

**MALAD KANDIVALI EDUCATION SOCIETY'S
NAGINDAS KHANDWALA COLLEGE OF COMMERCE, ARTS &
MANAGEMENT STUDIES & SHANTABEN NAGINDASKHANDWALA
COLLEGE OF SCIENCE
MALAD [W], MUMBAI – 64
AUTONOMOUS INSTITUTION
(Affiliated To University Of Mumbai)
Reaccredited 'A' Grade by NAAC | ISO 9001:2015 Certified**

CERTIFICATE

Name: Mr. Meet Nilesh Dave

RollNo: 311

Programme:BScIT

Semester:III

This is certified to be a bonafide record of practical works done by the above student in the college laboratory for the course **Data Structures (Course Code: 2032UISPR)**for the partial fulfilment of Third Semester of BSc IT during the academic year 2020-21.

The journal work is the original study work that has been duly approved in the year 2020-21 by the undersigned.

External Examiner

Mr. Gangashankar Singh
(Subject-In-Charge)

DateofExamination: (CollegeStamp)

Class: S.Y. B.Sc. ITSem- III

Roll No:311

Subject: Data Structures**INDEX**

Sr No	Date	Topic	Sign
1	04/09/2020	Implement the following for Array: a) Write a program to store the elements in 1-D array and provide an option to perform the operations like searching, sorting, merging, reversing the elements. b) Write a program to perform the Matrix addition, Multiplication and TransposeOperation.	
2	11/09/2020	Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate two linked lists.	
3	18/09/2020	Implement the following for Stack: a) Perform Stack operations using Array implementation.b. b) Implement Tower ofHanoi. c) WAP to scan a polynomial using linked list and add twopolynomials. d) WAP to calculate factorial and to compute the factors of a givenno. (i) using recursion, (ii) using iteration	
4	25/09/2020	Perform Queues operations using Circular Array implementation.	
5	01/10/2020	Write a program to search an element from a list. Give user the option to perform Linear or Binary search.	
6	09/10/2020	WAP to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.	
7	16/10/2020	Implement the following for Hashing: a) Write a program to implement the collisiontechnique. b) Write a program to implement the concept of linearprobing.	
8	23/10/2020	Write a program for inorder, postorder and preorder traversal of tree.	

DATA STRUCTURE PRACTICALS

Practical 1

Implement the following for Array

Practical 1A

AIM: Write a program to store the elements in 1-D array and provide an option to perform the operations like searching, sorting, merging, reversing the elements.

THEORY: Storing Data in Arrays. Assigning values to an element in an array is similar to assigning values to scalar variables. Simply reference an individual element of an array using the array name and the index inside parentheses, then use the assignment operator (=) followed by a value.

Following are the basic operations supported by an array.

Traverse – print all the array elements one by one.

Insertion – Adds an element at the given index.

Deletion – Deletes an element at the given index.

Search – Searches an element using the given index or by the value.

```
practical 1.py - C:\Users\meetd\Downloads\practical 1.py (3.8.5)
File Edit Format Run Options Window Help
def linear_search(values, search_for):
    search_at = 0
    search_res = False
    while search_at < len(values) and search_res is False:
        if values[search_at] == search_for:
            search_res = True
            print("Element found at position", search_at)
        else:
            search_at = search_at + 1
    return search_res
list = [12, 45, 57, 60, 89, 55]
print(linear_search(list, 55))
print(linear_search(list, 89))
list.reverse()
print(list)
list.sort()
print(list)
list.sort(reverse = True)
print(list)
```

The screenshot shows a Windows desktop environment. In the center is a Python 3.8.5 Shell window titled "Python 3.8.5 Shell". The window displays the following code and output:

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (In tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=====
RESTART: C:\Users\meetd\Downloads\practical 1 a.py =====
Element found at position 5
True
Element found at position 4
True
[55, 89, 60, 57, 45, 12]
[12, 45, 55, 57, 60, 89]
[89, 60, 57, 55, 45, 12]
pri>>> |
```

The taskbar at the bottom of the screen shows several pinned icons, including File Explorer, Microsoft Edge, OneDrive, Data Studio, Project, Mail, Office, Run SQ..., uTorrent, IDLE (P...), DS-IT L..., and a system tray icon for battery level and date/time.

PRACTICAL 1B

AIM: Write a program to perform the Matrix addition, Multiplication and Transpose Operation.

THEORY: add() – add elements of two matrices.

subtract() – subtract elements of two matrices.

divide() – divide elements of two matrices.

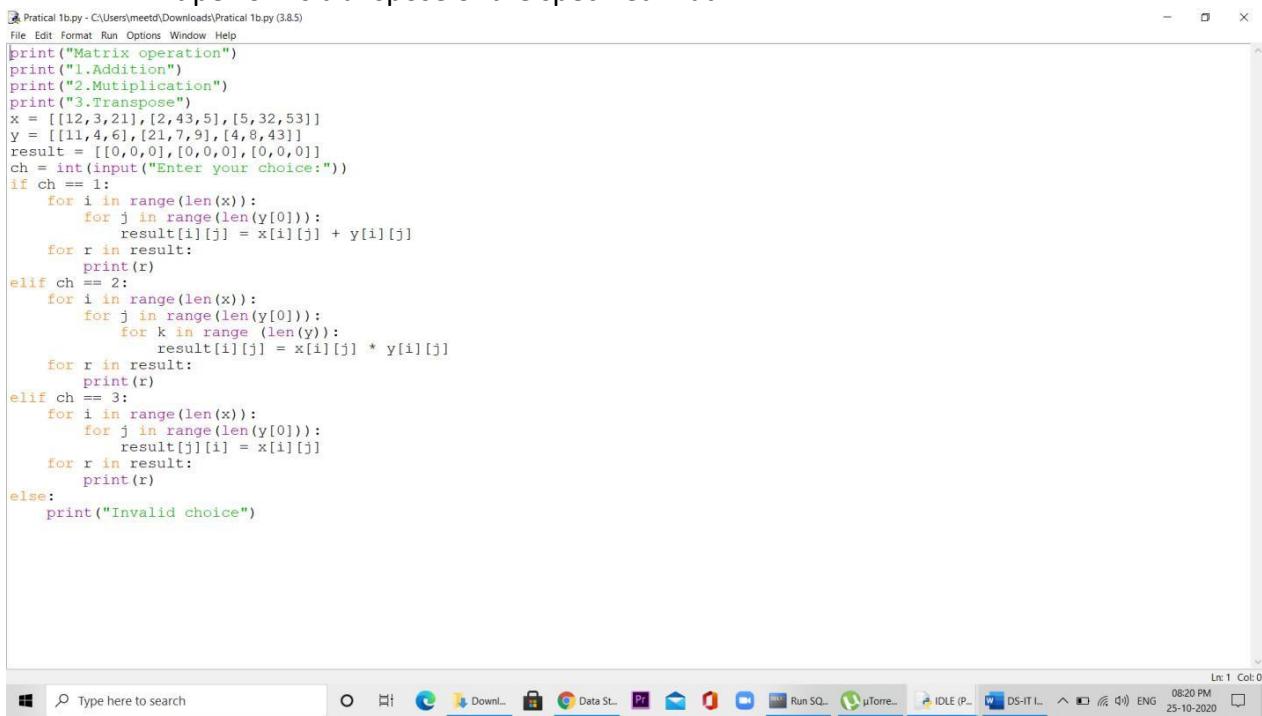
multiply() – multiply elements of two matrices.

dot() – It performs matrix multiplication, does not element wise multiplication.

sqrt() – square root of each element of matrix.

sum(x, axis) – add to all the elements in matrix. Second argument is optional, it is used when we want to compute the column sum if axis is 0 and row sum if axis is 1.

“T” – It performs transpose of the specified matrix.



```

Practical 1b.py - C:\Users\meetd\Downloads\Practical 1b.py (3.8.5)
File Edit Format Run Options Window Help
print("Matrix operation")
print("1.Addition")
print("2.Multiplication")
print("3.Transpose")
x = [[12,3,21],[2,43,5],[5,32,53]]
y = [[11,4,6],[21,7,9],[4,8,43]]
result = [[0,0,0],[0,0,0],[0,0,0]]
ch = int(input("Enter your choice:"))
if ch == 1:
    for i in range(len(x)):
        for j in range(len(y[0])):
            result[i][j] = x[i][j] + y[i][j]
    for r in result:
        print(r)
elif ch == 2:
    for i in range(len(x)):
        for j in range(len(y[0])):
            for k in range(len(y)):
                result[i][j] = x[i][j] * y[i][j]
    for r in result:
        print(r)
elif ch == 3:
    for i in range(len(x)):
        for j in range(len(y[0])):
            result[j][i] = x[i][j]
    for r in result:
        print(r)
else:
    print("Invalid choice")

```

The screenshot shows a Windows desktop environment. In the center is a terminal window titled "Python 3.8 Shell". The window displays the following text:

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=====
RESTART: C:\Users\meetd\Downloads\Practical 1b.py =====
Matrix operation
1.Addition
2.Multiplication
3.Transpose
Enter your choice:
```

The taskbar at the bottom of the screen shows several pinned icons, including File Explorer, Edge, Downloads, Data Sync, Microsoft Store, Pr, Mail, OneDrive, Run, SQ., µTorrent, IDLE (P...), DS-IT L..., and a few others. The system tray indicates the date as 25-10-2020, the time as 08:20 PM, and the language as ENG.

Practical 2

AIM: Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate two linked lists.

THEORY: A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer. Python does not have linked lists in its standard library. We implement the concept of linked lists using the concept of nodes as discussed in the previous chapter. We have already seen how we create a node class and how to traverse the elements of a node. In this chapter we are going to study the types of linked lists known as singly linked lists. In this type of data structure there is only one link between any two data elements. We create such a list and create additional methods to insert, update and remove elements from the list.

Insertion in a Linked List

Inserting element in the linked list involves reassigning the pointers from the existing nodes to the newly inserted node. Depending on whether the new data element is getting inserted at the beginning or at the middle or at the end of the linked list.

Deleting an Item form a Linked List

We can remove an existing node using the key for that node. In the below program we locate the previous node of the node which is to be deleted. Then point the next pointer of this node to the next node of the node to be deleted.

Searching in linked list

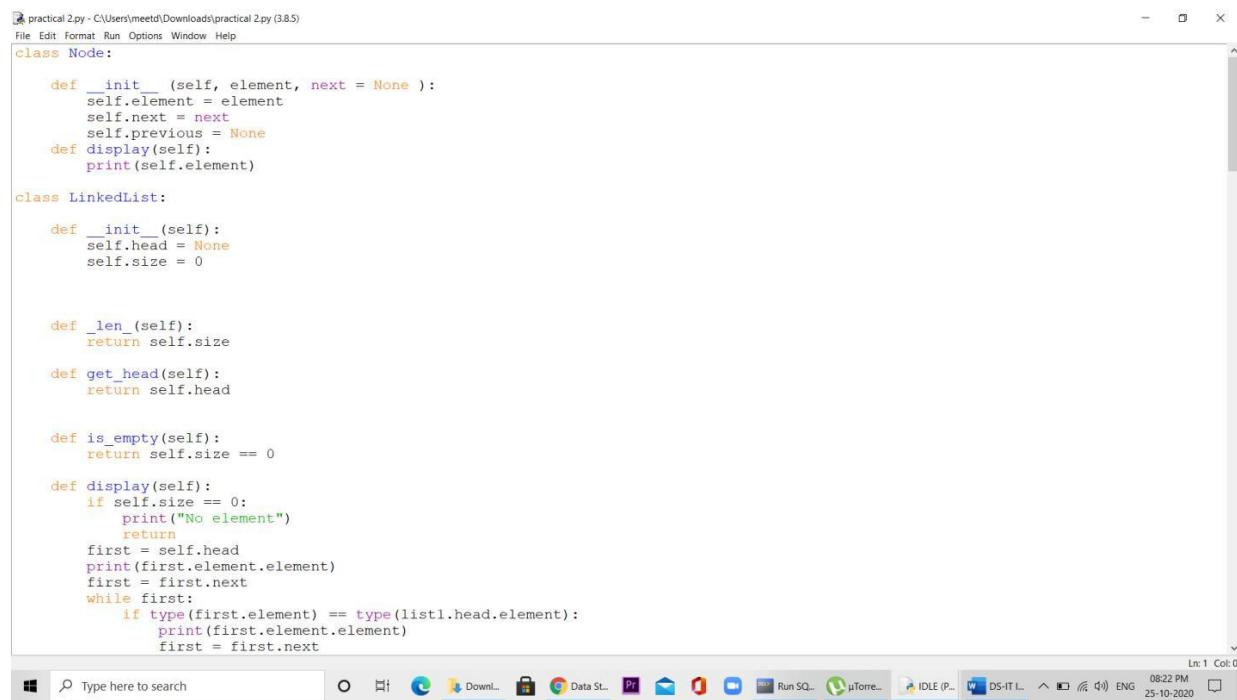
Searching is performed in order to find the location of a particular element in the list. Searching any element in the list needs traversing through the list and make the comparison of every element of the list with the specified element. If the element is matched with any of the list element then the location of the element is returned from the function.

Reversing a Linked List

To reverse a LinkedList recursively we need to divide the LinkedList into two parts: head and remaining. Head points to the first element initially. Remaining points to the next element from the head. We traverse theLinkedList recursively until the second last element.

Concatenating Linked Lists

Concatenate the two lists by traversing the first list until we reach it's a tail node and then point the next of the tail node to the head node of the second list. Store this concatenated list in the first list.



```
practical 2.py - C:\Users\meetd\Downloads\practical 2.py (3.8.5)
File Edit Format Run Options Window Help
class Node:
    def __init__(self, element, next = None):
        self.element = element
        self.next = next
        self.previous = None
    def display(self):
        print(self.element)

class LinkedList:
    def __init__(self):
        self.head = None
        self.size = 0

    def __len__(self):
        return self.size

    def get_head(self):
        return self.head

    def is_empty(self):
        return self.size == 0

    def display(self):
        if self.size == 0:
            print("No element")
            return
        first = self.head
        print(first.element.element)
        first = first.next
        while first:
            if type(first.element) == type(list1.head.element):
                print(first.element.element)
            first = first.next

Lr: 1 Col: 0
Type here to search 0 Download Data St... Run SQ... IDLE (P...) DS-IT L... 08:22 PM 25-10-2020 ENG
```

```

practical 2.py - C:\Users\meetd\Downloads\practical 2.py (3.8.5)
File Edit Format Run Options Window Help
    if type(first.element) == type(list1.head.element):
        print(first.element.element)
        first = first.next
    print(first.element)
    first = first.next

def reverse_display(self):
    if self.size == 0:
        print("No element")
        return None
    last = list1.get_tail()
    print(last.element)
    while last.previous:
        if type(last.previous.element) == type(list1.head):
            print(last.previous.element.element)
            if last.previous == self.head:
                return None
            else:
                last = last.previous
        print(last.previous.element)
    last = last.previous

def add_head(self,e):
    self.head = Node(e)
    self.size += 1

def get_tail(self):
    last_object = self.head
    while (last_object.next != None):
        last_object = last_object.next
    return last_object

def remove_head(self):
    if self.is_empty():
        print("Empty Singly linked list")

practical 2.py - C:\Users\meetd\Downloads\practical 2.py (3.8.5)
File Edit Format Run Options Window Help
def remove_head(self):
    if self.is_empty():
        print("Empty Singly linked list")
    else:
        print("Removing")
        self.head = self.head.next
        self.head.previous = None
        self.size -= 1

def add_tail(self,e):
    new_value = Node(e)
    new_value.previous = self.get_tail()
    self.get_tail().next = new_value
    self.size += 1

def find_second_last_element(self):

    if self.size >= 2:
        first = self.head
        temp_counter = self.size - 2
        while temp_counter > 0:
            first = first.next
            temp_counter -= 1
        return first

    else:
        print("Size not sufficient")

    return None

def remove_tail(self):
    if self.is_empty():
        print("Empty Singly linked list")

```

```

practical 2.py - C:\Users\meetd\Downloads\practical 2.py (3.8.5)
File Edit Format Run Options Window Help
def remove_tail(self):
    if self.is_empty():
        print("Empty Singly linked list")
    elif self.size == 1:
        self.head == None
        self.size -= 1
    else:
        Node = self.find_second_last_element()
        if Node:
            Node.next = None
            self.size -= 1

def get_node_at(self,index):
    element_node = self.head
    counter = 0
    if index == 0:
        return element_node.element
    if index > self.size-1:
        print("Index out of bound")
        return None
    while(counter < index):
        element_node = element_node.next
        counter += 1
    return element_node

def get_previous_node_at(self,index):
    if index == 0:
        print('No previous value')
        return None
    return list1.get_node_at(index).previous

def remove_between_list(self,position):
    if position > self.size-1:
        print("Index out of bound")
    elif position == self.size-1:
        self.remove_tail()
    elif position == 0:
        self.remove_head()
    else:
        self.remove_node_at(position)

def add_between_list(self,position,element):
    if position > self.size:
        print("Index out of bound")
    elif position == self.size:
        self.add_tail(element)
    elif position == 0:
        self.add_head(element)
    else:
        prev_node = self.get_node_at(position-1)
        next_node = self.get_node_at(position+1)
        prev_node.next = next_node
        next_node.previous = prev_node
        self.size += 1

def search (self,search_value):
    index = 0
    while (index < self.size):
        value = self.get_node_at(index)
        if type(value.element) == type(list1.head):
            print("Searching at " + str(index) + " and value is " + str(value.element.element))
        else:
            print("Searching at " + str(index) + " and value is " + str(value.element))
        if value.element == search_value:
            print("Found value at " + str(index) + " location")
            return True
Ln 1 Col: 0
08:23 PM 25-10-2020

practical 2.py - C:\Users\meetd\Downloads\practical 2.py (3.8.5)
File Edit Format Run Options Window Help
self.remove_tail()
elif position == 0:
    self.remove_head()
else:
    prev_node = self.get_node_at(position-1)
    next_node = self.get_node_at(position+1)
    prev_node.next = next_node
    next_node.previous = prev_node
    self.size -= 1

def add_between_list(self,position,element):
    element_node = Node(element)
    if position > self.size:
        print("Index out of bound")
    elif position == self.size:
        self.add_tail(element)
    elif position == 0:
        self.add_head(element)
    else:
        prev_node = self.get_node_at(position-1)
        current_node = self.get_node_at(position)
        prev_node.next = element_node
        element_node.previous = prev_node
        element_node.next = current_node
        current_node.previous = element_node
        self.size += 1

def search (self,search_value):
    index = 0
    while (index < self.size):
        value = self.get_node_at(index)
        if type(value.element) == type(list1.head):
            print("Searching at " + str(index) + " and value is " + str(value.element.element))
        else:
            print("Searching at " + str(index) + " and value is " + str(value.element))
        if value.element == search_value:
            print("Found value at " + str(index) + " location")
            return True
Ln 1 Col: 0
08:24 PM 25-10-2020

```

The image shows a Windows desktop environment with three main windows open:

- Code Editor Window:** The title bar reads "practical 2.py - C:\Users\meetd\Downloads\practical 2.py (3.8.5)". The content of the file is a Python script for a linked list class. It includes methods for searching, merging, and displaying the list. The script creates two lists, merges them, and then searches for the value 'element 6'.
- Python Shell Window:** The title bar reads "Python 3.8.5 Shell". The window shows the output of running the script. It prints the elements of both lists and then performs a search operation, outputting "Found value at 5 location".
- Taskbar:** At the bottom of the screen, the taskbar displays various pinned icons and the system tray.

Practical 3

Implement the following for Stack.

Practical 3A

AIM: Perform Stack operations using Array implementation.

THEORY: Stacks is one of the earliest data structures defined in computer science. In simple words, Stack is a linear collection of items. It is a collection of objects that supports fast last-in, first-out (LIFO) semantics for insertion and deletion. It is an array or list structure of function calls and parameters used in modern computer programming and CPU architecture. Similar to a stack of plates at a restaurant, elements in a stack are added or removed from the top of the stack, in a “last in, first out” order. Unlike lists or arrays, random access is not allowed for the objects contained in the stack.

There are two types of operations in Stack-

Push– To add data into the stack.

Pop– To remove data from the stack

```
Practical 3a.py - C:\Users\meetd\Downloads\Practical 3a.py (3.8.5)
File Edit Format Run Options Window Help
class Stack:
    def __init__(self):
        self.stack_arr = []
    def push(self,value):
        self.stack_arr.append(value)
    def pop(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            self.stack_arr.pop()
    def get_head(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            return self.stack_arr[-1]
    def display(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            print(self.stack_arr)
stack = Stack()
stack.push(4)
stack.push(5)
stack.push(6)
stack.pop()
stack.display()
stack.get_head()
```

Ln: 1 Col: 0

Windows Type here to search ○ Download Data St... Pr Run SQ... μTorre... IDLE (P... DS-IT L... 08:27 PM ENG 25-10-2020

```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\meetd\Downloads\Practical 3a.py =====
[4, 5]
>>> |
```

Ln: 6 Col: 4

Windows Type here to search ○ Download Data St... Pr Run SQ... μTorre... IDLE (P... DS-IT L... 08:28 PM ENG 25-10-2020

Practical3B

AIM: Implement Tower of Hanoi.

THEORY: We are given n disks and a series of rods, we need to transfer all the disks to the final rod under the given constraints—We can move only one disk at a time. Only the uppermost disk

```

prac3b.py - C:\Users\meetd\Desktop\prac3b.py (3.8.5)
File Edit Format Run Options Window Help
class Stack:
    def __init__(self):
        self.stack_arr = []
    def push(self,value):
        self.stack_arr.append(value)
    def pop(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            self.stack_arr.pop()
    def get_head(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            return self.stack_arr[-1]
    def display(self):
        if len(self.stack_arr) == 0:
            print('Stack is empty!')
            return None
        else:
            print(self.stack_arr)

A = Stack()
B = Stack()
C = Stack()
def Hanoi(n, fromrod,to,temp):
    if n == 1:
        fromrod.pop()
        to.push('disk 1')
        if to.display() != None:
            print(to.display())
    else:
        Hanoi(n-1, fromrod,temp,to)
        fromrod.pop()
        to.push('disk 1')
        if to.display() != None:
            print(to.display())
        Hanoi(n-1, temp,to,fromrod)

n = int(input('Enter the number of the disk in rod A : '))
for i in range(n):
    A.push(f'disk {i+1} ')
Hanoi(n, A, C, B)

```

The screenshot shows a Windows desktop environment. In the center is a terminal window titled "Python 3.8.5 Shell". The window displays the following text:

```
Python 3.8.5 (tags/v3.8.5:580fb00, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ===== RESTART: C:\Users\meetd\Desktop\prac3b.py =====
Enter the number of the disk in rod A : 2
['disk 1']
['disk 2']
['disk 2', 'disk 1']
>>>
```

The taskbar at the bottom of the screen contains several icons, including "Type here to search", "Downloads", "Data Stru...", "Prac3b.py...", "Python 3...", "DS-IT Ind...", and system icons for battery, signal, and date/time (11:32 AM, 26-10-2020). The status bar at the bottom right of the terminal window shows "Ln: 9 Col: 4".

Practical 3C

AIM: WAP to scan a polynomial using linked list and add two polynomials.

THEORY: Polynomial is a mathematical expression that consists of variables and coefficients. for example $x^2 - 4x + 7$

In the Polynomial linked list, the coefficients and exponents of the polynomial are defined as the data node of the list. For adding two polynomials that are stored as a linked list. We need to add the coefficients of variables with the same power. In a linked list node contains 3 members, coefficient value link to the next node. a linked list that is used to store Polynomial looks like –

Polynomial : $4x^7 + 12x^2 + 45$

```
Practical 3c.py - C:\Users\meetd\Downloads\Practical 3c.py (3.8.5)
File Edit Format Run Options Window Help
if type(first.element) == type(my_list.head.element):
    print(first.element.element)
    first = first.next
print(first.element)
first = first.next

def reverse_display(self):
    if self.size == 0:
        print("No element")
        return None
    last = my_list.get_tail()
    print(last.element)
    while last.previous:
        if type(last.previous.element) == type(my_list.head):
            print(last.previous.element.element)
            if last.previous == self.head:
                return None
            else:
                last = last.previous
        print(last.previous.element)
    last = last.previous

def add_head(self,e):
    #temp = self.head
    self.head = Node(e)
    #self.head.next = temp
    self.size += 1

def get_tail(self):
    last_object = self.head
    while (last_object.next != None):
        last_object = last_object.next
    return last_object

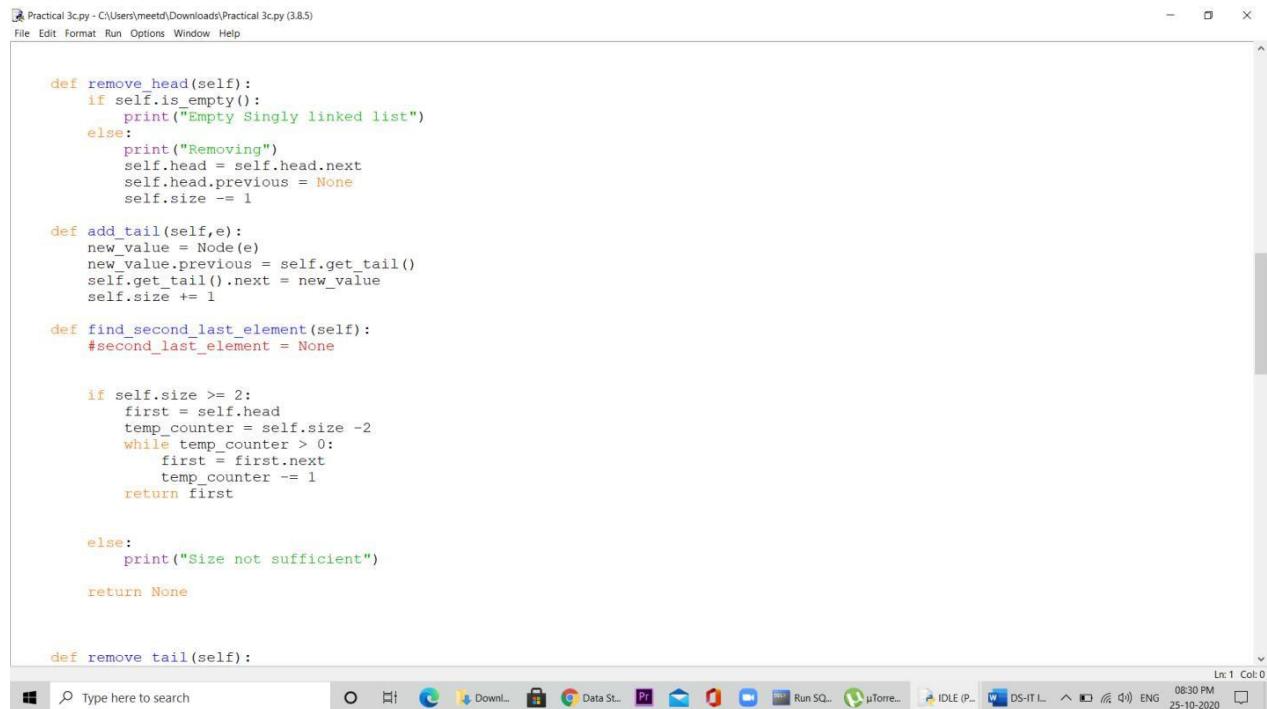
def remove_head(self):
    def remove_tail(self):
        if self.is_empty():
            print("Empty Singly linked list")
        elif self.size == 1:
            self.head == None
            self.size -= 1
        else:
            Node = self.find_second_last_element()
            if Node:
                Node.next = None
                self.size -= 1

    def get_node_at(self,index):
        element_node = self.head
        counter = 0
        if index == 0:
            return element_node.element
        if index > self.size-1:
            print("Index out of bound")
            return None
        while(counter < index):
            element_node = element_node.next
            counter += 1
        return element_node

    def get_previous_node_at(self,index):
        if index == 0:
            print('No previous value')
            return None
        return my_list.get_node_at(index).previous

    def remove_between_list(self,position):
        if position > self.size-1:
            print("Index out of bound")
        elif position == self.size-1:
            self.remove_tail()

Ln: 1 Col: 0
```



```
Practical 3c.py - C:\Users\meeted\Downloads\Practical 3c.py (3.8.5)
File Edit Format Run Options Window Help

def remove_head(self):
    if self.is_empty():
        print("Empty Singly linked list")
    else:
        print("Removing")
        self.head = self.head.next
        self.head.previous = None
        self.size -= 1

def add_tail(self,e):
    new_value = Node(e)
    new_value.previous = self.get_tail()
    self.get_tail().next = new_value
    self.size += 1

def find_second_last_element(self):
    #second_last_element = None

    if self.size >= 2:
        first = self.head
        temp_counter = self.size - 2
        while temp_counter > 0:
            first = first.next
            temp_counter -= 1
        return first

    else:
        print("Size not sufficient")

    return None

def remove_tail(self):
```

Type here to search

Ln: 1 Col: 0

08:30 PM ENG 25-10-2020

```

Practical 3c.py - C:\Users\meetd\Downloads\Practical 3c.py (3.8.5)
File Edit Format Run Options Window Help
    print("Index out of bound")
    elif position == self.size-1:
        self.remove_tail()
    elif position == 0:
        self.remove_head()
    else:
        prev_node = self.get_node_at(position-1)
        next_node = self.get_node_at(position+1)
        prev_node.next = next_node
        next_node.previous = prev_node
        self.size -= 1

def add_between_list(self,position,element):
    element_node = Node(element)
    if position > self.size:
        print("Index out of bound")
    elif position == self.size:
        self.add_tail(element)
    elif position == 0:
        self.add_head(element)
    else:
        prev_node = self.get_node_at(position-1)
        current_node = self.get_node_at(position)
        prev_node.next = element_node
        element_node.previous = prev_node
        element_node.next = current_node
        current_node.previous = element_node
        self.size += 1

def search (self,search_value):
    index = 0
    while (index < self.size):
        value = self.get_node_at(index)
        if value.element == search_value:
            return value.element
        index += 1
    print("Not Found")
    return False

Ln: 1 Col: 0

```

Type here to search

08:30 PM 25-10-2020


```

Practical 3c.py - C:\Users\meetd\Downloads\Practical 3c.py (3.8.5)
File Edit Format Run Options Window Help
def search (self,search_value):
    index = 0
    while (index < self.size):
        value = self.get_node_at(index)
        if value.element == search_value:
            return value.element
        index += 1
    print("Not Found")
    return False

def merge(self,linkedlist_value):
    if self.size > 0:
        last_node = self.get_node_at(self.size-1)
        last_node.next = linkedlist_value.head
        linkedlist_value.head.previous = last_node
        self.size = self.size + linkedlist_value.size

    else:
        self.head = linkedlist_value.head
        self.size = linkedlist_value.size

my_list = LinkedList()
order = int(input('Enter the order for polynomial : '))
my_list.add_head(Node(int(input("Enter coefficient for power {order} : "))))
for i in reversed(range(order)):
    my_list.add_tail(int(input("Enter coefficient for power {i} : ")))

my_list2 = LinkedList()
my_list2.add_head(Node(int(input("Enter coefficient for power {order} : "))))
for i in reversed(range(order)):
    my_list2.add_tail(int(input("Enter coefficient for power {i} : ")))

for i in range(order + 1):
    print(my_list.get_node_at(i).element + my_list2.get_node_at(i).element)

Ln: 1 Col: 0

```

Type here to search

08:31 PM 25-10-2020

The screenshot shows a Windows desktop environment. In the center is a terminal window titled "Python 3.8.5 Shell". The window displays a script named "Practical_3c.py" which prompts the user to enter coefficients for a polynomial of degree 5. The user enters coefficients for powers 5 through 0, resulting in the polynomial $11x^5 + 70x^4 + 11x^3 + 5x^2 + 7x + 2$. The terminal window has a standard Windows title bar with icons for File, Edit, Shell, Debug, Options, Window, and Help. It also shows the Python version and build information. The bottom right corner of the terminal window indicates "Ln: 24 Col: 4". Below the terminal is a taskbar with various pinned icons including File Explorer, Edge, Downloads, Data St..., Pr..., Run SQ..., Run SQ..., IDLE (P...), DS-IT I..., and a system tray icon. The taskbar also shows the date and time as "25-10-2020 08:31 PM".

```
>>> ===== RESTART: C:\Users\meetd\Downloads\Practical_3c.py =====
Enter the order for polynomial : 5
Enter coefficient for power 5 : 6
Enter coefficient for power 4 : 5
Enter coefficient for power 3 : 5
Enter coefficient for power 2 : 5
Enter coefficient for power 1 : 5
Enter coefficient for power 0 : 5
Enter coefficient for power 5 : 5
Enter coefficient for power 4 : 65
Enter coefficient for power 3 : 6
Enter coefficient for power 2 : 0
Enter coefficient for power 1 : 2
Enter coefficient for power 0 : 2
11
70
11
5
7
2
>>>
```

Practical 3d

AIM:WAP to calculate factorial and to compute the factors of a given no.

(i) using recursion, (ii) using iteration

THEORY: The factorial of a number is the product of all the integers from 1 to that number.

For example, the factorial of 6 is $1*2*3*4*5*6 = 720$. Factorial is not defined for negative numbers and the factorial of zero is one, $0! = 1$.

(I) Recursion

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

(II) Iteration

Repeating identical or similar tasks without making errors is something that computers do well and people do poorly. Repeated execution of a set of statements is called iteration. Because iteration is so common, Python provides several language features to make it easier

```

Practical 3d.py - C:\Users\meetd\Downloads\Practical 3d.py (3.8.5)
File Edit Format Run Options Window Help
factorial = 1
n = int(input("Enter Number: "))
for i in range(1,n+1):
    factorial = factorial * i

print(f'Factorial is : {factorial}')

fact = []
for i in range(1,n+1):
    if (n/i).is_integer():
        fact.append(i)

print(f'Factors of the given numbers is : {fact}')

factorial = 1
index = 1
n = int(input("Enter number : "))
def calculate_factorial(n,factorial,index):
    if index == n:
        print(f'Factorial is : {factorial}')
        return True
    else:
        index = index + 1
        calculate_factorial(n,factorial * index,index)
calculate_factorial(n,factorial,index)

fact = []
def calculate_factors(n,factors,index):
    if index == n+1:
        print(f'Factors of the given numbers is : {factors}')
        return True
    elif (n/index).is_integer():
        factors.append(index)
        index += 1
        calculate_factors(n,factors,index)
    else:
        index += 1
        calculate_factors(n,factors,index)
calculate_factors(n,factors,index)

```

The screenshot shows a Windows desktop environment. In the center is a terminal window titled "Python 3.8.5 Shell". The window displays the following Python session:

```
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ===== RESTART: C:\Users\meetd\Downloads\Practical 3d.py =====
Enter Number: 5
Factorial is : 120
Factors of the given numbers is : [1, 5]
Enter number : 5
Factorial is : 120
Factors of the given numbers is : [1, 5]
>>> |
```

The taskbar at the bottom of the screen shows several pinned icons, including File Explorer, Microsoft Edge, OneDrive, Data St..., Pr, Run SQ..., µTorr..., IDLE (P...), DS-IT L..., and a few others. The system tray indicates the date as 25-10-2020 and the time as 08:35 PM.

Practical 4

AIM: Perform Queues operations using Circular Array implementation.

THEORY: Circular queue avoids the wastage of space in a regular queue implementation using arrays.

Circular Queue works by the process of circular increment i.e. when we try to increment the pointer and we reach the end of the queue, we start from the beginning of the queue.

Here, the circular increment is performed by modulo division with the queue size. That is,

if REAR + 1 == 5 (overflow!), REAR = (REAR + 1)%5 = 0 (start of queue)

The circular queue work as follows:

two pointers FRONT and REAR

FRONT track the first element of the queue

REAR track the last elements of the queue

initially, set value of FRONT and REAR to -1

1. Enqueue Operation

check if the queue is full for the first element, set value of FRONT to 0

circularly increase the REAR index by 1 (i.e. if the rear reaches the end, next it would be at the start of the queue)

add the new element in the position pointed to by REAR

2. Dequeue Operation. check if the queue is empty return the value pointed by FRONT circularly increase the FRONT index by 1 for the last element, reset the values of FRONT and REAR to -1.

```

Practical 4.py - C:\Users\meetd\Downloads\Practical 4.py (3.8.5)
File Edit Format Run Options Window Help
class ArrayQueue:
    """FIFO queue implementation using a Python list as underlying storage."""
    DEFAULT_CAPACITY = 10           # moderate capacity for all new queues

    def __init__(self):
        """Create an empty queue."""
        self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
        self._size = 0
        self._front = 0
        self._back = 0

    def __len__(self):
        """Return the number of elements in the queue."""
        return self._size

    def is_empty(self):
        """Return True if the queue is empty."""
        return self._size == 0

    def first(self):
        """Return (but do not remove) the element at the front of the queue.
        Raise Empty exception if the queue is empty.
        """
        if self.is_empty():
            raise Empty('Queue is empty')
        return self._data[self._front]

    def dequeueStart(self):
        """Remove and return the first element of the queue (i.e., FIFO).
        Raise Empty exception if the queue is empty.
        """
        if self.is_empty():
            raise Empty('Queue is empty')
        answer = self._data[self._front]
        self._data[self._front] = None          # help garbage collection
        self._front = (self._front + 1) % len(self._data)
        self._size -= 1
        return answer

```

```

Practical 4.py - C:\Users\meetd\Downloads\Practical 4.py (3.8.5)
File Edit Format Run Options Window Help
        self._data[self._front] = None          # help garbage collection
        self._front = (self._front + 1) % len(self._data)
        self._size -= 1
        self._back = (self._front + self._size - 1) % len(self._data)
        return answer

    def dequeueEnd(self):
        """Remove and return the Last element of the queue.
        Raise Empty exception if the queue is empty.
        """
        if self.is_empty():
            raise Empty('Queue is empty')
        back = (self._front + self._size - 1) % len(self._data)
        answer = self._data[back]
        self._data[back] = None          # help garbage collection
        self._front = self._front
        self._size -= 1
        self._back = (self._front + self._size - 1) % len(self._data)
        return answer

    def enqueueEnd(self, e):
        """Add an element to the back of queue."""
        if self._size == len(self._data):
            self._resize(2 * len(self._data))      # double the array size
        avail = (self._front + self._size) % len(self._data)
        self._data[avail] = e
        self._size += 1
        self._back = (self._front + self._size - 1) % len(self._data)

    def enqueueStart(self, e):
        """Add an element to the start of queue."""
        if self._size == len(self._data):
            self._resize(2 * len(self._data))      # double the array size
        self._front = (self._front - 1) % len(self._data)
        avail = (self._front + self._size) % len(self._data)
        self._data[self._front] = e
        self._size += 1
        self._back = (self._front + self._size - 1) % len(self._data)

```

```

Practical 4.py - C:\Users\meetc\Downloads\Practical 4.py (3.8.5)
File Edit Format Run Options Window Help
avail = (self._front + self._size) % len(self._data)
self._data[self._front] = e
self._size += 1
self._back = (self._front + self._size - 1) % len(self._data)

def _resize(self, cap):           # we assume cap >= len(self)
    """Resize to a new list of capacity >= len(self)."""
    old = self._data            # keep track of existing list
    self._data = [None] * cap    # allocate list with new capacity
    walk = self._front
    for k in range(self._size):  # only consider existing elements
        self._data[k] = old[walk] # intentionally shift indices
        walk = (1 + walk) % len(old) # use old size as modulus
    self._front = 0              # front has been realigned
    self._back = (self._front + self._size - 1) % len(self._data)

queue = ArrayQueue()
queue.enqueueEnd(1)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue._data
queue.enqueueEnd(2)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue._data
queue.dequeueStart()
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.enqueueEnd(3)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.enqueueEnd(4)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.dequeueStart()
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.enqueueStart(5)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.dequeueEnd()
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")
queue.enqueueEnd(6)
print(f"First Element: {queue._data[queue._front]}, Last Element: {queue._data[queue._back]}")

```

```

Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=====
RESTART: C:\Users\meetc\Downloads\Practical 4.py =====
First Element: 1, Last Element: 1
First Element: 1, Last Element: 2
First Element: 2, Last Element: 2
First Element: 2, Last Element: 3
First Element: 2, Last Element: 4
First Element: 3, Last Element: 4
First Element: 5, Last Element: 4
First Element: 5, Last Element: 3
First Element: 5, Last Element: 6
>>> |

```

Practical 5

AIM: Write a program to search an element from a list. Give user the option to perform Linear or Binary search.

THEORY: Linear Search:

This linear search is a basic search algorithm which searches all the elements in the list and finds the required value..... This is also known as sequential search.

Binary Search:

In computer science, a binary search or half-interval search algorithm finds the position of a target value within a sorted array. The binary search algorithm can be classified as a dichotomy divide-and-conquer search algorithm and executes in logarithmic time.

```

practical_5.py - C:\Users\meetd\Downloads\practical_5.py (3.8.5)
File Edit Format Run Options Window Help
class Array:

    def __init__(self, array, number):
        self.lst = sorted(array)
        self.number = number

    def binary_search(self, lst, n, start, end):

        if start <= end:
            mid = (end + start) // 2
            if lst[mid] == n:
                return f'position: {mid}'
            elif lst[mid] > n:
                return binary_search(lst, n, start, mid-1)
            else:
                return binary_search(lst, n, mid + 1, end)
        else:
            return -1

    def linear_search(self, lst, n):
        for i in range(len(lst)):
            if lst[i] == n:
                return f'Position :{i}'
        return -1

    def run_search(self):
        while True:
            print('Select the searching algorithm:')
            print('1. Linear Search.')
            print('2. Binary Search.')
            print('3. quit.')
            opt = int(input('Option: '))
            if opt == 2:
                print(search.binary_search(self.lst, self.number, 0, len(self.lst)-1))
            elif opt == 1:
                print(search.linear_search(self.lst, self.number))
            else:
                break

```

```

practical_5.py - C:\Users\meetd\Downloads\practical_5.py (3.8.5)
File Edit Format Run Options Window Help
    return f'position: {mid}'
elif lst[mid] > n:
    return binary_search(lst,n,start,mid-1)
else:
    return binary_search(lst,n,mid + 1,end)
else:
    return -1

def linear_search(self,lst,n):
    for i in range(len(lst)):
        if lst[i] == n:
            return f'Position :{i}'
    return -1

def run_search(self):
    while True:
        print('Select the searching algorithm:')
        print('1. Linear Search.')
        print('2. Binary Search.')
        print('3. quit.')
        opt = int(input('Option: '))
        if opt == 2:
            print(search.binary_search(self.lst,self.number,0,len(lst)-1))
        elif opt == 1:
            print(search.linear_search(self.lst,self.number))
        else:
            break

lst = [1,2,3,4,5,6,7,8]
number = 4
search = Array(lst,number)
search.run_search()

```

Ln 1 Col 0

Windows Taskbar: Type here to search, File Explorer, Edge, Downloads, Data Sync, Print, Mail, OneDrive, Run, Task View, File Explorer, IDLE (Python 3.8.5), DS-IT I..., ENG 09:13 PM 25-10-2020

Python 3.8 Shell

File Edit Shell Debug Options Window Help

Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

>>>

===== RESTART: C:\Users\meetd\Downloads\practical_5.py =====

Select the searching algorithm:

1. Linear Search.

2. Binary Search.

3. quit.

Option: 1

Position :3

Select the searching algorithm:

1. Linear Search.

2. Binary Search.

3. quit.

Option: 2

position: 3

Select the searching algorithm:

1. Linear Search.

2. Binary Search.

3. quit.

Option: 3

>>> |

Ln 22 Col 4

Windows Taskbar: Type here to search, File Explorer, Edge, Downloads, Data Sync, Print, Mail, OneDrive, Run, Task View, File Explorer, IDLE (Python 3.8.5), DS-IT I..., ENG 09:13 PM 25-10-2020

Practical 6

AIM: WAP to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.

THEORY: Bubble Sort: Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Selection Sort:

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array

Insertion Sort:

Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

```

practical_6.py - C:\Users\meetd\Downloads\practical_6.py (3.8.5)
File Edit Format Run Options Window Help
class Sorting:
    def __init__(self, lst):
        self.lst = lst

    def bubble_sort(self, lst):
        for i in range(len(lst)):
            for j in range(len(lst)):
                if lst[i] < lst[j]:
                    lst[i], lst[j] = lst[j], lst[i]
                else:
                    pass
        return lst

    def selection_sort(self, lst):
        for i in range(len(lst)):
            smallest_element = i
            for j in range(i+1, len(lst)):
                if lst[smallest_element] > lst[j]:
                    smallest_element = j
            lst[i], lst[smallest_element] = lst[smallest_element], lst[i]
        return lst

    def insertion_sort(self, lst):
        for i in range(1, len(lst)):
            index = lst[i]
            j = i-1
            while j >= 0 and index < lst[j] :
                lst[j + 1] = lst[j]
                j -= 1
            lst[j + 1] = index
        return lst

    def run_sort(self):
        while True:
            print('Select the sorting algorithm:')
            print('1. Bubble Sort.')
            print('2. Selection Sort.')

```

The image shows a Windows desktop environment with three open windows:

- Code Editor Window:** The title bar reads "practical_6.py - C:\Users\meetd\Downloads\practical_6.py (3.8.5)". The code implements three sorting algorithms: Bubble Sort, Selection Sort, and Insertion Sort. It also includes a menu bar with File, Edit, Format, Run, Options, Window, and Help.
- Python Shell Window:** The title bar reads "Python 3.8.5 Shell". The window displays the output of running the "practical_6.py" script. It shows the program prompting for a sorting algorithm selection (1. Bubble Sort, 2. Selection Sort, 3. Insertion Sort, 4. Quit), and then performing bubble sort on the list [4, 2, 3, 9, 12] for each option.
- Command Prompt Window:** The title bar reads "cmd". The window shows the command "dir" being run, listing files and folders in the current directory.

```

practical_6.py - C:\Users\meetd\Downloads\practical_6.py (3.8.5)
File Edit Format Run Options Window Help
for i in range(len(lst)):
    smallest_element = i
    for j in range(i+1, len(lst)):
        if lst[smallest_element] > lst[j]:
            smallest_element = j
    lst[i], lst[smallest_element] = lst[smallest_element], lst[i]
return lst

def insertion_sort(self, lst):
    for i in range(1, len(lst)):
        index = lst[i]
        j = i-1
        while j >= 0 and index < lst[j] :
            lst[j + 1] = lst[j]
            j -= 1
        lst[j + 1] = index
    return lst

def run_sort(self):
    while True:
        print('Select the sorting algorithm:')
        print('1. Bubble Sort.')
        print('2. Selection Sort.')
        print('3. Insertion Sort.')
        print('4. Quit')
        opt = int(input('Option: '))
        if opt == 1:
            print(sort.bubble_sort(self.lst))
        elif opt == 2:
            print(sort.selection_sort(self.lst))
        elif opt == 3:
            print(sort.insertion_sort(self.lst))
        else:
            break
lst = [4,2,3,9,12]
sort = Sorting(lst)
sort.run_sort()

Windows PowerShell
Type here to search
O Downloads Data St... Pr Run SQL... IDLE (Py... DS-IT Ind... ENG 09:22 PM 25-10-2020
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ===== RESTART: C:\Users\meetd\Downloads\practical_6.py =====
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option: 1
[1, 2, 3, 4, 9, 12]
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option: 2
[1, 2, 3, 4, 9, 12]
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option: 3
[1, 2, 3, 4, 9, 12]
Select the sorting algorithm:
1. Bubble Sort.
2. Selection Sort.
3. Insertion Sort.
4. Quit
Option: 4
>>> |
```

Practical 7

Implement the following for Hashing.

THEORY: Hashing: Hashing is an important Data Structure which is designed to use a special function called the Hash function which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used.

Practical 7A

AIM: Write a program to implement the collision technique.

THEORY: Collisions:

A Hash Collision Attack is an attempt to find two input strings of a hash function that produce the same hashresult. If two separate inputs produce the same hashoutput, it is called a collision.

Collision Techniques:

Separate Chaining:

The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

Open Addressing:

Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, the size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).

The image shows a Windows desktop environment with three open windows:

- Code Editor Window:** The title bar says "prac7a.py - C:/Users/meetd/Desktop/prac7a.py (3.8.5)". The code implements a simple Hash class with methods for initialization, getting key values, and calculating hash functions. It includes logic to detect collisions when inserting values into a list of keys.
- Python Terminal Window:** The title bar says "Python 3.8.5 Shell". It shows the command-line interface with the Python interpreter. The output displays the execution of the script, showing the initial state of the list, the detection of collisions, and the final state after processing.
- File Explorer Window:** The title bar says "File Explorer". It shows a list of files and folders, including "prac7a.py" and "DS-IT L...".

Practical7B

AIM: Write a program to implement the concept of linear probing.

THEORY: Linear probing is a scheme in computer programming for resolving collisions in hash tables, data structures for maintaining a collection of key–value pairs and looking up the value associated with a given key.....Along with quadratic probing and double hashing, linear probing is a form of open addressing.

```

prac7a.py - C:/Users/meetd/Desktop/prac7a.py (3.8.5)
File Edit Format Run Options Window Help
class Hash:
    def __init__(self, keys: int, lower_range: int, higher_range: int) -> None:
        self.value = self.hash_function(keys, lower_range, higher_range)

    def get_key_value(self) -> int:
        return self.value

    @staticmethod
    def hash_function(keys: int, lower_range: int, higher_range: int) -> int:
        if lower_range == 0 and higher_range > 0:
            return keys % higher_range

if __name__ == '__main__':
    linear_probing = True
    list_of_keys = [23, 43, 1, 87]
    list_of_list_index = [None]*4
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        list_index = Hash(value, 0, len(list_of_keys)).get_key_value()
        print("Hash value for " + str(value) + " is :" + str(list_index))
        if list_of_list_index[list_index]:
            print("Collision detected for " + str(value))
            if linear_probing:
                old_list_index = list_index
                if list_index == len(list_of_list_index) - 1:
                    list_index = 0
                else:
                    list_index += 1
                list_full = False
                while list_of_list_index[list_index]:
                    if list_index == old_list_index:
                        list_full = True
                        break
                    if list_index + 1 == len(list_of_list_index):
                        list_index = 0
                    else:
                        list_index += 1
                if list_full:
                    print("List was full . Could not save")
                else:
                    list_of_list_index[list_index] = value
            else:
                list_of_list_index[list_index] = value
    print("After: " + str(list_of_list_index))

prac7a.py - C:/Users/meetd/Desktop/prac7a.py (3.8.5)
File Edit Format Run Options Window Help
def hash_function(keys: int, lower_range: int, higher_range: int) -> int:
    if lower_range == 0 and higher_range > 0:
        return keys % higher_range

if __name__ == '__main__':
    linear_probing = True
    list_of_keys = [23, 43, 1, 87]
    list_of_list_index = [None]*4
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        list_index = Hash(value, 0, len(list_of_keys)).get_key_value()
        print("Hash value for " + str(value) + " is :" + str(list_index))
        if list_of_list_index[list_index]:
            print("Collision detected for " + str(value))
            if linear_probing:
                old_list_index = list_index
                if list_index == len(list_of_list_index) - 1:
                    list_index = 0
                else:
                    list_index += 1
                list_full = False
                while list_of_list_index[list_index]:
                    if list_index == old_list_index:
                        list_full = True
                        break
                    if list_index + 1 == len(list_of_list_index):
                        list_index = 0
                    else:
                        list_index += 1
                if list_full:
                    print("List was full . Could not save")
                else:
                    list_of_list_index[list_index] = value
            else:
                list_of_list_index[list_index] = value
    print("After: " + str(list_of_list_index))

```

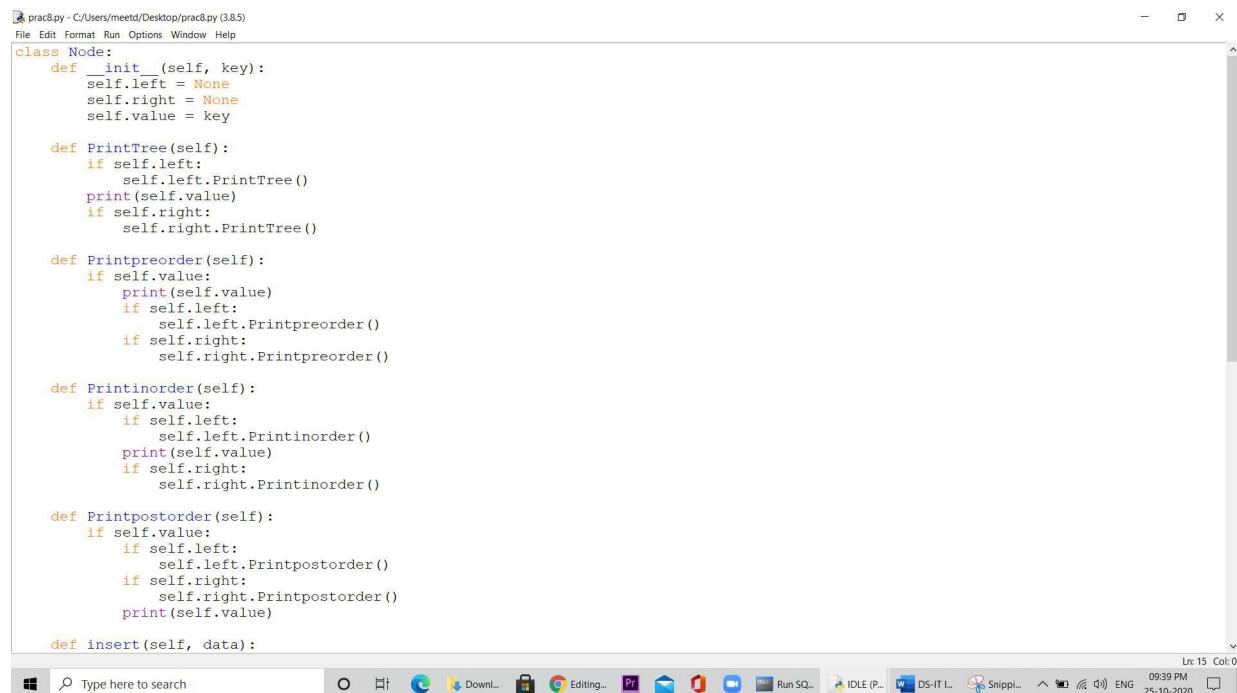
```
| Before : [None, None, None, None]
| Hash value for 23 is :3
| Hash value for 43 is :3
| Collision detected for 43
| Hash value for 1 is :1
| Hash value for 87 is :3
| Collision detected for 87
| After: [43, 1, 87, 23]
>>> |
```

Practical 8

AIM: Inorder: In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder traversal is reversed can be used.

Preorder: Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree.

Postorder: Postorder traversal is also useful to get the postfix expression of an expression tree.



```
prac8.py - C:/Users/meetd/Desktop/prac8.py (3.8.5)
File Edit Format Run Options Window Help
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.value = key

    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print(self.value)
        if self.right:
            self.right.PrintTree()

    def Printpreorder(self):
        if self.value:
            print(self.value)
        if self.left:
            self.left.Printpreorder()
        if self.right:
            self.right.Printpreorder()

    def Printinorder(self):
        if self.value:
            if self.left:
                self.left.Printinorder()
            print(self.value)
            if self.right:
                self.right.Printinorder()

    def Printpostorder(self):
        if self.value:
            if self.left:
                self.left.Printpostorder()
            if self.right:
                self.right.Printpostorder()
            print(self.value)

    def insert(self, data):
```

The image shows a Windows desktop environment with two open windows. The top window is a code editor titled 'prac8.py - C:/Users/meetd/Desktop/prac8.py (3.8.5)'. It contains Python code for inserting data into a binary search tree and printing it in various orders. The bottom window is a terminal titled 'Python 3.8.5 Shell' showing the execution of the script and its output.

```

prac8.py - C:/Users/meetd/Desktop/prac8.py (3.8.5)
File Edit Format Run Options Window Help

def insert(self, data):
    if self.value:
        if data < self.value:
            if self.left is None:
                self.left = Node(data)
            else:
                self.left.insert(data)
        elif data > self.value:
            if self.right is None:
                self.right = Node(data)
            else:
                self.right.insert(data)
    else:
        self.value = data

if __name__ == '__main__':
    root = Node(10)
    root.left = Node(12)
    root.right = Node(5)
    print("Without any order")
    root.PrintTree()
    root_1 = Node(None)
    root_1.insert(28)
    root_1.insert(4)
    root_1.insert(13)
    root_1.insert(130)
    root_1.insert(123)
    print("Now ordering with insert")
    root_1.PrintTree()
    print("Pre order")
    root_1.Printpreorder()
    print("In Order")
    root_1.Printinorder()
    print("Post Order")
    root_1.Printpostorder()

Ln: 15 Col: 0

Type here to search 09:39 PM 25-10-2020
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=====
RESTART: C:/Users/meetd/Desktop/prac8.py =====
Without any order
12
10
5
Now ordering with insert
4
13
28
123
130
Pre order
28
4
13
130
123
In Order
4
13
28
123
130
Post Order
13
4
123
130
28
>>>

Ln: 33 Col: 4

Type here to search 09:40 PM 25-10-2020
File Edit Format Run Options Window Help

```