1. Hosts and Total Number of Request

☆ Hosts and the Total Number of Requests

In this challenge, write a program to analyze a log file and summarize the results. Given a text file of an http requests log, list the number of requests from each host. Output should be directed to a file as described in the Program Description below.

The format of the log file, a text file with a *.txt* extension, follows. Each line contains a single log record with the following columns (in order):

1. The *hostname* of the *host* making the request.
2. This column's values are missing and were replaced by a hyphen.
3. This column's values are missing and were replaced by a hyphen.
4. A timestamp enclosed in square brackets following the format *[DD/mmm/YYYY:HH:MM:SS -0400]*, where *DD* is the day of the month, *mmm* is the name of the month, *YYYY* is the year, *HH:MM:SS* is the time in *24*-hour format, and *-0400* is the time zone.
5. The *request*, enclosed in quotes (e.g., *"GET /images/NASA-logosmall.gif HTTP/1.0"*).
6. The *HTTP response code*.
7. The total number of *bytes* sent in the response.

▶ Example log file entry

Function Description

Your function must create a unique list of hostnames with their number of requests and output to a file named *records_filename* where *filename* is replaced with the input *filename*. Each hostname should be followed by a space then the number of requests and a newline. Order doesn't matter.

Constraints

- The log file has a maximum of 2×10^5 lines of records.

▶ Input Format

▼ Sample Case 0

Sample Input 0

```
hosts_access_log_00.txt
```

Sample Output 0

Given *filename = "hosts_access_log_00.txt"*, process the records in *hosts_access_log_00.txt* and create an output file named *records_hosts_access_log_00.txt* shich contains the following rows:

```
burger.letters.com 3
d104.aa.net 3
unicomp6.unicomp.net 4
```

Explanation 0

The log file *hosts_access_log_00.txt* contains the following log records:

```
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0" 200 0
d104.aa.net - - [01/Jul/1995:00:00:13 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/KSC-logosmall.gif HTTP/1.0" 200 1204
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
```

When the data is consolidated, it confirms the following:

1. The host *unicomp6.unicomp.net* made *4* requests.
2. The host *burger.letters.com* made *3* requests.
3. The host *d104.aa.net* made *3* requests.


```
public void countRequests(String input, String output) {
    if (input == null || input.length() == 0) return;

    Map<String,Integer> result = new HashMap<>();
    try {
        FileReader file = new FileReader(input);
        BufferedReader br = new BufferedReader(file);
        String line = null;
        while ((line = br.readLine()) != null) {
            String hostname = line.split( regex: "-")[0].replace( target: " ", replacement: "");
            result.put(hostname,result.getOrDefault(hostname, defaultValue: 0) + 1);
        }

        File writer = new File(output);
        BufferedWriter out = new BufferedWriter(new FileWriter(writer));
        for (Map.Entry<String,Integer> entry: result.entrySet()) {
            out.write( str: entry.getKey() + " " + entry.getValue() + "\n");
        }
        out.flush();
        out.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

2. Missing Words

Missing Words

Given two strings, one is a subsequence if all of the elements of the first string occur in the same order within the second string. They do not have to be contiguous in the second string, but order must be maintained. For example, given the string "I like cheese", the words "I" and "cheese" are one possible subsequence of that string.

In this challenge, you will be given two strings, *s* and *t*, where *t* is a subsequence of *s*, report the words of *s, missing in t,* in the order they are missing. Revisiting the earlier example, if *s = I like cheese* and *t = like*, then *like* is the longest subsequence, and *[I, cheese]* is the list of missing words in order.

Function Description

Complete the function *missingWords* in the editor below. It must return an array of strings containing any words in *s* that are missing from *t* in the order they occur within *s*.

missingWords has the following parameter(s):

s: a sentence of space-separated words
t: a sentence of space-separated words

Constraints

- Strings *s* and *t* consist of English alphabetic letters (i.e., *a–z* and *A–Z*) and spaces only.
- $1 \le |t| \le |s| \le 10^6$
- *1* ≤ length of any word in *s* or *t* ≤ *15*
- It is guaranteed that string *t* is a subsequence of string *s*.

▶ **Input Format for Custom Testing**

▼ **Sample Case 0**

Sample Input 0

```
I am using HackerRank to improve programming
am HackerRank to improve
```

Sample Output 0

```
I
using
programming
```

Explanation 0

The missing words are:

1. I
2. using
3. programming

We add these words *in order* to the array *["I", "using", "programming"]*, then return this array as our answer.

YOUR ANSWER

We recommend you take a quick tour of our editor before you proceed. The timer will pause up to 90 seconds for the tour. Start tour

```java
public List<String> missingWords(String s, String t) {
    List<String> res = new ArrayList<>();
    if (s == null || s.length() == 0) return res;

    String[] parent = s.split( regex: " ");
    if (t == null || t.length() == 0) {
        for (String word: parent) {
            res.add(word);
        }
    }

    String[] sub = t.split( regex: " ");
    int i = 0, j = 0;
    while (i < parent.length && j < sub.length) {
        if (parent[i].equals(sub[j])) {
            i++;
            j++;
        } else {
            res.add(parent[i++]);
        }
    }

    while (i < parent.length) {
        res.add(parent[i++]);
    }

    return res;
}
```

3. Distinct Pairs

☆ Distinct Pairs

In this challenge, you will be given an array of integers and a target value. Determine the number of *distinct* pairs of elements in the array that sum to the target value. Two pairs (a, b) and (c, d) are considered to be distinct if and only if the values in sorted order do not match, i.e., (1, 9) and (9, 1) are indistinct but (1, 9) and (9, 2) are distinct.

For instance, given the array [1, 2, 3, 6, 7, 8, 9, 1], and a target value of 10, the seven pairs (1,9), (2,8), (3,7), (8, 2), (9, 1), (9, 1), and (1, 9) all sum to 10 and only three distinct pairs: (1, 9), (2, 8), and (3, 7).

Function Description

Complete the function *numberOfPairs* in the editor below. The function must return an integer, the total number of *distinct* pairs of elements in the array that sum to the target value.

numberOfPairs has the following parameter(s):

a[a[0],...a[n-1]]: an array of integers to select pairs from
k: target integer value to sum to

Constraints

- $1 \le n \le 5 \times 10^5$
- $0 \le a[i] \le 10^9$
- $0 \le k \le 5 \times 10^9$

▸ **Input Format for Custom Testing**

▾ **Sample Case 0**

Sample Input 0

```
6
1
3
46
1
3
9
47
```

Sample Output 0

```
1
```

Explanation 0

a = [1, 3, 46, 1, 3, 9], k = 47

There are *4* pairs of unique elements where *a[i] + a[j] = k*:

1. *(a[0] = 1, a[2] = 46)*
2. *(a[2] = 46, a[0] = 1)*
3. *(a[2] = 46, a[3] = 1)*
4. *(a[3] = 1, a[2] = 46)*

In the list above, all four pairs contain the same values. We only have *1 distinct* pair, *(1, 46)*.

▸ **Sample Case 1**

```java
public static int numberOfPairs(List<Integer> a, long k) {
    int count = 0;
    Collections.sort(a); // Sort array elements

    int l = 0;
    int r = a.size() - 1;
    Set<Integer> set = new HashSet<>();
    while(l < r) {
        int first = a.get(l);
        int second = a.get(r);
        if(!set.contains(first) && !set.contains(second) && first + second == k) {
            count++;
            l++;
            r--;
            set.add(first);
            set.add(second);
        }
        else if(set.contains(first) || first + second < k)
            l++;
        else
            r--;
    }
    return count;
}
```

4. Stock Analysis

☆ Stock Analysis

An investor makes buying and selling decisions based on a set of observations that are recorded and analyzed. To have the most valid data, investors get data from multiple sources that are retrieved in order from least to most preferred.

Data is aggregated using the *eliminate algorithm* which arrives at a single final value to use for each parameter. In short, as new parameters, they are added to the list. If a later, thus more preferred, data source provides a value for a parameter that is already in your list, its value supersedes the one from an earlier source. The eliminate algorithm is described below for data that relates to parameter *Pi* received from two sources:

1: If a parameter *Pi* is present in both source 1 and source 2, the parameter from the higher priority source, source 2, is used in the final parameter list

2: If a parameter *Pi* is present only in one of the sources, it is directly added to the final parameter list

The result of performing the above two operations until all the parameters from source 1 and source 2 are exhausted is the result of Eliminate-algorithm(source 1, source 2). Each time a new value for a parameter is encountered from a higher preferred site, the old data is superseded. Assuming three sources S1, S2, S3, Eliminate-algorithm(S1, S2, S3) = Eliminate-algorithm(Eliminate-algorithm(S1,S2), S3).

Given a list of sources S1, S2…..Sn, find the final parameter list given by Eliminate-algorithm(S1, S2….Sn). Maintain your results in the order a key was first encountered.

A very simple example is that you receive only a *rating* parameter of buy, sell or hold from three sources in increasing order of preference: *[buy, sell, hold]*. A 'buy' rating comes in from source *1*, immediately superseded by 'sell' from source *2*, immediately superseded by 'hold' from source *3*. The final rating is the only one that hasn't been superseded, so you use 'hold' as the rating for the analysts to see.

As a more complex example, you receive data from two sources as follows:

```
P1:x P2:y P5:z
P1:b P5:a P3:w
```

The first row represents source *1*, the second, source *2* and the second source is preferred. Start the analysis at source *1*. Enter all of those items into our list, now *results =[[P1,x],[P2,y],[P5,z]]* and move on to source *2*. The first datapoint is for *key P1* and that is already in the list. As source *2* is higher authority, replace results.index['P1'][1] with the new value 'b'. Do the same with *P5*. Next is a new key, *key P3*, so it is added to the list: *result = [[P1,b],[P2,y],[P5,a],[P3,w]]*. Return a list of the second data element from each element, *final = [b,y,a,w]*.

▶ Input Format For Custom Testing

▼ Sample Case 0

Sample Input 0

```
2
3
P1:a P3:b P5:x
P1:b P2:q P5:x
```

Sample Output 0

```
b
b
x
q
```

Explanation 0

Final parameter list
P1 b (Source 2)
P3 b (Source 1)
P5 x (Source 2)
P2 q (Source 2)

```
private List<String> computeParameterValue(List<List<String>> sources) {
    List<String> res = new ArrayList<>();
    if (sources == null || sources.size() == 0) return res;
    Map<String, String> map = new LinkedHashMap<>();

    for (List<String> source: sources) {
        for (String parameters: source) {
            String parameter = parameters.split( regex: ":")[0];
            String value = parameters.split( regex: ":")[1];
            map.put(parameter,value);
        }
    }

    res.addAll(map.values());
    return res;
}
```

5. K subsequence

☆ K-Subsequences

We define a *k-subsequence* of an array as follows:

- It is a subsequence of contiguous elements in the array, i.e. a subarray.
- The *sum* of the subsequence's elements, *s*, is evenly divisible by *k* (i.e.: *s % k = 0*).

Given an array of integers, determine the number of *k-subsequences* it contains. For example, *k = 5* and the array *nums = [5, 10, 11, 9, 5]*. The *10 k-subsequences* are: *{5}, {5, 10}, {5, 10, 11, 9}, {5, 10, 11, 9, 5}, {10}, {10, 11, 9}, {10, 11, 9, 5}, {11, 9}, {11, 9, 5}, {5}.*

Function Description

Complete the function *kSub* in the editor below. The function must return a long integer that represents the number of *k-subsequences* in the array *nums*.

kSub has the following parameter(s):

k: an integer that the sum of the subsequence must be divisible by
nums[nums[0],...nums[n-1]]: an array of integers

Constraints

- $1 \le n \le 3 \times 10^5$
- $1 \le k \le 100$
- $1 \le nums[i] \le 10^4$

▶ **Input Format For Custom Testing**

▼ **Sample Case 0**

Sample Input For Custom Testing

Sample Input 0

```
3
5
1
2
3
4
1
```

Sample Output 0

```
4
```

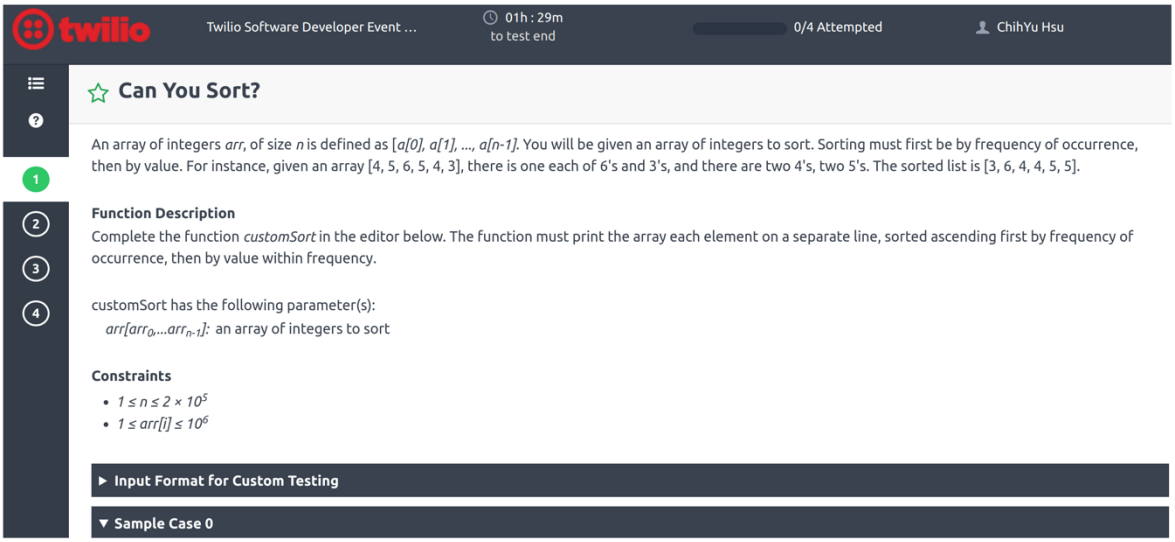
Explanation 0

The *4* contiguous subsequences of *nums* having sums that are evenly divisible by *k = 3* are *{3}, {1, 2}, {1, 2, 3}, {2, 3, 4}*.

```java
private long kSub(int k, List<Integer> nums) {
    int[] sum = new int[nums.size()];
    long count = 0;
    sum[0] = nums.get(0);
    for (int i = 1; i < nums.size(); i++) {
        sum[i] = sum[i - 1] + nums.get(i);
    }

    int[] modsOfK = new int[k];
    for (int i = 0; i < sum.length; i++) {
        //To handle with negative
        int mod = (sum[i] % k + k) % k;
        if (mod == 0) {
            count++;
        }
        count += modsOfK[mod];
        modsOfK[mod] += 1;
    }
    return count;
}
```

6. Can you sort?

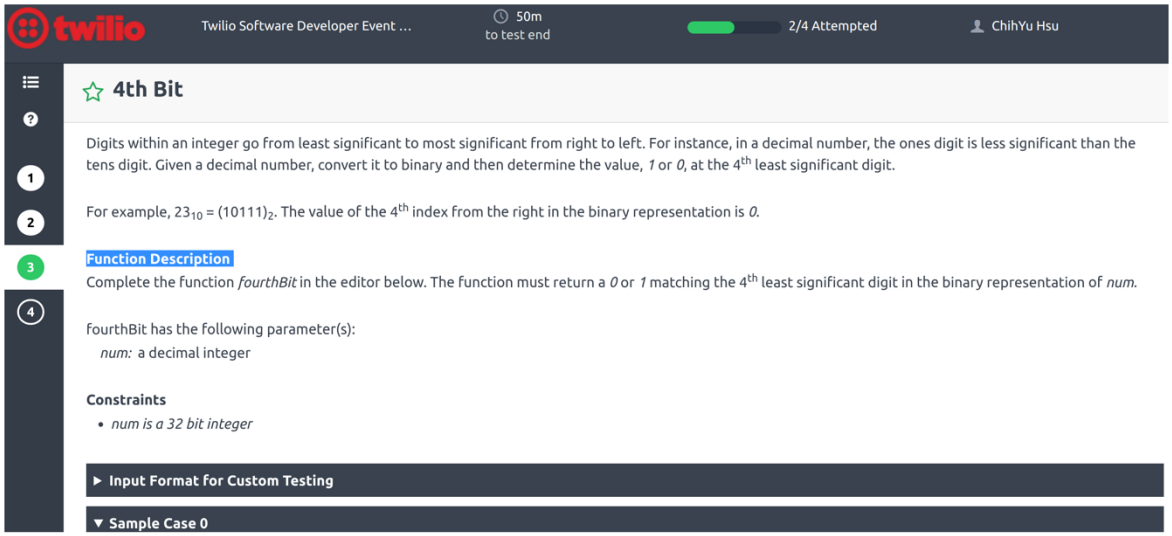



```
private List<Integer> sortArr(List<Integer> arr) {
    if (arr == null || arr.size() <= 1) return arr;
    List<Integer> res = new ArrayList<>();
    Map<Integer,Integer> map = new TreeMap<>();
    for (Integer num: arr) {
        map.put(num, map.getOrDefault(num, defaultValue: 0) + 1);
    }

    List<Map.Entry<Integer,Integer>> list = new ArrayList<>(map.entrySet());
    Collections.sort(list, new Comparator<Map.Entry<Integer, Integer>>() {
        @Override
        public int compare(Map.Entry<Integer, Integer> o1, Map.Entry<Integer, Integer> o2) {
            if (o1.getValue() < o2.getValue()) {
                return -1;
            } else  if (o1.getValue() > o2.getValue()) {
                return 1;
            } else {
                return o1.getKey().compareTo(o2.getKey());
            }
        }
    });

    for (Map.Entry<Integer,Integer> entry:list) {
        for (int i = 0; i < entry.getValue(); i++) {
            res.add(entry.getKey());
        }
    }
    return res;
}
```

7. 4th Bits




```java
private int fourthBits(int num) {
    int count = 0, remain = 0;
    while (num != 0 && count < 4) {
        remain =  num % 2;
        num /= 2;
        count++;
    }
    if (count < 4) return 0;
    return remain;
}
```