**Q1. CamelCase**

There is a sequence of words in [CamelCase](https://en.wikipedia.org/wiki/CamelCase) as a string of letters, having the following properties:

* It is a concatenation of one or more *words* consisting of English letters.
* All letters in the first word are *lowercase*.
* For each of the subsequent words, the first letter is *uppercase* and rest of the letters are *lowercase*.

Given , determine the number of words in s.

**Example**  
s = “oneTwoThree”

There are  words in the string: 'one', 'Two', 'Three'.

**Function Description**

Complete the *camelcase* function in the editor below.

camelcase has the following parameter(s):

* *string s*: the string to analyze

**Returns**

* *int:* the number of words in s

**Sample Input:**

saveChangesInTheEditor

**Sample Output:**

5

**Answer:-**

// Complete the 'camelcase' function below.

// The function is expected to return an INTEGER.

// The function accepts STRING s as parameter.

**int camelcase(char\* s) {**

**int count = 1;**

**int i;**

**for (i = 0; s[i] != '\0'; i++) {**

**if (s[i] >= 'A' && s[i] <= 'Z') {**

**count++;**

**}**

**}**

**return count;**

**}**

**Q2. MiniMaxSum**

Given five positive integers, find the minimum and maximum values that can be calculated by summing exactly four of the five integers. Then print the respective minimum and maximum values as a single line of two space-separated long integers.

**Example**  
arr = [1,3,5,7,9]

The minimum sum is 1+3+5+7=16 and the maximum sum is 3+5+7+9=24 . The function prints

**16 24**

**Function Description**

Complete the miniMaxSum  function with the following parameter(s):

* arr : an array of  integers

**Print**

Print two space-separated integers on one line: the minimum sum and the maximum sum of 4 of 5 elements .No value should be returned.

**Note** For some languages, like C, C++, and Java, the sums may require that you use a long integer due to their size.

**Sample Input :**

1 2 3 4 5

**Sample Output:**

10 14

***Answer :-***

**void miniMaxSum(int arr\_count, int\* arr) {**

**for (int i = 0; i < arr\_count - 1; i++) {**

**for (int j = 0; j < arr\_count - i - 1; j++) {**

**if (arr[j] > arr[j + 1]) {**

**int temp = arr[j];**

**arr[j] = arr[j + 1];**

**arr[j + 1] = temp;**

**}**

**}**

**}**

**long min\_sum = 0, max\_sum = 0;**

**for (int i = 0; i < 4; i++) {**

**min\_sum += arr[i];**

**max\_sum += arr[i + 1];**

**}**

**printf("%ld %ld\n", min\_sum, max\_sum);**

**}**

**Q3. Missing Numbers**

Given two arrays of integers, find which elements in the second array are missing from the first array.

**Example**  
arr=[7,2,5,3,5,3]

brr=[7,2,5,4,6,3,5,3]

The  array is the orginal list. The numbers missing are [4,6].

**Notes**

* If a number occurs multiple times in the lists, you must ensure that the frequency of that number in both lists is the same. If that is not the case, then it is also a missing number.
* Return the missing numbers sorted ascending.
* Only include a missing number once, even if it is missing multiple times.
* The difference between the maximum and minimum numbers in the original list is less than or equal to 100.

**Function Description**

Complete the *missingNumbers* function in the editor below. It should return a sorted array of missing numbers.

missingNumbers has the following parameter(s):

* *int arr[n]:* the array with missing numbers
* *int brr[m]:* the original array of numbers

**Returns**

* *int[]:* an array of integers

**Input Format**

There will be four lines of input:

 n- the size of the first list, arr  
The next line contains n space-separated integers arr[i]  
m - the size of the second list brr,   
The next line contains m space-separated integers brr[i]

*Sample Input:*

*10*

*203 204 205 206 207 208 203 204 205 206*

*13*

*203 204 204 205 206 207 205 208 203 206 205 206 204*

*Sample Output:*

*204 205 206*

***Answer:-***

*//Complete the 'missingNumbers' function below.*

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

*//The function accepts following parameters:*

*// 1. INTEGER\_ARRAY arr*

*// 2. INTEGER\_ARRAY brr*

**#define MAX 10001**

**int\* missingNumbers(int arr\_count, int\* arr, int brr\_count, int\* brr, int\* result\_count) {**

**int count\_arr[MAX] = {0};**

**int count\_brr[MAX] = {0};**

**int\* result = malloc(brr\_count \* sizeof(int));**

**\*result\_count = 0;**

**for (int i = 0; i < arr\_count; i++) {**

**count\_arr[arr[i]]++;**

**}**

**for (int i = 0; i < brr\_count; i++) {**

**count\_brr[brr[i]]++;**

**}**

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

**if (count\_brr[i] > count\_arr[i]) {**

**result[\*result\_count] = i;**

**(\*result\_count)++;**

**}**

**}**

**return result;**

**}**