DATA STRUCTURES USING PYTHON LABORATORY

Lab Manual for II Year B. Tech (IT) Students

LIST OF EXPERIMENTS:

1. Implement simple ADTs as Python classes
2. Implement recursive algorithms in Python
3. Implement List ADT using Python arrays
4. Linked list implementations of List
5. Implementation of Stack and Queue ADTs
6. Applications of List, Stack and Queue ADTs
7. Implementation of sorting and searching algorithms
8. Implementation of Hash tables
9. Tree representation and traversal algorithms
10. Implementation of Binary Search Trees
11. Implementation of Heaps
12. Graph representation and Traversal algorithms
13. Implementation of single source shortest path algorithm
14. Implementation of minimum spanning tree algorithms

Ex.No:01	IMPLEMENT SIMPLE ADTs AS PYTHON CLASSES
Date:	

AIM:

To write a python program to insert, delete and display elements of list using Classes.

ALGORITHM:

- Create a class and using a constructor initialize values for a list.
- Create methods for inserting, deleting and displaying elements of the list.
- Create an object for the class.
- Using the object, call the respective function depending on the choice taken from the user.
- Print the final list.
- Exit

RESOURCE:

Python 3.7.3

PROGRAM:

```
class list_ADT():
    def __init__(self):
        self.l = []

    def insert_at_end(self, a):
        self.l.append(a)

    def insert_at_begininning(self, a):
        self.l.insert(0, a)

    def insert_at_middle(self, ind, a):
        self.l.insert(ind, a)

    def remove(self, b):
        self.l.remove(b)

    def display(self):
        return (self.l)

print("Implementation of List using Python Class")
obj = list_ADT()

while(1):
    print("1.INSERT AT BEGINNING")
    print("2.INSERT AT MIDDLE")
    print("3.INSERT AT END")
    print("4.DELETE AN ELEMENT")
    print("5.DISPLAY THE LIST")
    print("6.Exit")

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

    if choice == 1:
        n = int(input("Enter element to insert: "))
        obj.insert_at_begininning(n)
    elif choice == 2:
        n = int(input("Enter element to insert: "))
        pos = int(input("Enter at which index element to insert: "))
        obj.insert_at_middle(pos, n)
    elif choice == 3:
        n = int(input("Enter element to append: "))
        obj.insert_at_end(n)
    elif choice == 4:
        n = int(input("Enter element to remove: "))
        obj.remove(n)
    elif choice == 5:
        print("List is ", obj.display())
    else:
        print("PROGRAM EXIT")
        break
```

OUTPUT:

```
Implementation of List using Python Class
1.INSERT AT BEGINNING
2.INSERT AT MIDDLE
3.INSERT AT END
4.DELETE AN ELEMENT
5.DISPLAY THE LIST
6.Exit
Enter choice: 1
Enter element to insert: 1
1.INSERT AT BEGINNING
2.INSERT AT MIDDLE
3.INSERT AT END
4.DELETE AN ELEMENT
5.DISPLAY THE LIST
6.Exit
Enter choice: 3
Enter number to append: 5
1.INSERT AT BEGINNING
2.INSERT AT MIDDLE
3.INSERT AT END
4.DELETE AN ELEMENT
5.DISPLAY THE LIST
6.Exit
Enter choice: 2
Enter element to insert: 3
Enter at which index element to insert:1
1.INSERT AT BEGINNING
2.INSERT AT MIDDLE
3.INSERT AT END
4.DELETE AN ELEMENT
5.DISPLAY THE LIST
6.Exit
Enter choice: 5
List is  [1, 3, 5]
1.INSERT AT BEGINNING
```

```
2.INSERT AT MIDDLE
3.INSERT AT END
4.DELETE AN ELEMENT
5.DISPLAY THE LIST
6.Exit
                    Enter choice: 6
                    PROGRAM EXIT.
```

RESULT:

Thus, the Python program that appends, deletes and displays elements of a list using classes was written and executed successfully.

Ex.No:02.a)	**IMPLEMENT RECURSIVE ALGORITHMS IN PYTHON FOR FACTORIAL NUMBER**
Date:	

AIM:

To write a python program to find the factorial of a number using recursion.

ALGORITHM:

- Get a number from the user and store it in a variable.
- Pass the number as an argument to a recursive factorial function.
- Define the base condition, if the number is equal to 0 or 1 then return 1.
- Otherwise call the function recursively with the number minus 1 multiplied by the number itself.
- Then return the result and print the factorial of the number.
- Exit.

RESOURCE:

Python 3.7.3

PROGRAM:

```
def recur_factorial(n):
   if n == 1 or n == 0:
       return 1
   else:
       return n * recur_factorial(n - 1)

num = int(input("Enter a number: "))
if num < 0:
   print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   print("The factorial of", num, "is", recur_factorial(num))
```

OUTPUT:

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

Enter a number:-2
        Sorry, factorial does not exist for negative numbers
```

RESULT:

Thus, the Python program to find the factorial of a number was written and executed successfully.

Ex.No:02.b)	IMPLEMENT RECURSIVE ALGORITHMS IN PYTHON FOR FIBONACCI SERIES

AIM:

To write a python program to find the Fibonacci Series using recursion.

ALGORITHM:

- Get a number from the user and store it in a variable.
- Pass the number as an argument to a recursive factorial function.
- Define the base condition, if the number is equal to 0 or 1 then return 1.
- Otherwise call the function recursively with the number minus 1 and 2 and add them.
- Then return the result and print the fibonacci series.
- Exit.

RESOURCE:

Python 3.7.3

PROGRAM:

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

terms = int(input("Enter no. of terms of Fibonacci Series: "))
if terms <= 0:
   print("Please enter a positive integer")
else:
   print("Fibonacci series:")
   for i in range(terms):
       print(recur_fibonacci(i), end=" ")
```

OUTPUT:

```
Enter number of terms of Fibonacci Series:8
Fibonacci series:

0 1 1 2 3 5 8 1 3
```

RESULT:

Thus, the Python program to find the Fibonacci series was written and executed successfully.

Ex.No:03	IMPLEMENT LIST ADT USING PYTHON ARRAYS
Date:	

AIM:

To write a python program to implement List ADT using Array in Python.

ALGORITHM:

- Create a class and using a constructor initialize values for a list.
- Create methods for inserting ,deleting and displaying elements of the list.
- Create an object for the class.
- Using the object, call the respective function depending on the choice taken from the user.
- Print the final list.
- Exit.

RESOURCE:

Python 3.7.3

PROGRAM:

```
import array
arr = array.array('i', [])
print("The newly created array is: ", end=" ")
while(1):
   print("\n1. Insert at end")
   print("2. Insert at beginning")
   print("3. Insert at middle")
   print("4. Delete an element")
   print("5. Display")
   print("6. Exit")

   ch = int(input("Enter your choice: "))

   if ch == 1:
        x = int(input("Enter an element to insert at the end: "))
        arr.append(x)
    elif ch == 2:
        x = int(input("Enter an element to insert at the beginning: "))
        arr.insert(0, x)
    elif ch == 3:
        x = int(input("Enter an element to insert in the list: "))
        y = int(input("Enter the index position to insert the element: "))
        arr.insert(y, x)
    elif ch == 4:
        x = int(input("Enter the element to remove: "))
        arr.remove(x)
    elif ch == 5:
        print("Current array:", arr)
    elif ch == 6:
        print("Program Exit")
        break
    else:
        print("Invalid choice. Try again!")
```

OUTPUT:

```
The new created array is:
1. Insert at end
2. Insert at beginning
3. Insert at Middle
4. Delete an element
5. Display
Enter your choice: 1
Enter an element to insert in the list: 5

1. Insert at end
2. Insert at beginning
3. Insert at Middle
4. Delete an element
5. Display
Enter your choice: 2
Enter an element to insert in the list: 1
```

```
1. Insert at end
2. Insert at beginning
3. Insert at Middle
4. Delete an element
5. Display
Enter your choice: 3
Enter an element to insert in the list: 2
Enter the index position to insert the element: 1

1. Insert at end
2. Insert at beginning
3. Insert at Middle
4. Delete an element
5. Display
Enter your choice: 5
Array ('i', [1, 2, 5])

1. Insert at end
2. Insert at beginning
3. Insert at Middle
4. Delete an element
5. Display
Enter your choice: 4
Enter the element to remove: 2

1. Insert at end
2. Insert at beginning
3. Insert at Middle
4. Delete an element 5. Display
Enter your
choice: 5 array
('i', [1, 5])

1. Insert at end
2. Insert at beginning
3. Insert at Middle
4. Delete an element 5. Display
Enter your choice: 6
```

```
        Program Exit
```

RESULT:

Thus the python program for List ADT using Array in Python was written and executed successfully.

Ex.No:04	**LINKED LIST IMPLEMENTATIONS OF LIST**
Date:	

AIM:

To write a python program to implement list using linked list.

ALGORITHM:

- Create a class node and initialize data and next of class Node are none.
- Create a class linked list where all the nodes can be linked.
- Create methods for inserting a node at beginning, end and middle.
- Create a method for deleting an element based on the index.
- Create a method to display the elements of the linked list.
- Create an object for the class linked list.
- Using the object, call the respective function depending on the choice taken from the user.
- Print the final list.
- Exit.

RESOURCE:

Python 3.7.3

PROGRAM:

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

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

   def insert_at_beginning(self, newElement):
      newNode = Node(newElement)
      newNode.next = self.head
```

```
    self.head = newNode

def insert_at_end(self, newElement):
    newNode = Node(newElement)
    if self.head is None:
        self.head = newNode
        return
    temp = self.head
    while temp.next is not None:
        temp = temp.next
    temp.next = newNode

def insert_at_middle(self, newElement, position):
    newNode = Node(newElement)
    if position < 0:
        print("\nPosition should be >= 0.")
    elif position == 0:
        newNode.next = self.head
        self.head = newNode
    else:
        temp = self.head
        for i in range(1, position):
            if temp is not None:
                temp = temp.next
        if temp is not None:
            newNode.next = temp.next
            temp.next = newNode
        else:
            print("\nThe previous node is null.")

def delete(self, position):
    if position < 0:
        print("\nPosition should be >= 0.")
    if self.head is not None:
        if position == 0:
            temp = self.head
            self.head = self.head.next
```

```
                temp = None
            else:
                temp = self.head
                for i in range(1, position):
                    if temp is not None:
                        temp = temp.next
                if temp is not None and temp.next is not None:
                    nodeToDelete = temp.next
                    temp.next = temp.next.next
                    nodeToDelete = None
                else:
                    print("\nThe node is already null.")

    def display(self):
        temp = self.head
        if temp is not None:
            print("The list contains:", end=" ")
            while temp is not None:
                print(temp.data, end=" ")
                temp = temp.next
            print()
        else:
            print("The list is empty.")

ll = LinkedList()

while True:
    print("1. Insert at Beginning")
    print("2. Insert at Middle")
    print("3. Insert at End")
    print("4. Delete an Element")
    print("5. Display the Elements")
    print("6. Exit")

    choice = int(input("Enter choice: "))
    if choice == 1:
        n = int(input("Enter element to insert: "))
    ll.insert_at_beginning(n)
elif choice == 2:
    n = int(input("Enter element to insert: "))
    pos = int(input("Enter at which index element to insert: "))
    ll.insert_at_middle(n, pos)
elif choice == 3:
    n = int(input("Enter number to append: "))
    ll.insert_at_end(n)
elif choice == 4:
    n = int(input("Enter the index of the element to be removed: "))
    ll.delete(n)
elif choice == 5:
    ll.display()
elif choice == 6:
    print("PROGRAM EXIT")
    break
else:
    print("Invalid choice. Try again!")
```

OUTPUT:

```
1. Insert at Beginning
2. Insert at Middle
3. Insert at End
4. Delete an Element
5. Display the elements
6. Exit
Enter choice: 1
Enter element to insert: 10
1. Insert at Beginning
2. Insert at Middle
3. Insert at End
4. Delete an Element
5. Display the elements
6. Exit
Enter choice: 3
Enter number to append: 50
1. Insert at Beginning
2. Insert at Middle
```

```
3. Insert at End
4. Delete an Element
5. Display the elements
6. Exit
Enter choice: 2
Enter element to insert: 30
Enter at which index element to insert: 1
1. Insert at Beginning
2. Insert at Middle
3. Insert at End
4. Delete an Element
5. Display the elements
6. Exit
Enter choice: 5
The list contains: 10 30 50
List is none
1. Insert at Beginning
2. Insert at Middle
3. Insert at End
4. Delete an Element
5. Display the elements
6. Exit
Enter choice: 4
Enter the index of the element to be removed: 1
1. Insert at Beginning
2. Insert at Middle
3. Insert at End
4. Delete an Element
5. Display the elements
6. Exit
Enter choice: 5
The list contains: 10 50
List is none
1. Insert at Beginning
2. Insert at Middle
3. Insert at End
4. Delete an Element
5. Display the elements
```

```
6. Exit
Enter choice: 6
PROGRAM EXIT
```

RESULT:

Thus the python program to implement list using linked list was written and executed successfully.

Ex.No:05. a)	**IMPLEMENTATION OF STACK ADT**
Date:	

AIM:

To write a python program to implement Stack ADT.

ALGORITHM:

- Create a method to initialize an empty stack.
- Create a method to push the elements into the stack using append ().
- Create a method to pop the element at the top of the stack using pop ().
- Create a method to find the top of the stack.
- Create a list by calling its respective method.
- Depending on the choice taken from the user call the respective method Print the final content of the stack.
- Exit.

RESOURCE:

Python 3.7.3

PROGRAM:

```
def createstack():
   stack = []
   return stack

def push(stack, newElement):
   stack.append(newElement)
   print(newElement, "is added into the
stack.")

def pop(stack):
   return stack.pop()

def topElement(stack):
   return stack[len(stack) - 1]

print("Creating a Stack")
mystack = createstack()
print(mystack)

while True:
   print("\n1. PUSH")
   print("2. POP")
   print("3. DISPLAY")
   print("4. FIND THE TOP OF THE STACK")
   print("5. EXIT")
   choice = int(input("Enter your choice: "))

   if choice == 1:
       ele = int(input("Enter an element to push
into stack: "))
       push(mystack, ele)
   elif choice == 2:
       print(pop(mystack), "is popped from the
stack.")
   elif choice == 3:
       print("The stack contains", mystack)
   elif choice == 4:
       print("Top of the stack is",
topElement(mystack))
   elif choice == 5:
       print("PROGRAM EXIT")
       break
   else:
       print("Invalid choice. Please try again.")
```

OUTPUT:

```
Creating a Stack
1. PUSH
```

```
2. POP
3. DISPLAY
4. FIND THE TOP OF THE STACK
5. EXIT
Enter your choice: 1 Enter an element
to push into stack: 10 10 is added into
the stack.
1. PUSH
2. POP
3. DISPLAY
4. FIND THE TOP OF THE STACK
5. EXIT
Enter your choice: 1 Enter an element
to push into stack: 20 20 is added into
the stack.
1. PUSH
2. POP
3. DISPLAY
4. FIND THE TOP OF THE STACK
5. EXIT
Enter your choice: 1 Enter an element
to push into stack: 30 30 is added into
the stack.
1. PUSH
2. POP
3. DISPLAY
4. FIND THE TOP OF THE STACK
5. EXIT
Enter your choice: 1 Enter an element
to push into stack: 40 40 is added into
the stack.
1. PUSH
2. POP
3. DISPLAY
4. FIND THE TOP OF THE STACK
5. EXIT
Enter your choice: 3
The stack contains [10, 20, 30, 40]
1. PUSH
```

```
2. POP
3. DISPLAY
4. FIND THE TOP OF THE STACK
5. EXIT Enter your choice: 2
40 is popped from the stack.
1. PUSH
2. POP
3. DISPLAY
4. FIND THE TOP OF THE STACK
5. EXIT
Enter your choice: 5
PROGRAM EXIT
```

RESULT:

Thus the python program to implement Stack ADT was written and executed successfully.

Ex.No:05. b)	**IMPLEMENTATION OF QUEUE ADT**
Date:	

AIM:

To write a python program to implement Queue ADT.

ALGORITHM:

- Create a method to initialize an empty queue.
- Create a method to push the elements into the queue using append ().
- Create a method to pop the element at the top of the queue using pop ().
- Create a list by calling its respective method.
- Depending on the choice taken from the user call the respective method.
- Print the final content of the queue.
- Exit.

RESOURCE:

Python 3.7.3

PROGRAM:

```
def createqueue():
   que = []
   return que

def enqueue(que,
newElement):
   que.append(newElement)
   print(newElement, "is
enqueued.")

def dequeue(que):
    print(que.pop(0), "is
dequeued.")

print("Creating a Queue")
q = createqueue()
print(q)

while(1):
    print("\n1. Enqueue")
    print("2. Dequeue")
    print("3. Display")
    print("4. Exit")

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

    if choice == 1:
        ele = int(input("Enter an
element to push into queue:
"))
        enqueue(q, ele)
    elif choice == 2:
        dequeue(q)
    elif choice == 3:
        print("The queue
contains", q)
    elif choice == 4:
        print("PROGRAM EXIT")
        break
    else:
        print("Invalid choice.
Please try again.")
```

OUTPUT:

```
Creating a Queue
1.Enqueue
2.Dequeue
3.Display
```

```
4.Exit
Enter your choice:1
Enter an element to push into queue:10 10
is enqueued.
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:1
Enter an element to push into queue:15 15
is enqueued.
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:1
Enter an element to push into queue:20 20
is enqueued.
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:1
Enter an element to push into queue:25 25
is enqueued.
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:3
The queue contains [10, 15, 20, 25]
1.Enqueue
2.Dequeue
3.Display
4.Exit Enter your
choice:2 10 is
dequeued.
1.Enqueue
2.Dequeue
```

```
3.Display
4.Exit
Enter your choice:3
The queue contains [15, 20, 25]
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:4
PROGRAM EXIT
```

RESULT:

Thus the python program to implement Queue ADT was written and executed successfully.

Ex.No:06. a)	POLYNOMIAL ADDITION
Date:	

AIM:

To write a python program to perform Polynomial Addition.

ALGORITHM:

- Create a method to add the coefficients of two polynomials.
- Create a method to print the polynomial equation.
- Calculate the length of two polynomial equation.
- Print the polynomials by calling the corresponding function.
- Call the function to add the coefficients of two polynomials.
- Print the final polynomial.
- Exit.

RESOURCE:

Python 3.7.3

PROGRAM:

```
def add(A, B, m, n):
   size = max(m, n)
   sum = [0 for i in
range(size)]
   for i in range(0, m, 1):
      sum[i] = A[i]
   for i in range(n):
      sum[i] += B[i]
   return sum

# A utility function to print a
polynomial
def printPoly(poly, n):
    for i in range(n):
        print(poly[i], end="")
        if i != 0:
            print("x^", i, end="")
        if i != n - 1:
            print(" + ", end="")

# The following array
represents polynomial 5 +
10x^2 + 6x^3
A = [5, 10, 6]
# The following array
represents polynomial
1+2x+ 4x^2
B = [1, 2, 4]

m = len(A)
n = len(B)

print("First polynomial is")
printPoly(A, m)
print("\n")

print("Second polynomial
is")
printPoly(B, n)
print("\n")

sum = add(A, B, m, n)
size = max(m, n)

print("Sum Polynomial is")
printPoly(sum, size)

print("\nProgram Exit")
```

OUTPUT:

```
                    First polynomial is
                    5 + 10x^ 1 + 6x^ 2
      Second polynomial is
      1 + 2x^ 1 + 4x^ 2
      sum Polynomial is 6
      + 12x^ 1 + 10x^ 2
                     Program Exit
```

RESULT:

Thus the python program to perform polynomial addition was written and executed successfully.

Ex.No:06. b)	**POSTFIX EXPRESSION EVALUATION**
Date:	

AIM:

To write a python program to evaluate the postfix expression.

ALGORITHM:

- Create a class to create a list that stores the postfix expression.
- Create a method to add the symbols of the expression into the list.
- Create a method to pop the symbols from the stack when there is an operator.
- Create a method to perform the arithmetic operation.
- Call the appropriate functions to evaluate postfix expression and print the result.
- Exit.

RESOURCE:

Python 3.7.3

PROGRAM

```
class evaluate_postfix:
    def __init__(self):
        self.items = []
        self.size = -1

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)
        self.size += 1

    def pop(self):
        if self.isEmpty():
            return 0
        else:
            self.size -= 1
            return self.items.pop()

    def seek(self):
        if self.isEmpty():
            return False
        else:
            return self.items[self.size]

    def evalute(self, expr):
        for i in expr:
            if i in '0123456789':
                self.push(i)
            else:
                op1 = self.pop()
                op2 = self.pop()
                result = self.cal(op2, op1, i)
                self.push(result)
        return self.pop()

    def cal(self, op2, op1, i):
        if i == '*':
            return int(op2) * int(op1)
        elif i == '/':
            return int(op2) / int(op1)
        elif i == '+':
            return int(op2) + int(op1)
        elif i == '-':
            return int(op2) - int(op1)

s = evaluate_postfix()
expr = input('Enter the postfix expression: ')
value = s.evalute(expr)
print('The result of postfix expression', expr, 'is', value)
```

OUTPUT:

```
Enter the postfix expression 724*+2- The
result of postfix expression 724*+2- is 13
```

RESULT:

Thus the python program to evaluate the postfix expression was written and executed successfully.

Ex.No:07 a)	IMPLEMENTATION OF TREE TRAVERSAL
Date:	

AIM:

To write a python program to search a number in list using linear search.

ALGORITHM:

- Start the program
- Read the size of list
- Read the elements of List
- Read the element to find in a list
- Call the function linearsearch() with list name and element to find in the list Stop

Linearsearch:

1. set pos to -1
2. for i in range 0 to len(a) check if a[i] is equal to the element to find then set pos as i and print element is present and goto step 4
3. if pos is equal to -1 then print element is not present
4. return

RESOURCE:

Python 3.7.3

PROGRAM:

```
def linearsearch(a, f):
   pos = -1
   for i in range(0, len(a)):
      if a[i] == f:
            pos = i
            print("Element is present in position", pos + 1)
            break
    if pos == -1:
        print("Element is not present in the list")
    return

a = []
print("Enter the size of the list:")
n = int(input())

print("Enter the elements of the list:")
for i in range(0, n):
    x = int(input())
    a.append(x)

print("Enter element to find in the list:")
find = int(input())

linearsearch(a, find)
```

OUTPUT:

```
Enter the Size of List
5
Enter the elements of List
12
35
25
57
46
Enter Element to find in the List
30
Element is Present.
```

RESULT:

Thus the python program to search a number in list using linear search was written and executed successfully.

Ex.No:07 b)	BINARY SEARCH
Date:	

AIM:

To write a python program to search a number in list using binary search.

ALGORITHM:

- Start the program
- Read the size of list
- Read the elements of List a in ascending order
- Read the element to find in a list
- Call the function binarysearch() with list name and element to find in the list Stop

_Binarysearch:

1. Set pos to -1, first to 0 and last to len(list)
2. while first is less than or equal to last then goto step 3 else goto step
3. set (first+last)//2 to mid
4. Compare if a[mid] is equal to the element to find then set pos as mid and print the position and goto step 6 else goto step 5
5. If element to find is less than a[mid] then set last as mid-1 else set first as mid+1
6. if pos is equal to -1 then print element is not present
7. return

RESOURCE:

Python 3.7.3

PROGRAM: def
binarysearch(a,f):

```
pos=-1 first=0 last=len(a)
while(first<=last and first<len(a)):
```

```
mid=(first+last)//2 if(a[mid]==f): pos=mid
print("Element is Present in position",pos+1)
break else: if(f<a[mid]): last=mid-1 else:
first=mid+1 if(pos==-1): print("Element is not
present in the list") return a=[] print("Enter the Size
of List") n=int(input()) print("Enter the elements of
List in ascending order") for i in range(0,n):
x=int(input()) a.append(x) print("Enter Element to
find in the List") find=int(input())
binarysearch(a,find)
```

OUTPUT:

```
Enter the Size of List
5
Enter the elements of List in ascending order
12
14
15
17
25
Enter Element to find in the List
25
 Element is Present in position 5
```

RESULT:

Thus the python program to search a number in list using binary search was written and executed successfully.

Ex.No:07 c)	**SELECTION SORT**
Date:	

AIM:

To write a python program to sort the numbers using selection sort.

ALGORITHM:

- Start the program.
- Read the size of list.
- Read the elements of list .
- Call the function selectionsort(a).
- Print the list a.

selectionsort(a)

1. If i in range 0 to length(a) then continue else goto step 5.
2. set min as i.
3. If k in range i+1 to length(a) then continue else goto step 1.
4. Compare if a[k] is less than a[min] then set min as k and call swap(a,min,i) else goto step 3.
5. return.

swap(a,x,y)

1. Store a[x] to tmp.
2. Store a[y] to a[x].
3. Store tmp to a[y].
4. return.

RESOURCE:

Python 3.7.3

Program:

```
def selectionsort(a):
    for i in range(len(a)):
        min_index = i
        for k in range(i + 1, len(a)):
            if a[k] < a[min_index]:
                min_index = k
        swap(a, min_index, i)

def swap(a, x, y):
    temp = a[x]
    a[x] = a[y]
    a[y] = temp

# Input section
a = []
print("Enter Size of List")
n = int(input())
print("Enter Elements of List")
for i in range(n):
    x = int(input())
    a.append(x)

selectionsort(a)
print("After Sorting, the list is:", a)
```

OUTPUT:

```
Enter Size of List
4
Enter Elements of List
23
18
63
35
After Sorting the list is
 [18, 23, 35, 63]
```

RESULT:

Thus the python program to sort the numbers using selection sort was written and executed successfully

Ex.No: 7d)	INSERTION SORT
Date:	

AIM:

To write a python program to sort the numbers using insertion sort.

ALGORITHM:

- Start the program
- Read the size of list
- Read the elements of list
- Call the function insertionsort(a) Print the list a

<u>Insertionsort(a)</u>

1. If i is in range 0 to length(a) then continue else goto step 6
2. set current as a[i] and pos as i
3. while pos is greater then 0 and also a[pos-1] is greater than current the continue else goto step 5
4. Set a[pos] as a[pos-1] and pos as pos-1
6. Set a[pos] as current
7. return

RESOURCE:

Python 3.7.3

```
def insertionsort(a):
   for i in range(1, len(a)):
        current = a[i]
        pos = i
        while pos > 0 and a[pos - 1] > current:
            a[pos] = a[pos - 1]
            pos -= 1
        a[pos] = current

# Input section
a = []
print("Enter Size of List:")
n = int(input())
print("Enter Elements of List:")
for i in range(n):
    x = int(input())
    a.append(x)

insertionsort(a)
print("After Sorting, the list is:", a)
```

OUTPUT:

```
Enter Size of List
6
Enter Elements of List
41
26
32
11
45
68
After Sorting the list is
 [11, 26, 32, 41, 45, 68]
```

RESULT:

Thus the python program to sort the numbers using insertion sort was written and executed successfully.

Ex.No: 8 a)	**IMPLEMENTATION OF HASH TABLE**
Date:	

AIM:

To write a python program to implement hash table.

ALGORITHM:

- Create a list to store the key and values.
- Create a method to find the hash key
- Create a method to insert the key and values in hash table.
- Create a method to delete the value of the given key.
- Call the corresponding method to insert and delete the key, value pairs in the hash table.

RESOURCE:

Python 3.7.3

```
hash_table = [[] for _ in range(10)]

def hashing_func(key):
    return key % len(hash_table)

def insert(hash_table, key, value):
    hash_key = hashing_func(key)
    hash_table[hash_key].append((key, value))

def delete(hash_table, key):
    index = hashing_func(key)
    bucket = hash_table[index]
    for pair in bucket:
        if pair[0] == key:
            bucket.remove(pair)
            break

insert(hash_table, 10, 'Nepal')
print(hash_table)

insert(hash_table, 25, 'USA')
```

```
print(hash_table)

delete(hash_table, 25)
print(hash_table)
```

OUTPUT:

```
[(10, 'Nepal'), [], [], [], [], [], [], [], [], []]
[(10, 'Nepal'), [], [], [], [], (25, 'USA'), [], [], [],
[]] [(10, 'Nepal'), [], [], [], [], [], [], [], [], []]
```

RESULT:

Thus the python program to implement hash table was written and executed successfully.

Ex.No: 8 b)	IMPLEMENTATION OF HASHING
Date:	

AIM:

To write a python program to implement hashing

ALGORITHM:

- Create a list to store the key and values.
- Create a method to add the values into hash table.
- In this method, generate hashkey using the built-in function hash().
- Then if the hashkey exists append the value of the key to the bucket of hashkey.
- Create a method to delete the value of the given key.
- Create a method to search whether the given key is exist or not.
- Call the corresponding method to insert, search and delete the key, value pairs in the hash table

RESOURCE:

Python 3.7.3

PROGRAM:

```
hashTable = [[] for _ in range(10)]

def insertValue(hashTable, key, value):
    hashKey = hash(key) % len(hashTable)
    key_exists = False
    bucket = hashTable[hashKey]
    for i, kv in enumerate(bucket):
        k, v = kv
        if key == k:
            key_exists = True
            break
    if key_exists:
        bucket[i] = (key, value)
    else:
        bucket.append((key, value))

def deleteValue(hashTable, key):
    hashKey = hash(key) % len(hashTable)
    key_exists = False
    bucket = hashTable[hashKey]
    for i, kv in enumerate(bucket):
        k, v = kv
        if key == k:
            key_exists = True
            break
    if key_exists:
        del bucket[i]
        print('Key {} deleted'.format(key))
    else:
        print('Key {} not found'.format(key))

def searchTable(hashTable, key):
    hashKey = hash(key) % len(hashTable)
    bucket = hashTable[hashKey]
    for kv in bucket:
        k, v = kv
        if key == k:
            return v

insertValue(hashTable, 25, 'USA')
insertValue(hashTable, 10, 'INDIA')
insertValue(hashTable, 20, 'NEPAL')
print(hashTable)
```

```
s = searchTable(hashTable, 10)
if s:
    print("Key is found and its value is", s)
else:
    print("Key is not found")

deleteValue(hashTable, 20)
print(hashTable)
```

OUTPUT:

```
[[(10, 'INDIA'), (20, 'NEPAL')], [], [], [], [], [(25, 'USA')], [], [], [], []]
        Key is found and its value is INDIA
        Key 20 deleted
        [[(10, 'INDIA')], [], [], [], [], [(25, 'USA')], [], [], [], []]
```

RESULT:

Thus the python program to implement hashing was written and executed successfully

Ex.No:09	IMPLEMENTATION OF TREE TRAVERSAL
Date:	

AIM:

To write a python program to implement tree traversal.

ALGORITHM:

- Create a class Node to store the nodes of a tree.
- Create a method to print the preorder of tree.
- Create a method to print the inorder of tree.
- Create a method to print the postorder of tree.
- Create the nodes of tree by creating object for the class Node.
- Call the traversal functions to display the node values.

RESOURCE:

Python 3.7.3

PROGRAM:

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

def Inorder(root):
    if root:
        Inorder(root.left)
        print(root.val, end=" ")
        Inorder(root.right)
```

```
def Postorder(root):
    if root:
        Postorder(root.left)
        Postorder(root.right)
        print(root.val, end=" ")

def Preorder(root):
    if root:
        print(root.val, end=" ")
        Preorder(root.left)
        Preorder(root.right)

# Driver code
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)

print("Preorder traversal of binary tree is")
Preorder(root)

print("\nInorder traversal of binary tree is")
Inorder(root)

print("\nPostorder traversal of binary tree is")
Postorder(root)
```

OUTPUT:

```
Preorder traversal of binary tree is
1 2 4 5 3 6 7
Inorder traversal of binary tree is
4 2 5 1 6 3 7
Postorder traversal of binary tree is
4 5 2 6 7 3 1
```

RESULT:

Thus the python program to implement tree traversal was written and executed successfully.

Ex.No:10	**IMPLEMENTATION OF BINARY SEARCH TREE**
Date:	

AIM:

To write a python program to implement binary search tree.

ALGORITHM:

- Create a class Node to store the nodes of a tree.
- Create a method to print the inorder of tree.
- Create a method to insert the nodes by checking the values of the nodes.
- Create a method to find the minimum value of the node in the tree.
- Create a method to delete a node without changing the property of binary search tree.
- Call the appropriate methods to insert and delete the nodes in the tree.
- Call the function to print the nodes of the tree in inorder.

RESOURCE:

Python 3.7.3

PROGRAM:

```
# Binary Search Tree operations in Python

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

def inorder(root):
    if root is not None:
        inorder(root.left)
        print(str(root.key), end=' ')
        inorder(root.right)

def insert(node, key):
    if node is None:
        return Node(key)
    if key < node.key:
        node.left = insert(node.left, key)
    else:
        node.right = insert(node.right, key)
    return node

def minValueNode(node):
    current = node
    while(current.left is not None):
        current = current.left
    return current

def deleteNode(root, key):
    if root is None:
        return root
    if key < root.key:
        root.left = deleteNode(root.left, key)
    elif key > root.key:
        root.right = deleteNode(root.right, key)
    else:
        if root.left is None:
            temp = root.right
            root = None
            return temp
        elif root.right is None:
            temp = root.left
            root = None
            return temp
        temp = minValueNode(root.right)
        root.key = temp.key
        root.right = deleteNode(root.right, temp.key)
    return root

# Driver Code
root = None
root = insert(root, 8)
root = insert(root, 3)
root = insert(root, 1)
root = insert(root, 6)
root = insert(root, 7)
root = insert(root, 10)
root = insert(root, 14)
root = insert(root, 4)

print("Inorder traversal: ", end=' ')
inorder(root)

print("\nDelete 10")
root = deleteNode(root, 10)

print("Inorder traversal: ", end=' ')
inorder(root)
```

OUTPUT:

```
Preorder traversal of binary tree is
1 2 4 5 3 6 7
Inorder traversal of binary tree is
4 2 5 1 6 3 7
Postorder traversal of binary tree is
4 5 2 6 7 3 1
```

RESULT:

Thus the python program to implement binary search tree was written and executed successfully.

Ex.No:11	IMPLEMENTATION OF MIN-HEAP
Date:	

AIM:

To write a python program to implement min-heap

ALGORITHM:

- Create a class min-heap to define the size of the heap.
- Create a method to return the position of the parent of current node.
- Create a method to return the position of the left child of current node.
- Create a method to return the position of the right child of current node.
- Create a method to check whether the past node is leaf node or not.
- Create a method to swap the nodes of the heap.
- Create a method to rearrange the nodes that satisfy the property of min heap.
- Create the methods to insert and delete the nodes in the heap.
- Call the appropriate function to insert the nodes in the heap.
- Call the appropriate function to delete the nodes in the heap.
- Finally print the nodes present in the heap.

RESOURCE:

Python 3.7.3

PROGRAM:

```
import sys

class MinHeap:
    def __init__(self, maxsize):
        self.maxsize = maxsize
        self.size = 0
        self.Heap = [0] * (self.maxsize + 1)
        self.Heap[0] = -1 * sys.maxsize
        self.FRONT = 1

    def parent(self, pos):
        return pos // 2

    def leftChild(self, pos):
        return 2 * pos

    def rightChild(self, pos):
        return (2 * pos) + 1

    def isLeaf(self, pos):
        return pos * 2 > self.size

    def swap(self, fpos, spos):
        self.Heap[fpos], self.Heap[spos] = self.Heap[spos], self.Heap[fpos]

    def minHeapify(self, pos):
        if not self.isLeaf(pos):
            if (self.Heap[pos] > self.Heap[self.leftChild(pos)] or
                self.Heap[pos] > self.Heap[self.rightChild(pos)]):
                if self.Heap[self.leftChild(pos)] < self.Heap[self.rightChild(pos)]:
                    self.swap(pos, self.leftChild(pos))
                    self.minHeapify(self.leftChild(pos))
                else:
                    self.swap(pos, self.rightChild(pos))
                    self.minHeapify(self.rightChild(pos))

    def insert(self, element):
        if self.size >= self.maxsize:
            return
        self.size += 1
        self.Heap[self.size] = element
        current = self.size
        while self.Heap[current] < self.Heap[self.parent(current)]:
            self.swap(current, self.parent(current))
            current = self.parent(current)

    def Print(self):
        for i in range(1, (self.size // 2) + 1):
            print(" PARENT : " + str(self.Heap[i]) +
                  " LEFT CHILD : " + str(self.Heap[2 * i]) +
                  " RIGHT CHILD : " + str(self.Heap[2 * i + 1]))

    def minHeap(self):
        for pos in range(self.size // 2, 0, -1):
            self.minHeapify(pos)

    def remove(self):
        popped = self.Heap[self.FRONT]
        self.Heap[self.FRONT] = self.Heap[self.size]
        self.size -= 1
        self.minHeapify(self.FRONT)
        return popped

# Driver Code
if __name__ == "__main__":
    print('The minHeap is ')
    minHeap = MinHeap(15)
    minHeap.insert(5)
    minHeap.insert(3)
    minHeap.insert(17)
    minHeap.insert(10)
    minHeap.insert(84)
    minHeap.insert(19)
    minHeap.insert(6)
    minHeap.insert(22)
    minHeap.insert(9)
    minHeap.minHeap()
    minHeap.Print()
    print("The Min val is " + str(minHeap.remove()))
```

OUTPUT: Preorder traversal of binary tree is

```
2 4 5 3 6 7
 Inorder traversal of binary tree is
4 2 5 1 6 3 7
 Postorder traversal of binary tree is
 4 5 2 6 7 3 1
```

RESULT:

Thus the python program to implement min-heap was written and executed successfully.

Ex.No:12 a)	IMPLEMENTATION OF GRAPH TRAVERSAL - BFS
Date:	

AIM:

To write a python program to implement breadth first search traversal algorithm in graph.

ALGORITHM:

- Import the module collection to perform queue operations.
- Create a method to perform breadth first search.
- In the driver code create a dictionary for the nodes present in a graph Call the breadth first search function to traverse all the nodes in the graph.
- Exit.

RESOURCE:

Python 3.7.3

Program:

```
import collections

def bfs(graph, root):
    visited, queue = set(), collections.deque([root])
    visited.add(root)

    while queue:
        vertex = queue.popleft()
        print(str(vertex) + " ", end="")

        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)

if __name__ == '__main__':
    graph = {
        0: [1, 2],
        1: [2],
        2: [3],
        3: [1, 2]
    }
    print("Following is Breadth First Traversal: ")
    bfs(graph, 0)
```

OUTPUT:

```
Following is Breadth First Traversal:
0 1 2 3
```

RESULT:

Thus the python program to implement breadth first search traversal algorithm in graph was written and executed successfully.

Ex.No:12 b)	**IMPLEMENTATION OF GRAPH TRAVERSAL - DFS**
Date:	

AIM:

To write a python program to implement depth first search traversal algorithm in graph.

ALGORITHM:

- Import the module collection to perform queue operations.
- Create a method to perform depth first search.
- In the driver code create a dictionary for the nodes present in a graph Call the depth first search function to traverse all the nodes in the graph.
- Exit.

RESOURCE:

Python 3.7.3

PROGRAM:

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)
    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited

graph = {
    '0': set(['1', '2']),
    '1': set(['0', '3', '4']),
    '2': set(['0']),
    '3': set(['1']),
    '4': set(['2', '3'])
}

print("Following is Depth First Traversal: ")
dfs(graph, '0')
```

OUTPUT:

```
Following is Depth First Traversal:

0 1 2 3
```

RESULT:

Thus the python program to implement depth first search traversal algorithm in graph was written and executed successfully.

Ex.No:13	**DIJIKSTRA'S SHORTEST PATH ALGORITHM**
Date:	

AIM:

To write a python program to implement Dijikstra's shortest path algorithm in graph.

ALGORITHM:

- Create a class graph to initialize the vertices.
- Create a method to print the distance from source vertex to destination vertex.
- Create a method to find the minimum distance from one vertex to another.
- Create a method to implement dijikstra'a algorithm that finds the shortest path from the start node to the next node.
- Exit.

RESOURCE:

Python 3.7.3

PROGRAM:

```
class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printSolution(self, dist):
        print("Vertex \t Distance from Source")
        for node in range(self.V):
```

```
        print(node, "\t\t", dist[node])

    def minDistance(self, dist, sptSet):
        min = 1e7
        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v
        return min_index

    def dijkstra(self, src):
        dist = [1e7] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):
            u = self.minDistance(dist, sptSet)
            sptSet[u] = True
            for v in range(self.V):
                if (self.graph[u][v] > 0 and
                    sptSet[v] == False and
                    dist[v] > dist[u] + self.graph[u][v]):
                    dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)

# Driver program
g = Graph(9)
g.graph = [
    [0, 4, 0, 0, 0, 0, 0, 8, 0],
    [4, 0, 8, 0, 0, 0, 0, 11, 0],
    [0, 8, 0, 7, 0, 4, 0, 0, 2],
    [0, 0, 7, 0, 9, 14, 0, 0, 0],
    [0, 0, 0, 9, 0, 10, 0, 0, 0],
    [0, 0, 4, 14, 10, 0, 2, 0, 0],
    [0, 0, 0, 0, 0, 2, 0, 1, 6],
    [8, 11, 0, 0, 0, 0, 1, 0, 7],
    [0, 0, 2, 0, 0, 0, 6, 7, 0]
]

g.dijkstra(0)
```

OUTPUT:

```
Vertex   Distance from Source
0                  0
1                  4
2                  12
3                  19
4                  21
5                  11
6                  9
7                  8
8                  14
```

RESULT:

Thus the python program to implement Dijikstra's shortest path algorithm in graph was written and executed successfully

.

Ex.No:14	MIMNIMUM SPANNING TREE – KRUSKAL'S ALGORITHM
Date:	

AIM:

To write a python program to implement Kruskal's algorithm of minimum spanning tree in graph.

ALGORITHM:

- Create a class graph to initialize the vertices.
- Create a method to add edge from one vertex to another.
- Create a method to find the parent of a vertex and return it.
- Create a method for kruskal's algorithm that finds the shortest path from the start node to the next node.
- Exit.

RESOURCE:

Python 3.7.3

PROGRAM:

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def apply_union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)

        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1

    def kruskal_algo(self):
        result = []
        i, e = 0, 0

        self.graph = sorted(self.graph, key=lambda item: item[2])
        parent = []
        rank = []

        for node in range(self.V):
            parent.append(node)
            rank.append(0)

        while e < self.V - 1:
            u, v, w = self.graph[i]
            i += 1
            x = self.find(parent, u)
            y = self.find(parent, v)

            if x != y:
                e += 1
                result.append([u, v, w])
                self.apply_union(parent, rank, x, y)

        for u, v, weight in result:
            print("%d - %d: %d" % (u, v, weight))

# Driver code
g = Graph(6)
g.add_edge(0, 1, 4)
g.add_edge(0, 2, 4)
g.add_edge(1, 2, 2)
g.add_edge(1, 0, 4)
g.add_edge(2, 0, 4)
g.add_edge(2, 1, 2)
g.add_edge(2, 3, 3)
g.add_edge(2, 5, 2)
g.add_edge(2, 4, 4)
g.add_edge(3, 2, 3)
g.add_edge(3, 4, 3)
g.add_edge(4, 2, 4)
g.add_edge(4, 3, 3)
g.add_edge(5, 2, 2)
g.add_edge(5, 4, 3)

g.kruskal_algo()
```

OUTPUT:

```
    1 - 2: 2
    2 - 5: 2
    2 - 3: 3
```

```
3 - 4: 3
0 - 1: 4
```

RESULT:

Thus the python program to implement Kruskal’s algorithm of minimum spanning tree in graph was written and executed successfully.

VIVA QUESTIONS:

1. What is a Data Structure?
2. Types of Data Structures and give examples?
3. What is a Singly Linked List?
4. What is Doubly Linked List?
5. What is a Stack?
6. What are the Applications of Stack?
7. What is a Queue?
8. What are the applications of Queues?
9. What is a Circular Queue?
10. What is Dequeue? (Double Ended Queue)
11. What is a Tree?
12. What is a Binary Tree?
13. What is Tree Traversal? List different Tree Traversal Techniques?
14. What is a Binary Search Tree? Give one example?
15. What is the best, average and worst case time complexity of insertion, deletion and Search operations in a Binary Search Tree?
16. What is an AVL Search Tree?
17. What is AVL Balance Condition?
18. What is a Full (perfect) Binary Tree?
19. What is a Complete Binary Tree?
20. What is a Min Heap?
21. What is a Max Heap?
22. What is a Binary Heap?
23. What is a Graph?
24. Name various Graph Representation Techniques?
25. What is a Spanning Tree?
26. Name the methods to construct a Minimum cost Spanning Tree?

27. What is Linear Search?
28. What is Binary Search method?
29. What is Sorting?
30. Study the procedure of following Sorting techniques?
31. What is Hashing?
32. List different Hashing methods.
33. What is Collision?
34. List different collision resolution strategies?
35. List one application of Hashing.