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. Aakash Sanjay Patil

Roll No: 344 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: 344**

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

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.

GitHub Link:

1a: https://github.com/aakashpatil219/DS/blob/master/Practical1a.py

1b: https://github.com/aakashpatil219/DS/blob/master/Practical1b.py

Theory:

Searching: 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.

Sorting: Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order.

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.

Merging: Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity big O(n log n), it is one of the most respected algorithms. Merge sort first divides the array into equal halves and then combines them in a sorted manner.

Reversing: reverse () is an inbuilt method in Python programming language that reverses objects of list in place. Returns: The reverse () method does not return any value but reverse the given object from the list

Code a:

```
Practical1a.py - D:\SYCS-4074\DataStructure\Practical1a.py (3.8.5)
File  Edit  Format  Run  Options  Window  Help
list1 = [32,45,5,19,47,9,13,17,22,62,74,1]
size = len(list1)
def searching(n):
        print("SEARCHING")
        print(list1)
        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)

def sorting():
        print("SORTING")
        print(list1)
        for i in range(len(list1)):
                for j in range(i):
                        if list1[i] < list1[j]:
                                list1[i], list1[j] = list1[j], list1[i]
        print("sorted = ",list1)

def merging():
        print("MERGING")
        list2 = [2,10,26,62,74,76,88,89]
        print("list1 = ",list1)
        print("list2 = ",list2)
        merge = list1+list2
        print("merged list = ",merge)

def revrse():
        print("REVERSED")
        print(list1)
        size = len(list1)-1
        i = 0
        while size >0:
                list1[i], list1[size] = list1[size], list1[i]
                size -= 1
                i += 1
                if size<=i:
                        break
        print(list1)

searching(74)
sorting()
merging()
revrse()
```

Output a:

```
Python 3.8.5 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:\SYCS-4074\DataStructure\Practical1a.py ==============
SEARCHING
[32, 45, 5, 19, 47, 9, 13, 17, 22, 62, 74, 1]
index of  74  is  10
SORTING
[32, 45, 5, 19, 47, 9, 13, 17, 22, 62, 74, 1]
sorted =  [1, 5, 9, 13, 17, 19, 22, 32, 45, 47, 62, 74]
MERGING
list1 =  [1, 5, 9, 13, 17, 19, 22, 32, 45, 47, 62, 74]
list2 =  [2, 10, 26, 62, 74, 76, 88, 89]
merged list =  [1, 5, 9, 13, 17, 19, 22, 32, 45, 47, 62, 74, 2, 10, 26, 62, 74, 76, 88, 89]
REVERSED
[1, 5, 9, 13, 17, 19, 22, 32, 45, 47, 62, 74]
[74, 62, 47, 45, 32, 22, 19, 17, 13, 9, 5, 1]
>>>
```

Code b:

Practical1b.py - D:\SYCS-4074\DataStructure\Practical1b.py (3.8.5)

File Edit Format Run Options Window Help

```
mat1 = [[1, 2], [3, 4]]
mat2 = [[1, 2], [3, 4]]
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()

mat1 = [[7, 4], [6, 4]]
mat2 = [[8, 2], [4, 3]]
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):
  for j in range(0, 2):
       print(mat3[i][j], end = "")
       print()

mat1 = [[7, 4], [6, 4]]
mat2 = [[8, 2], [4, 3]]
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 b:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:\SYCS-4074\DataStructure\Practical1b.py ==============
Addition of two matrices
Addition of two matrices
Addition of two matrices
Addition of two matrices
2
4
6
8
Subtraction of two matrices
Subtraction of two matrices
Subtraction of two matrices
Subtraction of two matrices
-1
2
2
1
Multiplication of two matrices
Multiplication of two matrices
Multiplication of two matrices
Multiplication of two matrices
56
8
24
12
>>>
```

Practical – 2

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

Github Link: https://github.com/aakashpatil219/DS/blob/master/Practical2.py

Theory:

Linked List:

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.

Searching: 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.

Sorting: Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order.

Adding & Removing: Adding elements at top of the list or bottom of the list and removing the elements from top or bottom of the list.

Merging: Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity big O(n log n), it is one of the most respected algorithms. Merge sort first divides the array into equal halves and then combines them in a sorted manner.

Reversing: reverse () is an inbuilt method in Python programming language that reverses objects of list in place. Returns: The reverse () method does not return any value but reverse the given object from the list

Iteration: In Python, the iterative statements are also known as looping statements or repetitive statements. The iterative statements are used to execute a part of the program repeatedly as long as a given condition is true.

Recursion: Recursion can also be seen as self-referential function composition. We apply a function to an argument, then pass that result on as an argument to a second application of the same function, and so on.

Code:

Practical2.py - D:\SYCS-4074\DataStructure\Practical2.py (3.8.5)

File Edit Format Run Options Window Help

```python
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
        self.lst = []

    def __len__(self):
        return self.size

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

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

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

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

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

Practical2.py - D:\SYCS-4074\DataStructure\Practical2.py (3.8.5)

File Edit Format Run Options Window Help

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

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

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

    def 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 rev_lst(self):
        prev = None
        current = self.head
        while(current is not None):
            next = current.next
            current.next = prev
            prev = current
            current = next
        self.head = prev
        print("Reversed Linked List")
        LinkedList.display(self)

    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
```

Practical2.py - D:\SYCS-4074\DataStructure\Practical2.py (3.8.5)

File Edit Format Run Options Window Help

```
            last_node.next = linkedlist_value.head
            self.size = self.size + linkedlist_value.size
        else:
            self.head = linkedlist_value.head
            self.size = linkedlist_value.size


L = LinkedList()
L.add_head(0)
L2 = LinkedList()
L2.add_head(2)
L2.add_head(5)
L2.add_head(8)
L2.add_head(13)
L2.add_head(33)
print("Second Linked List")
L2.display()

print("Insertion at head (Ih) or tail (It)  Deletion at head (Dh) or tail (Dt) Searching (S) Reverse & Display (R)&(D) ConcatList (C) Display list (D) Break from loop (B)")

while True:
    ins = input("Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): ")
    if ins=="Ih" or ins=="ih":
        while True:
            ah = input("enter value to insert at head: ")
            if ah=="B" or ah=="b":
                break
            L.add_head(ah)
            L.display()

    elif ins=="It" or ins=="it":
        while True:
            ah = input("enter value to insert at tail: ")
            if ah=="B" or ah=="b":
                break
            L.add_tail(ah)
            L.display()

    elif ins=="Dh" or ins=="dh":
        L.remove_head()

    elif ins=="Dt" or ins=="dt":
        L.remove_tail()

    elif ins=="S" or ins=="s":
        ah = input("enter value to Serach: ")
        L.search(ah)

    elif ins=="R" or ins=="r":
        L.rev_lst()

    elif ins=="C" or ins=="c":
        print("Merged List")
        L.merge(L2)
        L.display()

    elif ins=="B" or ins=="b":
        break

    elif ins=="D" or ins=="d":
        L.display()

    else:
        continue
```

Output:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: D:\SYCS-4074\DataStructure\Practical2.py ==============
Second Linked List
13
8
5
2
33
Insertion at head (Ih) or tail (It)  Deletion at head (Dh) or tail (Dt) Searching (S)
Reverse & Display (R)&(D) ConcatList (C) Display list (D) Break from loop (B)
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): ih
enter value to insert at head: 1
1
0
enter value to insert at head: 23
23
1
0
enter value to insert at head: 45
45
23
1
0
enter value to insert at head: b
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): it
enter value to insert at tail: 32
45
23
1
0
32
enter value to insert at tail: 54
45
23
1
0
32
54
enter value to insert at tail: b
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): dh
Removing
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): dt
Removing
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): d
23
1
0
32
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): r
Reversed Linked List
32
0
```

Ln: 74 Col: 4

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
enter value to insert at head: 45
45
23
1
0
enter value to insert at head: b
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): it
enter value to insert at tail: 32
45
23
1
0
32
enter value to insert at tail: 54
45
23
1
0
32
54
enter value to insert at tail: b
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): dh
Removing
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): dt
Removing
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): d
23
1
0
32
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): r
Reversed Linked List
32
0
1
23
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): s
enter value to Serach: 1
Searching at 0 and value is 32
Searching at 1 and value is 0
Searching at 2 and value is 1
Found value at 2 location
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): c
Merged List
32
0
1
23
13
8
5
2
33
Enter a Choice (Ih)(It)(Dh)(Dt)(S)(R)(C)(D)(B): b
>>>
```

Ln: 74 Col: 4

Practical – 3

Aim: Implement the following for Stack:

a. Perform Stack operations using Array implementation.

b. Implement Tower of Hanoi (optional)

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

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

GitHub:

3a: https://github.com/aakashpatil219/DS/blob/master/Practical3a.py

3c: https://github.com/aakashpatil219/DS/blob/master/Practical3c.py

3d: https://github.com/aakashpatil219/DS/blob/master/Practical3d.py

Theory:

Stack Operations:

In a stack the element inserted last in sequence will come out first as we can remove only from the top of the stack. Such feature is known as Last in First Out(LIFO) feature. The operations of adding and removing the elements is known as PUSH and POP. In the following program we implement it as add and and remove functions. We dclare an empty list and use the append() and pop() methods to add and remove the data elements.

Linked List:

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 3a:

Practical3a.py - D:\SYCS-4074\DataStructure\Practical3a.py (3.8.5)

File Edit Format Run Options Window Help

```
class Deque:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items ==[]
    def addFront(self,items):
        self.items.append(items)
    def addLast(self,items):
        self.items.insert(0,items)
    def removeFront(self):
        return self.items.pop()
    def removeLast(self):
        return self.items.pop(0)
    def size(self):
        return len(self.items)
    def display(self):
        print(self.items)
D = Deque()
D.addFront(1)
D.addFront(11)
D.addFront(21)
D.addLast(4)
D.addLast(14)
D.addLast(24)
print("adding from head and tail")
D.display()
D.removeFront()
D.removeLast()
print("removing from head and tail")
D.display()
```

Output 3a:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:\SYCS-4074\DataStructure\Practical3a.py ==============
adding from head and tail
[24, 14, 4, 1, 11, 21]
removing from head and tail
[14, 4, 1, 11]
>>>
```

Code 3c:

Practical3c.py - D:\SYCS-4074\DataStructure\Practical3c.py (3.8.5)

File Edit Format Run Options Window Help

```
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 = [7, 2, 10, 6]
        B = [3, 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 3c:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:\SYCS-4074\DataStructure\Practical3c.py ==============
First polynomial is
7 + 2x^ 1 + 10x^ 2 + 6x^ 3
Second polynomial is
3 + 2x^ 1 + 4x^ 2
sum polynomial is
10 + 4x^ 1 + 14x^ 2 + 6x^ 3
>>>
```

Code 3d:

Practical3d.py - D:\SYCS-4074\DataStructure\Practical3d.py (3.8.5)

File Edit Format Run Options Window Help

```
def recur_factorial(n):
    if n == 1:
        return n
    else:
        return n*recur_factorial(n-1)

def rfactor(n,i):
        if i<=n:
                if n%i==0:
                        print(i)
                rfactor(n,i+1)

def itrfact(n):
        fact = 1
        for i in range(1,n+1):
                fact = fact*i
        return fact

def i_factors(n):
    i = 1
    while(i < n+1):
        if n % i == 0:
            print(i)
        i = i + 1

print ("To find the factorial and all factors of a number")
v=input("Enter the (R) for using Recursion and (I) for Iteration:")
if v in ("i", "I", "r", "R"):
        num = int(input("Enter no. "))
        if v=="r" or v=="R":
                print("Factorial ",num," using Recursion")
                print(recur_factorial(num))
                print("factors of ",num," using Recursion")
                rfactor(num,1)
        elif v=="i" or v=="I":
                print("Factorial ",num," using Iteration")
                print(itrfact(num))
                print("factors of ",num," using Iteration")
                i_factors(num)
        else:
                print("Invalid Choice")
else:
        print("Invalid choice")
```

Output 3d:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:\SYCS-4074\DataStructure\Practical3d.py ==============
To find the factorial and all factors of a number
Enter the (R) for using Recursion and (I) for Iteration:R
Enter no. 6
Factorial  6  using Recursion
720
factors of  6  using Recursion
1
2
3
6
>>>
============== RESTART: D:\SYCS-4074\DataStructure\Practical3d.py ==============
To find the factorial and all factors of a number
Enter the (R) for using Recursion and (I) for Iteration:I
Enter no. 5
Factorial  5  using Iteration
120
factors of  5  using Iteration
1
5
>>>
```

Practical – 4

Aim: Perform Queues operations using Circular Array implementation.

GitHub Link: https://github.com/aakashpatil219/DS/blob/master/Practical4.py

Theory:

Before you go ahead and read this tutorial, I highly recommend you to read our previous tutorial on Queues as we will be building off of those concepts. Circular Queues are widely used and are often tested on job interviews. A Circular Queue can be seen as an improvement over the Linear Queue because:

1. There is no need to reset Head and Tail pointers since they reset themselves. This means that once the Head or Tail reaches the end of the Queue, it resets itself to 0.
2. The Tail and Head can point to the same location - this means the Queue is empty
3. The Head can be greater than the Tail or vice-versa. This is possible because the Head and Tail pointers are allowed to cross each other.

Code:

Practical4.py - D:\SYCS-4074\DataStructure\Practical4.py (3.8.5)

File Edit Format Run Options Window Help

```
class MyCircularQueue():
    def __init__(self, k):
        self.k = k
        self.queue = [None] * k
        self.head = self.tail = -1

    # Insert an element into the circular queue
    def enqueue(self, data):
        if ((self.tail + 1) % self.k == self.head):
            print("The circular queue is full\n")
        elif (self.head == -1):
            self.head = 0
            self.tail = 0
            self.queue[self.tail] = data
        else:
            self.tail = (self.tail + 1) % self.k
            self.queue[self.tail] = data

    # Delete an element from the circular queue
    def dequeue(self):
        if (self.head == -1):
            print("The circular queue is empty\n")
        elif (self.head == self.tail):
            temp = self.queue[self.head]
            self.head = -1
            self.tail = -1
            return temp
        else:
            temp = self.queue[self.head]
            self.head = (self.head + 1) % self.k
            return temp

    def printCQueue(self):
        if(self.head == -1):
            print("No element in the circular queue")
        elif (self.tail >= self.head):
            for i in range(self.head, self.tail + 1):
                print(self.queue[i], end=" ")
            print()
        else:
            for i in range(self.head, self.k):
                print(self.queue[i], end=" ")
            for i in range(0, self.tail + 1):
                print(self.queue[i], end=" ")
            print()

obj = MyCircularQueue(5)
obj.enqueue(14)
obj.enqueue(74)
obj.enqueue(76)
obj.enqueue(88)
obj.enqueue(89)
print("Initial queue")
obj.printCQueue()

obj.dequeue()
print("After removing an element from the queue")
obj.printCQueue()
```

Output:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: D:\SYCS-4074\DataStructure\Practical4.py ==============
Initial queue
14 74 76 88 89
After removing an element from the queue
74 76 88 89
>>>
```

Practical – 5

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

GitHub Link: https://github.com/aakashpatil219/DS/blob/master/Practical-5.py

Theory:

Linear search: Linear search is a very basic and simple search algorithm. In Linear search, we search an element or value in a given array by traversing the array from the starting, till the desired element or value is found.

1. Traverse the array using a for loop.
2. In every iteration, compare the target value with the current value of the array.
    - If the values match, return the current index of the array.
    - If the values do not match, move on to the next array element.
3. If no match is found, return -1.

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.

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else if x is greater than the mid element, then x can only lie in right half sub array after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.

Code:

Practical5.py - D:\SYCS-4074\DataStructure\Practical5.py (3.8.5)

File Edit Format Run Options Window Help

```
list1 = [45,9,6,1,74,88,56,34,89,76,99]
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 -1

def lsearching(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 \n ")
if n=="L" or n=="l":
        v = int(input("Enter a no. from the list1 "))
        lsearching(v)
elif n=="B" or n=="b":
        v = int(input("Enter a no. from the list1 "))
        print("index of ",v," is ",binary_search(v))
else:
        print("Invalid input")
```

Output:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
=============== RESTART: D:\SYCS-4074\DataStructure\Practical5.py ==============
List =  [45, 9, 6, 1, 74, 88, 56, 34, 89, 76, 99]
Enter (L) for Linear search and  (B) for Binary search
 L
Enter a no. from the list1 74
LINEAR SEARCHING
index of  74  is  4
>>> 
=============== RESTART: D:\SYCS-4074\DataStructure\Practical5.py ==============
List =  [45, 9, 6, 1, 74, 88, 56, 34, 89, 76, 99]
Enter (L) for Linear search and  (B) for Binary search
 B
Enter a no. from the list1 99
BINARY SEARCHING
index of  99  is  10
>>> 
```

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.

GitHub Link: https://github.com/aakashpatil219/DS/blob/master/Practical-6.py

Theory:

Bubble Sort: Bubble Sort is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements. Bubble Sort compares all the element one by one and sort them based on their values.

If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will swap both the elements, and then move on to compare the second and the third element, and so on.

Selection Sort: Selection sort is conceptually the simplest sorting algorithm. This algorithm will first find the smallest element in the array and swap it with the element in the first position, then it will find the second smallest element and swap it with the element in the second position, and it will keep on doing this until the entire array is sorted.

Insertion sort: Insertion sort is the sorting mechanism where the sorted array is built having one item at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order. The analogy can be understood from the style we arrange a deck of cards. This sort works on the principle of inserting an element at a particular position, hence the name Insertion Sort.

Code:

Practical6.py - D:\SYCS-4074\DataStructure\Practical6.py (3.8.5)

File Edit Format Run Options Window Help

```
list1 = [74,8,81,45,76,79,0,-45,3,27,1,2,34,56]
print("List = ",list1)
n = len(list1)
def bubbleSort():
    print("Bubble Sorting")
    for i in range(n-1):
        for j in range(0, n-i-1):
            if list1[j] > list1[j+1] :
                list1[j], list1[j+1] = list1[j+1], list1[j]
    print(list1)

def SelectionSort():
        print("Selection Sorting")
        for i in range(n):
                for j in range(i):
                        if list1[i]<list1[j]:
                                list1[i],list1[j] = list1[j],list1[i]
        print(list1)

def InsertionSort():
    print("Insertion Sorting")
    for i in range(1, n):
        c = list1[i]
        j = i-1
        while j >=0 and c < list1[j] :
                list1[j+1] = list1[j]
                j -= 1
        list1[j+1] = c
    print(list1)

inp = input("Enter (B) for Bubble Sort, (S) for selection Sort and (I) for Insertion Sort \n")
if inp=="B" or inp=="b":
        bubbleSort()
elif inp=="S" or inp=="s":
        SelectionSort()
elif inp=="I" or inp=="i":
        InsertionSort()
else:
        print("Invalid input")
```

Output:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: D:\SYCS-4074\DataStructure\Practical6.py ==============
List =  [74, 8, 81, 45, 76, 79, 0, -45, 3, 27, 1, 2, 34, 56]
Enter (B) for Bubble Sort, (S) for selection Sort and (I) for Insertion Sort
B
Bubble Sorting
[-45, 0, 1, 2, 3, 8, 27, 34, 45, 56, 74, 76, 79, 81]
>>>
=============== RESTART: D:\SYCS-4074\DataStructure\Practical6.py ==============
List =  [74, 8, 81, 45, 76, 79, 0, -45, 3, 27, 1, 2, 34, 56]
Enter (B) for Bubble Sort, (S) for selection Sort and (I) for Insertion Sort
S
Selection Sorting
[-45, 0, 1, 2, 3, 8, 27, 34, 45, 56, 74, 76, 79, 81]
>>>
=============== RESTART: D:\SYCS-4074\DataStructure\Practical6.py ==============
List =  [74, 8, 81, 45, 76, 79, 0, -45, 3, 27, 1, 2, 34, 56]
Enter (B) for Bubble Sort, (S) for selection Sort and (I) for Insertion Sort
I
Insertion Sorting
[-45, 0, 1, 2, 3, 8, 27, 34, 45, 56, 74, 76, 79, 81]
>>>
```

Practical – 7

Aim: 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.

GitHub Link:

7a: https://github.com/aakashpatil219/DS/blob/master/Practical7a.py

7b: https://github.com/aakashpatil219/DS/blob/master/Practical7b.py

Theory:

Collision technique: Collision Resolution Techniques in data structure are the techniques used for handling collision in hashing. Separate Chaining is a collision resolution technique that handles collision by creating a linked list to the bucket of hash table for which collision occurs.

Concept of linear probing:

Linear probing is s technique for resolving hash collisions of values of hash function.

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.

Linear probing can provide high performance because of its good locality of reference, but is more sensitive to the quality of its hash function than some other collision resolution schemes.

Code 7a:

Practical7a.py - D:\SYCS-4074\DataStructure\Practical7a.py (3.8.5)

File Edit Format Run Options Window Help

```
size_list=int(input("Enter the size of list:"))

def search_from_hash(key,hash_list):
    searched_index=hash_function(key)
    if hash_list[searched_index]:
        print("Value found")
    else:
        print("Value not in list")

def hash_function(value):
    global size_list
    return value%size_list

def map_hash2index(hash_return_value):
    return hash_return_value

def create_hash_table(list_values,main_list):
    for value in list_values:
        hash_return_value=hash_function(value)
        list_index=map_hash2index(hash_return_value)
        if main_list[list_index]:
            print("collision detected")
        else:
            main_list[list_index]=value

list_values =[11,21,31,41,51,61]

main_list=[None for x in range(size_list)]
print(main_list)
create_hash_table(list_values,main_list)
print(main_list)

search_from_hash(30,main_list)
```

Output 7a:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:\SYCS-4074\DataStructure\Practical7a.py ==============
Enter the size of list:6
[None, None, None, None, None, None]
collision detected
collision detected
collision detected
[None, 31, None, 21, None, 11]
Value not in list
>>>
```

Code 7b:

Practical7b.py - D:\SYCS-4074\DataStructure\Practical7b.py (3.8.5)

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 = [74,43,12,87]
    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()
        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 7b:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:\SYCS-4074\DataStructure\Practical7b.py ==============
Before : [None, None, None, None]
hash value for 74 is :2
hash value for 43 is :3
hash value for 12 is :0
hash value for 87 is :3
Collission detected for 87
After: [12, 87, 74, 43]
>>>
```

Practical – 8

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

GitHub Link: https://github.com/aakashpatil219/DS/blob/master/Practical8.py

Theory:

Inorder Traversal: For binary search trees (BST), Inorder Traversal specifies the nodes in non-descending order. In order to obtain nodes from BST in non-increasing order, a variation of inorder traversal may be used where inorder traversal is reversed.

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

Postorder Traversal: Postorder traversal is used to get the postfix expression of an expression given

Inorder:

Traverse the left sub-tree, recursively call inorder(root => left).

Visit and print the root node.

Traverse the right sub-tree, recursively call inorder(root => right).

Preorder:

Visit and print the root node.

Traverse the left sub-tree, recursively call inorder(root => left).

Traverse the right sub-tree, recursively call inorder(root => right).

Postorder:

Traverse the left sub-tree, recursively call inorder(root => left).

Traverse the right sub-tree, recursively call inorder(root => right).

Visit and print the root node.

Code:

Practical8.py - D:\SYCS-4074\DataStructure\Practical8.py (3.8.5)

File Edit Format Run Options Window Help

```
import random

random.seed(20)
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:

Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: D:\SYCS-4074\DataStructure\Practical8.py ==============
inorder
14 20 21 35 43 75 83 88 89 94 100

preorder
20 14 94 89 21 35 88 83 43 75 100

postorder
14 75 43 83 88 35 21 89 100 94 20
>>>
```