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.**

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

min_val = min(a)

max_val = max(a)

print(f"Min = {min_val}, Max = {max_val}")
```

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.**

```
 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}")
```

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.**

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]
        merge_sort(L)
        merge_sort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
        arr = [31, 23, 35, 27, 11, 21, 15, 28]
        merge_sort(arr)
        print("Sorted array:", 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.**

```
def merge_sort(arr):
    comparisons = 0
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]
        comparisons += merge_sort(left_half)
        comparisons += merge_sort(right_half)
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1
            comparisons += 1
        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1
        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1
    return comparisons
arr = [12, 4, 78, 23, 45, 67, 89, 1]
comparisons = merge_sort(arr)
print("Sorted Array:", arr)
print("Number of 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.**

```
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:
        pivot_index = partition(arr, low, high)
        print(arr)
        quick_sort(arr, low, pivot_index - 1)
        quick_sort(arr, pivot_index + 1, high)
N = 9
a = [10, 16, 8, 12, 15, 6, 3, 9, 5]
quick_sort(a, 0, N - 1)
print("Sorted Array:", a)
```

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.**

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[len(arr) // 2]
        less = [x for x in arr if x < pivot]
        equal = [x for x in arr if x == pivot]
        greater = [x for x in arr if x > pivot]
        return quick_sort(less) + equal + quick_sort(greater)
array = [19, 72, 35, 46, 58, 91, 22, 31]
sorted_array = quick_sort(array)
print(sorted_array)
```

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.**

```
    def binary_search(arr, target):

    left, right = 0, len(arr) - 1

    comparisons = 0

    while left <= right:

        mid = left + (right - left) // 2

        comparisons += 1

        if arr[mid] == target:

            return mid, comparisons

        elif arr[mid] < target:

            left = mid + 1

        else:

            right = mid - 1

    return -1, comparisons

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

target = 20

index, comparisons = binary_search(arr, target)

if index != -1:

    print(f"Element {target} found at index {index}.")

else:

    print(f"Element {target} not found in the array.")

print(f"Number of comparisons made: {comparisons}")
```

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?**

```
  def binary_search(arr, target):
  low = 0
  high = len(arr) - 1
  while low <= high:
    mid = (low + high) // 2
    if arr[mid] == target:
      return mid
    elif arr[mid] < target:
      low = mid + 1
    else:
      high = mid - 1
  return -1
sorted_array = [3, 9, 14, 19, 25, 31, 42, 47, 53]
target_element = 31
result = binary_search(sorted_array, target_element)
print("Position of element 31 in the sorted array:", result)
```

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).**

```
    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]
  points = [[1, 3], [-2, 2], [5, 8], [0, 1]]
  k = 2
  print(kClosest(points, k))
```

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.

```
from collections import defaultdict
def fourSumCount(A, B, C, D):
    AB_sum = defaultdict(int)
    count = 0
    for a in A:
        for b in B:
            AB_sum[a + b] += 1
    for c in C:
        for d in D:
            count += AB_sum[-c - d]
    return count
A = [1, 2]
B = [-2, -1]
C = [-1, 2]
D = [0, 2]
print(fourSumCount(A, B, C, D))
```

11.To Implement the Median of Medians algorithm ensures that you handle the worst-case time complexity efficiently while finding the k-th smallest element in an unsorted array.

```
def median_of_medians(arr, k):

def partition(arr, l, r, pivot_idx):

    pivot_value = arr[pivot_idx]

    arr[pivot_idx], arr[r] = arr[r], arr[pivot_idx]

    store_idx = l

    for i in range(l, r):

    if arr[i] < pivot_value:

            arr[store_idx], arr[i] = arr[i], arr[store_idx]

            store_idx += 1

    arr[store_idx], arr[r] = arr[r], arr[store_idx]

    return store_idx

def select(arr, l, r, k):

    if l == r:

        return arr[l]

    pivot_idx = l

    while True:

        pivot_idx = partition(arr, l, r, pivot_idx)

        if pivot_idx == k:

            return arr[k]

        elif k < pivot_idx:

            r = pivot_idx - 1

        else:

            l = pivot_idx + 1

n = len(arr)
```

```
    groups = [arr[i:i + 5] for i in range(0, n, 5)]
    medians = [sorted(group)[len(group) // 2] for group in groups]
    if len(medians) <= 5:
        pivot = sorted(medians)[len(medians) // 2]
    else:
        pivot = median_of_medians(medians, len(medians) // 2)
    pivot_idx = arr.index(pivot)
    arr[pivot_idx], arr[n - 1] = arr[n - 1], arr[pivot_idx]
    return select(arr, 0, n - 1, k)
arr = [12, 3, 5, 7, 19]
k = 2
result = median_of_medians(arr, k)
print(result)
```

12.To Implement a function median_of_medians(arr, k) that takes an unsorted array arr and an integer k, and returns the k-th smallest element in the array.

```
def median_of_medians(arr, k):
    sublists = [arr[j:j+5] for j in range(0, len(arr), 5)]
    medians = [sorted(sublist)[len(sublist)//2] for sublist in sublists]
    if len(medians) <= 5:
        pivot = sorted(medians)[len(medians)//2]
    else:
        pivot = median_of_medians(medians, len(medians)//2)
    low = [elem for elem in arr if elem < pivot]
    high = [elem for elem in arr if elem > pivot]
   equal = [elem for elem in arr if elem == pivot]
    if k <= len(low):
        return median_of_medians(low, k)
    elif k > len(low) + len(equal):
        return median_of_medians(high, k - len(low) - len(equal))
    else:
        return pivot
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k = 6
```

```
print(median_of_medians(arr, k))
```

13.Write a program to implement Meet in the Middle Technique. Given an array of integers and a target sum, find the subset whose sum is closest to the target. You will use the Meet in the Middle technique to efficiently find this subset.

```
from itertools import chain, combinations

def subsets(arr):

    return chain(*[combinations(arr, i) for i in range(len(arr)+1)])

def closest_sum_subset(arr, target):

    n = len(arr)

    half = n // 2

    left_half = list(subsets(arr[:half]))

    right_half = list(subsets(arr[half:]))

    left_half_sums = [sum(subset) for subset in left_half]

    right_half_sums = [sum(subset) for subset in right_half]

    left_half_sums.sort()

    right_half_sums.sort()

    closest_sum = float('inf')

    closest_subset = None

    i = 0

    j = len(right_half_sums) - 1

    while i < len(left_half_sums) and j >= 0:

    current_sum = left_half_sums[i] + right_half_sums[j]

    if abs(target - current_sum) < abs(target - closest_sum):

            closest_sum = current_sum

            closest_subset = left_half[i] + right_half[j]

        if current_sum > target:

            j -= 1

        else:

            i += 1

    return closest_subset

    arr = [45, 34, 4, 12, 5, 2]

    target_sum = 42

    result = closest_sum_subset(arr, target_sum)

    print(result)
```

14.Write a program to implement Meet in the Middle Technique. Given a large array of integers and an exact sum E, determine if there is any subset that sums exactly to E. Utilize the Meet in the Middle technique to handle the potentially large size of the array. Return true if there is a subset that sums exactly to E, otherwise return false.

```
def is_subset_sum(arr, n, E):
    half = n // 2
    subset1 = []
    subset2 = []
    for i in range(1 << half):
        sum = 0
    for j in range(half):
            if i & (1 << j):
                sum += arr[j]
        subset1.append(sum)
    for i in range(1 << (n - half)):
        sum = 0
        for j in range(n - half):
            if i & (1 << j):
                sum += arr[half + j]
        subset2.append(sum)
    subset1.sort()
    for s in subset2:
        if s == E or E - s in subset1:
            return True
    return False
arr = [1, 3, 9, 2, 7, 12]
E = 15
n = len(arr)
if is_subset_sum(arr, n, E):
    print("Subset with sum E exists")
else:
    print("No subset with sum E")
```

15**. Given two 2×2 Matrices A and B**

A=[1 7 ; 3 5]   B= [1 3 ; 7 5]

Use Strassen's matrix multiplication algorithm to compute the product matrix C such that C=A×B.

```
def karatsuba(x, y):
    if x < 10 or y < 10:
        return x * y
    m = max(len(str(x)), len(str(y)))
    m2 = m // 2
    high1, low1 = divmod(x, 10**m2)
    high2, low2 = divmod(y, 10**m2)
    z0 = karatsuba(low1, low2)
    z1 = karatsuba((low1 + high1), (low2 + high2))
    z2 = karatsuba(high1, high2)
    return z2 * 10*(2*m2) + (z1 - z2 - z0) * 10*m2 + z0
# Test Case
x = 1234
y = 5678
z = karatsuba(x, y)
print(f"Result: {x} x {y} = {z}")
```

16. **Given two integers X=1234 and Y=5678: Use the Karatsuba algorithm to compute the product Z=X x Y**

```
    def karatsuba(x, y):
    if x < 10 or y < 10:
        return x * y
    m = max(len(str(x)), len(str(y)))
    m2 = m // 2
    high1, low1 = divmod(x, 10**m2)
    high2, low2 = divmod(y, 10**m2)

    z0 = karatsuba(low1, low2)

    z1 = karatsuba((low1 + high1), (low2 + high2))

    z2 = karatsuba(high1, high2)

    return z2 * 10**(2*m2) + (z1 - z2 - z0) * 10**m2 + z0

x = 1234

y = 5678

z = karatsuba(x, y)

print(f"Result of {x} x {y} using Karatsuba Algorithm: {z}")
```