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

**Remove Duplicates from a Sorted Array**

**import** java.util.Arrays;

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

**public** **static** **int**[] removeDuplicates(**int**[] a) {

**int** j = 0;

**int**[] newArr = **null**;

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

**try** {

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

// Duplicates found

} **else** {

j++;

a[j] = a[i + 1];

}

} **catch** (IndexOutOfBoundsException ibe) {

**break**;

}

}

**int** newSize = (j + 1);

newArr = **new** **int**[newSize];

newArr = Arrays.*copyOf*(a, newSize);

**return** newArr;

}

**public** **static** **int**[] removeDuplicates1(**int**[] a) {

**int** j = 0;

**int**[] newArr = **null**;

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

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

// Duplicates found

} **else** {

j++;

a[j] = a[i + 1];

}

}

**int** newSize = (j + 1);

newArr = **new** **int**[newSize];

newArr = Arrays.*copyOf*(a, newSize);

**return** newArr;

}

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

**int**[] a = { 6, 7, 8, 9, 9, 10, 11, 11, 11, 12, 13, 13, 14, 14, 14,

14, 15, 15, 17, 19 };

// int[] a = { -3, -2, -2, -2, 0, 0, 1, 1, 2, 3, 4, 5, 5, 5, 6, 6, 6, 6 };

**for** (**int** i : a)

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

System.***out***.println("\n");

**int**[] c = *removeDuplicates*(a);

System.***out***.println("\n\n");

a = c;

**for** (**int** i : a)

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

}

}

**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);  
 }  
  
 **public static void** main(String[] args) {  
 **int**[] a = {1,2,3,5,5,6,7,8,9,10};  
 *findMissingAndDuplicate*(a);  
 }  
}

# **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; }

}

}

public static void main(String[] args) {

int a[]={1,1,1,2,3,4,4,5,6,7,7,9,10};

showDuplicates(a);

}

}

# **Swap Two Numbers**

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

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

# **Delete an element from an integer array**

**import** java.util.Arrays;

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

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

**int** j = 0;

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

**try** {

**if** (a[i] == val) {

// Found element

} **else** {

a[j] = a[i];

j++;

}

} **catch** (IndexOutOfBoundsException ibe) {

**break**;

}

}

**int** newSize = j;

**int**[] newArr = Arrays.*copyOf*(a, newSize);

**return** newArr;

}

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

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

a = *deleteFromArray*(a, 1); // It will delete all occurrences of that number

**for** (**int** i : a)

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

}

}

# **Print 1 to N recursively**

**public** **class** Print1ToN {

**public** **static** **void** print1ToNDescendingOrder(**int** n) {

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

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

*print1ToNDescendingOrder*(n - 1);

}

}

**public** **static** **void** print1ToNAscendingOrder(**int** n) {

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

*print1ToNAscendingOrder*(n - 1);

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

}

}

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

**if** (n == 0) **return**;

*print1ToN*(n - 1);

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

}

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

// print1ToNDescendingOrder(5);//Prints 5 4 3 2 1

*print1ToNAscendingOrder*(5);//prints 1 2 3 4 5

}

}

**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));  
 }  
}

# **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;** }  
  
 **public static void** main(String[] args) {  
 **int**[] a = {5,4,1,3,2,7};  
 System.***out***.println(**"Missing :::"**+*findMissing*(a,7));  
 }  
}

# **Reverse an Integer Array**

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

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

**public static void** reverse1(**char**[] ch) {

**for** (**int** i = 0 , j = ch.**length** - 1 ; i < j ; i++ , j-- ) {  
 **char** temp = ch[i];  
 ch[i] = ch[j];  
 ch[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**

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

**Shuffle an integer array**

**import** java.util.Random;

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

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

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

Random random = **new** Random();

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

**int** randomPosition = random.nextInt(a.length);

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

a[i] = a[randomPosition];

a[randomPosition] = temp;

}

**for**( **int** k : a )

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

}

}

**Second Largest in an Array**

public class SecondLargestNumberInArray {

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);

}

public static void main(String[] args) {

int arr[] = { 1, 23, 47, 81, 92, 88, 52, 48, 56, 66, 65, 76, 71, 85,

49, 53, 56, 61, 65, 84 };

secondLargeNumber(arr);

}

}

**\*\* 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

//Rotate Left

**public static void** rotateLeft( Object[] a ) {  
 Object firstElement = a[0];  
 **for** (**int** i = 1; i < a.**length**; i++) {  
 a[i-1] = a[i];  
 }  
 a[ a.**length** -1 ] = firstElement;  
}

//Rotate Right  
**public static void** rotateRight( Object[] a ) {  
 Object lastElement = a[ a.**length** - 1 ];  
 **for** (**int** i = a.**length** - 1; i > 0 ; i--) {  
 a[i] = a[i - 1];  
 }  
 a[ 0 ] = lastElement;  
}

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

import java.util.Hashtable;

public class SumOfTwoElementsInArray {

//Below is the best

public static void printPairs(int[] array, int sum) {

Hashtable numberSet= new Hashtable();

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

**if(numberSet.containsKey(array[i])) {**

**System.out.println("Here is the values "+ array[i] +" and " +numberSet.get(array[i]));**

}

else

**numberSet.put(sum-array[i], array[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**

/\*

\* 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;

}

}

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);

}

}

Output

------

C K

**Alternative**

**public static char** getFirstNonRepeatedChar(String str) {  
 Map<Character, Integer> counts = **new** LinkedHashMap<>(str.length());  
 **for** (**char** c : str.toCharArray()) {  
 counts.put(c, counts.containsKey(c) ? counts.get(c) + 1 : 1);  
 }  
 **for** (Entry<Character, Integer> entry : counts.entrySet()) {  
 **if** (entry.getValue() == 1) {  
 **return** entry.getKey();  
 }  
 }  
 **throw new** RuntimeException(**"didn't find any non repeated Character"**);  
}

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

public class StringEquals {

//Two strings of equal length

public static int isTwoStringsEqual(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;

}

public static void main(String[] args) {

String s1 = "abcd";

String s2 = "aqcd";

boolean flag = isTwoStringsEqual(s1, s2) == 0 ? true : false;

System.out.println(flag);

}

}

# **String to number conversion**

public class String2Number {

public static void main(String[] args) {

String s = "1234";

long number=0;

char[] ch = s.toCharArray();

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

**number= number\*10 + ( ch[i]-'0' );**

}

System.out.println(number);

}

}

**String compression**

**Input String : aaabbbdccaabcc**

**Output String : a5b4c4d1**

import java.util.Map;

import java.util.TreeMap;

public class StringCompression {

public static void main(String[] args) {

String s = "aaabbbdccaabcc";

char[] ch = s.toCharArray();

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

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

char token = ch[i];

if( map.containsKey(token )) {

int val = map.get(token);

map.put(token, ++val);

}

else

map.put(token, 1);

}

System.out.println(map);

}

}

**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);  
  
 }  
}

**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");

}

}

**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);

}

}

**Arrays.copyOfRange()**

**public static int[] copyOfRange(int[] original, int from, int to)**

The source code is given below.

**public static int**[] copyOfRange(**int**[] original, **int** from, **int** to) {  
 **int** newLength = to - from;  
 **if** (newLength < 0)  
 **throw new** IllegalArgumentException(from + **" > "** + to);  
 **int**[] copy = **new int**[newLength];  
 System.*arraycopy*(original, from, copy, 0,  
 Math.*min*(original.**length** - from, newLength));  
 **return** copy;  
}

**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;  
}

**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 void** main(String[] args) {  
  
 String s1 = **"post"**;  
 String s2 = **"pots"**;  
  
 **int** val1 = 1;  
 **int** val2 = 1;  
 **for** (**int** i = 0; i < s1.length(); i++) {  
 val1 = val1\*s1.charAt(i);  
 val2 = val2\*s2.charAt(i);  
 }  
  
 System.***out***.println(val1+**"-----"**+val2);  
 }  
}