GOVERNMENT OF KARNATAKA

DEPARTMENT OF COLLEGIATE & TECHNICAL EDUCATION

DATA STRUCTURES WITH PYTHON

LAB RECORD

Academic Year	**:**	**2022 – 2023**
Course Code	**:**	**20CS41P**
Semester	**:**	**IV Semester**
Branch	**:**	**Computer Science & Engineering**

Submitted By:

Student Name :

Register Number :

Department of Computer Science and Engineering

GOVERNMENT POLYTECHNIC

SIDDAPUR (U.K) – 581355

GOVERNMENT OF KARNATAKA

DEPARTMENT OF COLLEGIATE & TECHNICAL EDUCATION

GOVERNMENT POLYTECHNIC
SIDDAPUR (U.K) - 581355

Certificate

This is to Certify that it is bonafied record of Practical work done by Mr./Ms. _____ bearing the Register No. _____ of IV semester, Computer Science & Engineering Branch in the `Data structures with Python - 20CS41P` ***Laboratory during the academic year 2022 – 2023.***

Course Coordinator **Head of the Department**

Examiners: *1.* _____

2. _____

Index

Name: **Register Number:**

Branch: Computer Science & Engineering **Class: IV Semester**

Institution: Government Polytechnic, Siddapur

Week No	Practical Exercise	Marks obtained	Initials

Average Marks: ____ / 10

Signature of the student **Signature of the Course Coordinator**

Week 1 Practice

Exp #1: Python Program to Demonstrate and use Basic Data Structures.

Python Program:

```
a=10
b=True
c=15.2
d="Siddapur"

print("a =",a,"\tType:", type(a))

print("b =",b,"\tType:", type(b))

print("c =",c,"\tType:", type(c))

print("d =",d,"\tType:", type(d))
```

Output:

Algorithmic Solution:

Week 1 Practice

Exp #2: Implement an ADT with all its operations.

Python Program:

```
class Date:
    def __init__(self,dd,mm,yy):
        self.Day = dd
        self.Month = mm
        self.Year = yy

    def getDay(self):
        return self.Day

    def getMonth(self):
        return self.Month

    def getYear(self):
        return self.Year

    def displayDate(self):
        print("\n{}-{}-{}".format(self.Day,self.Month,self.Year))

dt=Date("15","03","2023")

print("Day :",dt.getDay())
print("Month:",dt.getMonth())
print("Year :",dt.getYear())

dt.displayDate()
```

Output:

Algorithmic Solution:

Week 2 Practice

Exp #1: Implement an ADT and compute its time complexity.

Python Program:

```
import time
class Student:
    def __init__(self,d,m,y,x):
        self.fname=d
        self.lname=m
        self.sem=y
        self.age=x

    def displayFullName(self):
        print("Full Name:",self.fname,self.lname)

    def updateSem(self,s):
        self.sem = s

    def displayStudInfo(self):
        print("Name: ",self.fname,self.lname)
        print("Semester: ",self.sem)
        print("Age: ",self.age)

start = time.time()

s1 = Student("A", "B", 3, 20)

s1.displayStudInfo()
s1.updateSem(4)
print("----------------------------")
s1.displayStudInfo()
print("----------------------------")

end = time.time()
print("Time Taken:", end-start,"ms")
```

Output:

Algorithmic Solution:

Week 3 Practice

Exp #1: Implement Linear Search and compute its time complexity.

Python Program:

```
import time

def linearSearch(a, key):
    n=len(a)
    for i in range(n):
        if key == a[i]:
            return True
    return False

n=int(input("Enter Size of the List\n"))

a=list()
print("Enter List elements")
for i in range(n):
    num=int(input())
    a.append(num)

key=int(input("Enter Key element to be searched\n"))

print("\nSearch List: ", a)
print("Key:", key)

start=time.time()
res=linearSearch(a,key)
end=time.time()

if res==True:
    print("\nSuccessful Search")
else:
    print("\nUnsuccessful Search")

print("Execution Time:",end-start,"seconds")
```

Output:

Algorithmic Solution:

Week 3 Practice

Exp #2: Implement Bubble, Selection and Insertion sorting algorithms and compute its time complexity.

Python Program:

```
import time
def bubbleSort(a):
    n=len(a)
    start = time.time()
    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
    end = time.time()
    print("\n\nExecution Time:",end-start,"ms")
def selectionSort(a):
    n=len(a)
    start = time.time()
    for i in range(n-1):
        min=i
        for j in range(i+1,n):
            if a[j]< a[min]:
                min=j
        temp=a[i]
        a[i]=a[min]
        a[min]=temp
    end = time.time()
    print("\n\nExecution Time:",end-start,"ms")
def insertionSort(a):
    n=len(a)
    start = time.time()
    for i in range(1,n):
        val=a[i]
        j=i-1
        while j>=0 and a[j]>val:
            a[j+1]=a[j]
            j=j-1
        a[j+1]=val
    end = time.time()
    print("\n\nExecution Time:",end-start,"ms")

n=int(input("Enter Size of the List\n"))

inputList=list()
print("Enter List elements")
for i in range(n):
    num=int(input())
    inputList.append(num)
print("----------------------------------------")
```

```
a=inputList.copy()
print("Bubble Sort\nUnsorted List")
for i in a:
   print(i,end="\t")
bubbleSort(a)
print("\nSorted List")
for i in a:
   print(i,end="\t")
print("\n-----------------------------------------")

a=inputList.copy()
print("Selection Sort\nUnsorted List")
for i in a:
   print(i,end="\t")
selectionSort(a)
print("\nSorted List")
for i in a:
   print(i,end="\t")
print("\n-----------------------------------------")

a=inputList.copy()
print("Insertion Sort\nUnsorted List")
for i in a:
   print(i,end="\t")
insertionSort(a)
print("\nSorted List")
for i in a:
   print(i,end="\t")
print("\n-----------------------------------------")
```

Output:

Algorithmic Solution:

Week 4 Practice

Exp #1: Implement Binary Search and compute its time complexity.

Python Program:

```
import time
def binarySearch(a, key):
    start=0
    end=len(a)-1
    while start<=end:
        mid=(start+end)//2
        if key == a[mid]:
            return True
        elif key<a[mid]:
            end=mid-1
        else:
            start=mid+1
    return False

n=int(input("Enter Size of the List\n"))

a=list()
print("Enter List elements")
for i in range(n):
    num=int(input())
    a.append(num)

key=int(input("Enter Key element to be searched\n"))

print("\nSearch List: ", a)
print("Key:", key)

start=time.time()
res=binarySearch(a,key)
end=time.time()

if res==True:
    print("\nSuccessful Search")
else:
    print("\nUnsuccessful Search")

print("Execution Time:",end-start,"seconds")
```

Output:

Algorithmic Solution:

Week 4 Practice

Exp #2: Implement Merge & Quick Sort algorithms, compute its time complexity.

Python Program:

```
import time

def mergeLists(A,B):
    i=0
    j=0
    newList = list()
    while i<len(A) and j <len(B):
        if A[i]< B[j]:
            newList.append(A[i])
            i=i+1
        else:
            newList.append(B[j])
            j=j+1
    while i < len(A):
        newList.append(A[i])
        i=i+1
    while j < len(B):
        newList.append(B[j])
        j=j+1
    return newList

def mergeSort(a):
    if len(a)<=1:
        return a
    else:
        mid=len(a)//2
        lefthalf=mergeSort(a[:mid])
        righthalf=mergeSort(a[mid:])
        newList=mergeLists(lefthalf,righthalf)
        return newList

def partition(a,l,r):
    pivot=l
    i=l+1
    j=r
    while(i<=j):
        while i<=j and a[i] <= a[pivot]:
            i=i+1
        while a[j] > a[pivot]:
            j=j-1
        if i < j:
            temp=a[i]
            a[i]=a[j]
            a[j]=temp
    temp=a[pivot]
    a[pivot]=a[j]
    a[j]=temp
    return j
```

```
def quickSort(a,i,j):
    if i<j:
        s=partition(a,i,j)
        quickSort(a,i,s-1)
        quickSort(a,s+1,j)

n=int(input("Enter Size of the List\n"))

a=list()
print("Enter List elements")
for i in range(n):
    num=int(input())
    a.append(num)
print("-----------------------------------")

print("Merge Sort\nUnsorted List")
for i in a:
    print(i,end="\t")

start=time.time()
ns=mergeSort(a)
end=time.time()
print("\n\nSorted List")
for i in ns:
    print(i,end="\t")
print("\nExecution Time:",end-start,"Seconds")
print("----------------------------------------")

print("Quick Sort\nUnsorted List")
for i in a:
    print(i,end="\t")

start=time.time()
quickSort(a,0,n-1)
end=time.time()
print("\n\nSorted List")
for i in a:
    print(i,end="\t")
print("\nExecution Time:",end-start,"Seconds")
print("----------------------------------------")
```

Output:

Algorithmic Solution:

Week 4 Practice

Exp #3: Implement Fibonacci Sequence with dynamic programming.

Python Program:

```
def Fib(n):
    f1=0
    f2=1
    a=list()
    i=2
    a.append(f1)
    a.append(f2)
    for i in range(2,n):
        f3=f1+f2
        a.append(f3)
        f1=f2
        f2=f3
    return a

n=int(input("Enter size of Fibonacci Sequence\n"))

fibList = Fib(n)
print("Fibonacci Sequence:")
for i in fibList:
    print(i,end="\t")
```

Output:

Algorithmic Solution:

Week 5 Practice

Exp #1: Implement Singly Linked List.

Python Program:

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

class LinkedList:
    def __init__(self,value):
        self.head = Node(value)

    def prepend(self,value):
        newNode=Node(value)
        newNode.next=self.head
        self.head = newNode
        print(value,"Prepended to the List")

    def insert(self,value,index):
        newNode = Node(value)
        predNode = None
        curNode = self.head
        i=0
        while curNode is not None and i !=index:
            predNode = curNode
            curNode = curNode.next
            i=i+1
        predNode.next=newNode
        newNode.next=curNode
        print(value,"inserted to the List at the index:",index)

    def printLL(self):
        curNode = self.head
        print()
        print("List Items are")
        while curNode is not None:
            print(curNode.data,end="-->")
            curNode = curNode.next

    def remove(self,target):
        predNode = None
        curNode = self.head
        while curNode is not None and curNode.data !=target:
            predNode = curNode
            curNode = curNode.next
        if curNode is not None:
            print("\n",target," deleted from the List",sep="")
            if curNode is self.head:
                self.head=curNode.next
            else:
                predNode.next = curNode.next
        else:
            print("\n",target," not found in the list",sep="")
```

```
    def search(self,target):
        curNode = self.head
        while curNode is not None and curNode.data !=target:
            curNode = curNode.next
        return curNode is not None


LL = LinkedList(10)
LL.prepend(20)
LL.prepend(30)
LL.prepend(5)
LL.insert(15,2)
LL.printLL()

LL.remove(20)
LL.printLL()

LL.remove(10)
LL.remove(100)
print("5 exists in List:",LL.search(5))
print("20 exists in List:",LL.search(20))
LL.printLL()
```

Output:

Algorithmic Solution:

Week 6 Practice

Exp #1: Implement linked list Iterators.

Python Program:

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

class LinkedList:
    def __init__(self,value):
        self.head = Node(value)

    def append(self,value):
        newNode = Node(value)
        curNode = self.head
        while curNode.next is not None:
            curNode = curNode.next
        curNode.next= newNode


class LLIterator:
    def __init__(self,head):
        self.curNode=head

    def __iter__(self):
        return self

    def __next__(self):
        if self.curNode is None:
            raise StopIteration
        else:
            item=self.curNode.data
            self.curNode = self.curNode.next
            return item

LL = LinkedList(10)
LL.append(25)
LL.append(20)
LL.append(30)
L = LLIterator(LL.head)
print("Data Items in Linked List are")
for i in L:
    print(i)
```

Output:

Algorithmic Solution:

Week 7 Practice

Exp #1: Implement Doubly Linked List.

Python Program:

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

class LinkedList:
    def __init__(self,value):
        self.head = Node(value)
        self.tail = self.head

    def append(self,value):
        newNode = Node(value)
        curNode = self.head
        while curNode.next is not None:
            curNode = curNode.next
        curNode.next= newNode
        newNode.prev = curNode
        self.tail=newNode
        print("Node",value,"appended.")
    def traversal(self):
        curNode = self.head
        if curNode is None:
            print("Doubly Linked List is empty.")
        else:
            print("Contents of Doubly Linked List are")
            while curNode is not None:
                print(curNode.data,end="\t")
                curNode = curNode.next
            print( )
    def remove(self,target):
        curNode = self.head
        while curNode is not None and curNode.data !=target:
            curNode = curNode.next
        if curNode is not None:
            if curNode is self.head:
                curNode.next.prev=None
                self.head=curNode.next
            elif curNode is self.tail:
                curNode.prev.next=None
                self.tail=curNode.prev
            else:
                curNode.prev.next = curNode.next
                curNode.next.prev = curNode.prev
            print("Node",target,"deleted.")
        else:
            print(target,"not found in the list")
```

```
    def search(self,target):
        curNode = self.head
        while curNode is not None:
            if curNode.data == target:
                return True
            else:
                curNode = curNode.next
        return False

LL = LinkedList(10)
LL.traversal()
LL.append(20)
LL.append(30)
LL.traversal()
print("Node 40 exits?:",LL.search(40))
print("Node 20 exits?:",LL.search(20))
print("Node 10 exits?:",LL.search(30))
LL.remove(10)
LL.traversal()
LL.remove(30)
LL.traversal()
LL.remove(50)
LL.traversal()
```

Output:

Algorithmic Solution:

Week 7 Practice

Exp #2: Implement Circular Doubly Linked List.

Python Program:

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

class LinkedList:
    def __init__(self,value):
        self.head = Node(value)
        self.tail=self.head
        self.head.next=self.head
        self.head.prev=self.head

    def append(self,value):
        newNode = Node(value)
        curNode = self.head
        while curNode is not self.tail:
            curNode = curNode.next
        curNode.next= newNode
        newNode.prev = curNode
        newNode.next = self.head
        self.head.prev=newNode
        self.tail=newNode
        print("Node",value,"appended")

    def traversal(self):
        curNode = self.head
        if curNode is None:
            print("Circular Doubly Linked List is empty")
        else:
            print("Contents of Circular Doubly Linked List are")
            while curNode is not self.tail:
                print(curNode.data,end="\t")
                curNode = curNode.next
            print(curNode.data)

    def remove(self,target):
        curNode = self.head
        while curNode is not self.tail and curNode.data !=target:
            curNode = curNode.next
        if curNode is not self.tail:
            if curNode is self.head:
                curNode.next.prev=self.tail
                self.head=curNode.next
                self.tail.next=self.head
            else:
                curNode.prev.next = curNode.next
                curNode.next.prev = curNode.prev
            print("Node",target,"deleted")
```

```
            elif curNode is self.tail:
                    self.tail.prev.next=self.head
                    self.head.prev=self.tail.prev
                    self.tail=self.tail.prev
                    print("Node",target,"deleted")
             else:
                 print(target,"not found in the list")

LL = LinkedList(10)
LL.traversal()
LL.append(20)
LL.append(30)
LL.traversal()
LL.remove(20)
LL.traversal()
LL.remove(30)
LL.traversal()
```

Output:

Algorithmic Solution:

Week 8 Practice

Exp #1: Implement Stack Data Structure.

Python Program:

```
class Stack:
    def __init__(self):
        self.items = list()

    def push(self,value):
        self.items.append(value)

    def pop(self):
        if self.isEmpty() == True:
            print("Stack is empty, cannot remove")
        else:
            return self.items.pop()

    def peek(self):
        if self.isEmpty() == True:
            print("Stack is empty")
        else:
            return self.items[-1]

    def display(self):
        print("Stack items are")
        for i in self.items:
            print(i,end="\t")

    def isEmpty(self):
        if len(self.items) == 0:
            return True
        else:
            return False

    def length(self):
        return len(self.items)

S = Stack()
S.push(10)
S.push(20)
S.push(30)
S.display()
print("\nTop Item of the Stack:",S.peek())
print("Length: ", S.length())
print("Deleted Item:",S.pop())
S.display()
print("\nTop Item of the Stack:",S.peek())
print("Deleted Item:",S.pop())
S.display()
print("\nTop Item of the Stack:",S.peek())
print("Stack Empty?: ",S.isEmpty())
print("Deleted Item:",S.pop())
print("Stack Empty?: ",S.isEmpty())
print("Length: ", S.length())
```

Output:

Algorithmic Solution:

Week 8 Practice

Exp #2: Implement Bracket matching using Stack.

Python Program:

```
class Stack:
    def __init__(self):
        self.items = list()

    def push(self,value):
        self.items.append(value)

    def pop(self):
        if self.isEmpty() == True:
            print("Stack is empty, cannot remove")
        else:
            return self.items.pop()

    def isEmpty(self):
        if len(self.items) == 0:
            return True
        else:
            return False

    def length(self):
        return len(self.items)

def isBalanced(text):
    S = Stack()
    for char in text:
        if char in ["[","{","("]:
            S.push(char)
        elif char in ["]","}",")"]:
            temp = S.pop()
            if temp == "(":
                if char != ")":
                    return False
            if temp == "{":
                if char != "}":
                    return False
            if temp == "[":
                if char != "]":
                    return False
        else:
            continue
    if S.isEmpty() == True:
        return True

text = input("Enter Text\n")
res=isBalanced(text)
if res==True:
    print("Brackets are matched\nValid and Balnaced Expression")
else:
    print("Brackets are not matched\nInvalid expression")
```

Output:

Algorithmic Solution:

Week 9 Practice

Exp #1: Program to demonstrate recursive operations (Factorial/Fibonacci).

Python Program:

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

def fib(n):
    if n>=1:
        if n==1 or n==0:
            return 1
        else:
            return fib(n-1) + fib(n-2)
    else:
        return 0

num=int(input("Enter positive integer number\n"))
res1=factorial(num)
print("Facorial of",num,"is",res1)
res2=fib(num)
print("{}th term of Fibonacci series is {}".format(num,res2))
```

Output:

Algorithmic Solution:

Week 9 Practice

Exp #2: Implement solution for Towers of Hanoi.

Python Program:

```
def move(n,src,dest,temp):
    if n >= 1:
        move(n-1,src,temp,dest)
        print("Move {} -> {}".format(src,dest))
        move(n-1,temp,dest,src)

num=int(input("Enter number of discs\n"))
move(num,1,3,2)
```

Output:

Algorithmic Solution:

Week 10 Practice

Exp #1: Implement Queue.

Python Program:

```
class Queue:
    def __init__(self):
        self.items = list()

    def enqueue(self,value):
        self.items.append(value)
        print(value,"inserted.")

    def dequeue(self):
        if self.isEmpty() == True:
            print("Queue is empty, cannot remove")
        else:
            return self.items.pop(0)

    def display(self):
        if self.isEmpty() == True:
            print("Queue is empty")
        else:
            print("Queue items are")
            for i in self.items:
                print(i,end="\t")
                print()

    def isEmpty(self):
        if len(self.items) == 0:
            return True
        else:
            return False

    def length(self):
        return len(self.items)

Q = Queue()
Q.enqueue(10)
Q.enqueue(20)
Q.enqueue(40)
Q.enqueue(50)
Q.display()
print("Length: ", Q.length())
print(Q.dequeue(),"deleted")
Q.display()
Q.enqueue(30)
print(Q.dequeue(),"deleted")
Q.display()
print("Oueue Empty?: ",Q.isEmpty())
print(Q.dequeue(),"deleted")
print("Queue Empty?: ",Q.isEmpty())
print("Queue Length: ", Q.length())
```

Output:

Algorithmic Solution:

Week 10 Practice

Exp #2: Implement Priority Queue.

Python Program:

```
class PQItem:
    def __init__(self,value,priority):
        self.value=value
        self.priority=priority

class PriorityQueue:
    def __init__(self):
        self.items = list()

    def enqueue(self,value,priority):
        item=PQItem(value,priority)
        self.items.append(item)
        print(value,"with priority:",priority,"inserted")

    def dequeue(self):
        if self.isEmpty() == True:
            print("Queue is empty, cannot remove")
        else:
            highest = self.items[0].priority
            index = 0
            for i in range(1,len(self.items)):
                if self.items[i].priority < highest:
                    highest = self.items[i].priority
                    index = i
            item = self.items.pop(index)
            return item.value

    def display(self):
        if self.isEmpty() == True:
            print("Queue is empty")
        else:
            print("Queue items are")
            print("------------------")
            print("Value \t Priority")
            print("------------------")
            for i in self.items:
                print(i.value,"\t",i.priority)
                print("------------------")

    def isEmpty(self):
        if len(self.items) == 0:
            return True
        else:
            return False

    def length(self):
        return len(self.items)
```

```
Q = PriorityQueue()
print("Queue Empty?:",Q.isEmpty())
Q.enqueue(10,2)
Q.enqueue(20,0)
Q.enqueue(30,1)
Q.enqueue(40,3)
Q.display()
print("Queue Empty?:",Q.isEmpty())
print("Queue Length:", Q.length())
print("Deleted Item:",Q.dequeue())
print("Deleted Item:",Q.dequeue())
print("Deleted Item:",Q.dequeue())
Q.display()
print("Deleted Item:",Q.dequeue())
print("Deleted Item:",Q.dequeue())
Q.display()
print("Queue Empty?:",Q.isEmpty())
print("Queue Length:", Q.length())
```

Output:

Algorithmic Solution:

Week 11 Practice

Exp #1: Implement Binary Search Tree and its operations using list.

Python Program:

```
class Node:
   def __init__(self,value):
      self.data = value
      self.left = None
      self.right =None

class BST:
   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
      print(value,"inserted")

   def delete(self,key):
      curNode = self.root
      parentNode =None
      while curNode is not None:
         if key == curNode.data:
            if temp=="Left":
               parentNode.left=None
            else:
               parentNode.right=None
            print(key,"Node Deleted")
            return True
         elif key <curNode.data:
            parentNode = curNode
            curNode = curNode.left
            temp="Left"
         else:
            parentNode = curNode
            curNode=curNode.right
            temp="Right"
      print(key,"Node not found")
      return False
```

```
    def search(self,key):
        curNode = self.root
        while curNode is not None:
            if key == curNode.data:
                return True
            elif key <curNode.data:
                curNode = curNode.left
            else:
                curNode=curNode.right
        return False

    def preorder(self, rt):
        print(rt.data, end="\t")
        if rt.left is not None:
            self.preorder(rt.left)
        if rt.right is not None:
            self.preorder(rt.right)

    def inorder(self, rt):
        if rt.left is not None:
            self.inorder(rt.left)
        print(rt.data, end="\t")
        if rt.right is not None:
            self.inorder(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="\t")

BT = BST()
ls = [25,10,35,20,65,45,24]
for i in ls:
    BT.insert(i)
print("\nPre-order Traversal")
BT.preorder(BT.root)
print("\nIn-order Traversal")
BT.inorder(BT.root)
print("\nPost-order Traversal")
BT.postorder(BT.root)
print("\n35 exists:", BT.search(35))
print("65 exists:", BT.search(65))
BT.delete(75)
BT.delete(24)
print("In-order Traversal")
BT.inorder(BT.root)
```

Output:

Algorithmic Solution:

Week 12 Practice

Exp #1: Implementation of BFS.

Python Program:

```
class Queue:
    def __init__(self):
        self.items=list()

    def enqueue(self,value):
        self.items.append(value)

    def dequeue(self):
        if len(self.items) != 0:
            return self.items.pop(0)

    def isEmpty(self):
        if len(self.items) == 0:
            return True
        else:
            return False

class Node:
    def __init__(self,value):
        self.data = value
        self.left = None
        self.right =None

class BST:
    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
        print(value,"inserted")
```

```
def BFS(root):
    Q = Queue()
    Q.enqueue(root)
    while Q.isEmpty() != True:
        node=Q.dequeue()
        print(node.data,end="\t")
        if node.left is not None:
            Q.enqueue(node.left)
        if node.right is not None:
            Q.enqueue(node.right)

BT = BST()
ls = [25,10,35,20,5,30,40]
for i in ls:
    BT.insert(i)
print("BFS Traversal")
BFS(BT.root)
```

Output:

Algorithmic Solution:

Week 12 Practice

Exp #2: Implementation of DFS.

Python Program:

```
class Stack:
    def __init__(self):
        self.items=list()

    def push(self,value):
        self.items.append(value)

    def pop(self):
        if len(self.items) != 0:
            return self.items.pop()

    def isEmpty(self):
        if len(self.items) == 0:
            return True
        else:
            return False

class Node:
    def __init__(self,value):
        self.data = value
        self.left = None
        self.right =None

class BST:
    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
        print(value,"inserted")
```

```
def DFS(root):
    S = Stack()
    S.push(root)
    while S.isEmpty() != True:
        node=S.pop()
        print(node.data,end="\t")
        if node.right is not None:
            S.push(node.right)
        if node.left is not None:
            S.push(node.left)

BT = BST()
ls = [25,10,35,20,5,30,40]
for i in ls:
    BT.insert(i)
print("\nDFS Traversal")
DFS(BT.root)
```

Output:

Algorithmic Solution:

Week 13 Practice

Exp #1: Implement Hash Functions.

Python Program:

```
class Hash:
    def __init__(self):
        self.buckets=[[],[],[],[],[]]

    def insert(self,key):
        buc_index = key % 5
        self.buckets[buc_index].append(key)
        print(key,"inserted in Bucket No.",buc_index+1)

    def search(self,key):
        buc_index = key % 5
        if key in self.buckets[buc_index]:
            print(key,"present in bucket No.",buc_index + 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()
print("Hash operations\n\t1.Insert\n\t2.Search\n\t3.Display\n\t4.Quit")
ch=int(input("Enter your choice\n"))
while ch in [1,2,3]:
    if ch == 1:
        key=int(input("\nEnter key to be inserted\n"))
        hsh.insert(key)
        print("----------------------------------------")
    elif ch == 2:
        key=int(input("\nEnter key to be searched\n"))
        hsh.search(key)
        print("----------------------------------------")
    elif ch == 3:
        hsh.display()
        print("\n----------------------------------------")
    print("\nHash operations\n\t1.Insert\n\t2.Search\n\t3.Display\n\t4.Quit")
    ch=int(input("Enter your choice\n"))
```

Output:

Algorithmic Solution: