CODING (INPUT):

```
print("varun nadar \n 1702")
a=[4,21,29,64,25,24,3]
j=0
print(a)
search=int(input("enter no:"))
for i in range (len(a)):
    if (search==a[i]):
        print("num found at:",i+1)
        j=1
        break
if(j==0):
    print("not found")
```

pIcode

OUTPUT :

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ================ RESTART: C:\Users\Desktop\SEM2\DATA STRUCTURE\prac1.py ===
varun nadar
1702 21 29
[4, 21, 29, 64, 25, 24, 31]
enter no:21
num found at: 2
```

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 of that number.

If all elements are checked one by one and elements not found then display message number is not so

Practical - 2 :

Aim : To search a number in a item from a list using linear search sorted method

Theory : Linear search sorted method:

Searching and sorting are different types of data structure ing. To average data in ascending or descending order.

Searching - To search elements and to display the location of element.

In linear search sorted method the elements are arranged in ascending or descending order manner. That is all what is meant by searching there a sorted list.

Sorted linear search:

The user is supposed to enter da sorted manner

User has to given an elemen through the sorted

CODING INPUT

```
p2code

print("sarun nadar VA 1702")
a=[3,4,21,24,25,29,64]
i=0
print(a)
search=int(input("enter no:"))
if (search<a[0]) or (search>a[6]):
    print("no. doesnt exist")
else:
    for i in range (len(a)):
        if search==a[i]:
            print("num found at:",i+1)
            i=1
            break
if i==0:
    print("num not found")
```

element is found displ
an updation at value
from location 0.
data
numbers
does not EXIST.
an ordered list of elem
can check the condition tha
the entered number
starting point till the las
if not then without any
we can say number in

CODING INPUT

p3code

```
print("varun nadar \n 1702")
a=[3,4,21,24,25,29,64]
print(a)
search=input("enter no to be searched:")
l=0
h=len(a)-1
m=(l+h)/2
if ((search<=a[l])and(search>=a[h])):
    print("does not exist")
else:
    while(l!=h):
        if(search==a[m]):
            print ("number found at:",m)
            break
        else:
            if (search<a[m]):
                h=m-1
                m=(l+h)/2
            else:
                l=m+1
                m=(l+h)/2
if search==a[l]:
    print("number found at:",m)
else:
    print("not found")
```

reduced to zero

Binary search works :-

let the following array is sorted
assume that we need to search the
location of 51

low						high
9	14	27	34	51	78	83
0	1	2	3	4	5	6

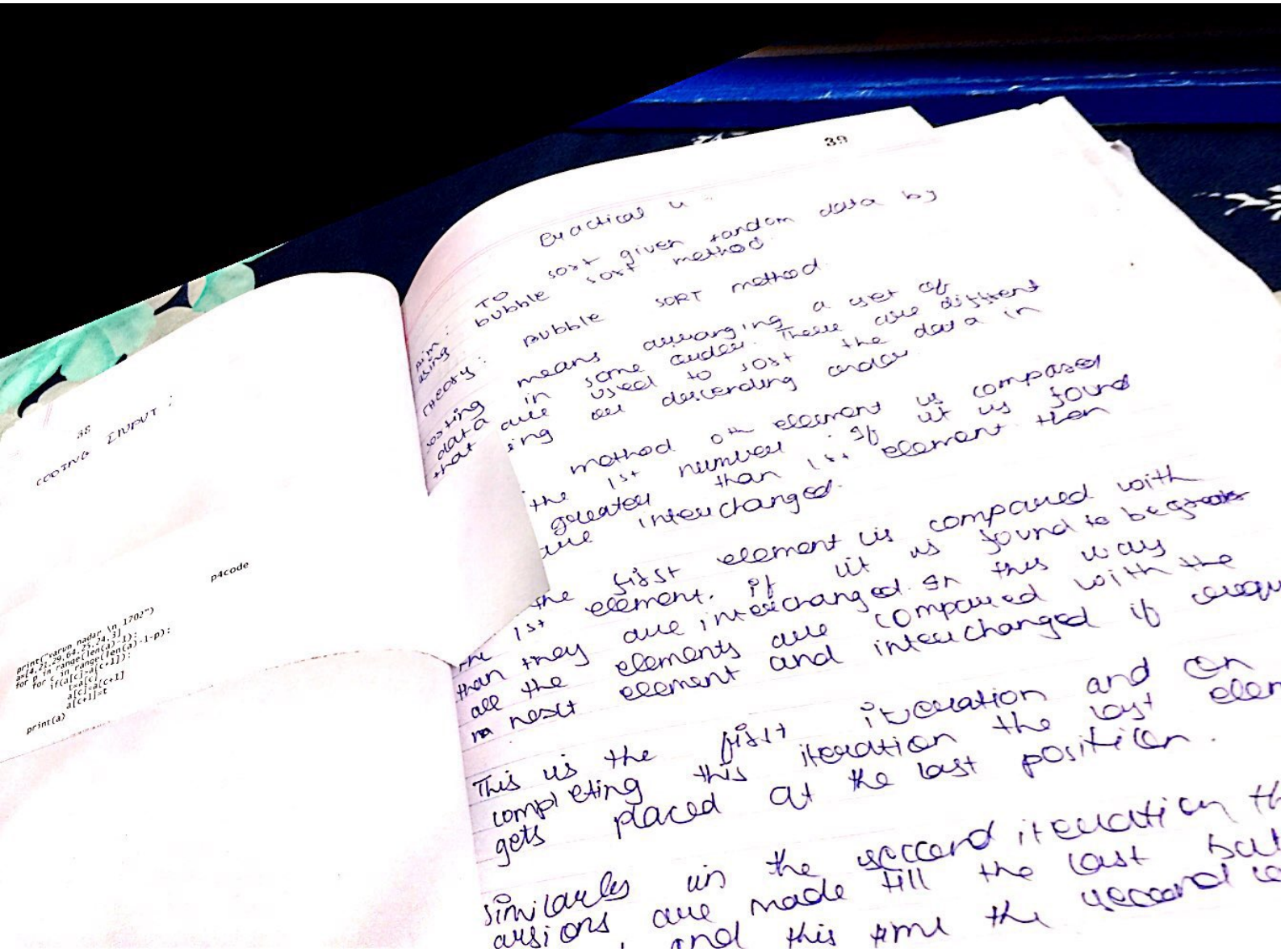
First, determine half of the array
formula

mid = low + (high - low) /2

Here, it is 0 + (6-0) /2 = 3

The mid = 3

Now, we compare the stored value
location
3 , with value being searched
we change our low to mid +1 and
the new mid value again



48

CODING ENDING :

p4code

```
print("varun nagar \n 1702")
a=[4,21,29,64,25,24,3]
for p in range(len(a)-1):
    for c in range(len(a)-1-p):
        if(a[c]>a[c+1]):
            t=a[c]
            a[c]=a[c+1]
            a[c+1]=t
print(a)
```

39

Practical 4 :

Aim : To bubble sort given random data by using bubble sort method

Theory : bubble SORT method means arranging a set of sorting in same order. These are different data are used to sort the data in that order ascending order

In the method one element is compared the 1st number if it is found greater than 1st, then value interchanged.

Then the first element is compared with the 1st element. If it is found to be greater than they are interchanged. In this way all the elements are compared with the next element and interchanged if

This is the first iteration and on completing this iteration the last element gets placed at the last position.

Similarly in the second iteration the second last comparisons are made till the last but one and this time the second

elements gets placed at the last position in list

As a result after deletion the list becomes a sorted list

coding INPUT :

p5code

```
print("varun nadar \n 1702")
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("empty stack")
    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()
```

:A

in either case is not the
but the interface is the use
allowed to pop or push
the array with few
creations.

CODING & INPUT :

P6code

```
print("saarun nadar \n 3702")
class queue:
    global f
    global r
    def __init__(self):
        self.f=0
        self.r=0
        self.data=[0,0,0,0,0,0]
    def add(self,data):
        n=len(self.data)
        if self.r<n-1:
            self.data[self.r]=data
            self.r=self.r+1
        else:
            print("queue is full")
    def remove(self):
        n=len(self.data)
        if self.f<n-1:
            print(self.f,self.data[self.f])
            self.f=self.f+1
        else:
            print("queue 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()
```


Practical 6 :

Aim : adding and deleting data in a queue

Theory : Queue

A queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities at the other end of the sequence. The end at which elements are added to the queue is called the back or rear end and the end at which elements are removed is called the head or front end.

The operation of adding an element to the rear of the queue is known as ENQUEUE

The operation of removing an element from the front is known as DEQUEUE

The operations of a queue are similar to those of a stack - it is first out (FIFO) data structure in FIFO data structure, the first element added to the queue will be the first one to be removed.

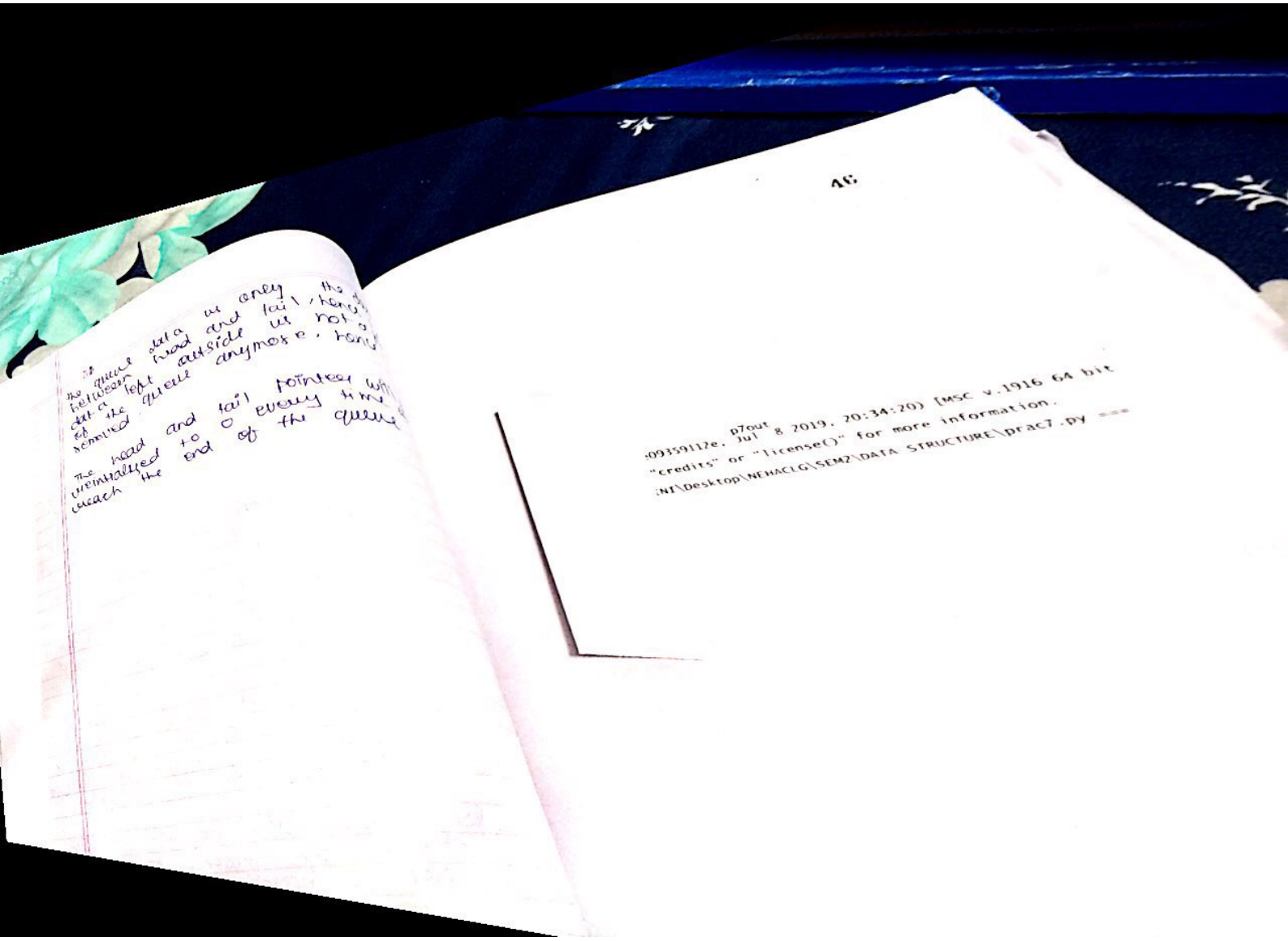
A queue is an example of a linear data structure or abstract data type or a sequential collection.

OUTPUT

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ========== RESTART: C:\Users\Desktop\SEM2\DATA STRUCTURE\prac6.py ==========
Varun Nadar
1702
queue is full
30
40
50
60
70
queue empty
>>>
```

p7code

```
print("varun nadar \n 1702")
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 empty")
                self.f=s
q=Queue()
q.add(30)
q.add(40)
q.add(50)
q.add(60)
q.add(70)
q.add(80)
q.remove()
```



p8code

```
print("varun nadar \n 1702")
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==node:
            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()
```

```
on 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
)] on win32
"help", "copyright", "credits" or "license()" for more information.
>>> %Run C:/Users/MOHINI/Desktop/NEHACLG/SEM2/DATA STRUCTURE/prac8.py
START: C:/Users/MOHINI/Desktop/NEHACLG/SEM2/DATA STRUCTURE/prac8.py ===
nadar
```

Practical 9

Evaluate an expression using Evaluation

Postfix evaluation:

Postfix notation is used to represent expression. The expression written from are evaluated later to infix notation as parenthesis are not required in postfix. We convert infix to postfix

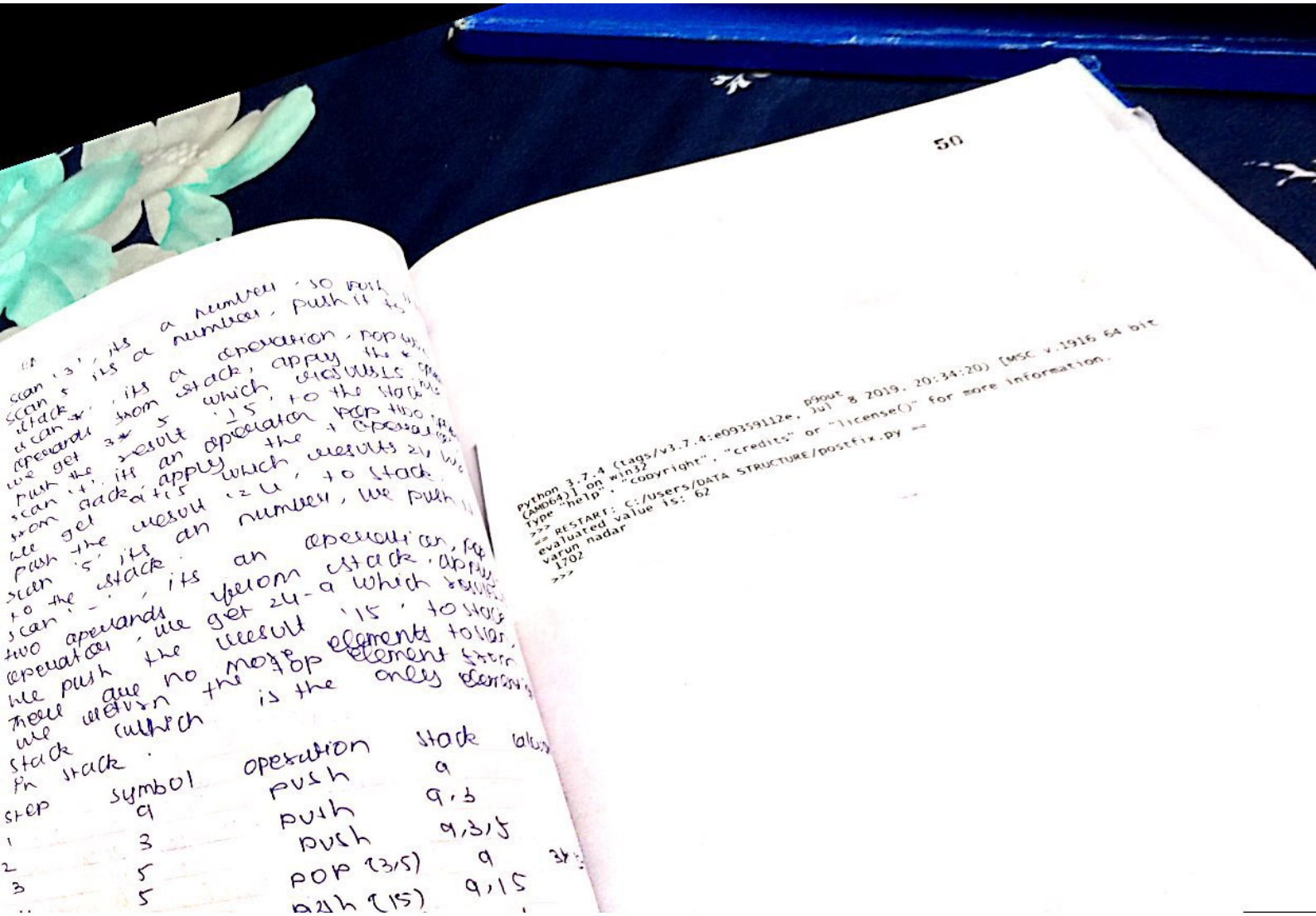
Algorithm for evaluation postfix expression

Use a stack to store operands for the given expression and do the following every scanned element
- If the element is a number, push it onto the stack
- If the element is a operator, pop two operands from the stack, evaluate the operation and push the result back to the stack

Repeat the above steps until the end of the expression is encountered. The final value in the stack is the result of the postfix expression.

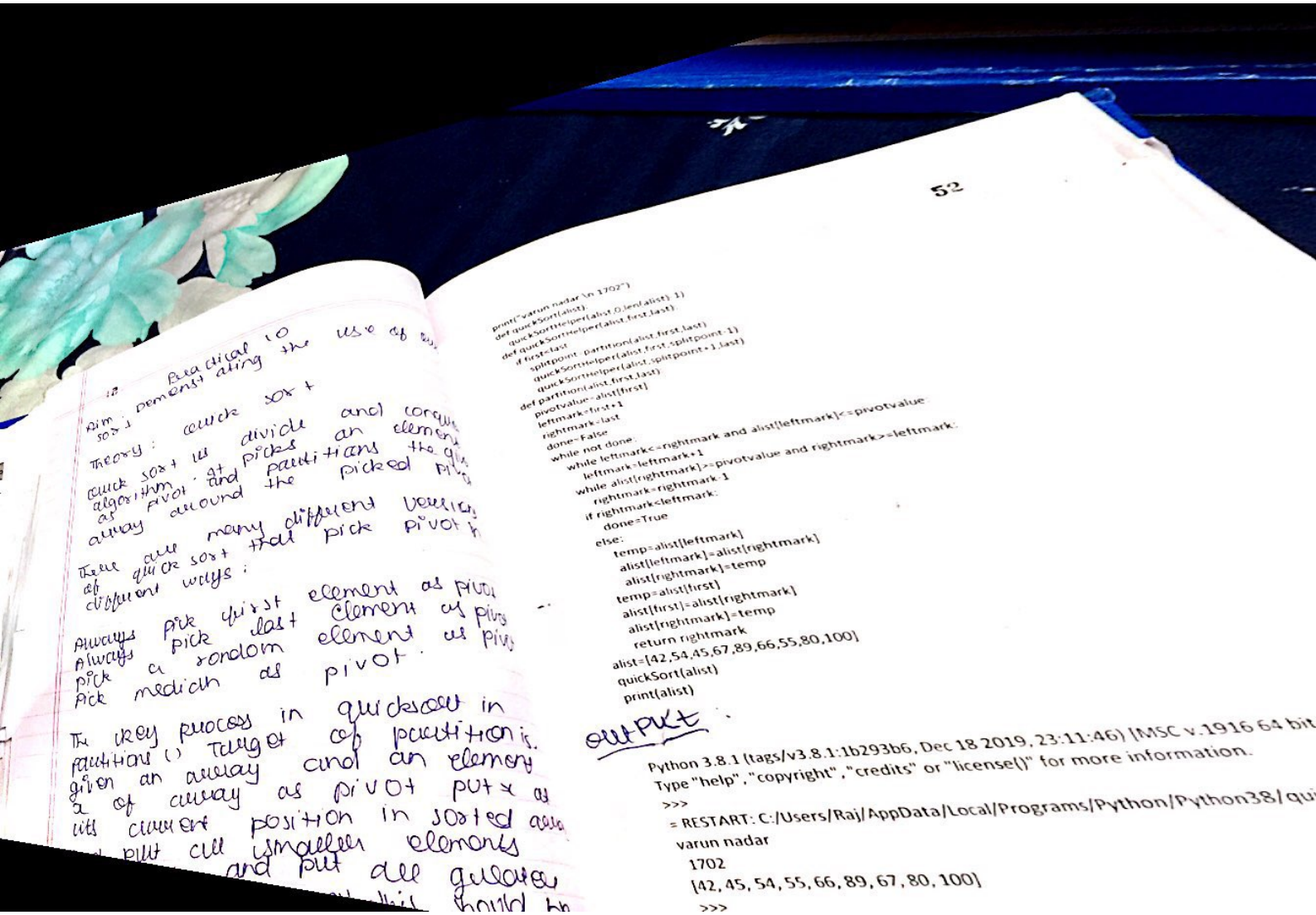
Expression: "9 3 5 * + 9 -"

'9' : its a number, so push it to stack



50

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ========== RESTART: C:/Users/DATA STRUCTURE/postfix.py ==========
evaluated value is: 62
Varun Nadar
1702
>>>
```



52

```
print("varun nadar \n 1702")
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=[42,54,45,67,89,66,55,80,100]
quickSort(alist)
print(alist)
```

output

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 bit
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Raj/AppData/Local/Programs/Python/Python38/qu
varun nadar
 1702
 [42, 45, 54, 55, 66, 89, 67, 80, 100]
 >>>
```

Analysis of quick sort:

Time taken by quicksort in general can be written as following

$$T(n) = T(k) + T(n-k-1) + \Theta(n)$$

The first two terms are for two recursive calls, the last term is for the partition process. k is the number of elements which are smaller than pivot

The time taken by quicksort depends upon the input array and partition strategy

Following are three cases

Worst case: $T(n) = T(0) + T(n-1) + \Theta(n)$
which is equivalent to
$T(n) = T(n-1) + \Theta(n)$
The above recurrence is $\Theta(n^2)$

Best case: $T(n) = 2T(n/2) + \Theta(n)$
The recurrence is $\Theta(n \log n)$

Average case: $T(n) = T(n/a) + T(an/10) + \Theta(n)$

Practical - 11

Aim: Demonstrating the use of mergesort

Theory merge sort

It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.

The merge() function is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one

Time complexity: Sorting arrays on different machines. Merge sort is a recursive algorithm and time complexity can be expressed as following recurrence relation

$T(n) = 2T(n/2) + O(n)$

Application of merge sort

```
        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=[12,11,13,5,6,7,52,47,21]
print("Before Mergesort\n",arr)
n=len(arr)
mergesort(arr,0,n-1)
print("After Mergesort\n",arr)
```

Aim: Practical 12 the use of binary tree

Theory: Binary tree:

A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.

A recursive definition using just set theory can be that a non-empty binary tree is a tuple [L, S, R], where L and R are binary trees or the empty set and S is a single node.

Types of binary trees.

Full Binary tree: All levels are filled except possibly the last one and all nodes are filled in as far left as possible.

Complete binary tree: All levels are present possibly the last one filled in as

```
print('Varun Nadar/n1702')
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 on left of",h.data)
                        break
                else:
                    if h.r!=None:
                        h=h.r
                    else:
                        h.r=newnode
                        print(newnode.data,"added on right of",h.data)
                        break
    def preorder(self,start):
        if start!=None:
            print(start.data)
            self.preorder(start.l)
            self.preorder(start.r)
    def inorder(self,start):
        if start!=None:
            self.inorder(start.l)
            print(start.data)
            self.inorder(start.r)
    def postorder(self,start):
        if start!=None:
            self.inorder(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)
```

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ========== RESTART: C:/Users/Raj/Desktop/vrn.py ==========
Varun Nadar/n1702
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
preorder
100
80
70
10
60
15
12
78
85
88
inorder
10
12
15
60
70
78
80
85
88
100
postorder
10
12
15
60
70
78
80
85
88
100
>>>
```