TOPIC 3 : DIVIDE AND CONQUER

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 :

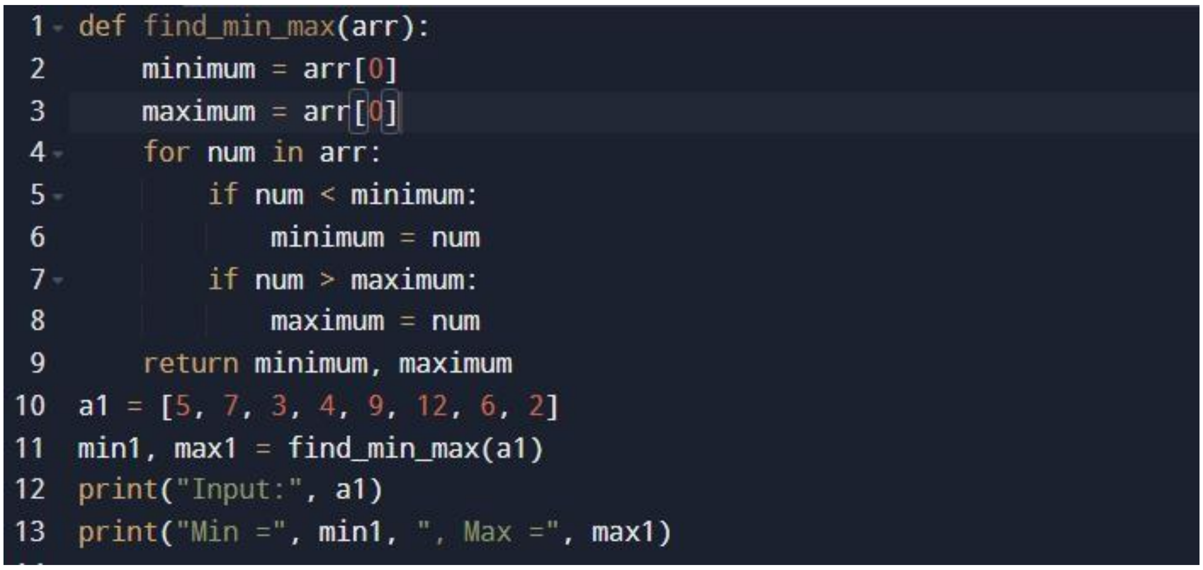
Input : N= 9, a[] = {1,3,5,7,9,11,13,15,17}

Output : Min = 1, Max = 17 Test

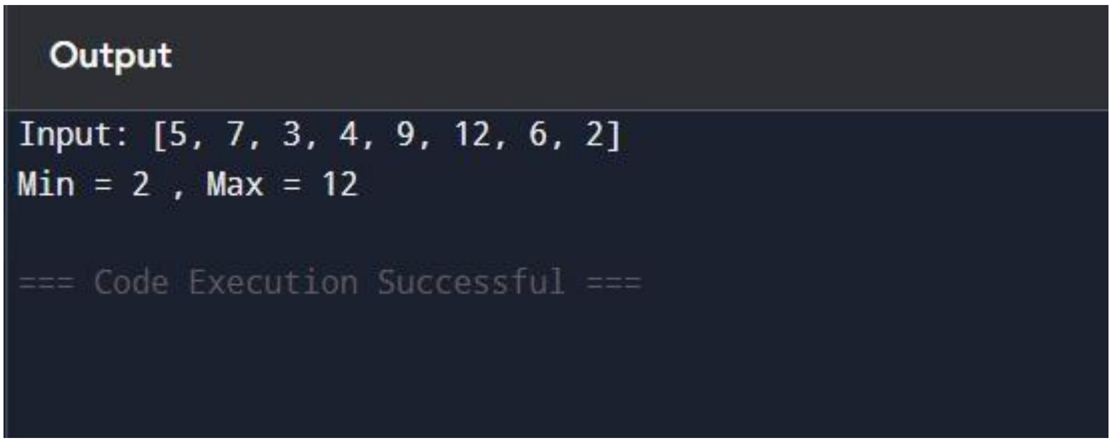
Cases :

Input : N= 10, a[] = {22,34,35,36,43,67, 12,13,15,17}

Output : Min 12, Max 67



```python
def find_min_max(arr):
    minimum = arr[0]
    maximum = arr[0]
    for num in arr:
        if num < minimum:
            minimum = num
        if num > maximum:
            maximum = num
    return minimum, maximum
a1 = [5, 7, 3, 4, 9, 12, 6, 2]
min1, max1 = find_min_max(a1)
print("Input:", a1)
print("Min =", min1, ", Max =", max1)
```


Output

```
Input: [5, 7, 3, 4, 9, 12, 6, 2]
Min = 2 , Max = 12

=== Code Execution Successful ===
```

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 :

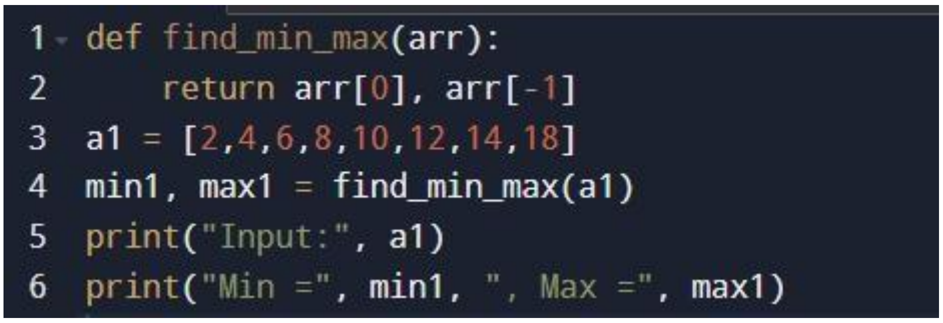
Input : N= 9, a[] = {11,13,15,17,19,21,23,35,37}

Output : Min = 11, Max = 37 Test

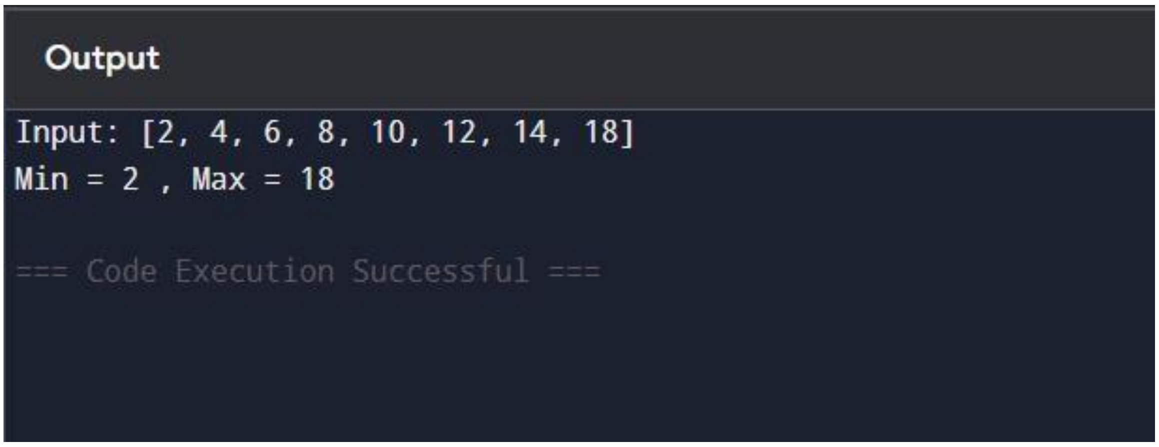
Cases :

Input : N= 10, a[] = {22,34,35,36,43,67, 12,13,15,17}

Output : Min 12, Max 67



```
def find_min_max(arr):
    return arr[0], arr[-1]
a1 = [2,4,6,8,10,12,14,18]
min1, max1 = find_min_max(a1)
print("Input:", a1)
print("Min =", min1, ", Max =", max1)
```



Output

```
Input: [2, 4, 6, 8, 10, 12, 14, 18]
Min = 2 , Max = 18

=== Code Execution Successful ===
```

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.

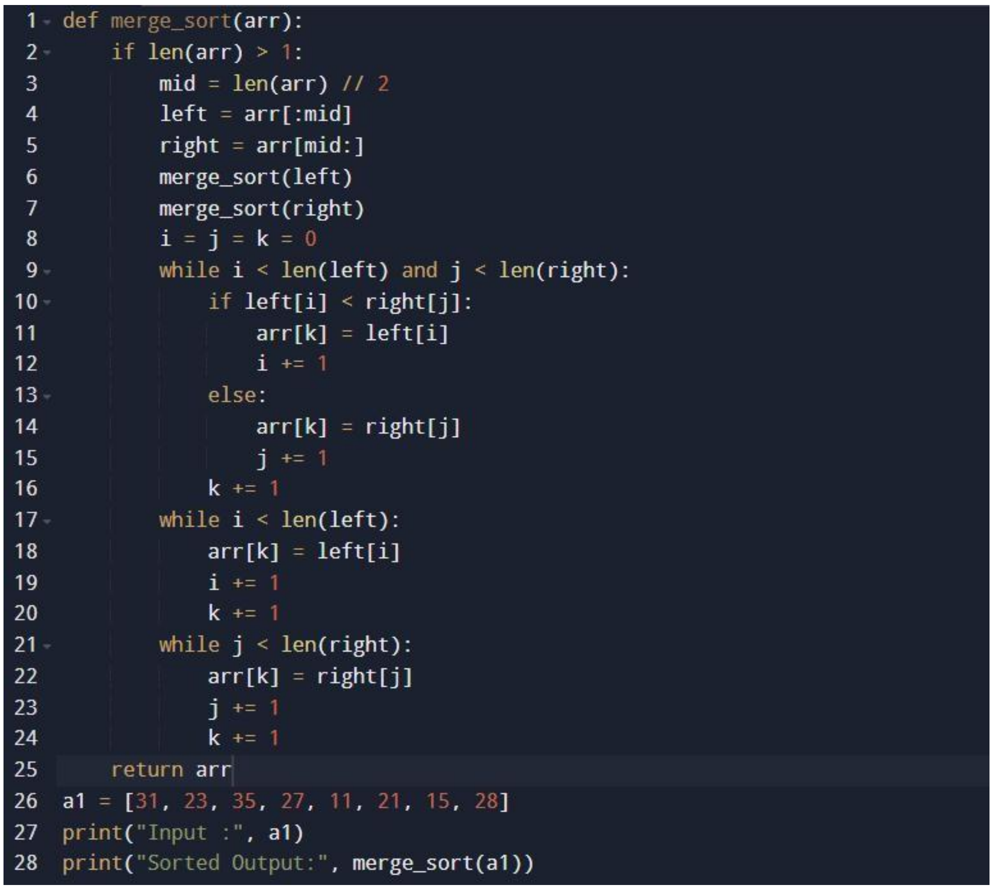
Test Cases :

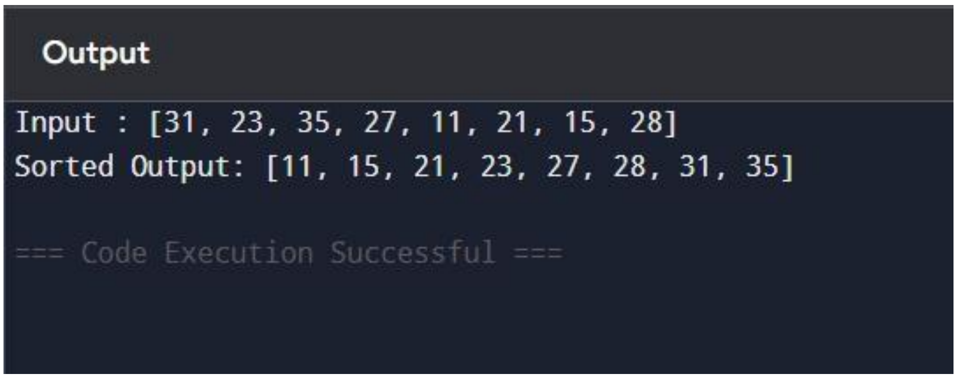
Input : N= 8, a[] = {31,23,35,27,11,21,15,28} Output : 11,15,21,23,27,28,31,35

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


```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]
        merge_sort(left)
        merge_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                arr[k] = left[i]
                i += 1
            else:
                arr[k] = right[j]
                j += 1
            k += 1
        while i < len(left):
            arr[k] = left[i]
            i += 1
            k += 1
        while j < len(right):
            arr[k] = right[j]
            j += 1
            k += 1
    return arr
a1 = [31, 23, 35, 27, 11, 21, 15, 28]
print("Input :", a1)
print("Sorted Output:", merge_sort(a1))
```


Output

```
Input : [31, 23, 35, 27, 11, 21, 15, 28]
Sorted Output: [11, 15, 21, 23, 27, 28, 31, 35]

=== Code Execution Successful ===
```

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 :

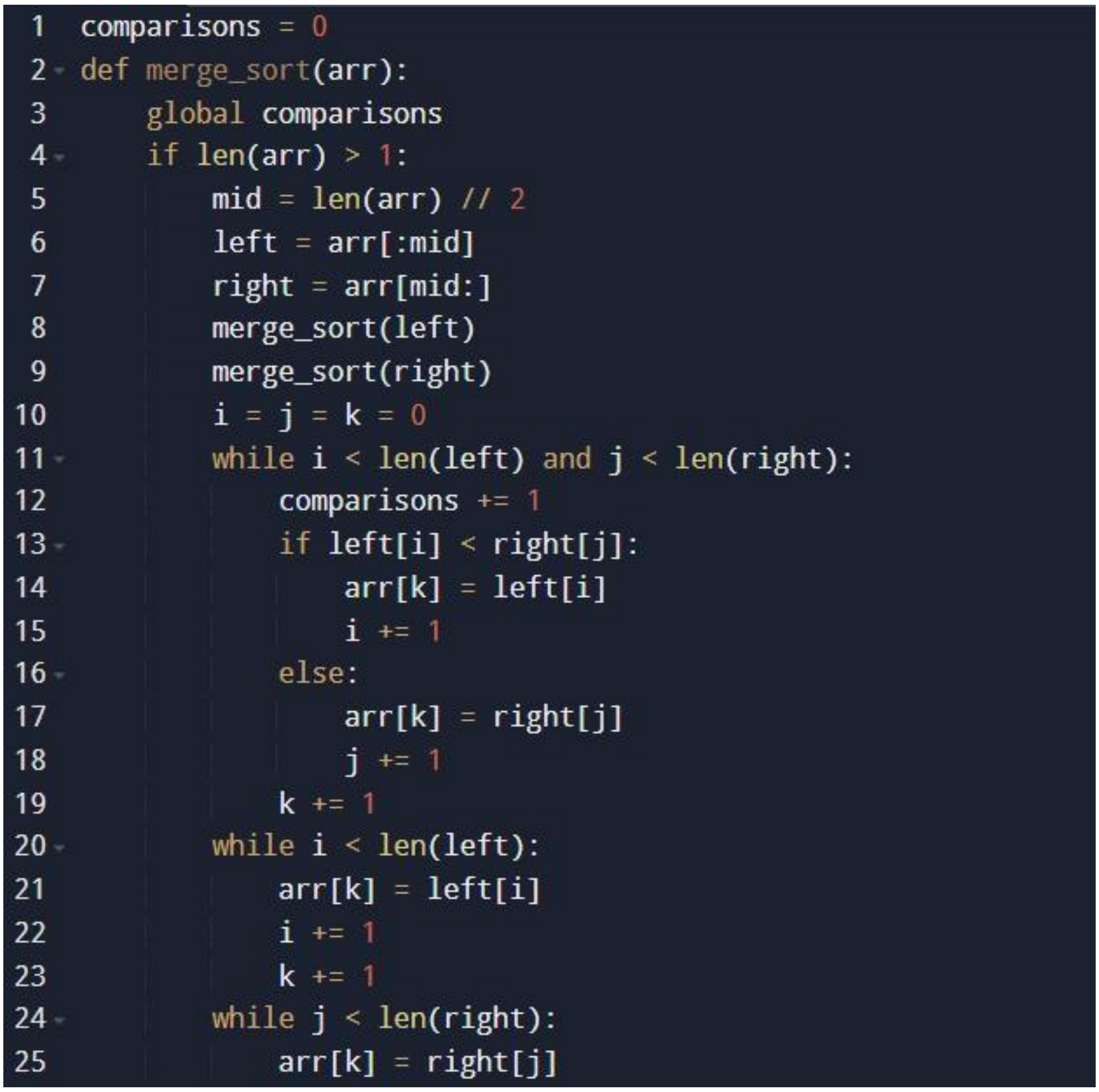
Input : N= 8, a[] = {12,4,78,23,45,67,89,1}

Output : 1,4,12,23,45,67,78,89

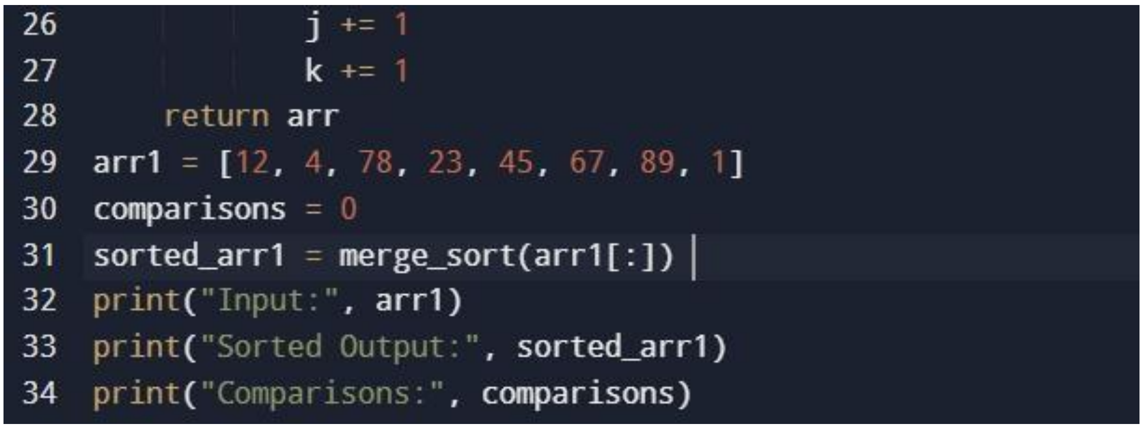
Test Cases :

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

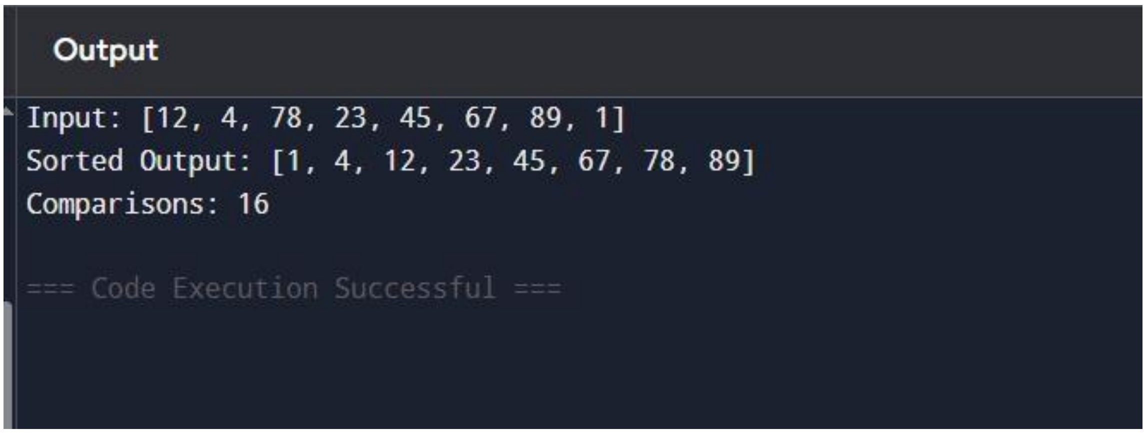
3,9,10,27,38,43,82,



```python
comparisons = 0
def merge_sort(arr):
    global comparisons
    if len(arr) > 1:
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]
        merge_sort(left)
        merge_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            comparisons += 1
            if left[i] < right[j]:
                arr[k] = left[i]
                i += 1
            else:
                arr[k] = right[j]
                j += 1
            k += 1
        while i < len(left):
            arr[k] = left[i]
            i += 1
            k += 1
        while j < len(right):
            arr[k] = right[j]
```



```python
            j += 1
            k += 1
    return arr
arr1 = [12, 4, 78, 23, 45, 67, 89, 1]
comparisons = 0
sorted_arr1 = merge_sort(arr1[:])
print("Input:", arr1)
print("Sorted Output:", sorted_arr1)
print("Comparisons:", comparisons)
```



```
Input: [12, 4, 78, 23, 45, 67, 89, 1]
Sorted Output: [1, 4, 12, 23, 45, 67, 78, 89]
Comparisons: 16

=== Code Execution Successful ===
```

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}

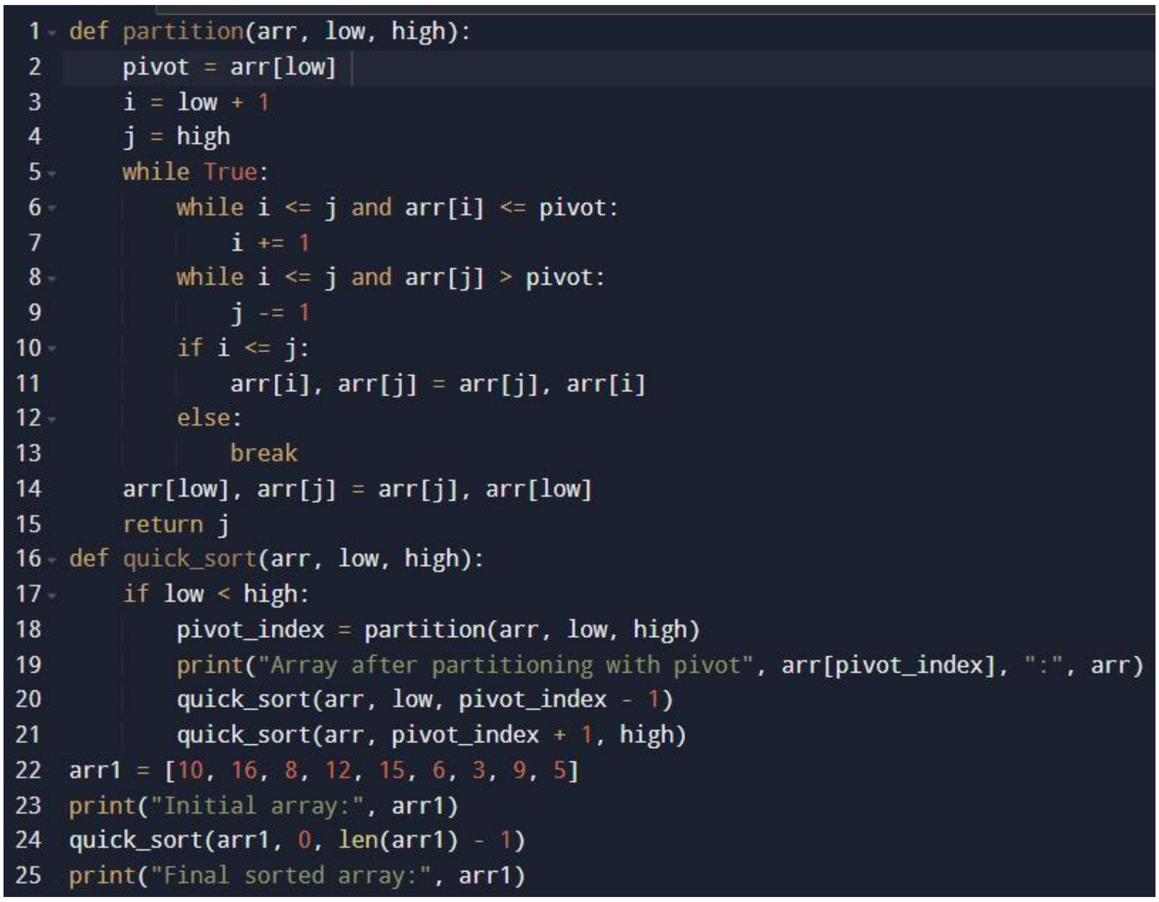
Output : 3,5,6,8,9,10,12,15,16

Test Cases :

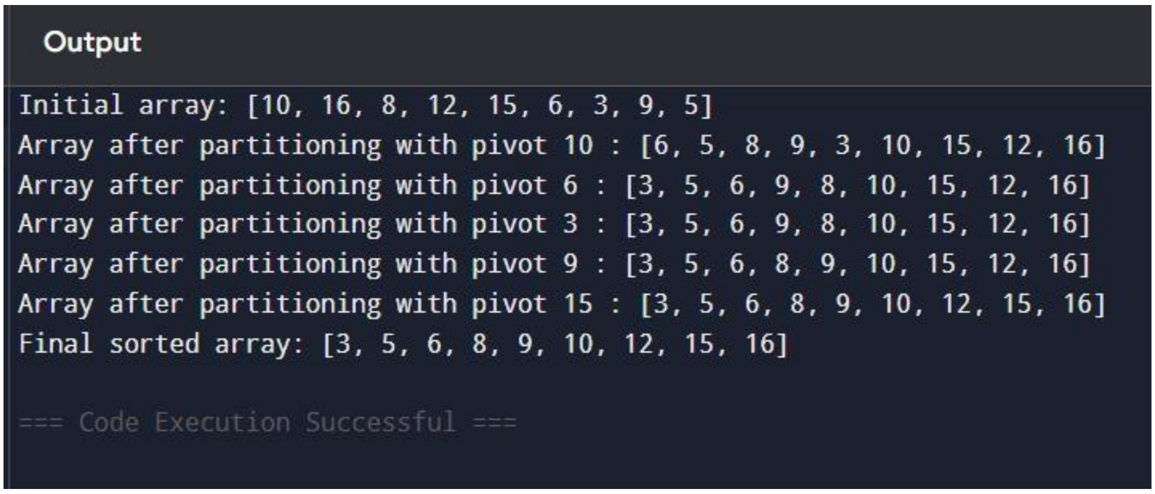
Input : N= 8, a[] = {12,4,78,23,45,67,89,1} Output :
1,4,12,23,45,67,78,89

Test Cases :

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



```python
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("Array after partitioning with pivot", arr[pivot_index], ":", arr)
        quick_sort(arr, low, pivot_index - 1)
        quick_sort(arr, pivot_index + 1, high)
arr1 = [10, 16, 8, 12, 15, 6, 3, 9, 5]
print("Initial array:", arr1)
quick_sort(arr1, 0, len(arr1) - 1)
print("Final sorted array:", arr1)
```



Output

```
Initial array: [10, 16, 8, 12, 15, 6, 3, 9, 5]
Array after partitioning with pivot 10 : [6, 5, 8, 9, 3, 10, 15, 12, 16]
Array after partitioning with pivot 6 : [3, 5, 6, 9, 8, 10, 15, 12, 16]
Array after partitioning with pivot 3 : [3, 5, 6, 9, 8, 10, 15, 12, 16]
Array after partitioning with pivot 9 : [3, 5, 6, 8, 9, 10, 15, 12, 16]
Array after partitioning with pivot 15 : [3, 5, 6, 8, 9, 10, 12, 15, 16]
Final sorted array: [3, 5, 6, 8, 9, 10, 12, 15, 16]


=== Code Execution Successful ===
```

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

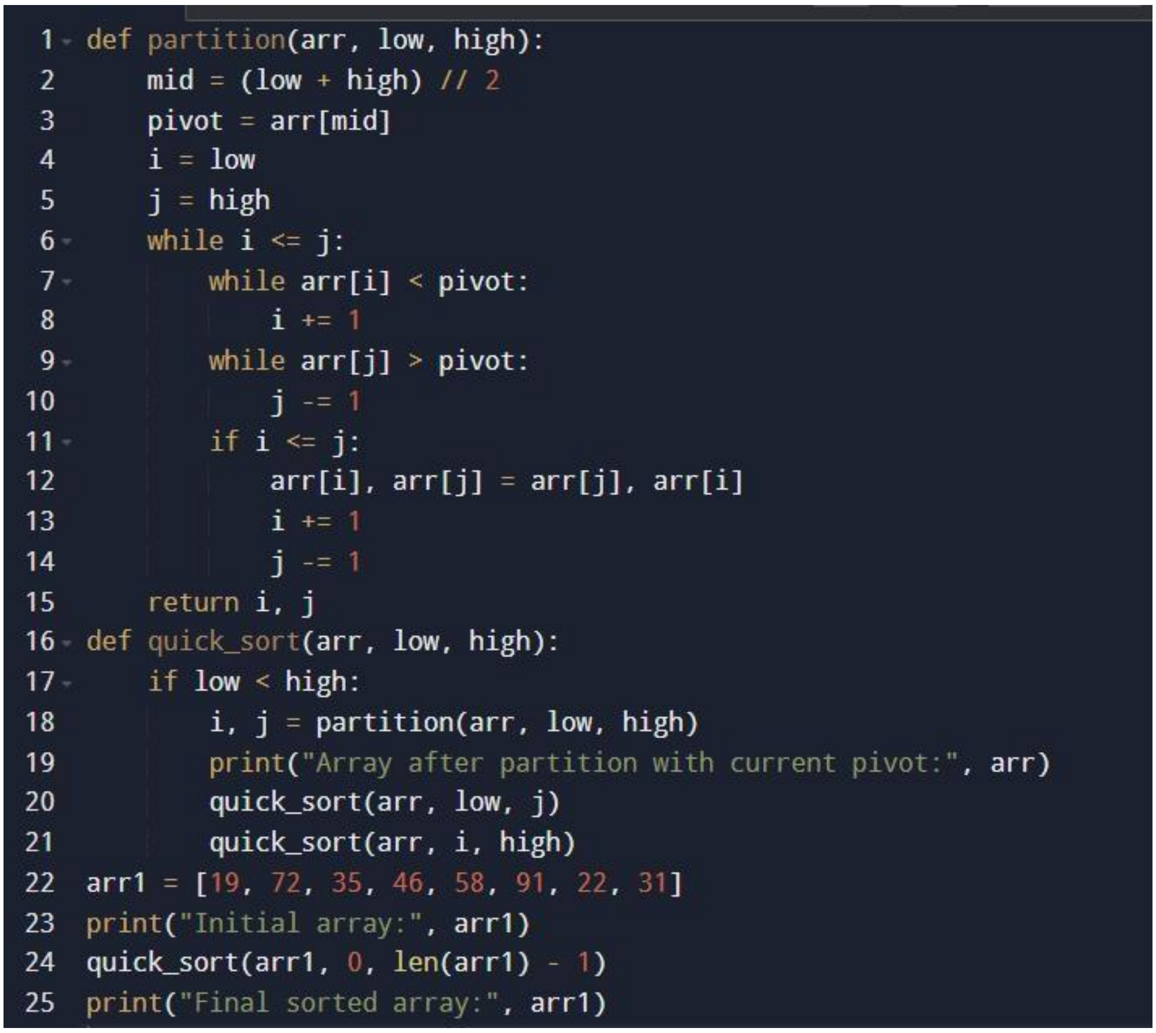
Test Cases :

Input : N= 8, a[] = {31,23,35,27,11,21,15,28} Output
: 11,15,21,23,27,28,31,35

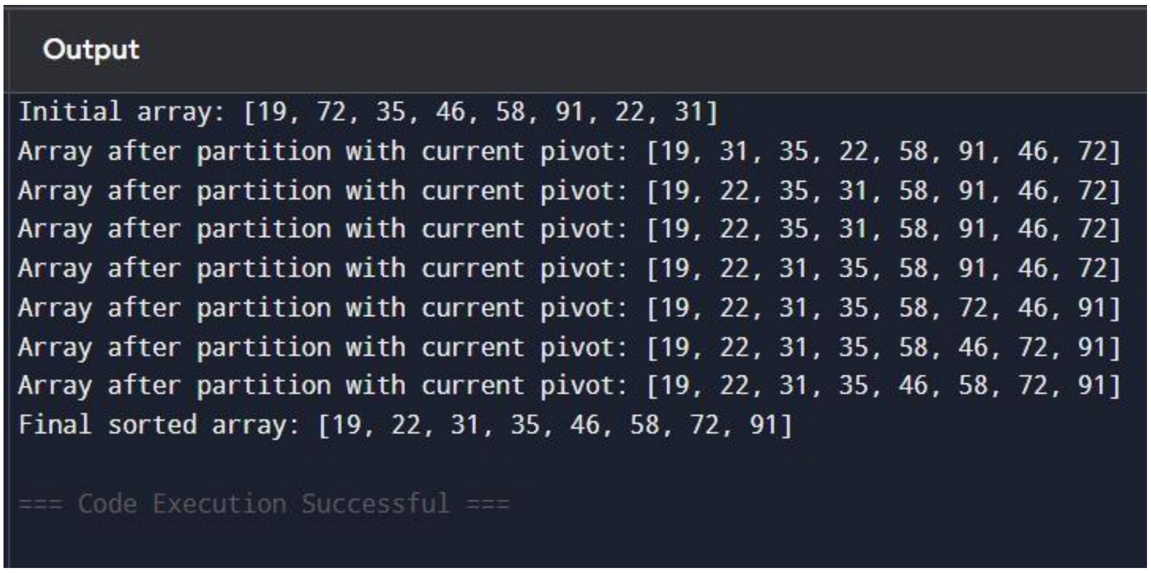
Test Cases :

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



```python
def partition(arr, low, high):
    mid = (low + high) // 2
    pivot = arr[mid]
    i = low
    j = high
    while i <= j:
        while arr[i] < pivot:
            i += 1
        while arr[j] > pivot:
            j -= 1
        if i <= j:
            arr[i], arr[j] = arr[j], arr[i]
            i += 1
            j -= 1
    return i, j
def quick_sort(arr, low, high):
    if low < high:
        i, j = partition(arr, low, high)
        print("Array after partition with current pivot:", arr)
        quick_sort(arr, low, j)
        quick_sort(arr, i, high)
arr1 = [19, 72, 35, 46, 58, 91, 22, 31]
print("Initial array:", arr1)
quick_sort(arr1, 0, len(arr1) - 1)
print("Final sorted array:", arr1)
```



Output

```
Initial array: [19, 72, 35, 46, 58, 91, 22, 31]
Array after partition with current pivot: [19, 31, 35, 22, 58, 91, 46, 72]
Array after partition with current pivot: [19, 22, 35, 31, 58, 91, 46, 72]
Array after partition with current pivot: [19, 22, 35, 31, 58, 91, 46, 72]
Array after partition with current pivot: [19, 22, 31, 35, 58, 91, 46, 72]
Array after partition with current pivot: [19, 22, 31, 35, 58, 72, 46, 91]
Array after partition with current pivot: [19, 22, 31, 35, 58, 46, 72, 91]
Array after partition with current pivot: [19, 22, 31, 35, 46, 58, 72, 91]
Final sorted array: [19, 22, 31, 35, 46, 58, 72, 91]

=== Code Execution Successful ===
```

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

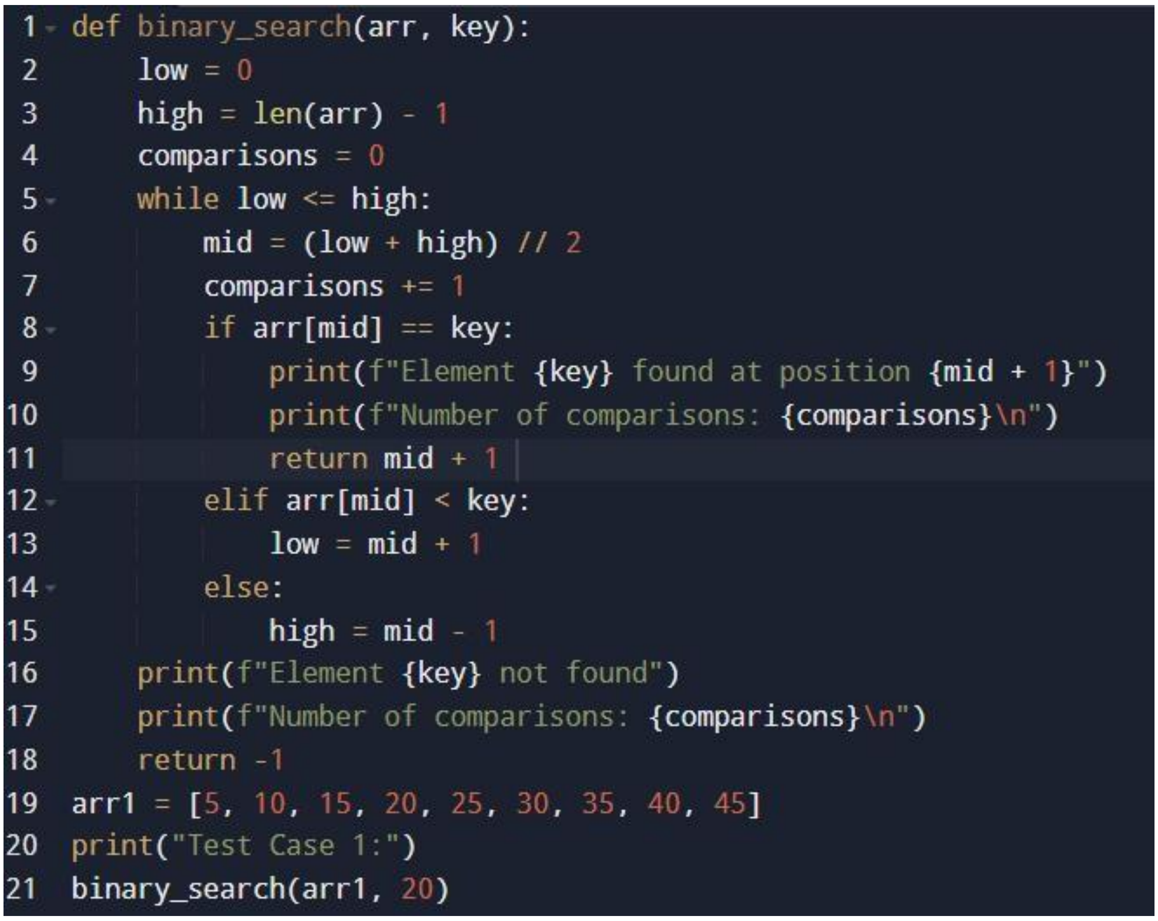
Output : 4 Test

cases

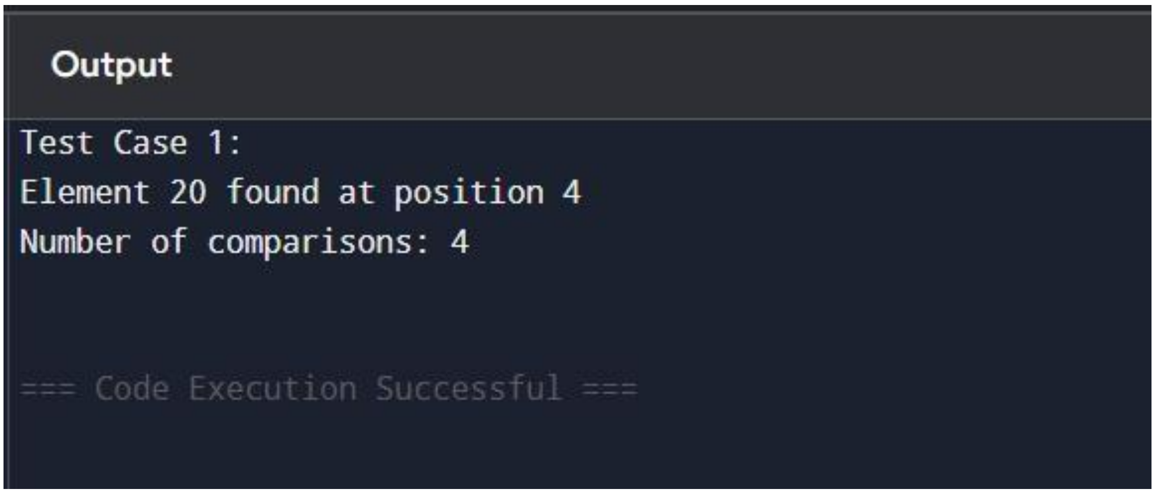
Input : N= 6, a[] = {10,20,30,40,50,60}, search key = 50 Output : 5

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

Output : 2



```python
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:
            print(f"Element {key} found at position {mid + 1}")
            print(f"Number of comparisons: {comparisons}\n")
            return mid + 1
        elif arr[mid] < key:
            low = mid + 1
        else:
            high = mid - 1
    print(f"Element {key} not found")
    print(f"Number of comparisons: {comparisons}\n")
    return -1
arr1 = [5, 10, 15, 20, 25, 30, 35, 40, 45]
print("Test Case 1:")
binary_search(arr1, 20)
```


Output

```
Test Case 1:
Element 20 found at position 4
Number of comparisons: 4


=== Code Execution Successful ===
```

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 Test

cases

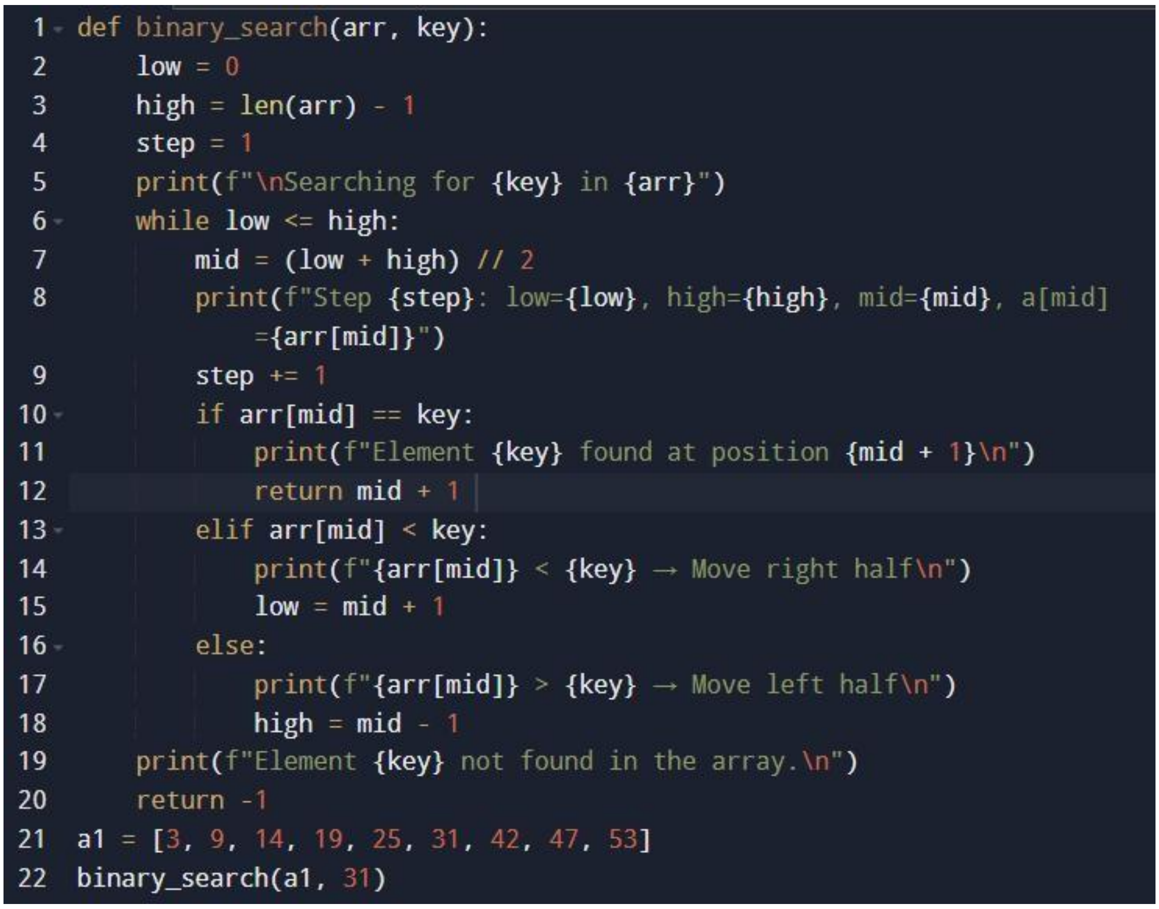
Input : N= 7, a[] = {13,19,24,29,35,41,42}, search key = 42

Output : 7 Test

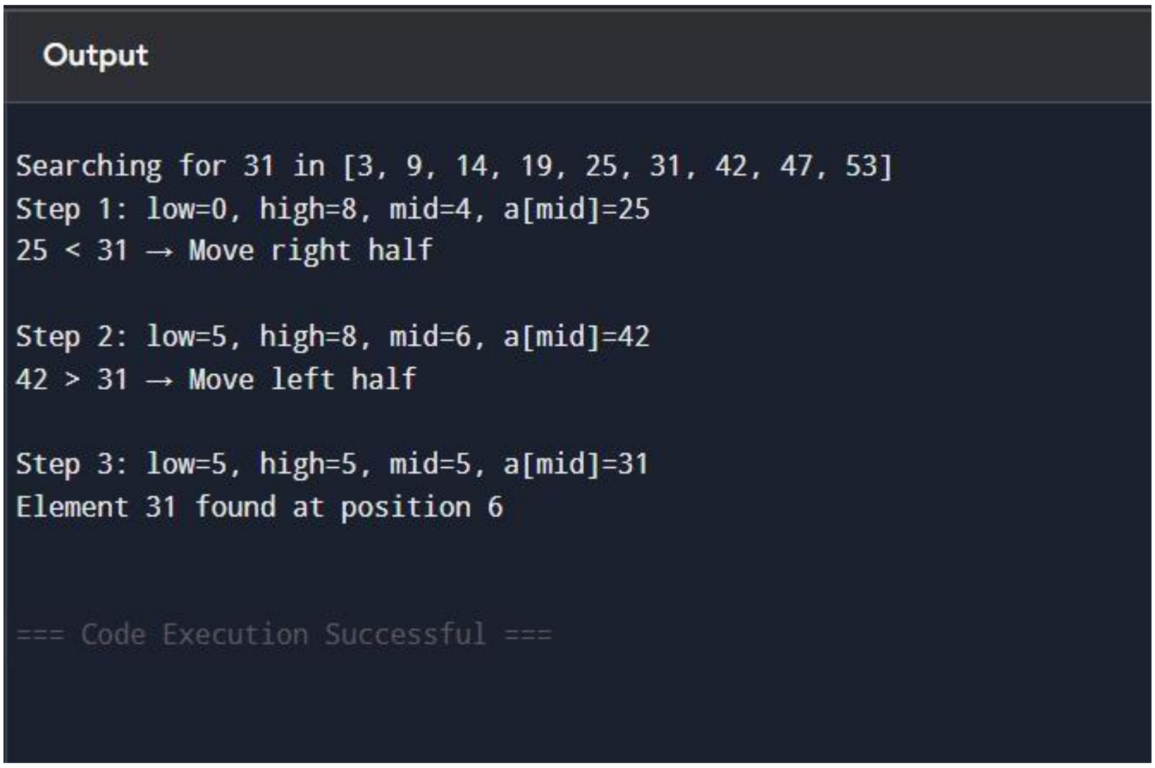
cases

Input : N= 6, a[] = {20,40,60,80,100,120}, search key = 60

Output : 3



```python
def binary_search(arr, key):
    low = 0
    high = len(arr) - 1
    step = 1
    print(f"\nSearching for {key} in {arr}")
    while low <= high:
        mid = (low + high) // 2
        print(f"Step {step}: low={low}, high={high}, mid={mid}, a[mid]
            ={arr[mid]}")
        step += 1
        if arr[mid] == key:
            print(f"Element {key} found at position {mid + 1}\n")
            return mid + 1
        elif arr[mid] < key:
            print(f"{arr[mid]} < {key} → Move right half\n")
            low = mid + 1
        else:
            print(f"{arr[mid]} > {key} → Move left half\n")
            high = mid - 1
    print(f"Element {key} not found in the array.\n")
    return -1
a1 = [3, 9, 14, 19, 25, 31, 42, 47, 53]
binary_search(a1, 31)
```



Output

```

Searching for 31 in [3, 9, 14, 19, 25, 31, 42, 47, 53]
Step 1: low=0, high=8, mid=4, a[mid]=25
25 < 31 → Move right half

Step 2: low=5, high=8, mid=6, a[mid]=42
42 > 31 → Move left half

Step 3: low=5, high=5, mid=5, a[mid]=31
Element 31 found at position 6



=== Code Execution Successful ===
```

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

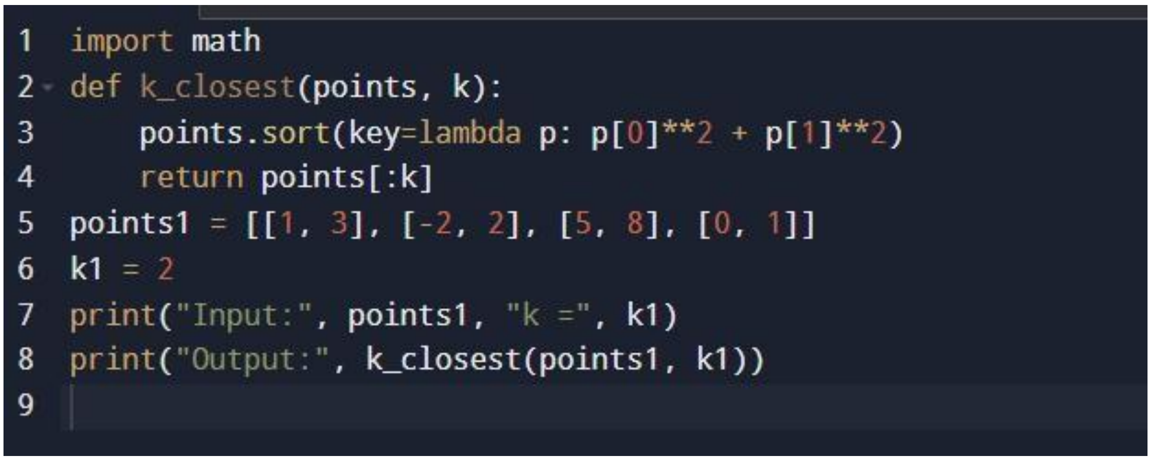
Output:[[-2, 2], [0, 1]]

(ii) **Input**: points = [[1, 3], [-2, 2]], k = 1

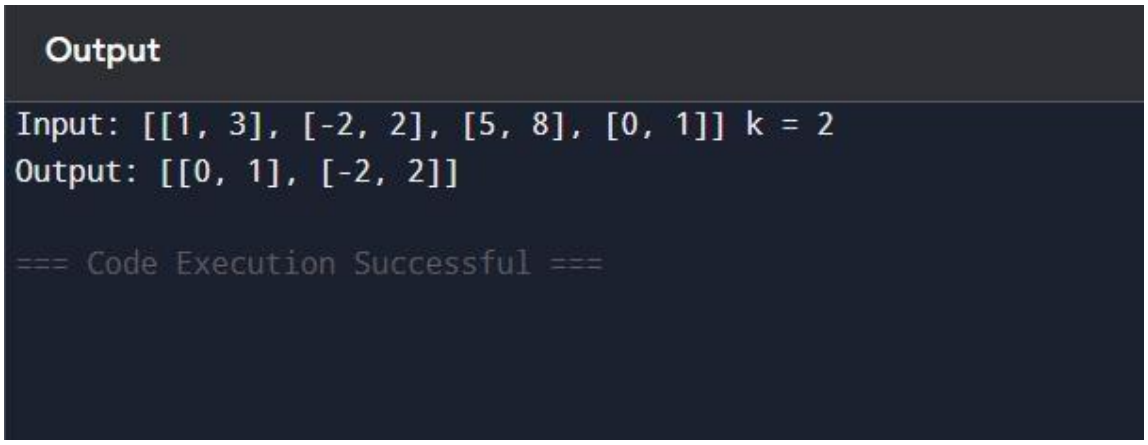
Output: [[-2, 2]]

(iii) **Input**: points = [[3, 3], [5, -1], [-2, 4]], k = 2

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



```python
import math
def k_closest(points, k):
    points.sort(key=lambda p: p[0]**2 + p[1]**2)
    return points[:k]
points1 = [[1, 3], [-2, 2], [5, 8], [0, 1]]
k1 = 2
print("Input:", points1, "k =", k1)
print("Output:", k_closest(points1, k1))

```



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


=== Code Execution Successful ===
```

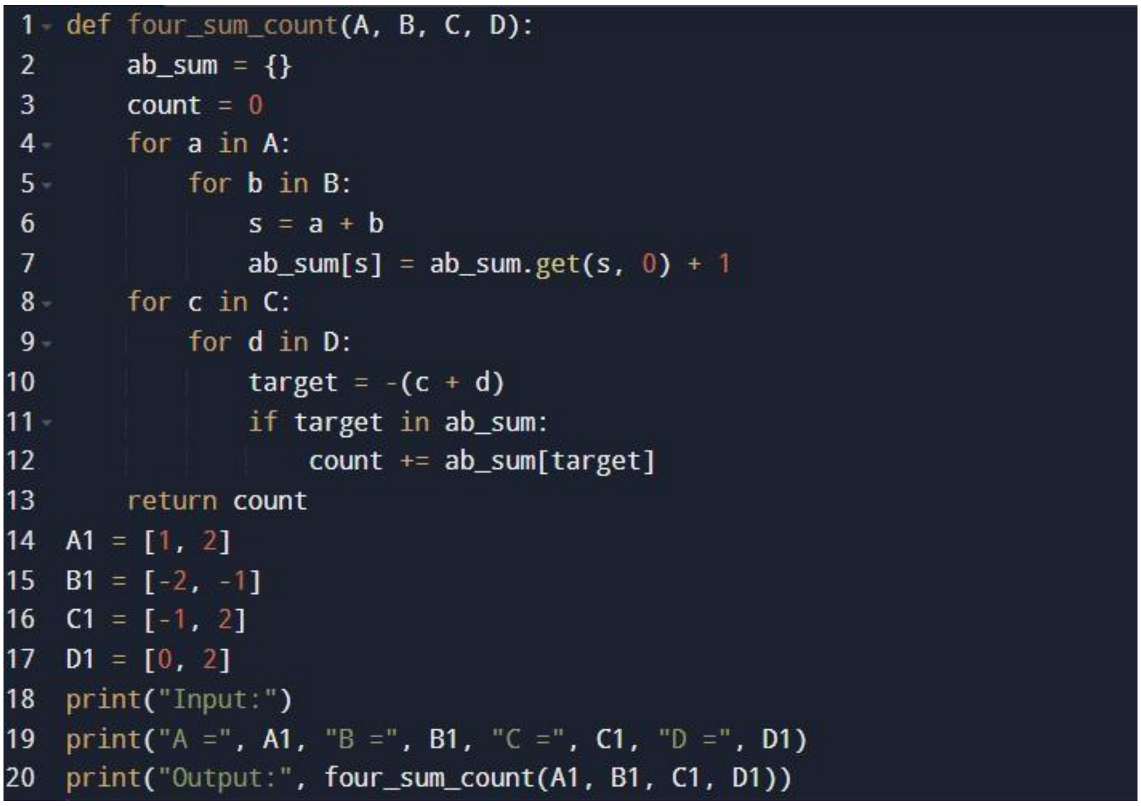
10. Given four lists A, B, C, D of integer values,Write a program to compute how many tuples n(i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero.

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

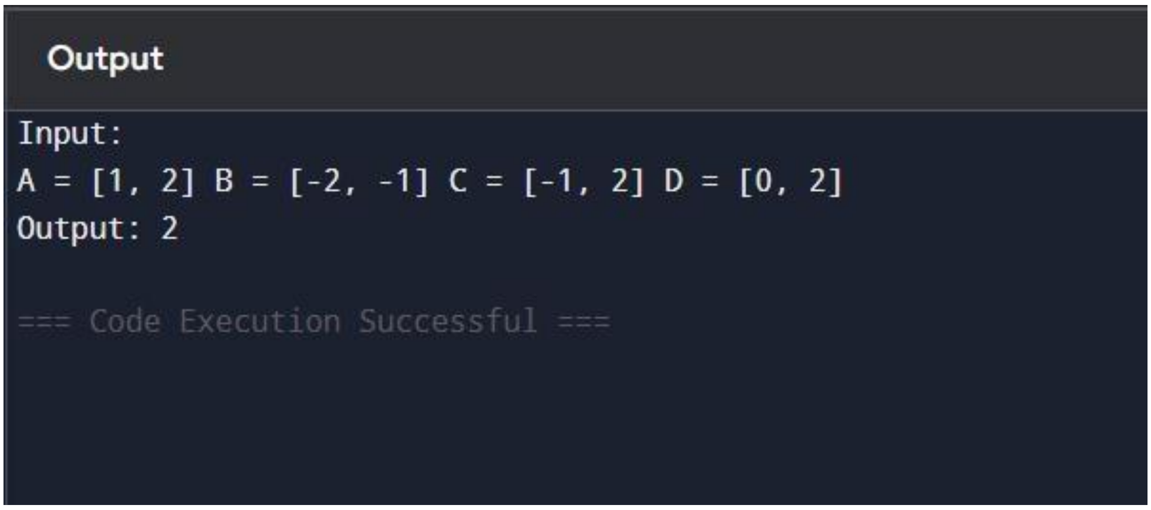
Output: 2

(ii) **Input**: A = [0], B = [0], C = [0], D = [0]

Output: 1



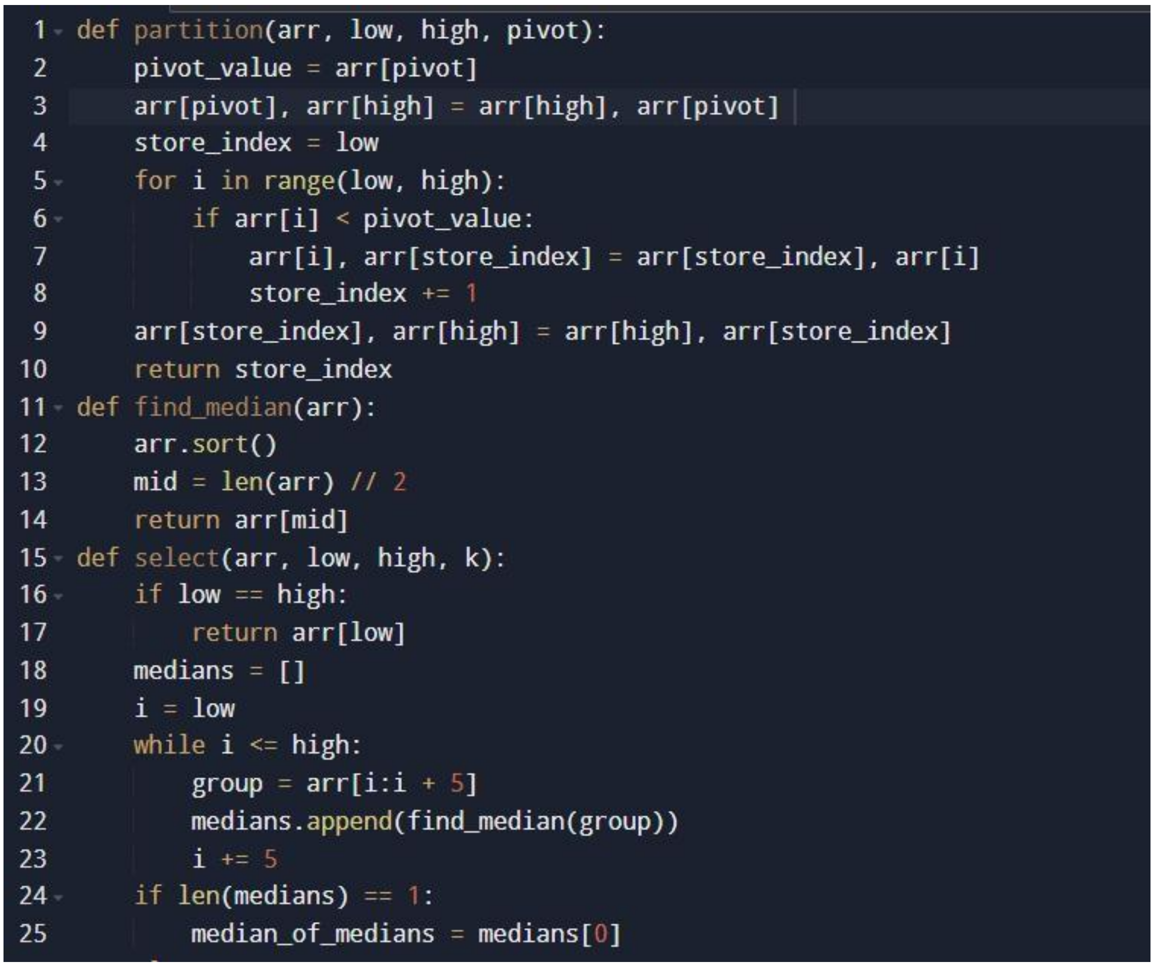
```python
def four_sum_count(A, B, C, D):
    ab_sum = {}
    count = 0
    for a in A:
        for b in B:
            s = a + b
            ab_sum[s] = ab_sum.get(s, 0) + 1
    for c in C:
        for d in D:
            target = -(c + d)
            if target in ab_sum:
                count += ab_sum[target]
    return count
A1 = [1, 2]
B1 = [-2, -1]
C1 = [-1, 2]
D1 = [0, 2]
print("Input:")
print("A =", A1, "B =", B1, "C =", C1, "D =", D1)
print("Output:", four_sum_count(A1, B1, C1, D1))
```



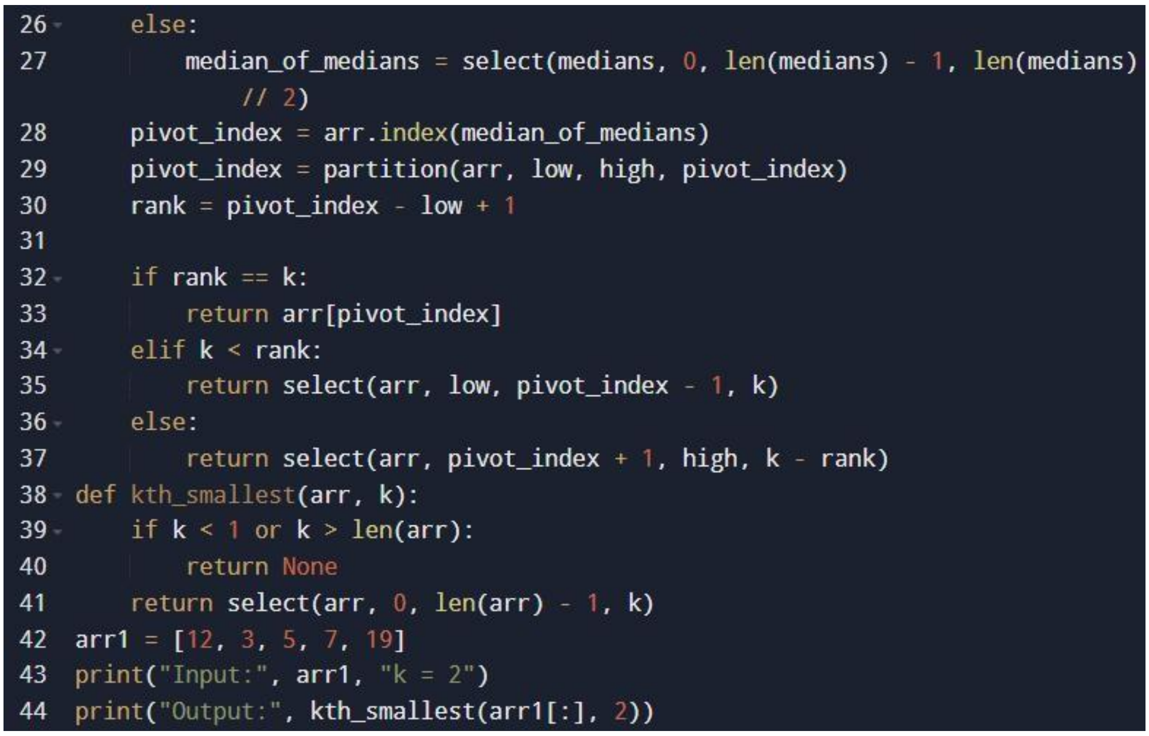
```
Input:
A = [1, 2] B = [-2, -1] C = [-1, 2] D = [0, 2]
Output: 2

=== Code Execution Successful ===
```

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.

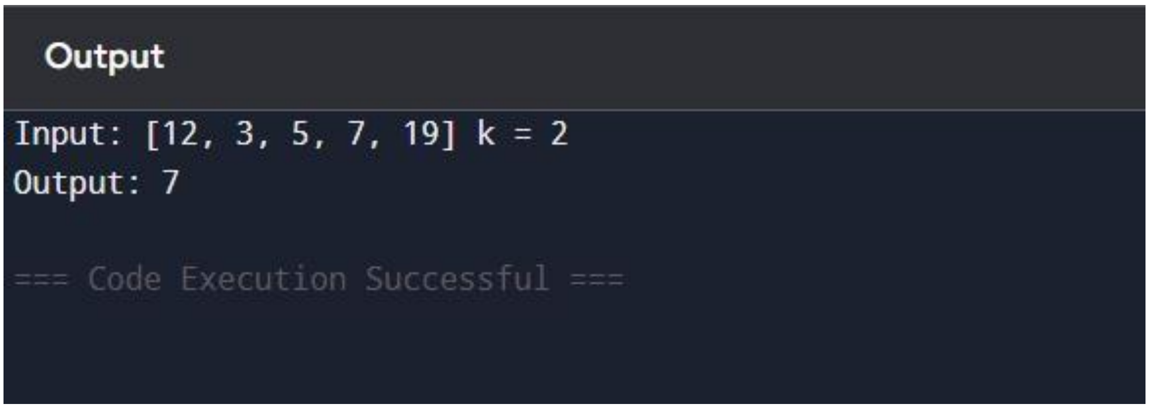
arr = [12, 3, 5, 7, 19] k = 2	Expected Output:5
arr = [12, 3, 5, 7, 4, 19, 26] k = 3	Expected Output:5
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6	Expected Output:6



```python
def partition(arr, low, high, pivot):
    pivot_value = arr[pivot]
    arr[pivot], arr[high] = arr[high], arr[pivot]
    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 find_median(arr):
    arr.sort()
    mid = len(arr) // 2
    return arr[mid]
def select(arr, low, high, k):
    if low == high:
        return arr[low]
    medians = []
    i = low
    while i <= high:
        group = arr[i:i + 5]
        medians.append(find_median(group))
        i += 5
    if len(medians) == 1:
        median_of_medians = medians[0]
```



```python
    else:
        median_of_medians = select(medians, 0, len(medians) - 1, len(medians) // 2)
    pivot_index = arr.index(median_of_medians)
    pivot_index = partition(arr, low, high, pivot_index)
    rank = pivot_index - low + 1

    if rank == k:
        return arr[pivot_index]
    elif k < rank:
        return select(arr, low, pivot_index - 1, k)
    else:
        return select(arr, pivot_index + 1, high, k - rank)
def kth_smallest(arr, k):
    if k < 1 or k > len(arr):
        return None
    return select(arr, 0, len(arr) - 1, k)
arr1 = [12, 3, 5, 7, 19]
print("Input:", arr1, "k = 2")
print("Output:", kth_smallest(arr1[:], 2))
```



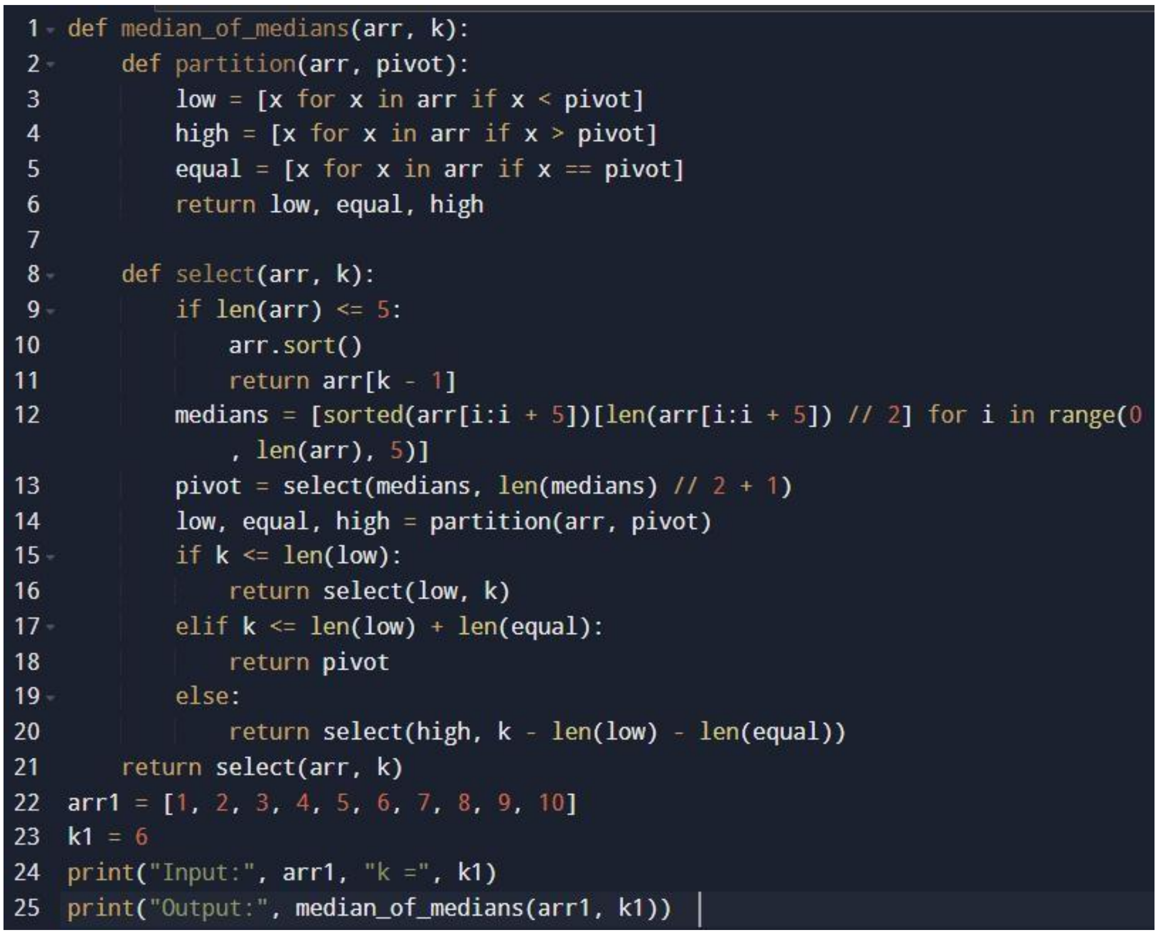
```
Input: [12, 3, 5, 7, 19] k = 2
Output: 7

=== Code Execution Successful ===
```

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.

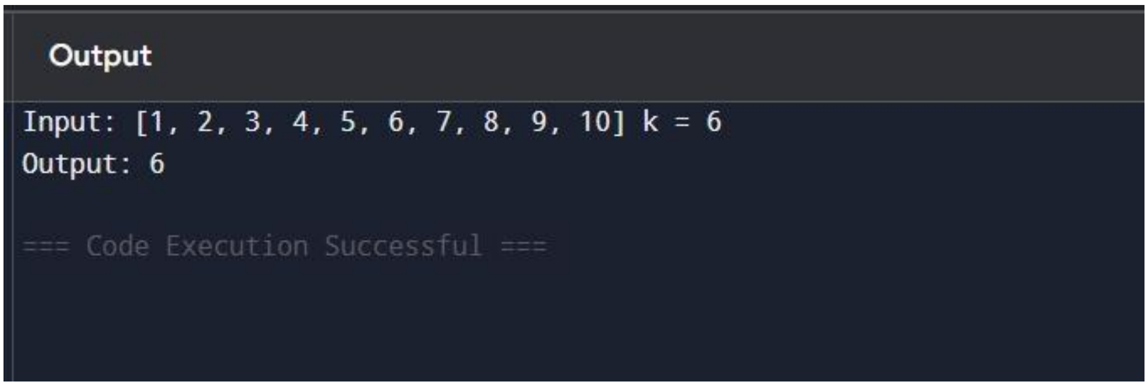
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6

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.



```python
def median_of_medians(arr, k):
    def partition(arr, pivot):
        low = [x for x in arr if x < pivot]
        high = [x for x in arr if x > pivot]
        equal = [x for x in arr if x == pivot]
        return low, equal, high

    def select(arr, k):
        if len(arr) <= 5:
            arr.sort()
            return arr[k - 1]
        medians = [sorted(arr[i:i + 5])[len(arr[i:i + 5]) // 2] for i in range(0
            , len(arr), 5)]
        pivot = select(medians, len(medians) // 2 + 1)
        low, equal, high = partition(arr, pivot)
        if k <= len(low):
            return select(low, k)
        elif k <= len(low) + len(equal):
            return pivot
        else:
            return select(high, k - len(low) - len(equal))
    return select(arr, k)
arr1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k1 = 6
print("Input:", arr1, "k =", k1)
print("Output:", median_of_medians(arr1, k1))
```



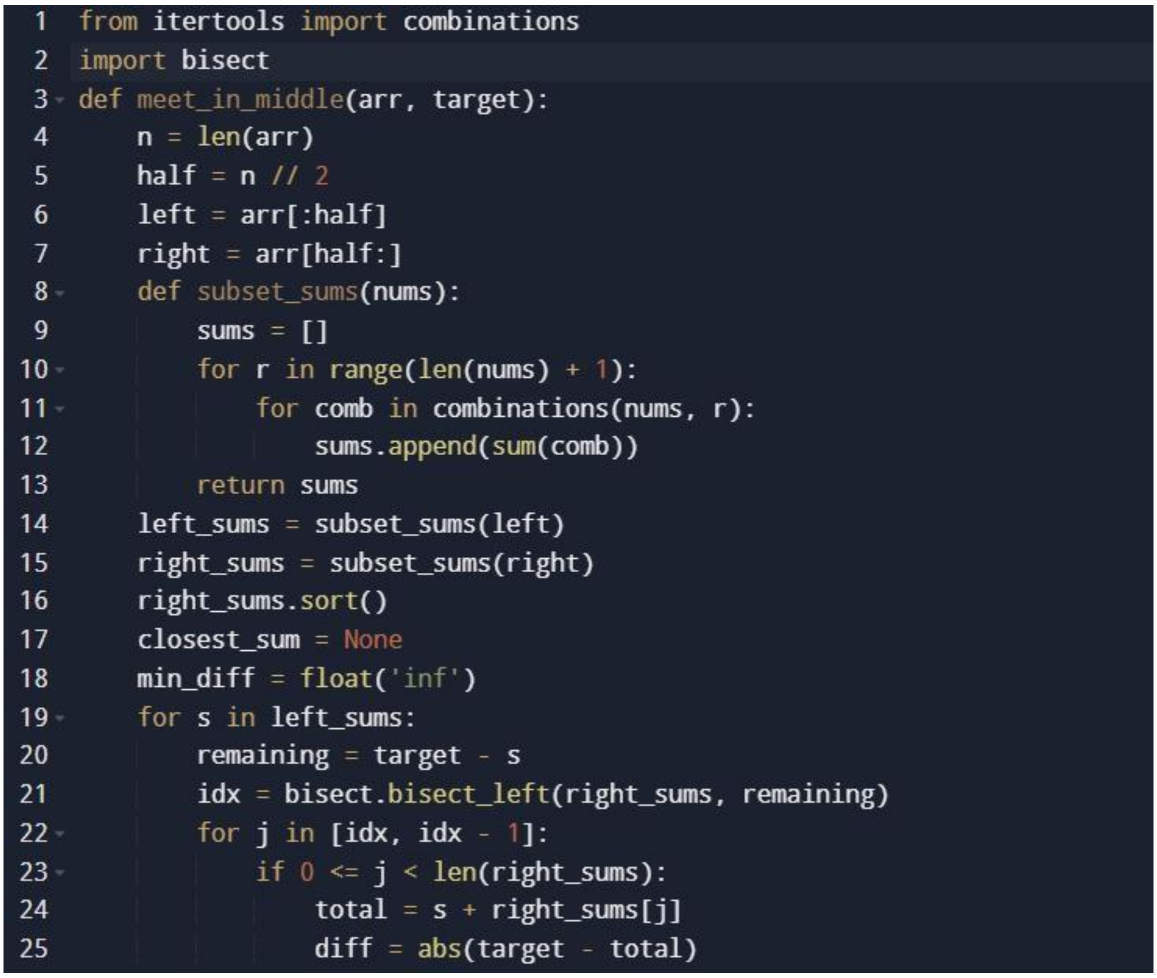
Output

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

=== Code Execution Successful ===
```

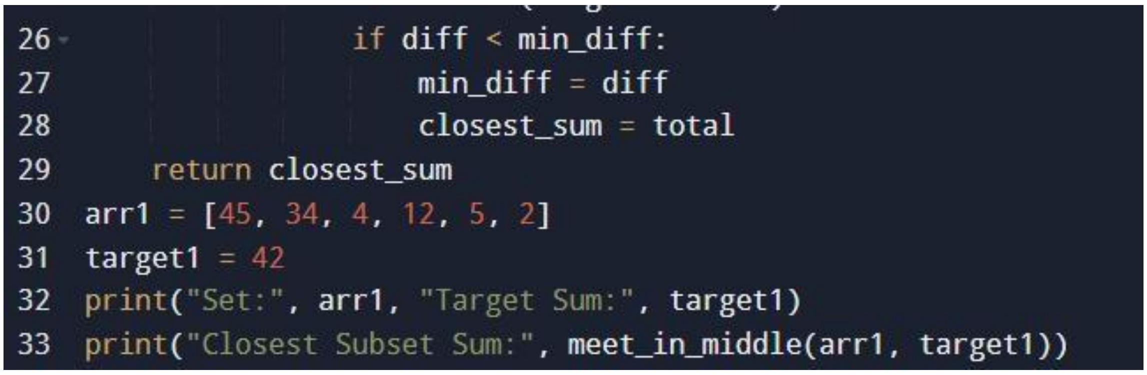
13. Write a program to implement Meet in the Middle Technique. Given an array of integers and a target sum, find the subset whose sum is closest to the target. You will use the Meet in the Middle technique to efficiently find this subset.

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

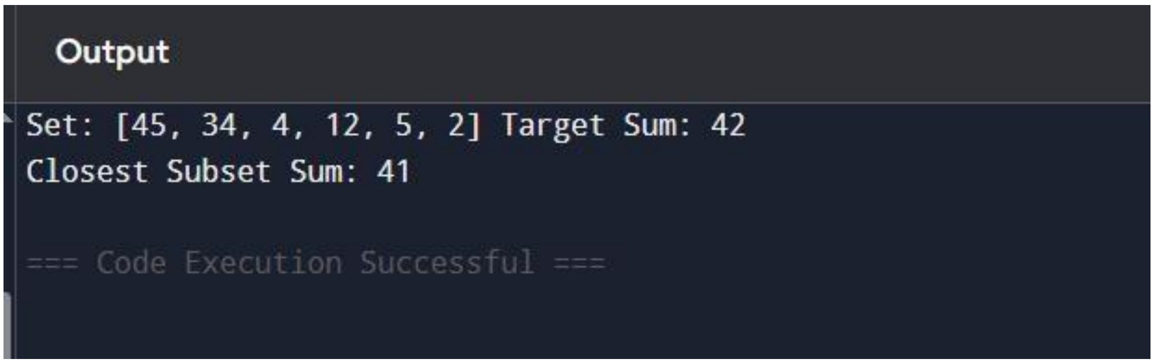
b) Set[]= {1, 3, 2, 7, 4, 6}   Target sum = 10:



```python
from itertools import combinations
import bisect
def meet_in_middle(arr, target):
    n = len(arr)
    half = n // 2
    left = arr[:half]
    right = arr[half:]
    def subset_sums(nums):
        sums = []
        for r in range(len(nums) + 1):
            for comb in combinations(nums, r):
                sums.append(sum(comb))
        return sums
    left_sums = subset_sums(left)
    right_sums = subset_sums(right)
    right_sums.sort()
    closest_sum = None
    min_diff = float('inf')
    for s in left_sums:
        remaining = target - s
        idx = bisect.bisect_left(right_sums, remaining)
        for j in [idx, idx - 1]:
            if 0 <= j < len(right_sums):
                total = s + right_sums[j]
                diff = abs(target - total)
```



```python
                if diff < min_diff:
                    min_diff = diff
                    closest_sum = total
    return closest_sum
arr1 = [45, 34, 4, 12, 5, 2]
target1 = 42
print("Set:", arr1, "Target Sum:", target1)
print("Closest Subset Sum:", meet_in_middle(arr1, target1))
```

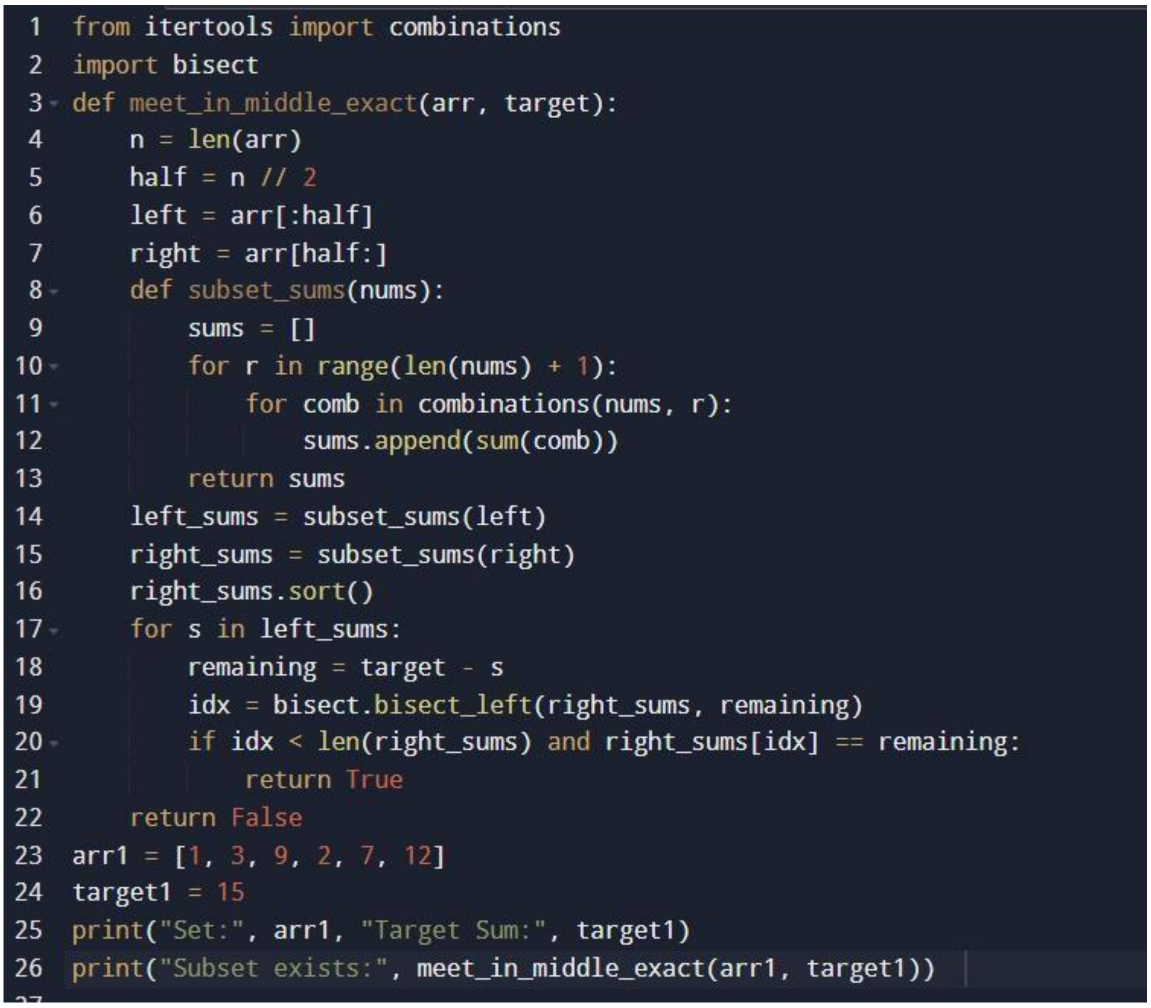


```
Set: [45, 34, 4, 12, 5, 2] Target Sum: 42
Closest Subset Sum: 41

=== Code Execution Successful ===
```

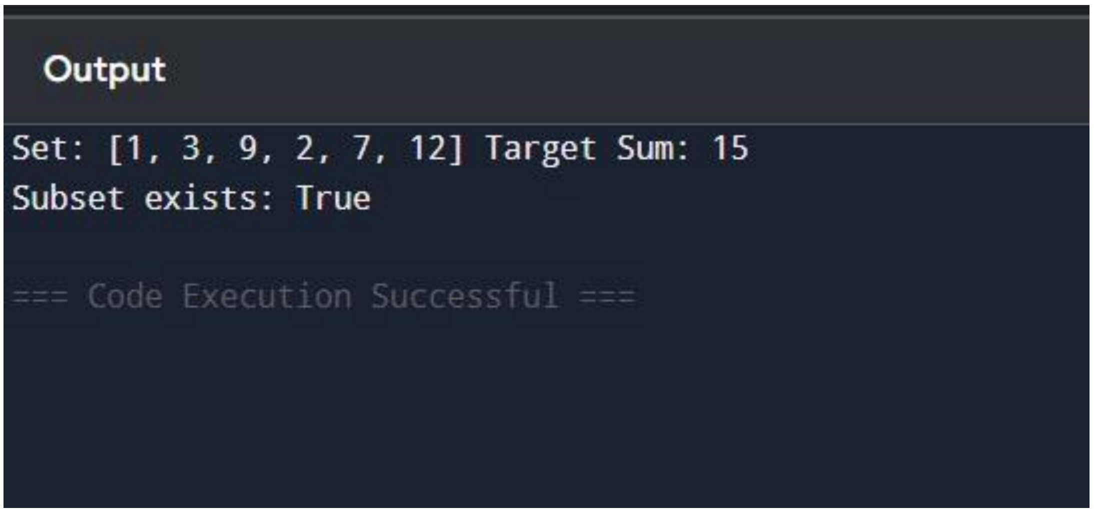
14. Write a program to implement Meet in the Middle Technique. Given a large array of integers and an exact sum E, determine if there is any subset that sums exactly to E. Utilize the Meet in the Middle technique to handle the potentially large size of the array. Return true if there is a subset that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

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

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



```python
from itertools import combinations
import bisect
def meet_in_middle_exact(arr, target):
    n = len(arr)
    half = n // 2
    left = arr[:half]
    right = arr[half:]
    def subset_sums(nums):
        sums = []
        for r in range(len(nums) + 1):
            for comb in combinations(nums, r):
                sums.append(sum(comb))
        return sums
    left_sums = subset_sums(left)
    right_sums = subset_sums(right)
    right_sums.sort()
    for s in left_sums:
        remaining = target - s
        idx = bisect.bisect_left(right_sums, remaining)
        if idx < len(right_sums) and right_sums[idx] == remaining:
            return True
    return False
arr1 = [1, 3, 9, 2, 7, 12]
target1 = 15
print("Set:", arr1, "Target Sum:", target1)
print("Subset exists:", meet_in_middle_exact(arr1, target1))
```



Output

```
Set: [1, 3, 9, 2, 7, 12] Target Sum: 15
Subset exists: True


=== Code Execution Successful ===
```

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:

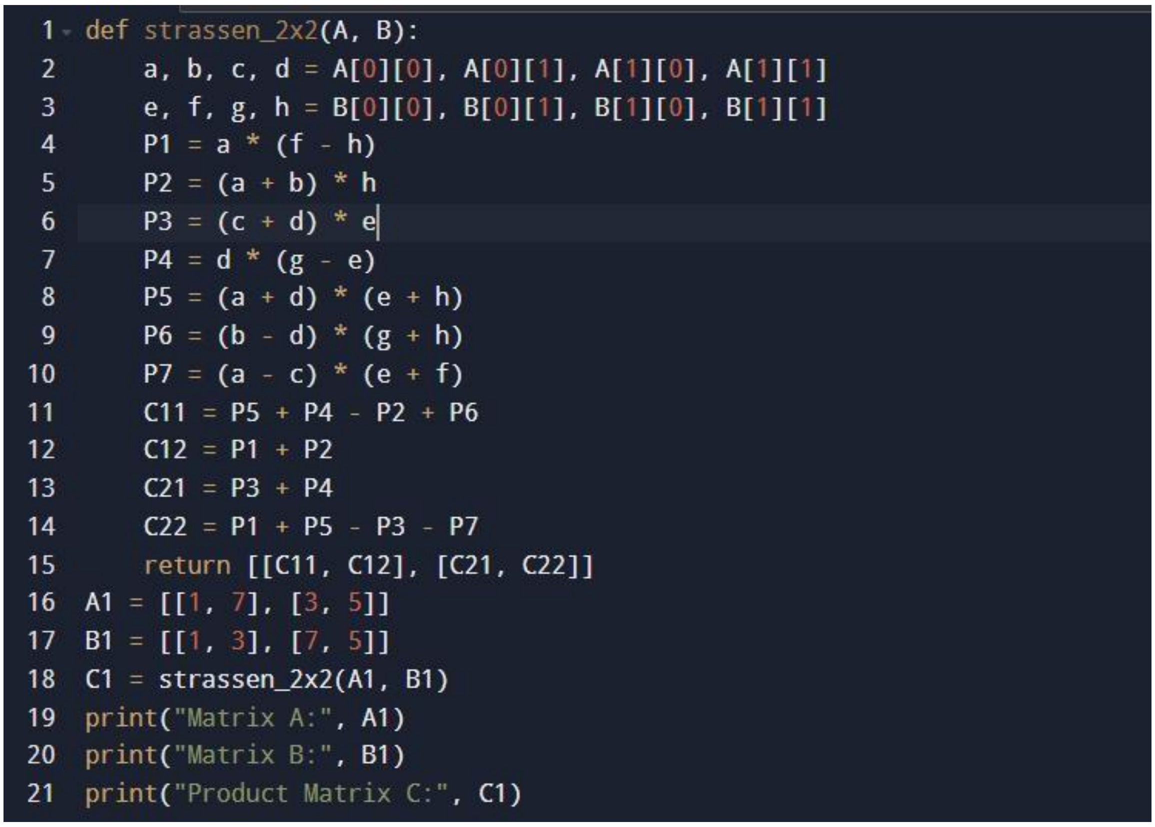
A=(1 7   B=(6 8

3 5   ),   4   2)

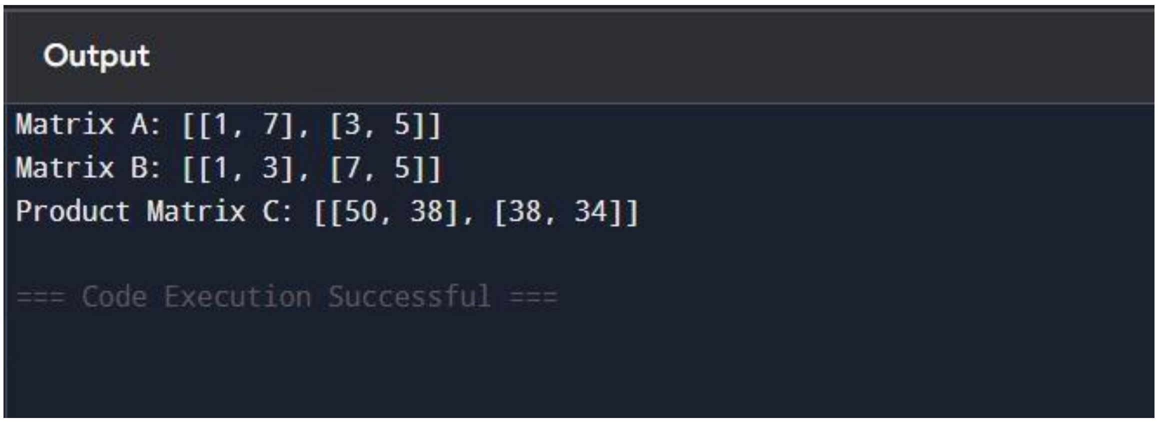
Expected Output: C=(18

14

62 66)



```python
def strassen_2x2(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]
    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)
    C11 = P5 + P4 - P2 + P6
    C12 = P1 + P2
    C21 = P3 + P4
    C22 = P1 + P5 - P3 - P7
    return [[C11, C12], [C21, C22]]
A1 = [[1, 7], [3, 5]]
B1 = [[1, 3], [7, 5]]
C1 = strassen_2x2(A1, B1)
print("Matrix A:", A1)
print("Matrix B:", B1)
print("Product Matrix C:", C1)
```



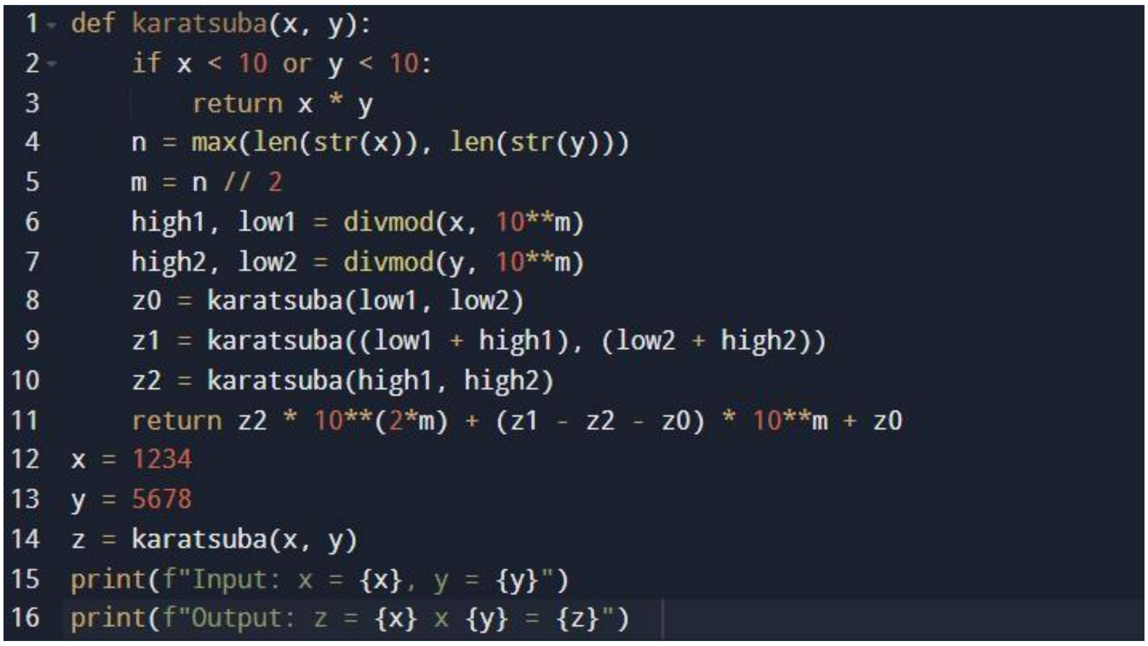
```
Matrix A: [[1, 7], [3, 5]]
Matrix B: [[1, 3], [7, 5]]
Product Matrix C: [[50, 38], [38, 34]]

=== Code Execution Successful ===
```

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

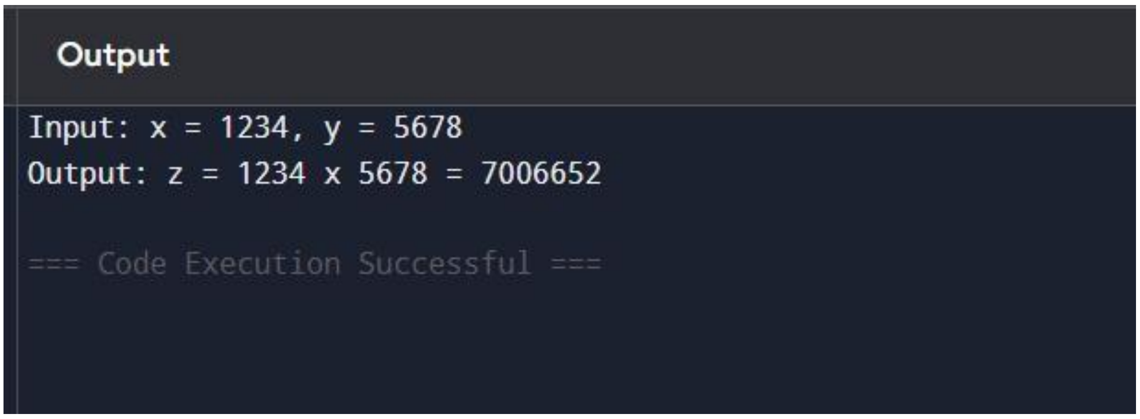
Test Case 1:

Input: x=1234,y=5678

Expected Output: z=1234×5678=7016652



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



Output

```
Input: x = 1234, y = 5678
Output: z = 1234 x 5678 = 7006652


=== Code Execution Successful ===
```