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

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

PROGRAM:

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}")

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

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

PROGRAM:

def find\_max\_min(arr):

return min(arr), max(arr)

arr1 = [2, 4, 6, 8, 10, 12, 14, 18]

min\_val1, max\_val1 = find\_max\_min(arr1)

print(f"Input: {arr1}")

print(f"Output: Min = {min\_val1}, Max = {max\_val1}")

arr2 = [11, 13, 15, 17, 19, 21, 23, 35, 37]

min\_val2, max\_val2 = find\_max\_min(arr2)

print(f"Input: {arr2}")

print(f"Output: 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"Input: {arr3}")

print(f"Output: Min = {min\_val3}, Max = {max\_val3}")

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

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

PROGRAM:

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

return arr

# Test Cases

arr1 = [31, 23, 35, 27, 11, 21, 15, 28]

arr2 = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]

print("Input Array 1:", arr1)

print("Sorted Array 1:", merge\_sort(arr1))

print("Input Array 2:", arr2)

print("Sorted Array 2:", merge\_sort(arr2))

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

Test Cases :

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

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

PROGRAM:

def merge\_sort(arr):

comparisons = 0

def merge(left, right):

nonlocal comparisons

merged = []

i = j = 0

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

comparisons += 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

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)

1. 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,

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)

# Perform Quick Sort

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

1. 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

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

PROGRAM:

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)

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

sorted\_array = quick\_sort(array)

print(sorted\_array)

1. 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

Test cases

Input : N= 6, a[] = {10,20,30,40,50,60}, search key = 50

Output : 5

Input : N= 7, a[] = {21,32,40,54,65,76,87}, search key = 32

Output : 2

PROGRAM:

def binary\_search(arr, x):

low = 0

high = len(arr) - 1

mid = 0

count = 0

while low <= high:

mid = (high + low) // 2

count += 1

if arr[mid] < x:

low = mid + 1

elif arr[mid] > x:

high = mid - 1

else:

return mid, count

return -1, count

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

x = 20

result, comparisons = binary\_search(arr, x)

print("Element found at index:", result)

print("Number of comparisons made:", comparisons)

1. 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

PROGRAM:

def binary\_search(arr, x):

low = 0

high = len(arr) - 1

mid = 0

while low <= high:

mid = (high + low) // 2

if arr[mid] < x:

low = mid + 1

elif arr[mid] > x:

high = mid - 1

else:

return mid

return -1

arr1 = [3, 9, 14, 19, 25, 31, 42, 47, 53]

x1 = 31

print(binary\_search(arr1, x1)) # Output: 5

arr2 = [13, 19, 24, 29, 35, 41, 42]

x2 = 42

print(binary\_search(arr2, x2)) # Output: 6

arr3 = [20, 40, 60, 80, 100, 120]

x3 = 60

print(binary\_search(arr3, x3)) # Output: 2

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

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

1. **Input**: points = [[1, 3], [-2, 2]], k = 1

**Output**: [[-2, 2]]

1. **Input**: points = [[3, 3], [5, -1], [-2, 4]], k = 2

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

def binary\_search(arr, x):

low = 0

high = len(arr) - 1

mid = 0

while low <= high:

mid = (high + low) // 2

if arr[mid] < x:

low = mid + 1

elif arr[mid] > x:

high = mid - 1

else:

return mid

return -1

# Test cases

arr1 = [3, 9, 14, 19, 25, 31, 42, 47, 53]

x1 = 31

print(binary\_search(arr1, x1)) # Output: 5

arr2 = [13, 19, 24, 29, 35, 41, 42]

x2 = 42

print(binary\_search(arr2, x2))

arr3 = [20, 40, 60, 80, 100, 120]

x3 = 60

print(binary\_search(arr3, x3))

1. 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.
2. **Input**: A = [1, 2], B = [-2, -1], C = [-1, 2], D = [0, 2]

**Output**: 2

1. **Input**: A = [0], B = [0], C = [0], D = [0]

**Output**: 1

**PROGRAM:**

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

# Test Cases

A1 = [1, 2]

B1 = [-2, -1]

C1 = [-1, 2]

D1 = [0, 2]

print(f"Output for Test Case 1: {fourSumCount(A1, B1, C1, D1)}")

A2 = [0]

B2 = [0]

C2 = [0]

D2 = [0]

print(f"Output for Test Case 2: {fourSumCount(A2, B2, C2, D2)}")

1. 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.

arr = [12, 3, 5, 7, 19] k = 2 Expected Output:5

arr = [12, 3, 5, 7, 4, 19, 26] k = 3 Expected Output:5

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6 Expected Output:6

PROGRAM:

import statistics

def partition(arr, l, r, pivot\_idx):

pivot\_val = 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\_val:

arr[store\_idx], arr[i] = arr[i], arr[store\_idx]

store\_idx += 1

arr[r], arr[store\_idx] = arr[store\_idx], arr[r]

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

def median\_of\_medians(arr, k):

n = len(arr)

if n == 0:

return None

if n <= 5:

return sorted(arr)[k]

medians = [statistics.median(arr[i:i+5]) for i in range(0, n, 5)]

pivot = median\_of\_medians(medians, len(medians)//2)

pivot\_idx = arr.index(pivot)

left = [x for x in arr if x < pivot]

right = [x for x in arr if x > pivot]

if k < len(left):

return median\_of\_medians(left, k)

elif k >= n - len(right):

return median\_of\_medians(right, k - (n - len(right)))

else:

return pivot

# Test Cases

arr1 = [12, 3, 5, 7, 19]

k1 = 2

print(median\_of\_medians(arr1, k1)) # Expected Output: 5

arr2 = [12, 3, 5, 7, 4, 19, 26]

k2 = 3

print(median\_of\_medians(arr2, k2)) # Expected Output: 5

arr3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

k3 = 6

print(median\_of\_medians(arr3, k3)) # Expected Output: 6

1. 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.

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6

arr = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27] k = 5

Output: An integer representing the k-th smallest element in the array.

PROGRAM:

def median\_of\_medians(arr, k):

def partition(arr, l, r):

pivot = arr[r]

i = l

for j in range(l, r):

if arr[j] <= pivot:

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

i += 1

arr[i], arr[r] = arr[r], arr[i]

return i

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

if l == r:

return arr[l]

pivot\_index = partition(arr, l, r)

if k == pivot\_index:

return arr[k]

elif k < pivot\_index:

return select(arr, l, pivot\_index - 1, k)

else:

return select(arr, pivot\_index + 1, r, k)

return select(arr, 0, len(arr) - 1, k - 1)

# Test Cases

arr1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

k1 = 6

print(median\_of\_medians(arr1, k1)) # Output: 6

arr2 = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27]

k2 = 5

print(median\_of\_medians(arr2, k2)) # Output: 21

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.

a) Set[] = {45, 34, 4, 12, 5, 2} Target Sum : 42

b) Set[]= {1, 3, 2, 7, 4, 6} Target sum = 10:

PROGRAM:

from itertools import chain, combinations

def subsets(arr):

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

def closest\_subset\_sum(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]

right\_half\_sums.sort()

closest\_sum = float('inf')

closest\_subset = None

for sum\_l in left\_half\_sums:

idx = bisect\_left(right\_half\_sums, target - sum\_l)

if idx < len(right\_half\_sums):

sum\_r = right\_half\_sums[idx]

total\_sum = sum\_l + sum\_r

if abs(target - total\_sum) < abs(target - closest\_sum):

closest\_sum = total\_sum

closest\_subset = (sum\_l, sum\_r)

return closest\_subset

set1 = [45, 34, 4, 12, 5, 2]

target\_sum1 = 42

print("Closest Subset for Set 1:", closest\_subset\_sum(set1, target\_sum1))

set2 = [1, 3, 2, 7, 4, 6]

target\_sum2 = 10

print("Closest Subset for Set 2:", closest\_subset\_sum(set2, target\_sum2))

1. 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.

a) E = {1, 3, 9, 2, 7, 12} exact Sum = 15

b) E = {3, 34, 4, 12, 5, 2} exact Sum = 15

PROGRAM:

def is\_subset\_sum(arr, n, sum):

if sum == 0:

return True

if n == 0 and sum != 0:

return False

if arr[n-1] > sum:

return is\_subset\_sum(arr, n-1, sum)

return is\_subset\_sum(arr, n-1, sum) or is\_subset\_sum(arr, n-1, sum-arr[n-1])

def meet\_in\_the\_middle(arr, exact\_sum):

n = len(arr)

if n == 0:

return False

mid = n // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

left\_sums = []

right\_sums = []

for i in range(1 << len(left\_half)):

total = sum(left\_half[j] for j in range(len(left\_half)) if i & (1 << j))

left\_sums.append(total)

for i in range(1 << len(right\_half)):

total = sum(right\_half[j] for j in range(len(right\_half)) if i & (1 << j))

right\_sums.append(total)

left\_sums.sort()

right\_sums.sort()

for sum\_l in left\_sums:

if sum\_l == exact\_sum:

return True

if exact\_sum - sum\_l in right\_sums:

return True

return False

# Test Cases

arr\_a = [1, 3, 9, 2, 7, 12]

exact\_sum\_a = 15

print(meet\_in\_the\_middle(arr\_a, exact\_sum\_a)) # Output: True

arr\_b = [3, 34, 4, 12, 5, 2]

exact\_sum\_b = 15

print(meet\_in\_the\_middle(arr\_b, exact\_sum\_b)) # Output: False

1. Given two 2×2 Matrices A and B

A=(1 7​ B=( 1 3

3 5​) 7 5)

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

**Test Cases:**

Consider the following matrices for testing your implementation:

**Test Case 1:**

A=(1 7 B=( 6 8

​3 5​), 4 2)

Expected Output:

C=(18 14

62 66)

PROGRAM:

def strassen\_matrix\_multiply(A, B):

if len(A) == 2:

a, b, c, d = A[0][0], A[0][1], A[1][0], A[1][1]

e, f, g, h = B[0][0], B[0][1], B[1][0], B[1][1]

p1 = a \* (f - h)

p2 = (a + b) \* h

p3 = (c + d) \* e

p4 = d \* (g - e)

p5 = (a + d) \* (e + h)

p6 = (b - d) \* (g + h)

p7 = (a - c) \* (e + f)

C = [[p5 + p4 - p2 + p6, p1 + p2], [p3 + p4, p1 + p5 - p3 - p7]]

return C

else:

return "Input matrices are not 2x2."

# Test Case

A = [[1, 7], [3, 5]]

B = [[6, 8], [4, 2]]

C = strassen\_matrix\_multiply(A, B)

print(C)

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

**Test Case 1:**

Input: x=1234,y=5678

Expected Output: z=1234×5678=7016652

PROGRAM:

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 of {x} x {y} using Karatsuba Algorithm: {z}")