32.Write a Program to find both the maximum and minimum values in the array. Implement using any programming language of your choice. Execute your code and provide the maximum and minimum values found.

Input : N= 8, a[] = {5,7,3,4,9,12,6,2}

Output : Min = 2, Max = 12

Test Cases :

Input : N= 9, a[] = {1,3,5,7,9,11,13,15,17}

Output : Min = 1, Max = 17

Test Cases :

Input : N= 10, a[] = {22,34,35,36,43,67, 12,13,15,17}

Output : Min 12, Max 67

Aim: Find the maximum & minimum values in array.

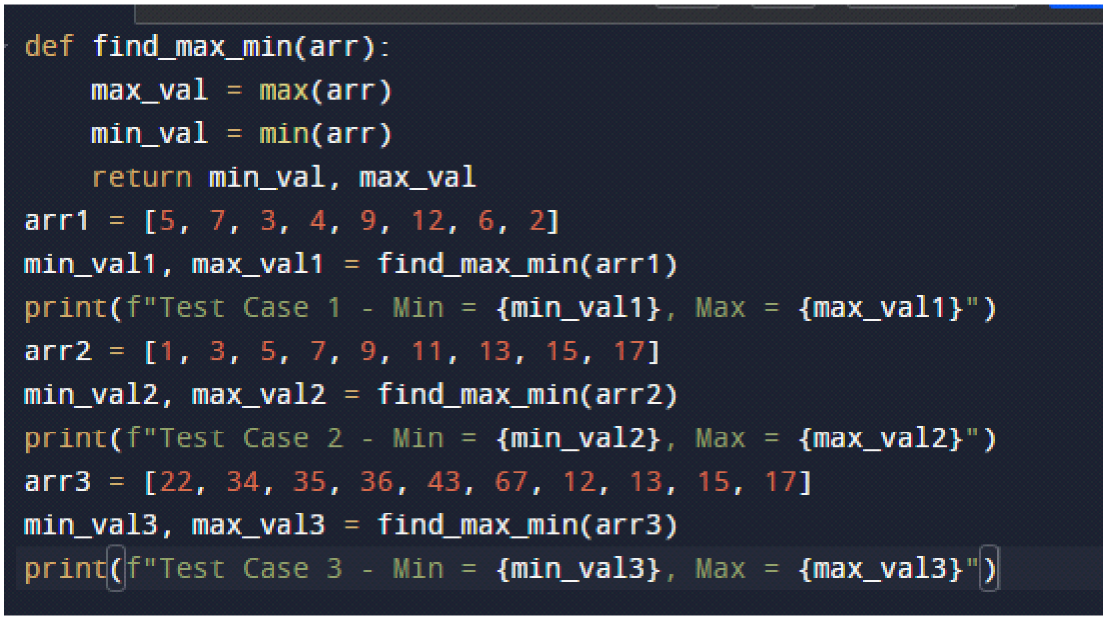
Algorithm:

Step-1: Intialize the max & min with 1st element of array

Step-2: Iterate through the array updating,min & max if a smaller or larger

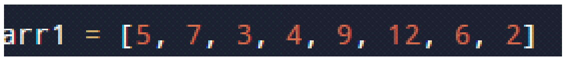
Step-3: Return min & max

Program:



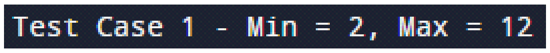
```python
def find_max_min(arr):
    max_val = max(arr)
    min_val = min(arr)
    return min_val, max_val
arr1 = [5, 7, 3, 4, 9, 12, 6, 2]
min_val1, max_val1 = find_max_min(arr1)
print(f"Test Case 1 - Min = {min_val1}, Max = {max_val1}")
arr2 = [1, 3, 5, 7, 9, 11, 13, 15, 17]
min_val2, max_val2 = find_max_min(arr2)
print(f"Test Case 2 - Min = {min_val2}, Max = {max_val2}")
arr3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]
min_val3, max_val3 = find_max_min(arr3)
print(f"Test Case 3 - Min = {min_val3}, Max = {max_val3}")
```

Input:



```
arr1 = [5, 7, 3, 4, 9, 12, 6, 2]
```

Output:



```
Test Case 1 - Min = 2, Max = 12
```

Time Complexity: T(n)=O(n)

Result: The program correctly finds the min & max values input array

33.Consider an array of integers sorted in ascending order: 2,4,6,8,10,12,14,18. Write a Program to find both the maximum and minimum values in the array. Implement using any programming language of your choice. Execute your code and provide the maximum and minimum values found.

Input : N=8, 2,4,6,8,10,12,14,18.

Output : Min = 2, Max =18

Test Cases :

Input : N= 9, a[] = {11,13,15,17,19,21,23,35,37}

Output : Min = 11, Max = 37

Test Cases :

Input : N= 10, a[] = {22,34,35,36,43,67, 12,13,15,17}

Output : Min 12, Max 67

Aim: Find the maximum & minimum values in array.

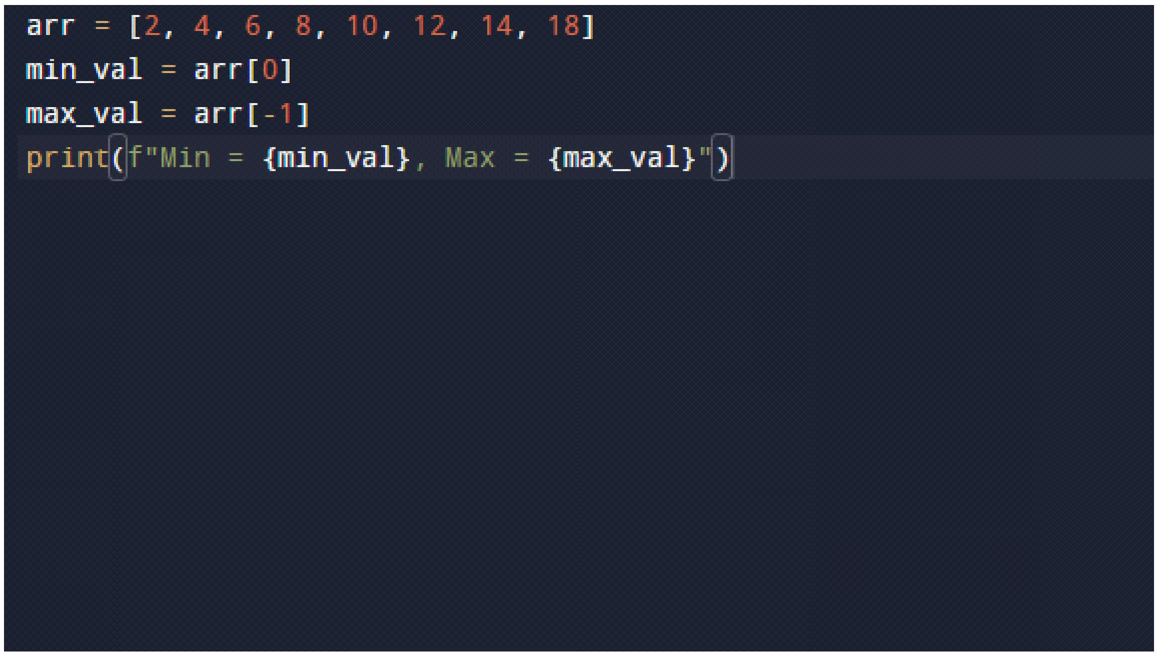
Algorithm:

Step-1: Since the array,the min values

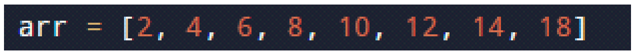
Step-2: The max values is last element

Step-3: Return the first & last element as the min & max values

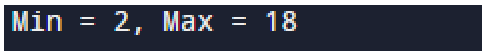
Program:


```python
arr = [2, 4, 6, 8, 10, 12, 14, 18]
min_val = arr[0]
max_val = arr[-1]
print(f"Min = {min_val}, Max = {max_val}")
```

Input:


```
arr = [2, 4, 6, 8, 10, 12, 14, 18]
```

Output:


```
Min = 2, Max = 18
```

Time Complexity: T(n)=O(n)

Result: The program correctly finds the min & max values in the sorted array

34.You are given an unsorted array 31,23,35,27,11,21,15,28. Write a program for Merge Sort and implement using any programming language of your choice.

Test Cases :

Input : N= 8, a[] = {31,23,35,27,11,21,15,28}

Output : 11,15,21,23,27,28,31,35

Test Cases :

Input : N= 10, a[] = {22,34,25,36,43,67, 52,13,65,17}

Output : 13,17,22,25,34,36,43,52,65,67

Aim: Implement the merge sort algorithm to sort an unsorted array

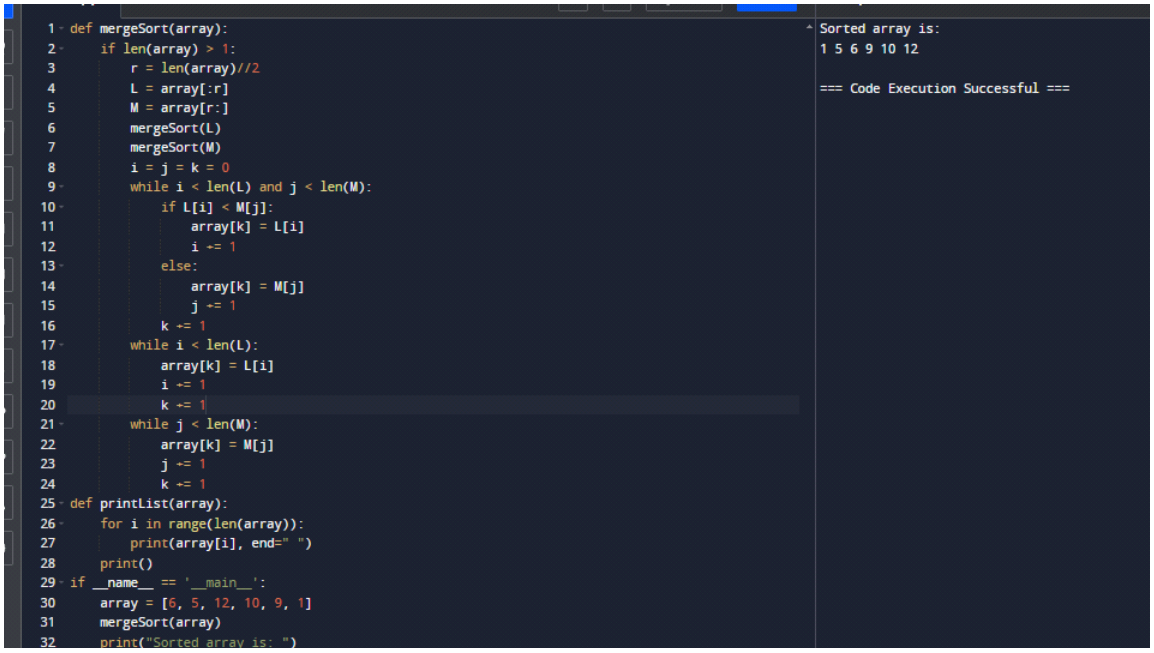
Algorithm:

Step-1: Divide the array into two halves untill

Step-2: Merge the 2 halves in sorted order

Step-3: Repeat step 2 untill the entire array is sorted

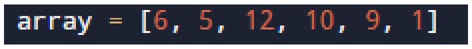
Program:



```python
def mergeSort(array):
    if len(array) > 1:
        r = len(array)//2
        L = array[:r]
        M = array[r:]
        mergeSort(L)
        mergeSort(M)
        i = j = k = 0
        while i < len(L) and j < len(M):
            if L[i] < M[j]:
                array[k] = L[i]
                i += 1
            else:
                array[k] = M[j]
                j += 1
            k += 1
        while i < len(L):
            array[k] = L[i]
            i += 1
            k += 1
        while j < len(M):
            array[k] = M[j]
            j += 1
            k += 1
def printList(array):
    for i in range(len(array)):
        print(array[i], end=" ")
    print()
if __name__ == '__main__':
    array = [6, 5, 12, 10, 9, 1]
    mergeSort(array)
    print("Sorted array is: ")
```

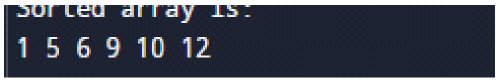
```
Sorted array is:
1 5 6 9 10 12

=== Code Execution Successful ===
```

Input:



```python
array = [6, 5, 12, 10, 9, 1]
```

Output:



```
Sorted array is:
1 5 6 9 10 12
```

Time Complexity: T(n)=O(nlogn)

Result: The program correctly sort the unsorted array using merge sort algorithm

35.Implement the Merge Sort algorithm in a programming language of your choice and test it on the array 12,4,78,23,45,67,89,1. Modify your implementation to count the number of comparisons made during the sorting process. Print this count along with the sorted array.

Test Cases :

Input : N= 8, a[] = {12,4,78,23,45,67,89,1}

Output : 1,4,12,23,45,67,78,89

Test Cases :

Input : N= 7, a[] = {38,27,43,3,9,82,10}

Output : 3,9,10,27,38,43,82

Aim: Implement the merge sort algorithm to sort an unsorted array

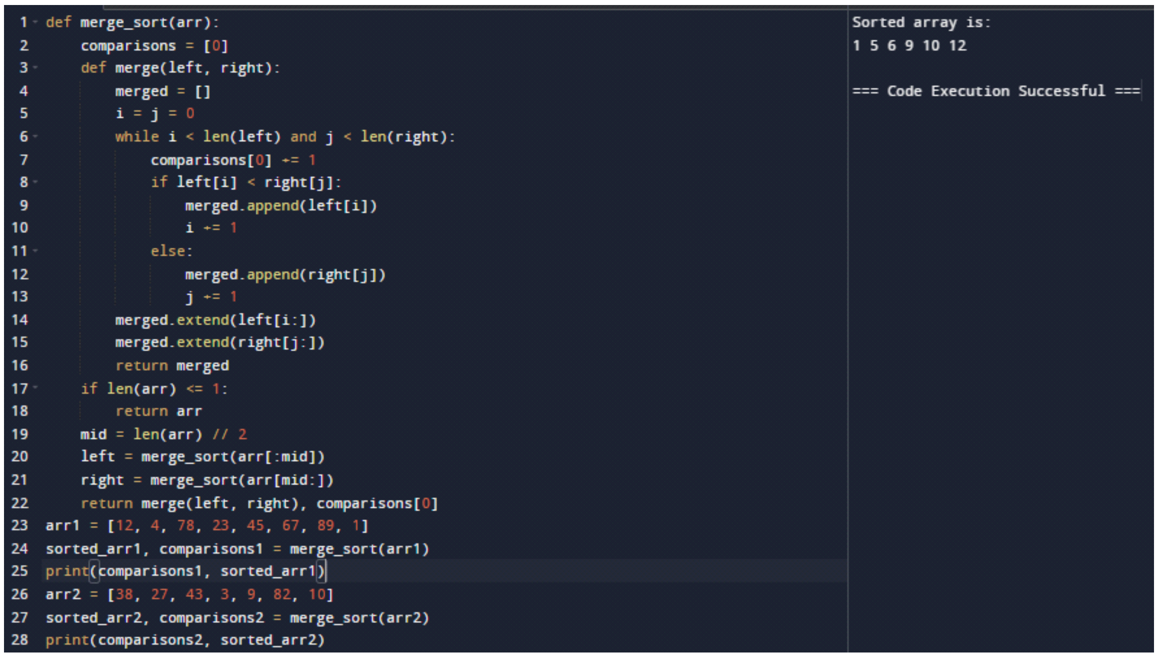
Algorithm:

Step-1: Divide the array into two halves untill each

Step-2: Merge adjacent sub-arrays by comparing elements

Step-3: Increment a counter for each comparison made during merge process

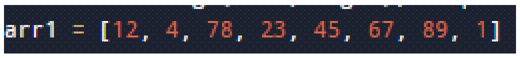
Program:



```
def merge_sort(arr):
    comparisons = [0]
    def merge(left, right):
        merged = []
        i = j = 0
        while i < len(left) and j < len(right):
            comparisons[0] += 1
            if left[i] < right[j]:
                merged.append(left[i])
                i += 1
            else:
                merged.append(right[j])
                j += 1
        merged.extend(left[i:])
        merged.extend(right[j:])
        return merged
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right), comparisons[0]
arr1 = [12, 4, 78, 23, 45, 67, 89, 1]
sorted_arr1, comparisons1 = merge_sort(arr1)
print(comparisons1, sorted_arr1)
arr2 = [38, 27, 43, 3, 9, 82, 10]
sorted_arr2, comparisons2 = merge_sort(arr2)
print(comparisons2, sorted_arr2)
```

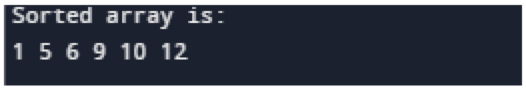
```
Sorted array is:
1 5 6 9 10 12

=== Code Execution Successful ===
```

Input:



```
arr1 = [12, 4, 78, 23, 45, 67, 89, 1]
```

Output:



```
Sorted array is:
1 5 6 9 10 12
```

Time Complexity: T(n)=O(nlogn)

Result: The implement correctly sort the input array and counts the num of comparision made during the sorting process

36.Given an unsorted array 10,16,8,12,15,6,3,9,5 Write a program to perform Quick Sort.

Choose the first element as the pivot and partition the array accordingly. Show the array

after this partition. Recursively apply Quick Sort on the sub-arrays formed. Display the

array after each recursive call until the entire array is sorted.

Input : N= 9, a[]= {10,16,8,12,15,6,3,9,5}

Output : 3,5,6,8,9,10,12,15,16

Test Cases :

Input : N= 8, a[] = {12,4,78,23,45,67,89,1}

Output : 1,4,12,23,45,67,78,89

Test Cases :

Input : N= 7, a[] = {38,27,43,3,9,82,10}

Output : 3,9,10,27,38,43,82

Aim: Implement the quick sort algorithm to sort an unsorted array of algorithm

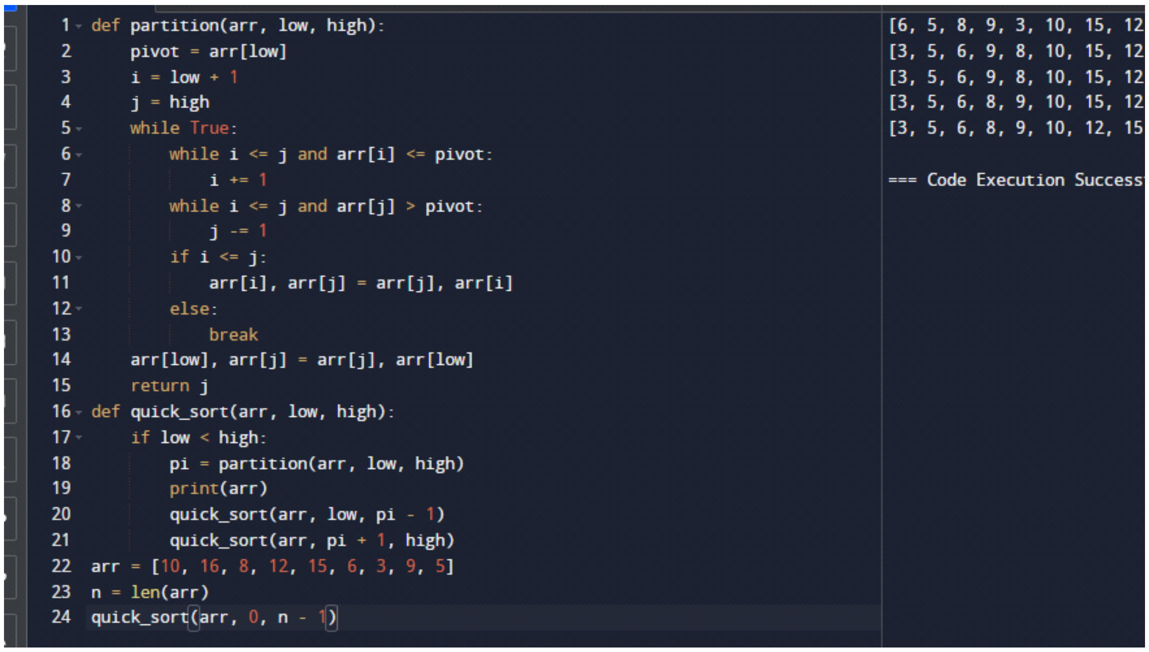
Algorithm:

Step-1: Select 1st element as pivot

Step-2: Parition the array around the pivot placing elements less than pivot

Step-3: Recursively apply quick sort on sub-array formed

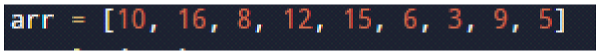
Program:



```
def partition(arr, low, high):
    pivot = arr[low]
    i = low + 1
    j = high
    while True:
        while i <= j and arr[i] <= pivot:
            i += 1
        while i <= j and arr[j] > pivot:
            j -= 1
        if i <= j:
            arr[i], arr[j] = arr[j], arr[i]
        else:
            break
    arr[low], arr[j] = arr[j], arr[low]
    return j
def quick_sort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        print(arr)
        quick_sort(arr, low, pi - 1)
        quick_sort(arr, pi + 1, high)
arr = [10, 16, 8, 12, 15, 6, 3, 9, 5]
n = len(arr)
quick_sort(arr, 0, n - 1)
```

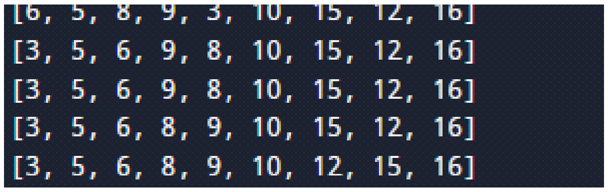
```
[6, 5, 8, 9, 3, 10, 15, 12
[3, 5, 6, 9, 8, 10, 15, 12
[3, 5, 6, 9, 8, 10, 15, 12
[3, 5, 6, 8, 9, 10, 15, 12
[3, 5, 6, 8, 9, 10, 12, 15

=== Code Execution Success
```

Input:



```
arr = [10, 16, 8, 12, 15, 6, 3, 9, 5]
```

Output:



```
[6, 5, 8, 9, 3, 10, 15, 12, 16]
[3, 5, 6, 9, 8, 10, 15, 12, 16]
[3, 5, 6, 9, 8, 10, 15, 12, 16]
[3, 5, 6, 8, 9, 10, 15, 12, 16]
[3, 5, 6, 8, 9, 10, 12, 15, 16]
```

Time Complexity: T(n)=O(nlogn)

Result: The implement correctly sort the input array using quickly sort

37. Implement the Quick Sort algorithm in a programming language of your choice and test it on the array 19,72,35,46,58,91,22,31. Choose the middle element as the pivot and partition the array accordingly. Show the array after this partition. Recursively apply Quick Sort on the sub-arrays formed. Display the array after each recursive call until the entire array is sorted. Execute your code and show the sorted array.

Input : N= 8, a[] = {19,72,35,46,58,91,22,31}

Output : 19,22,31,35,46,58,72,91

Aim: Implement the quick sort algorithm to sort an unsorted array of integers.

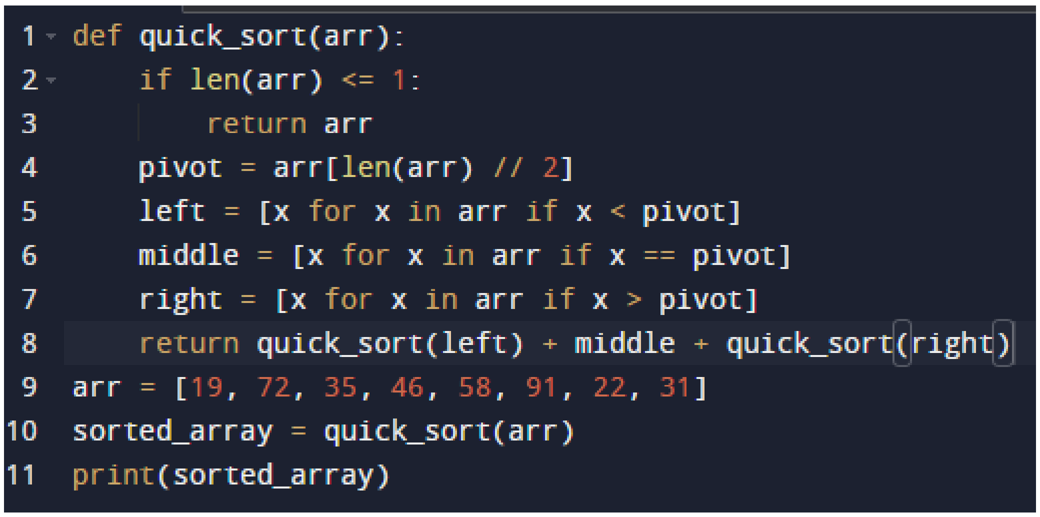
Algorithm:

step-1: Select the middle elementas pivot

step-2: Partition the array around the pivot, placing elements apply quick sort on the pivot

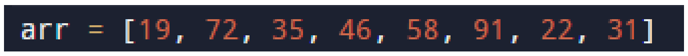
srep-3: Recursively apply quick sort on the sub-arrays

Program:



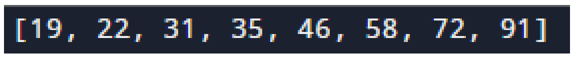
```python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)
arr = [19, 72, 35, 46, 58, 91, 22, 31]
sorted_array = quick_sort(arr)
print(sorted_array)
```

Input:



```
arr = [19, 72, 35, 46, 58, 91, 22, 31]
```

Output:



```
[19, 22, 31, 35, 46, 58, 72, 91]
```

Time complexity:

T(n)=O(nlogn)

Result:

The implentation correctly sorts the input array using algorithm

38. Implement the Binary Search algorithm in a programming language of your choice and test it on the array 5,10,15,20,25,30,35,40,45 to find the position of the element 20. Execute your code and provide the index of the element 20. Modify your implementation to count the number of comparisons made during the search process. Print this count along with the result.

Input : N= 9, a[] = {5,10,15,20,25,30,35,40,45}, search key = 20

Output : 4

Aim: Implementation the binary search algorithm to find position of an element in sorted array

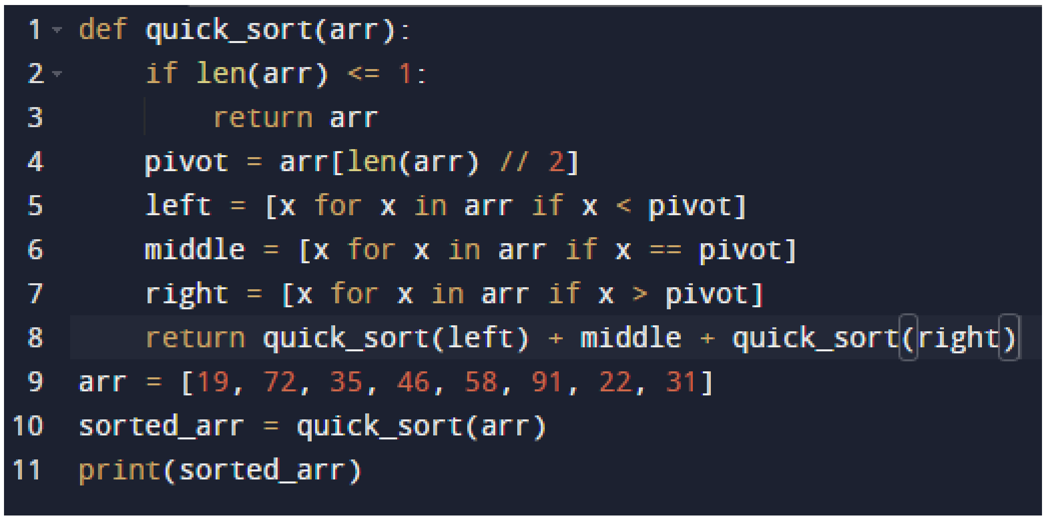
Algorithm:

step-1: Initialize low and high to the start and end of array and calculate mid

step-2: compare the element at mid with target update low (or) high

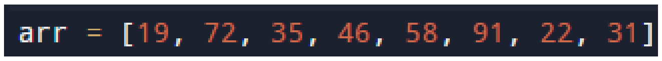
step-3: Repeat steps 1-2 until low>high then return index

Program:



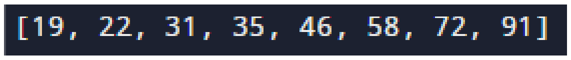
```python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)
arr = [19, 72, 35, 46, 58, 91, 22, 31]
sorted_arr = quick_sort(arr)
print(sorted_arr)
```

Input:



```
arr = [19, 72, 35, 46, 58, 91, 22, 31]
```

Output:



```
[19, 22, 31, 35, 46, 58, 72, 91]
```

Time complexity:

T(n)=O(logn)

Result:

The program successfully finds the position of element 20 in the array using binary search.

39.You are given a sorted array 3,9,14,19,25,31,42,47,53 and asked to find the position of the element 31 using Binary Search. Show the mid-point calculations and the steps involved in finding the element. Display, what would happen if the array was not sorted, how would this impact the performance and correctness of the Binary Search algorithm?

Input : N= 9, a[] = {3,9,14,19,25,31,42,47,53}, search key = 31

Output : 6

Test cases

Input : N= 7, a[] = {13,19,24,29,35,41,42}, search key = 42

Output : 7

Test cases

Input : N= 6, a[] = {20,40,60,80,100,120}, search key = 60

Output : 3

Aim: To find the position of the element in the given sorted array using binary search

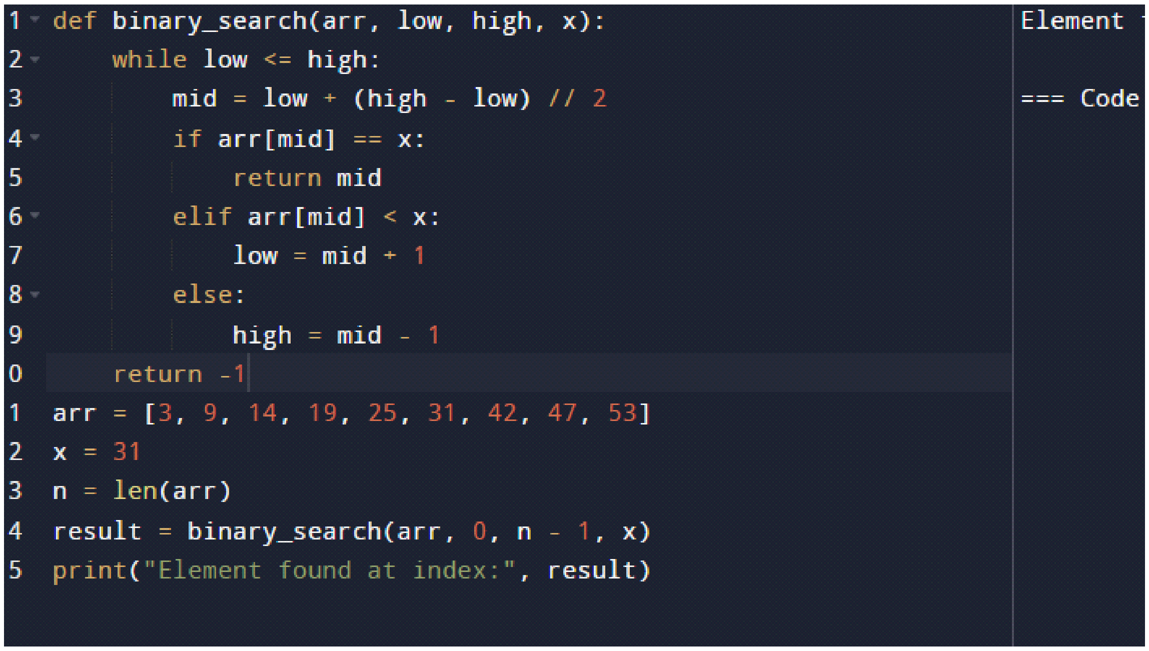
Algorithm:

Step-1: Intialize low & high to the sort and end of array

Step-2: Compare the element mid with the target.Update low & high

Step-3: Repat step 1-2 untill low>high then return the index

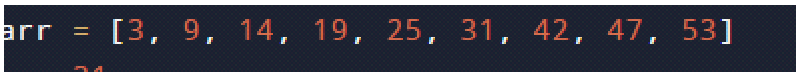
Program:



```
def binary_search(arr, low, high, x):
    while low <= high:
        mid = low + (high - low) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1
arr = [3, 9, 14, 19, 25, 31, 42, 47, 53]
x = 31
n = len(arr)
result = binary_search(arr, 0, n - 1, x)
print("Element found at index:", result)
```

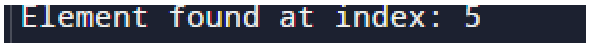
```
Element

=== Code
```

Input:



```
arr = [3, 9, 14, 19, 25, 31, 42, 47, 53]
```

Output:



```
Element found at index: 5
```

Time Complexity: T(n)=O(logn)

Result: The element is found at index 5 in array

40.Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).

(i) Input : points = [[1,3],[-2,2],[5,8],[0,1]],k=2

Output:[[-2, 2], [0, 1]]

(ii) Input: points = [[1, 3], [-2, 2]], k = 1

Output: [[-2, 2]]

(iii) Input: points = [[3, 3], [5, -1], [-2, 4]], k = 2

Output: [[3, 3], [-2, 4]]

Aim: Find the k-closest points to the origin (0,0) in the given array 0f points

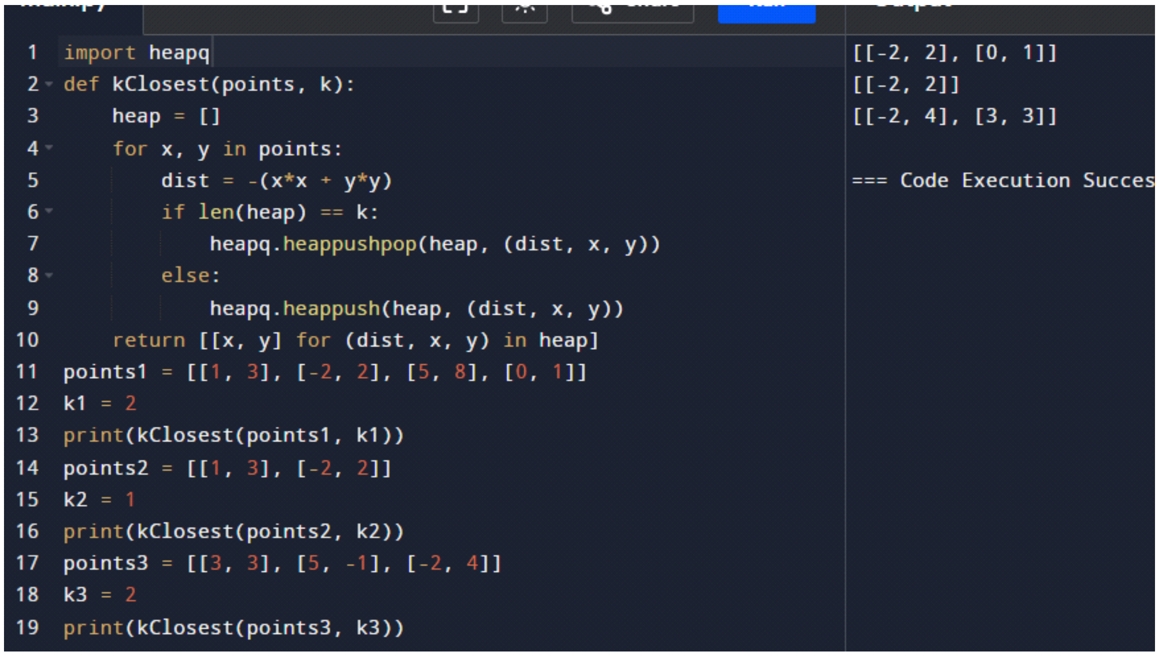
Algorithm:

Step-1: Calculate the eucliden distance of each point from origin

Step-2: Sort the point based on their distance from orign

Step-3: Return the first k points from sorted array

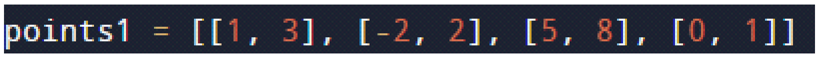
Program:



```
import heapq
def kClosest(points, k):
    heap = []
    for x, y in points:
        dist = -(x*x + y*y)
        if len(heap) == k:
            heapq.heappushpop(heap, (dist, x, y))
        else:
            heapq.heappush(heap, (dist, x, y))
    return [[x, y] for (dist, x, y) in heap]
points1 = [[1, 3], [-2, 2], [5, 8], [0, 1]]
k1 = 2
print(kClosest(points1, k1))
points2 = [[1, 3], [-2, 2]]
k2 = 1
print(kClosest(points2, k2))
points3 = [[3, 3], [5, -1], [-2, 4]]
k3 = 2
print(kClosest(points3, k3))
```

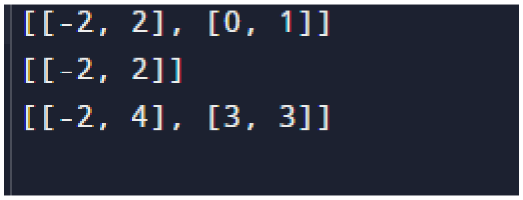
```
[[-2, 2], [0, 1]]
[[-2, 2]]
[[-2, 4], [3, 3]]

=== Code Execution Succes
```

Input:



```
points1 = [[1, 3], [-2, 2], [5, 8], [0, 1]]
```

Output:



```
[[-2, 2], [0, 1]]
[[-2, 2]]
[[-2, 4], [3, 3]]
```

Time Complexity: T(n)=O(nlogk)

Result: The k closest points to the origin are returned in output

41. Given four lists A, B, C, D of integer values,Write a program to compute how many tuples n(i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero. (i) Input: A = [1, 2], B = [-2, -1], C = [-1, 2], D = [0, 2] Output: 2 (ii) Input: A = [0], B = [0], C = [0], D = [0] Output: 1

Aim:

Compute the number of tuples (i,j,k,l)(i,j,k,l) such that A[i]+B[j]+C[k]+D[l]=0$A[i]+B[j]+C[k]+D[l]=0$ given four integer lists AA, BB, CC, and DD.

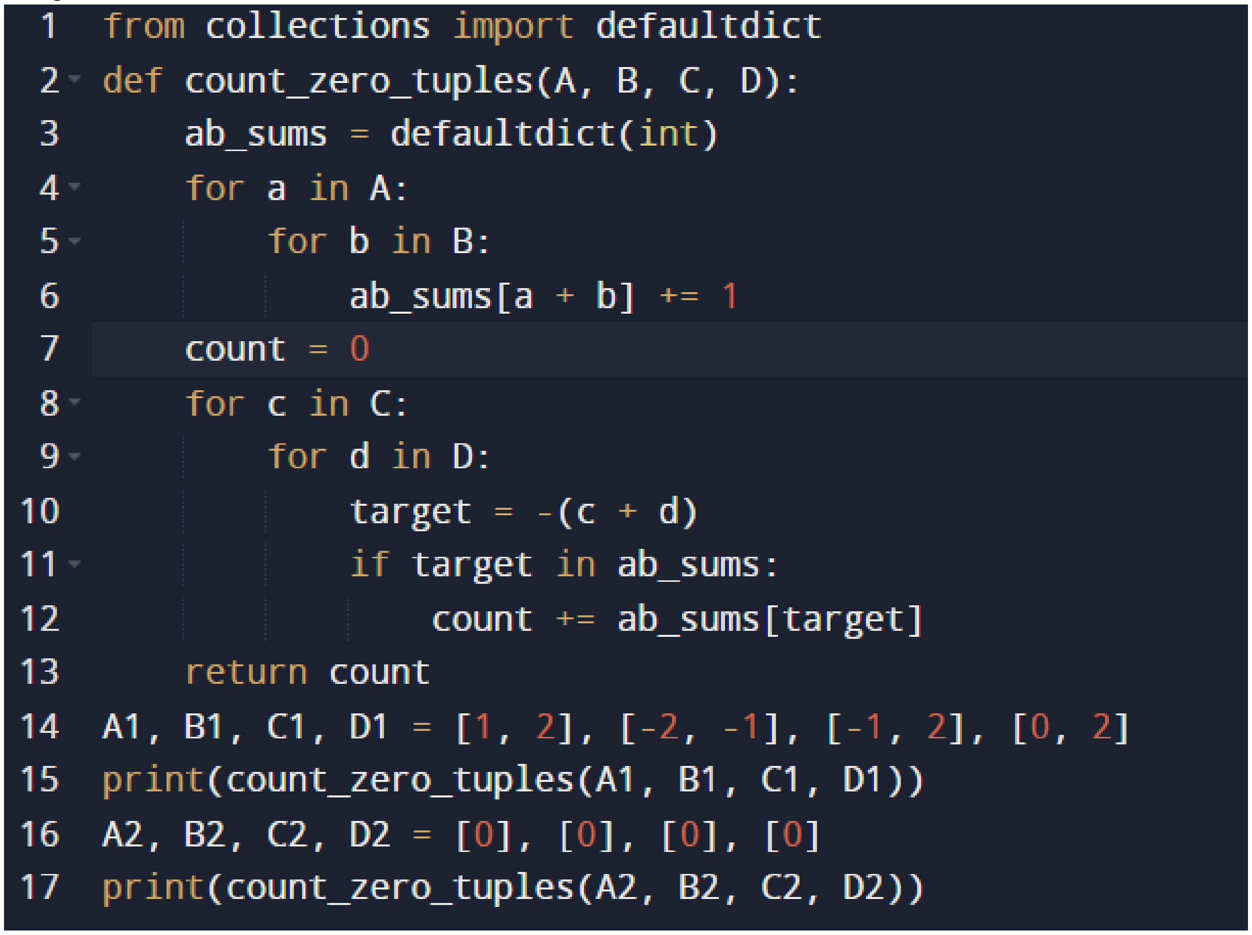
Algorithm:

Step-1:Compute all possible sums of pairs from lists AA and BB, storing them in a dictionary with their counts.

Step-2:Compute all possible sums of pairs from lists CC and DD, and for each sum, check if its negation exists in the dictionary from step 1.

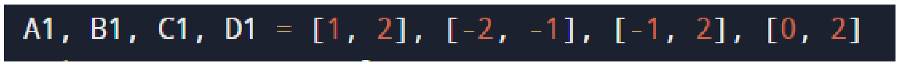
Step-3:Count valid tuples where the sums from step 2 match the negations of sums from step 1.

Program:



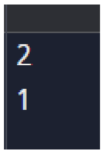
```python
from collections import defaultdict
def count_zero_tuples(A, B, C, D):
    ab_sums = defaultdict(int)
    for a in A:
        for b in B:
            ab_sums[a + b] += 1
    count = 0
    for c in C:
        for d in D:
            target = -(c + d)
            if target in ab_sums:
                count += ab_sums[target]
    return count
A1, B1, C1, D1 = [1, 2], [-2, -1], [-1, 2], [0, 2]
print(count_zero_tuples(A1, B1, C1, D1))
A2, B2, C2, D2 = [0], [0], [0], [0]
print(count_zero_tuples(A2, B2, C2, D2))
```

Input:



```
A1, B1, C1, D1 = [1, 2], [-2, -1], [-1, 2], [0, 2]
```

Output:



```
2
1
```

Time Complexity: T(n)=O(n^2)

Result: The program correctly computes the number of valid tuples for given inputs.