**Top 10 Coding Questions on Arrays & Searching**

📌 **1. Find the Second Largest and Second Smallest Element**  
Given an array of N integers, find the second largest and second smallest elements without sorting the array.

📌 **2. Spiral Traversal of a Matrix**  
Given an N x M matrix, print its elements in spiral order, starting from the top-left corner and moving inward in a clockwise direction.

📌 **3. Linear Search on an Unsorted Array**  
Given an array of N elements and a target value X, search for X in the array using the linear search technique and return its index. If X is not found, return -1.

📌 **4. Binary Search - First and Last Occurrence**  
Given a sorted array and a target element X, find the index of its first and last occurrence using binary search. If X is not found, return -1, -1.

📌 **5. Finding Lower Bound and Upper Bound**  
Given a sorted array and a target value X, find:

* The lower bound: the first index where arr[i] >= X.
* The upper bound: the first index where arr[i] > X.

📌 **6. Aggressive Cows**

Given N stalls and C cows, place the cows in such a way that the minimum distance between any two cows is maximized.

📌 **7. Painter’s Partition Problem**  
Given N boards and K painters, allocate the boards among the painters such that the maximum time taken by a painter is minimized.

📌 **8. Find the Maximum in a Bitonic Array**  
Given a bitonic array (first increasing, then decreasing), find the maximum element using an optimized approach.

📌 **9. Find the Minimum Cost to Make Array Elements Equal**  
Given an array of N integers, find an integer X such that the sum of absolute differences |arr[i] - X| is minimized.

📌 **10. Median of Two Sorted Arrays**  
Given two sorted arrays of size N and M, find the median of the combined sorted array in O(log(N + M)) time.

### ****📌 1. Find the Second Largest and Second Smallest Element****

Given an array of N integers, find the second largest and second smallest elements without sorting the array.

C++

#include <iostream>

#include <climits>

using namespace std;

void findSecondLargestAndSmallest(int arr[], int n) {

if (n < 2) {

cout << "Not enough elements" << endl;

return;

}

int largest = INT\_MIN, secondLargest = INT\_MIN;

int smallest = INT\_MAX, secondSmallest = INT\_MAX;

for (int i = 0; i < n; i++) {

if (arr[i] > largest) {

secondLargest = largest;

largest = arr[i];

} else if (arr[i] > secondLargest && arr[i] != largest) {

secondLargest = arr[i];

}

if (arr[i] < smallest) {

secondSmallest = smallest;

smallest = arr[i];

} else if (arr[i] < secondSmallest && arr[i] != smallest) {

secondSmallest = arr[i];

}

}

if (secondLargest == INT\_MIN || secondSmallest == INT\_MAX) {

cout << "No second largest or second smallest exists" << endl;

} else {

cout << "Second Largest: " << secondLargest << endl;

cout << "Second Smallest: " << secondSmallest << endl;

}

}

int main() {

int arr[] = {12, 35, 1, 10, 34, 1};

int n = sizeof(arr) / sizeof(arr[0]);

findSecondLargestAndSmallest(arr, n);

return 0;

}

Java:

import java.util.\*;

public class SecondLargestSmallest {

public static void findSecondLargestAndSmallest(int[] arr) {

if (arr.length < 2) {

System.out.println("Not enough elements");

return;

}

int largest = Integer.MIN\_VALUE, secondLargest = Integer.MIN\_VALUE;

int smallest = Integer.MAX\_VALUE, secondSmallest = Integer.MAX\_VALUE;

for (int num : arr) {

if (num > largest) {

secondLargest = largest;

largest = num;

} else if (num > secondLargest && num != largest) {

secondLargest = num;

}

if (num < smallest) {

secondSmallest = smallest;

smallest = num;

} else if (num < secondSmallest && num != smallest) {

secondSmallest = num;

}

}

if (secondLargest == Integer.MIN\_VALUE || secondSmallest == Integer.MAX\_VALUE) {

System.out.println("No second largest or second smallest exists");

} else {

System.out.println("Second Largest: " + secondLargest);

System.out.println("Second Smallest: " + secondSmallest);

}

}

public static void main(String[] args) {

int[] arr = {12, 35, 1, 10, 34, 1};

findSecondLargestAndSmallest(arr);

}

}

Python

def find\_second\_largest\_and\_smallest(arr):

if len(arr) < 2:

print("Not enough elements")

return

largest, second\_largest = float('-inf'), float('-inf')

smallest, second\_smallest = float('inf'), float('inf')

for num in arr:

if num > largest:

second\_largest = largest

largest = num

elif num > second\_largest and num != largest:

second\_largest = num

if num < smallest:

second\_smallest = smallest

smallest = num

elif num < second\_smallest and num != smallest:

second\_smallest = num

if second\_largest == float('-inf') or second\_smallest == float('inf'):

print("No second largest or second smallest exists")

else:

print("Second Largest:", second\_largest)

print("Second Smallest:", second\_smallest)

# Example usage

arr = [12, 35, 1, 10, 34, 1]

find\_second\_largest\_and\_smallest(arr)

### ****📌 2. Spiral Traversal of a Matrix****

Given an N x M matrix, print its elements in spiral order, starting from the top-left corner and moving inward in a clockwise direction.

C++

#include <iostream>

#include <vector>

using namespace std;

void spiralOrder(vector<vector<int>> &matrix) {

int rows = matrix.size(), cols = matrix[0].size();

int top = 0, bottom = rows - 1, left = 0, right = cols - 1;

while (top <= bottom && left <= right) {

for (int i = left; i <= right; i++)

cout << matrix[top][i] << " ";

top++;

for (int i = top; i <= bottom; i++)

cout << matrix[i][right] << " ";

right--;

if (top <= bottom) {

for (int i = right; i >= left; i--)

cout << matrix[bottom][i] << " ";

bottom--;

}

if (left <= right) {

for (int i = bottom; i >= top; i--)

cout << matrix[i][left] << " ";

left++;

}

}

cout << endl;

}

int main() {

vector<vector<int>> matrix = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}

};

spiralOrder(matrix);

return 0;

}

Java:  
import java.util.\*;

public class SpiralMatrix {

public static void spiralOrder(int[][] matrix) {

int rows = matrix.length, cols = matrix[0].length;

int top = 0, bottom = rows - 1, left = 0, right = cols - 1;

while (top <= bottom && left <= right) {

for (int i = left; i <= right; i++)

System.out.print(matrix[top][i] + " ");

top++;

for (int i = top; i <= bottom; i++)

System.out.print(matrix[i][right] + " ");

right--;

if (top <= bottom) {

for (int i = right; i >= left; i--)

System.out.print(matrix[bottom][i] + " ");

bottom--;

}

if (left <= right) {

for (int i = bottom; i >= top; i--)

System.out.print(matrix[i][left] + " ");

left++;

}

}

System.out.println();

}

public static void main(String[] args) {

int[][] matrix = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}

};

spiralOrder(matrix);

}

}

Python:

def spiral\_order(matrix):

if not matrix:

return []

result = []

top, bottom, left, right = 0, len(matrix) - 1, 0, len(matrix[0]) - 1

while top <= bottom and left <= right:

for i in range(left, right + 1):

result.append(matrix[top][i])

top += 1

for i in range(top, bottom + 1):

result.append(matrix[i][right])

right -= 1

if top <= bottom:

for i in range(right, left - 1, -1):

result.append(matrix[bottom][i])

bottom -= 1

if left <= right:

for i in range(bottom, top - 1, -1):

result.append(matrix[i][left])

left += 1

print(" ".join(map(str, result)))

# Example usage

matrix = [

[1, 2, 3, 4],

[5, 6, 7, 8],

[9, 10, 11, 12],

[13, 14, 15, 16]

]

spiral\_order(matrix)

### ****📌 3. Linear Search on an Unsorted Array****

Given an array of N elements and a target value X, search for X in the array using the linear search technique and return its index. If X is not found, return -1.

C++

#include <iostream>

using namespace std;

int linearSearch(int arr[], int n, int x) {

for (int i = 0; i < n; i++) {

if (arr[i] == x)

return i; // Return index if found

}

return -1; // Return -1 if not found

}

int main() {

int arr[] = {10, 25, 30, 15, 40, 50};

int n = sizeof(arr) / sizeof(arr[0]);

int x = 15;

int result = linearSearch(arr, n, x);

if (result != -1)

cout << "Element found at index: " << result << endl;

else

cout << "Element not found" << endl;

return 0;

}

Java:  
import java.util.\*;

public class LinearSearch {

public static int linearSearch(int[] arr, int x) {

for (int i = 0; i < arr.length; i++) {

if (arr[i] == x)

return i; // Return index if found

}

return -1; // Return -1 if not found

}

public static void main(String[] args) {

int[] arr = {10, 25, 30, 15, 40, 50};

int x = 15;

int result = linearSearch(arr, x);

if (result != -1)

System.out.println("Element found at index: " + result);

else

System.out.println("Element not found");

}

}

Python:

def linear\_search(arr, x):

for i in range(len(arr)):

if arr[i] == x:

return i # Return index if found

return -1 # Return -1 if not found

# Example usage

arr = [10, 25, 30, 15, 40, 50]

x = 15

result = linear\_search(arr, x)

if result != -1:

print(f"Element found at index: {result}")

else:

print("Element not found")

### ****📌 4. Binary Search - First and Last Occurrence****

Given a sorted array and a target element X, find the index of its first and last occurrence using binary search. If X is not found, return -1, -1.

C++:

#include <iostream>

#include <vector>

using namespace std;

int findFirstOccurrence(vector<int>& arr, int x) {

int left = 0, right = arr.size() - 1, result = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == x) {

result = mid;

right = mid - 1; // Search in the left half

} else if (arr[mid] < x) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

int findLastOccurrence(vector<int>& arr, int x) {

int left = 0, right = arr.size() - 1, result = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == x) {

result = mid;

left = mid + 1; // Search in the right half

} else if (arr[mid] < x) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

int main() {

vector<int> arr = {1, 2, 2, 2, 3, 4, 5};

int x = 2;

int first = findFirstOccurrence(arr, x);

int last = findLastOccurrence(arr, x);

cout << "First Occurrence: " << first << ", Last Occurrence: " << last << endl;

return 0;

}

Java:

import java.util.\*;

public class BinarySearchFirstLast {

public static int findFirstOccurrence(int[] arr, int x) {

int left = 0, right = arr.length - 1, result = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == x) {

result = mid;

right = mid - 1; // Search in the left half

} else if (arr[mid] < x) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

public static int findLastOccurrence(int[] arr, int x) {

int left = 0, right = arr.length - 1, result = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == x) {

result = mid;

left = mid + 1; // Search in the right half

} else if (arr[mid] < x) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

public static void main(String[] args) {

int[] arr = {1, 2, 2, 2, 3, 4, 5};

int x = 2;

int first = findFirstOccurrence(arr, x);

int last = findLastOccurrence(arr, x);

System.out.println("First Occurrence: " + first + ", Last Occurrence: " + last);

}

}

Python:

def find\_first\_occurrence(arr, x):

left, right, result = 0, len(arr) - 1, -1

while left <= right:

mid = left + (right - left) // 2

if arr[mid] == x:

result = mid

right = mid - 1 # Search in the left half

elif arr[mid] < x:

left = mid + 1

else:

right = mid - 1

return result

def find\_last\_occurrence(arr, x):

left, right, result = 0, len(arr) - 1, -1

while left <= right:

mid = left + (right - left) // 2

if arr[mid] == x:

result = mid

left = mid + 1 # Search in the right half

elif arr[mid] < x:

left = mid + 1

else:

right = mid - 1

return result

# Example usage

arr = [1, 2, 2, 2, 3, 4, 5]

x = 2

first = find\_first\_occurrence(arr, x)

last = find\_last\_occurrence(arr, x)

print("First Occurrence:", first, ", Last Occurrence:", last)

**📌 5. Finding Lower Bound and Upper Bound**

Given a sorted array and a target value X, find:

* **Lower Bound:** The first index where arr[i] >= X.
* **Upper Bound:** The first index where arr[i] > X.

C++:

#include <iostream>

#include <vector>

using namespace std;

int lowerBound(vector<int>& arr, int x) {

int left = 0, right = arr.size();

while (left < right) {

int mid = left + (right - left) / 2;

if (arr[mid] >= x)

right = mid;

else

left = mid + 1;

}

return left;

}

int upperBound(vector<int>& arr, int x) {

int left = 0, right = arr.size();

while (left < right) {

int mid = left + (right - left) / 2;

if (arr[mid] > x)

right = mid;

else

left = mid + 1;

}

return left;

}

int main() {

vector<int> arr = {1, 2, 2, 3, 5, 6, 8};

int x = 2;

cout << "Lower Bound: " << lowerBound(arr, x) << endl;

cout << "Upper Bound: " << upperBound(arr, x) << endl;

return 0;

}

Java:

import java.util.\*;

public class LowerUpperBound {

public static int lowerBound(int[] arr, int x) {

int left = 0, right = arr.length;

while (left < right) {

int mid = left + (right - left) / 2;

if (arr[mid] >= x)

right = mid;

else

left = mid + 1;

}

return left;

}

public static int upperBound(int[] arr, int x) {

int left = 0, right = arr.length;

while (left < right) {

int mid = left + (right - left) / 2;

if (arr[mid] > x)

right = mid;

else

left = mid + 1;

}

return left;

}

public static void main(String[] args) {

int[] arr = {1, 2, 2, 3, 5, 6, 8};

int x = 2;

System.out.println("Lower Bound: " + lowerBound(arr, x));

System.out.println("Upper Bound: " + upperBound(arr, x));

}

}

Python:

def lower\_bound(arr, x):

left, right = 0, len(arr)

while left < right:

mid = left + (right - left) // 2

if arr[mid] >= x:

right = mid

else:

left = mid + 1

return left

def upper\_bound(arr, x):

left, right = 0, len(arr)

while left < right:

mid = left + (right - left) // 2

if arr[mid] > x:

right = mid

else:

left = mid + 1

return left

# Example usage

arr = [1, 2, 2, 3, 5, 6, 8]

x = 2

print("Lower Bound:", lower\_bound(arr, x))

print("Upper Bound:", upper\_bound(arr, x))

📌 **6. Aggressive Cows**

Given N stalls and C cows, place the cows in such a way that the minimum distance between any two cows is maximized.

**Approach:**

* Sort the stalls to ensure they are in increasing order.
* Use **binary search on the answer** (minimum possible distance between cows).
* Check feasibility using a greedy approach:
  + Place the first cow at the first stall.
  + Try placing the next cow at least mid distance apart.
  + If all cows can be placed, increase mid (search in the right half).
  + If not, decrease mid (search in the left half).

C++:

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

bool canPlaceCows(vector<int>& stalls, int cows, int minDist) {

int count = 1, lastPos = stalls[0];

for (int i = 1; i < stalls.size(); i++) {

if (stalls[i] - lastPos >= minDist) {

count++;

lastPos = stalls[i];

if (count == cows) return true;

}

}

return false;

}

int aggressiveCows(vector<int>& stalls, int cows) {

sort(stalls.begin(), stalls.end());

int left = 1, right = stalls.back() - stalls.front(), result = 0;

while (left <= right) {

int mid = left + (right - left) / 2;

if (canPlaceCows(stalls, cows, mid)) {

result = mid;

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

int main() {

vector<int> stalls = {1, 2, 8, 4, 9};

int cows = 3;

cout << "Maximum Minimum Distance: " << aggressiveCows(stalls, cows) << endl;

return 0;

}

Java:

import java.util.\*;

public class AggressiveCows {

public static boolean canPlaceCows(int[] stalls, int cows, int minDist) {

int count = 1, lastPos = stalls[0];

for (int i = 1; i < stalls.length; i++) {

if (stalls[i] - lastPos >= minDist) {

count++;

lastPos = stalls[i];

if (count == cows) return true;

}

}

return false;

}

public static int aggressiveCows(int[] stalls, int cows) {

Arrays.sort(stalls);

int left = 1, right = stalls[stalls.length - 1] - stalls[0], result = 0;

while (left <= right) {

int mid = left + (right - left) / 2;

if (canPlaceCows(stalls, cows, mid)) {

result = mid;

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

public static void main(String[] args) {

int[] stalls = {1, 2, 8, 4, 9};

int cows = 3;

System.out.println("Maximum Minimum Distance: " + aggressiveCows(stalls, cows));

}

}

Python:  
def can\_place\_cows(stalls, cows, min\_dist):

count, last\_pos = 1, stalls[0]

for i in range(1, len(stalls)):

if stalls[i] - last\_pos >= min\_dist:

count += 1

last\_pos = stalls[i]

if count == cows:

return True

return False

def aggressive\_cows(stalls, cows):

stalls.sort()

left, right, result = 1, stalls[-1] - stalls[0], 0

while left <= right:

mid = left + (right - left) // 2

if can\_place\_cows(stalls, cows, mid):

result = mid

left = mid + 1

else:

right = mid - 1

return result

# Example usage

stalls = [1, 2, 8, 4, 9]

cows = 3

print("Maximum Minimum Distance:", aggressive\_cows(stalls, cows))

### ****📌 7. Painter’s Partition Problem****

Given N boards and K painters, allocate the boards among the painters such that the maximum time taken by a painter is minimized.

**Approach:**

* **Binary Search on Answer:** The answer lies between the **maximum** board length (lower bound) and the **sum** of all board lengths (upper bound).

C++:  
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

bool isPossible(vector<int>& boards, int painters, int maxTime) {

int count = 1, sum = 0;

for (int b : boards) {

if (sum + b > maxTime) {

count++;

sum = b;

if (count > painters) return false;

} else {

sum += b;

}

}

return true;

}

int paintersPartition(vector<int>& boards, int painters) {

int left = \*max\_element(boards.begin(), boards.end());

int right = accumulate(boards.begin(), boards.end(), 0);

int result = right;

while (left <= right) {

int mid = left + (right - left) / 2;

if (isPossible(boards, painters, mid)) {

result = mid;

right = mid - 1;

} else {

left = mid + 1;

}

}

return result;

}

int main() {

vector<int> boards = {10, 20, 30, 40};

int painters = 2;

cout << "Minimum Time Required: " << paintersPartition(boards, painters) << endl;

return 0;

}

Java:

import java.util.\*;

public class PaintersPartition {

public static boolean isPossible(int[] boards, int painters, int maxTime) {

int count = 1, sum = 0;

for (int b : boards) {

if (sum + b > maxTime) {

count++;

sum = b;

if (count > painters) return false;

} else {

sum += b;

}

}

return true;

}

public static int paintersPartition(int[] boards, int painters) {

int left = Arrays.stream(boards).max().getAsInt();

int right = Arrays.stream(boards).sum();

int result = right;

while (left <= right) {

int mid = left + (right - left) / 2;

if (isPossible(boards, painters, mid)) {

result = mid;

right = mid - 1;

} else {

left = mid + 1;

}

}

return result;

}

public static void main(String[] args) {

int[] boards = {10, 20, 30, 40};

int painters = 2;

System.out.println("Minimum Time Required: " + paintersPartition(boards, painters));

}

}

Python:

def is\_possible(boards, painters, max\_time):

count, total = 1, 0

for b in boards:

if total + b > max\_time:

count += 1

total = b

if count > painters:

return False

else:

total += b

return True

def painters\_partition(boards, painters):

left, right = max(boards), sum(boards)

result = right

while left <= right:

mid = left + (right - left) // 2

if is\_possible(boards, painters, mid):

result = mid

right = mid - 1

else:

left = mid + 1

return result

# Example usage

boards = [10, 20, 30, 40]

painters = 2

print("Minimum Time Required:", painters\_partition(boards, painters))

### ****📌 8. Find the Maximum in a Bitonic Array****

Given a **bitonic array** (an array that first increases and then decreases), find the **maximum element** using an optimized approach.

**Approach (Binary Search)**

* A **bitonic array** has a single peak element where arr[i-1] < arr[i] > arr[i+1].
* Use **binary search** to locate this peak efficiently:
  + If arr[mid] < arr[mid+1], the peak lies to the right (left = mid + 1).
  + Else, the peak lies to the left or at mid (right = mid).
* Continue until left == right, which gives the peak element.

C++:  
#include <iostream>

#include <vector>

using namespace std;

int findMaximum(vector<int>& arr) {

int left = 0, right = arr.size() - 1;

while (left < right) {

int mid = left + (right - left) / 2;

if (arr[mid] < arr[mid + 1])

left = mid + 1; // Move right

else

right = mid; // Move left

}

return arr[left]; // Peak element

}

int main() {

vector<int> arr = {1, 3, 8, 12, 4, 2};

cout << "Maximum Element: " << findMaximum(arr) << endl;

return 0;

}

Java:  
public class BitonicArrayMax {

public static int findMaximum(int[] arr) {

int left = 0, right = arr.length - 1;

while (left < right) {

int mid = left + (right - left) / 2;

if (arr[mid] < arr[mid + 1])

left = mid + 1; // Move right

else

right = mid; // Move left

}

return arr[left]; // Peak element

}

public static void main(String[] args) {

int[] arr = {1, 3, 8, 12, 4, 2};

System.out.println("Maximum Element: " + findMaximum(arr));

}

}

Python:

def find\_maximum(arr):

left, right = 0, len(arr) - 1

while left < right:

mid = left + (right - left) // 2

if arr[mid] < arr[mid + 1]:

left = mid + 1 # Move right

else:

right = mid # Move left

return arr[left] # Peak element

# Example usage

arr = [1, 3, 8, 12, 4, 2]

print("Maximum Element:", find\_maximum(arr))

📌 **9. Find the Minimum Cost to Make Array Elements Equal**  
Given an array of N integers, find an integer X such that the sum of absolute differences |arr[i] - X| is minimized.

C++

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int minCostToMakeEqual(vector<int>& arr) {

sort(arr.begin(), arr.end());

int median = arr[arr.size() / 2];

int cost = 0;

for (int num : arr) {

cost += abs(num - median);

}

return cost;

}

int main() {

vector<int> arr = {1, 2, 3, 9, 10};

cout << "Minimum Cost: " << minCostToMakeEqual(arr) << endl;

return 0;

}

Java:

import java.util.\*;

public class MinCostEqualArray {

public static int minCostToMakeEqual(int[] arr) {

Arrays.sort(arr);

int median = arr[arr.length / 2];

int cost = 0;

for (int num : arr) {

cost += Math.abs(num - median);

}

return cost;

}

public static void main(String[] args) {

int[] arr = {1, 2, 3, 9, 10};

System.out.println("Minimum Cost: " + minCostToMakeEqual(arr));

}

}

Python:

def min\_cost\_to\_make\_equal(arr):

arr.sort()

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

return sum(abs(num - median) for num in arr)

# Example usage

arr = [1, 2, 3, 9, 10]

print("Minimum Cost:", min\_cost\_to\_make\_equal(arr))

📌 **10. Median of Two Sorted Arrays**  
Given two sorted arrays of size N and M, find the median of the combined sorted array in O(log(N + M)) time.

C++:  
#include <iostream>

#include <vector>

#include <climits>

using namespace std;

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

if (nums1.size() > nums2.size())

return findMedianSortedArrays(nums2, nums1);

int n1 = nums1.size(), n2 = nums2.size();

int low = 0, high = n1;

while (low <= high) {

int cut1 = (low + high) / 2;

int cut2 = (n1 + n2 + 1) / 2 - cut1;

int left1 = (cut1 == 0) ? INT\_MIN : nums1[cut1 - 1];

int left2 = (cut2 == 0) ? INT\_MIN : nums2[cut2 - 1];

int right1 = (cut1 == n1) ? INT\_MAX : nums1[cut1];

int right2 = (cut2 == n2) ? INT\_MAX : nums2[cut2];

if (left1 <= right2 && left2 <= right1) {

if ((n1 + n2) % 2 == 0)

return (max(left1, left2) + min(right1, right2)) / 2.0;

else

return max(left1, left2);

} else if (left1 > right2) {

high = cut1 - 1;

} else {

low = cut1 + 1;

}

}

return 0.0;

}

int main() {

vector<int> nums1 = {1, 3};

vector<int> nums2 = {2};

cout << "Median: " << findMedianSortedArrays(nums1, nums2) << endl;

return 0;

}

Java:

public class MedianTwoSortedArrays {

public static double findMedianSortedArrays(int[] nums1, int[] nums2) {

if (nums1.length > nums2.length)

return findMedianSortedArrays(nums2, nums1);

int n1 = nums1.length, n2 = nums2.length;

int low = 0, high = n1;

while (low <= high) {

int cut1 = (low + high) / 2;

int cut2 = (n1 + n2 + 1) / 2 - cut1;

int left1 = (cut1 == 0) ? Integer.MIN\_VALUE : nums1[cut1 - 1];

int left2 = (cut2 == 0) ? Integer.MIN\_VALUE : nums2[cut2 - 1];

int right1 = (cut1 == n1) ? Integer.MAX\_VALUE : nums1[cut1];

int right2 = (cut2 == n2) ? Integer.MAX\_VALUE : nums2[cut2];

if (left1 <= right2 && left2 <= right1) {

if ((n1 + n2) % 2 == 0)

return (Math.max(left1, left2) + Math.min(right1, right2)) / 2.0;

else

return Math.max(left1, left2);

} else if (left1 > right2) {

high = cut1 - 1;

} else {

low = cut1 + 1;

}

}

return 0.0;

}

public static void main(String[] args) {

int[] nums1 = {1, 3};

int[] nums2 = {2};

System.out.println("Median: " + findMedianSortedArrays(nums1, nums2));

}

}

Python:

def find\_median\_sorted\_arrays(nums1, nums2):

if len(nums1) > len(nums2):

return find\_median\_sorted\_arrays(nums2, nums1)

n1, n2 = len(nums1), len(nums2)

low, high = 0, n1

while low <= high:

cut1 = (low + high) // 2

cut2 = (n1 + n2 + 1) // 2 - cut1

left1 = float('-inf') if cut1 == 0 else nums1[cut1 - 1]

left2 = float('-inf') if cut2 == 0 else nums2[cut2 - 1]

right1 = float('inf') if cut1 == n1 else nums1[cut1]

right2 = float('inf') if cut2 == n2 else nums2[cut2]

if left1 <= right2 and left2 <= right1:

if (n1 + n2) % 2 == 0:

return (max(left1, left2) + min(right1, right2)) / 2.0

else:

return max(left1, left2)

elif left1 > right2:

high = cut1 - 1

else:

low = cut1 + 1

return 0.0

# Example usage

nums1 = [1, 3]

nums2 = [2]

print("Median:", find\_median\_sorted\_arrays(nums1, nums2))

**🎉 Congratulations! 🎉**

You've successfully completed this **Array & Searching** problem set! 🚀 I hope these questions have strengthened your problem-solving skills and given you a deeper understanding of key concepts.

If you found this helpful, feel free to **connect with me on Instagram** for more coding insights, problem discussions, and full-stack development content!

📌 **Instagram:** [SYNTAX\_ERROR](https://www.instagram.com/SYNTAX_ERROR)

**Happy Coding! Keep Learning & Keep Growing!** 💡🔥