Sales by match

import java.io.\*;

import java.util.\*;

class Result {

public static int sockMerchant(int n, List<Integer> ar) {

// Write your code here

int pairs = 0;

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

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

int sock = ar.get(i);

if (set.contains(sock)) {

pairs++;

set.remove(sock);

} else {

set.add(sock);

}

}

return pairs;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int n = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> ar = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(Collectors.toList());

int result = Result.sockMerchant(n, ar);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

Simple array sum

import java.io.\*;

import java.util.\*;

class Result {

public static int simpleArraySum(List<Integer> ar) {

// write your code here

int sum = 0;

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

sum += ar.get(i);

}

return sum;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int arCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> ar = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(Collectors.toList());

int result = Result.simpleArraySum(ar);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

Breaking records

import java.io.\*;

import java.util.\*;

import java.util.stream.\*;

class Result {

public static List<Integer> breakingRecords(List<Integer> scores) {

int highestScore = scores.get(0);

int lowestScore = scores.get(0);

int numHighRecords = 0;

int numLowRecords = 0;

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

int currentScore = scores.get(i);

if (currentScore > highestScore) {

highestScore = currentScore;

numHighRecords++;

} else if (currentScore < lowestScore) {

lowestScore = currentScore;

numLowRecords++;

}

}

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

result.add(numHighRecords);

result.add(numLowRecords);

return result;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int n = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> scores = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(Collectors.toList());

List<Integer> result = Result.breakingRecords(scores);

bufferedWriter.write(result.stream().map(Object::toString).collect(Collectors.joining(" ")));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

Number Line Jumps

import java.io.\*;

import java.util.\*;

class Result {

public static String kangaroo(int x1, int v1, int x2, int v2) {

// If the second kangaroo is faster, it will eventually catch up

// and overtake the first kangaroo. In this case, they will never meet

// again, so we return "NO".

if (v2 >= v1) {

return "NO";

}

// Calculate the number of jumps required for the second kangaroo

// to catch up with the first kangaroo.

int jumps = (x2 - x1) / (v1 - v2);

// If the number of jumps is negative or the remaining distance

// is not divisible by the relative speed, the kangaroos will not meet

// again, so we return "NO".

if (jumps < 0 || (x1 + v1 \* jumps) != (x2 + v2 \* jumps)) {

return "NO";

}

// Otherwise, the kangaroos will meet again at the same point,

// so we return "YES".

return "YES";

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

int x1 = Integer.parseInt(firstMultipleInput[0]);

int v1 = Integer.parseInt(firstMultipleInput[1]);

int x2 = Integer.parseInt(firstMultipleInput[2]);

int v2 = Integer.parseInt(firstMultipleInput[3]);

String result = Result.kangaroo(x1, v1, x2, v2);

bufferedWriter.write(result);

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

Staircase

class Result {

public static void staircase(int n) {

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

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

System.out.print(" ");

}

for (int k = 1; k <= i; k++) {

System.out.print("#");

}

System.out.println();

}

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader (new InputStreamReader (System.in));

int n = Integer.parseInt(bufferedReader.readLine().trim());

Result.staircase(n);

bufferedReader.close();

}

}

Compare the triplets

import java.io.\*;

import java.util.\*;

import java.util.stream.\*;

class Result {

/\*

\* Complete the 'compareTriplets' function below.

\*

\* The function is expected to return an INTEGER\_ARRAY.

\* The function accepts following parameters:

\* 1. INTEGER\_ARRAY a

\* 2. INTEGER\_ARRAY b

\*/

public static List<Integer> compareTriplets(List<Integer> a, List<Integer> b) {

int aliceScore = 0;

int bobScore = 0;

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

int aliceRating = a.get(i);

int bobRating = b.get(i);

if (aliceRating > bobRating) {

aliceScore++;

} else if (bobRating > aliceRating) {

bobScore++;

}

}

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

result.add(aliceScore);

result.add(bobScore);

return result;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

List<Integer> a = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(Collectors.toList());

List<Integer> b = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(Collectors.toList());

List<Integer> result = Result.compareTriplets(a, b);

bufferedWriter.write(

result.stream()

.map(Object::toString)

.collect(Collectors.joining(" "))

+ "\n"

);

bufferedReader.close();

bufferedWriter.close();

}

}

Quicksort 1 partition

import java.io.\*;

import java.util.\*;

class Result {

/\*

\* Complete the 'quickSort' function below.

\*

\* The function is expected to return an INTEGER\_ARRAY.

\* The function accepts INTEGER\_ARRAY arr as parameter.

\*/

public static List<Integer> quickSort(List<Integer> arr) {

if (arr.size() <= 1) {

return arr;

}

int pivot = arr.get(0);

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

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

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

int element = arr.get(i);

if (element < pivot) {

left.add(element);

} else {

right.add(element);

}

}

List<Integer> sortedLeft = quickSort(left);

List<Integer> sortedRight = quickSort(right);

sortedLeft.add(pivot);

sortedLeft.addAll(sortedRight);

return sortedLeft;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int n = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(Collectors.toList());

List<Integer> result = Result.quickSort(arr);

bufferedWriter.write(

result.stream()

.map(Object::toString)

.collect(Collectors.joining(" "))

+ "\n"

);

bufferedReader.close();

bufferedWriter.close();

}

}

Between Two Sets

class Result {

/\*

\* Complete the 'getTotalX' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER\_ARRAY a

\* 2. INTEGER\_ARRAY b

\*/

public static int getTotalX(List<Integer> a, List<Integer> b) {

int lcm = a.get(0);

int gcd = b.get(0);

// Find the LCM of all integers in list a

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

lcm = getLCM(lcm, a.get(i));

}

// Find the GCD of all integers in list b

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

gcd = getGCD(gcd, b.get(i));

}

int count = 0;

// Count the number of multiples of the LCM that evenly divide the GCD

for (int i = lcm, j = 2; i <= gcd; i = lcm \* j, j++) {

if (gcd % i == 0) {

count++;

}

}

return count;

}

// Helper function to calculate the GCD of two integers

private static int getGCD(int a, int b) {

if (b == 0) {

return a;

}

return getGCD(b, a % b);

}

// Helper function to calculate the LCM of two integers

private static int getLCM(int a, int b) {

return (a \* b) / getGCD(a, b);

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

int n = Integer.parseInt(firstMultipleInput[0]);

int m = Integer.parseInt(firstMultipleInput[1]);

List<Integer> a = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList());

List<Integer> b = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList());

int total = Result.getTotalX(a, b);

bufferedWriter.write(String.valueOf(total));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

Super Reduced String

class Result {

/\*

\* Complete the 'superReducedString' function below.

\*

\* The function is expected to return a STRING.

\* The function accepts STRING s as parameter.

\*/

public static String superReducedString(String s) {

Stack<Character> stack = new Stack<>();

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

char c = s.charAt(i);

if (!stack.isEmpty() && stack.peek() == c) {

stack.pop();

} else {

stack.push(c);

}

}

if (stack.isEmpty()) {

return "Empty String";

} else {

StringBuilder sb = new StringBuilder();

while (!stack.isEmpty()) {

sb.append(stack.pop());

}

return sb.reverse().toString();

}

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

String s = bufferedReader.readLine();

String result = Result.superReducedString(s);

bufferedWriter.write(result);

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

Bill Division

import java.io.\*;

import java.util.\*;

class Result {

/\*

\* Complete the 'bonAppetit' function below.

\*

\* The function accepts following parameters:

\* 1. INTEGER\_ARRAY bill

\* 2. INTEGER k

\* 3. INTEGER b

\*/

public static void bonAppetit(List<Integer> bill, int k, int b) {

int totalCost = 0;

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

if (i != k) {

totalCost += bill.get(i);

}

}

int annaCost = totalCost / 2;

if (annaCost == b) {

System.out.println("Bon Appetit");

} else {

System.out.println(b - annaCost);

}

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

int n = Integer.parseInt(firstMultipleInput[0]);

int k = Integer.parseInt(firstMultipleInput[1]);

List<Integer> bill = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList());

int b = Integer.parseInt(bufferedReader.readLine().trim());

Result.bonAppetit(bill, k, b);

bufferedReader.close();

}

}

A Very Big Sum

class Result {

/\*

\* Complete the 'aVeryBigSum' function below.

\*

\* The function is expected to return a LONG\_INTEGER.

\* The function accepts LONG\_INTEGER\_ARRAY ar as parameter.

\*/

public static long aVeryBigSum(List<Long> ar) {

long sum = 0;

for (Long num : ar) {

sum += num;

}

return sum;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int arCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Long> ar = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Long::parseLong)

.collect(toList());

long result = Result.aVeryBigSum(ar);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

Plus Minus

class Result {

/\*

\* Complete the 'plusMinus' function below.

\*

\* The function accepts INTEGER\_ARRAY arr as parameter.

\*/

public static void plusMinus(List<Integer> arr) {

int n = arr.size();

double positiveCount = 0;

double negativeCount = 0;

double zeroCount = 0;

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

int value = arr.get(i);

if (value > 0) {

positiveCount++;

} else if (value < 0) {

negativeCount++;

} else {

zeroCount++;

}

}

double positiveFraction = positiveCount / n;

double negativeFraction = negativeCount / n;

double zeroFraction = zeroCount / n;

System.out.printf("%.6f\n", positiveFraction);

System.out.printf("%.6f\n", negativeFraction);

System.out.printf("%.6f\n", zeroFraction);

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList());

Result.plusMinus(arr);

bufferedReader.close();

}

}

Subarray Division

import java.io.\*;

import java.util.\*;

class Result {

/\*

\* Complete the 'birthday' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER\_ARRAY s

\* 2. INTEGER d

\* 3. INTEGER m

\*/

public static int birthday(List<Integer> s, int d, int m) {

int count = 0;

int sum = 0;

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

sum += s.get(i);

}

if (sum == d) {

count++;

}

for (int i = m; i < s.size(); i++) {

sum = sum - s.get(i - m) + s.get(i);

if (sum == d) {

count++;

}

}

return count;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int n = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> s = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList());

String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

int d = Integer.parseInt(firstMultipleInput[0]);

int m = Integer.parseInt(firstMultipleInput[1]);

int result = Result.birthday(s, d, m);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

Migratory Birds

import java.util.\*;

class Result {

/\*

\* Complete the 'migratoryBirds' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts INTEGER\_ARRAY arr as parameter.

\*/

public static int migratoryBirds(List<Integer> arr) {

int[] frequency = new int[6];

for(int birdType : arr) {

frequency[birdType]++;

}

int maxFrequency = 0;

int mostCommonBird = 0;

for(int i=1; i<=5; i++) {

if(frequency[i] > maxFrequency) {

maxFrequency = frequency[i];

mostCommonBird = i;

}

}

return mostCommonBird;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int arrCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList());

int result = Result.migratoryBirds(arr);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

Strange Counter 30m

class Result {

public static long strangeCounter(long t) {

long value = 3;

while (t > value) {

t -= value;

value \*= 2;

}

return value - t + 1;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

long t = Long.parseLong(bufferedReader.readLine().trim());

long result = Result.strangeCounter(t);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

Insertion Sort part 1

class Result {

/\*

\* Complete the 'insertionSort1' function below.

\*

\* The function accepts following parameters:

\* 1. INTEGER n

\* 2. INTEGER\_ARRAY arr

\*/

public static void insertionSort1(int n, List<Integer> arr) {

int unsorted = arr.get(n - 1);

int i = n - 2;

while (i >= 0 && arr.get(i) > unsorted) {

arr.set(i + 1, arr.get(i));

printArray(arr);

i--;

}

arr.set(i + 1, unsorted);

printArray(arr);

}

public static void printArray(List<Integer> arr) {

for (int i : arr) {

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

}

System.out.println();

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList());

Result.insertionSort1(n, arr);

bufferedReader.close();

}

}

Sherlock-and-valid-string

#!/bin/python3

import math

import os

import random

import re

import sys

from collections import Counter

# Complete the isValid function below.

def isValid(s):

d = Counter(s)

counts = Counter(d.values())

if len(counts) == 1:

return "YES"

elif len(counts) > 2:

return "NO"

else:

max\_v = max(counts.values())

k1, k2 = counts.keys()

if (max\_v == len(d) - 1):

if (abs(k1 - k2) == 1):

return "YES"

elif (min(k1, k2) == 1):

if counts[1] == 1:

return "YES"

else:

return "NO"

else:

return "NO"

else:

return "NO"

if \_\_name\_\_ == '\_\_main\_\_':

fptr = open(os.environ['OUTPUT\_PATH'], 'w')

s = input()

result = isValid(s)

fptr.write(result + '\n')

fptr.close()

Making Anagrams

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'makingAnagrams' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. STRING s1

\* 2. STRING s2

\*/

public static int makingAnagrams(String s1, String s2) {

int deletions = 0;

var LetterCount1 = new HashMap<Character, Integer>();

var LetterCount2 = new HashMap<Character, Integer>();

for (char c = 'a'; c <= 'z'; c++)

{

LetterCount1.put(c, 0);

LetterCount2.put(c, 0);

}

for (int i = 0; i < Math.max(s1.length(),s2.length()); i++)

{

if (i < s1.length())

LetterCount1.put(s1.charAt(i), LetterCount1.get(s1.charAt(i)) + 1);

if (i < s2.length())

LetterCount2.put(s2.charAt(i), LetterCount2.get(s2.charAt(i)) + 1);

}

for (char c = 'a'; c <= 'z'; c++)

deletions += Math.abs(LetterCount1.get(c) - LetterCount2.get(c));

return deletions;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

String s1 = bufferedReader.readLine();

String s2 = bufferedReader.readLine();

int result = Result.makingAnagrams(s1, s2);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'makingAnagrams' function below.

#

# The function is expected to return an INTEGER.

# The function accepts following parameters:

# 1. STRING s1

# 2. STRING s2

#

def makingAnagrams(s1, s2):

# initialize variables

deletions = 0

letter\_count\_1 = {}

letter\_count\_2 = {}

# count the occurrences of each letter in s1 and s2

for c in "abcdefghijklmnopqrstuvwxyz":

letter\_count\_1[c] = 0

letter\_count\_2[c] = 0

for c in s1:

letter\_count\_1[c] += 1

for c in s2:

letter\_count\_2[c] += 1

# calculate the number of deletions needed

for c in "abcdefghijklmnopqrstuvwxyz":

deletions += abs(letter\_count\_1[c] - letter\_count\_2[c])

return deletions

if \_\_name\_\_ == '\_\_main\_\_':

fptr = open(os.environ['OUTPUT\_PATH'], 'w')

s1 = input()

s2 = input()

result = makingAnagrams(s1, s2)

fptr.write(str(result) + '\n')

fptr.close()

Correctness and invariant

import java.io.\*;

import java.util.\*;

public class Solution {

public static void insertionSort(int[] A){

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

int value = A[i];

int j = i - 1;

while(j >= 0 && A[j] > value){

A[j + 1] = A[j];

j = j - 1;

}

A[j + 1] = value;

}

printArray(A);

}

static void printArray(int[] ar) {

for(int n: ar){

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

}

}

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int n = in.nextInt();

int[] ar = new int[n];

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

ar[i] = in.nextInt();

}

insertionSort(ar);

}

}

Running Time of Algorithms

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'runningTime' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts INTEGER\_ARRAY arr as parameter.

\*/

public static int runningTime(List<Integer> arr) {

int shifts = 0;

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

int value = arr.get(i);

int j = i - 1;

while (j >= 0 && arr.get(j) > value) {

arr.set(j + 1, arr.get(j));

j--;

shifts++;

}

arr.set(j + 1, value);

}

return shifts;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int n = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList());

int result = Result.runningTime(arr);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

Insertion sort part2

class Result {

public static void insertionSort2(int n, List<Integer> arr) {

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

int key = arr.get(i);

int j = i - 1;

while (j >= 0 && arr.get(j) > key) {

arr.set(j + 1, arr.get(j));

j = j - 1;

}

arr.set(j + 1, key);

System.out.println(arr.stream().map(String::valueOf).collect(joining(" ")));

}

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList());

Result.insertionSort2(n, arr);

bufferedReader.close();

}

}

Counting sort

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'countingSort' function below.

\*

\* The function is expected to return an INTEGER\_ARRAY.

\* The function accepts INTEGER\_ARRAY arr as parameter.

\*/

public static List<Integer> countingSort(List<Integer> arr) {

// Initialize the count array to store the count of each element

int[] count = new int[100];

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

count[arr.get(i)]++;

}

// Modify the count array to store the cumulative sum of each element

for (int i = 1; i < 100; i++) {

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

}

// Initialize the result array

List<Integer> result = new ArrayList<Integer>(arr.size());

// Traverse the input array in reverse order and fill the result array

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

int num = arr.get(i);

int index = count[num] - 1;

result.add(index, num);

count[num]--;

}

return result;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int n = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.map(Integer::parseInt)

.collect(toList());

List<Integer> result = Result.countingSort(arr);

bufferedWriter.write(

result.stream()

.map(Object::toString)

.collect(joining(" "))

+ "\n"

);

bufferedReader.close();

bufferedWriter.close();

}

}

TIME CONVERSION

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'timeConversion' function below.

\*

\* The function is expected to return a STRING.

\* The function accepts STRING s as parameter.

\*/

public static String timeConversion(String s) {

String f=s.substring(8);

int hr=Integer.parseInt(s.substring(0,2));

String militaryTime=s.substring(2,8);

String ans="";

if(f.equals("AM")){

if(hr==12){

ans="00"+militaryTime;

}

else

ans=s.substring(0,8);

}

if(f.equals("PM")){

if(hr==12){

ans="12"+militaryTime;

}

else

ans=String.valueOf(hr+12)+militaryTime;

}

return ans;

}

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

String s = bufferedReader.readLine();

String result = Result.timeConversion(s);

bufferedWriter.write(result);

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}