1. Implement an ADT with all its operations.

```
class date:
    def __init__(self,a,b,c):
        self.d=a
        self.m=b
        self.y=c
    def day(self):
        print("Day = ", self.d)
    def month(self):
        print("Month = ", self.m)
    def year(self):
        print("year = ", self.y)
    def monthName(self):
        months = ["Unknown","January","Febuary","March","April","May","June","July",
"August","September","October","November","December"]
        print("Month Name:",months[self.m])
    def isLeapYear(self):
        if (self.y % 400 == 0) and (self.y % 100 == 0):
            print("It is a Leap year")
        elif (self.y % 4 == 0) and (self.y % 100 != 0):
            print("It is a Leap year")
        else:
            print("It is not a Leap year")
d1 = date(3,8,2000)
d1.day()
d1.month()
d1.year()
d1.monthName()
d1.isLeapYear()
```

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

```
import time
class stack:
    def __init__(self):
        self.items = []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def display(self):
        return (self.items)
```

```
s=stack()
start = time.time()
print("push operations")
s.push(11)
s.push(12)
s.push(13)
print(s.display())
print("pop operations")
print(s.pop())
print(s.pop())
print(s.display())
end = time.time()
print("Runtime of the program is", end - start)
```

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

```
import time
def linearsearch(a, key):
    n = len(a)
    for i in range(n):
        if a[i] == key:
            return i;
    return -1
a = [13,24,35,46,57,68,79]
start = time.time()
print("the array elements are:",a)
k = int(input("enter the key element to search:"))
i = linearsearch(a,k)
if i == -1:
    print("Search UnSuccessful")
else:
    print("Search Successful key found at location:",i+1)
end = time.time()
print("Runtime of the program is", end-start)
```

4. Implement Bubble Sort and compute space and time complexities, plot graph using asymptomatic notations

```
def bubblesort(a):
    n = len(a)
```

```
    for i in range(n-1):
        for j in range(n-1-i):
            if a[j]>a[j+1]:
                temp = a[j]
                a[j] = a[j+1]
                a[j+1] = temp
x = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",x)
bubblesort(x)
print("After sorting:",x)
```

5. Implement Selection Sort and compute space and time complexities, plot graph using asymptomatic notations

```
def selectionsort(a):
    n = len(a)
    for i in range(n-2):
        min = i
        for j in range(i+1,n-1):
            if a[j]<a[min]:
                min=j
        temp = a[i]
        a[i] = a[min]
        a[min] = temp
x = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",x)
selectionsort(x)
print("After sorting:",x)
```

6. Implement Binary Search and compute space and time complexities, plot graph using asymptomatic notations

```
import time
def binarysearch(a, key):
    low = 0
    high = len(a) - 1
    while low <= high:
        mid = (high + low) // 2
        if a[mid] == key:
            return mid
        elif key < a[mid]:
            high = mid - 1
        else :
```

```
            low = mid + 1
    return -1
start = time.time()
a = [13,24,35,46,57,68,79]
print("the array elements are:",a)
k = int(input("enter the key element to search:"))
r = binarysearch(a,k)
if r == -1:
    print("Search UnSuccessful")
else:
    print("Search Successful key found at location:",r+1)
end = time.time()
print("Runtime of the program is:", end-start)
```

7. Implement Binary Search using Recursion and compute space and time complexities, plot graph using asymptomatic notations

```
def binarysearch(a, low, high, key):
    if low <= high:
        mid = (high + low) // 2
        if a[mid] == key:
            print("Search Successful key found at location:",mid+1)
            return
        elif key < a[mid]:
            binarysearch(a, low, mid-1, k)
        else :
            binarysearch(a, mid + 1, high, k)
    else:
        print("Search UnSuccessful")
a = [13,24,35,46,57,68,79]
print("the array elements are:",a)
k = int(input("enter the key element to search:"))
binarysearch(a, 0, len(a)-1, k)
```

8. Implement Fibonacci sequence with dynamic programming.

```
def fib(n):
    if n<=1:
        return n
    f = [0, 1]
    for i in range(2, n+1):
        f.append(f[i-1] + f[i-2])
    print("The Fibonacci sequence is:",f)
    return f[n]
```

```
n=int(input("Enter the term:"))
print("The Fibonacci value is:",fib(n))
```

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

```
class Node:
    def __init__(self, data = None):
        self.data = data
        self.next = None
class SinglyLinkedList:
    def __init__(self):
        self.first = None
    def insertFirst(self, data):
        temp = Node(data)
        temp.next=self.first
        self.first=temp
    def removeFirst(self):
        if(self.first== None):
            print("list is empty")
        else:
            cur=self.first
            self.first=self.first.next
            print("the deleted item is",cur.data)
    def display(self):
        if(self.first== None):
            print("list is empty")
            return
        cur = self.first
        while(cur):
          print(cur.data, end = " ")
          cur = cur.next
#Singly Linked List
sll = SinglyLinkedList()
while(True):
    ch = int(input("\nEnter your choice 1-insert 2-delete 3-display 4-exit :"))
    if(ch == 1):
        item = input("Enter the element to insert:")
        sll.insertFirst(item)
        sll.display()
    elif(ch == 2):
        sll.removeFirst()
        sll.display()
    elif(ch == 3):
        sll.display()
    else:
        break
```

10. Implement Stack Data Structure.

```
class Stack:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return len(self.items) == 0
    def push(self,item):
        self.items.append(item)
    def pop(self):
        if self.isEmpty():
            print("Stack is Empty")
        else:
            item = self.items[-1]
            del(self.items[-1])
            print("The popped element is:",item)
    def display(self):
        if self.isEmpty():
            print("Stack is Empty")
        else:
            for i in reversed(self.items):
                print(i)
    def peek(self):
        if self.isEmpty():
            print("Stack is Empty")
        else:
            print("Top item is ", self.items[-1])
s = Stack()
while(True):
    print("1:push 2:pop 3:display 4:peek 5:exit")
    choice = int(input("Enter your choice:"))
    if choice == 1:
        item = input("Enter the item to push:")
        s.push(item)
    elif choice == 2:
        s.pop()
    elif choice == 3:
        s.display()
    elif choice == 4:
        s.peek()
    else:
        break
```

11. Implement bracket matching using stack.

```
class Stack:
    def __init__(self):
        self.items = []
    def push(self,item):
       self.items.append(item)
    def pop(self):
        if len(self.items) is 0:
            print("Stack is Empty")
        else:
            item = self.items[-1]
            del(self.items[-1])
            return item
def check_brackets(expr):
    s = Stack()
    for token in expr:
        if token in "{[(":
            s.push(token)
        elif token in "}])":
            if  len(s.items) == 0:
                return False
            else:
                left = s.pop()
                if (token == "}" and left != "{") or (token == "]" and left != "[") or (token == ")" and left != "(") :
                    return False
    if  len(s.items) == 0:
                return True
expr =input("Enter the Expertion:")
result = check_brackets(expr)
if result:
    print("The Given Expression is Valid")
else:
    print("The Given Expression is Invalid")
```

12. Program to demonstrate recursive operations (factorial/ Fibonacci)

a) Factorial

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

b) Fibonacci

```
def fib(n):
    if n<=1:
        return n
    return fib(n-1) + fib(n-2)
n=int(input("Enter the range:"))
print("The fibonacci value is:",fib(n))
```

13. Implement Queue Data Structure.

```
class Queue:
    def __init__(self):
        self.items = []
    def enqueue(self,item):
        self.items.append(item)
    def dequeue(self):
        if self.isEmpty():
            print("Queue is Empty cannot delete")
        else:
            item=self.items.pop(0)
            print("Deleted Item is:",item)
    def display(self):
        if self.isEmpty():
            print("Queue is Empty")
        else:
            print(self.items)
    def length(self):
        return len(self.items)
    def isEmpty(self):
        return len(self.items) == 0
q = Queue()
while True:
    print("1:Enqueue 2:Dequeue 3:Display 4:Length 5:Exit")
    choice = int(input("Enter your choice:"))
    if choice==1:
        item=input("Enter the element:")
        q.enqueue(item)
    elif choice==2:
        q.dequeue()
    elif choice==3:
        q.display()
    elif choice==4:
        n = q.length()
```

```
        print("Length of the queue is ",n)
    elif choice==5:
        break
```

14. Implement Binary Search Tree and its operations using list.

```
class Node:
    def __init__(self,value):
        self.data = value
        self.left = None
        self.right =None
class BinarySearchTree:
    def __init__(self):
        self.root=None
    def insert(self,value):
        newNode=Node(value)
        if self.root is None:
            self.root = newNode
        else:
            curNode = self.root
            while curNode is not None:
                if value < curNode.data:
                    if curNode.left is None:
                        curNode.left=newNode
                        break
                    else:
                        curNode = curNode.left
                else:
                    if curNode.right is None:
                        curNode.right=newNode
                        break
                    else:
                        curNode=curNode.right
    def preorder(self, rt):
        print(rt.data, end="")
        if rt.left is not None:
            self.preorder(rt.left)
        if rt.right is not None:
            self.preorder(rt.right)
    def postorder(self, rt):
        if rt.left is not None:
            self.postorder(rt.left)
        if rt.right is not None:
```

```
            self.postorder(rt.right)
        print(rt.data, end="")
    def inorder(self, rt):
        if rt.left is not None:
            self.inorder(rt.left)
        print(rt.data, end="")
        if rt.right is not None:
            self.inorder(rt.right)
bst = BinarySearchTree()
ls = [25,10,35,20,65,45,24]
for i in ls:
    bst.insert(i)
print("\nPre-order traversal is:")
bst.preorder(bst.root)
print("\nPost-order traversal is:")
bst.postorder(bst.root)
print("\nIn-order traversal is:")
bst.inorder(bst.root)
```

15. Implement Hash functions.

```
class Hash:
    def __init__(self):
        self.buckets=[[],[],[],[],[]]
    def insert(self,key):
        bindex = key % 5
        self.buckets[bindex].append(key)
        print(key,"inserted in Bucket No.",bindex+1)
    def search(self,key):
        bindex = key % 5
        if key in self.buckets[bindex]:
            print(key,"present in bucket No.",bindex+1)
        else:
            print(key,"is not present in any of the buckets")
    def display(self):
        for i in range(0,5):
            print("\nBucket No.",i+1,end=":")
            for j in self.buckets[i]:
                print(j,end="->")
hsh = Hash()
while True:
    print("\nHash operations 1.Insert 2.Search 3.Display 4.Quit")
```

```
ch=int(input("Enter your choice:"))
if ch == 1:
    key=int(input("Enter key to be inserted:"))
    hsh.insert(key)
elif ch == 2:
    key=int(input("\nEnter key to be searched:"))
    hsh.search(key)
elif ch == 3:
    hsh.display()
else:
    break
```