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: Ms. Siddhi Santosh Shirke

Roll No. 337 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)

Class: S.Y. B.Sc. IT Sem- III **Roll No: 337**

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.	

Practical 1

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:

Array is a container which can hold items and these items should be of the same type. Two important terms of an array are:

1. **Element**– Each item stored in an array is called an element.
2. **Index** – Each location of an element in an array has a numerical index, which is used to identify the element.

    a. Index starts with 0.

    b. Each element can be accessed via its index.

Some of the operations that are performed on array are:

1. Search:We can search an element by the given value or by the given index.In this program the element to be searched is given by the user and thereafter the specific index is return
2. Sort : We can sort array in an order. In this program bubble Sort is used to sort the element of an array
3. Merging : Merging of two arrays is possible .In this program by the append function the second array's element are merge into first array.
4. Reversing : The elements of an array can be also reversed.

Code

```
Array1 =[1,2,3,4,5,6]
print(" Array 1",Array1)
Array1.sort()
print("Sorted Array 1",Array1)

def bubbleSort(arr):
    for i in range (0,len(arr)-1):
        for j in range (0,len(arr)-1):
            if arr[j]>arr[j+1]:
                arr[j],arr[j+1]=arr[j+1],arr[j]
    return arr

def search():
    i =int(input("Enter element to search in the list:"))
    if i in Array1:
        print("The element is in the list at index",Array1.index(i))
    else:
        print("The element is not in the list")
search()
```

```
def  reverse():
    print("Reversing the list :",Array1[::-1])

reverse()

Array2 = [8,9,7,10]
print("List2 :",Array2)

bubbleSort(Array2)
print("Sorted list 2 :",Array2)

def merge_list(arr1,arr2):
    for i in range(len(arr2)):
        arr1.append(arr2[i])
    print("Merging both the lists :",arr1)
merge_list(Array1,Array2)
```

Output

```
 Array 1 [1, 2, 3, 4, 5, 6]
Sorted Array 1 [1, 2, 3, 4, 5, 6]
Enter element to search in the list:6
The element is in the list at index 5
Reversing the list : [6, 5, 4, 3, 2, 1]
List2 : [8, 9, 7, 10]
Sorted list 2 : [7, 8, 9, 10]
Merging both the lists : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Practical 1B

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

Theory:

Matrix is a special case of two dimensional array where each data element is of strictly same size. So every matrix is also a two dimensional array but not vice versa. Matrices are very important data structures for many mathematical and scientific calculations.

Matirx Operations :

Addition : If two matrices have the same dimensions, they may be added together. The result is a new matrix with the same dimensions.

Multiplication: You can only multiply two matrices if their dimensions are compatible , which means the number of columns in the first matrix is the same as the number of rows in the second matrix.. The product of matrices is known as matrix product

Tranpose: The transpose of a matrix is, in effect, the matrix rotated.Here the rows are converted into columns and columns into rows.

Code:

```
def add_matrix(mat1,mat2):
    result = []
    for i in range(len(A)):
        rows = []
        for j in range(len(A[0])):
            rows.append(A[i][j]+B[i][j])

        result.append(rows)
    print("Addition",result)

def multiple_matrix(mat1,mat2):
    result = [ [0,0,0],[0,0,0],[0,0,0]]
    for i in range (len(mat1)):

        for j in range(len(mat2[0])):
                    for k in range(len(mat2)):
                        result[i][j]+= A[i][j]* B[i][j]
    print("Multiplication",result)
```

```
def transpose(matrix):
    result=[ [0,0,0],[0,0,0],[0,0,0]]
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            result[j][i]=matrix[i][j]
    print(result)

A = [[2,3,4],[4,2,1],[8,5,2]]
B = [[3,6,7],[7,2,3],[5,2,8]]
add_matrix(A,B)
transpose(A)
multiple_matrix(A,B)
```

Output:

```
Matrix A [[2, 3, 4], [4, 2, 1], [8, 5, 2]]
Matrix B [[3, 6, 7], [7, 2, 3], [5, 2, 8]]
Addition [[5, 9, 11], [11, 4, 4], [13, 7, 10]]
Transpose [[2, 4, 8], [3, 2, 5], [4, 1, 2]]
Multiplication [[18, 54, 84], [84, 12, 9], [120, 30, 48]]
```

Practical 2

Practical 2a

Aim : To 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.
Types Of Linked List:

1. Singly Linked List
2. Doubly Linked List

A Singly Linked List can be traversed only in forward direction.A Doubly Linked List can be traversed in forward and backwark direction.
Insertion in 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.In this program all three cases are been implemented.
Deletion in Linked List: We can remove an existing node using the key for that node. 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.
Reverse the list: To print the list in reverse order by using the head and previous element we have traverse back.
Merge the list: Two linked list can also be concatenated .

Code:

```
class Node:
    def __init__(self, data,next=None):
        self.prev = None
        self.next = next
        self.data = data
        self.head=None
        self.size = 0
        self.tail = None
    def display(self):
        print(self.data)
    def is_empty(self):
        return self.size == 0
        #ADDING AT START
    def add_front(self,value):
        temp = self.head
        self.head = Node(value)
        self.head.next = temp
        self.size += 1
```

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

#Delete front
    def delete_front(self):
        self.head = self.head.next
        self.head.prev = None
        self.size -= 1

    def find_2nd_last_value(self):
        if (self.size >= 2):
            first = self.head
            temp = self.size-2
            while temp >0:
                first = first.next
                temp -= 1
            return first
```

```
        else:
            print("Size not sufficient")
        return None

#ADDING LAST
    def add_last(self,value):
        new_value = Node(value)
        self.get_tail().next = new_value
        self.size += 1
#DELETE LAST
    def delete_last(self):
        if  (self.is_empty()):
          print  ("Empty list")
        elif self.size == 1:
            self.head == None
            self.size -= 1
        else:
            Node = self.find_2nd_last_value()
            if Node:
                Node.next =None
                self.size -= 1
    def display(self):
        newest = self.head
        while newest:
            print(newest.data,end=',  ')
            newest = newest.next
```

```
    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

#Deleting in between
    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 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
```

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

        if value.data == search_value:
            print(value.data, "  value found at " , " location   ", str(index) )
            return True
        index += 1
    print("Not Found")
    return False
```

```
        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
```

```
obj1 = Node(1)
obj1.add_front(12)
obj1.add_front(11)
obj1.add_front(10)
print("Linked List 1")
obj1.display()
obj = Node(1)

obj.add_front(1)
obj.add_front(2)
obj.add_front(3)
obj.add_front(4)
obj.add_front(5)
obj.add_front(6)

obj.display()
print("Delete from front")
obj.delete_front()
obj.display()
obj.get_tail()
```

```
obj.add_last(7)
obj.add_last(8)
obj.add_last(9)
obj.add_last(10)
obj.display()
print("Delete 10 ")
obj.delete_last()
obj.display()
print("Delete 5th element ")
obj.remove_between_list(5)
obj.search(2)
obj.display()
print("Reversing the list")
obj.reverse_list()
obj.display()
print("Merging two linked list")
obj.merge(obj1)
```

Output:

Linked List 1
10, 11, 12,
Linked List 2
6, 5, 4, 3, 2, 1,
Delete from front
5, 4, 3, 2, 1,
5, 4, 3, 2, 1, 7, 8, 9, 10,
Delete 10
5, 4, 3, 2, 1, 7, 8, 9,
Delete 5th element
2 value found at location 3
5, 4, 3, 2, 1, 8, 9,
Reversing the list
9, 8, 1, 2, 3, 4, 5,
Merging two linked list
9, 8, 1, 2, 3, 4, 5, 10, 11, 12,

Practical 3

Practical 3a

Aim : To Perform Stack operations using Array implementation.

Theory:

Stack inherites Last In First Out (LIFO) feature. In a stack the element insreted last in sequence will come out first as we can remove only from the top of the stack. The operations of adding and removing the elements is known as PUSH and POP.We use append() and pop() to add and remove the element also the on the topmost item can also be displayed after every insertion or deletion.

Code:

```
class Stack_operations:
    def __init__(self):
        self.list = []
    #last in first out
    def is_empty(self):
        if len(self.list)==0:
            return 0
        else:
            return 1
    def  push(self,item):
        self.list.append(item)
        print(self.list)
```

```
    def pop(self):
        if (self.is_empty == 0):
            return "List is empty"
        else:
            self.list.pop()
            print("Removing last in element")
            print(self.list)

    def  top_item(self):
        if (self.is_empty() == 0):
            return "List is empty"
        else:
            print("Element at the top is :",self.list[len(self.list)-1])
```

```
obj=Stack_operations()
obj.push(1)
obj.push(2)
obj.push(3)
obj.push(4)
obj.pop()
obj.top_item()
```

Output:

```
[1]
[1, 2]
[1, 2, 3]
[1, 2, 3, 4]
Removing last in element
[1, 2, 3]
Element at the top is : 3
```

Practical 3b

Aim: To Implement Tower of Hanoi

Theory:

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:
1) Only one disk can be moved at a time.
2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3) No disk may be placed on top of a smaller disk.

Code:

```
def TowerOfHanoi(n , rod1, rod2, aux_rod):
    if n == 1:
        print ("Move disk 1 from rod",rod1,"to rod",rod2)
        return
    TowerOfHanoi(n-1, rod1, aux_rod, rod2)
    print ("Move disk",n,"from rod",rod1,"to rod",rod2)
    TowerOfHanoi(n-1, aux_rod, rod2, rod1)

n = 4
TowerOfHanoi(n, 'A', 'C', 'B')
```

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

Practical 3c

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

Theory:

Polynomials are the algebraic expressions which consist of variables and coefficients.In an algebraic expression if the power of the variables are whole numbers then that algebraic expression is known as polynomial. Linked list is a data structure that stores each element as an object in a node of the list. every note contains two parts data han and links to the next node. In this program Polynomial linked list, the coefficients and exponents of the polynomial are defined as the data node of the list.

Code:

```
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

def display(self):
    if self.size ==0:
        print("no element")
    first = self.head
    i=3
    print(first.element.element,"x ^",i,"+",end="")
    first = first.next
    i=2
    while first:
        if i == 0 :
            print(first.element,"x ^",i)
        else:
            print(first.element,"x ^",i," ",end="+" )
        i=i-1
        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 add_tail(self,e):
    new_value = Node(e)
    self.get_tail().next = new_value
    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
```

```
polynomial_order = 3

poly1 = LinkedList()
print("Polynomial 1:")

poly1.add_head(Node(int(input("coefficient for power {} : ".format(polynomial_order)))))
for i in reversed(range(polynomial_order)):
    poly1.add_tail(int(input("coefficient for power {} : ".format(i))))

print("\n")
poly2 = LinkedList()
print("Polynomial 2")

poly2.add_head(Node(int(input("coefficient for power {} : ".format(polynomial_order)))))
for i in reversed(range(polynomial_order)):
   poly2.add_tail(int(input("coefficient for power  {}: ".format(i))))

print("Adding cooeficients of polynomial 1 and 2 ")
poly1.display()

poly2.display()
print(poly1.get_node_at(0).element + poly2.get_node_at(0).element, "x^3 + " ,
      poly1.get_node_at(1).element + poly2.get_node_at(1).element, "x^2 + ",
      poly1.get_node_at(2).element + poly2.get_node_at(2).element, "x + ",
      poly1.get_node_at(3).element + poly2.get_node_at(3).element)
```

Output:

```
Polynomial 1:
coefficient for power 3 : 5
coefficient for power 2 : 4
coefficient for power 1 : 8
coefficient for power 0 : 2


Polynomial 2
coefficient for power 3 : 2
coefficient for power  2: 4
coefficient for power  1: 6
coefficient for power  0: 5
Adding cooeficients of polynomial 1 and 2
5 x ^ 3 +4 x ^ 2  +8 x ^ 1  +2 x ^ 0
2 x ^ 3 +4 x ^ 2  +6 x ^ 1  +5 x ^ 0
7 x^3 +  8 x^2 +  14 x +  7
```

Practical 3D

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

Theory:

Python also accepts function recursion, which means a defined function can call itself. Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.Iteration is repeated execution of a set of statements while Recursion is a way of programming in which function call itself until it reaches some satisfactory condition.

Code:

```
#Factorial and factors using iteration
num = int(input("Enter a number: "))
factorial = 1
factors = []
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
      if (num%i )==0:
         factors.append(i)
   print("Factorial is   using iteration of ",num," is ",factorial)
   print("Factors using iteration of  ",num," is ", factors)
```

```
# Factorial of a number using recursion

def factorial_recursion(n):
  if n == 1:
     return n
  else:
     return n*factorial_recursion(n-1)
num = int(input("Enter a number: "))

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
      if (num%i )==0:
         factors.append(i)
   print("Factorial using recursion of ",num,"  is ",factorial_recursion(num))
```

```
def factors_recursion(x,i):
    if i==x+1:
       return
    if x%i == 0:
       print(i)
    return factors_recursion(x,i+1)
print("Factors of 10 is")
factors_recursion(10,1)
```

Output:

```
Enter a number: 5
Factorial is   using iteration of  5   is  120
Factors using iteration of   5  is  [1, 5]
Enter a number: 5
Factorial using recursion of  5   is  120
Factors of 10 is
1
2
5
10
```

Practical4

Aim : To Perform Queues operations using Circular Array implementation.

Theory:

An array is called circular if the first element is next pointer of last element. Circular arrays are used to implement queue.
Here First In First Out concept is implemented. The Queue operations such dequeue and enqueue is implemented which means adding element to the first and removing element from the first. Also after every insertion and deletion we can also trace the first and last element in the list i.e. the next element to be removed and the last element to be removed can be traced

Code:

```
#First in first out

class Queue_operation:
    def __init__(self):
        self.head=None
        self.tail=None
        self.data =[]
        print("Original List")
        print(self.data)


    def is_empty():
        if len(self.data)==0:
            return 0
        else:
            return 1
```

```
def dequeue(self):
    if  (self.is_empty)==0:
        print ("List is Empty")
    else:
        print("First element",self.data[0])
        self.head=self.data[1]
        self.data.pop(0)
        print("Removed first element")
        print(self.data)
        print("Head = > ",self.head)

def enqueue(self,e):
    print("Adding  to the last")
    self.data.append(e)
    print(self.data)
    self.head=self.data[0]
    self.tail=e
    print("Tail = > ",self.tail)
    print("Head => ",self.head)
```

```
def get_first_element(self):
    print("Element  to be removed next will be",self.data[0])

def get_last_element(self):
    print("Element  to be removed last will be",self.data[-1])
```

```
ob =Queue_operation()
ob.is_empty
ob.enqueue(1)
ob.enqueue(2)
ob.enqueue(3)
ob.enqueue(4)
ob.dequeue()
ob.enqueue(5)
ob.enqueue(6)
ob.enqueue(7)
ob.dequeue()
ob.enqueue(8)
ob.dequeue()
ob.dequeue()
ob.get_first_element()
ob.get_last_element()
```

Output:

```
Original List
[]
Adding  to the last
[1]
Tail = >  1
Head =>  1
Adding  to the last
[1, 2]
Tail = >  2
Head =>  1
Adding  to the last
[1, 2, 3]
Tail = >  3
Head =>  1
Adding  to the last
[1, 2, 3, 4]
Tail = >  4
```

```
Head =>  1
First element 1
Removed first element
[2, 3, 4]
Head = >  2
Adding  to the last
[2, 3, 4, 5]
Tail = >  5
Head =>  2
Adding  to the last
[2, 3, 4, 5, 6]
Tail = >  6
Head =>  2
Adding  to the last
[2, 3, 4, 5, 6, 7]
Tail = >  7
Head =>  2
First element 2
Removed first element
[3, 4, 5, 6, 7]
```

```
Head = >  3
Adding  to the last
[3, 4, 5, 6, 7, 8]
Tail = >  8
Head =>  3
First element 3
Removed first element
[4, 5, 6, 7, 8]
Head = >  4
First element 4
Removed first element
[5, 6, 7, 8]
Head = >  5
Element  to be removed next will be 5
Element  to be removed last will be 8
```

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:

Linear Search is rarely used practically because other search algorithm such as binary search allow significantly faster search.It is the most basic type of algorithm. A linear search sequentially moves through your collection or data structure looking for matching value.In other words it looks down a list one item at a time ,without jumping.

It starts from the leftmost element of array and one by one compare x with each element of array

If x match with an element return true.

If x does not match with any of element then it reurns -1.

Binary Search requires sorted array. It than repeated divides the sorted array into half .If the number to be searched is equal to the middle element it return the mid item. If the number is greater than the mid element then the number can only lie in right half subarray after the mid element So then we divide the right else if x is smaller we divide the left half and so on.

Code:

```
def  linear_search(nlist,element):
    index_count = 0
    nlist_size = len(nlist)
    while index_count < nlist_size:
        temp = nlist[index_count]
        if temp == element:
            return index_count
        index_count += 1
    return -1

def binary(item,search,start,end):
    if start >end:
        return -1
    mid = (start+end)//2
    if item[mid] == search:
        return mid
    if search < item[mid]:
        return binary(item,search,start,mid-1)
    else:
        return binary(item,search,mid+1,end)
```

```
item_list = ["apple","orange","melon","banana","pineapple"]

choose = str(input("Enter your choice for binary or linear search :  ")).lower()


if choose == "binary":
    search_val = "apple"
    print("Index Number  of  apple",(binary(item_list,search_val,0,len(item_list))))

elif choose == "linear":
    print("Index Number  of pineapple",(linear_search(item_list,"pineapple")))
    #print(linear_search(item_list,"cherry"))
else:
    print("Invalid choice")
```

Output:

```
Enter your choice for binary or linear search :  Binary
Index Number  of  apple 0
>>>
```

```
Enter your choice for binary or linear search :  Linear
Index Number  of pineapple 4
```

Practical 6

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

Theory:

Bubble Sort : It is the simplest sorting algorithm .It checks the adjacent element and if they are not in order it swaps. This process goes until all are in order.

Insertion Sort: In this every number in the list compares itself with the previous number and swaps .It always starts comparing from index[1].here the sorted array is built having one item at a time.

Selection Sort: The selection sort algorithm sorts an array by repeatedly finding the minimum element.(considering ascending order) from the unsorted part and putting it at the beginning The algorithm maintains two subarray:

1. The subarray which is sorted
2. Remaining subarray which is unsorted.

In every iteration of selection sort the minimum element from the unsorted subarray is picked and moved to the sorted subarray.

Code:

```
def bubbleSort(arr):
    for i in range (0,len(arr)-1):
        for j in range (0,len(arr)-1):
            if arr[j]>arr[j+1]:
                arr[j],arr[j+1]=arr[j+1],arr[j]
    return arr


def selection (nlist):
    for i in range (len(nlist)- 1):
        index = 0
        for j in range(len(nlist)-i):
            if nlist[j] > nlist[index]:
                index = j
        nlist[len(nlist)-i-1],nlist[index] = nlist[index],nlist[len(nlist)-i-1]
        return nlist
```

```
def insertion_sort(list1):
    for i in range(1,len(list1)):
        key = list1[i]
        j = i-1
        while j >= 0 and key < list1[j]:
            list1[j+1] = list1[j]
            j=j-1
        list1[j+1]=key
    return list1

alist=[4,0,30,22,-23,67,1,-7]
n_list = ["apple","orange","melon","banana","pineapple"]
print("Before any  sort",alist)
print("Before any sort",n_list)
choose =int(input("Enter choice Bubble Sort as 1,Insertion Sort as 2, Selecction Sort as 3 : "))
if choose == 1 :
   print("After bubble sort",bubbleSort(alist))
   print("After bubble sort",bubbleSort(n_list))
elif choose==2:
   print("After  insertion sort",insertion_sort(alist))
   print("After  insertion sort",insertion_sort(n_list))
elif choose==3:
   print("After selection  sort",selection(alist))
   print("After selection  sort",selection(n_list))
else:
   print("Invalid choice")
```

Output:

```
========== RESTART: C:\Users\Shirke\Documents\DS\practical6.py =
Before any  sort [4, 0, 30, 22, -23, 67, 1, -7]
Before any sort ['apple', 'orange', 'melon', 'banana', 'pineapple']
Enter choice Bubble Sort as 1,Insertion Sort as 2, Selecction Sort as 3 : 1
After bubble sort [-23, -7, 0, 1, 4, 22, 30, 67]
After bubble sort ['apple', 'banana', 'melon', 'orange', 'pineapple']
>>>
```

```
Before any  sort [4, 0, 30, 22, -23, 67, 1, -7]
Before any sort ['apple', 'orange', 'melon', 'banana', 'pineapple']
Enter choice Bubble Sort as 1,Insertion Sort as 2, Selecction Sort as 3 : 2
After  insertion sort [-23, -7, 0, 1, 4, 22, 30, 67]
After  insertion sort ['apple', 'banana', 'melon', 'orange', 'pineapple']
>>>
```

```
Before any  sort [4, 0, 30, 22, -23, 67, 1, -7]
Before any sort ['apple', 'orange', 'melon', 'banana', 'pineapple']
Enter choice Bubble Sort as 1,Insertion Sort as 2, Selecction Sort as 3 : 3
After selection  sort [-7, 4, 0, 22, -23, 30, 1, 67]
After selection  sort ['apple', 'orange', 'melon', 'banana', 'pineapple']
```

Practical 7

Aim : To implement the following for Hashing

Practical 7a

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

Theory:

Hashing is the practice of taking a string or input key, a variable created for storing narrative data, and representing it with a hash value, which is typically determined by an algorithm and constitutes a much shorter string than the original.Hashing is also a method of sorting key values in a database table in an efficient manner. A hash function is used to generate the new value according to a mathematical algorithm. The result of a hash function is known as a hash value or simply, a hash.

Collision :A situation when the resultant hashes for two or more data elements in the data set U, maps to the same location in the has table, is called a hash collision

Code:

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

    def get_key_value(self):
        return self.value

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

if __name__ == '__main__':
    list_of_keys = [23,43,1,87]
    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()
        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))
```

Output:

```
Before : [None, None, None, None]
3
3
Collission detected
1
3
Collission detected
After: [None, 1, None, 23]
```

Practical 7b

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

Theory :

Open Hashing (Separate chaining):

Open Hashing, is a technique in which the data is not directly stored at the hash key index (k) of the Hash table. Rather the data at the key index (k) in the hash table is a pointer to the head of the data structure where the data is actually stored. In the most simple and common implementations the data structure adopted for storing the element is a linked-list. In this technique when a data needs to be searched, it might become necessary (worst case) to traverse all the nodes in the linked list to retrieve the data

Closed hashing (open Addressing):

In this technique a hash table with pre-identified size is considered. All items are stored in the hash table itself. In addition to the data, each hash bucket also maintains the three states: EMPTY, OCCUPIED, DELETED. While inserting, if a collision occurs, alternative cells are tried until an empty bucket is found. For which one of the following technique is adopted.

Linear Probing: 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.

Code:

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

    def get_key_value(self):
        return self.value

    def hashfunction(self,keys,lowerrange, higherrange):
        if lowerrange == 0 and higherrange > 0:
            return keys%(higherrange)
```

```
if __name__ == '__main__':
    linear_probing = True
    list_of_keys = [4,8,10,34]
    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:
```

```
                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))
```

Output:

```
Before : [None, None, None, None]
0
hash value for 4 is :0
0
hash value for 8 is :0
2
hash value for 10 is :2
2
hash value for 34 is :2
After: [None, None, 34, None]
```

Practical 8

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

Theory:

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree –

1. Inorder Traversal
2. Preorder Traversal
3. Postorder Traversal

Inorder : In this traversal method, the left subtree is visited first, then the root and later the right sub-tree

Preorder: In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.

Postorder: In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.

Code:

```
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()
```

```
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)
```

```
root = Node(10)
root.left = Node(12)
root.right=Node(5)
print('Without any ordering')
root.PrintTree()
print('Now ordering with insert' )
root1_=Node(None)
root1_.insert(13)
root1_.insert(4)
root1_.PrintTree()
print("Next Node")
root1 = Node(None)
root1.insert(62)
root1.insert(1)
root1.insert(7)
root1.insert(15)
root1.insert(101)
root1.insert(102)
root1.PrintTree()
```

Output:

Without any ordering
12
10
5
Now ordering with insert
4
13
Next Node
1
7
15
62
101
102

Preorder
62
1
7
15
101
102

Inorder
1
7
15
62
101
102
Postorder
15
7
1
102
101
62