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. **BHAVESH .B. BALDANIYA**

Roll No: **306** Programme: BSc CS 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. CS Sem- III **Roll No: 306**

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.	

Git Hub Link: https://github.com/bhaveshbaldaniya8850/DS-Practicals-

Practical 1a

Aim : 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.

Theory:

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

Following are the basic operations supported by an array.

Traverse − print all the array elements one by one.

nsertion − Adds an element at the given index.

Deletion − Deletes an element at the given index.

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

Code:

```
Practical 1a.py
Saved

# Implement the following for 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.
arr1=[12,35,42,22,1,6,54]
arr2=['hello','world']
arr1.index(35)
print(arr1)
arr1.sort()
print(arr1)
arr1.extend(arr2)
print(arr1)
arr1.reverse()
print(arr1)
```

Output:

```
Terminal

[12, 35, 42, 22, 1, 6, 54]
[1, 6, 12, 22, 35, 42, 54]
[1, 6, 12, 22, 35, 42, 54, 'hello', 'world
['world', 'hello', 54, 42, 35, 22, 12, 6,

Process finished.
```

Practical 1b

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

Theory:

add() − add elements of two matrices.

subtract() − subtract elements of two matrices.

divide() − divide elements of two matrices.

multiply() − multiply elements of two matrices.

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

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

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

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

Code:

Practical 1b.py

Saved

```
# Program to add two matrices
X = [[11,7,3],
      [4 ,5,6],
      [7 ,8,9]]

Y = [[5,8,1],
      [6,7,3],
      [4,5,9]]

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)

# Program to multiply two matrices
 # 3x3 matrix
X = [[12,7,3],
     [4 ,5,6],
     [7 ,8,9]]
# 3x4 matrix
Y = [[5,8,1,2],
     [6,7,3,0],
     [4,5,9,1]]
# result is 3x4
result = [[0,0,0,0],
           [0,0,0,0],
           [0,0,0,0]]
 # iterate through rows of X
for i in range(len(X)):
# iterate through columns of Y
    for j in range(len(Y[0])):
# iterate through rows of Y
        for k in range(len(Y)):
            result[i][j] += X[i][k] * Y[k][j]
            for r in result:
                print(r)
```

Practical 1b.py

Saved

```python
# Program to multiply two matrices
 # 3x3 matrix
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]
# 3x4 matrix
Y = [[5,8,1,2],
    [6,7,3,0],
    [4,5,9,1]]
# result is 3x4
result = [[0,0,0,0],
         [0,0,0,0],
         [0,0,0,0]]
 # iterate through rows of X
for i in range(len(X)):
# iterate through columns of Y
    for j in range(len(Y[0])):
# iterate through rows of Y
        for k in range(len(Y)):
            result[i][j] += X[i][k] * Y[k][j]
            for r in result:
                print(r)

# Program to transpose a matrix
X = [[12,7], [4 ,5], [3 ,8]]
result = [[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[j][i] = X[i][j]
        for r in result:
            print(r)
```


Practical 1b.py

Saved

```python
# Program to multiply two matrices
 # 3x3 matrix
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]
# 3x4 matrix
Y = [[5,8,1,2],
    [6,7,3,0],
    [4,5,9,1]]
# result is 3x4
result = [[0,0,0,0],
         [0,0,0,0],
         [0,0,0,0]]
 # iterate through rows of X
for i in range(len(X)):
# iterate through columns of Y
    for j in range(len(Y[0])):
# iterate through rows of Y
        for k in range(len(Y)):
            result[i][j] += X[i][k] * Y[k][j]
            for r in result:
                print(r)
```

Output:

```
Terminal
[16, 0, 0]
[0, 0, 0]
[0, 0, 0]
[16, 15, 0]
[0, 0, 0]
[0, 0, 0]
[16, 15, 4]
[0, 0, 0]
[0, 0, 0]
[16, 15, 4]
[10, 0, 0]
[0, 0, 0]
[16, 15, 4]
[10, 12, 0]
[0, 0, 0]
[16, 15, 4]
[10, 12, 9]
[0, 0, 0]
```

```
Terminal
[74, 97, 73, 14]
[119, 157, 0, 0]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 7, 0]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 31, 0]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 0]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 14]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 14]
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]
[12, 0, 0]
[0, 0, 0]
[12, 0, 0]
[7, 0, 0]
[12, 4, 0]
[7, 0, 0]
[12, 4, 0]
[7, 5, 0]
[12, 4, 3]
[7, 5, 0]
[12, 4, 3]
[7, 5, 8]

Process finished.
```

Practical 2

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

Theory:

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

Insertion in a Linked List

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

Deleting an Item form a Linked List

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

Searching in linked list

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

Reversing a Linked List

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

Concatenating Linked Lists

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

Code:

Practical 2.py

Saved

```
class Stack():
    def __init__(self):
        self.items = ['4','3','2','1','Bhavesh'

    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)

    def search(self, a):
        l = self.items
        for i in l:
            if i == 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)
    def shoting_element(self):
        #bubble shotting
```

Practical 2.py

Saved

```
    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)
    #reverse
    def reverse(self):
        l=self.items
        print(l[::-1])

    def remove_value_from_particular_index(self
        l=self.items
        l.pop(a)
        print(l)

class merge1(Stack):
    #inheritance
    def __init__(self):
        Stack.__init__(self)
        self.items1 = ['4','3','2','1','6']

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

Practical 2.py

Saved

```
s = Stack()
# Inserting the values
s.end('-1')
s.start('-2')
s.start('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()
#reversing the list
print("Reversing  the values :  ")
s.reverse()

print("remove value from particular index whic
s.remove_value_from_particular_index(0)

s1=merge1()
print("merge")
s1.merge()
```

Output:

```
×      Terminal
-1
6
7
search the specific value :
found Value :   -2
Display the values one by one :
-2
-1
5
-2
4
3
2
1
Bhavesh
-1
6
7
peek (End  Value) : 7
treverse the values :
['-2', '-1', '5', '-2', '4', '3', '2', '1'
Shotting the values :
['-1', '-1', '-2', '-2', '1', '2', '3', '4
Reversing  the values :
['Bhavesh', '7', '6', '5', '4', '3', '2',
remove value from particular index which i
['-1', '-2', '-2', '1', '2', '3', '4', '5'
merge
['1', '1', '2', '2', '3', '3', '4', '4', '

Process finished.
```

Practical 3a

Aim: Implement the following for Stack:

a. Perform Stack operations using Array implementation.

Theory:

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

There are two types of operations in Stack-

Push– To add data into the stack.

Pop– To remove data from the stack

Code:

Practical 3a.py
Saved

```python
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:

Terminal

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

Process finished.
```

Practical 3b

Aim: b. Implement Tower of Hanoi

Theory:

We are given n disks and a series of rods, we need to transfer all the disks to the final rod under the given constraints−

We can move only one disk at a time.

Only the uppermost disk

Code:

Practical 3b.py

Saved

```
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:

```
×    Terminal

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

Process finished.
```

Practical 3c

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

Theory:

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

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

For adding two polynomials that are stored as a linked list. We need to add the coefficients of variables with the same power. In a linked list node contains 3 members, coefficient value link to the next node.

a linked list that is used to store Polynomial looks like −

Polynomial : 4x7 + 12x2 + 45

Code:

Practical 3c.py
Saved

```python
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)
```

Practical 3c.py

Saved

```python

    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:

```
×    Terminal
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
Process finished.
```

Practical 3d

Aim: d. 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.

Recursion

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

Iteration

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

Code: Using recursion

```
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))
```

Output:

```
Enter a number: 6
The factorial of 6 is 720
```

Code : Using iterations

```python
def fact(number):

    fact = 1

    for number in range(number, 1,-1):

        fact = fact * number
    return fact

number = int(input("Enter The Number : "))

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

Output:

```
Enter The Number : 7
Factorial is 5040
```

Practical 4

Aim: Perform Queues operations using Circular Array implementation.

Theory:

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

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

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

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

The circular queue work as follows:

two pointers FRONT and REAR

FRONT track the first element of the queue

REAR track the last elements of the queue

initially, set value of FRONT and REARto -1

1.Enqueue Operation

check if the queue is full

for the first element, set value of FRONTto 0

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

add the new element in the position pointed to by REAR

2. Dequeue Operation

check if the queue is empty

return the value pointed by FRONT

circularly increase the FRONT index by 1

for the last element, reset the values of FRONT and REAR to –1

Code:

Practical 4.py

Saved

```
class CircularQueue:

    #Constructor
    def __init__(self):
        self.queue = list()
        self.head = 0
        self.tail = 0
        self.maxSize = 8

    #Adding elements to the queue
    def enqueue(self,data):
        if self.size() == self.maxSize-1:
            return ("Queue Full!")
        self.queue.append(data)
        self.tail = (self.tail + 1) % self.maxSi
        return True

    #Removing elements from the queue
    def dequeue(self):
        if self.size()==0:
            return ("Queue Empty!")
        data = self.queue[self.head]
        self.head = (self.head + 1) % self.maxSi
        return data

    #Calculating the size of the queue
    def size(self):
        if self.tail>=self.head:
            return (self.tail-self.head)
        return (self.maxSize - (self.head-self.t

q = CircularQueue()
print(q.enqueue(1))
print(q.enqueue(2))
print(q.enqueue(3))
print(q.enqueue(4))
print(q.enqueue(5))
print(q.enqueue(6))
print(q.enqueue(7))
print(q.enqueue(8))
print(q.enqueue(9))
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
```

```
Practical 4.py
Saved

            return True

    #Removing elements from the queue
    def dequeue(self):
        if self.size()==0:
            return ("Queue Empty!")
        data = self.queue[self.head]
        self.head = (self.head + 1) % self.maxSi
        return data

    #Calculating the size of the queue
    def size(self):
        if self.tail>=self.head:
            return (self.tail-self.head)
        return (self.maxSize - (self.head-self.t

q = CircularQueue()
print(q.enqueue(1))
print(q.enqueue(2))
print(q.enqueue(3))
print(q.enqueue(4))
print(q.enqueue(5))
print(q.enqueue(6))
print(q.enqueue(7))
print(q.enqueue(8))
print(q.enqueue(9))
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
```

Output:

```
× Terminal
True
True
True
True
True
True
True
Queue Full!
Queue Full!
1
2
3
4
5
6
7
Queue Empty!
Queue Empty!

Process finished.
```

Practical 5

Aim:

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

Theory:

Linear Search:

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

Binary Search:

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

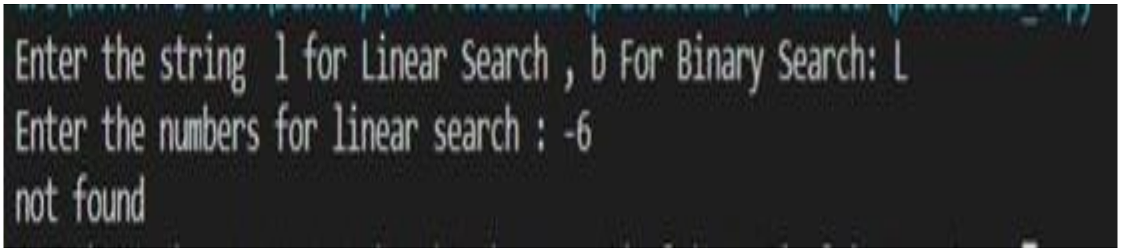
Code:

```python
a = str(input("Enter the string  l for Linear Search , b For Binary Search: "))
pos = -1
list = [0,1,2,3,4,5,6,7,8,45,72]
if a == 'b' or a == 'B':
    def search(list,n):
        l = 0
        u = len(list)-1
        while l <=u:
            mid = (l+u)//2

            if list[mid] == n:
                globals()['pos']= mid
                return True
            else:
                if list[mid]<n:
                    l = mid+1
                else:
                    u = mid-1
        return False

    list.sort()
    n= int(input("Enter the numbers for binary  search : "))
    if search(list, n):
        print("Number Found ")
    else:
        print("Not Found ")
```

```python
elif a == 'l' or a == 'L':
    #pos = -1
    def search(list ,n):
        i = 0

        while i < len(list):
            if list[i] == n:
                return True
            i = i+1

        return False

    list.sort()
    n= int(input("Enter the numbers for linear search : "))
    if search(list ,n):
        print("Number found ")
    else:
        print("not found")

else:
    print("enter valid input")
```

Output:

```
Enter the string  l for Linear Search , b For Binary Search: B
Enter the numbers for binary  search : 5
Number Found
```



```
Enter the string  l for Linear Search , b For Binary Search: L
Enter the numbers for linear search : -6
not found
```

Practical 6

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

Theory:

Bubble Sort:

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

Selection Sort:

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

Insertion Sort:

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

Code:

```
nums = [5,4,4072,-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)
```

```python
    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:

```
Local\Programs\Python\Python37\python.exe' 'c:\Users\BHAVYA D SHAH\.vscode\extensions\m
ers\BHAVYA D SHAH\Desktop\Ds Practicals\practicals\DS-master\Practical_6.py'
enter the string i for insertion sort , b for bubble sort , s for selection sort : i
[-1, 4, 5, 4072]
Local\Programs\Python\Python37\python.exe' 'c:\Users\BHAVYA D SHAH\.vscode\extensions\m
ers\BHAVYA D SHAH\Desktop\Ds Practicals\practicals\DS-master\Practical_6.py'
enter the string i for insertion sort , b for bubble sort , s for selection sort : b
[-1, 4, 5, 4072]
PS C:\Users\BHAVYA D SHAH\Desktop\Ds Practicals\practicals\DS-master>  cd 'c:\Users\BHA
Local\Programs\Python\Python37\python.exe' 'c:\Users\BHAVYA D SHAH\.vscode\extensions\m
ers\BHAVYA D SHAH\Desktop\Ds Practicals\practicals\DS-master\Practical_6.py'
enter the string i for insertion sort , b for bubble sort , s for selection sort : s
[-1, 4, 5, 4072]
PS C:\Users\BHAVYA D SHAH\Desktop\Ds Practicals\practicals\DS-master>  cd 'c:\Users\BHA
Local\Programs\Python\Python37\python.exe' 'c:\Users\BHAVYA D SHAH\.vscode\extensions\m
ers\BHAVYA D SHAH\Desktop\Ds Practicals\practicals\DS-master\Practical_6.py'
enter the string i for insertion sort , b for bubble sort , s for selection sort : q
Enter valid input
PS C:\Users\BHAVYA D SHAH\Desktop\Ds Practicals\practicals\DS-master>
```

Practical 7a

Aim: Implement the following for Hashing

a. Write a program to implement the collision technique

Theory: When the hash value of a key maps to an already occupied bucket of the hash table, it is called as a **Collision**.

Hashing:

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

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

• 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

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

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

Code:

```
Practical 7a.py
Saved

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:

```
×    Terminal

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

Process finished.
```

Practical 7b

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

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

Code:

Practical 7b.py

Saved

```
class Hash:
    def __init__(self, keys, lowerrange, higherr
        self.value = self.hashfunction(keys,lowe

    def get_key_value(self):
        return self.value

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

if __name__ == '__main__':
    linear_probing = True
    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)).g
        list_index = Hash(value,0,len(list_of_ke
        print("hash value for " + str(value) + "
        if list_of_list_index[list_index]:
            print("Collission detected for " + s
            if linear_probing:
                old_list_index = list_index
                if list_index == len(list_of_lis
                    list_index = 0
                else:
                    list_index += 1
                list_full = False
                while list_of_list_index[list_in
                    if list_index == old_list_in
                        list_full = True
                        break
                    if list_index+1 == len(list_
                        list_index = 0
                    else:
                        list_index += 1
                if list_full:
                    print("List was full . Could
                else:
                    list_of_list_index[list_inde
```

Output:

```
×    Terminal

Before : [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 :3
Collission detected for 87
After: [43, 1, 87, 23]

Process finished.
```

Practical 8

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

- **Theory:** 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 traversals reversed can be used.

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

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

Code:

Practical 8.py

Saved

```
class Node:
    def __init__(self,key):
        self.left = None
        self.right = None
        self.val = key



# A function to do inorder tree traversal
def printInorder(root):

    if root:

        # First recur on left child
        printInorder(root.left)

        # then print the data of node
        print(root.val),

        # now recur on right child
        printInorder(root.right)
```

Practical 8.py

Saved

```
# A function to do postorder tree traversal

def printPostorder(root):


    if root:


        # First recur on left child

        printPostorder(root.left)


        # the recur on right child

        printPostorder(root.right)


        # now print the data of node

        print(root.val),




# A function to do preorder tree traversal

def printPreorder(root):


    if root:


        # First print the data of node
```

Practical 8.py
Saved

```python
        # First print the data of node
        print(root.val),

        # Then recur on left child
        printPreorder(root.left)

        # Finally recur on right child
        printPreorder(root.right)



# Driver code
root = Node(1)
root.left       = Node(2)
root.right      = Node(3)
root.left.left  = Node(4)
root.left.right  = Node(5)
print ("Preorder traversal of binary tree is")
printPreorder(root)


print ("\nInorder traversal of binary tree is")
printInorder(root)
```

Output:

Terminal

```
Preorder traversal of binary tree is
1
2
4
5
3

Inorder traversal of binary tree is
4
2
5
1
3

Process finished.
```