



MALAD KANDIVALI EDUCATION SOCIETY'S
NAGINDAS KHANDWALA COLLEGE OF COMMERCE, ARTS &
MANAGEMENT STUDIES & SHANTABEN NAGINDAS KHANDWALA
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: Ms. Gunveen Kaur Bindra

Roll No: 368

Programme: BSc IT

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)

Date of Examination:

(College Stamp)

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 Transpose Operation.	
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 of Hanoi. c) WAP to scan a polynomial using linked list and add two polynomials. d) WAP to calculate factorial and to compute the factors of a given no. (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 collision technique. b) Write a program to implement the concept of linear probing.	
8	23/10/2020	Write a program for inorder, postorder and preorder traversal of tree.	

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL – 1(A)

Aim: To 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:

A one-dimensional **array** is a type of linear **array**. Accessing its elements involves a single subscript which can either represent a row or column index.

- 1. Searching-** Searching is an operation or a technique that helps find the place of a given element or value in the list
- 2. Sorting-** Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order. In Numpy, we can perform various sorting operations using the various functions that are provided in the library like sort, lexsort, argsort etc.
numpy.sort() : This function returns a sorted copy of an array.
- 3. Merging-** To merge array elements we have to copy first array's elements into third array first then copy second array's elements into third array after the index of first array elements.
- 4. Reversing-** Every list in Python has a built-in reverse() method you can call to reverse the contents of the list object in-place. Reversing the list in-place means won't create a new list and copy the existing elements to it in reverse order. Instead, it directly modifies the original list object.

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

CODE:

```
File Edit Format Run Options Window Help
11=[10,50,66,25,44,78]
print("List 1: ",11)

11.sort()#it will sort the elements of list1
print("sorted list 1",11)

def search_11():
    i = int(input("Enter element to search in the list:"))
    if i in 11:
        print("The element is in the list")
    else:
        print("The element is not in the list")

search_11()

def reverse_list():
    print("Reversing the list1: ",11[::-1])

reverse_list()

12 = ["car", "pen", "book", "crayons"]
print("List 2: ",12)

12.sort() # it will sort the elements of list2
print("sorted list 2 :", 12)

def merge_list():
    13 = 11 + 12
    print("Merging both the lists: ", 13)

merge_list()
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

OUTPUT:

```
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\simra\OneDrive\Desktop\gunveen\SYIT\DS\ds pracs\Practicalla.py
List 1: [10, 50, 66, 25, 44, 78]
sorted list 1 [10, 25, 44, 50, 66, 78]
Enter element to search in the list:50
The element is in the list
Reversing the list1: [78, 66, 50, 44, 25, 10]
List 2: ['car', 'pen', 'book', 'crayons']
sorted list 2 : ['book', 'car', 'crayons', 'pen']
Merging both the lists: [10, 25, 44, 50, 66, 78, 'book', 'car', 'crayons', 'pen']
>>>
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL – 1(B)

Aim: To write a program to perform the Matrix addition, Multiplication and Transpose Operation.

THEORY:

1. Matrix Addition:

- To add the matrices, we simply traverse through both matrices element by element and insert the smaller element (one with smaller row and col value) into the resultant matrix.
- If we come across an element with the same row and column value, we simply add their values and insert the added data into the resultant matrix.

2. Matrix Multiplication:

- To multiply the matrices, we first calculate transpose of the second matrix to simplify our comparisons and maintain the sorted order.
- So, the resultant matrix is obtained by traversing through the entire length of both matrices and summing the appropriate multiplied values.

3. Matrix Transpose:

- To transpose a matrix, we can simply change every column value to the row value and vice-versa, however, in this case, the resultant matrix won't be sorted as we require.
- Hence, we initially determine the number of elements less than the current element's column being inserted in order to get the exact index of the resultant matrix where the current element should be placed.
- This is done by maintaining an array index [] whose ith value indicates the number of elements in the matrix less than the column i.

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

CODE AND OUTPUT:

```
In [1]: # Program to add two matrices
X = [[1,8,5],
      [2,4,6],
      [3,7,9]]

Y = [[10,11,12],
      [13,14,15],
      [16,17,18]]

result = [[0,0,0],
           [0,0,0],
           [0,0,0]]

# iterate through rows
for i in range(len(X)):
    # iterate through columns
    for j in range(len(X[0])):
        result[i][j] = X[i][j] + Y[i][j]

for r in result:
    print(r)

[11, 13, 15]
[17, 19, 21]
[23, 25, 27]
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
In [2]: #Program to find multiplication of a matrix
for i in range(len(X)):
    for j in range(len(Y[0])):
        for k in range(len(Y)):
            result[i][j] += X[i][k] * Y[k][j]
for r in result:
    print(r)
```

```
[95, 103, 111]
[218, 235, 252]
[341, 367, 393]
```

```
In [3]: # Program to transpose a matrix
# iterate through rows
for i in range(len(X)):
    # iterate through columns
    for j in range(len(X[0])):
        result[i][j] = X[i][j] + Y[i][j]

for r in result:
    print(r)
```

```
[11, 13, 15]
[17, 19, 21]
[23, 25, 27]
```


NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

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 singly linked list, in its simplest form, is a collection of nodes that collectively form a linear sequence. Each node stores a reference to an object that is an element of the sequence, as well as a reference to the next node of the list

- Head points to the first node of the linked list
- Next pointer of the last node is NULL, so if the next current node is NULL, we have reached the end of the linked list.

There are three common types of Linked List.

- Singly Linked List
- Doubly Linked List
- Circular Linked List
- **To insert Elements to a Linked List**

You can add elements to either the beginning, middle or end of the linked list.

- **To Delete from a Linked List**

You can delete either from the beginning, end or from a particular position.

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

CODE AND OUTPUT:

```
class Node:

    def __init__(self, element, next = None ):
        self.element = element
        self.next = next

    def display(self):
        print(self.element)

class LinkedList:

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

    def _len_(self):
        return self.size

    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)
        first = first.next
        while first:

            print(first.element)
            first = first.next

    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
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
def remove_head(self):
    if self.is_empty():
        print("Empty Singly linked list")
    else:
        print("Removing")
        self.head = self.head.next
        self.size -= 1

def add_tail(self,e):
    new_value = Node(e)
    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
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
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 > self.size-1:
        print("Index out of bound")
        return None
    while(counter < index):
        element_node = element_node.next
        counter += 1
    return element_node
```

```
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:
        prev_node = self.get_node_at(position-1)
        next_node = self.get_node_at(position+1)
        prev_node.next = next_node
        self.size -= 1

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)
        current_node = self.get_node_at(position)
        prev_node.next = element
        element.next = current_node
        self.size += 1
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
def search (self,search_value):
    index = 0
    while (index < self.size):
        value = self.get_node_at(index)
        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
        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
        self.size = self.size + linkedlist_value.size

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

Doubly Linked List

```
class Node:
    def __init__(self,element,prev,next):
        self.element=element
        self.prev = prev
        self.next = next

class DoublyLinkedList:

    def __init__(self):
        self._head = Node(None,None,None)
        self._tail = Node(None,None,None)
        self._head.next=self._tail
        self._tail.prev=self._head
        self._size = 0

    def __len__(self):
        return self._size
    def is_empty(self):
        return self._size == 0
    def display(self):
        newest = self._head
        while newest:
            print(newest.element,end='-->')
            newest = newest.next
        print()
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
def add_head(self,e):
    temp = self._head
    if self.is_empty():
        self._head=temp
        self._tail=temp
    else:
        self._head = Node(e,None,None)
        self._head.next = temp
        self._size += 1

def add_tail(self,e):
    temp = Node(e,None,None)
    if self.is_empty():
        self._head=temp
        self._tail=temp
    else:
        self._tail.next = temp
        temp.prev=self._tail
        self._tail=temp
        self._size += 1

def remove_head(self):
    if self.is_empty():
        print("List is empty")
    value = self._head.element
    self._head = self._head.next
    self._head.prev = None
    self._size -= 1
    if self.is_empty():
        self._tail = None
    return value
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
def remove_tail(self):
    if self.is_empty():
        print("List is empty")
    newest = self._head
    i = 0
    while i < len(self) - 2:
        newest = newest.next
        i += 1
    self._tail = newest
    newest = newest.next
    value = newest.element
    self._tail.next = None
    self._size -= 1
    return value

def search_item(self, x):
    if self._head is None:
        print("List has no elements")
        return
    n = self._head
    while n is not None:
        if n.element == x:
            print("Item found")
            return True
        n = n.next
    print("item not found")
    return False
```

```
def reverse_list(self):
    first = self._head
    second = first.next
    first.next = None
    first.prev = second
    while second is not None:
        second.prev = second.next
        second.next = first
        first = second
        second = second.prev
    self._head = first
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

OUTPUT:

```
l1 = LinkedList()  
l1.remove_head()
```

Empty Singly linked list

```
list1=LinkedList()  
list1.add_head(3)  
list1.add_head(2)  
list1.add_head(1)  
list1.add_tail(4)  
list1.add_tail(5)  
list1.display()  
list1.remove_head()  
list1.remove_tail()  
list1.add_between_list(3,9)  
list1.remove_between_list(2)  
list1.display()
```

1
2
3
4
5
Removing
2
3
9

```
list2=LinkedList()  
list2.add_head(8)  
list2.add_head(7)  
list2.add_head(6)  
list1.merge(list2)  
print('merge')  
list1.display()
```

merge
2
3
9
6
7
8

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368
Doubly Linked List

```
l = DoublyLinkedList()  
l.add_tail(10)  
l.add_tail(20)  
l.add_tail(30)  
l.add_tail(40)  
l.display()
```

10-->20-->30-->40-->

```
print('Delete',l.remove_head())  
l.display()
```

Delete 10
20-->30-->40-->

```
print('Delete',l.remove_tail())  
l.display()
```

Delete 40
20-->30-->

```
print("Adding")  
l.add_head(70)  
l.add_head(80)  
l.add_head(90)  
l.add_head(100)  
l.display()
```

Adding
100-->90-->80-->70-->20-->30-->

```
print("Reversing the list")  
l.reverse_list()  
l.display()
```

Reversing the list
30-->20-->70-->80-->90-->100-->

```
l.search_item(70)  
l.display()
```

Item found
30-->20-->70-->80-->90-->100-->

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL – 3(A)

Aim: To perform Stack operations using Array implementation.

THEORY:

In array implementation, the stack is formed by using the array. All the operations regarding the stack are performed using arrays

Push and Pop operations in arrays

1. Push

- Check if the stack is full.
- If the stack is not full, increment top to the next location.
- Assign data to the top element.

2. Pop

- Check if the stack is empty.
- If the stack is not empty, copy top in a temporary variable.
- Decrement top to the previous location.
- Delete the temporary variable.

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

CODE:

```
#Python program to perform stack operations using array implementation.
class ArrayStack:
    def __init__(self):
        self._data = []

    def __len__(self):
        return len(self._data)

    def is_empty(self):
        return len(self._data)==0

    def push(self,e):
        self._data.append(e)

    def pop(self):
        if self.is_empty():
            raise exception('Stack is empty')
        else:
            return self._data.pop()

    def top(self):
        if self.is_empty():
            raise exception('Stack is empty')
        else:
            return self._data[-1]
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

OUTPUT:

```
In [24]: s=ArrayStack()  
         s.push(10)  
         s.push(20)  
         print('Stack',s._data)  
  
Stack [10, 20]
```

```
In [25]: print('Length',len(s))  
  
Length 2
```

```
In [26]: print('Is_empty',s.is_empty())  
  
Is_empty False
```

```
In [27]: print('Popped',s.pop())  
         print('Stack',s._data)  
  
Popped 20  
Stack [10]
```

```
In [28]: print('Popped',s.pop())  
  
Popped 10
```

```
In [29]: print('Is_empty',s.is_empty())  
  
Is_empty True
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

```
In [29]: print('Is_empty',s.is_empty())
```

Is_empty True

```
In [30]: s.push(30)  
s.push(40)
```

```
In [31]: print('Stack',s._data)
```

Stack [30, 40]

```
In [32]: print('Top element',s.top())
```

Top element 40

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL – 3(B)

Aim: To write a program to implement Tower of Hanoi.

THEORY:

Tower of Hanoi is a mathematical puzzle which consists of three towers (pegs) and more than one ring.

These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

Rules

1. The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are –
2. Only one disk can be moved among the towers at any given time.
3. Only the "top" disk can be removed.
4. No large disk can sit over a small disk.

CODE:

```
# tower of hanoi
def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
    if n == 1:
        print ("Move disk 1 from rod",from_rod,"to rod",to_rod)
        return
    TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
    print ("Move disk",n,"from rod",from_rod,"to rod",to_rod)
    TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)
# main
n = 4
TowerOfHanoi(n, 'A', 'C', 'B')
# A, B, C are the rod
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

OUTPUT:

```
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL – 3(C)

Aim: To write a program to scan a polynomial using linked list and add two polynomial.

THEORY:

Linked list is a data structure that stores each element as an object in a node of the list.

Polynomial is a mathematical expression that consists of variables and coefficients.

For example $x^2 - 4x + 7$

Summing **two polynomials** simply means to sum the coefficients of the same powers, if these situations occur. At this point, you simply need to notice that: The constant term (i.e. 1) appears only in the first **polynomial**, so we have nothing to sum. The same goes with the linear factor (i.e. x)

In the **Polynomial linked list**, the coefficients and exponents of the polynomial are defined as the data node of the list.

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368
CODE:

#WAP to scan a polynomial using linked list and add two polynomial.

```
class Node:

    def __init__(self, element, next = None ):
        self.element = element
        self.next = next

    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
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
def display(self):
    if self.size ==0:
        print("no element")
    first = self.head
    print(first.element.element)
    first = first.next
    while first:
        print(first.element)
        first = first.next

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):
    if self.is_empty():
        print("Empty Singly linked list")
    else:
        print("Removing")
        self.head = self.head.next
        self.size -= 1
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
def add_tail(self,e):
    new_value = Node(e)
    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")
    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
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
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 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:
        prev_node = self.get_node_at(position - 1)
        next_node = self.get_node_at(position + 1)
        prev_node.next = next_node
        self.size -= 1
```

```
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)
        current_node = self.get_node_at(position)
        prev_node.next = element
        element.next = current_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
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

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

```
order = 3

l1 = LinkedList()
print("Polynomial 1:")

l1.add_head(Node(int(input(f"coefficient for power {order} : "))))
for i in reversed(range(order)):
    l1.add_tail(int(input(f"coefficient for power {i} : ")))

l2 = LinkedList()
print("Polynomial 2")

l2.add_head(Node(int(input(f"coefficient for power {order} : "))))
for i in reversed(range(order)):
    l2.add_tail(int(input(f"coefficient for power {i} : ")))

print("Adding coefficients of polynomial 1 and 2 ")
print(l1.get_node_at(0).element + l2.get_node_at(0).element, "x^3 + " ,
      l1.get_node_at(1).element + l2.get_node_at(1).element, "x^2 + " ,
      l1.get_node_at(2).element + l2.get_node_at(2).element, "x + " ,
      l1.get_node_at(3).element + l2.get_node_at(3).element)
```

OUTPUT:

```
Polynomial 1:
coefficient for power 3 : 3
coefficient for power 2 : 3
coefficient for power 1 : 3
coefficient for power 0 : 3
Polynomial 2
coefficient for power 3 : 3
coefficient for power 2 : 3
coefficient for power 1 : 3
coefficient for power 0 : 3
Adding coefficients of polynomial 1 and 2
6 x^3 + 6 x^2 + 6 x + 6
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL – 3(D)

Aim: To write a program to calculate factorial and to compute the factors of a given no. (i) using recursion, (ii) using iteration.

THEORY:

Factorial of a given number

If you want to find a factorial of an positive integer, keep multiplying it with all the positive integers less than that number. The final result that you get is the Factorial of that number.

- **Using recursion**

1. User must enter a number and store it in a variable.
2. The number is passed as an argument to a recursive factorial function.
3. The base condition is that the number has to be lesser than or equal to 1 and return 1 if it is
4. Otherwise the function is called recursively with the number minus 1 multiplied by the number itself.
5. The result is returned and the factorial of the number is printed.

- **Using iteration.**

We shall use Python For Loop to find Factorial. For a **range (1, n+1)** of given n, multiply the element over each iteration and return the result after coming out of the loop.

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368
CODE AND OUTPUT:

```
In [6]: #WAP to calculate factorial and to compute the factors of a given no.  
#using recursion  
#using iteration.
```

```
In [7]: #to find factorial of a given number using recursion  
def recur_factorial(n):  
    if n == 1:  
        return n  
    else:  
        return n*recur_factorial(n-1)  
  
num = int(input("Enter a number: "))  
  
# check if the number is negative  
if num < 0:  
    print("Sorry, factorial does not exist for negative numbers")  
elif num == 0:  
    print("The factorial of 0 is 1")  
else:  
    print("The factorial of", num, "is", recur_factorial(num))
```

Enter a number: 5
The factorial of 5 is 120

```
In [8]: #to find factorial of a given number using iteration  
num = int(input("Enter a number: "))  
factorial = 1  
if num < 0 :  
    print("Factorial does not exist")  
elif num == 0 or num==1:  
    print("Factorial is of ",num,"is 1")  
else:  
    for i in range(1,num+1):  
        factorial = factorial* i  
    print("Factorial of ",num," is ",factorial)
```

Enter a number: 6
Factorial of 6 is 720

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
In [9]: #to find factors of a given number using recursion
def factors_recursion(x,i):
    if i==x+1:
        return
    if x%i == 0:
        print(i)
    return factors_recursion(x,i+1) #simpler version of the problem

factors_recursion(10,1)
```

1
2
5
10

```
In [11]: #to find factors of a given number using iteration
def iter_factors(x):
    print ("The factors of",x,"are:")
    for i in range(1, x + 1):
        if x % i == 0:
            #if the number divided by i is zero, then i is a factor of that number
            print (i)

number = int(input("Enter a number: "))

print (iter_factors(number))
```

Enter a number: 6
The factors of 6 are:
1
2
3
6
None

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL - 4

Aim: To perform Queues operations using Circular Array implementation.

Theory:

1. Initialize the queue, size of the queue (maxSize), head and tail pointers
2. Enqueue:
 - Check if the number of elements (size) is equal to the size of the queue (maxSize):
 - If yes, throw error message "Queue Full!"
 - If no, append the new element and increment the tail pointer.
3. Dequeue:
 - Check if the number of elements (size) is equal to 0:
 - If yes, throw error message "Queue Empty!".
 - If no, increment head pointer.
4. is_empty (): Returns True if queue is empty, False otherwise.
5. Size:
 - If $\text{tail} \geq \text{head}$, $\text{size} = \text{tail} - \text{head}$
 - if $\text{head} > \text{tail}$, $\text{size} = \text{maxSize} - (\text{head} - \text{tail})$

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368
CODE:

```
In [1]: #Python program to perform Queues operations using Circular Array implementation.
```

```
In [16]: #Performing Queue operations using Circular Array implementation
class ArrayQueue:

    '''FIFO queue implementation using a Python list as underlying storage.'''
    DEFAULT_CAPACITY = 6

    # 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

    def display(self):
        return self._data

    def __len__(self):
        return self._size

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

    def first(self):
        if self.is_empty():
            raise Exception( "Queue is empty" )
        return self._data[self._front]
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
def dequeue(self):
    if self.is_empty():
        raise Exception( "Queue is empty" )
    answer = self._data[self._front]
    self._data[self._front] = None
    self._front = (self._front + 1) % len(self._data)
    self._size -= 1
    return answer

def enqueue(self, e):
    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

def _resize(self, cap):
    old = self._data
    self._data = [None] * cap
    walk = self._front
    for k in range(self._size):
        self._data[k] = old[walk]
        walk = (1 + walk) % len(old)
    self._front = 0
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

OUTPUT:

```
obj = ArrayQueue()  
obj.enqueue(1)  
obj.enqueue(2)  
obj.enqueue(3)  
obj.enqueue(4)  
obj.enqueue(5)  
obj.enqueue(6)  
print(obj.display())
```

[1, 2, 3, 4, 5, 6]

```
obj.dequeue()  
obj.dequeue()  
obj.dequeue()  
print(obj.display())
```

[None, None, None, 4, 5, 6]

```
obj.enqueue(1)  
print(obj.display())
```

[1, None, None, 4, 5, 6]

```
obj.enqueue(2)  
print(obj.display())
```

[1, 2, None, 4, 5, 6]

```
obj.enqueue(3)  
print(obj.display())
```

[1, 2, 3, 4, 5, 6]

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL - 5

Aim: To write a program to search an element from a list. Give user the option to perform Linear or Binary search.

Theory:

A. Linear Search

- We start searching a list for a particular value from the first item in the list.
- We move from item to item from the first item to the last item in the list.
- If the desired item/value is found in the list then the position of that item in the list is returned/printed.
- If the desired item/value is not found in the list then we conclude that the desired item was not present in the list.
- The output result will be the position of the searched element/item in the list.

In programming, the counting of a list or array starts from 0. So, the first element of a list is said to be in 0th position. The second element of the list is said to be in the 1st position and so on.

B. Binary Search

- Binary search algorithm doesn't go on searching in a sequential manner like in linear search.
- It directly moves to the middle item and compares it with the searched item/value.
- If the compared item is same as the searched item then the algorithm terminates.
- Otherwise, if the compared item is greater than the searched item then we can skip entire second half (right half) of the list of the compared item.
- If the compared item is less than the searched item then we can skip entire first half (left half) of the list from the compared item.
- After that, we repeat the above process again for one of the halves of the list.
- This algorithm runs until the sub-list is reduced to zero, i.e. until there is no possibility of dividing the list into two halves.

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

CODE

```
In [1]: lst=['car','pen','crayons','pencil','perfume']  
print(lst)  
  
['car', 'pen', 'crayons', 'pencil', 'perfume']
```

```
In [2]: #Python Program to sort a list using linear search  
def linear_search(lst,n):  
    index_counter = 0  
    list_size = len(lst)  
    while index_counter < list_size:  
        temp_value = lst[index_counter]  
        if temp_value == n:  
            return index_counter  
        index_counter +=1  
    return -1
```

```
In [3]: #Python Program to sort a list using binary search  
def binary_search(lst, n):  
    first = 0  
    last = len(lst)-1  
    index = -1  
    while (first <= last) and (index == -1):  
        mid = (first+last)//2  
        if lst[mid] == n:  
            index = mid  
        else:  
            if n<lst[mid]:  
                last = mid -1  
            else:  
                first = mid +1  
    return index
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

OUTPUT:

```
In [4]: a = input("Enter 1 to perform linear_search,\nEnter 2 to perform binary_search: ")
if a == '1':
    print(linear_search(lst,'crayons'))
elif a == '2':
    print(binary_search(lst,'pen'))
```

```
Enter 1 to perform linear_search,
Enter 2 to perform binary_search: 2
-1
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL - 06

Aim: To write a program to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.

Theory:

A. Bubble Sort

- Bubble Sort is a simple sorting algorithm that compares each element in the list with its adjacent element and sort that pair of elements if they are not in order.
- If there are n items in the list then there will be $(n-1)$ pairs of items that need to be compared in the first pass.
- After the first pass, the largest element of the list will be placed in its proper place.
- Now, in the second pass, there will be $(n-1)$ items left for sorting. This means that there will be $(n-2)$ pairs of items that need to be compared in the second pass.
- In the second pass, the second largest element of the list will be placed in its proper place.
- In this way, other elements of the list are sorted.

B. 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.
 - i. The subarray which is already sorted.
 - ii. Remaining subarray which is unsorted.
- In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

C. Insertion Sort

- Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.
- Iterate over the input elements by growing the sorted array at each iteration.
- Compare the current element with the largest value available in the sorted array.
- If the current element is greater, then it leaves the element in its place and moves on to the next element else it finds its correct position in the sorted array and moves it to that position in the array.
- This is achieved by shifting all the elements towards the right, which are larger than the current element, in the sorted array to one position ahead.

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368
CODE:

```
In [9]: list_students_rolls=[20,5,54,33,64,-9,58]
print(list_students_rolls)
```

```
[20, 5, 54, 33, 64, -9, 58]
[20, 5, 54, 33, 64, -9, 58]
```

```
In [10]: #Python Program to sort a list using bubble sort
def bubble_sort(list_students_rolls):
    for i in range(0,len(list_students_rolls)):
        for j in range(0,len(list_students_rolls)-1):
            if list_students_rolls[j]>list_students_rolls[j+1]:
                list_students_rolls[j],list_students_rolls[j+1] = list_students_rolls[j+1],list_students_rolls[j]
    print("Bubble Sort:",list_students_rolls)
```

```
In [11]: #Python Program to sort a list using selection sort
def selection_sort(list_students_rolls):
    for i in range(len(list_students_rolls)):
        min_val_index = i
        for j in range(i+1,len(list_students_rolls)):
            if list_students_rolls[min_val_index]>list_students_rolls[j]:
                min_val_index = j
        list_students_rolls[i],list_students_rolls[min_val_index] = list_students_rolls[min_val_index],list_students_rolls[i]
    print("Selection Sort:",list_students_rolls)
```

```
In [12]: #Python Program to sort a list using insertion sort
def insertion_sort(list_students_rolls):
    for i in range(len(list_students_rolls)):
        value = list_students_rolls[i]
        j = i-1

        while j>=0 and value < list_students_rolls[j]:
            list_students_rolls[j+1] = list_students_rolls[j]
            j-=1

        list_students_rolls[j+1] = value
    print("Insertion Sort:",list_students_rolls)
```

NAME: Gunveen Bindra

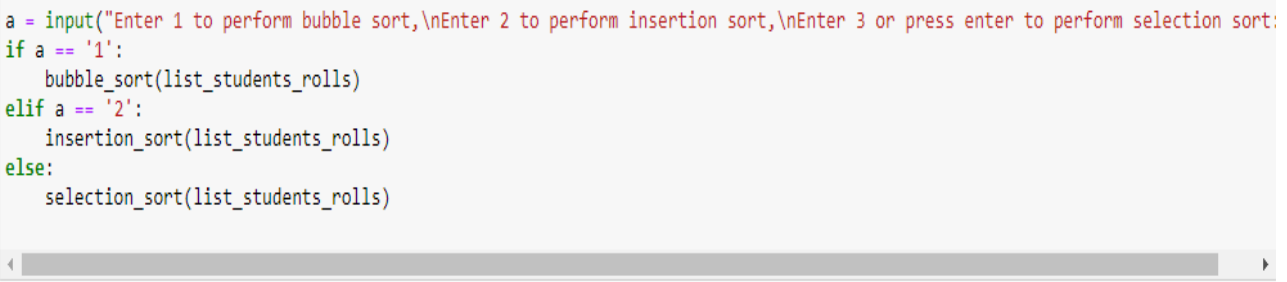
CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

OUTPUT:

```
In [13]: a = input("Enter 1 to perform bubble sort,\nEnter 2 to perform insertion sort,\nEnter 3 or press enter to perform selection sort:")
if a == '1':
    bubble_sort(list_students_rolls)
elif a == '2':
    insertion_sort(list_students_rolls)
else:
    selection_sort(list_students_rolls)
```



```
Enter 1 to perform bubble sort,
Enter 2 to perform insertion sort,
Enter 3 or press enter to perform selection sort: 2
Insertion Sort: [-9, 5, 20, 33, 54, 58, 64]
Insertion Sort: [-9, 5, 20, 33, 54, 58, 64]
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL – 7(A)

Aim: To write a program to implement the collision technique.

Theory:

- A collision occurs when two items or values get the same slot or index, i.e. the hashing function generates same slot number for multiple items.
- If proper collision resolution steps are not taken then the previous item in the slot will be replaced by the new item whenever the collision occurs.

CODE:

```
File Edit Format Run Options Window Help
#Python program to implement the collision technique.

class Hash:
    def __init__(self, keys, lowerrange, higherrange):
        self.value = self.hashfunction(keys, lowerrange, higherrange)

    def get_key_value(self):
        return self.value

#Creating hash function
    def hashfunction(self, keys, lowerrange, higherrange):
        if lowerrange == 0 and higherrange > 0:
            return keys%(higherrange)

if __name__ == '__main__':

    list_of_keys = [48, 21, 54, 18, 56, 62]
    list_of_list_index = [None, None, None, None, None, None]

    print("Before : " + str(list_of_list_index))

    for value in list_of_keys:
        #print(Hash(value, 0, len(list_of_keys)).get_key_value())
        list_index = Hash(value, 0, len(list_of_keys)).get_key_value()
        if list_of_list_index[list_index]:
            print("Collission detected")
        else:
            list_of_list_index[list_index] = value

    print("After: " + str(list_of_list_index))
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

OUTPUT:

```
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [I
4)] on win32
Type "copyright", "credits" or "license()" for more inform
>>>
  RESTART: C:/Users/simra/OneDrive/Desktop/gunveen/SYIT/DS.
Y
Before : [None, None, None, None, None, None]
Collision detected
Collision detected
Collision detected
After: [48, None, 56, 21, None, None]
>>> |
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL – 7(B)

Aim: To write a program to implement the concept of linear probing.

Theory:

- One way to resolve collision is to find another open slot whenever there is a collision and store the item in that open slot.
- The search for open slot starts from the slot where the collision happened.
- It moves sequentially through the slots until an empty slot is encountered.
- The movement is in a circular fashion. It can move to the first slot while searching for an empty slot. Hence, covering the entire hash table.
- This kind of sequential search is called Linear Probing.

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

CODE:

```
File Edit Format Run Options Window Help
#Python program to implement the concept of linear probing.
class Hash:
    def __init__(self, keys, lowerrange, higherrange):
        self.value = self.hashfunction(keys, lowerrange, higherrange)

    def get_key_value(self):
        return self.value

#creates a hash function
    def hashfunction(self, keys, lowerrange, higherrange):
        if lowerrange == 0 and higherrange > 0:
            return keys%(higherrange)

if __name__ == '__main__':
    linear_probing = True
    list_of_keys = [21,46,18,54]
    list_of_list_index = [None, None, None, None]
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        #print(Hash(value,0,len(list_of_keys)).get_key_value())
        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("Collission 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
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001


NEW ROLL NO: 368

```
        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))
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

OUTPUT:

 Python 3.6.5 Shell

```
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [
4)] on win32
Type "copyright", "credits" or "license()" for more infor
>>>
RESTART: C:/Users/simra/OneDrive/Desktop/gunveen/SYIT/DS
Y
Before : [None, None, None, None]
hash value for 21 is :1
hash value for 46 is :2
hash value for 18 is :2
Collision detected for 18
hash value for 54 is :2
Collision detected for 54
After: [54, 21, 46, 18]
>>> |
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

PRACTICAL - 08

Aim: To write a program for inorder, postorder and preorder traversal of tree.

THEORY:

A. Pre Order Traversal

- In this traversal we first visit root, then left, then right.
- In the below python program, we use the Node class to create place holders for the root node as well as the left and right nodes.
- Then we create an insert function to add data to the tree.
- Finally, the Pre-order traversal logic is implemented by creating an empty list and adding the root node first followed by the left node.
- At last the right node is added to complete the Pre-order traversal. Please note that this process is repeated for each sub-tree until all the nodes are traversed.

B. Inorder traversal.

- In order traversal means visiting first left, then root and then right.
- In the below python program, we use the Node class to create place holders for the root node as well as the left and right nodes.
- Then we create an insert function to add data to the tree.
- Finally, the Inorder traversal logic is implemented by creating an empty list and adding the left node first followed by the root or parent node.
- At last the right node is added to complete the Inorder traversal. Please note that this process is repeated for each sub-tree until all the nodes are traversed.

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

C. Post Order traversal

- In this traversal we first visit left, then right and then root.
 - In the below python program, we use the Node class to create place holders for the root node as well as the left and right nodes.
 - Then we create an insert function to add data to the tree.
 - Finally, the Post-order traversal logic is implemented by creating an empty list and adding the left node first followed by the right node.
 - At last the root or parent node is added to complete the Post-order traversal.
- Please note that this process is repeated for each sub-tree until all the nodes are traversed.

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368
CODE:

```
File Edit Format Run Options Window Help
class Node :
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

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

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

    def printPreorder(self):
        if self.val:
            print(self.val)
            if self.left:
                self.left.printPreorder()
            if self.right:
                self.right.printPreorder()
```

NAME: Gunveen Bindra

CLASS: SYIT

OLD ROLL NO: 3001

NEW ROLL NO: 368

```
def printInorder(self):
    if self.val:
        if self.left:
            self.left.printInorder()
        print(self.val)
        if self.right:
            self.right.printInorder()

def printPostorder(self):
    if self.val:
        if self.left:
            self.left.printPostorder()
        if self.right:
            self.right.printPostorder()
        print(self.val)
```

```
root1 = Node(48)
root1.left = Node(64)
root1.right=Node(21)
print('Without any ordering')
root1.PrintTree()
print('Now ordering with insert' )
root1_=Node(None)
root1_.insert(18)
root1_.insert(1)
root1_.PrintTree()
print("Next Node")
root1 = Node(None)
root1.insert(56)
root1.insert(78)
root1.insert(34)
root1.insert(14)
root1.insert(98)
root1.insert(54)
root1.PrintTree()

print("Preorder")
root1.printPreorder()

print("Inorder")
root1.printInorder()

print("Postorder")
root1.printPostorder()
```

NAME: Gunveen Bindra
CLASS: SYIT
OLD ROLL NO: 3001
NEW ROLL NO: 368

OUTPUT:

```
File Edit Shell Debug Options Window Help
Type "copyright", "credits" or "license()" for more inform
>>>
  RESTART: C:\Users\simra\OneDrive\Desktop\gunveen\SYIT\DS\

Without any ordering
64
48
21
Now ordering with insert
1
18
Next Node
14
34
54
56
78
98
Preorder
56
34
14
54
78
98
Inorder
14
34
54
56
78
98
Postorder
14
54
34
98
78
56
```