1a. Python program to Use and demonstrate basic data structures

```
# List example
print("List Example:")
numbers = [1, 2, 3, 4, 5]
print("Original List:", numbers)
numbers.append(6)
print("List after appending 6:", numbers)
numbers.remove(3)
print("List after removing 3:", numbers)
print("Element at index 2:", numbers[2])

print("\nSet Example:")
fruits = {"apple", "banana", "orange"}
print("Original Set:", fruits)
fruits.add("grape")
print("Set after adding grape:", fruits)
fruits.remove("banana")
print("Set after removing banana:", fruits)
print("Is 'orange' in the set?", "orange" in fruits)

print("Dictionary Example:")
student_info = {
    "id": 1,
    "name": "John Doe",
    "age": 20,
    "grade": "A"
}
print("Original Dictionary:", student_info)
print("Name:", student_info["name"])
print("Age:", student_info["age"])
student_info["age"] = 21
print("Updated Dictionary:", student_info)
removed_grade = student_info.pop("grade")
print("Dictionary after removing 'grade':", student_info)
print("Removed grade:", removed_grade)
```

Output:

```
List Example:
Original List: [1, 2, 3, 4, 5]
List after appending 6: [1, 2, 3, 4, 5, 6]
List after removing 3: [1, 2, 4, 5, 6]
Element at index 2: 4

Set Example:
Original Set: {'banana', 'apple', 'orange'}
Set after adding grape: {'banana', 'apple', 'orange', 'grape'}
Set after removing banana: {'apple', 'orange', 'grape'}
Is 'orange' in the set? True

Dictionary Example:
Original Dictionary: {'id': 1, 'name': 'John Doe', 'age': 20, 'grade': 'A'}
Name: John Doe
Age: 20
Updated Dictionary: {'id': 1, 'name': 'John Doe', 'age': 21, 'grade': 'A'}
Dictionary after removing 'grade': {'id': 1, 'name': 'John Doe', 'age': 21}
Removed grade: A
```

1b. Implement an ADT with all its operations.

I. Employee

```
class employee:
    def  _ _ init___(self):
        self.items=[ ]
    def insert(self):
        n=int(input("Enter the number of records"))
        for i in range(n):
            ename=input("Employee Name: ")
            eid=int(input("Employee ID: "))
            salary=float(input("Employee Salary: "))
            self.items.append([ename,eid,salary])
    def delete(self):
        eid=int(input("Enter Employee ID to delete"))
        for row in self.items:
            for i in row:
                if i==eid:
                    self.items.remove(row)
    def display(self):
        for i in self.items:
            print(i)
e=employee()
e.insert()
e.display()
e.delete()
e.display()
```

OUTPUT :

```
Enter the number of records2
Employee Name: abc
Employee ID: 1
Employee Salary: 200
Employee Name: def
Employee ID: 2
Employee Salary: 200
['abc', 1, 200.0]
['def', 2, 200.0]
Enter Employee ID to delete2
['abc', 1, 200.0]
```

II. Student ADT

```
class student:
    def _init_(self):
        self.items=[]
    def insert(self):
        n = int(input("Enter the number of records: "))
        for i in range(n):
            name=input("Student Name: ")
            regno=int(input("Register Number: "))
            branch=input("Branch: ")
            result=input("Result: ")
            self.items.append([name,regno,branch,result])
    def delete(self):
        regno=int(input("Enter Register Number to delete: "))
        for row in self.items:
            for i in row:
                if i==regno:
                    self.items.remove(row)
    def display(self):
        for i in self.items:
            print(i)
s=student()
s.insert()
s.display()
s.delete()
s.display()
```

OUTPUT:

```
Enter the number of records: 2
Student Name: abc
Register Number: 1
Branch: CS
Result: Pass
Student Name: def
Register Number: 2
Branch: CS
Result: Fail
['abc', 1, 'CS', 'Pass']
['def', 2, 'CS', 'Fail']
Enter Register Number to delete: 2
['abc', 1, 'CS', 'Pass']
```

Week-2
Algorithm Analysis

2a. Implement an Employee ADT and Compute space and time complexities.

```
import time
start=time.time()
class employee:
    def __init__(self):
        self.items=[]

    def insert(self):
        n=int(input("Enter the number of records"))
        for i in range(n):
            ename=input("Employee Name: ")
            eid=int(input("Employee ID: "))
            salary=float(input("Employee Salary: "))
            self.items.append([ename,eid,salary])

    def delete(self):
        eid=int(input("Enter Employee ID to delete"))
        for row in self.items:
            for i in row:
                if i==eid:
                    self.items.remove(row)

    def display(self):
        for i in self.items:
            print(i)
e=employee()
e.insert()
e.display()
e.delete()
e.display()
end=time.time()
print ({end-start})
```

<u>OUTPUT :</u>

```
Enter the number of records2
Employee Name: abc
Employee ID: 1
Employee Salary: 200
Employee Name: def
Employee ID: 2
Employee Salary: 200
['abc', 1, 200.0]
['def', 2, 200.0]
Enter Employee ID to delete2
['abc', 1, 200.0]
{31.155455112457275}
```

2b. Implement an Student ADT and Compute space and time complexities

```
class student:
    def __init__(self):
        self.items=[]
    def insert(self):
        n = int(input("Enter the number of records: "))
        for i in range(n):
            name=input("Student Name: ")
            regno=int(input("Register Number: "))
            branch=input("Branch: ")
            result=input("Result: ")
            self.items.append([name,regno,branch,result])
    def delete(self):
        regno=int(input("Enter Register Number to delete: "))
        for row in self.items:
            for i in row:
                if i==regno:
                    self.items.remove(row)
    def display(self):
        for i in self.items:
            print(i)
s=student()
s.insert()
s.display()
s.delate()
s.display()
```

OUTPUT:

```
Enter the number of records: 2
Student Name: abc
Register Number: 1
Branch: CS
Result: Pass
Student Name: def
Register Number: 2
Branch: CS
Result: Fail
['abc', 1, 'CS', 'Pass']
['def', 2, 'CS', 'Fail']
Enter Register Number to delete: 2
['abc', 1, 'CS', 'Pass']
```

2 c. Implement above solution using array and Compute space and time complexities

```
class StudentADT:
    def __init__(self):
        self.students = []

    def add(self, student_id, name, age, grade):
        self.students.append({
            'student_id': student_id,
            'name': name,
            'age': age,
            'grade': grade
        })

    def display(self, student_id):
        for student in self.students:
            if student['student_id'] == student_id:
                return student
        return None

    def remove(self, student_id):
        for i, student in enumerate(self.students):
            if student['student_id'] == student_id:
                del self.students[i]
                return True
        return False

s = StudentADT()
s.add(1, "John Doe", 20, "A")
s.add(2, "Jane Smith", 22, "B")
print("All Students:", s.students)

# Removing a student
s.remove(2)

# Displaying all students
print("All Students:", s.students)
```

Output:

```
All Students: [{'student_id': 1, 'name': 'John Doe', 'age': 20, 'grade': 'A'}, {'student_id': 2, 'name': 'Jane Smith', 'age':
22, 'grade': 'B'}]
All Students: [{'student_id': 1, 'name': 'John Doe', 'age': 20, 'grade': 'A'}]
```

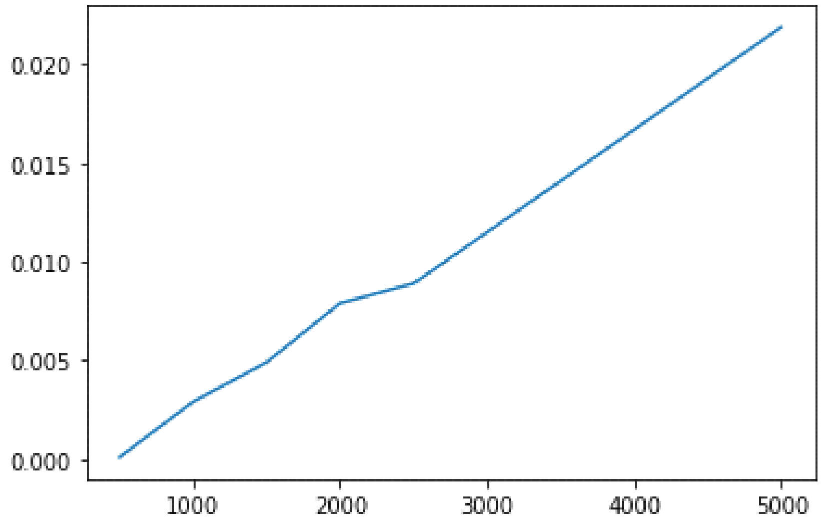
3a. Implement Linear Search compute space and time complexities, plot graph using asymptomatic notations.

```
import matplotlib.pyplot as plt
import numpy as np
import time
start=time.time()
def linear(ar,n,k):
    for i in range(n):
        if ar[i]==k:
            return i
    return-1
arr = [ ]
n = int(input("enter the no of elements"))
for i in range(n):
    arr.append(int(input('enter the element')))
print("Array elements are ", arr)
k=int(input("enter the key elements to search"))
res=linear(arr,len(arr),k)
if res== -1:
    print("element not found")
else:
    print("element %d found at index" %k , res)
end=time.time()
print({end-start})
xpoints=np.array([500,1000,1500,2000,2500,5000])
ypoints=np.array([0.00009,0.0029,0.0049,0.0079,0.0089,0.00219])
plt.plot(xpoints,ypoints)
plt.show()
```

Output:

```
enter the no of elements5
enter the element10
enter the element20
enter the element30
enter the element40
enter the element50
Array elements are  [10, 20, 30, 40, 50]
enter the key elements to search30
element 30 found at index 2

enter the no of elements5
enter the element1
enter the element2
enter the element3
enter the element4
enter the element5
Array elements are  [1, 2, 3, 4, 5]
enter the key elements to search6
element not found
{7.044062852859497}
```



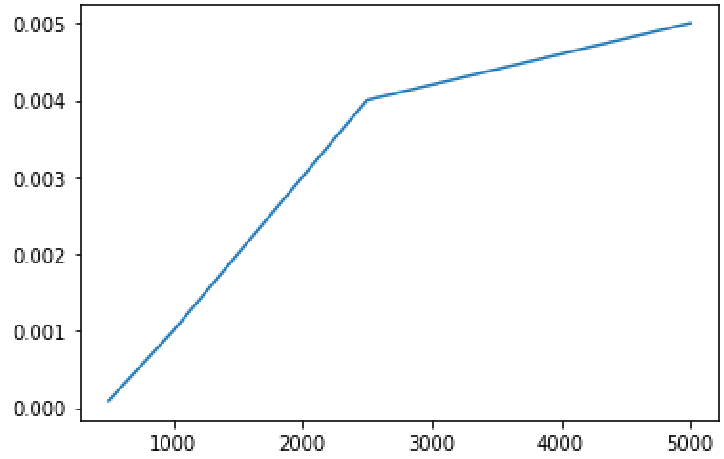
3b. Implement Bubble sorting algorithms compute space and time complexities, plot graph using asymptomatic notations

```
import matplotlib.pyplot as plt
import numpy as np
import time
start = time.time()
def bubble_sort(ar):
    n = len(ar)
    for i in range(n - 1):
        for j in range(0, n - 1 - i):
            if (ar[j] > ar[j + 1]):
                ar[j], ar[j + 1] = ar[j + 1], ar[j]

arr = [ ]
n = int(input("enter the no of elements"))
for i in range(n):
    arr.append(int(input('enter the element')))
print("unsorted array", arr)
bubble_sort(arr)
print("sorted arr:",arr)
end = time.time()
print({end - start})
xpoints = np.array([500,1000,1500,2000,2500,5000])
ypoints = np.array([0.00009,0.0010,0.0020,0.0030,0.0040,0.0050])
plt.plot(xpoints,ypoints)
plt.show()
```

Output:

```
enter the no of elements5
enter the element10
enter the element5
enter the element6
enter the element78
enter the element42
unsorted array [10, 5, 6, 78, 42]
sorted array [5 6 10 42 78 ]
 {16.526478052139282}
```

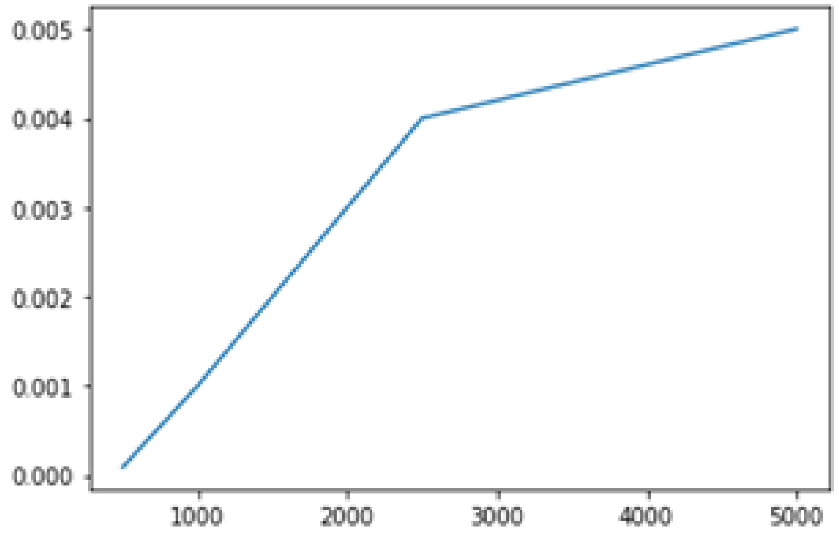


3c. Implement Selection sorting algorithms compute space and time complexities, plot graph using asymptomatic notations

```
import matplotlib.pyplot as plt
import numpy as np
import time
start = time.time()
def selection_sort(arr):
    n=len(arr)
    for i in range(n):
        min=i
        for j in range(i+1,n):
              if arr[min]>arr[j]:
                  min=j
        arr[i], arr[min]= arr[min],arr[i]
arr= [ ]
n=int(input("enter the number of element"))
for i in range (n):
    arr.append(int(input("enter the element")))
print("unsorted array",arr)
selection_sort(arr)
print("sorted array:",arr)
end=time.time()
print({end-start})
xpoints = np.array([500,1000,1500,2000,2500,5000])
ypoints = np.array([0.00009,0.0010,0.0020,0.0030,0.0040,0.0050])
plt.plot(xpoints,ypoints)
plt.show()
```

Output:

```
enter the number of element5
enter the element4
enter the element5
enter the element-3
enter the element0
enter the element1
unsorted array [4, 5, -3, 0, 1]
sorted array  [-3, 0, 1, 4, 5]
{12.0227792263031}
```

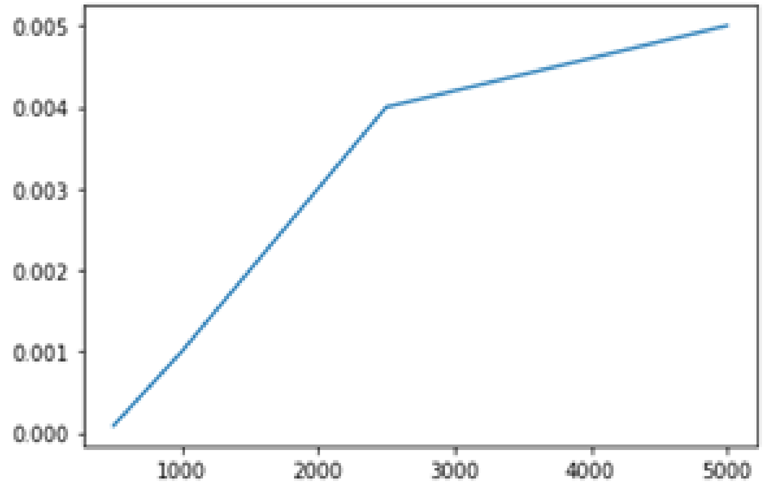


3d. Implement insertion sorting algorithms compute space and time complexities, plot graph using asymptomatic notations

```
import matplotlib.pyplot as plt
import numpy as np
import time
start=time.time()
def insertionSort(a):
    for i in range(1, len(a)):
        key = a[i]
        j = i - 1
        while j >= 0 and key < a[j]:
            a[j + 1] ,a[j]= a[j], a[j+1]
            j = j - 1
arr=[]
n=int(input("enter the number of element"))
for i in range (n):
    arr.append(int(input("enter the element")))
print("unsorted array",arr)
insertionSort(arr)
print ("Sorted array is:",arr)
end=time.time()
print({end-start})
xpoints = np.array([500,1000,1500,2000,2500,5000])
ypoints = np.array([0.00009,0.0010,0.0020,0.0030,0.0040,0.0050])
plt.plot(xpoints,ypoints)
plt.show()
```

Output:

```
enter the number of element 5
enter the element7
enter the element4
enter the element6
enter the element12
enter the element10
unsorted array [7, 4, 6, 12, 10]
sorted array [4, 6, 7, 10, 12]
{9.21582579612732}
```

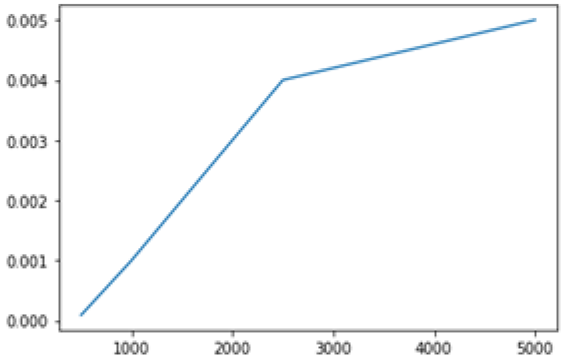

0.005
0.004
0.003
0.002
0.001
0.000
1000
2000
3000
4000
5000

4a. Implement Binary Search using recursion Compute space and time complexities, plot graph using asymptomatic notations and compare two.

```
import matplotlib.pyplot as plt
import numpy as np
import time
start = time.time()
def binary_search(arr, low, high, x):
  if high >= low:
        mid = (high + low) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            return binary_search(arr, low, mid - 1, x)
        else:
            return binary_search(arr, mid + 1, high, x)
  else:
    return  -1
arr = [ ]
n=int(input("enter the number of element"))
for i in range (n):
    arr.append(int(input("enter the element")))
print(" sorted array",arr)
x = int(input("Enter the key elements"))
result = binary_search(arr, 0, len(arr)-1, x)
if result !=-1:
 print("Element is present at index", str(result))
else:
 print("Element is not present in array")
end=time.time()
print({end-start})
xpoints=np.array([500,1000,1500,2000,2500,5000])
ypoints=np.array([0.00009,0.0010,0.0020,0.0030,0.0040,0.0050])
plt.plot(xpoints,ypoints)
plt.show()
```

OUTPUT:-

```
enter the number of element5
enter the element10
enter the element20
enter the element25
enter the element30
enter the element40
sorted array [10, 20, 25, 30, 40]
Enter the key elements20
Element is present at index 1
{18.297078371047974}
```



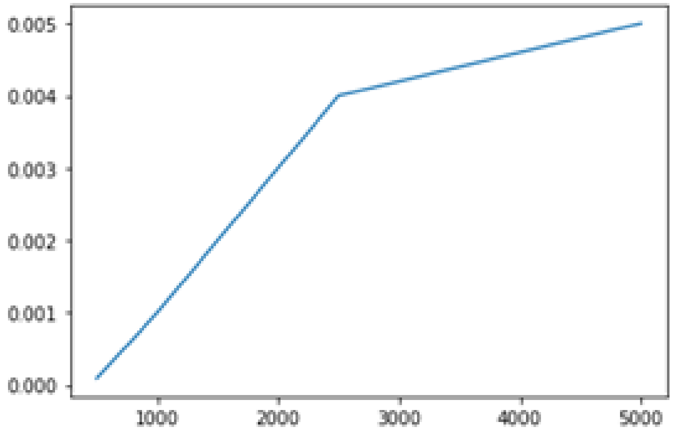
4b. Implement Merge sorting algorithms compute space and time complexities, plot graph using asymptomatic notations and compare all solutions.

```
import matplotlib.pyplot as plt
import numpy as np
import time
start = time.time()
def mergeSort(array):
    if len(array) > 1:
        r = len(array)//2
        L = array[:r]
        M = array[r:]
        mergeSort(L)
        mergeSort(M)
        i = j = k = 0
        while i < len(L) and j < len(M):
            if L[i] < M[j]:
                array[k] = L[i]
                i += 1
            else:
                array[k] = M[j]
                j += 1
            k += 1
        while i < len(L):
            array[k] = L[i]
            i += 1
            k += 1
        while j < len(M):
            array[k] = M[j]
            j += 1
            k += 1
def printList(array):
    for i in range(len(array)):
        print(array[i], end=" ")
    print()


array = [ ]
n=int(input("enter the number of element"))
for i in range (n):
    array.append(int(input("enter the element")))
print("Unsorted array is: ")
printList(array)
mergeSort(array)
print("Sorted array is: ")
printList(array)
end = time.time()
print({end - start})
xpoints = np.array([500, 1000, 1500, 2000, 2500, 5000])
ypoints = np.array([0.00009, 0.0010, 0.0020, 0.0030, 0.0040, 0.0050])
plt.plot(xpoints, ypoints)
plt.show()
```

OUTPUT:-

```
enter the number of element 8
enter the element5
enter the element4
enter the element7
enter the element1
enter the element2
enter the element4
enter the element5
enter the element6
Unsorted array is:
5 4 7 1 2 4 5 6
Sorted array is:
1 2 4 4 5 5 6 7
{13.86066722869873}
```

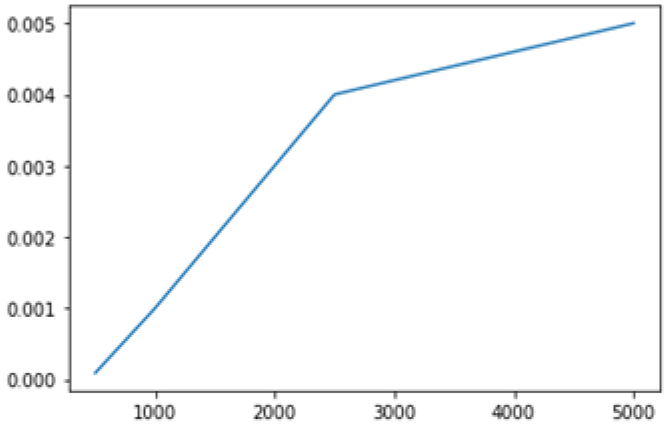


4c. Implement Quick sorting algorithms compute space and time complexities, plot graph using asymptomatic notations and compare all solutions.

```
import matplotlib.pyplot as plt
import numpy as np
import time
start = time.time()
def partition(arr, low, high):
    pivot = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] <= pivot:
            i = i + 1
            (arr[i], arr[j]) = (arr[j], arr[i])
    (arr[i + 1], arr[high]) = (arr[high], arr[i + 1])
    return i + 1
def quick_sort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quick_sort(array, low, pi - 1)
        quick_sort(array, pi + 1, high)
array = [ ]
n=int(input("enter the number of element"))
for i in range (n):
    array.append(int(input("enter the element")))
size = len(array)
print("unsorted array", array)
quick_sort(array, 0, size - 1)
print("sorted array", array)
end = time.time()
print(end-start)
xpoints = np.array([500, 1000, 1500, 2000, 2500, 5000])
ypoints = np.array([0.00009, 0.0010, 0.0020, 0.0030, 0.0040, 0.0050])
plt.plot(xpoints, ypoints)
plt.show()
```

OUTPUT:-

```
enter the number of element 9
enter the element9
enter the element5
enter the element4
enter the element7
enter the element3
enter the element6
enter the element2
enter the element5
enter the element4
unsorted array [9, 5, 4, 7, 3, 6, 2, 5, 4]
Unsorted array is:
9 5 4 7 3 6 2 5 4
Sorted array is:
2 3 4 4 5 5 6 7 9
{18.112088680267334}
```



4d. Implement Fibonacci sequence with dynamic programming.

```
def fibonacci(n):
    list = [0, 1]
    a=0
    b=1
    if n<0:
        print("incorrect input")
    elif n==0:
        return [0]
    elif n==1:
        return list
    else:
        for i in range(2,n+1):
            c=a+b
            a=b
            b=c
            list.append(c)
        return  list

n = int(input("Enter the number : "))
list = fibonacci(n)
print("The fibonacci list is :", list)
print("The fibonacci of the given number is :", list[n])
```

Output:

```
Enter the number : 8
The fibonacci list is : [0, 1, 1, 2, 3, 5, 8, 13, 21]
The fibonacci of the given number is : 21
```

5. Implement singly linked list (Traversing the Nodes, searching for a Node, Prepending Nodes and Removing Nodes)

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

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

    def prepend(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node

    def insertAfter(self, prev_node, new_data):
        if prev_node is None:
            print("The given previous node must inLinkedList.")
            return
        new_node = Node(new_data)
        new_node.next = prev_node.next
        prev_node.next = new_node

    def append(self, new_data):
        new_node = Node(new_data)
        if self.head is None:
            self.head = new_node
            return
        last = self.head
        while (last.next):
            last = last.next
        last.next = new_node

    def deleteNode(self, position):
        if self.head is None:
            print("The linked list is empty")
            return
        temp = self.head
        if position == 0:
            self.head = temp.next
            temp = None
            return
        for i in range(position - 1):
            temp = temp.next
            if temp is None:
                break
        if temp is None:
            return
        if temp.next is None:
            print("The position is not found, No elements deleted")
            return

        print("The position is found")
        next = temp.next.next
        temp.next = None
        temp.next = next

def search(self, key):
        current = self.head
        while current is not None:
            if current.data == key:
                return True
            current = current.next
        return False
    def traversal(self):
        temp = self.head
        while (temp):
            print(str(temp.data) + " ", end="\n")
            temp = temp.next


l = LinkedList()
l.append(1)
l.prepend(2)
l.prepend(3)
l.append(4)
l.insertAfter(l.head.next, 5)
print('linked list elements are :')
l.traversal()
n = int(input("Enter the position of element to delete : "))
l.deleteNode(n)
print("\nAfter deleting an element:")
l.traversal()
print()
find = int(input("Enter the element to find"))
if l.search(find):
    print(str(find) + " is found")
else:
    print(str(find) + " is not found")
```

Output:

```
linked list elements are :
3
2
5
1
4
Enter the position of element to delete : 3
The position is found
After deleting an element:
3
2
5
4

Enter the element to find 1
1 is not found
```

7.1 Implementation of Doubly Linked list

```
class Node:
    def __init__ (self,data):
        self.data=data
        self.next=None
        self.prev=None

class doubly:
    def __init__ (self):
        self.head=None

    #To Append a Node
    def append(self,data):
        newnode=Node(data)
        if self.head==None:
            self.head=newnode
            self.head.prev=None
            self.head.next=None
        else:
            last = self.head
            while (last.next is not None):
                last = last.next

            last.next = newnode
            newnode.prev = last

    #To Display the Nodes
    def display(self):
        current=self.head
        if self.head==None:
            print("List is Empty")
            return
        print("Node of Doubly Linked List")
        while current != None:
            print(current.data)
            current=current.next

    #To Delete a Node
    def deletenode(self,key):
        temp=self.head
        if temp is not None:
            if temp.data==key:
                self.head=temp.next
                temp=None
                return
        while temp is not None:
            if temp.data is key:
                break
            prev=temp
            temp=temp.next
        if temp==None:
            return
        prev.next=temp.next
        temp=None

d=doubly()
d.append(1)
d.append(2)
d.append(3)
d.append(4)
d.append(5)
d.append(6)
d.display()
d.deletenode(2)
print("After deleting elements ")
d.display()
```

Output:

```
Node of Doubly Linked List
1
2
3
4
5
6
After deleting elements
Node of Doubly Linked List
1
3
4
5
6
```

7.2 Implementation of Circular linked list

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class Circullarll:
    def __init__(self):
        self.head=Node(None)
        self.head.next=self.head
    def append(self,data):
        newnode=Node(data)
        if (self.head == None):
            self.head = newnode
            newnode.next = self.head
            return
        else:
            temp = self.head
            while (temp.next != self.head):
                temp = temp.next
            temp.next = newnode
            newnode.next = self.head

    def display(self):
        current=self.head
        if self.head is None:
            print("List is Empty")
            return
        else:
            print("Node in Linkedlist are")
            # print(current.data)
            while (current.next!=self.head):
                current=current.next
                print (current.data)

    def deletenode(self, key):
        temp = self.head
        if temp.next is not temp:
            if temp.data == key:
                self.head = temp.next
                temp = None
                return
        while temp.next is not self.head:
            if temp.data is key:
                break
            prev = temp
            temp = temp.next
        if temp == self.head:
            return
        prev.next = temp.next
        # temp = None


a=Circullarll()
a.append(1)
a.append(2)
a.append(3)
a.append(4)
a.display()
a.deletenode(2)
print("After deleting the element")
a.display()
```

Output:

```
Node in Linkedlist are
1
2
3
4
After deleting the element
Node in Linkedlist are
1
3
4
```

8.1 Implement Stack Data Structure

```
class Stack:
    def __init__(self):
        self.items=[]
    def isEmpty(self):
        return self.items==[]
    def push(self):
        data = int(input("Enter the number : "))
        self.items.append(data)
    def pop(self):
        return self.items.pop()
    def display(self):
        if self.items==[]:
            print("List is empty")
        else:
            for i in self.items:
                print(i)
s=Stack()
while True:
    print("1. Push ")
    print("2. Pop")
    print("3. Display")
    print("4. Quit")
    ch=input("Enter the value for operation: ")
    if ch=="1":
        s.push()
    elif ch=="2":
        if s.isEmpty():
            print("Stack is empty")
            break
        else:
            print("Removed value: ",s.pop())
    elif ch=="3":
         s.display()
    elif ch=="4":
        break
```

Output:

```
1. Push
2. Pop
3. Display
4. Quit
Enter the value for operation: 1
Enter the number : 3
1. Push
2. Pop
3. Display
4. Quit
Enter the value for operation: 1
Enter the number : 5
1. Push
2. Pop
3. Display
4. Quit
Enter the value for operation: 3
3
5
1. Push
2. Pop
3. Display
4. Quit
Enter the value for operation: 2
Removed value:  5
1. Push
```

```
2. Pop
3. Display
4. Quit
Enter the value for operation: 4
```

8.2 Implement bracket matching using stack.

```
open_list=["[", "{", "("]
close_list=["]", "}", ")"]
def check(mystr):
   stack=[]
   for i in mystr:
       if i in open_list:
           stack.append(i)
       elif i in close_list:
           por=close_list.index(i)
           if ((len(stack)>=0) and (open_list[por]==stack[len(stack)-1])):
               stack.pop()
           else:
               return "Unbalanced"
   if len(stack)==0:
       return "Balanced"
   else:
       return "Unbalanced"

string = input("Enter the brackets : ")
print(string, " is: ",check(string))
```

Output1:

```
            Enter the brackets : ({})[]{}
            ({})[]{}  is:  Balanced
```

Output2:

```
            Enter the brackets : ({)}[]{}
            ({)}[]{}  is:  Unbalanced
```

9.1 Program to demonstrate recursive operations (Factorial/ Fibonacci).

a)Fibonnaci

```
def fib(n):
   if n<=1:
       return n
   else:
       return fib(n-1)+fib(n-2)

n = int(input("Enter the number: "))
if n<0:
   print("Enter the Positive number..")
else:
    fib_ser = []
    for i in range(n):
        fib_ser.append(fib(i))
    print("The fibonnaci series is :", fib_ser )
    print("The fibonnaci of given number is :", fib_ser[-1])
```

Output:

```
 Enter the number: 5
Enter the number: 5
The fibonnaci series is : [0, 1, 1, 2, 3]
The fibonnaci of given number is : 3
```

b) Factorial

```
def factorial(x):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))
num = int(input("Enter a number: "))
result = factorial(num)
print("The factorial of", num, "is", result)
```

Output:

```
Enter a number: 5
The factorial of 5 is 120
```

9.2 Implement solution for Towers of Hanoi.

```
 def TOH (n, spole, dpole, ipole):
    if (n == 1):
        print("move disc 1 from pole", spole, "to pole", dpole)
        return
    TOH (n-1, spole, ipole, dpole)
    print("move disc", n, "from pole", spole, "to pole", dpole)
    TOH (n-1, ipole, dpole, spole)
n = 3
TOH(n, 'A', 'B', 'C')
```

Output:

```
move disc 1 from pole A to pole B
move disc 2 from pole A to pole C
move disc 1 from pole B to pole C
move disc 3 from pole A to pole B
move disc 1 from pole C to pole A
move disc 2 from pole C to pole B
move disc 1 from pole A to pole B
```

10.1 Implement Queue

```
class queue:
    def __init__(self):
        self.queue=[]
    def insert(self):
            data=int(input("enter the number"))
            self.queue.append(data)
    def delete(self):
        if len(self.queue)<1:
            return None
        else:
            print("remeoved element is
",self.queue.pop(0))
    def display(self):
        if (len(self.queue)==0):
            print("queue empty")
        else:
            print(self.queue)

q=queue()
while True:
    print('queue operation \n'
          '1.append\n'
         '2.delete\n'
          '3.display \n'
           '4.exit')
    c=int(input("enter your choice"))
    if c==1:
        q.insert()
    elif c==2:
        q.delete()
    elif c==3:
        q.display()
    elif c==4:
        break
    else:
        print("invalid input")
```

Output:

```
queue operation
1.append
2.delete
3.display
4.exit
enter your choice1
enter the number10
queue operation
1.append
2.delete
3.display
4.exit
enter your choice1
enter the number20
queue operation
1.append
2.delete
3.display
4.exit
enter your choice1
enter the number30
queue operation
1.append
2.delete
3.display
4.exit
enter your choice3
[10, 20, 30]
queue operation
1.append
2.delete
3.display
4.exit
enter your choice2
remeoved element is  10
queue operation
1.append
2.delete
3.display
4.exit
enter your choice3
[20, 30]
queue operation
1.append
2.delete
3.display
4.exit
enter your choice4
```

10.2 Implement Priority Queue

```
class pq:
    def __init__(self):
        self.queue=[]
    def insert(self):
            data=int(input("enter the number"))
            self.queue.append(data)
    def delete(self):
        maxvalue=0
        for i in range(len(self.queue)):
            if self.queue[i]>self.queue[ maxvalue]:
                maxvalue=i
        item=self.queue[ maxvalue]
        del self.queue[ maxvalue]
        print("removed element is ",item)
        print()
    def display(self):
        if (len(self.queue) == 0):
            print("queue empty")
        else:
            print(self.queue)

q=pq()
while True:
    print('queue operation\n 1.append 2.delete 3.display 4.exit')
    c=int(input("enter your choice"))
    if c==1:
        q.insert()
    elif c==2:
        q.delete()
    elif c==3:
        q.display()
    elif c==4:
        break
    else:
        print("invalid input")
```

Output:

```
queue operation
 1.append 2.delete 3.display 4.exit
enter your choice 1
enter the number10
queue operation
 1.append 2.delete 3.display 4.exit
enter your choice1
enter the number40
queue operation
 1.append 2.delete 3.display 4.exit
enter your choice1

enter the number20
queue operation
 1.append 2.delete 3.display 4.exit
```

```
enter your choice1
enter the number27
queue operation
 1.append 2.delete 3.display 4.exit
enter your choice3
[10, 40, 20, 27]
queue operation
 1.append 2.delete 3.display 4.exit
enter your choice2
removed element is  40

queue operation
 1.append 2.delete 3.display 4.exit
enter your choice3
[10, 20, 27]
queue operation
 1.append 2.delete 3.display 4.exit
enter your choice4
```

11. Implement Binary search tree and its operations using list.

```
class BSTNode:
    def __init__(self, val=None):
        self.left = None
        self.right = None
        self.val = val
    def insert(self, val):
        if not self.val:
            self.val = val
            return
        if self.val == val:
            return
        if val < self.val:
            if self.left:
                self.left.insert(val)
                return
            self.left = BSTNode(val)
            return
        if self.right:
            self.right.insert(val)
            return
        self.right = BSTNode(val)
    def preorder(self, vals):
        if self.val is not None:
            vals.append(self.val)
        if self.left is not None:
            self.left.preorder(vals)
        if self.right is not None:
            self.right.preorder(vals)
        return vals
    def inorder(self, vals):
        if self.left is not None:
            self.left.inorder(vals)
        if self.val is not None:
            vals.append(self.val)
        if self.right is not None:
            self.right.inorder(vals)
        return vals
    def postorder(self, vals):
        if self.left is not None:
            self.left.postorder(vals)
        if self.right is not None:
            self.right.postorder(vals)
        if self.val is not None:
            vals.append(self.val)
        return vals
nums = [12, 6, 18, 19, 21, 11, 3, 5, 4, 24,18]
bst = BSTNode()
for num in nums:
    bst.insert(num)
print("preorder:")
print(bst.preorder([]))
print("postorder:")
print(bst.postorder([]))
print("inorder:")
print(bst.inorder([]))
```

Output:

```
preorder:
[12, 6, 3, 5, 4, 11, 18, 19, 21, 24]
postorder:
[4, 5, 3, 11, 6, 24, 21, 19, 18, 12]
inorder:
[3, 4, 5, 6, 11, 12, 18, 19, 21, 24]
```

12a. Implementations of BFS.

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = []
queue = []

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)
  while queue:
    m = queue.pop(0)
    print (m, end = " ")
    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

print("Following is the Breadth-First Search")
bfs(visited, graph, '5')
```

Output:

```
Following is the Breadth-First Search
5 3 7 2 4 8
```

12b. Implementation of DFS

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}
visited = set()
def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

Output:

```
Following is the Depth-First Search
5
3
2
4
8
7
```

13. Implement Hash functions.

a)**Demonstration working of hash function**

```
val1 = 121
val2 = 121.09
val3 = "GeeksforGeeks"
print("The integer value is " ,hash(val1))
print("The float value is " ,hash(val2))
print("The string value is " ,hash(val3))
tuple = (1,2,3,4,5)
print("The tuple value is " ,hash(tuple))
```

Output:

```
The integer value is  121

The float value is  207525870829240441

The string value is  -2963701940165539148

The tuple value is  -5659871693760987716
```

13 b. Hash function for custom objects

```
class Student:
    def __init__(self,name,email):
        self.name = name
        self.email = email

s=Student("Arun", "arun@abc.com")
result = hash(s)
print("Hash value=",result)
```

Output:

```
Hash value= 164879298963
```