DATA STRUCTURES USING PYTHON

LAB MANUAL

B.TECH
(II YEAR – I SEM)
(2022-23)

DEPARTMENT OF INFORMATION TECHNOLOGY

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

INDEX

S.No	Date	Name of the Activity/Experiment	Grade/ Marks	Faculty Signature

DEPARTMENT OF INFORMATION TECHNOLOGY

VISION

- To achieve high quality in technical education that provides the skills and attitude to adapt to the global needs of the Information Technology sector, through academic and research excellence.

MISSION

- To equip the students with the cognizance for problem solving and to improve the teaching learning pedagogy by using innovative techniques.
- To strengthen the knowledge base of the faculty and students with motivation towards possession of effective academic skills and relevant research experience.
- To promote the necessary moral and ethical values among the engineers, for the betterment of the society.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: PROFESSIONALISM & CITIZENSHIP

To create and sustain a community of learning in which students acquire knowledge and learn to apply it professionally with due consideration for ethical, ecological and economic issues.

PEO2: TECHNICAL ACCOMPLISHMENTS

To provide knowledge-based services to satisfy the needs of society and the industry by providing hands on experience in various technologies in core field.

PEO3: INVENTION, INNOVATION AND CREATIVITY

To make the students to design, experiment, analyze, interpret in the core field with the help of other multi-disciplinary concepts wherever applicable.

PEO4: PROFESSIONAL DEVELOPMENT

To educate the students to disseminate research findings with good soft skills and become a successful entrepreneur.

PEO5: HUMAN RESOURCE DEVELOPMENT

To graduate the students in building national capabilities in technology, education and research.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Information Technology, the graduates will have the following Program Specific Outcomes:

1. **Fundamentals and critical knowledge of the Computer System:** Able to understand the working principles of the computer System and its components, Apply the knowledge to build, asses, and analyze the software and hardware aspects of it.

2. **The comprehensive and Applicative knowledge of Software Development:** Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.

3. **Applications of Computing Domain & Research:** Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

PROGRAM OUTCOMES (POs)

Engineering Graduates should possess the following:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design / development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.

12. **Life- long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF INFORMATION TECHNOLOGY

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.

b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.

c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high-end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

HEAD OF THE DEPARTMENT **PRINCIPAL**

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

B.TECH - II- YEAR I-SEM-IT **L/T/P/C**

-/-/3/1.5

(R20A0583) DATA STRUCTURES USING PYTHON LAB

COURSE OBJECTIVES:

1. To understand a range of Object-Oriented Programming, as well as in-depth data and information processing techniques.
2. To understand the how linear data structures works
3. To understand the how non-linear data structures works
4. To understand the how searching sorting works.
5. To understand the how Trees and Graphs works in DS.

1. Write a Python program for class, Flower, that has three instance variables of type str,int, and float, that respectively represent the name of the flower, its number of petals, and its price. Your class must include a constructor method that initializes each variable to an appropriate value, and your class should include methods for setting the value of each type, and retrieving the value of each type.

2. Develop an inheritance hierarchy based upon a Polygon class that has abstract methods area( ) and perimeter( ). Implement classes Triangle, Quadrilateral, Pentagon, that extend this base class, with the obvious meanings for the area( ) and perimeter( ) methods. Write a simple program that allows users to create polygons of the various types and input their geometric dimensions, and the program then outputs their area and perimeter.

3. Develop an inheritance hierarchy based upon a Polygon class that has abstract methods area( ) and perimeter( ). Implement classes Triangle, Quadrilateral, Pentagon,that extend this base class, with the obvious meanings for the area( ) and perimeter( ) methods.write a simple program that allows users to create polygons of the various types and input their geometric dimensions, and the program then outputs their area and perimeter.

4. Write a Python program to illustrate the following comprehensions:
   a) List Comprehensions b) Dictionary Comprehensions
   c) Set Comprehensions d) Generator Comprehensions

5. Write a Python program to generate the combinations of n distinct objects taken from the elements of a given list. **Example:** Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9] Combinations of 2 distinct objects: [1, 2] [1, 3] [1, 4] [1, 5] .... [7, 8] [7, 9] [8, 9].

6. Write a Python program for Linear Search and Binary search
7. Write a program to implement Bubble Sort and Selection Sort
8. Write a program to implement Merge sort and Quick sort
9. Write a program to implement stacks and Queues
10. Write a program to implement linked list
11. Write a program to implement Double Linked list
12. Write a program to implement Binary search Tree

COURSE OUTCOMES:

- To understand a range of Object-Oriented Programming, as well as in-depth data and information processing techniques.
- To understand the how linear data structures works
- To understand the how non-linear data structures works
- To understand the how searching sorting works.
- To understand the how Trees and Graphs works in DS.

Table of Contents

WEEK 1:

Write a Python program for class, Flower, that has three instance variables of type str, int, and float, that respectively represent the name of the flower, its number of petals, and its price. Your class must include a constructor method that initializes each variable to an appropriate value, and your class should include methods for setting the value of each type, and retrieving the value of each type.

Program:

```
class Flower:
#Common base class for all Flowers
        def_init_(self, petalName, petalNumber, petalPrice):self.name = petalName
                self.petals = petalNumberself.price = petalPrice

        def setName(self, petalName):self.name = petalName

        def setPetals(self, petalNumber):self.petals =
                petalNumber

        def setPrice(self, petalPrice):self.price = petalPrice

        def getName(self): return
                self.name

        def getPetals(self): return
                self.petals

        def getPrice(self): return
                self.price


#This would create first object of Flower classf1 = Flower("Sunflower",
2, 1000)
print ("Flower Details:") print ("Name: ",
f1.getName())
print ("Number of petals:", f1.getPetals())print ("Price:",f1.getPrice())

print ("\n")

#This would create second object of Flower classf2 = Flower("Rose", 5,
2000)
f2.setPrice(3333)f2.setPetals(6)
print ("Flower Details:") print ("Name: ",
f2.getName())
print ("Number of petals:", f2.getPetals())print ("Price:",f2.getPrice())
```

Signature of the Faculty

Output:

Exercise Programs:

WEEK 2:

Develop an inheritance hierarchy based upon a Polygon class that has abstract methods area( ) and perimeter( ). Implement classes Triangle, Quadrilateral, Pentagon, that extend this base class, with the obvious meanings for the area( ) and perimeter( ) methods. Write a simple program that allows users to create polygons of the various types and input their geometric dimensions, and the program then outputs their area and perimeter.

Program:

```
from abc import abstractmethod, ABCMetaimport math

class Polygon(metaclass = ABCMeta):
        def____init____(self, side_lengths = [1,1,1], num_sides = 3):self._side_lengths =
                side_lengths
                self._num_sizes = 3

        @abstractmethoddef
        area(self):
                pass

        @abstractmethod
        def perimeter(self):pass

        def____repr____(self):
                return (str(self._side_lengths))



class Triangle(Polygon):
        def____init____(self, side_lengths): super().____
                __init____(side_lengths, 3) self._perimeter =
                self.perimeter()self._area = self.area()


        def perimeter(self): return(sum(self._side_lengths))

        def area(self):

                #Area of Triangle
                s = self._perimeter/2product = s
                for i in self._side_lengths:product*=(s-i)
                return product**0.5


class Quadrilateral(Polygon):
        def____init____(self, side_lengths): super().____
                __init____(side_lengths, 4) self._perimeter =
                self.perimeter()

                self._area = self.area()

        def perimeter(self): return(sum(self._side_lengths))
```

```
    def area(self):
        # Area of an irregular Quadrilateral semiperimeter =
        sum(self._side_lengths) / 2
        return math.sqrt((semiperimeter - self._side_lengths[0]) *(semiperimeter -
                        self._side_lengths[1]) * (semiperimeter - self._side_lengths[2]) *
                        (semiperimeter - self._side_lengths[3]))

class Pentagon(Polygon):
    def__init__(self, side_lengths): super().__
        __init__(side_lengths, 5) self._perimeter =
        self.perimeter()self._area = self.area()

    def perimeter(self): return((self._side_lengths) * 5)

    def area(self):

        # Area of a regular Pentagona =
        self._side_lengths
        return (math.sqrt(5 * (5 + 2 * (math.sqrt(5)))) * a * a) / 4

#object of Triangle
t1 = Triangle([1,2,2]) print(t1.perimeter(), t1.area())

#object of Quadrilateral
q1 = Quadrilateral([1,1,1,1]) print(q1.perimeter(),
q1.area())

#object of Pentagonp1 =
Pentagon(1)
print(p1.perimeter(), p1.area())
```

Output:

Signature of the Faculty

WEEK 3:

Develop an inheritance hierarchy based upon a Polygon class that has abstract methods area( ) and perimeter( ). Implement classes Triangle, Quadrilateral, Pentagon, that extend this base class, with the obvious meanings for the area( ) and perimeter( ) methods. Write a simple program that allows users to create polygons of the various types and input their geometric dimensions, and the program then outputs their area and perimeter.

Program:

```
from abc import abstractmethod, ABCMetaimport math

class Polygon(metaclass = ABCMeta):
        def_init_(self, side_lengths = [1,1,1], num_sides = 3):self._side_lengths = side_lengths
                self._num_sizes = 3

        @abstractmethoddef
        area(self):
                pass

        @abstractmethod
        def perimeter(self):pass

        def _repr__(self):
                return (str(self._side_lengths))

class Triangle(Polygon):
        def_init__(self, side_lengths): super()._init_(side_lengths, 3)
                self._perimeter = self.perimeter()self._area =
                self.area()

        def perimeter(self): return(sum(self._side_lengths))

        def area(self):

                #Area of Triangle
                s = self._perimeter/2product = s
                for i in self._side_lengths:product*=(s-i)
                return product**0.5

class Quadrilateral(Polygon):
        def_init__(self, side_lengths): super()._init_(side_lengths, 4)
                self._perimeter = self.perimeter()

                self._area = self.area()


        def perimeter(self): return(sum(self._side_lengths))

        def area(self):
                # Area of an irregular Quadrilateral semiperimeter =
                sum(self._side_lengths) / 2
```

```
        return math.sqrt((semiperimeter - self._side_lengths[0]) *(semiperimeter -
                self._side_lengths[1]) * (semiperimeter - self._side_lengths[2]) *
                (semiperimeter - self._side_lengths[3]))

class Pentagon(Polygon):
    def__init__(self, side_lengths): super().__init__(side_lengths, 5)
        self._perimeter = self.perimeter()self._area =
        self.area()

    def perimeter(self): return((self._side_lengths) * 5)

    def area(self):

        # Area of a regular Pentagona =
        self._side_lengths
        return (math.sqrt(5 * (5 + 2 * (math.sqrt(5)))) * a * a) / 4

#object of Triangle
t1 = Triangle([1,2,2]) print(t1.perimeter(), t1.area())

#object of Quadrilateral
q1 = Quadrilateral([1,1,1,1]) print(q1.perimeter(), q1.area())

#object of Pentagonp1 =
Pentagon(1)
print(p1.perimeter(), p1.area())
```

Output:

Signature of the Faculty

Exercise Programs:

WEEK 4:

a) Write a Python program that inputs a list of words, separated by whitespace, and outputs how many times each word appears in the list.

Program:

```
words = input('Enter words:\n')
words=words.split(' ')

word_list = set(words)
word_count={}
for word in word_list:
if word != '':
word_count[word]=words.count(word)

print(word_count)
```

Output:

Signature of the Faculty

Exercise Programs:

b) Write a Python program to illustrate the following comprehensions:
   a) List Comprehensions b) Dictionary Comprehensions
   c) Set Comprehensions d) Generator Comprehensions

Comprehensions in Python

Comprehensions in Python provide us with a short and concise way to construct new sequences (such as lists, set, dictionary etc.) using sequences which have been already defined. Python supports the following 4 types of comprehensions:

a) List Comprehensions
b) Dictionary Comprehensions
c) Set Comprehensions
d) Generator Comprehensions

a) List Comprehensions:

List Comprehensions provide an elegant way to create new lists. The following is the basic structure of a list comprehension:

output_list = [output_exp for var in input_list if (var satisfies this condition)]

Note that list comprehension may or may not contain an if condition. List comprehensions can contain multiple **for** (nested list comprehensions).

Example: Suppose we want to create an output list which contains only the even numbers which are present in the input list. Let's see how to do this using *for loop* and *list comprehension* and decide which method suits better.

Using Loop:

```
#Constructing output list WITHOUT using List comprehensions
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]

output_list = []

#Using loop for constructing output list
for var in input_list:
if var % 2 == 0: output_list.append(var)

print("Output List using for loop:", output_list)
```

Output:

Using List Comprehension:

```
# Using List comprehensions
# for constructing output list

input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]

list_using_comp = [var for var in input_list if var % 2 == 0]

print("Output List using list comprehensions:",list_using_comp)
```

Output:

b) Dictionary Comprehensions:

Extending the idea of list comprehensions, we can also create a dictionary using dictionary comprehensions. The basic structure of a dictionary comprehension looks like below.

output_dict = {key:value for (key, value) in iterable if (key, value satisfy this condition)}

Example 1: Suppose we want to create an output dictionary which contains only the odd numbers that are present in the input list as keys and their cubes as values. Let's see how to do this using for loops and dictionary comprehension.

Using Loop:

```
input_list = [1, 2, 3, 4, 5, 6, 7]output_dict = {}

# Using loop for constructing output dictionaryfor var in input_list:
if var % 2 != 0: output_dict[var] = var**3

print("Output Dictionary using for loop:",output_dict)
```

Output:

Using Dictionary Comprehension:

```
# Using Dictionary comprehensions
# for constructing output dictionaryinput_list =

[1,2,3,4,5,6,7]

dict_using_comp = {var:var ** 3 for var in input_list if var % 2 != 0}

print("Output Dictionary using dictionary comprehensions:",dict_using_comp)
```

Output:

Example 2: Given two lists containing the names of states and their corresponding capitals, construct a dictionary which maps the states with their respective capitals. Let’s see how to do this using for loops and dictionary comprehension.

Using Loop:

```
state = ['Gujarat', 'Maharashtra', 'Rajasthan']capital = ['Gandhinagar',
'Mumbai', 'Jaipur']

output_dict = {}

# Using loop for constructing output dictionary
for (key, value) in zip(state, capital): output_dict[key] = value

print("Output Dictionary using for loop:", output_dict)
```

Output:

Using Dictionary Comprehension:

```
# Using Dictionary comprehensions
# for constructing output dictionary

state = ['Gujarat', 'Maharashtra', 'Rajasthan']capital = ['Gandhinagar',
'Mumbai', 'Jaipur']

dict_using_comp = {key:value for (key, value) in zip(state, capital)}

print("Output Dictionary using dictionarycomprehensions:",dict_using_comp)
```

Output:

c) Set Comprehensions:

Set comprehensions are pretty similar to list comprehensions. The only difference between them is that set comprehensions use curly brackets { }. Let's look at the following example to understand set comprehensions.

Example : Suppose we want to create an output set which contains only the even numbers that are present in the input list. Note that set will discard all the duplicate values. Let's see how we can do this using for loops and set comprehension.

Using Loop:

```
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]

output_set = set()

# Using loop for constructing output set
for var in input_list:
    if var % 2 == 0:
        output_set.add(var)

print("Output Set using for loop:", output_set)
```

Output:

Using Set Comprehension:

```
# Using Set comprehensions
# for constructing output set

input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]

set_using_comp = {var for var in input_list if var % 2 == 0}print("Output Set using set

comprehensions:",set_using_comp)
```

Output:

d) Generator Comprehensions:

Generator Comprehensions are very similar to list comprehensions. One difference between them is that generator comprehensions use circular brackets whereas list comprehensions use square brackets. The major difference between them is that generators don’t allocate memory for the whole list. Instead, they generate each value one by one which is why they are memory efficient. Let’s look at the following example to understand generator comprehension:

```
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]

output_gen = (var for var in input_list if var % 2 == 0)

print("Output values using generator comprehensions:", end = ' ')

for var in output_gen:
    print(var, end = ' ')
```

Output:

Signature of the Faculty

Exercise Programs:

WEEK 5:

Write a Python program to generate the combinations of n distinct objects taken from the elements of a given list. **Example:** Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9] Combinations of 2 distinct objects: [1, 2] [1, 3] [1, 4] [1, 5] [7, 8] [7, 9] [8, 9].

Program:

```
def combination(n, n_list):if n<=0:
        yield []return
    for i in range(len(n_list)):c_num = n_list[i:i+1]
        for a_num in combination(n-1, n_list[i+1:]):yield c_num + a_num
n_list = [1,2,3,4,5,6,7,8,9]
print("Original list:")print(n_list)
n = 2
result = combination(n, n_list) print("\nCombinations of",n,"distinct
objects:")for e in result:
     print(e)
```

Output:

Signature of the Faculty

Exercise Programs:

WEEK 6:

Write a program for Linear Search and Binary search

Linear Search Program:

```
def linearSearch(target, List):position = 0
        global iterationsiterations =
        0
        while position < len(List):iterations += 1
                if target == List[position]:return position
                position += 1
        return -1

if _name___== '__main__':
        List = [1, 2, 3, 4, 5, 6, 7, 8]
        target = 3
        answer = linearSearch(target, List)if answer != -1:
                print('Target found at index :', answer, 'in',iterations,'iterations')
        else:
                print('Target not found in the list')
```

Output:

Binary Search Program:

```
def binarySearch(target, List):

    left = 0
    right = len(List) - 1global
    iterations iterations = 0

    while left <= right:iterations +=
        1
        mid = (left + right) // 2if target ==
        List[mid]:
            return mid
        elif target < List[mid]:right = mid - 1
        else:
            left = mid + 1return -1

if _name___== '__main__':
    List = [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14]
    target = 12
    answer = binarySearch(target, List)if(answer != -1):
        print('Target',target,'found at position', answer, 'in',iterations,'iterations')
    else:
        print('Target not found')
```

Output:

Signature of the Faculty

Exercise Programs:

WEEK 7:

Write a program to implement Bubble Sort and Selection Sort

Bubble Sort Program:

```
def bubble_sort(alist):
    for i in range(len(alist) - 1, 0, -1):no_swap = True
        for j in range(0, i):
            if alist[j + 1] < alist[j]:
                alist[j], alist[j + 1] = alist[j + 1], alist[j]no_swap = False
        if no_swap:
            return


alist = input('Enter the list of numbers: ').split()alist = [int(x) for x in alist]
bubble_sort(alist) print('Sorted list: ', alist)
```

Output:

Selection Sort Program:

```
def selection_sort(alist):
    for i in range(0, len(alist) - 1):smallest = i
        for j in range(i + 1, len(alist)):if alist[j] <
            alist[smallest]:
                smallest = j
        alist[i], alist[smallest] = alist[smallest], alist[i]


alist = input('Enter the list of numbers: ').split()alist = [int(x) for x in alist]
selection_sort(alist)
print('Sorted list: ', alist)
```

Output:

Signature of the Faculty

Exercise Programs:

WEEK 8:

Write a program to implement Merge sort and Quick sort

Merge Sort Program:

```
def merge_sort(alist, start, end):
        '''Sorts the list from indexes start to end - 1 inclusive.'''if end - start > 1:
                mid = (start + end)//2 merge_sort(alist, start, mid)
                merge_sort(alist, mid, end) merge_list(alist, start, mid,
                end)

def merge_list(alist, start, mid, end):left = alist[start:mid]
        right = alist[mid:end]k = start
        i = 0
        j = 0
        while (start + i < mid and mid + j < end):if (left[i] <= right[j]):
                        alist[k] = left[i]i = i + 1
                else:
                        alist[k] = right[j]j = j + 1
                k = k + 1
        if start + i < mid:while k <
                end:
                        alist[k] = left[i]i = i + 1
                        k = k + 1
        else:
                while k < end:
                        alist[k] = right[j]j = j + 1
                        k = k + 1


alist = input('Enter the list of numbers: ').split()alist = [int(x) for x in alist]
merge_sort(alist, 0, len(alist))print('Sorted list: ', alist)
```

Output:

Quick Sort Program:

```
def quicksort(alist, start, end):
    '''Sorts the list from indexes start to end - 1 inclusive.'''if end - start > 1:
        p = partition(alist, start, end)quicksort(alist, start, p)
        quicksort(alist, p + 1, end)


def partition(alist, start, end):pivot = alist[start]
    i = start + 1j = end - 1

    while True:
        while (i <= j and alist[i] <= pivot):i = i + 1
        while (i <= j and alist[j] >= pivot):j = j - 1

        if i <= j:
            alist[i], alist[j] = alist[j], alist[i]else:
            alist[start], alist[j] = alist[j], alist[start]return j


alist = input('Enter the list of numbers: ').split()alist = [int(x) for x in alist]
quicksort(alist, 0, len(alist))print('Sorted list: ', alist)
```

Output:

Signature of the Faculty

Exercise Programs:

WEEK 9:

Write a program to implement Stacks and Queues

Stack Program:

```
# Custom stack implementation in Pythonclass Stack:

    # Constructor to initialize the stackdef_init_(self, size):
        self.arr = [None] * sizeself.capacity =
        size self.top = -1

    # Function to add an element `x` to the stackdef push(self, x):
        if self.isFull():
            print("Stack Overflow!! Calling exit()…")exit(1)

        print("Inserting", x, "into the stack…")self.top = self.top + 1
        self.arr[self.top] = x

    # Function to pop a top element from the stackdef pop(self):
        # check for stack underflowif
        self.isEmpty():
            print("Stack Underflow!! Calling exit()…")exit(1)

        print("Removing", self.peek(), "from the stack")

        #decrease stack size by 1 and (optionally) return the popped element
        top = self.arr[self.top]self.top = self.top - 1
        return top

    # Function to return the top element of the stackdef peek(self):
        if self.isEmpty():exit(1)
        return self.arr[self.top]

    # Function to return the size of the stackdef size(self):
        return self.top + 1

    # Function to check if the stack is empty or notdef isEmpty(self):
        return self.size() == 0


# Function to check if the stack is full or notdef isFull(self):
        return self.size() == self.capacity

if _name___== '__main__':

    stack = Stack(3)
stack.push(1)
stack.push(2)
stack.pop()
stack.pop()
stack.push(3)
```

```
print("Top element is", stack.peek())
print("The stack size is", stack.size())
stack.pop()
# check if the stack is emptyif stack.isEmpty():
        print("The stack is empty")else:
        print("The stack is not empty")
```

Output:

Queue Program:

```
# Custom queue implementation in Pythonclass Queue:
    # Initialize queue
    def _init__(self, size):
        self.q = [None] * size                # list to store queue elements self.capacity =
        size                                  # maximum capacity of the queueself.front =
        0 # front points to the front element in the queue self.rear = -1 # rear points to the last element in the
        queue self.count = 0 # current size of the queue


    # Function to dequeue the front elementdef pop(self):
        # check for queue underflowif
        self.isEmpty():
            print("Queue Underflow!! Terminating process.")exit(1)

        print("Removing element…", self.q[self.front])

        self.front = (self.front + 1) % self.capacityself.count = self.count - 1


    # Function to add an element to the queuedef append(self, value):
        # check for queue overflowif
        self.isFull():
            print("Overflow!! Terminating process.")exit(1)

        print("Inserting element…", value)

        self.rear = (self.rear + 1) % self.capacityself.q[self.rear] = value
        self.count = self.count + 1


    # Function to return the front element of the queuedef peek(self):
        if self.isEmpty():
            print("Queue UnderFlow!! Terminating process.")exit(1)

        return self.q[self.front]


    # Function to return the size of the queuedef size(self):
return self.count

# Function to check if the queue is empty or notdef isEmpty(self):
        return self.size() == 0


   # Function to check if the queue is full or notdef isFull(self):
        return self.size() == self.capacity


if _name___== '__main__':
```

```
# create a queue of capacity 5q = Queue(5)

q.append(1)
q.append(2)
q.append(3)

print("The queue size is", q.size()) print("The front element is",
q.peek())q.pop()
print("The front element is", q.peek())

q.pop()
q.pop()

if q.isEmpty():
        print("The queue is empty")else:
        print("The queue is not empty")
```

Output:

Signature of the Faculty

Exercise Programs:

WEEK 10:

Write a program to implement Singly Linked List

Program:

```
import os
from typing import NewType


class _Node:'''
        Creates a Node with two fields:
        1.  element (accesed using ._element)
        2.  link (accesed using ._link)'''
        _slots_= '_element', '_link'

        def_init__(self, element, link):'''
                Initialses _element and _link with element and link respectively.
                '''
                self._element = elementself._link = link


class LinkedList:'''
        Consists of member funtions to perform differentoperations on the linked list.
        '''

        def_init__(self):'''
                Initialses head, tail and size with None, None and 0 respectively.
                '''
                self._head  =  None self._tail
                = Noneself._size = 0

        def_len__(self):'''
                Returns length of linked list'''
                return self._size

        def isempty(self):'''
                Returns True if linked list is empty, otherwise False.'''
                return self._size == 0

        def addLast(self, e):'''
                Adds the passed element at the end of the linked list.'''
                newest = _Node(e, None)

                if self.isempty(): self._head = newest
                else:
                        self._tail._link = newest

                self._tail = newestself._size += 1
```

```
def addFirst(self, e):'''
        Adds the passed element at the beginning of the linked list.
        '''
        newest = _Node(e, None)

        if self.isempty(): self._head =
                newestself._tail = newest
        else:
                newest._link = self._headself._head = newest
        self._size += 1

def addAnywhere(self, e, index):'''
        Adds the passed element at the passed index position of the linked list.
        '''
        newest = _Node(e, None)

        i = index - 1 p =
        self._head

        if self.isempty(): self.addFirst(e)
        else:
                for i in range(i):p = p._link
                newest._link = p._linkp._link =
                newest
                print(f"Added Item at index {index}!\n\n")self._size += 1


def removeFirst(self):'''
        Removes element from the beginning of the linked list.Returns the removed element.
        '''
        if self.isempty():
                print("List is Empty. Cannot perform deletionoperation.")
                return
        e = self._head._element self._head =
        self._head._linkself._size = self._size - 1

        if self.isempty(): self._tail =
                None

        return e

def removeLast(self):'''
        Removes element from the end of the linked list.Returns the removed element.
        '''
        if self.isempty():
                print("List is Empty. Cannot perform deletionoperation.")
                return

        p = self._head
        if p._link == None:e =
                p._element
                self._head = Noneelse:
                while p._link._link != None:p = p._link
                e = p._link._elementp._link =
```

```
            None self._tail = p

        self._size = self._size - 1return e

    def removeAnywhere(self, index):'''
        Removes element from the passed index position of the linked list.Returns the removed element.
        '''
        p = self._headi =
        index - 1

        if index == 0:
            return self.removeFirst()elif index ==
        self._size - 1:
            return self.removeLast()else:
            for x in range(i):p =
                p._link
            e = p._link._element p._link =
            p._link._link

        self._size -= 1return e

    def display(self):
    '''Utility function to display the linked list.'''
        if self.isempty() == 0:p =
            self._head while p:
                print(p._element, end='-->')p = p._link
            print("NULL")else:
            print("Empty")

    def search(self, key):'''
        Searches for the passed element in the linked list.Returns the index position if
        found, else -1.
        '''
        p = self._headindex
        = 0 while p:
            if p._element == key:return index
            p = p._linkindex += 1
        return -1

##################################################################

def options():'''
    Prints Menu for operations'''
    options_list = ['Add Last', 'Add First', 'Add Anywhere',
                    'Remove First', 'Remove Last', 'Remove Anywhere','Display List', 'Print Size',
                    'Search', 'Exit']

    print("MENU")
    for i, option in enumerate(options_list):print(f'{i + 1}. {option}')

    choice = int(input("Enter choice: "))return choice
```

```
def switch_case(choice):'''
        Switch Case for operations'''
        if choice == 1:
                elem = int(input("Enter Item: "))L.addLast(elem)
                print("Added Item at Last!\n\n")

        elif choice == 2:
                elem = int(input("Enter Item: "))L.addFirst(elem)
                print("Added Item at First!\n\n")
        elif choice == 3:
                elem = int(input("Enter Item: ")) index = int(input("Enter
                Index: "))L.addAnywhere(elem, index)

        elif choice == 4:
                print("Removed Element from First:", L.removeFirst())

        elif choice == 5:
                print("Removed Element from last:", L.removeLast())

        elif choice == 6:
                index = int(input("Enter Index: "))
                print(f"Removed Item: {L.removeAnywhere(index)} !\n\n")

        elif choice == 7: print("List: ", end='')
                L.display()
                print("\n")

        elif choice == 8: print("Size:", len(L))print("\n")

        elif choice == 9:
                key = int(input("Enter item to search: "))if L.search(key) >= 0:
                        print(f"Item {key} found at index position
{L.search(key)}\n\n")else:
                        print("Item not in the list\n\n")

        elif choice == 10:import sys
                sys.exit()

#####################################################################

if __name__ == '__main__':L =
        LinkedList()
        while True:
                choice = options() switch_case(choice)
```

Output:

Signature of the Faculty

Exercise Programs:

WEEK 11:

Write a program to implement Doubly Linked list

Program:

```
import os

class _Node:'''
        Creates a Node with three fields:
        1. element (accessed using ._element)
        2. link (accessed using ._link)
        3. prev (accessed using ._prev)'''
        __slots__= '_element', '_link', '_prev'

        def__init__(self, element, link, prev):'''
                Initialses _element, _link and _prev with element, link and prev respectively.
                '''
                self._element = elementself._link = link
                self._prev = prev

class DoublyLL:'''
        Consists of member funtions to perform differentoperations on the doubly linked
        list.
        '''

        def__init__(self):'''
                Initialises head, tail and size with None, None and 0 respectively.
                '''
                self._head  =  None self._tail
                = Noneself._size = 0

        def__len__(self):'''
                Returns length of linked list'''
                return self._size

        def isempty(self):'''
                Returns True if doubly linked list is empty, otherwise False.
                '''
                return self._size == 0

        def addLast(self, e):'''
                Adds the passed element at the end of the doubly linked list.
                '''

                  newest = _Node(e, None, None)if

                  self.isempty():
                          self._head = newestelse:
                          self._tail._link = newestnewest._prev = self._tail
                  self._tail = newestself._size += 1
```

```
def addFirst(self, e):'''
    Adds the passed element at the beginning of the doubly linked list.
    '''
    newest = _Node(e, None, None)

    if self.isempty(): self._head =
        newestself._tail = newest
    else:
        newest._link = self._headself._head._prev =
        newest
    self._head = newest
    self._size += 1

def addAnywhere(self, e, index):'''
    Adds the passed element at the passed index position of thedoubly linked list.
    '''
    if index >= self._size:
        print(f'Index value out of range, it should be between
               0 - {self._size - 1}')elif self.isempty():
        print("List was empty, item will be added at the end")self.addLast(e)
    elif index == 0: self.addFirst(e)
    elif index == self._size - 1:self.addLast(e)
    else:
        newest = _Node(e, None, None)p =
        self._head
        for _ in range(index - 1):p = p._link
        newest._link = p._link
        p._link._prev = newest
        newest._prev = p p._link =
        newest self._size += 1

def removeFirst(self):'''
    Removes element from the beginning of the doubly linked list.Returns the removed element.
    '''
    if self.isempty():
        print('List is already empty')return
    e = self._head._element self._head =
    self._head._linkself._size -= 1

    if self.isempty(): self._tail =
        None
    else:
        self._head._prev = Nonereturn e

def removeLast(self):'''
    Removes element from the end of the doubly linked list.Returns the removed element.
    '''
    if self.isempty():
        print("List is already empty")return
    e = self._tail._element self._tail =
    self._tail._prevself._size -= 1

    if self.isempty(): self._head = None
```

```
        else:
            self._tail._link = Nonereturn e

    def removeAnywhere(self, index):'''
        Removes element from the passed index position of thedoubly linked list.
        Returns the removed element.'''
        if index >= self._size:
            print(f'Index value out of range, it should be between
                    0 - {self._size - 1}')elif self.isempty():
            print("List is empty")elif index == 0:
            return self.removeFirst()elif index ==
        self._size - 1:
            return self.removeLast()else:
            p = self._head
            for _ in range(index - 1):p = p._link
            e = p._link._element p._link =
            p._link._linkp._link._prev = p
            self._size -= 1
        return e
    def display(self):'''
        Utility function to display the doubly linked list.'''
        if self.isempty(): print("List is Empty")
            return

        p = self._head print("NULL<-->",
        end='')while p:
            print(p._element, end="<-->")p = p._link
        print("NULL")

        print(f"\nHead : {self._head._element}, Tail :
              {self._tail._element}")

##################################################################

def options():'''
    Prints Menu for operations'''
    options_list = ['Add Last', 'Add First', 'Add Anywhere',
                    'Remove First', 'Remove Last', 'Remove Anywhere','Display List', 'Exit']

    print("MENU")
    for i, option in enumerate(options_list):print(f'{i + 1}. {option}')

    choice = int(input("Enter choice: "))return choice


def switch_case(choice):'''
    Switch Case for operations'''
    os.system('cls')if choice
    == 1:
        elem = int(input("Enter Item: "))DL.addLast(elem)
        print("Added Item at Last!\n\n")
```

```
    elif choice == 2:
          elem = int(input("Enter Item: "))DL.addFirst(elem)
          print("Added Item at First!\n\n")

    elif choice == 3:
          elem = int(input("Enter Item: ")) index =
          int(input("Enter Index: "))DL.addAnywhere(elem,
          index)
   elif choice == 4:
         print("Removed Element from First:", DL.removeFirst())

   elif choice == 5:
         print("Removed Element from last:", DL.removeLast())

   elif choice == 6:
         index = int(input("Enter Index: "))
         print(f"Removed Item: {DL.removeAnywhere(index)} !\n\n")

   elif choice == 7: print("List:")
         DL.display() print("\n")

   elif choice == 8:import sys
         sys.exit()

##################################################################

if __name__ == '__main__':DL =
   DoublyLL()
   while True:
         choice = options() switch_case(choice)
```

Output:

Signature of the Faculty

Exercise Programs:

WEEK 12:

Write a program to implement Binary Search Tree

Program:

```
# Binary Search Treeclass
binarySearchTree:
    def_init_(self,val=None):self.val = val
        self.left = None self.right
        = None

    def insert(self,val):
        # check if there is no rootif (self.val ==
        None):
            self.val = val
        # check where to insertelse:
            # check for duplicate then stop and return
                                        if val == self.val: return 'no duplicates allowed in binary search tree'
            # check if value to be inserted < currentNode's valueif (val < self.val):
                                        # check if there is a left node to currentNode if true then recurse
                if(self.left): self.left.insert(val)
                # insert where left of currentNode when currentNode.left=None
                else: self.left = binarySearchTree(val)

            # same steps as above here the condition we check is value to be# inserted > currentNode's value
            else:
                if(self.right): self.right.insert(val)
                else: self.right = binarySearchTree(val)


    def breadthFirstSearch(self):currentNode =
        self bfs_list = []
        queue = [] queue.insert(0,currentNode)
        while(len(queue) > 0):
            currentNode = queue.pop() bfs_list.append(currentNode.val)
            if(currentNode.left):
                queue.insert(0,currentNode.left)if(currentNode.right):
                queue.insert(0,currentNode.right)return bfs_list

    # In order means first left child, then parent, at last right child
    def depthFirstSearch_INorder(self):return
        self.traverseInOrder([])
# Pre order means first parent, then left child, at last right child
    def depthFirstSearch_PREorder(self):return
        self.traversePreOrder([])

    # Post order means first left child, then right child , at last parent
    def depthFirstSearch_POSTorder(self):return
        self.traversePostOrder([])

    def traverseInOrder(self, lst):if (self.left):
            self.left.traverseInOrder(lst)lst.append(self.val)
```

```
            if (self.right): self.right.traverseInOrder(lst)
            return lst

        def traversePreOrder(self, lst):lst.append(self.val)
            if (self.left): self.left.traversePreOrder(lst)
            if (self.right): self.right.traversePreOrder(lst)
            return lst

        def traversePostOrder(self, lst):if (self.left):
                self.left.traversePostOrder(lst)if (self.right):
                self.right.traversePostOrder(lst)lst.append(self.val)
            return lst

        def findNodeAndItsParent(self,val, parent = None):
            # returning the node and its parent so we can delete the# node and reconstruct the
            tree from its parent
            if val == self.val: return self, parentif (val < self.val):
                if (self.left):
                    return self.left.findNodeAndItsParent(val, self)else: return 'Not found'
            else:
                if (self.right):
                    return self.right.findNodeAndItsParent(val, self)else: return 'Not found'

        # deleteing a node means we have to rearrange some part of the tree
        def delete(self,val):
            # check if the value we want to delete is in the tree
            if(self.findNodeAndItsParent(val)=='Not found'): return 'Node is not in tree'
            # we get the node we want to delete and its parent-node# from
            findNodeAndItsParent method
deleteing_node, parent_node = self.findNodeAndItsParent(val)# check how many children nodes does the node
we are going #to delete have by traversePreOrder from the deleteing_node nodes_effected =
deleteing_node.traversePreOrder([])

            # if len(nodes_effected)==1 means, the node to be deleted doesn't# have any children
            # so we can just check from its parent node the position(left or# right) of node we want to delete
            # and point the position to 'None' i.e node is deleted

            if (len(nodes_effected)==1):
                if (parent_node.left.val == deleteing_node.val) : parent_node.left = None
                else: parent_node.right = Nonereturn
                'Succesfully deleted'
            # if len(nodes_effected) > 1 which means the node we are# going to delete has
            'children',
            # so the tree must be rearranged from the deleteing_node
            else:
                # if the node we want to delete doesn't have any parent# means the node to be
                deleted is 'root' node
                if (parent_node == None): nodes_effected.remove(deleteing_node.val)
                    # make the 'root' nodee i.e self value,left,right to None,# this means we need to
                    implement a new tree again without# the deleted node
                    self.left = None self.right
                    = Noneself.val = None
```

```
    #construction of new tree for node in nodes_effected:
                    self.insert(node)
                return 'Succesfully deleted'

            # if the node we want to delete has a parent# traverse from
            parent_node
            nodes_effected = parent_node.traversePreOrder([])

            # deleting the node
            if (parent_node.left == deleteing_node) : parent_node.left = None
            else: parent_node.right = None

            # removeing the parent_node, deleteing_node and inserting# the nodes_effected in
            the tree nodes_effected.remove(deleteing_node.val)
            nodes_effected.remove(parent_node.val)
            for node in nodes_effected:self.insert(node)

        return 'Successfully deleted'


bst = binarySearchTree()
bst.insert(7) bst.insert(4)
bst.insert(9) bst.insert(0)
bst.insert(5) bst.insert(8)
bst.insert(13)
```

```
#            7
#          /   \
#        /       \
#       4          9
#      / \        /  \
#     0   5      8    13
```

```
# IN order - useful in sorting the tree in ascending orderprint('IN order:
',bst.depthFirstSearch_INorder())

# PRE order - useful in reconstructing a tree print('PRE order:' ,bst.depthFirstSearch_PREorder())

# POST order - useful in finding the leaf nodes print('POST order:',
bst.depthFirstSearch_POSTorder())

print(bst.delete(5))        print(bst.delete(9))
print(bst.delete(7))

# after deleting
print('IN order: ',bst.depthFirstSearch_INorder()) print('PRE order:' ,bst.depthFirstSearch_PREorder())
print('POST order:', bst.depthFirstSearch_POSTorder())
```

Output:

Signature of the Faculty

Exercise Programs: