    while not done:
        while leftmark<=rightmark and alist[leftmark]<=pivotvalue:
            leftmark=leftmark+1
        while alist[rightmark]>pivotvalue and rightmark>=leftmark:
            rightmark=rightmark-1
        if rightmark<leftmark:
            done=True
        else:
            temp=alist[leftmark]
            alist[leftmark]=alist[rightmark]
            alist[rightmark]=temp
    temp=alist[first]
    alist[first]=alist[rightmark]
```

```
                alist[rightmark]=temp
                return rightmark
alist=[42,54,45,67,89,66,57,80,100]
quicksort(alist)
print(alist)
print("priyanka kumari saw")
print("roll no:",1757)
```

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
[42, 45, 54, 57, 66, 67, 80, 89, 100]
priyanka kumari saw
roll no: 1757
>>>
```



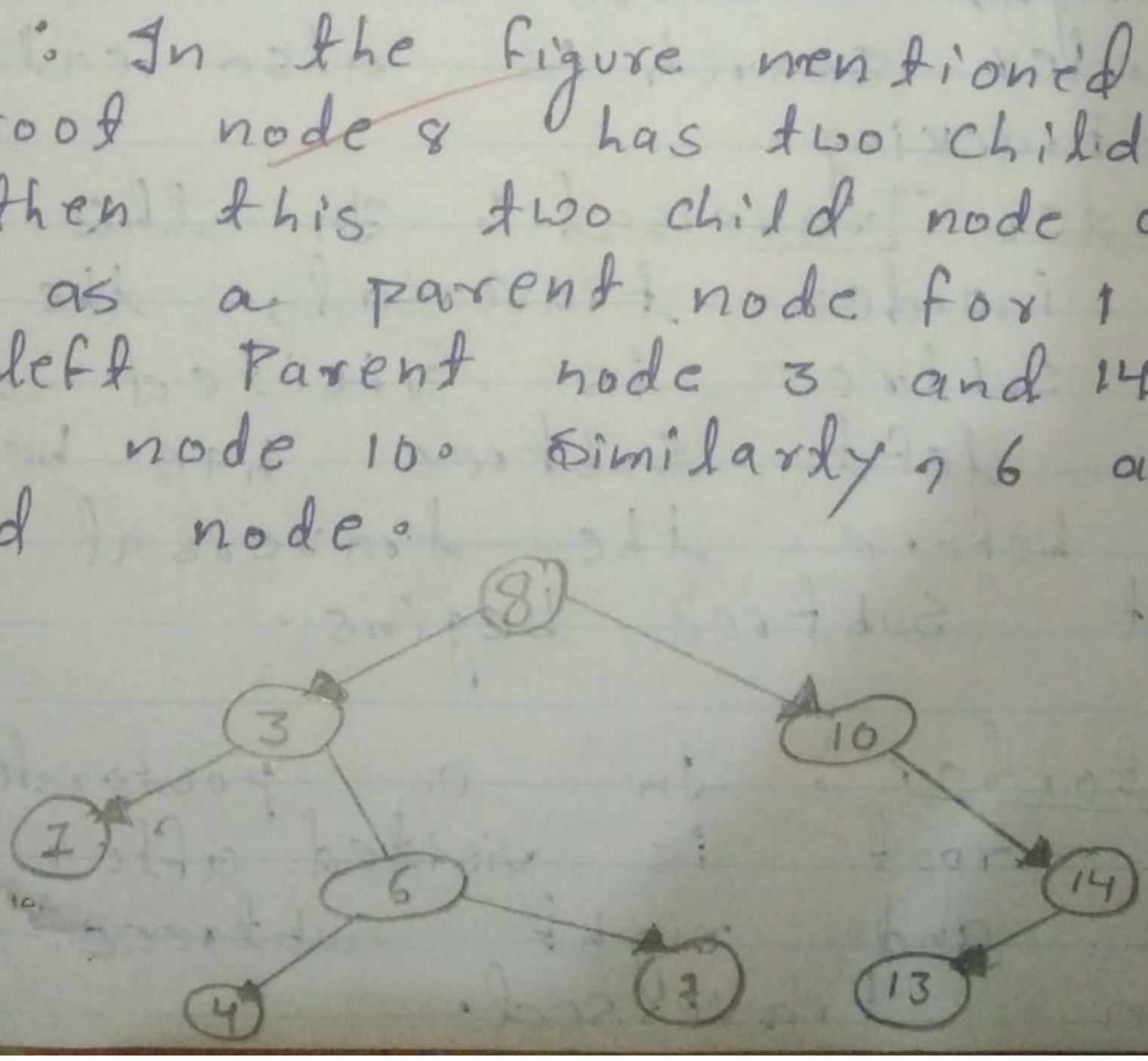
Practical-II2

Aim: Binary tree and Traversal.

Theory:

A binary tree is a special type of tree in which every node or vertex has either no child node or one child node or two child nodes. A binary tree is an important class of a tree data structure in which a node can have atmost two children. child node in a binary tree on the left is termed as 'Left child node' and node in the right is termed as the 'right child node'.

Example: In the figure mentioned below, the root node 8 has two children 3 and 10, then this two child node again acts as a parent node for 1 and 6 for left Parent node 3 and 14 for right Parent node 10. Similarly, 6 and 14 has a child node.



A tree traversal is a method of visiting every node in the tree. By visit, we mean that some type of operation is performed. For example; you may wish to print the contents of the nodes.

There are three common ways to traverse a binary tree:

1. Preorder
2. Inorder
3. Postorder

↳ Preorder Traversal: In a preorder traversal, each root node is visited before its left and right subtrees are traversed. Preorder search is also called backtracking.

↳ Inorder Traversal: In this case of inorder traversal, the root of each subtree is visited after its left subtree has been traversed but before the traversal of its right subtree begins.

↳ Postorder: In a postorder traversal each root is visited after its left and right subtrees have been traversed.

```
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=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,"added o
                                                n 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(start.l)
        self.inorder(start.r)
        print(start.data)
```

```
T=Tree()
T.add(100)
T.add(80)
T.add(70)
T.add(85)
T.add(10)
T.add(78)
T.add(60)
T.add(88)
T.add(15)
T.add(12)
print("preorder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
print("postorder")
T.postorder(T.root)
print("priyanka saw")
print("roll no:",1757)
```

```
preorder
100
80
70
10
60
15
12
78
85
88
inorder
10
12
15
60
70
78
80
85
88
100
postorder
10
```

output:

```
80 added on left of 100
70 added on left of 80
85 added on right of 80
10 added on left of 70
78 added on right of 70
60 added on right of 10
88 added on right of 85
15 added on left of 60
12 added on left of 15
```

```
80
85
88
100
priyanka saw
roll no: 1757
```

Practical-13

Aim: Merge Sort

Theory: Merge sort is a sorting technique based on divide and conquer techique. With Worst-case time complexity being O(n log n), It is one of the most respected algorithms.

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

In sorting n objects, merge sort has an average and worst-case performance of O(n log n). An example of merge sort. First divide the list into the smallest Unit (1 element), then compare each element with the adjacent list to sort and merge the two adjacent lists.

```
def sort(arr,l,m,r):
        n1=m-l+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+=l
                else:
                        arr[k]=R[j]
                        j+=1
                        k+=1
        while i<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):
        if l<r:
                m=int((l+(r-1))/2)
                mergesort(arr,l,m)
                mergesort(arr,m+1,r)
                sort(arr,l,m,r)
arr=[12,23,34,56,78,45,86,98,42]
print(arr)
n=len(arr)
mergesort(arr,0,n-1)
print(arr)
```

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:44:40) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
[12, 23, 34, 56, 78, 45, 86, 98, 42]
[12, 23, 56, 56, 42, 45, 78, 86, 98]
>>>
```