**1.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**

**Program:**

**2.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**

**Program:**

def find\_min\_max(arr):

min\_val = arr[0]

max\_val = arr[-1]

return min\_val, max\_val

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

min\_val, max\_val = find\_min\_max(array)

print(f"Array: {array}")

print(f"Min = {min\_val}, Max = {max\_val}")

**3.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**

**Program:**

def merge\_sort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

left = merge\_sort(arr[:mid])

right = merge\_sort(arr[mid:])

return merge(left, right)

def merge(left, right):

merged = []

while left and right:

if left[0] <= right[0]:

merged.append(left.pop(0))

else:

merged.append(right.pop(0))

merged.extend(left or right)

return merged

arr = [31, 23, 35, 27, 11, 21, 15, 28]

print("Input: ", arr)

print("Output: ", merge\_sort(arr))

**4.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**

**Program:**

def merge\_sort(arr):

if len(arr) <= 1:

return arr, 0

mid = len(arr) // 2

left, left\_cmp = merge\_sort(arr[:mid])

right, right\_cmp = merge\_sort(arr[mid:])

merged, merge\_cmp = merge(left, right)

return merged, left\_cmp + right\_cmp + merge\_cmp

def merge(left, right):

result = []

i = j = cmp = 0

while i < len(left) and j < len(right):

cmp += 1

if left[i] < right[j]:

result.append(left[i])

i += 1

else:

result.append(right[j])

j += 1

return result + left[i:] + right[j:], cmp

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

sorted\_arr, comparisons = merge\_sort(arr)

print("Sorted Array:", sorted\_arr)

print("Comparisons:", comparisons)

**5.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**

**Program:**

def quick\_sort(arr):

if len(arr) <= 1:

return arr

pivot = arr[0]

less = [x for x in arr[1:] if x <= pivot]

greater = [x for x in arr[1:] if x > pivot]

print("Pivot: ", pivot, "Less: ", less, "Greater: ", greater)

return quick\_sort(less) + [pivot] + quick\_sort(greater)

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

print("Input: ", arr)

sorted\_arr = quick\_sort(arr)

print("Output: ", sorted\_arr)

**6.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**

**Program:**

def quick\_sort(arr, low, high):

if low < high:

pi = partition(arr, low, high)

print(f'After partitioning with pivot at index {pi}: {arr}')

quick\_sort(arr, low, pi - 1)

quick\_sort(arr, pi + 1, high)

def partition(arr, low, high):

mid = (low + high) // 2

arr[mid], arr[high] = arr[high], arr[mid] # Move pivot to end for partitioning

pivot = arr[high]

i = low - 1

for j in range(low, high):

if arr[j] < pivot:

i += 1

arr[i], arr[j] = arr[j], arr[i]

arr[i + 1], arr[high] = arr[high], arr[i + 1]

return i + 1

def main():

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

print(f'Initial array: {arr}')

quick\_sort(arr, 0, len(arr) - 1)

print(f'Sorted array: {arr}')

if \_name\_ == "\_main\_":

main()

**7.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**

**Program:**

def binary\_search(arr, low, high, x):

global comparison\_count

while low <= high:

mid = (low + high) // 2

comparison\_count += 1

if arr[mid] == x:

return mid

elif arr[mid] < x:

low = mid + 1

else:

high = mid - 1

return -1

def main():

global comparison\_count

arr = [5, 10, 15, 20, 25, 30, 35, 40, 45]

search\_key = 20

comparison\_count = 0

index = binary\_search(arr, 0, len(arr) - 1, search\_key)

if index != -1:

print(f'Element {search\_key} is at index {index}')

else:

print(f'Element {search\_key} is not in the array')

print(f'Number of comparisons: {comparison\_count}')

if \_name\_ == "\_main\_":

main()

**8.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**

**Program:**

def binary\_search(arr, low, high, x):

step = 1

while low <= high:

mid = (low + high) // 2

print(f'Step {step}: Current low = {low}, high = {high}, mid = {mid}, mid\_value = {arr[mid]}')

if arr[mid] == x:

return mid

elif arr[mid] < x:

low = mid + 1

else:

high = mid - 1

step += 1

return -1

def main():

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

search\_key = 31

index = binary\_search(arr, 0, len(arr) - 1, search\_key)

if index != -1:

print(f'Element {search\_key} is at index {index}')

else:

print(f'Element {search\_key} is not in the array')

if \_name\_ == "\_main\_":

main()

**9.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]]**

**Program:**

import heapq

def kClosest(points, k):

return heapq.nsmallest(k, points, key=lambda x: x[0]\*\*2 + x[1]\*\*2)

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

k1 = 2

print(kClosest(points1, k1)) # Output: [[-2, 2], [0, 1]]

points2 = [[1, 3], [-2, 2]]

k2 = 1

print(kClosest(points2, k2)) # Output: [[-2, 2]]

points3 = [[3, 3], [5, -1], [-2, 4]]

k3 = 2

print(kClosest(points3, k3)) # Output: [[3, 3], [-2, 4]]

**10.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**

**Program:**

from collections import defaultdict

def count\_tuples(A, B, C, D):

sum\_ab = defaultdict(int)

for a in A:

for b in B:

sum\_ab[a + b] += 1

count = 0

for c in C:

for d in D:

count += sum\_ab[-(c + d)]

return count

def main():

A1 = [1, 2]

B1 = [-2, -1]

C1 = [-1, 2]

D1 = [0, 2]

result1 = count\_tuples(A1, B1, C1, D1)

print(f'Output for input A={A1}, B={B1}, C={C1}, D={D1}: {result1}')

A2 = [0]

B2 = [0]

C2 = [0]

D2 = [0]

result2 = count\_tuples(A2, B2, C2, D2)

print(f'Output for input A={A2}, B={B2}, C={C2}, D={D2}: {result2}')

if \_name\_ == "\_main\_":

main()