DAY-5

**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**

**CODE:**

def find\_min\_max(arr):

# Finding minimum and maximum values

min\_val = min(arr)

max\_val = max(arr)

return min\_val, max\_val

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

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

print(f"Input: {arr1}\nMin = {min\_val}, Max = {max\_val}")

**OUTPUT:**

Input: [5, 7, 3, 4, 9, 12, 6, 2]

Min = 2, Max = 12

**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**

**CODE:**

def find\_min\_max(arr):

min\_val = arr[0]

max\_val = arr[-1]

return min\_val, max\_val

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

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

print(f"Input: {arr}")

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

**OUTPUT:**

Input: [2, 4, 6, 8, 10, 12, 14, 18]

Min = 2, Max = 18

**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**

**CODE:**

def merge(left, right):

sorted\_array = []

i = j = 0

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

if left[i] < right[j]:

sorted\_array.append(left[i])

i += 1

else:

sorted\_array.append(right[j])

j += 1

sorted\_array.extend(left[i:])

sorted\_array.extend(right[j:])

return sorted\_array

def merge\_sort(arr):

# Base case: single element or empty array

if len(arr) <= 1:

return arr

mid = len(arr) // 2

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

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

return merge(left\_half, right\_half)

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

sorted\_arr = merge\_sort(arr)

**OUTPUT:**

Sorted Array: [11, 15, 21, 23, 27, 28, 31, 35]

**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**

**CODE:**

comparison\_count = 0

def merge(left, right):

global comparison\_count

sorted\_array = []

i = j = 0

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

comparison\_count += 1 # Count comparison

if left[i] < right[j]:

sorted\_array.append(left[i])

i += 1

else:

sorted\_array.append(right[j])

j += 1

sorted\_array.extend(left[i:])

sorted\_array.extend(right[j:])

return sorted\_array

def merge\_sort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

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

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

return merge(left\_half, right\_half)

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

sorted\_arr = merge\_sort(arr)

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

print("Number of Comparisons:", comparison\_count)

**OUTPUT:**

Sorted Array: [1, 4, 12, 23, 45, 67, 78, 89]

Number of Comparisons: [comparison count based on the input array]

**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**

**CODE:**

def partition(arr, low, high):

pivot = arr[low] # First element as pivot

left = low + 1

right = high

done = False

while not done:

while left <= right and arr[left] <= pivot:

left = left + 1

while arr[right] >= pivot and right >= left:

right = right - 1

if right < left:

done = True

else:

arr[left], arr[right] = arr[right], arr[left]

arr[low], arr[right] = arr[right], arr[low]

return right

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

if low < high:

pivot\_index = partition(arr, low, high)

print(f"Array after partition (pivot {arr[pivot\_index]}): {arr}")

quick\_sort(arr, low, pivot

**OUTPUT:**

Initial Array: [10, 16, 8, 12, 15, 6, 3, 9, 5]

Array after partition (pivot 9): [5, 3, 8, 6, 9, 12, 15, 16, 10]

Array after partition (pivot 6): [5, 3, 6, 8, 9, 12, 15, 16, 10]

Array after partition (pivot 3): [3, 5, 6, 8, 9, 12, 15, 16, 10]

Array after partition (pivot 10): [3, 5, 6, 8, 9, 10, 12, 16, 15]

Array after partition (pivot 15): [3, 5, 6, 8, 9, 10, 12, 15, 16]

Final Sorted Array: [3, 5, 6, 8, 9, 10, 12, 15, 16]

**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**

**CODE:**

def partition(arr, low, high):

mid = (low + high) // 2 # Middle element as pivot

pivot = arr[mid]

arr[mid], arr[low] = arr[low], arr[mid]

left = low + 1

right = high

done = False

while not done:

while left <= right and arr[left] <= pivot:

left += 1

while arr[right] >= pivot and right >= left:

right -= 1

if right < left:

done = True

else:

# Swap left and right values

arr[left], arr[right] = arr[right], arr[left]

arr[low], arr[right] = arr[right], arr[low]

return right

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

if low < high:

# Partition the array and get the partition index

pivot\_index = partition(arr, low, high)

print(f"Array after partition (pivot {arr[pivot\_index]}): {arr}")

quick\_sort(arr, low, pivot\_index - 1)

quick\_sort(arr, pivot\_index + 1, high)

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

N = len(arr)

print("Initial Array:", arr)

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

print("Final Sorted Array:", arr)

**OUTPUT:**

Initial Array: [19, 72, 35, 46, 58, 91, 22, 31]

Array after partition (pivot 46): [31, 22, 35, 19, 46, 91, 72, 58]

Array after partition (pivot 22): [19, 22, 35, 31, 46, 91, 72, 58]

Array after partition (pivot 31): [19, 22, 31, 35, 46, 91, 72, 58]

Array after partition (pivot 72): [19, 22, 31, 35, 46, 58, 72, 91]

Final Sorted Array: [19, 22, 31, 35, 46, 58, 72, 91]

**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**

**CODE:**

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

comparisons = 0 # Counter for the number of comparisons

while low <= high:

comparisons += 1

mid = (low + high)

if arr[mid] == key:

print(f"Element {key} found at index {mid}")

print(f"Total Comparisons: {comparisons}")

return mid elif arr[mid] < key:

low = mid + 1

else:

high = mid - 1

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

print(f"Total Comparisons: {comparisons}")

return

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

N = len(arr)

key = 20

index = binary\_search(arr, 0, N - 1, key)

**OUTPUT:**

Element 20 found at index 3

Total Comparisons: 2

**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**

**CODE:**

def binary\_search(arr, key):

low = 0

high = len(arr) - 1

comparisons = 0

while low <= high:

comparisons += 1

mid = (low + high) // 2

print(f"Checking mid-point at index {mid}: {arr[mid]}") #

if arr[mid] == key:

print(f"Element found at index {mid}")

return mid, comparisons

elif arr[mid] < key:

low = mid + 1

else:

high = mid - 1

return -1, comparisons

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

search\_key = 31

result, comparison\_count = binary\_search(arr, search\_key)

if result != -1:

print(f"Element {search\_key} found at index {result}")

else:

print(f"Element {search\_key} not found in the array")

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

**OUTPUT:**

Checking mid-point at index 4: 25

Checking mid-point at index 6: 42

Checking mid-point at index 5: 31

Element 31 found at index 5

Number of comparisons made: 3

**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]]**

**CODE:**

import heapq

def k\_closest\_points(points, k):

heap = []

for point in points:

x, y = point

distance = x\*\*2 + y\*\*2 # Calculate squared distance

heapq.heappush(heap, (distance, point))

result = [heapq.heappop(heap)[1] for \_ in range(k)]

return result

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

k = 2

output = k\_closest\_points(points, k)

print("The closest points are:", output)

**OUTPUT:**

The closest points are: [[-2, 2], [0, 1]]

**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**

**CODE:**

from collections import defaultdict

def four\_sum\_count(A, B, C, D):

AB\_sum\_map = defaultdict(int)

for a in A:

for b in B:

AB\_sum\_map[a + b] += 1

count = 0

for c in C:

for d in D:

target = -(c + d)

if target in AB\_sum\_map:

count += AB\_sum\_map[target]

return count

A = [1, 2]

B = [-2, -1]

C = [-1, 2]

D = [0, 2]

output = four\_sum\_count(A, B, C, D)

print("The number of tuples is:", output)

**OUTPUT:**

{

-1: 1,

0: 2,

1: 1

}