**Algorithm – Data structures – 2021-2022**

**Remove Duplicates from a Sorted Array**

**public static int[] deleteDuplicates(int[] a) { 🡸 Better way  
 int j = 0;  
 for (int i = 0; i < a.length; i++) {  
 if (i < a.length - 1 && a[i] == a[i + 1]) continue;  
 a[j++] = a[i];  
 }  
 int[] b = Arrays.*copyOf*(a, j);  
 return b;  
}**

**public** **class** RemoveDuplicatesFromSortedArray {

**public static int[] removeDuplicates(int[] a) {  
 int j = 0;  
 for(int i = 0; i < a.length; i++) {  
 try {  
 if( (a[i] ^ a[i+1]) != 0 ) {  
 j++;  
 a[j] = a[i+1];  
 }  
 } catch(IndexOutOfBoundsException ibe) { 🡸 Correct  
 break;  
 }  
 }  
 int newSize = (j+1);  
 int[] b = Arrays.*copyOf*(a, newSize);  
 return b;  
}**

}

**Delete an element or all elements from an integer array**

**import** java.util.Arrays;

**public** **class** DeleteAnElementFromArray {

**public static int[] removeAllElement(int[] a, int value) {  
 int j = 0;  
 for (int i = 0; i < a.length; i++) {  
 if (a[i] != value) a[j++] = a[i];  
 }  
 int[] b = Arrays.*copyOf*(a,j);  
 return b;  
}**

}

# **Show Duplicates in a sorted integer Array**

public class DupllicatesInArray {

public static void showDuplicates( int[] a ) {

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

try {

**if( (a[i]^a[i+1]) == 0 )**

**System.out.println(a[i]);**

}

catch (IndexOutOfBoundsException e) { break; }

}

}

}

**Find Missing and Duplicate in a sorted integer array**  
**public class** FindMissingAndDuplicate {  
 **public static void** findMissingAndDuplicate(**int**[] a) {  
 **int n = a.length;  
 int idealSum = ( ( n\*( n+1 ) )/2 );  
 int actualSum = 0;  
 int duplicate = 0;  
 for (int i = 0; i < n; i++) {  
 actualSum = actualSum+a[i];  
 try {  
 if( (a[i] ^ a[i+1]) == 0 )  
 duplicate = a[i];  
 }  
 catch (IndexOutOfBoundsException ibe) {  
 break;  
 }  
 }  
 int missing = idealSum - actualSum+duplicate;** System.***out***.println(**"Duplicate :::"**+duplicate+**"----"**+**"Missing :::"**+missing);  
 }  
 }

# **Find missing number from 1 to N**

**public class** FindMissingNo1ToN {  
 **public static int** findMissing(**int**[] a, **int** n ) {  
 **int idealSum = ( ( n\*( n+1 ) )/2 );  
 int actualSum = 0;  
 for (int i = 0; i < a.length; i++) {  
 actualSum = actualSum + a[i];  
 }  
 return idealSum - actualSum;** }  
}

# **Swap Two Numbers**

**int a = 5;  
int b = 6;**

**a = a ^ b;  
b = a ^ b;  
a = a ^ b;**

# **Print 1 to N recursively**

**public** **static** **void** printNTo1(**int** n) {

**if** (n != 0) {

System.***out***.print(n + "\t");

*printNTo1*(n - 1);

}

}

**public** **static** **void** print1ToN(**int** n) {

**if** (n != 0) {

*print1ToN*(n - 1);

System.***out***.print(n + "\t");

}

}

**Factorial of a number using recursion**

**public class** FactorialOfANumber {  
 **public static int** factorial( **int** n ) {  
 **if( n==0 ) return 1;  
 return n\**factorial*( n-1 );**  
 }  
 **public static void** main(String[] args) {  
 System.***out***.println(*factorial*(6));  
 }  
}

# **Fibonacci Series**

**public class** FibbonacciSeries {  
 **public static int** fibbo( **int** n ) {  
 **if( n == 0 ) return 0;  
 else if( n == 1) return 1;  
 else  
 return *fibbo*( n - 1)+*fibbo*( n - 2 );** }  
 **public static void** main(String[] args) {  
 **int** n = 7;  
 **for** (**int** i = 0; i < n; i++)   
 System.***out***.println(*fibbo*(i));  
 }  
}

# **Reverse an Integer Array**

**public static void reverseIntArray(int[] a) {  
 for (int i = 0, j = a.length - 1; i < j; i++, j--) {  
 *swap*(a, i, j);  
 }  
}  
  
public static void swap(int[] a, int i, int j) {  
 int temp = a[i];  
 a[i] = a[j];  
 a[j] = temp;  
}**

**Reverse a character array (**Best way to do it**)**

**public static void reverseCharArray(char[] a) {  
 for (int i = 0, j = a.length - 1; i < j; i++, j--) {  
 *swap*(a, i, j);  
 }  
}  
  
public static void swap(char[] a, int i, int j) {  
 char temp = a[i];  
 a[i] = a[j];  
 a[j] = temp;  
}**

# **Reverse a String**

**public static** String reverse(String s ) {  
 **if**( s == **null** || s.length() == 0 ) **return** s;

**return *reverse*(s.substring(1))** + **s.charAt(0)** ;   
}

**Alternative way to reverse a String**

Convert string to character array and reverse the character and form a new String.

**public static String reverse(String s) {**

**char[] ch = s.toCharArray();**

**reverseCharArray(ch);**

**return new String(ch);**

**}**

**public static void reverseCharArray(char[] a) {**

**for (int i = 0, j = a.length - 1; i < j; i++, j--) {**

**swap(a, i, j);**

**}**

**}**

**public static void swap(char[] a, int i, int j) {**

**char temp = a[i];**

**a[i] = a[j];**

**a[j] = temp;**

**}**

**Shuffle an integer array**

**import** java.util.Random;

**public** **class** MyShuffle {

**public static void** shuffle(**int**[] a) {  
 Random random = **new** Random();  
 **for**(**int** i = 0; i < a.**length**; i++) {  
 **int** randPos = random.nextInt(a.**length**);  
 *swap*(a,i,randPos);  
 }  
}  
**public static void** swap(**int**[] a, **int** i, **int** j) {  
 **int** temp = a[i];  
 a[i] = a[j];  
 a[j] = temp;  
}  
**public static void** main(String[] args) {  
 **int** a[]={1,2,3,4,5,6,7};  
 *shuffle*(a);  
 System.***out***.println(Arrays.*toString*(a));  
}

}

**Second Largest in an Array**

public static void secondLargeNumber(int[] arr) {

**int largest = arr[0];**

**int secondLargest = arr[0];**

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

**if (arr[i] > largest) {**

**secondLargest = largest;**

**largest = arr[i];**

**} else if (arr[i] > secondLargest) {**

**secondLargest = arr[i];**

**}**

**}**

System.out.println("second largest in array is:" + secondLargest);

}

}

**\*\* In case Nth largest or smallest, sort the array using Quick sort and find the nth position.**

**Array Rotation**

**Rotate Left**

Before : 1 2 3 4 5

After : 2 3 4 5 1

**Rotate Right**

Before : 1 2 3 4 5

After : 5 1 2 3 4

**public static void** rotateRightBy1(**int**[] a) { *// correct* **int** last = a[a.**length** - 1];  
 **for** (**int** i = a.**length** - 1; i > 0; i--)  
 a[i] = a[i - 1];  
 a[0] = last;  
}  
  
**public static void** rotateRightByN(**int**[] a, **int** n) { *// correct* **for** (**int** i = 0; i < n; i++)  
 *rotateRightBy1*(a);  
}  
  
**public static void** rotateLeftBy1(**int**[] a) { *// Correct* **int** first = a[0];  
 **for** (**int** i = 0; i < a.**length** - 1; i++)  
 a[i] = a[i + 1];  
 a[a.**length** - 1] = first;  
}  
  
**public static void** rotateLeftByN(**int**[] a, **int** n) { *// correct* **for** (**int** i = 0; i < n; i++)  
 *rotateLeftBy1*(a);  
}

**Better in an optimized way with Time complexity: o(n) and Space complexity: o(1)**

**public static int**[] rotateOptimized(**int**[] a, **int** n) {  
 a = *reverse*(a, 0, a.**length** - 1);  
 a = *reverse*(a, 0, n - 1);  
 a = *reverse*(a, n, a.**length** - 1);  
 **return** a;  
}  
  
**public static int**[] reverse(**int**[] a, **int** i, **int** j) {  
 **while** (i <= j) {  
 *swap*(a, i, j);  
 i++;  
 j--;  
 }  
 **return** a;  
}  
  
**public static void** swap(**int**[] a, **int** i, **int** j) {  
 **int** temp = a[i];  
 a[i] = a[j];  
 a[j] = temp;  
}

**Find out whether two strings are the same or not without using the equality operator not the equals method.**

//Two strings of equal length

**public static boolean stringEquals(String s1, String s2) {  
 int count = 0;  
 for(int i = 0; i < s1.length(); i++)   
 count = count + s1.charAt(i) ^ s2.charAt(i);  
 return (count == 0);  
}**

# **Write an algorithm so that a[i]+a[j] = k in a sorted array of integers**

import java.util.Hashtable;

public class SumOfTwoElementsInArray {

**public static void printPairs(int[] a, int sum) {  
 Map<Integer, Integer> map = new HashMap<>();  
 for (int i = 0; i < a.length; i++) {  
 if (map.containsKey(a[i])) {  
 System.*out*.println(a[i] + " and " + map.get(a[i]));  
 } else  
 map.put(sum - a[i], a[i]);  
 }  
}**

public static void main(String[] args) {

int[] arr = new int[]{1,2,3,4,5,6,7,9};

printPairs(arr, 7);//This is the best

}

}

**How to find a first non repeating character in a String**

public class FirstNonRepeatedChar {

public static void getfirstNonRepetedChar(char[] ch) {

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

int flag = 0;

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

if (ch[i] == ch[j]) {

flag++;

}

if (flag > 2) {

break;

Output

------

C K

}

}

if (flag == 1) {

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

}

}

}

public static void main(String[] args) {

char[] ch = { 'A', 'B', 'B', 'C', 'K', 'D', 'A', 'D' };

getfirstNonRepetedChar(ch);

}

}

**Alternative - How to find a first non repeating character in a String**

**public static void showFirstNonRepeatedChars(String s) {  
 Map<Character,Integer> map = new HashMap<>();  
 for(char ch : s.toCharArray()) {  
 map.put(ch, map.containsKey(ch)? map.get(ch)+1 : 1);  
 }  
 map.forEach( (k, v) -> {  
 if(v==1)  
 System.*out*.println(k);  
 });  
}**

**String compression**

**Input String : aaabbbdccaabcc**

**Output String : a5b4c4d1**

import java.util.Map;

import java.util.TreeMap;

public class StringCompression {

public static void compress(String s) {

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

        for(char ch : s.toCharArray()) {

            map.put(ch, map.containsKey(ch)? map.get(ch)+1 : 1);

        }

        StringBuilder sb = new StringBuilder();

        map.forEach((k,v)-> {

            sb.append(k).append(v);

        });

        System.out.println(sb.toString());

    }

public static void main(String[] args) {

String s = "aaabbbdccaabcc";

Compress(s);

}

}

**Write a thread-safe array-based queue implementation in java**

package com.ddlab.core.algorithm;

import java.util.ArrayList;

/\*

\* Write a thread-safe array-based queue implementation in java.

\* If one thread reaches to a limit, it should wait for dequeue thread to create a space and vice-versa.

\*/

public class ThreadSafeArrayQueue<E> {

private static final int size=10;

private ArrayList<E> arr= new ArrayList<E>(10);

public synchronized void enqueue(E item) throws InterruptedException {

while(arr.size() == size) {

wait();

}

if(arr.isEmpty()) {

notify();

}

System.out.println("Item added : "+item);

arr.add(item);

}

public synchronized E dequeue(int item) throws InterruptedException {

while(arr.isEmpty()) {

wait();

}

if(arr.size()==size) {

notify();

}

return arr.remove(0);

}

public static void main(String[] args) throws Exception {

ThreadSafeArrayQueue<String> thSafeQ = new ThreadSafeArrayQueue<String>();

for( int i = 0 ; i < 11 ; i++ )

thSafeQ.enqueue("Item-"+i);

}

}

**Insert an element into an already sorted array**

**Alternative way**

**public static int**[] insert(**int**[] a, **int** value) {  
 **int** index = Arrays.*binarySearch*(a, value);  
 System.***out***.println(**"Index ::: "** + index);  
 **int** newIndex = 0;  
 **if** (index < 0)  
 newIndex = -(index) - 1;  
 **else** newIndex = index + 1;  
 a = *insertElement*(a, value, newIndex);  
 **return** a;  
}  
  
**public static int**[] insertElement(**int**[] a, **int** value, **int** index) {  
 **int** length = a.**length**;  
 **int**[] destn = **new int**[length + 1];  
 System.*arraycopy*(a, 0, destn, 0, index);  
 destn[index] = value;  
 System.*arraycopy*(a, index, destn, index + 1, length - index);  
 **return** destn;  
}

**Better way to do it**

**public static int binarySearch(int[] a, int b) {**

**int low = 0;**

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

**while (low <= high) {**

**int mid = low + (high - low) / 2;**

**if (b < a[mid]) high = mid - 1;**

**else if (b > a[mid]) low = mid + 1;**

**else return mid;**

**}**

**return -(low + 1);**

**}**

**public static int[] addByIndex(int[] a, int key, int index) {**

**int[] b = new int[a.length + 1];**

**for (int i = 0; i < index; i++)**

**b[i] = a[i];**

**b[index] = key;**

**for (int i = index + 1; i < b.length; i++)**

**b[i] = a[i - 1];**

**return b;**

**}**

**public static int[] insertIntoSortedArray(int[] a, int b) {**

**int index = binarySearch(a, b);**

**System.out.println("Index Value : " + index);**

**int newIndex = 0;**

**if (index < 0)**

**newIndex = -(index) - 1;**

**else**

**newIndex = index + 1;**

**System.out.println("NewIndex value : " + newIndex);**

**int[] c = addByIndex(a, b, newIndex);**

**return c;**

**}**

**There is another way to do it using insertion sort**

**public static int[] insertIntoSortedArray1(int[] a, int val) {**

**int[] b = new int[a.length+1];**

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

**b[i] = a[i];**

**b[a.length] = val;**

**insertionSort(b);**

**return b;**

**}**

**public static void insertionSort(int[] a) {**

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

**for(int j = i; j > 0 && a[j] < a[j-1]; j--){**

**swap(a,j, j-1);**

**}**

**}**

**}**

**public static void swap(int[] a, int i, int j) {**

**int temp = a[i];**

**a[i] = a[j];**

**a[j] = temp;**

**}**

**Anagram – Two Strings are anagram or not**

<http://stackoverflow.com/questions/15045640/how-to-check-if-two-words-are-anagrams>

Fastest algorithm would be to map each of the 26 English characters to a unique prime number. Then calculate the product of the string. By the fundamental theorem of arithmetic, 2 strings are anagrams if and only if their products are the same.

An anagram of a string is **another string that contains the same characters, only the order of characters can be different**. For example, “abcd” and “dabc” are an anagram of each other.

Examples: **Listen, Silent, Triangle, Integral**

**public class** Anagram {

**public** **static** **boolean** areAnagrams(String s1, String s2) {

**int** val1 = 1;

**int** val2 = 1;

**for** (**int** i = 0; i < s1.length(); i++) {

val1 = val1\*s1.charAt(i);

val2 = val2\*s2.charAt(i);

}

**return** (val1 == val2);

}  
  
 **public static void** main(String[] args) {  
  
 String s1 = **"post"**;  
 String s2 = **"pots"**;

System.out.println(areAnagrams(s1, s2)); // True  
 }  
}

**String permutation**

**public** **class** StringPermutation {

**public** **static** **void** perm(String prefix, String s ) {

**int** n = s.length() ;

**if**( n == 0 ) System.*out*.println(prefix);

**else** {

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

*perm*( **prefix+s.charAt(i) , s.substring(0,i)+s.substring(i+1,n)** );

}

}

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

*perm*("","abc");

}

}

**Permutation of Integer Array (***Best way to do it***)**

*//* [*http://introcs.cs.princeton.edu/java/23recursion/Permutations.java.html*](http://introcs.cs.princeton.edu/java/23recursion/Permutations.java.html)

**public class** IntegerArrayPermutation {

**public static void** print(**int**[] arr) {  
 **for**( **int** p : arr)  
 System.***out***.print(p + **" "**);  
 System.***out***.println(**"\n"**);  
 }  
  
 **private static void** swap(**int**[] a, **int** i, **int** j) {  
 **int c = a[i];  
 a[i] = a[j];  
 a[j] = c;** }  
  
 **public static void** perm(**int**[] arr) {  
 **int N = arr.length;  
 *perm*(arr, N);** }  
  
 **private static void** perm(**int**[] a, **int** n) {  
 **if (n == 1) {  
 *print*(a);  
 return;  
 }  
 for (int i = 0; i < n; i++) {  
 *swap*(a, i, n-1);  
 *perm*(a, n-1);  
 *swap*(a, i, n-1);  
 }** }  
  
 **public static void** main(String[] args) {  
 **int**[] a = {1,2,3,4};  
 *perm*(a);  
 }  
}