AIM:

Write a Python Program to implement Linear Search.

ALGORITHM:

Step 1: Start the program

Step 2: Start searching from the leftmost element of given arr[] and one by one compare element x with each element of arr[]

Step 3: If x matches with any of the element, return the index value.

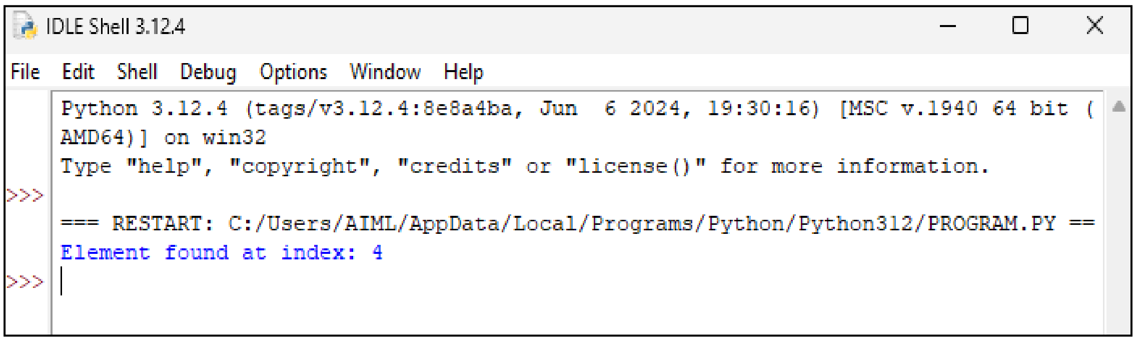
Step 4: If x doesn’t match with any of elements in arr[] , return -1 or element not found.

Step 5: Stop the program

PROGRAM:

```
def linear_Search(list1, n, key):
    # Searching list1 sequentially
    for i in range(0, n):
        if (list1[i] == key):
            return i
    return -1
list1 = [1, 3, 5, 4, 7, 9]
key = 7
n = len(list1)
res = linear_Search(list1, n, key)
if res == -1:
    print("Element not found")
else:
    print("Element found at index:", res)
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PROGRAM.PY ==
Element found at index: 4
>>>
```

RESULT:

Thus the Python program to perform linear search was executed successfully.

AIM:

Write a Python Program to implement Binary Search.

ALGORITHM:

Step 1: Find middle element of the array.

Step 2: Compare the value of the middle element with the target value.

Step 3: If they match, it is returned.

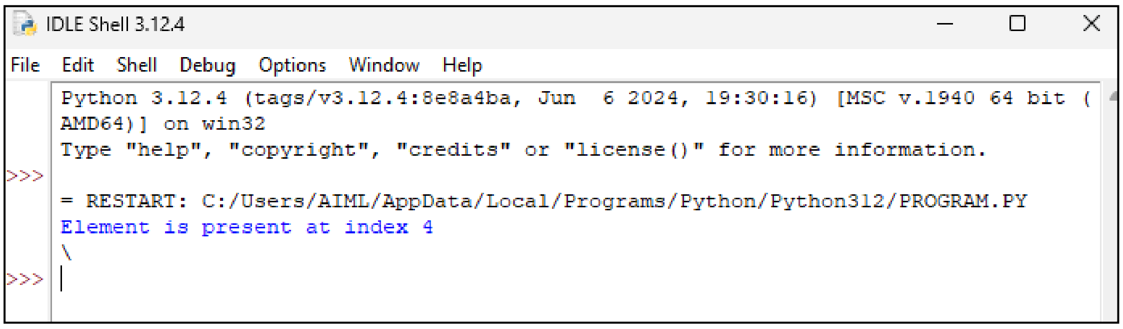
Step 4: If the value is less or greater than the target, the search continues in the lower or upper half of the array accordingly.

Step 5: The same procedure as in step 2-4 continues, but with a smaller part of the array. This continues until the target element is found or until there are no elements left.

PROGRAM:

```
# Iterative Binary Search Function
# It returns index of n in given list1 if present, else returns -1
def binary_search(list1, n):
    low = 0
    high = len(list1) - 1

    while low <= high:
        # Get the middle index
        mid = (high + low) // 2
        # Check if n is present at mid
        if list1[mid] < n:
            low = mid + 1
        elif list1[mid] > n:
            high = mid - 1
        else:
            return mid  # n is found

    return -1  # Element was not present in the list

# Initial list1
list1 = [12, 24, 32, 39, 45, 50, 54]
n = 45

# Function call
result = binary_search(list1, n)

if result != -1:
    print("Element is present at index", str(result))
else:
    print("Element is not present in list1")
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PROGRAM.PY
Element is present at index 4
\
>>> 
```

RESULT:

Thus the Python program to perform binary search was executed successfully.

AIM:

Write a Python program to implement Fibonacci Series using recursive algorithm.

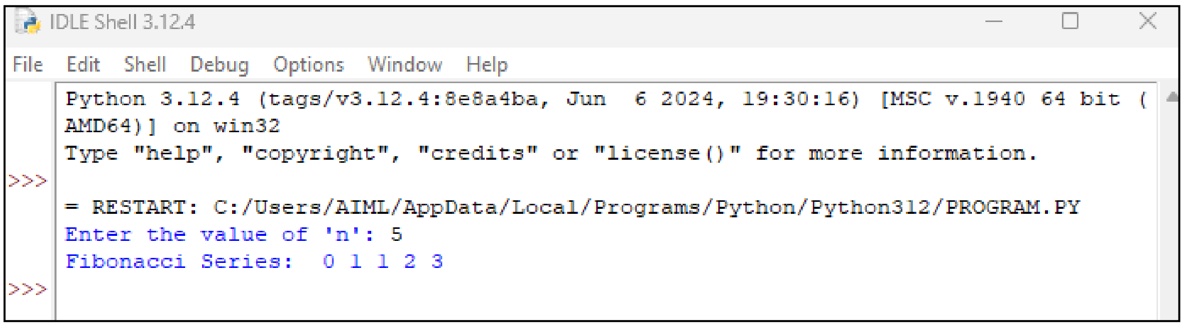
ALGORITHM :

```
Step 1: Start the Program
Step 2: Call the Procedure Recursive_Fibonacci(n)int f0, f1 f0 := 0
            f1 := 1
Step 3: If(n <= 1):
Step 4: Return n
Step 5: Return Recursive_Fibonacci(n-1) + Recursive_Fibonacci(n-2)
Step 6: Stop the program
```

PROGRAM:

```
# Python program to generate Fibonacci series until 'n' value
n = int(input("Enter the value of 'n': "))
a = 0
b = 1
count = 1
print("Fibonacci Series: ", end = " ")

while count <= n:
    print(a, end=" ")  # Print the current Fibonacci number
    sum = a + b       # Calculate the next Fibonacci number
    a = b             # Move to the next number in the series
    b = sum           # Update b to the next Fibonacci number
    count += 1       # Increment the counter
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PROGRAM.PY
Enter the value of 'n': 5
Fibonacci Series:  0 1 1 2 3
>>>
```

RESULT:

Thus, the above program for implementing Fibonacci series using recursion was successfully completed.

AIM:

Write a Python program to implement Fibonacci Search.

ALGORITHM:

Step 1: Initialize Variables:

- Set offset to -1.
- Initialize Fibonacci numbers: Fm2 = 0, Fm1 = 1.
- Calculate the next Fibonacci number Fm = Fm2 + Fm1.

Step 2: Find the Smallest Fibonacci Number Greater than or Equal to n:

- While Fm < n:
    - Update Fibonacci numbers:
        - Fm2 = Fm1
        - Fm1 = Fm
        - Fm = Fm2 + Fm1

Step 3: Perform the Fibonacci Search:

- While Fm > 1:
    - Calculate index i = min(offset + Fm2, n - 1).
    - If arr[i] < key:
            - Fm = Fm1
            - Fm1 = Fm2
            - Fm2 = Fm - Fm1
        - Update offset = i.
    - Else if arr[i] > key:
            - Fm = Fm2
            - Fm1 = Fm1 - Fm2
            - Fm2 = Fm - Fm1.
    - Else:
        - Return index i (key found).

Step 4: Check Last Element:

- If Fm1 == 1 and arr[offset + 1] == key, return offset + 1.

Step 5: Return Not Found:

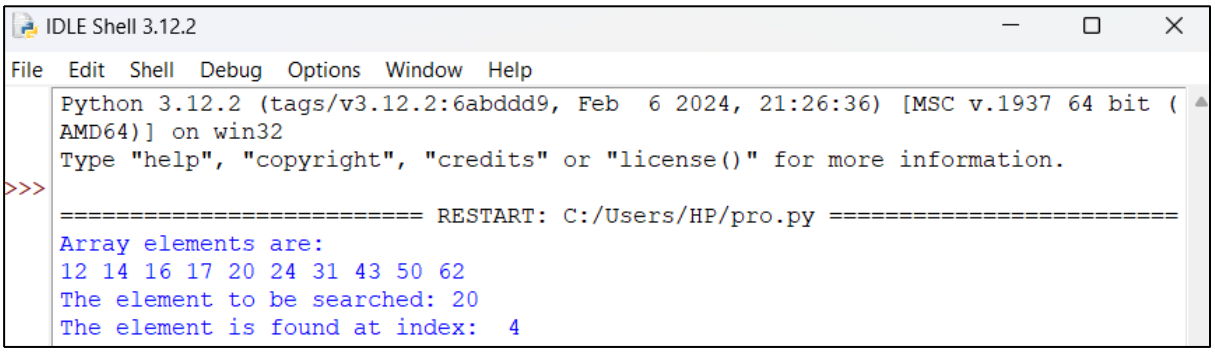
- If the key is not found, return -1.

PROGRAM:

```
def fibonacci_search(arr, n, key):
   offset = -1
   Fm2 = 0
   Fm1 = 1
   Fm = Fm2 + Fm1
   while (Fm < n):
      Fm2 = Fm1
      Fm1 = Fm
      Fm = Fm2 + Fm1
   while (Fm > 1):
      i = min(offset + Fm2, n - 1)
      if (arr[i] < key):
         Fm = Fm1
         Fm1 = Fm2
         Fm2 = Fm - Fm1
         offset = i
      elif (arr[i] > key):
         Fm = Fm2
         Fm1 = Fm1 - Fm2
         Fm2 = Fm - Fm1
      else:
         return i
   if (Fm1 == 1 and arr[offset + 1] == key):
      return offset + 1
   return -1
arr = [12, 14, 16, 17, 20, 24, 31, 43, 50, 62]
print("Array elements are: ")
for j in range(len(arr)):
   print(arr[j], end = " ")
n = len(arr);
key = 20
```

```
print("\nThe element to be searched:", key)
index = fibonacci_search(arr, n, key)
if(index >= 0):
   print("The element is found at index: ", (index))
else:
   print("Unsuccessful Search")
```

OUTPUT:



```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========================= RESTART: C:/Users/HP/pro.py ========================
Array elements are:
12 14 16 17 20 24 31 43 50 62
The element to be searched: 20
The element is found at index:  4
```

RESULT:

Thus, the above program for implementing Fibonacci search using recursion was successfully completed.

AIM:

Write a Python Program to implement Insertion Sort.

ALGORITHM:

Step 1: Start the program

Step 2: If it is the first element, then place it in the sorted sub-array.

Step 3: Pick the next element.

Step 4: Compare the picked element with all the elements in sorted sub-array.

Step 5: Shift all the elements in the sorted sub-array that are greater than the picked element to besorted.

Step 6: Insert the element at the desired place.

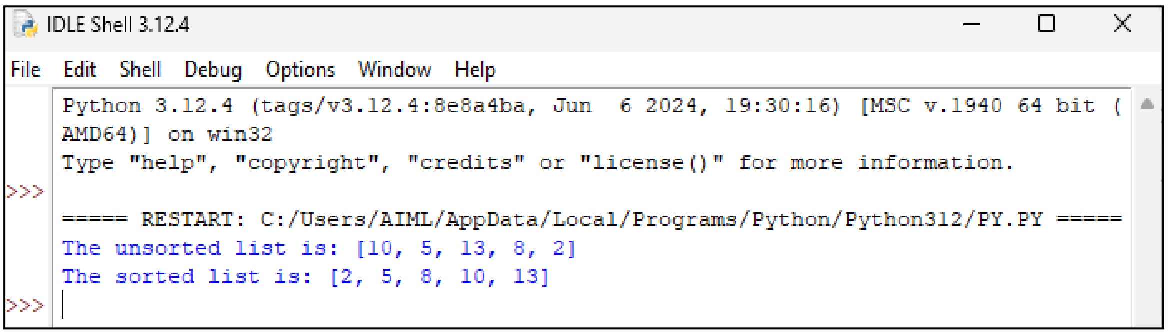
Step 7: Repeat the above steps until the array is completely sorted.

Step 8: Stop the program

PROGRAM:

```
def insertion_sort(list1):
    # Outer loop to traverse through 1 to len(list1)
    for i in range(1, len(list1)):
        value = list1[i]
        # Move elements of list1[0..i-1], that are greater than value,
        # to one position ahead of their current position
        j = i - 1
        while j >= 0 and value < list1[j]:
            list1[j + 1] = list1[j]
            j -= 1
        list1[j + 1] = value
    return list1

list1 = [10, 5, 13, 8, 2]
print("The unsorted list is:", list1)
print("The sorted list is:", insertion_sort(list1))
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY =====
The unsorted list is: [10, 5, 13, 8, 2]
The sorted list is: [2, 5, 8, 10, 13]
>>>
```

RESULT :

Thus the python program for insertion sort was executed successfully.

AIM:

To write Python program to implement selection sort technique.

ALGORITHM:

Step 1: Start

Step 2: Get the elements from the user

Step 3: Set MIN to location 0

Step 4: Search the minimum element in the list.

Step 5: Swap with value at location MIN

Step 6: Increment MIN to point to next element

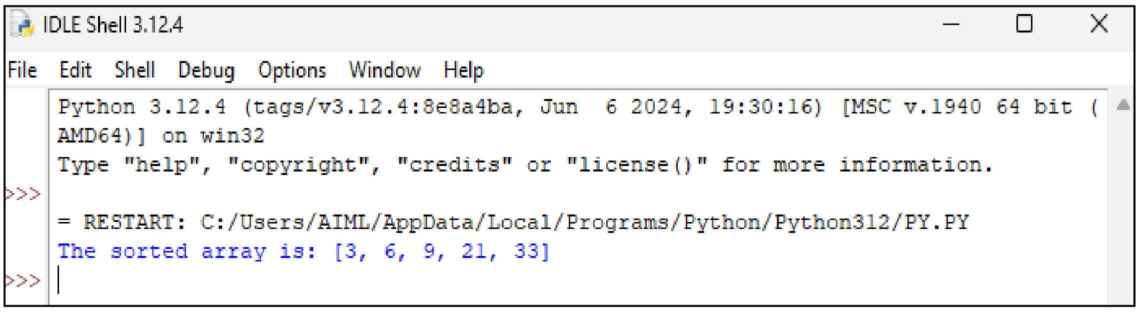
Step 7: Repeat until list is sorted.

Step 8: Stop

PROGRAM:

```
def selection_sort(array):
    length = len(array)
    for i in range(length - 1):
        minIndex = i
        for j in range(i + 1, length):
            if array[j] < array[minIndex]:
                minIndex = j
        # Swap the found minimum element with the first element
        array[i], array[minIndex] = array[minIndex], array[i]
    return array

array = [21, 6, 9, 33, 3]
print("The sorted array is:", selection_sort(array))
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
The sorted array is: [3, 6, 9, 21, 33]
>>> 
```

RESULT:

Thus the python program for selection sort was executed successfully.

AIM:

Write a Python Program to implement Quick Sort.

ALGORITHM:

Step 1: Start the program

Step 2: Choose the highest index value has pivot

Step 3: Take two variables to point left and right of the list excluding pivot

Step 4: left points to the low index

Step 5: right points to the high

Step 6: while value at left is less than pivot move right

Step 7: while value at right is greater than pivot move left

Step 8: if both step 6 and step 7 does not match swap left and right

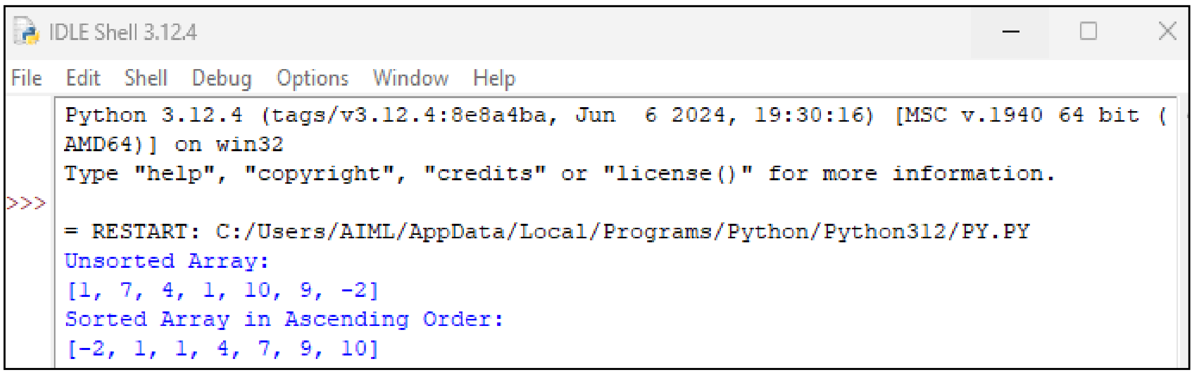
Step 9: if left ≥ right, the point where they met is new pivot

Step 10:Stop the program

PROGRAM:

```
# Function to find the partition position
def partition(array, low, high):
    # Choose the rightmost element as pivot
    pivot = array[high]
    # Pointer for the greater element
    i = low - 1
    # Traverse through all elements
    # Compare each element with pivot
    for j in range(low, high):
        if array[j] <= pivot:
            # If element smaller than pivot is found
            # Swap it with the greater element pointed by i
            i += 1
            # Swapping element at i with element at j
            array[i], array[j] = array[j], array[i]
    # Swap the pivot element with the greater element specified by i
    array[i + 1], array[high] = array[high], array[i + 1]
    # Return the position from where partition is done
    return i + 1

# Function to perform quicksort
def quickSort(array, low, high):
    if low < high:
        # Find pivot element such that
        # elements smaller than pivot are on the left
        # elements greater than pivot are on the right
        pi = partition(array, low, high)
        # Recursive call on the left of pivot
        quickSort(array, low, pi - 1)
        # Recursive call on the right of pivot
    quickSort(array, pi + 1, high)

data = [1, 7, 4, 1, 10, 9, -2]
print("Unsorted Array:")
print(data)
size = len(data)
quickSort(data, 0, size - 1)
print('Sorted Array in Ascending Order:')
print(data)
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
Unsorted Array:
[1, 7, 4, 1, 10, 9, -2]
Sorted Array in Ascending Order:
[-2, 1, 1, 4, 7, 9, 10]
```

RESULT :

Thus the python program for quick sort was executed successfully.

AIM:

To write Python program to implement heap sort.

ALGORITHM:

Step 1: Construct a Binary Tree with given list of Elements.

Step 2: Transform the Binary Tree into Min Heap.

Step 3: Delete the root element from Min Heap using Heapify method.

Step 4: Put the deleted element into the Sorted list.

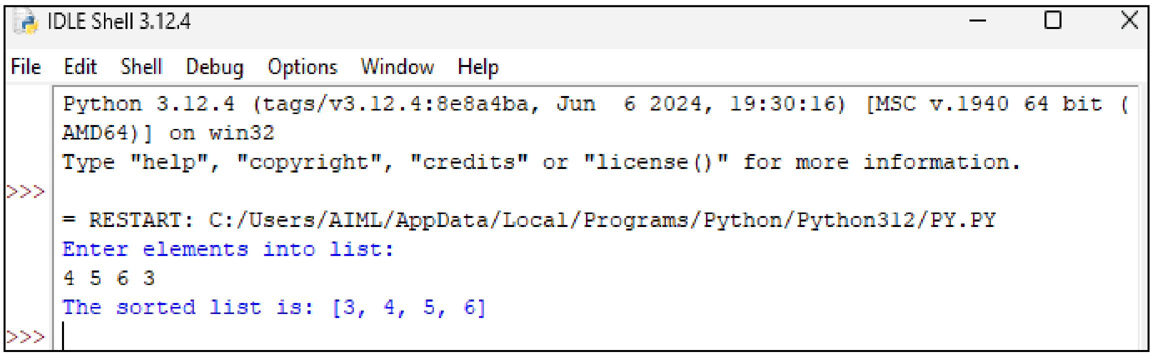
Step 5: Repeat the same until Min Heap becomes empty.

Step 6: Display the sorted list.

PROGRAM:

```
def s_sort(a):
    for i in range(len(a)):
        least = i
        for k in range(i + 1, len(a)):
            if a[k] < a[least]:
                least = k
        # Swap the found minimum element with the first element
        swap(a, least, i)

def swap(a, least, i):
    temp = a[least]
    a[least] = a[i]
    a[i] = temp

print("Enter elements into list:")
a = [int(x) for x in input().split()]
s_sort(a)
print("The sorted list is:", a)
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
Enter elements into list:
4 5 6 3
The sorted list is: [3, 4, 5, 6]
>>>
```

RESULT:

Thus the python program for heap sort was executed successfully.

AIM:

To write Python program to implement merge sort.

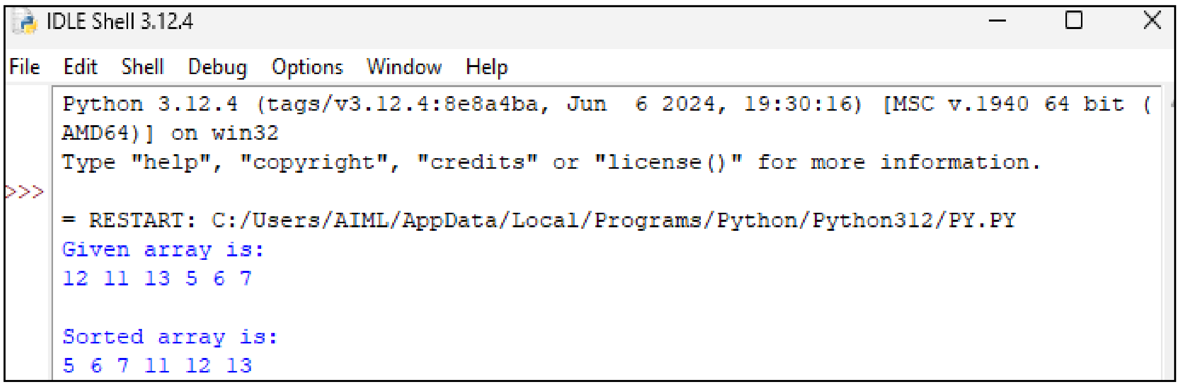
ALGORITHM:

```
Step 1: Start the program
Step 2: Declare array and left, right, mid variable
Step 3: Perform merge function.
        if left > rightreturn
        mid= (left+right)/2
        mergesort(array, left, mid)
        mergesort(array, mid+1, right)
        merge(array, left, mid, right)
Step 4: Stop the program
```

PROGRAM:

```
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    # Create temp arrays
    L = [0] * n1
    R = [0] * n2

    # Copy data to temp arrays L[] and R[]
    for i in range(n1):
        L[i] = arr[l + i]
    for j in range(n2):
        R[j] = arr[m + 1 + j]

    # Merge the temp arrays back into arr[l..r]
    i = 0  # Initial index of first subarray
    j = 0  # Initial index of second subarray
    k = l  # Initial index of merged subarray

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    # Copy the remaining elements of L[], if there are any
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1
    # Copy the remaining elements of R[], if there are any
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

# l is for left index and r is right index of the sub-array of arr to be sorted
def mergeSort(arr, l, r):
    if l < r:
        # Same as (l + r) // 2, but avoids overflow for large l and h
        m = l + (r - l) // 2
        # Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m + 1, r)
        merge(arr, l, m, r)

arr = [12, 11, 13, 5, 6, 7]
n = len(arr)
print("Given array is:")
for i in range(n):
    print("%d" % arr[i], end=" ")

mergeSort(arr, 0, n - 1)
print("\n\nSorted array is:")
for i in range(n):
    print("%d" % arr[i], end=" ")
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
Given array is:
12 11 13 5 6 7

Sorted array is:
5 6 7 11 12 13
```

RESULT:

Thus the python program for merge sort was executed successfully.

AIM:

Write a Python Program to implement Bubble Sort.

ALGORITHM:

Step 1: Start the program

Step 2: Get the total number of elements. Get the total number of items in the given list

Step 3: Determine the number of outer passes (n – 1) to be done. Its length is list minus one

Step 4: Perform inner passes (n – 1) times for outer pass 1. Get the first element value and compare it with the second value. If
the second value is less than the first value, then swap the positions

Step 5: Repeat step 3 passes until you reach the outer pass (n – 1). Get the next element in the list then repeat the process that was performed in step 3 until all the values have been placed in their correct ascending order.

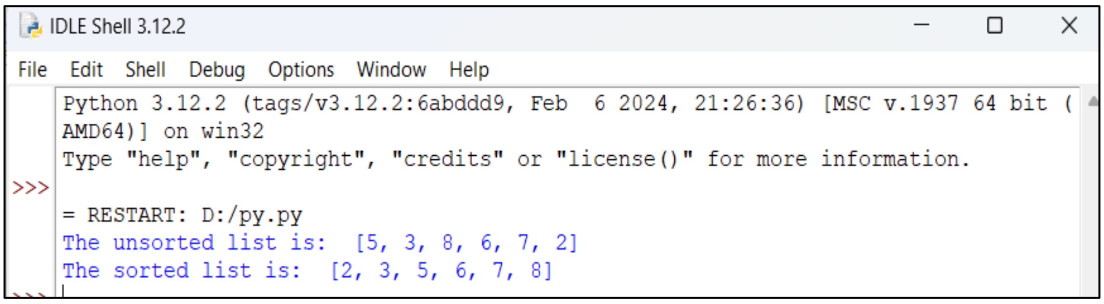
Step 6: Return the result when all passes have been done. Return the results of the sorted list

Step 7: Stop the program

PROGRAM:

```
def bubble_sort(list1):
    # Outer loop for traversing the entire list
    for i in range(len(list1) - 1):
        # Inner loop for comparing adjacent elements
        for j in range(len(list1) - 1 - i):
            if list1[j] > list1[j + 1]:
                # Swap if the element found is greater than the next element
                list1[j], list1[j + 1] = list1[j + 1], list1[j]
    return list1

list1 = [5, 3, 8, 6, 7, 2]
print("The unsorted list is: ", list1)  # Display the unsorted list
# Calling the bubble sort function
print("The sorted list is: ", bubble_sort(list1))
```

OUTPUT:



```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:/py.py
The unsorted list is:  [5, 3, 8, 6, 7, 2]
The sorted list is:  [2, 3, 5, 6, 7, 8]
>>>
```

RESULT:

Thus, the above program for implementing Bubble Sort was successfully completed.

AIM:

To write a python program to implement stack and its operation.

ALGORITHM:

1. Initialize Stack

Step 1: Create an empty list (or array) called stack.

2. Push Operation

Step 1: Define a function push(item).

Step 2: Add item to the end of stack.

Step 3: Print a confirmation message (optional).

3. Pop Operation

Step 1: Define a function pop().

Step 2: Check if stack is empty.

If empty: Return an error message or None.

Step 3: Remove the last item from stack.

Step 4: Return the removed item.

4. Peek Operation

Step 1: Define a function peek().

Step 2: Check if stack is empty.

If empty: Return an error message or None.

Step 3: Return the last item in stack without removing it.

5. Is Empty Operation

Step 1: Define a function is_empty().

Step 2: Return True if stack has no items, otherwise return False.

6. Size Operation

Step 1: Define a function size().

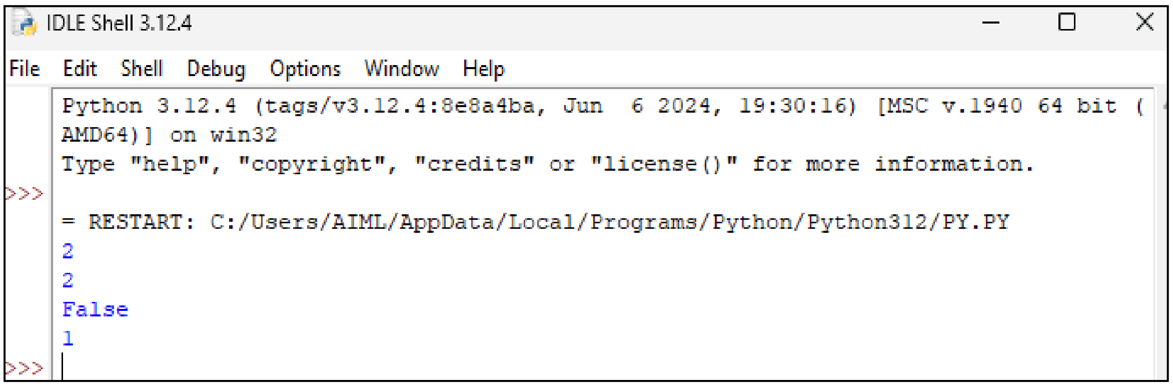
Step 2: Return the number of items in stack.

PROGRAM:

```
class Stack:
    def __init__(self):
        """Initialize an empty stack."""
        self.stack = []

    def push(self, item):
        """Add an item to the top of the stack."""
        self.stack.append(item)

    def pop(self):
        """Remove the item from the top of the stack and return it."""
        if not self.is_empty():
            return self.stack.pop()
        raise IndexError("pop from empty stack")

    def peek(self):
        """Return the item at the top of the stack without removing it."""
        if not self.is_empty():
            return self.stack[-1]
        raise IndexError("peek from empty stack")

    def is_empty(self):
        """Check if the stack is empty."""
        return len(self.stack) == 0

    def size(self):
        """Return the number of items in the stack."""
        return len(self.stack)
```

```
# Example usage
stack = Stack()
stack.push(1)
stack.push(2)
print(stack.peek())  # Output: 2
print(stack.pop())   # Output: 2
print(stack.is_empty())  # Output: False
print(stack.size())  # Output: 1
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
2
2
False
1
>>>
```

RESULT:

Thus, the above program has been executed successfully

AIM:

To write a Python program for evaluating arithmetic expression to postfix expression.

Evaluation rule of a Postfix Expression states:

1. While reading the expression from left to right, push the element in the stack if it is an operand.
2. Pop the two operands from the stack, if the element is an operator and then evaluate it.
3. Push back the result of the evaluation. Repeat it till the end of the expression.

ALGORITHM:

Step 1: Create a stack to store operands (or values).

Step 2: Scan the given expression and do following for every scanned element.

Step 3: If the element is a number, push it into the stack

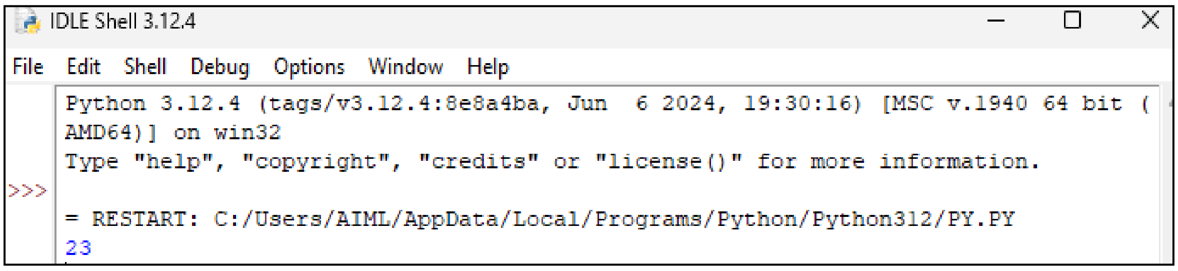
Step 4: If the element is a operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack

Step 5: When the expression is ended, the number in the stack is the final answer

PROGRAM:

```
def postfix_evaluation(s):
    s = s.split()
    stack = []

    for token in s:
        if token.isdigit():  # Check if the token is a number
            stack.append(int(token))  # Push the number onto the stack
        else:
            # Pop the two topmost numbers from the stack
            b = stack.pop()
            a = stack.pop()
            # Perform the appropriate operation
            if token == "+":
                stack.append(a + b)
            elif token == "-":
                stack.append(a - b)
            elif token == "*":
                stack.append(a * b)
            elif token == "/":
                if b != 0:  # Avoid division by zero
                    stack.append(a / b)
                else:
                    raise ValueError("Division by zero is not allowed.")

    return stack.pop()  # The final result is the last element in the stack

# Space-separated input for postfix evaluation
s = "10 2 8 * + 3 -"
val = postfix_evaluation(s)
print(val)
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
23
```

RESULT:

Thus the program for the arithmetic expression to postfix expression is evaluated successfully.

AIM:

Write a Python Program to implement Queue ADT.

ALGORITHM:

Step 1: Start the program

Step 2: Create a class Queue with instance variable items initialized to an empty list.

Step 3: Define methods enqueue, dequeue and is_empty inside the class Queue.

Step 4: The method enqueue appends data to items.

Step 5: The method dequeue dequeues the first element in items.

Step 6: The method is_empty returns True only if items is empty.

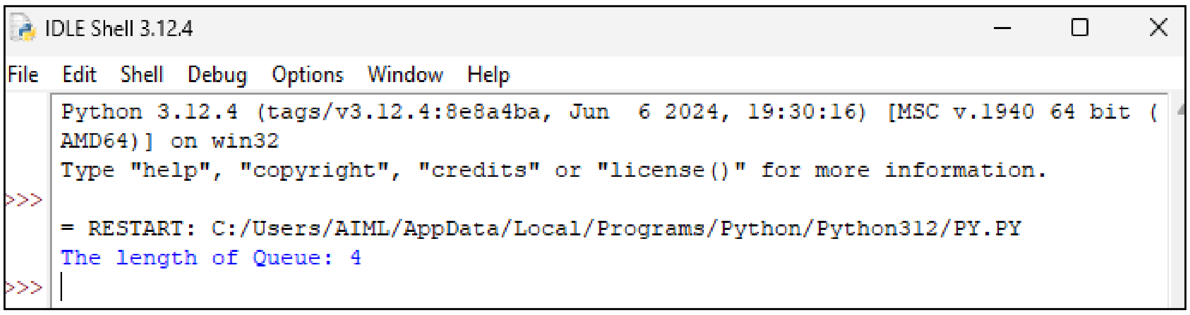
Step 7: Create an instance of Queue and present a menu to the user to perform operations on the queue.

Step 8: Stop the program

PROGRAM:

```
class Queue:
    def __init__(self):  # Fixed the constructor method name
        self.queue = list()

    def add_element(self, val):
        # Insert method to add element
        if val not in self.queue:
            self.queue.insert(0, val)  # Add element to the front of the queue
            return True
        return False

    def size(self):
        return len(self.queue)

# Create a Queue instance
TheQueue = Queue()
TheQueue.add_element("Apple")
TheQueue.add_element("Mango")
TheQueue.add_element("Guava")
TheQueue.add_element("Papaya")

# Print the length of the queue
print("The length of Queue:", TheQueue.size())
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
The length of Queue: 4
>>> 
```

RESULT:

Thus the program for implementing Queue ADT is executed successfully.

AIM:

Write a Python Program to implement Circular Queue ADT.

ALGORITHM:

Step 1: Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

Step 2: Declare all user defined functions used in circular queue implementation.

Step 3: Create a one dimensional array with above defined SIZE (int cQueue[SIZE])

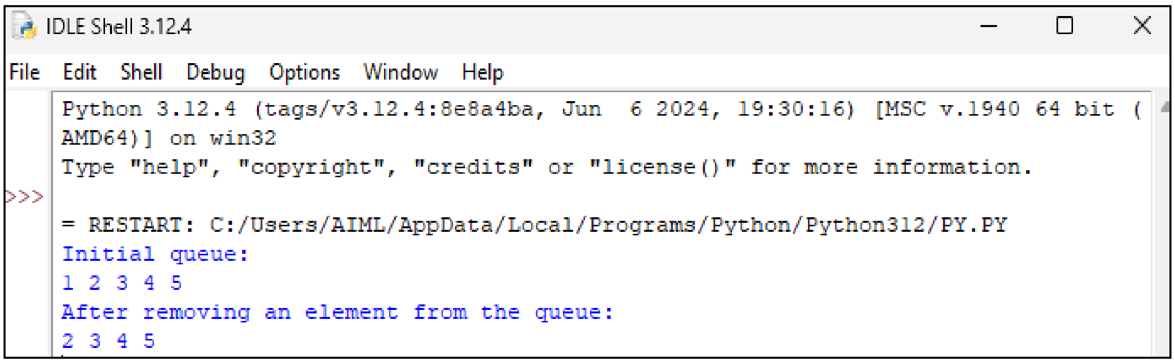
Step 4: Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1)

Step 5: Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on circular queue.

PROGRAM:

```
# Circular Queue implementation in Python
class MyCircularQueue:
    def __init__(self, k):
        self.k = k
        self.queue = [None] * k
        self.head = self.tail = -1

    # Insert an element into the circular queue
    def enqueue(self, data):
        if (self.tail + 1) % self.k == self.head:
            print("The circular queue is full\n")
        elif self.head == -1:
            self.head = 0
            self.tail = 0
            self.queue[self.tail] = data
        else:
            self.tail = (self.tail + 1) % self.k
            self.queue[self.tail] = data

    # Delete an element from the circular queue
    def dequeue(self):
        if self.head == -1:
            print("The circular queue is empty\n")
            return None
        elif self.head == self.tail:
            temp = self.queue[self.head]
            self.head = -1
            self.tail = -1
            return temp
        else:
            temp = self.queue[self.head]
            self.head = (self.head + 1) % self.k
            return temp

    def printCQueue(self):
        if self.head == -1:
            print("No element in the circular queue")
        elif self.tail >= self.head:
            for i in range(self.head, self.tail + 1):
                print(self.queue[i], end=" ")
            print()
        else:
            for i in range(self.head, self.k):
                print(self.queue[i], end=" ")
            for i in range(0, self.tail + 1):
                print(self.queue[i], end=" ")
            print()

# Your MyCircularQueue object will be instantiated and called as such:
obj = MyCircularQueue(5)
obj.enqueue(1)
obj.enqueue(2)
obj.enqueue(3)
obj.enqueue(4)
obj.enqueue(5)

print("Initial queue:")
obj.printCQueue()

obj.dequeue()
print("After removing an element from the queue:")
obj.printCQueue()
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
Initial queue:
1 2 3 4 5
After removing an element from the queue:
2 3 4 5
```

RESULT:

Thus the program for implementing Circular Queue ADT is executed successfully.

AIM:

Write a Python Program to implement Priority Queue ADT.

ALGORITHM:

Step 1: Start the program

Step 2: Create a class Queue with instance variable items initialized to an empty list.

Step 3: Define methods enqueue, dequeue and is_empty inside the class Queue.

Step 4: The method enqueue appends data to items.

Step 5: The method dequeue dequeues the first element in items.

Step 6: The method is_empty returns True only if items is empty.

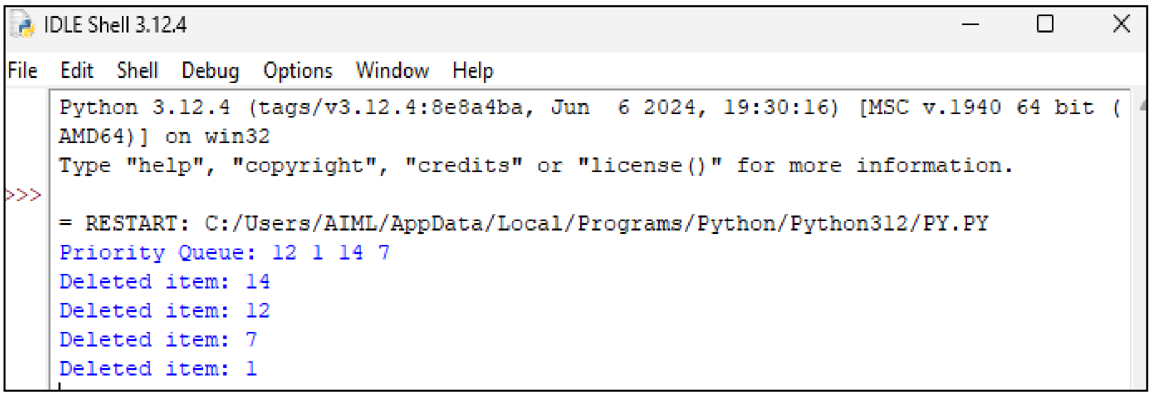
Step 7: Create an instance of Queue and present a menu to the user to perform operations on the queue.

PROGRAM:

```
# A simple implementation of Priority Queue using Queue.
class PriorityQueue(object):
    def __init__(self):  # Fixed the constructor method name
        self.queue = []

    def __str__(self):  # Fixed the string representation method
        return ' '.join([str(i) for i in self.queue])

    # Check if the queue is empty
    def isEmpty(self):
        return len(self.queue) == 0

    # Insert an element into the queue
    def insert(self, data):
        self.queue.append(data)

    # Pop an element based on priority
    def delete(self):
        try:
            max_val = 0
            for i in range(len(self.queue)):
                if self.queue[i] > self.queue[max_val]:
                    max_val = i
            item = self.queue[max_val]
            del self.queue[max_val]
            return item
        except IndexError:
            print("Queue is empty")
            exit()
```

```
if __name__ == '__main__':  # Fixed the main check
    myQueue = PriorityQueue()
    myQueue.insert(12)
    myQueue.insert(1)
    myQueue.insert(14)
    myQueue.insert(7)

    print("Priority Queue:", myQueue)

    while not myQueue.isEmpty():
        print("Deleted item:", myQueue.delete())
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
Priority Queue: 12 1 14 7
Deleted item: 14
Deleted item: 12
Deleted item: 7
Deleted item: 1
```

RESULT:

Thus the program for implementing Priority Queue ADT is executed successfully.

AIM:

Write a Python Program for Linked list Implementations of List.

ALGORITHM:

Step 1: Start the Program

Step 2: Declare the class

Step 3: Insert the elements in the list

Step 4: Display the elements present in the list

Step 5: Creating a header node

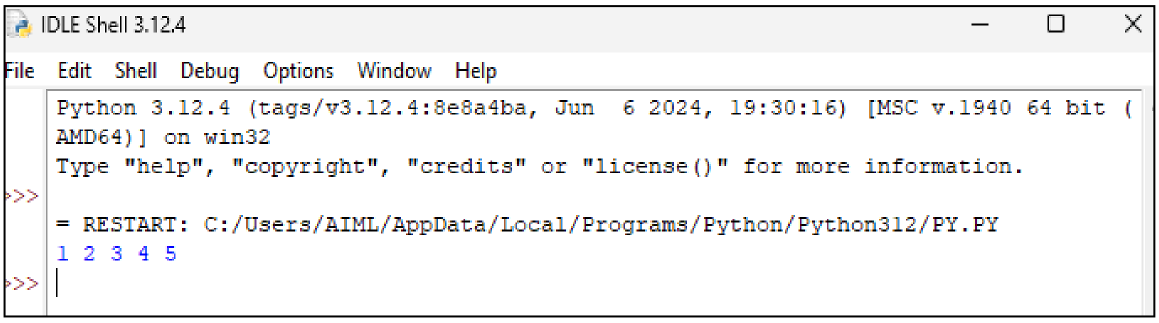
Step 6: Convert the given array to list

Step 7: Stop the program

PROGRAM:

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

# Function to insert a Node
def insert(root, item):
    temp = Node(item)  # Create a new node
    temp.next = root  # Point the new node to the current root
    return temp  # New node becomes the new root

def display(root):
    while root is not None:
        print(root.data, end=" ")
        root = root.next
    print()  # For better output formatting

def arrayToList(arr, n):
    root = None
    for i in range(n - 1, -1, -1):
        root = insert(root, arr[i])
    return root

if __name__ == '__main__':
    arr = [1, 2, 3, 4, 5]
    n = len(arr)
    root = arrayToList(arr, n)
    display(root)
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
1 2 3 4 5
>>>
```

RESULT:

Thus the program for linked list implementation is executed successfully.

AIM:

Write a Python Program for Doubly Linked list Implementations of List.

ALGORITHM:

Step 1: Define a Node class which represents a node in the list. It will have three properties: data, previous which will point to the previous node and next which will point to the next node.

Step 2: Define another class for creating a doubly linked list, and it has two nodes: head and tail. Initially, head and tail will point to null.

Step 3: addNode() will add node to the list:

Step 4: It first checks whether the head is null, then it will insert the node as the head.

Step 5: Both head and tail will point to a newly added node.

Step 6: Head's previous pointer will point to null and tail's next pointer will point to null.

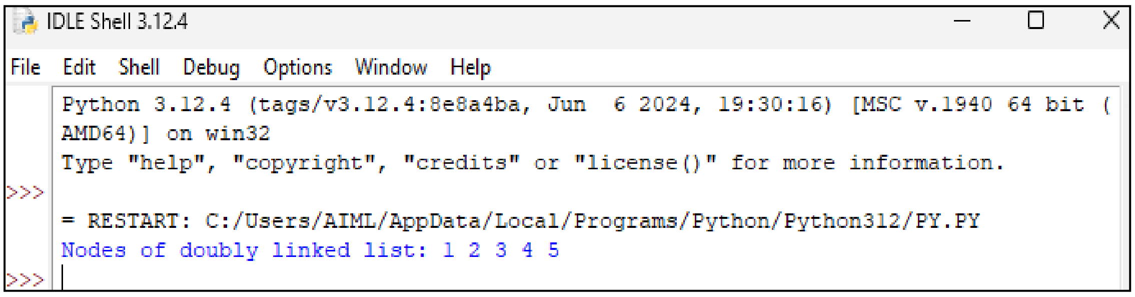
Step 7: If the head is not null, the new node will be inserted at the end of the list such that new node's previous pointer will point to tail.

Step 8: The new node will become the new tail. Tail's next pointer will point to null.

PROGRAM:

```
# Represent a node of doubly linked list
class Node:
    def __init__(self, data):
        self.data = data
        self.previous = None
        self.next = None

class DoublyLinkedList:
    # Represent the head and tail of the doubly linked list
    def __init__(self):
        self.head = None
        self.tail = None

    # addNode() will add a node to the list
    def addNode(self, data):
        # Create a new node
        newNode = Node(data)

        # If list is empty
        if self.head is None:
            # Both head and tail will point to newNode
            self.head = self.tail = newNode
            # head's previous will point to None
            self.head.previous = None
            # tail's next will point to None, as it is the last node of the list
            self.tail.next = None
        else:
            # newNode will be added after tail such that tail's next will point to newNode
            self.tail.next = newNode
            # newNode's previous will point to tail
            newNode.previous = self.tail
            # newNode will become new tail
            self.tail = newNode
            # As it is the last node, tail's next will point to None
            self.tail.next = None

    # display() will print out the nodes of the list
    def display(self):
        # Node current will point to head
        current = self.head
        if self.head is None:
            print("List is empty")
            return
        print("Nodes of doubly linked list: ", end="")
        while current is not None:
            # Prints each node by incrementing pointer
            print(current.data, end=" ")
            current = current.next
        print()  # For better output formatting

# Create a DoublyLinkedList instance and add nodes
dList = DoublyLinkedList()
dList.addNode(1)
dList.addNode(2)
dList.addNode(3)
dList.addNode(4)
dList.addNode(5)

# Displays the nodes present in the list
dList.display()
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
Nodes of doubly linked list: 1 2 3 4 5
>>>
```

RESULT:

Thus the program for Double linked list implementation is executed successfully.

AIM:

Write a Python Program for Circular Linked list Implementations of List.

ALGORITHM:

Step 1: Define a Node class which represents a node in the list. It has two properties data and next which will point to the next nod

Step 2: Define another class for creating the circular linked list, and it has two nodes: head and tail. It has two methods: add() an display() .

Step 3: add() will add the node to the list:

- o It first checks whether the head is null, then it will insert the node as the head.
- o Both head and tail will point to the newly added node.
- o If the head is not null, the new node will be the new tail, and the new tail will point to the head as it is a circular linked list.

Step 4: display() will show all the nodes present in the list.

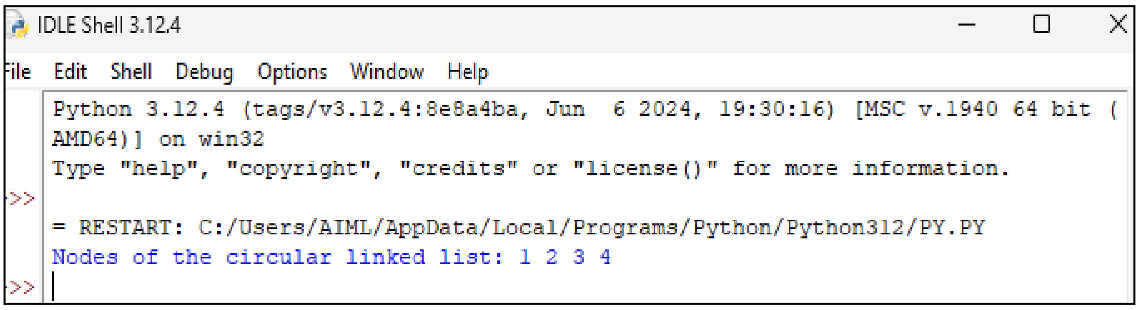
- o Define a new node 'current' that will point to the head.
- o Print current.data till current will points to head
- o Current will point to the next node in the list in each iteration

Step 5: Stop the program

PROGRAM:

```
# Represents the node of the list.
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class CreateList:
    # Declaring head and tail pointers as None.
    def __init__(self):
        self.head = None
        self.tail = None

    # This function will add the new node at the end of the list.
    def add(self, data):
        newNode = Node(data)
        # Checks if the list is empty.
        if self.head is None:
            # If the list is empty, both head and tail would point to the new node.
            self.head = newNode
            self.tail = newNode
            newNode.next = self.head  # Point to itself (circular)
        else:
            # tail will point to the new node.
            self.tail.next = newNode
            # New node will become the new tail.
            self.tail = newNode
            # Since it is a circular linked list, tail will point to head.
            self.tail.next = self.head

    # Displays all the nodes in the list.
    def display(self):
        current = self.head
        if self.head is None:
            print("List is empty")
            return
        else:
            print("Nodes of the circular linked list: ", end="")
            # Prints each node by incrementing the pointer.
            print(current.data, end=" ")
            while current.next != self.head:
                current = current.next
                print(current.data, end=" ")
            print()  # Newline for better output formatting

# Example usage:
if __name__ == '__main__':
    cl = CreateList()
    # Adds data to the list
    cl.add(1)
    cl.add(2)
    cl.add(3)
    cl.add(4)
    # Displays all the nodes present in the list
    cl.display()
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
Nodes of the circular linked list: 1 2 3 4
>>>
```

RESULT:

Thus the program for Circular linked list implementation is executed successfully.

AIM:

To write a python program to implement tree representation and traversal algorithm.

ALGORITHM:

Pre-order traversal

Step 1: Visit the root (we will print it when we visit to show the order of visiting)

Step 2: Traverse the left subtree in pre-order

Step 3: Traverse the right subtree in pre-order

In-order traversal

Visit the root node in between the left and right node (in)

Step 1: Traverse the left subtree in in-order

Step 2: Visit the root (we will print it when we visit to show the order of visiting)

Step 3: Traverse the right subtree in in-order

Post-order traversal

Visit the root node after (post) visiting the left and right subtree.

Step 1: Traverse the left subtree in in-order

Step 2: Traverse the right subtree in in-order

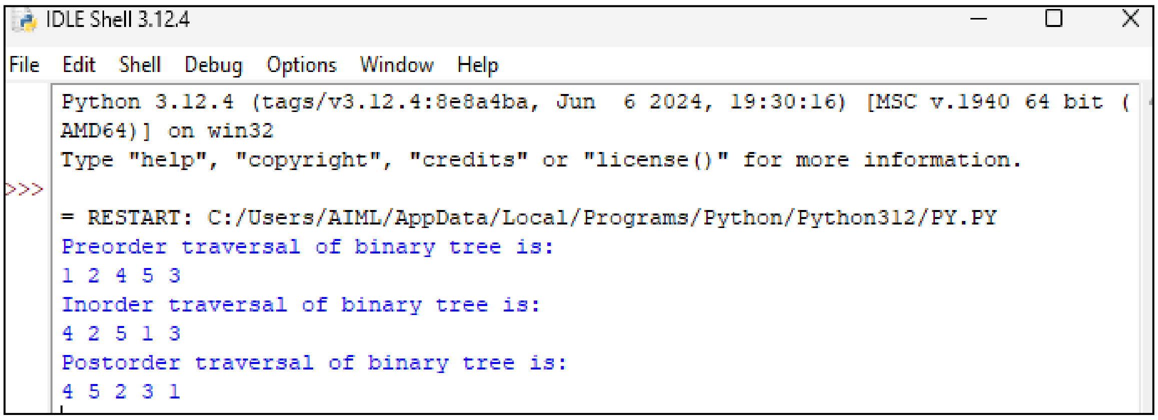
Step 3: Visit the root (we will print it when we visit to show the order of visiting)

PROGRAM:

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

def printInorder(root):
    if root:
        printInorder(root.left)
        print(root.val, end=" ")  # Updated to avoid newline
        printInorder(root.right)

def printPostorder(root):
    if root:
        printPostorder(root.left)
        printPostorder(root.right)
        print(root.val, end=" ")  # Updated to avoid newline

def printPreorder(root):
    if root:
        print(root.val, end=" ")  # Updated to avoid newline
        printPreorder(root.left)
        printPreorder(root.right)

# Construct the binary tree
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
```

```
# Print traversals
print("Preorder traversal of binary tree is:")
printPreorder(root)
print("\nInorder traversal of binary tree is:")
printInorder(root)
print("\nPostorder traversal of binary tree is:")
printPostorder(root)
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
Preorder traversal of binary tree is:
1 2 4 5 3
Inorder traversal of binary tree is:
4 2 5 1 3
Postorder traversal of binary tree is:
4 5 2 3 1
```

RESULT:

Thus, the above program for tree representation and traversal algorithm was successfully completed.

AIM:

To write a python program to implement Breadth-first search (BFS).

ALGORITHM:

Step 1: Start the program.

Step 2: Pick any node, visit the adjacent unvisited vertex, mark it as visited, display it, and insertit in a queue.

Step 3: If there are no remaining adjacent vertices left, remove the first vertex from the queue.

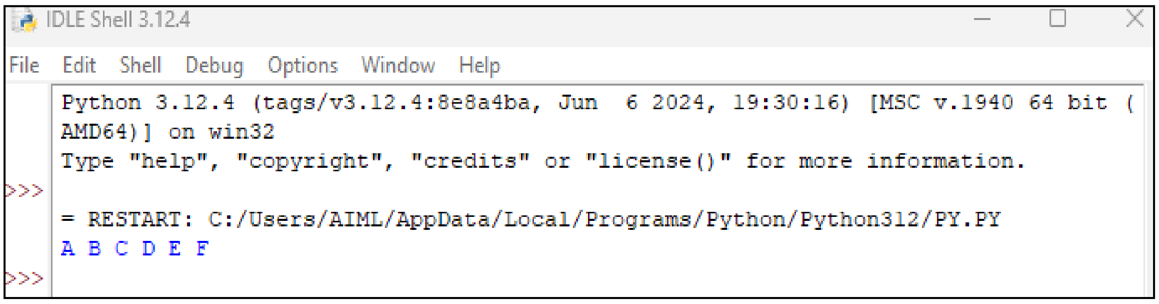
Step 4: Repeat step 2 and step 3 until the queue is empty or the desired node is found.

Step 5: Stop the program.

PROGRAM:

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

visited = []  # List to keep track of visited nodes
queue = []    # Initialize a queue

def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)

    while queue:
        s = queue.pop(0)
        print(s, end=" ")

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

# Start BFS from node 'A'
bfs(visited, graph, 'A')
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
A B C D E F
>>>
```

RESULT:

Thus, the above program for Breadth-first search was successfully completed

AIM:

To write a python program to implement Depth first search (DFS).

ALGORITHM:

Step 1: Start the program.

Step 2: Pick any node. If it is unvisited, mark it as visited and recur on all its adjacent nodes.

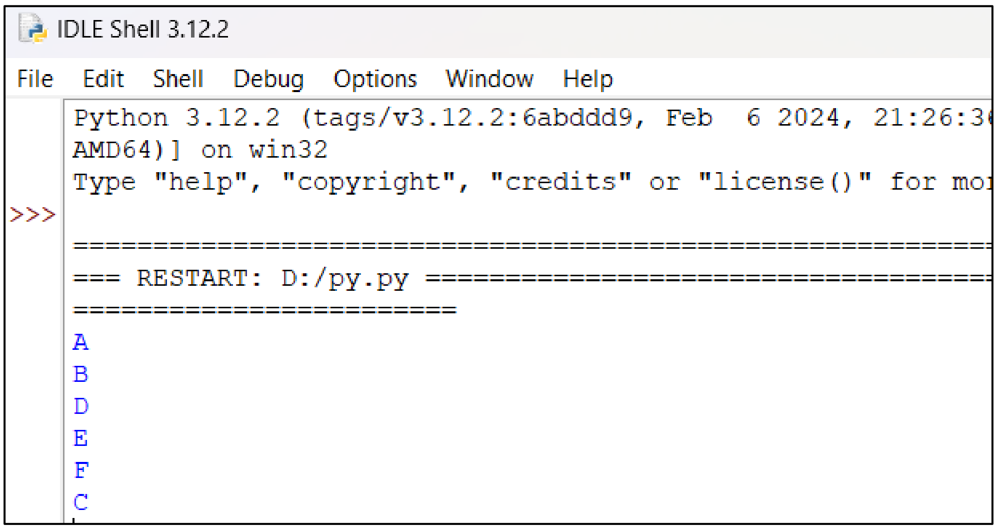
Step 3: Repeat until all the nodes are visited, or the node to be searched is found.

Step 4: Stop the program.

PROGRAM:

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

visited = set()  # Set to keep track of visited nodes

def dfs(visited, graph, node):
    if node not in visited:
        print(node)
        visited.add(node)

        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
dfs(visited, graph, 'A')
```

OUTPUT:



IDLE Shell 3.12.2

File Edit Shell Debug Options Window Help

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:3
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for mo
>>>
===========================================================
=== RESTART: D:/py.py =====================================
========================
A
B
D
E
F
C
```

RESULT:

Thus, the above program for Depth-first search was successfully completed.

AIM:

To write a python program to implement of single source shortest path algorithm.

ALGORITHM:

Step 1: Start the program.

Step 2: Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.

Step 3: Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

Step 4: While sptSet doesn’t include all vertices:

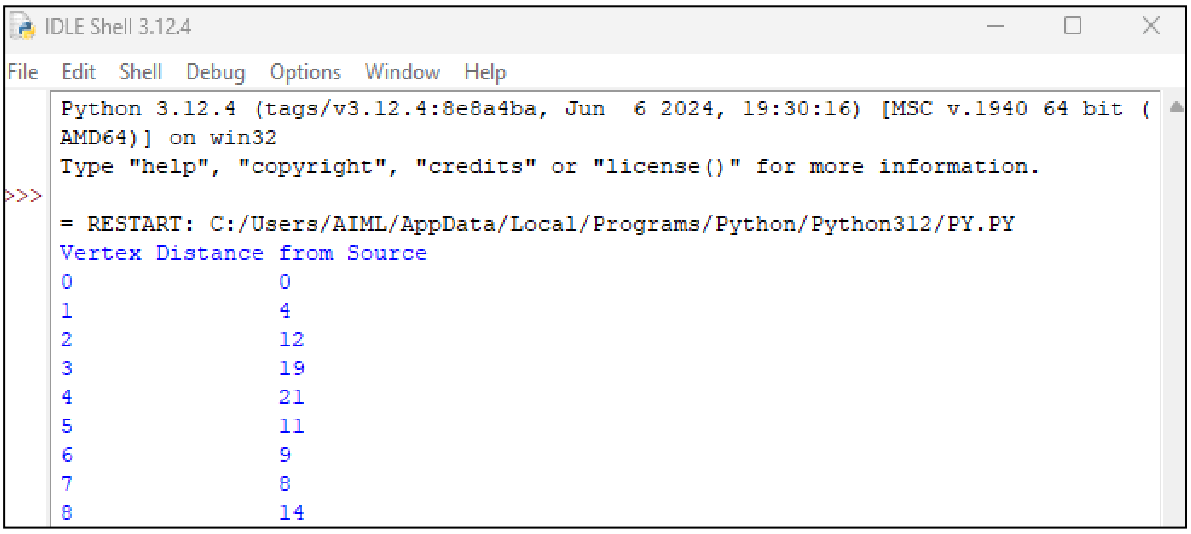
- Pick a vertex u which is not there in sptSet and has minimum distance value.
- Include u to sptSet.
- Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if the sum of a distance value ofu (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Step 5: Stop the program

PROGRAM:

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices  # Number of vertices
        self.graph = [[0] * vertices for _ in range(vertices)]  # Adjacency matrix

    def minDistance(self, dist, sptSet):
        # Initialize minimum distance for the next node
        min_val = float('inf')
        min_index = -1

        for v in range(self.V):
            if dist[v] < min_val and not sptSet[v]:
                min_val = dist[v]
                min_index = v

        return min_index

    def dijkstra(self, src):
        dist = [float('inf')] * self.V
        dist[src] = 0  # Distance from source to itself is always 0
        sptSet = [False] * self.V  # Shortest path tree set

        for cout in range(self.V):
            # Pick the minimum distance vertex from the set of vertices not yet processed
            u = self.minDistance(dist, sptSet)

            # Put the minimum distance vertex in the shortest path tree
            sptSet[u] = True

            # Update dist value of the adjacent vertices of the picked vertex
            for v in range(self.V):
                if (self.graph[u][v] > 0 and not sptSet[v] and
                        dist[v] > dist[u] + self.graph[u][v]):
                    dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)

    def printSolution(self, dist):
        print("Vertex Distance from Source")
        for i in range(self.V):
            print(f"{i}\t\t{dist[i]}")

# 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:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
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 above program for implementation of single source shortest path algorithm was successfully Completed.

AIM:

To write a python program to implement hash table and its operation.

ALGORITHM:

1. Initialize Hash Table

Step 1: Define a fixed-size array (or list) called table.

Step 2: Initialize all elements of table to None.

2. Hash Function

Step 1: Define a function hash(key).

Step 2: Compute the hash value for key (e.g., using modulo operation):

Step 3: Return the computed index.

3. Insert Operation

Step 1: Define a function insert(key, value).

Step 2: Compute the index using the hash function: index = hash(key).

Step 3: If table[index] is None:

Step 4: If there is a collision (i.e., table[index] is not None):

4. Retrieve Operation

Step 1: Define a function retrieve(key).

Step 2: Compute the index using the hash function: index = hash(key).

Step 3: Check table[index]:

If table[index] is None: Return None (key not found).

If there are multiple pairs (in case of chaining):

Traverse the linked list to find the (key, value) pair.

Step 4: Return the associated value.

5. Delete Operation

Step 1: Define a function delete(key).

Step 2: Compute the index using the hash function: index = hash(key).

Step 3: Check table[index]:

If table[index] is None: Return an error message (key not found).

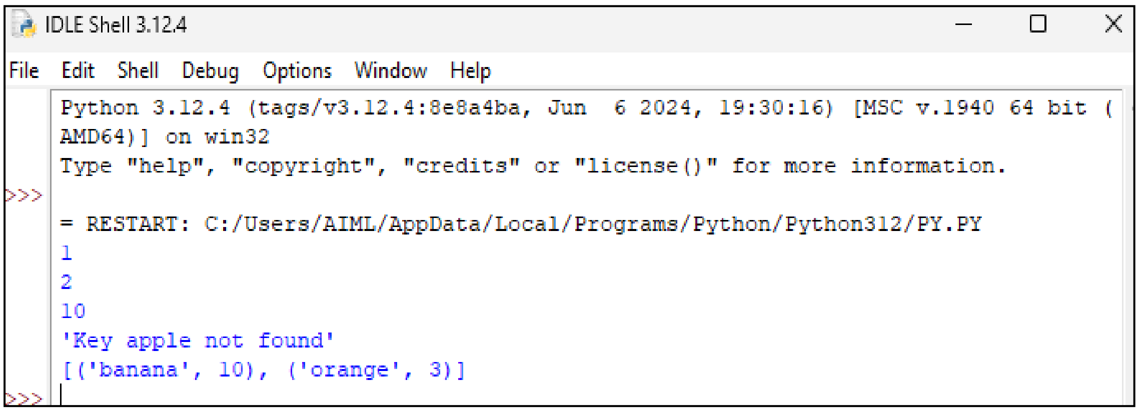
Traverse the linked list to find and remove the (key, value) pair.

Step 4: Set table[index] to None or remove the pair as appropriate.

PROGRAM:

```
class HashTable:
    def __init__(self, size=10):
        """Initialize the hash table with a given size."""
        self.size = size
        self.table = [[] for _ in range(size)]  # Create a list of empty lists (buckets)

    def _hash(self, key):
        """Compute the hash index for a given key."""
        return hash(key) % self.size

    def insert(self, key, value):
        """Insert a key-value pair into the hash table."""
        index = self._hash(key)
        bucket = self.table[index]

        # Check if key exists and update it
        for idx, kv in enumerate(bucket):
            if kv[0] == key:
                bucket[idx] = (key, value)
                return

        # Key does not exist, insert new key-value pair
        bucket.append((key, value))

    def delete(self, key):
        """Delete a key-value pair from the hash table."""
        index = self._hash(key)
        bucket = self.table[index]

        for idx, kv in enumerate(bucket):
            if kv[0] == key:
                del bucket[idx]
                return

        raise KeyError(f"Key {key} not found")

    def search(self, key):
        """Search for a value by key in the hash table."""
        index = self._hash(key)
        bucket = self.table[index]

        for kv in bucket:
            if kv[0] == key:
                return kv[1]

        raise KeyError(f"Key {key} not found")

    def update(self, key, value):
        """Update the value for a given key."""
        index = self._hash(key)
        bucket = self.table[index]

        for idx, kv in enumerate(bucket):
            if kv[0] == key:
                bucket[idx] = (key, value)
                return

        raise KeyError(f"Key {key} not found")

    def __repr__(self):
        """Return a string representation of the hash table."""
        items = []
        for bucket in self.table:
            items.extend(bucket)
        return str(items)

# Example usage
ht = HashTable()

# Insert key-value pairs
ht.insert('apple', 1)
ht.insert('banana', 2)
ht.insert('orange', 3)

# Search for keys
print(ht.search('apple'))  # Output: 1
print(ht.search('banana')) # Output: 2

# Update a key-value pair
ht.update('banana', 10)
print(ht.search('banana')) # Output: 10

# Delete a key-value pair
ht.delete('apple')
try:
    print(ht.search('apple'))  # Raises KeyError
except KeyError as e:
    print(e)

# Display the hash table
print(ht)  # Output: [('banana', 10), ('orange', 3)]
```

OUTPUT:



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/AIML/AppData/Local/Programs/Python/Python312/PY.PY
1
2
10
'Key apple not found'
[('banana', 10), ('orange', 3)]
>>>
```

RESULT:

Thus, the above program for implementation of hash table and its operation was successfully Completed.