**Company interview questions:**

1. **Find duplicates in an array.**

Use two loops. In the outer loop, pick elements one by one and count the number of occurrences of the picked element in the inner loop.

Java Code:

import java.util.Arrays;

public class Exercise12 {

public static void main(String[] args)

{

int[] my\_array = {1, 2, 5, 5, 6, 6, 7, 2};

for (int i = 0; i < my\_array.length-1; i++)

{

for (int j = i+1; j < my\_array.length; j++)

{

if ((my\_array[i] == my\_array[j]))

{

System.out.println("Duplicate Element : "+my\_array[j]);

}

}

}

}

}

Time Complexity: O(n\*n)

There is better solution for this by using HashSet, set does not have duplicate values.

private static void findDuplicatesUsingHashSet(int[] inputArray)

{

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

for (int element : inputArray)

{

if( ! set.add(element))

{

System.out.println("Duplicate Element : "+element);

}

}

Time complexity O(n)

1. **Find missing elements in an array.(if not duplicate)**

Sol: If there is no duplicates in array then algo is:

Step 1: Find total = (n+1)(n+2)/2

Step 2: Find sum = sum of all array elements

Step 3: missing element = total – sum

// Java program to find missing Number

class Main

{

// Function to ind missing number

static int getMissingNo (int a[], int n)

{

int i, total,sum;

total = (n+1)\*(n+2)/2;

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

sum += a[i];

return total-sum;

}

/\* program to test above function \*/

public static void main(String args[])

{

int a[] = {1,2,4,5,6};

int miss = getMissingNo(a,5);

System.out.println(miss);

}

}

Time Complexity : O(n)

1. **Find missing number in duplicate array?**

In this case, we need to use a different approach, something like a **roll-call** you would have seen in your school.  
  
The teacher has a register with names of all students, he goes through the list and mark absences on red. We can use the same approach to find all the missing numbers in the list.  
  
We can use an array as register and it's an index as names of the numbers. You need to [loop through the given array](http://javarevisited.blogspot.sg/2016/02/how-to-loop-through-array-in-java-with.html#axzz5DmwFLA1K) and tick marking all the numbers which are present by storing one of their respective indices. For example, if the first number in the given array is 5 (since the array is not sorted) then we store 1 on index 5 e.g. register[5] = 1  
  
Once we have gone through all the numbers is given, we can go through our register array and print all the indices where the value is zero. *These are absentees or missing numbers.*  
  
/\*

\* Java Program to **find missing numbers in an integer**

**\* array with duplicates**. Array may contains more

\* than one duplicates.

\*

\* input: {1, 1, 2, 3, 5, 5, 7, 9, 9, 9};

\* output: 4, 6, 8

\*/

**public** class Hello {

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

// given input

**int**[] **input** **=** { 1, 1, 2, 3, 5, 5, 7, 9, 9, 9 };

// let's create another array with same length

// by default all index will contain zero

// default value for int variable

**int**[] register **=** **new** **int**[**input**.**length**];

// now let's [iterate over given array](http://javarevisited.blogspot.sg/2013/11/java-array-101-for-programmers-and.html#axzz5DmwFLA1K) to

// mark all present numbers in our register

// array

**for** (**int** i **:** **input**) {

register[i] **=** 1;

}

// now, let's print all the absentees

System.out.println("missing numbers in given array");

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

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

System.out.println(i);

}

}

}

}

Output

missing numbers **in** given array

4

6

8

1. Write a java program to find out the first two max values from an array? (without sorting of array)

package com.java2novice.algos;

public class TwoMaxNumbers {

    public void printTwoMaxNumbers(int[] nums){

        int maxOne = 0;

        int maxTwo = 0;

        for(int n:nums){

            if(maxOne < n){

                maxTwo = maxOne;

                maxOne =n;

            } else if(maxTwo < n){

                maxTwo = n;

            }

        }

        System.out.println("First Max Number: "+maxOne);

        System.out.println("Second Max Number: "+maxTwo);

    }

    public static void main(String a[]){

        int num[] = {5,34,78,2,45,1,99,23};

        TwoMaxNumbers tmn = new TwoMaxNumbers();

        tmn.printTwoMaxNumbers(num);

    }

}

1. Write a java program to find out the first three max values from an array?

Sol 1: by iteration

class PrintLargest

{

    /\* Function to print three largest elements \*/

    static void print3largest(int arr[], int arr\_size)

    {

        int i, first, second, third;

        /\* There should be atleast three elements \*/

        if (arr\_size < 3)

        {

            System.out.print(" Invalid Input ");

            return;

        }

        third = first = second = Integer.MIN\_VALUE;

        for (i = 0; i < arr\_size ; i ++)

        {

            /\* If current element is greater than

            first\*/

            if (arr[i] > first)

            {

                third = second;

                second = first;

                first = arr[i];

            }

            /\* If arr[i] is in between first and

            second then update second  \*/

            else if (arr[i] > second)

            {

                third = second;

                second = arr[i];

            }

            else if (arr[i] > third)

                third = arr[i];

        }

        System.out.println("Three largest elements are " +

                       first + " " + second + " " + third);

    }

    /\* Driver program to test above function\*/

    public static void main (String[] args)

    {

        int arr[] = {12, 13, 1, 10, 34, 1};

        int n = arr.length;

        print3largest(arr, n);

    }

}

Sol 2: By sorting

**Another approach:**An efficient way to solve this problem is to use any **O(nLogn)**sorting algorithm & simply returning the last 3 largest elements .

import java.io.\*;

import java.util.Arrays;

class GFG {

     void find3largest(int[] arr)

    {

        Arrays.sort(arr); //It uses Tuned Quicksort with

                         //avg. case Time complexity = O(nLogn)

        int n = arr.length;

        int check = 0, count = 1;

        for(int i = 1; i <= n; i++){

            if(count<4){

                if(check!=arr[n-i])

                {

                   // to handle duplicate values

                    System.out.print(arr[n-i]+" ");

                    check = arr[n-i];

                    count++;

                }

            }

             else

                break;

        }

    }

   // Driver code

    public static void main(String[] args)

    {

        GFG obj = new GFG();

        int[] arr={12,45,1,-1,45,54,23,5,0,-10};

        obj.find3largest(arr);

    }

}

1. **[Given the below input and output and asked to write in Java.](https://www.careercup.com/question?id=5461598851301376)** [input : {1,2,3,4, &, 12,13,14,15}   
   output : {15,14,13,12,1,2,3,4}](https://www.careercup.com/question?id=5461598851301376)

[Assumption : '&' will always be in the middle.](https://www.careercup.com/question?id=5461598851301376)

public String[] reverseArrayWithModification(String[] arr) {

String[] out = new String[arr.length - 1];

int middle = arr.length / 2;

if (arr[middle].equals("&")) {

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

out[i] = arr[arr.length - 1 - i];

}

for (int i = middle + 1; i <= 2 \* middle; i ++) {

out[i - 1] = arr[i - middle - 1];

}

} else {

throw new IllegalArgumentException("Input array has no char '&'");

}

return out;

}

Complexity O(2\*n/2) = O(n)

1. **Merge two sorted array?**

Examples:

Input : arr1[] = { 1, 3, 4, 5}

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

Output : arr3[] = {1, 2, 3, 4, 5, 6, 7, 8}

Input : arr1[] = { 5, 8, 9}

arr2[] = {4, 7, 8}

Output : arr3[] = {4, 5, 7, 8, 8, 9}

public class Example {

public static void main (String[] args) {

int[] arr1 = {11, 34, 66, 75};

int n1 = arr1.length;

int[] arr2 = {1, 5, 19, 50, 89, 100};

int n2 = arr2.length;

int[] merge = new int[n1 + n2];

int i = 0, j = 0, k = 0, x;

while (i < n1 && j < n2) {

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

merge[k++] = arr1[i++];

else

merge[k++] = arr2[j++];

}

while (i < n1)

merge[k++] = arr1[i++];

while (j < n2)

merge[k++] = arr2[j++];

System.out.print("\nArray after merging: ");

for (x = 0; x < n1 + n2; x++)

System.out.print(merge[x] + " ");

}

}

**Time Complexity :**O(n1 + n2)

1. **Print number of words, vowels and frequency of each character**

**Example :**

**Input :** How Good GOD Is.

**Output :**

Number of words = 4

Number of vowels = 5

Number of upper case characters = 6

Character = Frequency = 3

Character = . Frequency = 1

Character = D Frequency = 1

Character = G Frequency = 2

Character = H Frequency = 1

Character = I Frequency = 1

Character = O Frequency = 1

Character = d Frequency = 1

Character = o Frequency = 3

Character = s Frequency = 1

Character = w Frequency = 1

|  |
| --- |
| // Java program to print Number of Words,  // Vowels and Frequency of Each Character  import java.util.\*;  import java.lang.\*;  import java.io.\*;    public class Stringfun  {      String str = "Geeks for Geeks.";        void words()      {          int wCount = 0, uCount = 0, vCount = 0;            for (int i = 0; i < str.length(); i++)          {              char c = str.charAt(i);                switch (c)              {              case ' ':              case '.':                  wCount++; // more delimiters can be given              }                switch (c)              {              // program for calculating number of vowels              case 'A':              case 'E':              case 'I':              case 'O':              case 'U':              case 'a':              case 'e':              case 'i':              case 'o':              case 'u':                  vCount++;              }                if (c >= 65 && c <= 90)              {                  uCount++;              }          }            System.out.println("Number of words = " + wCount);          System.out.println("Number of vowels = " + vCount);          System.out.println("Number of upper case characters = "                                                          + uCount);      }        // Function to calculate the frequency      // of each character in the string      void frequency()      {          // Creates an empty TreeMap          TreeMap<Character, Integer> hmap =                       new TreeMap<Character, Integer>();            // Traverse through the given array          for (int i = 0; i < str.length(); i++)          {              Integer c = hmap.get(str.charAt(i));                // If this is first occurrence of element              if (hmap.get(str.charAt(i)) == null)                 hmap.put(str.charAt(i), 1);                // If elements already exists in hash map              else                hmap.put(str.charAt(i), ++c);          }            // Print result          for (Map.Entry m:hmap.entrySet())            System.out.println("Character = " + m.getKey() +                           " Frequency = " + m.getValue());      }        // Driver program to run and test above program      public static void main(String args[]) throws IOException      {          Stringfun obj = new Stringfun();          obj.words();          obj.frequency();      }  } |

Generating subarrays using recursion

Examples:

Input : [1, 2, 3]

Output : [1], [1, 2], [2], [1, 2, 3], [2, 3], [3]

Approach: We use two pointers start and end to maintain the starting and ending point of the array and follow the steps given below:

Stop if we have reached the end of the array

Increment the end index if start has become greater than end

Print the subarray from index start to end and increment the starting index

class solution

{

// Recursive function to print all possible subarrays

// for given array

static void printSubArrays(int []arr, int start, int end)

{

    // Stop if we have reached the end of the array

    if (end == arr.length)

        return;

    // Increment the end point and start from 0

    else if (start > end)

        printSubArrays(arr, 0, end + 1);

    // Print the subarray and increment the starting point

    else

    {

        System.out.print("[");

        for (int i = start; i < end; i++){

            System.out.print(arr[i]+", ");

        }

        System.out.println(arr[end]+"]");

        printSubArrays(arr, start + 1, end);

    }

    return;

}

public static void main(String args[])

{

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

printSubArrays(arr, 0, 0);

}

}

1. Count distinct elements in an array

Input: arr[] = {12, 10, 9, 45, 2, 10, 10, 45}

Output: 12, 10, 9, 45, 2

|  |
| --- |
| // Java program to print all distinct  // elements in a given array  import java.io.\*;    class GFG {        static void printDistinct(int arr[], int n)      {          // Pick all elements one by one          for (int i = 0; i < n; i++)          {              // Check if the picked element              // is already printed              int j;              for (j = 0; j < i; j++)              if (arr[i] == arr[j])                  break;                // If not printed earlier,              // then print it              if (i == j)              System.out.print( arr[i] + " ");          }      }        // Driver program      public static void main (String[] args)      {          int arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10};          int n = arr.length;          printDistinct(arr, n);        }  } |

1. **Check whether an Array is Subarray of another Array**

Examples: Input : A[] = {2, 3, 0, 5, 1, 1, 2}, B[] = {3, 0, 5, 1}  
Output : Yes

Input : A[] = {1, 2, 3, 4, 5}, B[] = {2, 5, 6}  
Output : No

**Method 1 (Simple)**  
Use two loops: The outer loop picks all the elements of arr2[] one by one. The inner loop linearly searches for the element picked by outer loop. If all elements are found then return 1, else return 0.

class GFG {

    /\* Return true if arr2[] is a subset

    of arr1[] \*/

    static boolean isSubset(int arr1[],

                int arr2[], int m, int n)

    {

        int i = 0;

        int j = 0;

        for (i = 0; i < n; i++)

        {

            for (j = 0; j < m; j++)

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

                    break;

            /\* If the above inner loop

            was not broken at all then

            arr2[i] is not present in

            arr1[] \*/

            if (j == m)

                return false;

        }

        /\* If we reach here then all

        elements of arr2[] are present

        in arr1[] \*/

        return true;

    }

    // Driver code

    public static void main(String args[])

    {

        int arr1[] = {11, 1, 13, 21, 3, 7};

        int arr2[] = {11, 3, 7, 1};

        int m = arr1.length;

        int n = arr2.length;

        if(isSubset(arr1, arr2, m, n))

            System.out.print("arr2[] is "

                  + "subset of arr1[] ");

        else

            System.out.print("arr2[] is "

             + "not a subset of arr1[]");

    }

}

1. **Write a function to find the maximum sum of sub array where the array can have negative and positive numbers.**

import java.io.\*;

// Java program to print largest contiguous array sum

import java.util.\*;

class Kadane

{

    public static void main (String[] args)

    {

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

        System.out.println("Maximum contiguous sum is " +

                                       maxSubArraySum(a));

    }

    static int maxSubArraySum(int a[])

    {

        int size = a.length;

        int max\_so\_far = Integer.MIN\_VALUE, max\_ending\_here = 0;

        for (int i = 0; i < size; i++)

        {

            max\_ending\_here = max\_ending\_here + a[i];

            if (max\_so\_far < max\_ending\_here)

                max\_so\_far = max\_ending\_here;

            if (max\_ending\_here < 0)

                max\_ending\_here = 0;

        }

        return max\_so\_far;

    }

}

1. Remove duplicates from a given string.

StringBuilder sb = new StringBuilder();

int idx;

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

    char c = str.charAt(i);

    idx = str.indexOf(c, i + 1);

    if (idx == -1) {

        sb.append(c);

    }

}

Another way is to use set.

import java.util.\*;

class RemoveDuplicates

{

    /\* Function removes duplicate characters from the string

    This function work in-place \*/

    void removeDuplicates(String str)

    {

        LinkedHashSet<Character> lhs = new LinkedHashSet<>();

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

            lhs.add(str.charAt(i));

        // print string after deleting duplicate elements

        for(Character ch : lhs)

            System.out.print(ch);

    }

    /\* Driver program to test removeDuplicates \*/

    public static void main(String args[])

    {

        String str = "geeksforgeeks";

        RemoveDuplicates r = new RemoveDuplicates();

        r.removeDuplicates(str);

    }

}

1. **Find majority element in an array or element which repeated maximum.**

Input : arr[] = {1, 3, 2, 1, 4, 1}

Output : 1

1 appears three times in array which is maximum frequency.

A **simple solution**is to run two loops. The outer loop picks all elements one by one. The inner loop finds frequency of the picked element and compares with the maximum so far. Time complexity of this solution is O(n2).

Another solution is to sort the array. Use the built in Arrays.sort() method. Now compare the adjacent elements. Consider this example: 1 1 1 1 4 4 4 4 4 4 4 4 4 4 4 4 9 9 9 10 10 10 29 29 29 29 29 29

When the adjacent elements are not equal, you can stop counting that element. Time complexity would be nlogn.

An **efficient solution** is to use hashing. We create a hash table and store elements and their frequency counts as key value pairs. Finally we traverse the hash table and print the key with maximum value.

import java.util.HashMap;

import java.util.Map;

import java.util.Map.Entry;

class GFG {

    static int mostFrequent(int arr[], int n)

    {

        // Insert all elements in hash

        Map<Integer, Integer> hp =

               new HashMap<Integer, Integer>();

        for(int i = 0; i < n; i++)

        {

            int key = arr[i];

            if(hp.containsKey(key))

            {

                int freq = hp.get(key);

                freq++;

                hp.put(key, freq);

            }

            else

            {

                hp.put(key, 1);

            }

        }

        // find max frequency.

        int max\_count = 0, res = -1;

        for(Entry<Integer, Integer> val : hp.entrySet())

        {

            if (max\_count < val.getValue())

            {

                res = val.getKey();

                max\_count = val.getValue();

            }

        }

        return res;

    }

    // Driver code

    public static void main (String[] args) {

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

        int n = arr.length;

        System.out.println(mostFrequent(arr, n));

    }

}

**Time Complexity :** O(n)

1. **Median of Two sorted arrays?**

First merge two sorted arrays and then and find median.

1. **Binary search in sorted array**

Binary Search

Given a sorted array arr[] of n elements, write a function to search a given element x in arr[].

A simple approach is to do [linear search](http://quiz.geeksforgeeks.org/linear-search/)**.**The time complexity of above algorithm is O(n). Another approach to perform the same task is using Binary Search.

**Binary Search:** Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.

// Java implementation of iterative Binary Search

class BinarySearch {

    // Returns index of x if it is present in arr[],

    // else return -1

    int binarySearch(int arr[], int x)

    {

        int l = 0, r = arr.length - 1;

        while (l <= r) {

            int m = l + (r - l) / 2;

            // Check if x is present at mid

            if (arr[m] == x)

                return m;

            // If x greater, ignore left half

            if (arr[m] < x)

                l = m + 1;

            // If x is smaller, ignore right half

            else

                r = m - 1;

        }

        // if we reach here, then element was

        // not present

        return -1;

    }

    // Driver method to test above

    public static void main(String args[])

    {

        BinarySearch ob = new BinarySearch();

        int arr[] = { 2, 3, 4, 10, 40 };

        int n = arr.length;

        int x = 10;

        int result = ob.binarySearch(arr, x);

        if (result == -1)

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

        else

            System.out.println("Element found at "

                               + "index " + result);

    }

}

// Java implementation of recursive Binary Search

class BinarySearch {

    // Returns index of x if it is present in arr[l..

    // r], else return -1

    int binarySearch(int arr[], int l, int r, int x)

    {

        if (r >= l) {

            int mid = l + (r - l) / 2;

            // If the element is present at the

            // middle itself

            if (arr[mid] == x)

                return mid;

            // If element is smaller than mid, then

            // it can only be present in left subarray

            if (arr[mid] > x)

                return binarySearch(arr, l, mid - 1, x);

            // Else the element can only be present

            // in right subarray

            return binarySearch(arr, mid + 1, r, x);

        }

        // We reach here when element is not present

        // in array

        return -1;

    }

    // Driver method to test above

    public static void main(String args[])

    {

        BinarySearch ob = new BinarySearch();

        int arr[] = { 2, 3, 4, 10, 40 };

        int n = arr.length;

        int x = 10;

        int result = ob.binarySearch(arr, 0, n - 1, x);

        if (result == -1)

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

        else

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

    }

}

1. **Find the number which occurs odd no of times**

A **Simple Solution** is to run two nested loops. The outer loop picks all elements one by one and inner loop counts number of occurrences of the element picked by outer loop. Time complexity of this solution is O(n2).

class OddOccurrence {

    // funtion to find the element occurring odd

    // number of times

    static int getOddOccurrence(int arr[], int arr\_size)

    {

        int i;

        for (i = 0; i < arr\_size; i++) {

            int count = 0;

            for (int j = 0; j < arr\_size; j++) {

                if (arr[i] == arr[j])

                    count++;

            }

            if (count % 2 != 0)

                return arr[i];

        }

        return -1;

    }

    // driver code

    public static void main(String[] args)

    {

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

        int n = arr.length;

        System.out.println(getOddOccurrence(arr, n));

    }

}

A **Better Solution** is to use Hashing. Use array elements as key and their counts as value. Create an empty hash table. One by one traverse the given array elements and store counts. Time complexity of this solution is O(n). But it requires extra space for hashing.

import java.io.\*;

import java.util.HashMap;

class OddOccurrence

{

    // funtion to find the element occurring odd

    // number of times

    static int getOddOccurrence(int arr[], int n)

    {

        HashMap<Integer,Integer> hmap = new HashMap<>();

        // Putting all elements into the HashMap

        for(int i = 0; i < n; i++)

        {

            if(hmap.containsKey(arr[i]))

            {

                int val = hmap.get(arr[i]);

                // If array element is already present then

                // increase the count of that element.

                hmap.put(arr[i], val + 1);

            }

            else

                // if array element is not present then put

                // element into the HashMap and initialize

                // the count to one.

                hmap.put(arr[i], 1);

        }

        // Checking for odd occurrence of each element present

        // in the HashMap

        for(Integer a:hmap.keySet())

        {

            if(hmap.get(a) % 2 != 0)

                return a;

        }

        return -1;

    }

    // driver code

    public static void main(String[] args)

    {

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

        int n = arr.length;

        System.out.println(getOddOccurrence(arr, n));

    }

}

The **Best Solution** is to do bitwise XOR of all the elements. XOR of all elements gives us odd occurring element. Please note that XOR of two elements is 0 if both elements are same and XOR of a number x with 0 is x.

class OddOccurance

{

    int getOddOccurrence(int ar[], int ar\_size)

    {

        int i;

        int res = 0;

        for (i = 0; i < ar\_size; i++)

        {

            res = res ^ ar[i];

        }

        return res;

    }

    public static void main(String[] args)

    {

        OddOccurance occur = new OddOccurance();

        int ar[] = new int[]{2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};

        int n = ar.length;

        System.out.println(occur.getOddOccurrence(ar, n));

    }

}

**Time Complexity:**O(n)

1. **First non-repeating character in string**

There can be two solution, one is linear traversal and another by using HashMap.