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: Mr._Dhruv Modi_____

Roll No: __357_____ 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: 357**

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 1A

Aim: Implement the following array:- 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.

Repository Link: https://github.com/Dhruv0709/dhruv-modi

Theory:

Searching: You can search an array for a certain value, and return the indexes that get a match. To search an array, use the where() method.

Sorting: Sorting means putting elements in an ordered sequence. Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a specified array.

Merging: There are several ways to join, or concatenate, two or more lists in Python.One of the easiest ways are by using the + operator. Another way to join two lists are by appending all the items from list2 into list1, one by one or you can use the extend() method, which purpose is to add elements from one list to another list.

Reversing: **Using the reversed() built-in function.**
In this method, we neither reverse a list in-place(modify the original list), nor we create any copy of the list. Instead, we get a reverse iterator which we use to cycle through the list.

Using the reverse() built-in function.
Using the reverse() method we can reverse the contents of the list object **in-place** i.e., we don't need to create a new list instead we just copy the existing elements to the original list in reverse order. This method directly modifies the original list.

Using the slicing technique.
In this technique, a copy of the list is made and the list is not sorted in-place. Creating a copy requires more space to hold all of the existing elements. This exhausts more memory.

Code:

```
arr1=[12,15,17,13,14,16]
arr2=['hello','hi','hey']
arr3=[2,4,5,3,1,6]

index=arr1.index(13)
print(index)

arr1.sort()
print(arr1)

arr1.extend(arr2)
print(arr1)

arr3.reverse()
print(arr3)
```

Output:

```
3
[12, 13, 14, 15, 16, 17]
[12, 13, 14, 15, 16, 17, 'hello', 'hi', 'hey']
[6, 1, 3, 5, 4, 2]
```

PRACTICAL 1B

Aim: Write a program to perform a matrix addition, multiplication and transpose operation.

Theory:

Matrix Addition: In Python, we can implement a matrix as a nested list (list inside a list). We can treat each element as a row of the matrix.

For example X = [[1, 2], [4, 5], [3, 6]] would represent a 3x2 matrix. First row can be selected as X[0] and the element in first row, first column can be selected as X[0][0].

We can perform matrix addition in various ways in Python:

1. Matrix Addition using Nested Loop.
2. Matrix Addition using Nested List Comprehension.

Matrix Multiplication: In Python, we can implement a matrix as nested list (list inside a list).

We can treat each element as a row of the matrix.

For example X = [[1, 2], [4, 5], [3, 6]] would represent a 3x2 matrix.

The first row can be selected as X[0]. And, the element in first row, first column can be selected as X[0][0].

Multiplication of two matrices X and Y is defined only if the number of columns in X is equal to the number of rows Y.

If X is a n x m matrix and Y is a m x l matrix then, XY is defined and has the dimension n x l (but YX is not defined). Here are a couple of ways to implement matrix multiplication in Python.

We can perform matrix multiplication in various ways in Python:

1. Matrix Multiplication using Nested Loop.
2. Matrix Multiplication using Nested List Comprehension.

Matrix Transpose: In Python, we can implement a matrix as a nested list (list inside a list). We can treat each element as a row of the matrix.

For example X = [[1, 2], [4, 5], [3, 6]] would represent a 3x2 matrix. The first row can be selected as X[0]. And, the element in the first-row first column can be selected as X[0][0].

Transpose of a matrix is the interchanging of rows and columns. It is denoted as X'. The element at ith row and jth column in X will be placed at jth row and ith column in X'. So if X is a 3x2 matrix, X' will be a 2x3 matrix.

We can perform matrix transpose various ways in Python:

1. Matrix Transpose using Nested Loop.
2. Matrix Transpose using Nested List Comprehension.

Code:

```python
#Python program to perform matrix operations, matrix addition,matrix subtraction,matrix multiplication - addition


mat1 = [[10, 20], [50, 3]]
mat2 = [[3, 2], [40, 2]]
mat3 = [[0, 0], [0, 0]]
for i in range(0, 2):

   for j in range(0, 2):

       mat3[i][j] = mat1[i][j] + mat2[i][j]
       print("Addition of two matrices")
for i in range(0, 2):
  for j in range(0, 2):
       print(mat3[i][j], end = "")
       print()
```

```python
#Python program to perform matrix operations, matrix addition,matrix subtraction,matrix multiplication - subtraction


mat1 = [[10, 20], [50, 3]]
mat2 = [[3, 2], [40, 2]]
mat3 = [[0, 0], [0, 0]]
for i in range(0, 2):

   for j in range(0, 2):

       mat3[i][j] = mat1[i][j] - mat2[i][j]
       print("Subtraction of two matrices")
for i in range(0, 2):
```

```python
   for j in range(0, 2):
        print(mat3[i][j], end = "")
        print()




#Python program to multiply two matrices

mat1 = [[10, 20], [50, 3]]
mat2 = [[3, 2], [40, 2]]
mat3 = [[0, 0], [0, 0]]

for i in range(0, 2):

   for j in range(0, 2):

        mat3[i][j] = mat1[i][j] * mat2[i][j]
        print("Multiplication of two matrices")
for i in range(0, 2):
  for j in range(0, 2):
        print(mat3[i][j], end = "")
        print()
```

Output:

```
Addition of two matrices
Addition of two matrices
Addition of two matrices
Addition of two matrices
13
22
90
5
Subtraction of two matrices
Subtraction of two matrices
Subtraction of two matrices
Subtraction of two matrices
7
18
10
1
Multiplication of two matrices
Multiplication of two matrices
Multiplication of two matrices
Multiplication of two matrices
30
40
2000
6
```

PRACTICAL 2

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

Theory:

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, we have the below scenarios.

Inserting at the Beginning of the Linked List

This involves pointing the next pointer of the new data node to the current head of the linked list. So the current head of the linked list becomes the second data element and the new node becomes the head of the linked list.

Inserting at the End of the Linked List

This involves pointing the next pointer of the the current last node of the linked list to the new data node. So the current last node of the linked list becomes the second last data node and the new node becomes the last node of the 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.

Singly linked lists can be traversed in only forwrad direction starting form the first data element. We simply print the value of the next data element by assgining the pointer of the next node to the current data element.

Code:

```
from array import *
class Stack():
    def __init__(self):
        self.items = array('i',[4,3,2,3110])

    def end(self, item):
        self.items.append(item)
        print(item)

    def peek(self):
        if self.items:
            return self.items[-1]
        else:
            return None

    def size(self):
        if self.items:
            return len(self.items)
        else:
            return None

    def display(self):
        for i in self.items:
            print(i)

    def start(self, i):
        self.items.insert(0, i)
#searching
    def search(self, a):
        l = self.items
        for i in l:
            if i == a:
                print("found Value : ", a)
```

```
                    print("found Value : ", a)
                    break
            else:
                print("not found")

    def traverse(self):
        a = []
        l = self.items
        for i in l:
            a.append(i)
        print(a)
#shorting
    def shoting_element(self):
        #bubble shotting
        nums=self.items
        def sort(nums):
            for i in range(len(nums) - 1, 0, -1):
                for j in range(i):
                    if nums[j] > nums[j + 1]:
                        temp = nums[j]
                        nums[j] = nums[j + 1]
                        nums[j + 1] = temp

        sort(nums)
        print(nums)
    def reverse(self):
        l=self.items
        print(l[::-1])
class merge1(Stack):
    #inheritance is created to merfe two array
    def __init__(self):
        Stack.__init__(self)
        self.items1 = array('i',[4,3,2,1,6])

    def merge(self):
        l = self.items
        l1=self.items1
        a=(l+l1)
        print(a)

s = Stack()
# Inserting the values
s.end(5)
s.end(6)
s.end(7)
s.start(-1)
s.start(-2)
print("search the specific value : ")
s.search(-2)
print("Display the values one by one :")
s.display()
print("peek (End  Value) :", s.peek())
print("treverse the values : ")
s.traverse()
#Shotting element
print("Shotting the values : ")
s.shoting_element()

print("Reversing  the values : ")
s.reverse()
s1=merge1()
print("merge")
s1.merge()
```

Output:

```
5
6
7
search the specific value :
found Value :  -2
Display the values one by one :
-2
-1
4
3
2
3110
5
6
7
peek (End  Value) : 7
treverse the values :
[-2, -1, 4, 3, 2, 3110, 5, 6, 7]
Shotting the values :
array('i', [-2, -1, 2, 3, 4, 5, 6, 7, 3110])
Reversing  the values :
array('i', [3110, 7, 6, 5, 4, 3, 2, -1, -2])
merge
array('i', [4, 3, 2, 3110, 4, 3, 2, 1, 6])
```

PRACTICAL 3A

Aim: Implement the following for stack:- Perform stack operations using Array implementation.

Theory:

A stack data structure can be implemented using a one-dimensional array. But stack implemented using array stores only a fixed number of data values. This implementation is very simple. Just define a one dimensional array of specific size and insert or delete the values into that array by using **LIFO principle** with the help of a variable called **'top'**. Initially, the top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.

push(value) - Inserting value into the stack

In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at top position. Push function takes one integer value as parameter and inserts that value into the stack. We can use the following steps to push an element on to the stack...

- Step 1 - Check whether stack is FULL. (top == SIZE-1)
- Step 2 - If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function.
- Step 3 - If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (stack[top] = value).

pop() - Delete a value from the Stack

In a stack, pop() is a function used to delete an element from the stack. In a stack, the element is always deleted from top position. Pop function does not take any value as parameter. We can use the following steps to pop an element from the stack...

- Step 1 - Check whether stack is EMPTY. (top == -1)
- Step 2 - If it is EMPTY, then display "Stack is EMPTY!!! Deletion is not possible!!!" and terminate the function.
- Step 3 - If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).

display() - Displays the elements of a Stack

We can use the following steps to display the elements of a stack...

- Step 1 - Check whether stack is EMPTY. (top == -1)
- Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.
- Step 3 - If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).
- Step 3 - Repeat above step until i value becomes '0'.

Code:

```
from sys import maxsize


def createStack():
        stack = []
        return stack

def isEmpty(stack):
        return len(stack) == 0


def push(stack, item):
        stack.append(item)
        print(item + " pushed to stack ")


def pop(stack):
        if (isEmpty(stack)):
                return str(-maxsize -1)

        return stack.pop()


def peek(stack):
        if (isEmpty(stack)):
                return str(-maxsize -1)
        return stack[len(stack) - 1]


stack = createStack()
push(stack, str(10))
push(stack, str(20))
push(stack, str(30))
print(pop(stack) + " popped from stack")
```

Output:

```
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
>>>
```

PRACTICAL 3B

Aim: 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.
Note: Transferring the top n-1 disks from source rod to Auxiliary rod can again be thought of as a fresh problem and can be solved in the same manner.

Code:

```
def TowerOfHanoi(n , source, destination, auxiliary):
        if n==1:
                print ("Move disk 1 from source",source,"to destination",destination )
                return
        TowerOfHanoi(n-1, source, auxiliary, destination)
        print ("Move disk",n,"from source",source,"to destination",destination )
        TowerOfHanoi(n-1, auxiliary, destination, source)


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

Output:

```
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B
Move disk 3 from source A to destination C
Move disk 1 from source B to destination A
Move disk 2 from source B to destination C
Move disk 1 from source A to destination C
Move disk 4 from source A to destination B
Move disk 1 from source C to destination B
Move disk 2 from source C to destination A
Move disk 1 from source B to destination A
Move disk 3 from source C to destination B
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B
>>>
```

PRCTICAL 3C

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

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.

Code:

```
def add(A, B, m, n):
        size = max(m, n);
        sum = [0 for i in range(size)]
        for i in range(0, m, 1):
            sum[i] = A[i]

        for i in range(n):
            sum[i] += B[i]
        return sum

def printPoly(poly, n):
        for i in range(n):
            print(poly[i], end = "")
            if (i != 0):
                print("x^", i, end = "")
            if (i != n - 1):
                print(" + ", end = "")
if __name__ == '__main__':
        A = [5, 0, 10, 6]
        B = [1, 2, 4]
        m = len(A)
        n = len(B)

        print("First polynomial is")
        printPoly(A, m)
        print("\n", end = "")
        print("Second polynomial is")
        printPoly(B, n)
        print("\n", end = "")
        sum = add(A, B, m, n)
        size = max(m, n)

        print("sum polynomial is")
        printPoly(sum, size)
```

Output:

```
First polynomial is
5 + 0x^ 1 + 10x^ 2 + 6x^ 3
Second polynomial is
1 + 2x^ 1 + 4x^ 2
sum polynomial is
6 + 2x^ 1 + 14x^ 2 + 6x^ 3
>>>
```

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.

Iterative Solution:
Factorial can also be calculated iteratively as recursion can be costly for large numbers. Here we have shown the iterative approach using both for and while loop.
Using For loop

Code:

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

def factorial(n):

  fact = 1
  for i in range(1, n + 1):
    fact = fact * i
  return fact
if __name__ == '_main_':
    print("The Factorial of", n, "is", factorial(n))

#using iteration
def fact(number):
      fact = 1
      for number in range(5, 1,-1):
           fact = fact * number
              return fact

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

factorial = fact(number)
print("Factorial is "+str(factorial))
```

Output:

```
Enter a number: 6
The factorial of 6 is 720
Enter a number for iteration : 2
Factorial is 120
```

PRACTICAL 4

Aim: Perform Queues operations using Circular array implementation.

Theory:

A Circular Queue is a queue data structure but circular in shape, therefore after the last position, the next place in the queue is the first position.

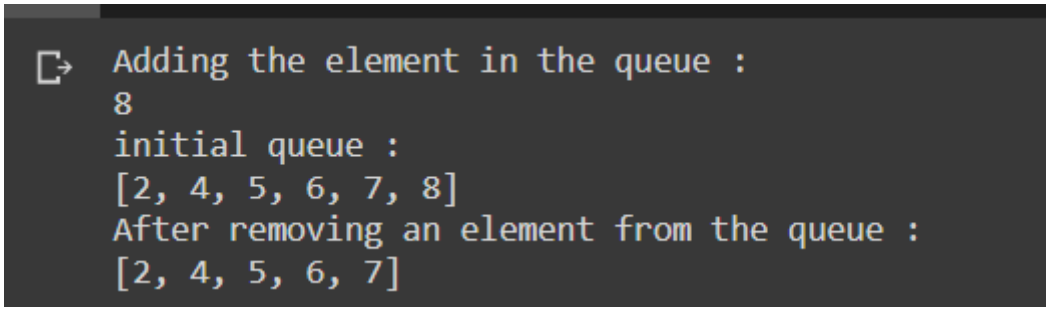
We recommend you to first go through the Linear Queue tutorial before Circular queue, as we will be extending the same implementation.

In case of Linear queue, we did not had the head and tail pointers because we used python **List** for implementing it. But in case of a circular queue, as the size of the queue is fixed, hence we will set a maxSize for our list used for queue implementation.

Code:

```
class Stack():
    def __init__(self):
        self.items = [2,4,5,6,7]



    def enque(self,item):
        self.items.append(item)
        print(item)


    def deque(self):
        b= self.items
        b.pop()
        print(b)

    def traverse(self):
        a = []
        l = self.items
        for i in l:
            a.append(i)
        print(a)

s=Stack()

print("Adding the element in the queue : ")
s.enque(8)
print("initial queue : ")
s.traverse()

print("After removing an element from the queue : ")
s.deque()
```

Output:


Adding the element in the queue :
8
initial queue :
[2, 4, 5, 6, 7, 8]
After removing an element from the queue :
[2, 4, 5, 6, 7]

PRACTICAL 5

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

Theory:

Searching is a very basic necessity when you store data in different data structures. The simplest approach is to go across every element in the data structure and match it with the value you are searching for. This is known as Linear search. It is inefficient and rarely used, but creating a program for it gives an idea about how we can implement some advanced search algorithms.

Linear Search:

In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data structure.

Binary Search:

Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Code:

```
list1 = [1,2,3,4,5,6,7,8,9,10]
print("List = ",list1)
size = len(list1)
def binary_search(x):
    print("BINARY SEARCHING")
    low = 0
    high = len(list1) - 1
    mid = 0
    while low <= high:
        mid = (high + low) // 2
        if list1[mid] < x:
            low = mid + 1
        elif list1[mid] > x:
            high = mid - 1
        else:
            return mid
    return "None it not in the list"
def linear_search(n):
  print("LINEAR SEARCHING")
  if n not in list1:
    print(n,"not in the list")
  else:
    for i in range(size):
      if list1[i]==n:
        print("index of ", n," is ",i)
n = input("Enter (L) for Linear search and  (B) for Binary search :")
if n=="L" or n=="l":
  y = int(input("Enter a no. from the given list1 "))
  linear_search(y)
elif n=="B" or n=="b":
  y = int(input("Enter a no. from the given list1 "))
  print("index of ",y," is ",binary_search(y))
else:
  print("Invalid input")
```

Output:

```
List =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Enter (L) for Linear search and  (B) for Binary search :l
Enter a no. from the given list1 3
LINEAR SEARCHING
index of  3  is  2
```

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:

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.

The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats. Below we see five such implementations of sorting in python.

Bubble Sort

It is a comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

Insertion Sort

Insertion sort involves finding the right place for a given element in a sorted list. So in beginning we compare the first two elements and sort them by comparing them. Then we pick the third element and find its proper position among the previous two sorted elements. This way we gradually go on adding more elements to the already sorted list by putting them in their proper position.

Selection Sort

In selection sort we start by finding the minimum value in a given list and move it to a sorted list. Then we repeat the process for each of the remaining elements in the unsorted list. The next element entering the sorted list is compared with the existing elements and placed at its correct position. So at the end all the elements from the unsorted list are sorted.

Code:

```
nums = [5,4,2,-1]
a = str(input("enter the string i for insertion sort , b for bubble sort , s for selection sort : "))
if a=='i' or a =='I':

    def insertion_sort(nums):
        for i in range(1, len(nums)):
            j = i-1
            nxt_element = nums[i]


            while (nums[j] > nxt_element) and (j >= 0):
                nums[j+1] = nums[j]
                j=j-1
            nums[j+1] = nxt_element


    insertion_sort(nums)
    print(nums)
elif a == 'b' or a == 'B':

    def sort(nums):
        for i in range(len(nums)-1,0,-1):
            for j in range(i):
                if nums[j]>nums[j+1]:
                    temp = nums[j]
                    nums[j]=nums[j+1]
                    nums[j+1] = temp
    sort(nums)
    print(nums)
```

```
elif a == 's' or a =='S':
    def sort(nums):
        for i in range(len(nums)):
            minpos = i
            for j in range(i,len(nums)):
                if nums[j] < nums[minpos]:
                    minpos=j
            temp = nums[i]
            nums[i] = nums[minpos]
            nums[minpos] =temp


    sort(nums)
    print(nums)
else:
    print("Enter valid input")
```

Output:

```
enter the string i for insertion sort , b for bubble sort , s for selection sort
 : b
[-1, 2, 4, 5]
>>>
```

PRACTICAL 7A

Aim: Implement the following for Hashing:- write a program to implement the collision technique.

Theory:

Collision Techniques: When one or more hash values compete with a single hash table slot, collisions occur. To resolve this, the next available empty slot is assigned to the current hash value. The most common methods are open addressing, chaining, probabilistic hashing, perfect hashing and coalesced hashing technique.

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:

        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]
Collission detected
Collission detected
After: [None, 1, None, 23]
>>>
```

PRACTICAL 7B

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

Theory:

Linear probing is a component of open addressing schemes for using a hash table to solve the dictionary problem. In the dictionary problem, a data structure should maintain a collection of key–value pairs subject to operations that insert or delete pairs from the collection or that search for the value associated with a given key. In open addressing solutions to this problem, the data structure is an array *T* (the hash table) whose cells *T*. (when nonempty) each store a single key–value pair. A hash function is used to map each key into the cell of *T* where that key should be stored, typically scrambling the keys so that keys with similar values are not placed near each other in the table. A hash collision occurs when the hash function maps a key into a cell that is already occupied by a different key. Linear probing is a strategy for resolving collisions, by placing the new key into the closest following empty cell.

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 = [23, 43, 1, 87,32,34,67,77,45,54]
    list_of_list_index = [None]*len(list_of_keys)
    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
                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, None, None, None, None, None, None]
hash value for 23 is :3
hash value for 43 is :3
Collission detected for 43
hash value for 1 is :1
hash value for 87 is :7
hash value for 32 is :2
hash value for 34 is :4
Collission detected for 34
hash value for 67 is :7
Collission detected for 67
hash value for 77 is :7
Collission detected for 77
hash value for 45 is :5
Collission detected for 45
hash value for 54 is :4
Collission detected for 54
After: [54, 1, 32, 23, 43, 34, 45, 87, 67, 77]
>>>
```

PRACTICAL 8

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

Theory:

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.

Uses of 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 s reversed can be used.

Uses of 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.

Uses of Postorder
Postorder traversal is used to delete the tree. Please see the question for deletion of tree for details. Postorder traversal is also useful to get the postfix expression of an expression tree.

Code:

```
import random

random.seed(23)

class Node:
    def __init__(self, val):
        self.val = val
        self.leftChild = None
        self.rightChild = None


def insert(root, key):
    if root is None:
        return Node(key)

    else:
        if root.val == key:
            return root
        elif root.val < key:
            root.rightChild = insert(root.rightChild, key)
        else:
            root.leftChild = insert(root.leftChild, key)

    return root


def PrintInorder(root):
    if root:
        PrintInorder(root.leftChild)
        print(root.val, end=" ")
        PrintInorder(root.rightChild)
```

```
def printPreorder(root):
    if root:
        print(root.val, end=" ")
        printPreorder(root.leftChild)
        printPreorder(root.rightChild)


def printPostorder(root):
    if root:
        printPostorder(root.leftChild)
        printPostorder(root.rightChild)
        print(root.val, end=" ")


tree = Node(20)
for i in range(10):
    insert(tree, random.randint(2, 100))


if __name__ == "__main__":
    print("inorder")
    PrintInorder(tree)
    print("\n")
    print("preorder")
    printPreorder(tree)
    print("\n")
    print("postorder")
    printPostorder(tree)
```

Output:

```
inorder
4 12 18 20 39 41 47 50 56 69 77

preorder
20 12 4 18 39 77 41 56 50 47 69

postorder
4 18 12 47 50 69 56 41 77 39 20
>>>
```