MALAD KANDIVALI EDUCATION SOCIETY'S

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

CERTIFICATE

Name:Ms .**ADITI RAJESH PATEKAR**

Roll No: **330**   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:-330**

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 (A)

AIM:

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:

Sorting:

The process of arranging elements of a data structure in an order(ascending or descending) is called sorting.

Merging :The process of combining elements of two data structures is called merging

Searching:

searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored. Based on the type of search operation, these algorithms are generally classified into two categories:

Sequential Search

Interval Search

Code:

```
ds pra1a.py - C:/New folder/ds pra1a.py (3.9.0)
File Edit Format Run Options Window Help
class list_fun:
    def __init__(self):
        self.l = []

    def adding(self,n):
        self.l.append(n)

    def searching(self,f):
        if f in self.l:
            print("found!!")
        else:
            print("not found")

    def sorting(self):
        self.l.sort()

    def reversing(self):
        self.l = self.l[::-1]

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

l1 = list_fun()
l1.adding(8)
l1.adding(3)
l1.adding(4)
l1.adding(1)
l1.adding(9)
l1.adding(2)

l1.searching(3)
l1.sorting()
l1.display()
l1.reversing()
l1.display()
Ln: 37 Col: 0
```

Output:

```
File Edit Shell Debug Options Window Help
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:/New folder/ds pra1a.py =====================
found!!
[1, 2, 3, 4, 8, 9]
[9, 8, 4, 3, 2, 1]
>>>
```

Practical 1(B)

AIM:

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:

File Edit Format Run Options Window Help

```
X = [[12, 7, 3],
     [4, 5, 6],
     [7, 8, 9]]


Y = [[5, 8, 1, 2],
     [6, 7, 3, 0],
     [4, 5, 9, 1]]

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

for i in range(len(X)):
    for j in range(len(Y[0])):
        for k in range(len(Y)):
            result[i][j] += X[i][k] * Y[k][j]

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

OUTPUT:

```
=================== RESTART: C:\Users\Dell\I
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]
>>> 
```

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:

File Edit Format Run Options Window Help

```
class Node:

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

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


class LinkedList:

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

    def __len__(self):
        return self.size

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

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

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

            print(first.element)
            first = first.next

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

File Edit Format Run Options Window Help

```python
        last_object = self.head
        while (last_object.next != None):
            last_object = last_object.next
        return last_object

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

    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

    def search(self, search_value):
        index = 0
        while (index < self.size):
            value = self.get_node_at(index)
            print("Searching at " + str(index) +
                  " and value is " + str(value.element))
            if value.element == search_value:
                print("Found value at " + str(index) + " location")
                return True
            index += 1
        print("Not Found")
        return False

    def merge(self, linkedlist_value):
```

File Edit Format Run Options Window Help

```
            counter += 1
        return element_node

    def search(self, search_value):
        index = 0
        while (index < self.size):
            value = self.get_node_at(index)
            print("Searching at " + str(index) +
                  " and value is " + str(value.element))
            if value.element == search_value:
                print("Found value at " + str(index) + " location")
                return True
            index += 1
        print("Not Found")
        return False

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

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


l1 = Node('AHMAD')
my_list = LinkedList()
my_list.add_head(l1)
my_list.add_tail('KAUSAR')
my_list.add_tail('ZEENAT')
my_list.add_tail('NAZNEEN')


my_list2 = LinkedList()
l2 = Node('ZAINAB')
my_list2.add_head(l2)
my_list2.add_tail('PRIYA')
my_list2.add_tail('KAUS')
my_list2.add_tail('ZEENU')
my_list.display()
my_list.merge(my_list2)
my_list.search('AHMAD')
```

OUTPUT:

```
================ RESTART: C:\Users\Dell
AHMAD
KAUSAR
ZEENAT
NAZNEEN
Searching at 0 and value is AHMAD
Found value at 0 location
>>>
```

Practical 3(A)

AIM:

Perform Stack operations using Array implementation.

THEORY:

Stacks is one of the earliest data structures defined in computer science. In simple words, Stack is a linear collection of items. It is a collection of 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:

File Edit Format Run Options Window Help

```python
class Stack:
    def __init__(self):
        self._data = []

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

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

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

    def top(self):
        if self.is_empty():
            raise Empty('stavk is empty')
        return self._data[-1]

    def pop(self):
        if self.is_empty():
            raise Empty('stavk is empty')
        return self._data.pop()

    def display(self):
        return self._data

s = Stack()
print(s.__len__())
print(s.is_empty())
s.push(1)
s.push(3)
s.push(5)
s.push(8)
print(s.display())
print(s.top())
s.pop()
s.pop()
print(s.display())

print(s.__len__())
print(s.is_empty())
```

OUTPUT:

File Edit Shell Debug Options Window Help

```
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/New folder/ds prac3a.py =====================
0
True
[1, 3, 5, 8]
8
[1, 3]
2
False
>>>
```

Practical 3(C)

AIM:

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 +7In 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:

File Edit Format Run Options Window Help

```
class Node:

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

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


class LinkedList:

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

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

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

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

File Edit Format Run Options Window Help

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

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

    def add_tail(self, e):
        new_value = Node(e)
        new_value.previous = self.get_tail()
        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


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

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

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

OUTPUT:

```
>>>
================== RESTART: C:\Users\Dell\
Enter the order for polynomial : 3
Enter coefficient for power 3 : 2
Enter coefficient for power 2 : 0
Enter coefficient for power 1 : 1
Enter coefficient for power 0 : 2
Enter coefficient for power 3 : 3
Enter coefficient for power 2 : 1
Enter coefficient for power 1 : 0
Enter coefficient for power 0 : 2
5
1
1
4
>>>
```

Practical 3(D)

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.

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:

File Edit Format Run Options Window Help

```
def factorial(number):
    if number < 0:
        print('Invalid entry! Cannot find factorial of a negative number')
        return -1
    if number == 1 or number == 0:
        return 1
    else:
        return number * factorial(number - 1)

def factorial_iteration(number):
    if number < 0:
        print('Invalid entry! Cannot find factorial of a negative number')
        return -1
    fact = 1
    while(number > 0):
        fact = fact * number
        number = number - 1
    return fact

if __name__ == '__main__':
    userInput = 5
    print('Factorial using Recursion of', userInput, 'is:', factorial(userInput))
    print('Factorial using Iteration of', userInput, 'is:', factorial_iteration(userInput))
```

OUTPUT:

File Edit Shell Debug Options Window Help

```
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/New folder/ds prac 3d.py ====================
Factorial using Recursion of 5 is: 120
Factorial using Iteration of 5 is: 120
>>> 
```

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 FRONT to 0

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

add the new element in the position pointed to by REAR

2. Dequeue Operation

check if the queue is empty

return the value pointed by FRONT

circularly increase the FRONT

index by 1for the last element, reset the values of FRONT and REAR to -1

CODE:

File Edit Format Run Options Window Help

```
class Node:
    def __init__(self, element, next=None):
        self.element = element
        self.next = next
        self.previous = None

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


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

    def len(self):
        return self.size

    def get_head(self):
        return self.head

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

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

    def reverse_display(self):
        if self.size == 0:
            print("No element")
            return None
        last = my_list.get_tail()
        print(last.element)
```

File Edit Format Run Options Window Help

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

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

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

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

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

    def find_second_last_element(self):
        if self.size >= 2:
```

File Edit Format Run Options Window Help

```
    def find_second_last_element(self):
        if self.size >= 2:
            first = self.head
            temp_counter = self.size - 2
            while temp_counter > 0:
                first = first.next
                temp_counter -= 1
            return first
        else:
            print("Size not sufficient")
        return None

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

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

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

File Edit Format Run Options Window Help

```python
    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
            next_node.previous = prev_node
            self.size -= 1

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

    def search(self, search_value):
        index = 0
        while (index < self.size):
            value = self.get_node_at(index)
            if type(value.element) == type(my_list.head):
                print("Searching at " + str(index) +
                      " and value is " + str(value.element.element))
            else:
                print("Searching at " + str(index) +
                      " and value is " + str(value.element))
            if value.element == search_value:
                print("Found value at " + str(index) + " location")
```

File Edit Format Run Options Window Help

```
            value = self.get_node_at(index)
            if type(value.element) == type(my_list.head):
                print("Searching at " + str(index) +
                      " and value is " + str(value.element.element))
            else:
                print("Searching at " + str(index) +
                      " and value is " + str(value.element))
            if value.element == search_value:
                print("Found value at " + str(index) + " location")
                return True
            index += 1
        print("Not Found")
        return False

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


l1 = Node('AHMAD')
my_list = LinkedList()
my_list.add_head(l1)
my_list.add_tail('ZEENAT')
my_list.add_tail('KAUSAR')
my_list.add_tail('NAZNEEN')
my_list.get_head().element.element
my_list.add_between_list(2, 'Element between')
my_list.remove_between_list(2)

my_list2 = LinkedList()
l2 = Node('KAUS')
my_list2.add_head(l2)
my_list2.add_tail('KAUSAR')
my_list2.add_tail('AHMAD')
my_list2.add_tail('SABA')
my_list.merge(my_list2)
my_list.get_previous_node_at(3).element
my_list.reverse_display()
```

OUTPUT:

```
'''
================== RESTART: C:\Users\Dell\I
SABA
AHMAD
KAUSAR
KAUS
NAZNEEN
KAUSAR
ZEENAT
AHMAD
>>> |
```

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:

File Edit Format Run Options Window Help

```
def linearSearch(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i


def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0
    while low <= high:
        mid = (high + low) // 2
        if arr[mid] < x:
            low = mid + 1

        elif arr[mid] > x:
            high = mid - 1

        else:
            return mid


arr = [2, 3, 4, 10, 40]
x = 10
opt = input("enter l or b  for searching")
if opt == "b":
    result = binary_search(arr, x)
    print("binary search")
    print(result)

elif opt == "l":
    result = linearSearch(arr, x)
    print("linear search")
    print(result)
```

OUTPUT:

```
>>>
==================== RESTART: C:\Users\Dell\D
enter l or b  for searchingl
linear search
3
>>> |
```

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:

File Edit Format Run Options Window Help

```
def selectionSort(A):
    for i in range(len(A)):
        small = i
        for j in range(i+1, len(A)):
            if A[small] > A[j]:
                small = j
        A[i], A[small] = A[small], A[i]


# insertion sort
def insertionSort(arr):

    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >= 0 and key < arr[j]:
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key


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


A = [64, 25, 12, 22, 11]
opt = input("enter i,b or s for sorting")
if opt == 'i':
    bubbleSort(A)
    print("insertion sort")
    print(A)
elif opt == 'b':
    insertionSort(A)
    print("bubblesort")
    print(A)
elif opt == 's':
    selectionSort(A)
    print("selection sort")
    print(A)
```

OUTPUT:

```
>>>
=================== RESTART: C:\Users\Dell\
enter i,b or s for sortings
selection sort
[11, 12, 22, 25, 64]
>>> |
```

Practical 7(A)

AIM:

Write a program to implement the collision technique.

THEORY:

Hashing:

Hashing is an important Data Structure which is designed to use a specia

l 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:

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:

File Edit Format Run Options Window Help

```python
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:

```
==================== RESTART: C:\Users\Dell\D
Before : [None, None, None, None]
Collission detected
Collission detected
After: [None, 1, None, 23]
>>> 
```

PRACTICAL 7(B)

AIM:

Write a program to implement the concept of linear probing.

CODE:

File Edit Format Run Options Window Help

```
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]
    list_of_list_index = [None, None, None, None]
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        # print(Hash(value,0,len(list_of_keys)).get_key_value())
        list_index = Hash(value, 0, len(list_of_keys)).get_key_value()
        print("hash value for " + str(value) + " is :" + str(list_index))
        if list_of_list_index[list_index]:
            print("Collission detected for " + str(value))
            if linear_probing:
                old_list_index = list_index
                if list_index == len(list_of_list_index)-1:
                    list_index = 0
                else:
                    list_index += 1
                list_full = False
                while list_of_list_index[list_index]:
                    if list_index == old_list_index:
                        list_full = True
                        break
                    if list_index+1 == len(list_of_list_index):
                        list_index = 0
                    else:
                        list_index += 1
                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:

```
...
==================== RESTART: C:\Users\Dell\D
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]
>>> |
```

Practical 8

AIM:

Write a program for inorder, postorders 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 traversal s 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:

File Edit Format Run Options Window Help

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


def printInorder(root):
    if root:
        printInorder(root.left)
        print(root.val),
        printInorder(root.right)


def printPostorder(root):
    if root:
        printPostorder(root.left)
        printPostorder(root.right)
        print(root.val),


def printPreorder(root):
    if root:
        print(root.val),
        printPreorder(root.left)
        printPreorder(root.right)


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("Inorder traversal of binary tree is", printInorder(root))
print("Postorder traversal of binary tree is", printPostorder(root))
```

OUTPUT:

```
===================== RESTART: C:\Users\Dell\Down
1
2
4
5
3
Preorder traversal of binary tree is None
4
2
5
1
3
Inorder traversal of binary tree is None
4
5
2
3
1
Postorder traversal of binary tree is None
>>> |
```