DAY-5 PROGRAMS

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

Code:

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

min\_val1 = arr1[0]

max\_val1 = arr1[0]

for num in arr1:

if num < min\_val1:

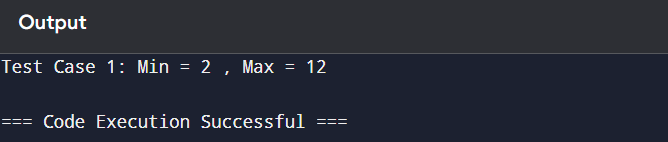
min\_val1 = num

if num > max\_val1:

max\_val1 = num

print("Test Case 1: Min =", min\_val1, ", Max =", max\_val1)

output:



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

Code:

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

min\_val1 = arr1[0]

max\_val1 = arr1[0]

for num in arr1:

if num < min\_val1:

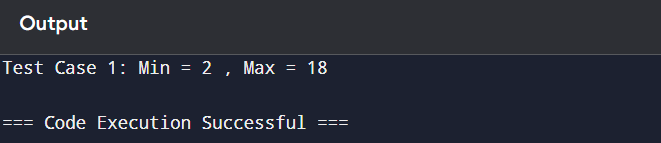
min\_val1 = num

if num > max\_val1:

max\_val1 = num

print("Test Case 1: Min =", min\_val1, ", Max =", max\_val1)

output:



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

Code:

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

merge\_sort(left\_half)

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

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

arr[k] = right\_half[j]

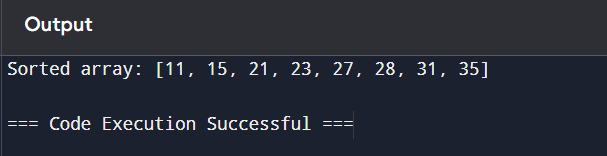
j += 1

k += 1

merge\_sort(arr)

print("Sorted array:", arr)

output:



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

Code:

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

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

merge\_sort(left\_half)

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

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

arr[k] = right\_half[j]

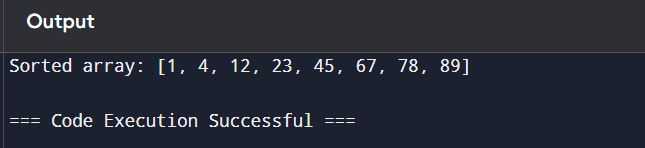
j += 1

k += 1

merge\_sort(arr)

print("Sorted array:", arr)

output:



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

Code:

def partition(arr, low, high):

pivot = arr[low]

left = low + 1

for right in range(low + 1, high + 1):

if arr[right] < pivot:

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

left += 1

arr[low], arr[left - 1] = arr[left - 1], arr[low]

return left - 1

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

if low < high:

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

print("After partitioning:", arr)

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

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

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

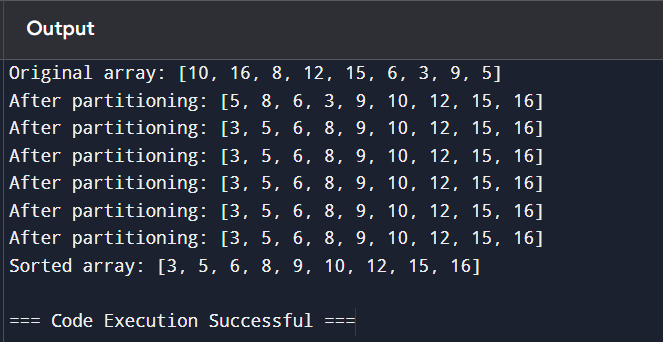
n = len(arr)

print("Original array:", arr)

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

print("Sorted array:", arr)

output:



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):

pivot = arr[low]

left = low + 1

for right in range(low + 1, high + 1):

if arr[right] < pivot:

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

left += 1

arr[low], arr[left - 1] = arr[left - 1], arr[low]

return left - 1

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

if low < high:

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

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

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

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

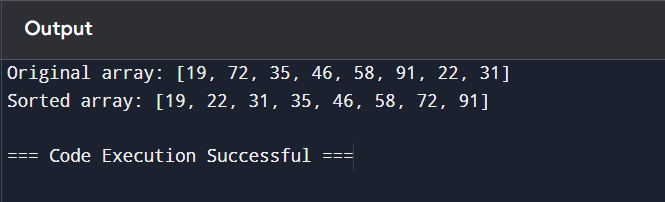
n = len(arr)

print("Original array:", arr)

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

print("Sorted array:", arr)

output:



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, key):

low = 0

high = len(arr) - 1

comparisons = 0

while low <= high:

mid = (low + high) // 2

comparisons += 1

if arr[mid] == key:

return mid, comparisons

elif arr[mid] < key:

low = mid + 1

else:

high = mid - 1

return -1, comparisons

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

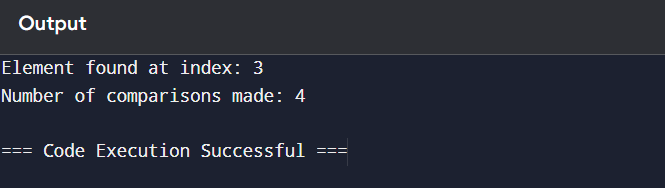
search\_key = 20

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

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

print("Number of comparisons made:", comparison\_count)

output:



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:

mid = (low + high) // 2

comparisons += 1

if arr[mid] == key:

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]

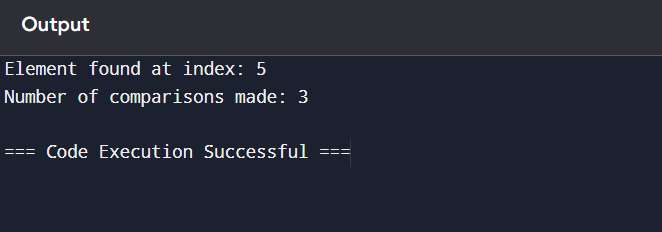
search\_key = 31

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

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

print("Number of comparisons made:", comparison\_count)

output:



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).

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

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

Code:

import heapq

def k\_closest(points, k):

distances = [(x\*2 + y\*2, (x, y)) for x, y in points]

closest\_points = heapq.nsmallest(k, distances)

return [point for \_, point in closest\_points]

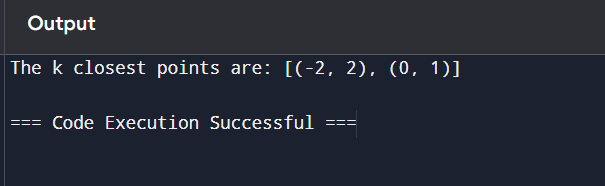
points1 = [[1, 3], [-2, 2], [5, 8], [0, 1]]

k1 = 2

result1 = k\_closest(points1, k1)

print("The k closest points are:", result1)

output:



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.

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

Output: 2

Code:

from collections import defaultdict

def count\_zero\_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:

target\_sum = -(c + d)

if target\_sum in sum\_ab:

count += sum\_ab[target\_sum]

return count

A = [1, 2]

B = [-2, -1]

C = [-1, 2]

D = [0, 2]

result = count\_zero\_tuples(A, B, C, D)

print("Output:", result)

output:

