**TOPIC 3 : DIVIDE AND CONQUER**

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

1. 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

1. 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

def mergeSort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

leftHalf = arr[:mid]

rightHalf = arr[mid:]

sortedLeft = mergeSort(leftHalf)

sortedRight = mergeSort(rightHalf)

return merge(sortedLeft, sortedRight)

def merge(left, right):

result = []

i = j = 0

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

if left[i] < right[j]:

result.append(left[i])

i += 1

else:

result.append(right[j])

j += 1

result.extend(left[i:])

result.extend(right[j:])

return result

unsortedArr = [3, 7, 6, -10, 15, 23.5, 55, -13]

sortedArr = mergeSort(unsortedArr)

print("Sorted array:", sortedArr)

1. 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,

def mergeSort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

leftHalf = arr[:mid]

rightHalf = arr[mid:]

sortedLeft = mergeSort(leftHalf)

sortedRight = mergeSort(rightHalf)

return merge(sortedLeft, sortedRight)

def merge(left, right):

result = []

i = j = 0

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

if left[i] < right[j]:

result.append(left[i])

i += 1

else:

result.append(right[j])

j += 1

result.extend(left[i:])

result.extend(right[j:])

return result

unsortedArr = [3, 7, 6, -10, 15, 23.5, 55, -13]

sortedArr = mergeSort(unsortedArr)

print("Sorted array:", sortedArr)

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,

def quick(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(left)+middle+quick(right)

a=[9,2,84,5756,5]

c=quick(a)

print(c)

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

def quick(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(left)+middle+quick(right)

a=[9,2,84,5756,5]

c=quick(a)

print(c)

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

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

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]]

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

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

def partition(arr, low, high, pivot):

pivot\_index = arr.index(pivot)

arr[pivot\_index], arr[high] = arr[high], arr[pivot\_index]

pivot\_value = arr[high]

store\_index = low

for i in range(low, high):

if arr[i] < pivot\_value:

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

store\_index += 1

arr[store\_index], arr[high] = arr[high], arr[store\_index]

return store\_index

def select(arr, low, high, k):

if low == high:

return arr[low]

while low <= high:

if high - low + 1 <= 5:

arr[low:high + 1] = sorted(arr[low:high + 1])

return arr[low + k]

medians = []

for i in range(low, high + 1, 5):

group = arr[i:i + 5]

group.sort()

median = group[len(group) // 2]

medians.append(median)

median\_of\_medians = select(medians, 0, len(medians) - 1, len(medians) // 2)

pivot\_index = partition(arr, low, high, median\_of\_medians)

if k == pivot\_index:

return arr[k]

elif k < pivot\_index:

high = pivot\_index - 1

else:

low = pivot\_index + 1

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

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

# Test the function with the given test cases

test\_cases = [

([12, 3, 5, 7, 19], 2, 5),

([12, 3, 5, 7, 4, 19, 26], 3, 5),

([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 6, 6)

]

results = []

for arr, k, expected in test\_cases:

result = median\_of\_medians(arr, k)

results.append((result, result == expected))

results

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.

1. 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:

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

15 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)

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

    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]

    M1 = (a + d) \* (e + h)

    M2 = (c + d) \* e

    M3 = a \* (f - h)

    M4 = d \* (g - e)

    M5 = (a + b) \* h

    M6 = (c - a) \* (e + f)

    M7 = (b - d) \* (g + h)

    C11 = M1 + M4 - M5 + M7

    C12 = M3 + M5

    C21 = M2 + M4

    C22 = M1 - M2 + M3 + M6

    return [[C11, C12], [C21, C22]]

# Test case 1

A1 = [[1, 7],

      [3, 5]]

B1 = [[6, 8],

      [4, 2]]

expected\_output1 = [[18, 14],

                    [62, 66]]

# Compute the product

output1 = strassen\_matrix\_multiplication(A1, B1)

print(output1)

16 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

def karatsuba(x, y):

    # Base case for recursion

    if x < 10 or y < 10:

        return x \* y

    # Calculate the size of the numbers

    n = max(len(str(x)), len(str(y)))

    m = n // 2

    # Split the numbers into two halves

    high1, low1 = divmod(x, 10\*\*m)

    high2, low2 = divmod(y, 10\*\*m)

    # Perform the three recursive multiplications

    z0 = karatsuba(low1, low2)

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

    z2 = karatsuba(high1, high2)

    # Combine the results

    return (z2 \* 10\*\*(2\*m)) + ((z1 - z2 - z0) \* 10\*\*m) + z0

# Test case 1

x = 1234

y = 5678

expected\_output = 1234 \* 5678

# Compute the product using Karatsuba algorithm

output = karatsuba(x, y)

print(output)