<https://www.youtube.com/watch?v=0HBIMjwte7s&list=PLUg9hRlm7gxR7e9OgtrsqgN7XEyNTZRPK>

Print Duplicate Element in an Array

Find Pairs with Given Sum in a Sorted Array

Find Common Elements in Two Array

Find Common Elements in Three Sorted Array

Delete an element from an Unsorted Array at a specified position

Segregate 0s and 1s in an Array

OR

Shift all 0s to left hand side and 1s to right hand side

OR

Shift all 0s to right hand side and 1s to left hand side

Remove Duplicate Elements from Unsorted Array

Find Second Smallest Number in an Array

Two Sum problem or Find a pair in an array with given sum

Merge two sorted array into a third array

Find all duplicate elements in array (all approaches)

Sort an Array of 0s 1s 2s

Find Maximum difference between Two Array Elements

Find Minimum Element in a Sorted and Rotated Array

How to rotate an array in both directions by K steps(31)

Find the Element that appears once in an Array

Reverse an Array

Maximum Subarray Sum

Find Maximum Sum of a Subarray of Size K

Find Top K most Frequent Elements

Find a peak element in an Array

Find majority element in an Array

Find second largest number in an array (multiple approaches)

Find first and last index of an element in a Sorted Array

Find length of longest increasing subarray

Find triplet with given sum in an array

Find triplets with Aero Sum (3 sum in java)

Sort an Array of 0s 1s 2s in Single Traversal

Search an Element in a Sorted and Rotated Array

Group Anagrams Together

Count Occurrences/Frequency of number in a sorted Array

Maximum Consecutive 1s in an array of 0s and 1s

Reverse Level Order Traversal of a Binary Tree

Maximum Consecutive Ones (If K Flip is Allowed)

Product of Array Except Self (Leetcode 238)

Find all numbers disappeared in an Array

Merge Overlapping Intervals

Minimum Size Subarray Sum Minimum Size Subarray Sum (Smallest Subarray with Sum Greater than or Equal to the Given Value)

Find the element that appears once in a Sorted Array in O(logn)

Find the duplicate Number in an Array without using Extra space/memory

Find First and Last Position of Element in Sorted Array

Sort Array By Parity

Contiguous Array

Plus One

**Print Duplicate Element in an Array**

**package** com.tutorial.basics;

**import** java.util.HashMap;

**import** java.util.HashSet;

**import** java.util.Iterator;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Set;

**public** **class** FindDuplicateElement {

**public** **static** **void** main(String[] args) {

**int**[] arr = { 1, 4, 5, 5, 4, 2, 1, 7 };

// Using Brute Force

// Using Set

// Using Map

*usingBruteForce*(arr);

System.***out***.println();

*usingSet*(arr);

System.***out***.println();

*usingMap*(arr);

}

**public** **static** **void** usingBruteForce(**int**[] arr) {

**int** n = arr.length;

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

**for** (**int** j = i + 1; j < n; j++) {

**if** (arr[i] == arr[j] && i != j) {

System.***out***.print(arr[j] + " ");

**break**;

}

}

}

}

**public** **static** **void** usingSet(**int**[] input) {

Set<Integer> duplicates = **new** HashSet<Integer>();

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

**for** (**int** j = 1; j < input.length; j++) {

**if** (input[i] == input[j] && i != j) {

// duplicate element found

duplicates.add(input[i]);

**break**;

}

}

}

duplicates.forEach(result -> System.***out***.print(result + " "));

}

**public** **static** **void** usingMap(**int**[] arr) {

**int** n = arr.length;

Map<Integer, Integer> map = **new** HashMap<>();

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

**int** num = arr[i];

**if** (map.containsKey(num)) {

map.put(num, map.get(num) + 1);

} **else** {

map.put(num, 1);

}

}

Set<Entry<Integer, Integer>> entrySet = map.entrySet();

Iterator<Entry<Integer, Integer>> iterator = entrySet.iterator();

**while** (iterator.hasNext()) {

Entry<Integer, Integer> entry = iterator.next();

Integer key = entry.getKey();

Integer value = entry.getValue();

**if** (value > 1)

System.***out***.print(key + " ");

}

}

}

**Find Pairs with Given Sum in a Sorted Array (Complexity O(n))**

**package** com.tutorial.basics;

**public** **class** PairsWithGivenSumInASortedArray {

**public** **static** **void** main(String[] args) {

// sorted array

**int**[] arr = { 1, 2, 3, 4, 5, 6, 7 };

// int sum = 9;

**int** sum = 8;

// pairs {2,7}, {3,6}, {4,5}

*findPair*(arr, sum);

}

**public** **static** **void** findPair(**int**[] arr, **int** sum) {

**int** low = 0;

**int** high = arr.length - 1;

**while** (low < high) {

// for (int i = 0; i < arr.length/2; i++) {

**if** (arr[low] + arr[high] > sum) {

high--;

} **else** **if** (arr[low] + arr[high] < sum) {

low++;

} **else** **if** (arr[low] + arr[high] == sum) {

System.***out***.println(arr[low] + " " + arr[high]);

low++;

high--;

}

// }

}

}

}

**Find Common Elements in Two Array (Complexity O(n^2)**

**package** com.tutorial.basics;

**public** **class** CommonElementInTwoArray {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int**[] arr1 = { 4, 3, 7, 9, 2 };

**int**[] arr2 = { 5, 1, 4, 8, 3 };

*findCommonElementInTwoArrays*(arr1, arr2);

}

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

**for** (**int** i = 0; i < arr1.length; i++) {

**for** (**int** j = 0; j < arr2.length; j++) {

**if** (arr1[i] == arr2[j]) {

System.***out***.print(arr1[i] + " ");

}

}

}

}

}

**Find Common Elements in Three Sorted Array**

**package** com.tutorial.basics;

**public** **class** CommonElementsInThreeSortedArray {

**public** **static** **void** main(String[] args) {

// sorted array

**int**[] arr1 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };

**int**[] arr2 = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };

**int**[] arr3 = { 5, 10, 15, 20 };

*findCommonElementsInThreeSortedArray*(arr1, arr2, arr3);

System.***out***.println();

*findCommonElementsinSortedArray*(arr1, arr2, arr3);

}

// O(n^3)

**public** **static** **void** findCommonElementsInThreeSortedArray(**int**[] arr1, **int**[] arr2, **int**[] arr3) {

**for** (**int** i = 0; i < arr1.length; i++) {

**for** (**int** j = 0; j < arr2.length; j++) {

**for** (**int** k = 0; k < arr3.length; k++) {

**if** (arr1[i] == arr2[j] && arr2[j] == arr3[k]) {

System.***out***.print(arr1[i] + " ");

}

}

}

}

}

// O(n)

**public** **static** **void** findCommonElementsinSortedArray(**int**[] arr1, **int**[] arr2, **int**[] arr3) {

**int** i = 0;

**int** j = 0;

**int** k = 0;

**while** (i < arr1.length && j < arr2.length && k <arr3.length) {

**if**(arr1[i]==arr2[j] && arr2[j]==arr3[k]) {

System.***out***.print(arr1[i] + " ");

i++;

j++;

k++;

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

j++;

}**else** **if**(arr2[j]>arr3[k]) {

k++;

}**else** {

i++;

}

}

}

}

**Delete an element from an Unsorted Array at a specified position**

**package** com.tutorial.basics;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.stream.Collectors;

**import** java.util.stream.IntStream;

// https://www.geeksforgeeks.org/remove-an-element-at-specific-index-from-an-array-in-java/

**public** **class** DeleteAnElementFromUnsortedArray {

**public** **static** **void** main(String[] args) {

**int**[] arr1 = { 1, 5, 8, 9, 4 };

**int**[] arr2 = { 1, 3, 9, 4, 5 };

**int**[] arr3 = { 1, 0, 9, 2, 5 };

**int**[] arr4 = { 1, 0, 8, 9, 9 };

**int** index = 2;

System.***out***.println(Arrays.*toString*(arr1));

arr1 = *deleteElementUsingCopyingElementsToNewArray*(arr1, index);

System.***out***.println(Arrays.*toString*(arr1));

System.***out***.println(Arrays.*toString*(arr2));

arr2 = *deleteElementUsingJava8Streams*(arr2, index);

System.***out***.println(Arrays.*toString*(arr2));

System.***out***.println(Arrays.*toString*(arr3));

arr3 = *deleteElementUsingArrayListAndStream*(arr3, index);

System.***out***.println(Arrays.*toString*(arr3));

System.***out***.println(Arrays.*toString*(arr4));

arr4 = *deleteElementUsingSystemArrayCopy*(arr4, index);

System.***out***.println(Arrays.*toString*(arr4));

}

// 1. Naive Approach/Basic Approach

**public** **static** **int**[] deleteElementUsingCopyingElementsToNewArray(**int**[] arr, **int** index) {

**int** n = arr.length;

**if** (arr == **null** || index < 0 || arr.length < index) {

**return** arr;

}

**int**[] newArr = **new** **int**[n - 1];

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

**if** (i == index) {

**continue**;

}

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

}

**return** newArr;

}

// 2. Java 8 Streams

**public** **static** **int**[] deleteElementUsingJava8Streams(**int**[] arr, **int** index) {

**int** n = arr.length;

**if** (arr == **null** || index < 0 || arr.length < index) {

**return** arr;

}

**return** IntStream.*range*(0, n).filter(i -> i != index).map(i -> arr[i]).toArray();

}

// 3. Using ArrayList

**public** **static** **int**[] deleteElementUsingArrayListAndStream(**int**[] arr, **int** index) {

**int** n = arr.length;

**if** (arr == **null** || index < 0 || arr.length < index) {

**return** arr;

}

List<Integer> list = IntStream.*of*(arr).boxed().collect(Collectors.*toList*());

list.remove(index);

**return** list.stream().mapToInt(Integer::intValue).toArray();

}

// 4. Using System.arraycopy

**public** **static** **int**[] deleteElementUsingSystemArrayCopy(**int**[] arr, **int** index) {

**int** n = arr.length;

**if** (arr == **null** || index < 0 || arr.length < index) {

**return** arr;

}

**int**[] newArr = **new** **int**[n - 1];

// System.arraycopy(src, srcPos, dest, destPos, length);

System.*arraycopy*(arr, 0, newArr, 0, index);

System.*arraycopy*(arr, index + 1, newArr, index, n - index - 1);

**return** newArr;

}

}

**Segregate 0s and 1s in an Array/ Shift all 0s to left hand side and 1s to right hand side /**

**Shift all 0s to right hand side and 1s to left hand side**

**package** com.tutorial.basics;

**import** java.util.Arrays;

**public** **class** Segregate0sAnd1sArray {

**public** **static** **void** main(String[] args) {

**int**[] arr1 = { 0, 1, 0, 1, 0, 0, 1 };

**int**[] arr2 = { 0, 1, 0, 1, 0, 0, 1 };

**int**[] arr3 = { 0, 1, 0, 1, 0, 0, 1 };

System.***out***.println(Arrays.*toString*(arr1));

arr1 = *countSort*(arr1);

System.***out***.println(Arrays.*toString*(arr1));

System.***out***.println(Arrays.*toString*(arr2));

arr2 = *segregate0sToLeftAnd1sToRight*(arr2);

System.***out***.println(Arrays.*toString*(arr2));

System.***out***.println(Arrays.*toString*(arr3));

arr3 = *segregate0sToRightAnd1sToLeft*(arr3);

System.***out***.println(Arrays.*toString*(arr3));

}

// 1. Using countSort Complexity O(n)

**public** **static** **int**[] countSort(**int**[] arr) {

**int** max = Arrays.*stream*(arr).max().getAsInt();

**int** min = Arrays.*stream*(arr).min().getAsInt();

**int** range = max - min + 1;

**int** count[] = **new** **int**[range];

**int** output[] = **new** **int**[arr.length];

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

count[arr[i] - min]++;

}

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

count[i] += count[i - 1];

}

**for** (**int** i = arr.length - 1; i >= 0; i--) {

output[count[arr[i] - min] - 1] = arr[i];

count[arr[i] - min]--;

}

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

arr[i] = output[i];

}

**return** arr;

}

// 2. Count Number of zeroes and insert ones at the end

**public** **static** **int**[] segregate0sToLeftAnd1sToRight(**int**[] arr) {

**int** j = 0;

**int** n = arr.length;

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

**if** (arr[i] == 0) {

arr[j++] = 0;

}

}

**while** (j < n) {

arr[j++] = 1;

}

**return** arr;

}

// 3. Count Number of ones and insert zeroes at the end

**public** **static** **int**[] segregate0sToRightAnd1sToLeft(**int**[] arr) {

**int** j = 0;

**int** n = arr.length;

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

**if** (arr[i] == 1) {

arr[j++] = 1;

}

}

**while** (j < n) {

arr[j++] = 0;

}

**return** arr;

}

}

**Remove Duplicate Elements from Unsorted Array**

**package** com.tutorial.basics;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.HashMap;

**import** java.util.HashSet;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.stream.Collectors;

**import** java.util.Set;

**import** java.util.TreeMap;

**public** **class** RemoveDuplicateElementsFromUnsortedArray {

**public** **static** **void** main(String[] args) {

**int**[] arr1 = { 9, -1, 9, -2, 4, 0, 0, -11, 6, 0, 6, -1, 8 };

**int**[] arr2 = { 9, -1, 9, -2, 4, 0, 0, -11, 6, 0, 6, -1, 8 };

**int**[] arr3 = { 9, -1, 9, -2, 4, 0, 0, -11, 6, 0, 6, -1, 8 };

**int**[] arr4 = { 9, -1, 9, -2, 4, 0, 0, -11, 6, 0, 6, -1, 8 };

System.***out***.println(Arrays.*toString*(arr1));

*removeDuplicateElementsBySorting*(arr1);

System.***out***.println(Arrays.*toString*(arr2));

System.***out***.println(Arrays.*toString*(*removeDuplicatesUsingJava8Stream*(arr2)));

System.***out***.println(Arrays.*toString*(arr3));

System.***out***.println(Arrays.*toString*(*removeDuplicatesUsingHashMap*(arr3)));

System.***out***.println(Arrays.*toString*(arr4));

System.***out***.println(Arrays.*toString*(*removeDuplicatesUsingSet*(arr4)));

}

// 1. Remove duplicates by Sorting and remove duplicates (Complexity (O(nlogn))

// sorted order

**public** **static** **void** removeDuplicateElementsBySorting(**int**[] arr) {

Arrays.*sort*(arr);

**int** n = arr.length;

**int**[] tempArr = **new** **int**[n];

**int** j = 0;

**for** (**int** i = 0; i < n - 1; i++) {

**if** (arr[i] != arr[i + 1]) {

tempArr[j++] = arr[i];

}

}

// last element of array

tempArr[j++] = arr[n - 1];

**for** (**int** i = 0; i < j; i++) {

arr[i] = tempArr[i];

}

**int**[] result = **new** **int**[j];

**for** (**int** i = 0; i < j; i++) {

result[i] = arr[i];

}

System.***out***.println(Arrays.*toString*(result));

}

// 2.Using HashMap Complexity (O(n))

**public** **static** **int**[] findDuplicatesUsingHashMap(**int**[] arr) {

**int** uniqueCount = 0;

**int** k = 0;

Map<Integer, Integer> map = **new** TreeMap<>();

**int** n = arr.length;

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

**int** num = arr[i];

**if** (map.containsKey(num)) {

map.put(num, map.get(num) + 1);

} **else** {

map.put(num, 1);

}

}

Set<Entry<Integer, Integer>> entrySet = map.entrySet();

Iterator<Entry<Integer, Integer>> itr = entrySet.iterator();

**while** (itr.hasNext()) {

Entry<Integer, Integer> entry = itr.next();

Integer key = entry.getKey();

Integer value = entry.getValue();

**if** (value > 1)

uniqueCount++;

}

**int**[] uniqueArr = **new** **int**[uniqueCount];

Set<Entry<Integer, Integer>> entrySet2 = map.entrySet();

Iterator<Entry<Integer, Integer>> itr2 = entrySet2.iterator();

**while** (itr2.hasNext()) {

Entry<Integer, Integer> entry2 = itr2.next();

Integer key = entry2.getKey();

Integer value = entry2.getValue();

**if** (value > 1)

uniqueArr[k++] = key;

}

**return** uniqueArr;

}

// 3. Remove duplicates using Stream

**public** **static** Integer[] removeDuplicatesUsingJava8Stream(**int**[] arr) {

**int** n = arr.length;

List<Integer> list = **new** ArrayList<>();

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

list.add(arr[i]);

}

List<Integer> uniqueList = list.stream().distinct().collect(Collectors.*toList*());

**return** uniqueList.toArray(**new** Integer[1]);

}

// Maintain insertion order

**public** **static** Integer[] removeDuplicatesUsingHashMap(**int**[] arr) {

Map<Integer, Integer> map = **new** HashMap<>();

**int** n = arr.length;

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

**int** num = arr[i];

**if** (map.containsKey(num)) {

map.put(num, map.get(num) + 1);

} **else** {

map.put(num, 1);

}

}

// map.forEach((k, v) -> System.out.print(k + " "));

Set<Integer> keySet = map.keySet();

Integer[] uniqueArr = keySet.toArray(**new** Integer[keySet.size()]);

**return** uniqueArr;

}

**public** **static** **int**[] removeDuplicatesUsingSet(**int**[] arr) {

Set<Integer> set = **new** HashSet<>();

**int** n = arr.length;

**for** (**int** i = 0; i < n - 1; i++) {

set.add(arr[i]);

}

**int** k = 0;

**int**[] uniqueArr = **new** **int**[set.size()];

**for** (**int** value : set) {

uniqueArr[k++] = value;

}

**return** uniqueArr;

}

}

**Find Second Smallest Number in an Array**

Two Sum problem or Find a pair in an array with given sum

Merge two sorted array into a third array

Find all duplicate elements in array (all approaches)

// 2.Using HashMap Complexity (O(n))

**public** **static** **int**[] findDuplicatesUsingHashMap(**int**[] arr) {

**int** uniqueCount = 0;

**int** k = 0;

Map<Integer, Integer> map = **new** TreeMap<>();

**int** n = arr.length;

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

**int** num = arr[i];

**if** (map.containsKey(num)) {

map.put(num, map.get(num) + 1);

} **else** {

map.put(num, 1);

}

}

Set<Entry<Integer, Integer>> entrySet = map.entrySet();

Iterator<Entry<Integer, Integer>> itr = entrySet.iterator();

**while** (itr.hasNext()) {

Entry<Integer, Integer> entry = itr.next();

Integer key = entry.getKey();

Integer value = entry.getValue();

**if** (value > 1)

uniqueCount++;

}

**int**[] uniqueArr = **new** **int**[uniqueCount];

Set<Entry<Integer, Integer>> entrySet2 = map.entrySet();

Iterator<Entry<Integer, Integer>> itr2 = entrySet2.iterator();

**while** (itr2.hasNext()) {

Entry<Integer, Integer> entry2 = itr2.next();

Integer key = entry2.getKey();

Integer value = entry2.getValue();

**if** (value > 1)

uniqueArr[k++] = key;

}

**return** uniqueArr;

}

Sort an Array of 0s 1s 2s

Find Maximum difference between Two Array Elements

Find Minimum Element in a Sorted and Rotated Array

How to rotate an array in both directions by K steps(31)

Find the Element that appears once in an Array

Reverse an Array

Maximum Subarray Sum

Find Maximum Sum of a Subarray of Size K

Find Top K most Frequent Elements

Find a peak element in an Array

Find majority element in an Array

Find second largest number in an array (multiple approaches)

Find first and last index of an element in a Sorted Array

Find length of longest increasing subarray

Find triplet with given sum in an array

Find triplets with Aero Sum (3 sum in java)

Sort an Array of 0s 1s 2s in Single Traversal

Search an Element in a Sorted and Rotated Array

Group Anagrams Together

Count Occurrences/Frequency of number in a sorted Array

Maximum Consecutive 1s in an array of 0s and 1s

Reverse Level Order Traversal of a Binary Tree

Maximum Consecutive Ones (If K Flip is Allowed)

Product of Array Except Self (Leetcode 238)

Find all numbers disappeared in an Array

Merge Overlapping Intervals

Minimum Size Subarray Sum Minimum Size Subarray Sum (Smallest Subarray with Sum Greater than or Equal to the Given Value)

Find the element that appears once in a Sorted Array in O(logn)

Find the duplicate Number in an Array without using Extra space/memory

Find First and Last Position of Element in Sorted Array

Sort Array By Parity

Contiguous Array

Plus One