**Top 10 Sorting Questions Asked in Google, Amazon, Flipkart, Wipro, and TCS**

📌 1. Sort an Array of 0s, 1s, and 2s (Dutch National Flag Algorithm)  
*Asked in: Amazon, Flipkart, Wipro*

* Given an array containing only 0s, 1s, and 2s, sort it in-place without using extra space.

📌 2. Kth Smallest and Kth Largest Element in an Array  
*Asked in: Google, Amazon, TCS*

* Given an array and an integer K, find the Kth smallest and Kth largest element efficiently using QuickSelect or Heap.

📌 3. Merge Two Sorted Arrays Without Extra Space  
*Asked in: Flipkart, Wipro, TCS*

* Given two sorted arrays, merge them into a single sorted array without using extra space.

📌 4. Count Inversions in an Array (Using Merge Sort)  
*Asked in: Amazon, Google, TCS*

* An inversion is a pair (i, j) such that arr[i] > arr[j] for i < j. Find the total number of inversions efficiently.

📌 5. Find the Minimum Difference Between Any Two Elements  
*Asked in: Google, Flipkart, Amazon*

Given an unsorted array, find the minimum absolute difference between any two elements using sorting.

📌 6. Find the Median of Two Sorted Arrays (O(log(N+M)) time complexity)  
*Asked in: Google, Amazon, Wipro*

* Given two sorted arrays of sizes N and M, find the median of the merged sorted array efficiently using binary search.

📌 7. Sort a Nearly Sorted (K-Sorted) Array  
*Asked in: Amazon, Flipkart, Wipro*

* Given an array where each element is at most K positions away from its correct position, sort it efficiently using a min-heap.

📌 8. Find the Largest Number That Can Be Formed from an Array of Numbers  
*Asked in: Google, Amazon, TCS*

* Given an array of numbers, arrange them to form the largest possible number as a string.

📌 9. Find the Minimum Number of Swaps Required to Sort an Array  
*Asked in: Flipkart, Amazon, Wipro*

* Given an unsorted array, find the minimum number of swaps required to sort it using cycle detection

📌 10. Find the Intersection of Two Sorted Arrays  
*Asked in: Google, TCS, Flipkart*

* Given two sorted arrays, find the intersection (common elements) in an optimal way.

📌 1. Sort an Array of 0s, 1s, and 2s (Dutch National Flag Algorithm)  
*Asked in: Amazon, Flipkart, Wipro*

* Given an array containing only 0s, 1s, and 2s, sort it in-place without using extra space.

C++ Solution

#include <iostream>

#include <vector>

using namespace std;

void sortColors(vector<int>& nums) {

int low = 0, mid = 0, high = nums.size() - 1;

while (mid <= high) {

if (nums[mid] == 0) {

swap(nums[low], nums[mid]);

low++, mid++;

} else if (nums[mid] == 1) {

mid++;

} else {

swap(nums[mid], nums[high]);

high--;

}

}

}

int main() {

vector<int> arr = {2, 0, 2, 1, 1, 0};

sortColors(arr);

for (int num : arr) {

cout << num << " ";

}

return 0;

}

Java Solution

import java.util.Arrays;

public class SortColors {

public static void sortColors(int[] nums) {

int low = 0, mid = 0, high = nums.length - 1;

while (mid <= high) {

if (nums[mid] == 0) {

int temp = nums[low];

nums[low] = nums[mid];

nums[mid] = temp;

low++;

mid++;

} else if (nums[mid] == 1) {

mid++;

} else {

int temp = nums[mid];

nums[mid] = nums[high];

nums[high] = temp;

high--;

}

}

}

public static void main(String[] args) {

int[] arr = {2, 0, 2, 1, 1, 0};

sortColors(arr);

System.out.println(Arrays.toString(arr));

}

}

Python Solution

def sortColors(nums):

low, mid, high = 0, 0, len(nums) - 1

while mid <= high:

if nums[mid] == 0:

nums[low], nums[mid] = nums[mid], nums[low]

low += 1

mid += 1

elif nums[mid] == 1:

mid += 1

else:

nums[mid], nums[high] = nums[high], nums[mid]

high -= 1

arr = [2, 0, 2, 1, 1, 0]

sortColors(arr)

print(arr)

📌 2. Kth Smallest and Kth Largest Element in an Array  
*Asked in: Google, Amazon, TCS*

* Given an array and an integer K, find the Kth smallest and Kth largest element efficiently using QuickSelect or Heap.

**C++ Solution** (Using QuickSelect & Heap)

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

// QuickSelect for Kth Smallest

int quickSelect(vector<int>& nums, int left, int right, int k) {

int pivot = nums[right], pIndex = left;

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

if (nums[i] <= pivot) {

swap(nums[i], nums[pIndex++]);

}

}

swap(nums[pIndex], nums[right]);

if (pIndex == k) return nums[pIndex];

else if (pIndex < k) return quickSelect(nums, pIndex + 1, right, k);

else return quickSelect(nums, left, pIndex - 1, k);

}

// Kth Smallest using Min Heap

int kthSmallestHeap(vector<int>& nums, int k) {

priority\_queue<int, vector<int>, greater<int>> minHeap(nums.begin(), nums.end());

while (--k) minHeap.pop();

return minHeap.top();

}

// Kth Largest using Max Heap

int kthLargestHeap(vector<int>& nums, int k) {

priority\_queue<int> maxHeap(nums.begin(), nums.end());

while (--k) maxHeap.pop();

return maxHeap.top();

}

int main() {

vector<int> arr = {7, 10, 4, 3, 20, 15};

int k = 3;

cout << "Kth Smallest (QuickSelect): " << quickSelect(arr, 0, arr.size() - 1, k - 1) << endl;

cout << "Kth Smallest (Min Heap): " << kthSmallestHeap(arr, k) << endl;

cout << "Kth Largest (Max Heap): " << kthLargestHeap(arr, k) << endl;

return 0;

}

**Java Solution** (Using QuickSelect & Heap)

import java.util.PriorityQueue;

import java.util.Arrays;

public class KthElement {

// QuickSelect for Kth Smallest

public static int quickSelect(int[] nums, int left, int right, int k) {

int pivot = nums[right], pIndex = left;

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

if (nums[i] <= pivot) {

int temp = nums[i];

nums[i] = nums[pIndex];

nums[pIndex] = temp;

pIndex++;

}

}

int temp = nums[pIndex];

nums[pIndex] = nums[right];

nums[right] = temp;

if (pIndex == k) return nums[pIndex];

else if (pIndex < k) return quickSelect(nums, pIndex + 1, right, k);

else return quickSelect(nums, left, pIndex - 1, k);

}

// Kth Smallest using Min Heap

public static int kthSmallestHeap(int[] nums, int k) {

PriorityQueue<Integer> minHeap = new PriorityQueue<>();

for (int num : nums) minHeap.add(num);

while (--k > 0) minHeap.poll();

return minHeap.peek();

}

// Kth Largest using Max Heap

public static int kthLargestHeap(int[] nums, int k) {

PriorityQueue<Integer> maxHeap = new PriorityQueue<>((a, b) -> b - a);

for (int num : nums) maxHeap.add(num);

while (--k > 0) maxHeap.poll();

return maxHeap.peek();

}

public static void main(String[] args) {

int[] arr = {7, 10, 4, 3, 20, 15};

int k = 3;

System.out.println("Kth Smallest (QuickSelect): " + quickSelect(Arrays.copyOf(arr, arr.length), 0, arr.length - 1, k - 1));

System.out.println("Kth Smallest (Min Heap): " + kthSmallestHeap(arr, k));

System.out.println("Kth Largest (Max Heap): " + kthLargestHeap(arr, k));

}

}

**Python Solution** (Using QuickSelect & Heap)

import heapq

# QuickSelect for Kth Smallest

def quickSelect(arr, left, right, k):

pivot = arr[right]

pIndex = left

for i in range(left, right):

if arr[i] <= pivot:

arr[i], arr[pIndex] = arr[pIndex], arr[i]

pIndex += 1

arr[pIndex], arr[right] = arr[right], arr[pIndex]

if pIndex == k:

return arr[pIndex]

elif pIndex < k:

return quickSelect(arr, pIndex + 1, right, k)

else:

return quickSelect(arr, left, pIndex - 1, k)

# Kth Smallest using Min Heap

def kthSmallestHeap(arr, k):

return heapq.nsmallest(k, arr)[-1]

# Kth Largest using Max Heap

def kthLargestHeap(arr, k):

return heapq.nlargest(k, arr)[-1]

arr = [7, 10, 4, 3, 20, 15]

k = 3

print("Kth Smallest (QuickSelect):", quickSelect(arr[:], 0, len(arr) - 1, k - 1))

print("Kth Smallest (Min Heap):", kthSmallestHeap(arr, k))

print("Kth Largest (Max Heap):", kthLargestHeap(arr, k))

📌 3. Merge Two Sorted Arrays Without Extra Space  
*Asked in: Flipkart, Wipro, TCS*

* Given two sorted arrays, merge them into a single sorted array without using extra space.

**C++ Solution** (Using Gap Method - O((N+M) log(N+M)))

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

// Function to merge two sorted arrays without extra space

void merge(vector<int>& arr1, vector<int>& arr2) {

int n = arr1.size(), m = arr2.size();

int gap = (n + m + 1) / 2;

while (gap > 0) {

int i = 0, j = gap;

while (j < (n + m)) {

if (j < n && arr1[i] > arr1[j]) {

swap(arr1[i], arr1[j]);

}

else if (j >= n && i < n && arr1[i] > arr2[j - n]) {

swap(arr1[i], arr2[j - n]);

}

else if (j >= n && i >= n && arr2[i - n] > arr2[j - n]) {

swap(arr2[i - n], arr2[j - n]);

}

i++, j++;

}

if (gap == 1) break;

gap = (gap + 1) / 2;

}

}

int main() {

vector<int> arr1 = {1, 3, 5, 7};

vector<int> arr2 = {2, 4, 6, 8};

merge(arr1, arr2);

for (int num : arr1) cout << num << " ";

for (int num : arr2) cout << num << " ";

return 0;

}

**Java Solution** (Using Gap Method - O((N+M) log(N+M)))

import java.util.Arrays;

public class MergeSortedArrays {

public static void merge(int[] arr1, int[] arr2) {

int n = arr1.length, m = arr2.length;

int gap = (n + m + 1) / 2;

while (gap > 0) {

int i = 0, j = gap;

while (j < (n + m)) {

if (j < n && arr1[i] > arr1[j]) {

int temp = arr1[i];

arr1[i] = arr1[j];

arr1[j] = temp;

}

else if (j >= n && i < n && arr1[i] > arr2[j - n]) {

int temp = arr1[i];

arr1[i] = arr2[j - n];

arr2[j - n] = temp;

}

else if (j >= n && i >= n && arr2[i - n] > arr2[j - n]) {

int temp = arr2[i - n];

arr2[i - n] = arr2[j - n];

arr2[j - n] = temp;

}

i++;

j++;

}

if (gap == 1) break;

gap = (gap + 1) / 2;

}

}

public static void main(String[] args) {

int[] arr1 = {1, 3, 5, 7};

int[] arr2 = {2, 4, 6, 8};

merge(arr1, arr2);

System.out.println(Arrays.toString(arr1) + " " + Arrays.toString(arr2));

}

}

**Python Solution** (Using Gap Method - O((N+M) log(N+M)))

def merge(arr1, arr2):

n, m = len(arr1), len(arr2)

gap = (n + m + 1) // 2

while gap > 0:

i, j = 0, gap

while j < (n + m):

if j < n and arr1[i] > arr1[j]:

arr1[i], arr1[j] = arr1[j], arr1[i]

elif j >= n and i < n and arr1[i] > arr2[j - n]:

arr1[i], arr2[j - n] = arr2[j - n], arr1[i]

elif j >= n and i >= n and arr2[i - n] > arr2[j - n]:

arr2[i - n], arr2[j - n] = arr2[j - n], arr2[i - n]

i += 1

j += 1

if gap == 1:

break

gap = (gap + 1) // 2

arr1 = [1, 3, 5, 7]

arr2 = [2, 4, 6, 8]

merge(arr1, arr2)

print(arr1, arr2)

📌 4. Count Inversions in an Array (Using Merge Sort)  
*Asked in: Amazon, Google, TCS*

* An inversion is a pair (i, j) such that arr[i] > arr[j] for i < j. Find the total number of inversions efficiently.

C++ Solution (Using Merge Sort - O(N log N))

#include <iostream>

#include <vector>

using namespace std;

long long merge(vector<int>& arr, int left, int mid, int right) {

vector<int> temp;

int i = left, j = mid + 1;

long long inv\_count = 0;

while (i <= mid && j <= right) {

if (arr[i] <= arr[j]) {

temp.push\_back(arr[i++]);

} else {

temp.push\_back(arr[j++]);

inv\_count += (mid - i + 1); // Counting inversions

}

}

while (i <= mid) temp.push\_back(arr[i++]);

while (j <= right) temp.push\_back(arr[j++]);

for (int k = left; k <= right; k++) arr[k] = temp[k - left];

return inv\_count;

}

long long mergeSort(vector<int>& arr, int left, int right) {

if (left >= right) return 0;

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

long long inv\_count = mergeSort(arr, left, mid);

inv\_count += mergeSort(arr, mid + 1, right);

inv\_count += merge(arr, left, mid, right);

return inv\_count;

}

int main() {

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

cout << "Inversion count: " << mergeSort(arr, 0, arr.size() - 1) << endl;

return 0;

### }

### Java Solution (Using Merge Sort - O(N log N))

public class CountInversions {

public static long merge(int[] arr, int left, int mid, int right) {

int[] temp = new int[right - left + 1];

int i = left, j = mid + 1, k = 0;

long inv\_count = 0;

while (i <= mid && j <= right) {

if (arr[i] <= arr[j]) {

temp[k++] = arr[i++];

} else {

temp[k++] = arr[j++];

inv\_count += (mid - i + 1); // Counting inversions

}

}

while (i <= mid) temp[k++] = arr[i++];

while (j <= right) temp[k++] = arr[j++];

System.arraycopy(temp, 0, arr, left, temp.length);

return inv\_count;

}

public static long mergeSort(int[] arr, int left, int right) {

if (left >= right) return 0;

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

long inv\_count = mergeSort(arr, left, mid);

inv\_count += mergeSort(arr, mid + 1, right);

inv\_count += merge(arr, left, mid, right);

return inv\_count;

}

public static void main(String[] args) {

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

System.out.println("Inversion count: " + mergeSort(arr, 0, arr.length - 1));

}

}

**Python Solution (Using Merge Sort - O(N log N))**

def merge(arr, temp, left, mid, right):

i, j, k = left, mid + 1, left

inv\_count = 0

while i <= mid and j <= right:

if arr[i] <= arr[j]:

temp[k] = arr[i]

i += 1

else:

temp[k] = arr[j]

inv\_count += (mid - i + 1) # Counting inversions

j += 1

k += 1

while i <= mid:

temp[k] = arr[i]

i += 1

k += 1

while j <= right:

temp[k] = arr[j]

j += 1

k += 1

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

arr[i] = temp[i]

return inv\_count

def merge\_sort(arr, temp, left, right):

if left >= right:

return 0

mid = (left + right) // 2

inv\_count = merge\_sort(arr, temp, left, mid)

inv\_count += merge\_sort(arr, temp, mid + 1, right)

inv\_count += merge(arr, temp, left, mid, right)

return inv\_count

arr = [8, 4, 2, 1]

temp = arr[:]

print("Inversion count:", merge\_sort(arr, temp, 0, len(arr) - 1))

📌 5. Find the Minimum Difference Between Any Two Elements  
*Asked in: Google, Flipkart, Amazon*

Given an unsorted array, find the minimum absolute difference between any two elements using sorting.

**C++ Solution (O(N log N))**

cpp

CopyEdit

#include <iostream>

#include <vector>

#include <algorithm>

#include <climits>

using namespace std;

int findMinDifference(vector<int>& arr) {

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

int minDiff = INT\_MAX;

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

minDiff = min(minDiff, arr[i] - arr[i - 1]);

}

return minDiff;

}

int main() {

vector<int> arr = {3, 8, 15, 2, 1};

cout << "Minimum Difference: " << findMinDifference(arr) << endl;

return 0;

}

**Java Solution (O(N log N))**

java

CopyEdit

import java.util.Arrays;

public class MinDifference {

public static int findMinDifference(int[] arr) {

Arrays.sort(arr);

int minDiff = Integer.MAX\_VALUE;

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

minDiff = Math.min(minDiff, arr[i] - arr[i - 1]);

}

return minDiff;

}

public static void main(String[] args) {

int[] arr = {3, 8, 15, 2, 1};

System.out.println("Minimum Difference: " + findMinDifference(arr));

}

}

**Python Solution (O(N log N))**

python

CopyEdit

def find\_min\_difference(arr):

arr.sort()

min\_diff = float('inf')

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

min\_diff = min(min\_diff, arr[i] - arr[i - 1])

return min\_diff

arr = [3, 8, 15, 2, 1]

print("Minimum Difference:", find\_min\_difference(arr))

📌 6. Find the Median of Two Sorted Arrays (O(log(N+M)) time complexity)  
*Asked in: Google, Amazon, Wipro*

* Given two sorted arrays of sizes N and M, find the median of the merged sorted array efficiently using binary search.

**C++ Solution (O(log(N + M)) Using Binary Search)**

cpp

CopyEdit

#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); // Ensure nums1 is the smaller array

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; // Should never reach here

}

int main() {

vector<int> nums1 = {1, 3, 8};

vector<int> nums2 = {7, 9, 10, 11};

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

return 0;

}

**Java Solution (O(log(N + M)) Using Binary Search)**

java

CopyEdit

public class MedianSortedArrays {

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

if (nums1.length > nums2.length)

return findMedianSortedArrays(nums2, nums1); // Ensure nums1 is the smaller array

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; // Should never reach here

}

public static void main(String[] args) {

int[] nums1 = {1, 3, 8};

int[] nums2 = {7, 9, 10, 11};

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

}

}

**Python Solution (O(log(N + M)) Using Binary Search)**

python

CopyEdit

def findMedianSortedArrays(nums1, nums2):

if len(nums1) > len(nums2):

return findMedianSortedArrays(nums2, nums1) # Ensure nums1 is the smaller array

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 # Should never reach here

nums1 = [1, 3, 8]

nums2 = [7, 9, 10, 11]

print("Median:", findMedianSortedArrays(nums1, nums2))

📌 7. Sort a Nearly Sorted (K-Sorted) Array  
*Asked in: Amazon, Flipkart, Wipro*

* Given an array where each element is at most K positions away from its correct position, sort it efficiently using a min-heap.

### C++ Solution (Using Min-Heap - O(N log K))

cpp

CopyEdit

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

void sortKSortedArray(vector<int>& arr, int k) {

priority\_queue<int, vector<int>, greater<int>> minHeap;

int index = 0;

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

minHeap.push(arr[i]);

if (minHeap.size() > k) {

arr[index++] = minHeap.top();

minHeap.pop();

}

}

while (!minHeap.empty()) {

arr[index++] = minHeap.top();

minHeap.pop();

}

}

int main() {

vector<int> arr = {6, 5, 3, 2, 8, 10, 9};

int k = 3;

sortKSortedArray(arr, k);

cout << "Sorted Array: ";

for (int num : arr) {

cout << num << " ";

}

return 0;

}

**Java Solution (Using Min-Heap - O(N log K))**

java

CopyEdit

import java.util.PriorityQueue;

public class KSortedArray {

public static void sortKSortedArray(int[] arr, int k) {

PriorityQueue<Integer> minHeap = new PriorityQueue<>();

int index = 0;

for (int num : arr) {

minHeap.add(num);

if (minHeap.size() > k) {

arr[index++] = minHeap.poll();

}

}

while (!minHeap.isEmpty()) {

arr[index++] = minHeap.poll();

}

}

public static void main(String[] args) {

int[] arr = {6, 5, 3, 2, 8, 10, 9};

int k = 3;

sortKSortedArray(arr, k);

System.out.print("Sorted Array: ");

for (int num : arr) {

System.out.print(num + " ");

}

}

}

**Python Solution (Using Min-Heap - O(N log K))**

python

CopyEdit

import heapq

def sort\_k\_sorted\_array(arr, k):

min\_heap = []

index = 0

for num in arr:

heapq.heappush(min\_heap, num)

if len(min\_heap) > k:

arr[index] = heapq.heappop(min\_heap)

index += 1

while min\_heap:

arr[index] = heapq.heappop(min\_heap)

index += 1

arr = [6, 5, 3, 2, 8, 10, 9]

k = 3

sort\_k\_sorted\_array(arr, k)

print("Sorted Array:", arr)

📌 8. Find the Largest Number That Can Be Formed from an Array of Numbers  
*Asked in: Google, Amazon, TCS*

* Given an array of numbers, arrange them to form the largest possible number as a string.

### C++ Solution (Using Custom Comparator - O(N log N))

cpp

CopyEdit

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

bool compare(string a, string b) {

return a + b > b + a;

}

string largestNumber(vector<int>& nums) {

vector<string> strNums;

for (int num : nums) {

strNums.push\_back(to\_string(num));

}

sort(strNums.begin(), strNums.end(), compare);

if (strNums[0] == "0") return "0"; // Handling case like [0,0]

string result = "";

for (string s : strNums) {

result += s;

}

return result;

}

int main() {

vector<int> nums = {3, 30, 34, 5, 9};

cout << "Largest Number: " << largestNumber(nums) << endl;

return 0;

}

**Java Solution (Using Custom Comparator - O(N log N))**

java

CopyEdit

import java.util.Arrays;

import java.util.Comparator;

public class LargestNumber {

public static String largestNumber(int[] nums) {

String[] strNums = new String[nums.length];

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

strNums[i] = String.valueOf(nums[i]);

}

Arrays.sort(strNums, (a, b) -> (b + a).compareTo(a + b));

if (strNums[0].equals("0")) return "0"; // Handling case like [0,0]

StringBuilder result = new StringBuilder();

for (String s : strNums) {

result.append(s);

}

return result.toString();

}

public static void main(String[] args) {

int[] nums = {3, 30, 34, 5, 9};

System.out.println("Largest Number: " + largestNumber(nums));

}

}

**Python Solution (Using Custom Comparator - O(N log N))**

python

CopyEdit

from functools import cmp\_to\_key

def compare(a, b):

return 1 if a + b < b + a else -1

def largest\_number(nums):

str\_nums = list(map(str, nums))

str\_nums.sort(key=cmp\_to\_key(compare))

if str\_nums[0] == "0":

return "0" # Handling case like [0,0]

return "".join(str\_nums)

nums = [3, 30, 34, 5, 9]

print("Largest Number:", largest\_number(nums))

📌 9. Find the Minimum Number of Swaps Required to Sort an Array  
*Asked in: Flipkart, Amazon, Wipro*

* Given an unsorted array, find the minimum number of swaps required to sort it using cycle detection

### C++ Solution (Using Cycle Detection - O(N log N))

cpp

CopyEdit

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int minSwaps(vector<int>& nums) {

int n = nums.size();

vector<pair<int, int>> arr(n);

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

arr[i] = {nums[i], i}; // Storing value and original index

}

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

vector<bool> visited(n, false);

int swaps = 0;

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

if (visited[i] || arr[i].second == i) continue;

int cycle\_size = 0;

int j = i;

while (!visited[j]) {

visited[j] = true;

j = arr[j].second;

cycle\_size++;

}

if (cycle\_size > 1) swaps += (cycle\_size - 1);

}

return swaps;

}

int main() {

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

cout << "Minimum Swaps: " << minSwaps(nums) << endl;

return 0;

}

**Java Solution (Using Cycle Detection - O(N log N))**

java

CopyEdit

import java.util.Arrays;

import java.util.Comparator;

public class MinSwaps {

public static int minSwaps(int[] nums) {

int n = nums.length;

int[][] arr = new int[n][2];

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

arr[i][0] = nums[i];

arr[i][1] = i;

}

Arrays.sort(arr, Comparator.comparingInt(a -> a[0]));

boolean[] visited = new boolean[n];

int swaps = 0;

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

if (visited[i] || arr[i][1] == i) continue;

int cycle\_size = 0;

int j = i;

while (!visited[j]) {

visited[j] = true;

j = arr[j][1];

cycle\_size++;

}

if (cycle\_size > 1) swaps += (cycle\_size - 1);

}

return swaps;

}

public static void main(String[] args) {

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

System.out.println("Minimum Swaps: " + minSwaps(nums));

}

}

**Python Solution (Using Cycle Detection - O(N log N))**

python

CopyEdit

def min\_swaps(nums):

n = len(nums)

arr = [(num, i) for i, num in enumerate(nums)]

arr.sort()

visited = [False] \* n

swaps = 0

for i in range(n):

if visited[i] or arr[i][1] == i:

continue

cycle\_size = 0

j = i

while not visited[j]:

visited[j] = True

j = arr[j][1]

cycle\_size += 1

if cycle\_size > 1:

swaps += (cycle\_size - 1)

return swaps

nums = [4, 3, 2, 1]

print("Minimum Swaps:", min\_swaps(nums))

📌 10. Find the Intersection of Two Sorted Arrays  
*Asked in: Google, TCS, Flipkart*

* Given two sorted arrays, find the intersection (common elements) in an optimal way.

### C++ Solution (Two Pointer Approach - O(N + M))

cpp

CopyEdit

#include <iostream>

#include <vector>

using namespace std;

vector<int> intersection(vector<int>& arr1, vector<int>& arr2) {

int i = 0, j = 0;

vector<int> result;

while (i < arr1.size() && j < arr2.size()) {

if (arr1[i] < arr2[j]) {

i++;

} else if (arr1[i] > arr2[j]) {

j++;

} else {

if (result.empty() || result.back() != arr1[i]) {

result.push\_back(arr1[i]);

}

i++; j++;

}

}

return result;

}

int main() {

vector<int> arr1 = {1, 2, 2, 3, 4, 5};

vector<int> arr2 = {2, 2, 4, 6};

vector<int> res = intersection(arr1, arr2);

cout << "Intersection: ";

for (int num : res) {

cout << num << " ";

}

cout << endl;

return 0;

}

**Java Solution (Two Pointer Approach - O(N + M))**

java

CopyEdit

import java.util.\*;

public class ArrayIntersection {

public static List<Integer> intersection(int[] arr1, int[] arr2) {

List<Integer> result = new ArrayList<>();

int i = 0, j = 0;

while (i < arr1.length && j < arr2.length) {

if (arr1[i] < arr2[j]) {

i++;

} else if (arr1[i] > arr2[j]) {

j++;

} else {

if (result.isEmpty() || result.get(result.size() - 1) != arr1[i]) {

result.add(arr1[i]);

}

i++;

j++;

}

}

return result;

}

public static void main(String[] args) {

int[] arr1 = {1, 2, 2, 3, 4, 5};

int[] arr2 = {2, 2, 4, 6};

List<Integer> res = intersection(arr1, arr2);

System.out.println("Intersection: " + res);

}

}

**Python Solution (Two Pointer Approach - O(N + M))**

python

CopyEdit

def intersection(arr1, arr2):

i, j = 0, 0

result = []

while i < len(arr1) and j < len(arr2):

if arr1[i] < arr2[j]:

i += 1

elif arr1[i] > arr2[j]:

j += 1

else:

if not result or result[-1] != arr1[i]:

result.append(arr1[i])

i += 1

j += 1

return result

arr1 = [1, 2, 2, 3, 4, 5]

arr2 = [2, 2, 4, 6]

print("Intersection:", intersection(arr1, arr2))

## 🎉 **Congratulations on Completing This PDF!** 🎉

You’ve taken a big step in mastering **Arrays, Searching, and Sorting**—some of the most crucial topics for coding interviews! 🚀 Keep practicing and refining your problem-solving skills.

If you found this helpful, stay connected with me for more coding content, problem-solving techniques, and DSA tips! Follow me on Instagram:

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

Happy Coding! 💻🔥