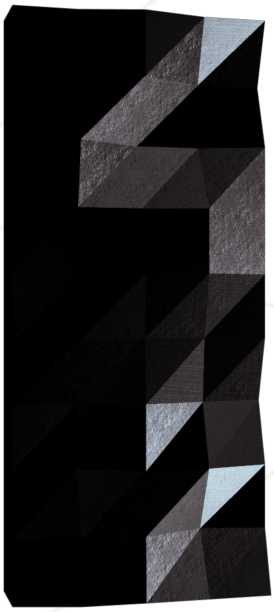

Starfleet

Interview

Staff 42 pedago@42.fr

Summary: This document is an interview question for the Starfleet Piscine.

Contents

I General rules **2**
I.1 During the interview . . . 3

II Array of integers in range **4**
II.1 Interview question . . . 4
II.2 Acceptable answers (no constraint) . . . 5
II.2.1 Brute force solution . . . 5
II.2.2 Sort the array . . . 5
II.2.3 Solution with hash table . . . 6
II.3 Follow up question . . . 6
II.3.1 Hints . . . 6
II.4 Best solution . . . 6
II.4.1 Counting directly in the array . . . 6

Chapter I

General rules

- The interview should last between `45 minutes`.
- Both the interviewer and the interviewed student must be present.
- The interviewed student should write his code using a `whiteboard`, with the language of her/his choice.
- At the end of the interview, the interviewer evaluates the student based on the provided criteria.

Read carefully the interview question and solutions, and make sure you `understand` them before the interview. You can't share this document with other students, as they might be interviewed on the same question. Giving them the answer would prevent them from having to solve an unknown question during an interview.

I.1 During the interview

During the interview, we ask you to :

- Make sure the interviewed student `understands` the question.
- Give her/him any `clarification` on the subject that she/he might need.
- Let her/him come up with a solution before you guide her/him to the best solution given the constraints (time and space).
- Ask the student what is the `complexity` of her/his algorithm ? Can it be improved and how ?
- `Guide` her/him to the best solution without giving the answer. You may refer to the `hints` for that.
- You want to evaluate how the interviewed student thinks, so ask her/him to `explain everything` that she/he thinks or writes (there should be no silences).
- If you see a mistake in the code, wait untill the end and give her/him a chance to correct it by her/himself.
- Ask the student to show how the algorithm works on an `example`.
- Ask the student to explain how `limit cases` are handled.
- Bring out to the student any mistake she/he might have done.
- Give `feedback` on her/his performances after the interview.
- Be `fair` in your evaluation.

As always, stay mannerly, polite, respectful and constructive during the interview. If the interview is carried out smoothly, you will both benefit from it !

Chapter II

Array of integers in range

II.1 Interview question

You are given an array of n integers which can contain integers from 0 to (n-1) only. Some elements can be repeated multiple times and some other elements can be absent from the array.

Write a running code to print the count of every element in the range along with the element number (present in the array or not).

NOTE: The array isn't necessarily sorted.

EXAMPLE :

```
Input : {4, 3, 4, 0, 0} with n = 5

Output :
Element 0 [count : 2]
Element 1 [count : 0]
Element 2 [count : 0]
Element 3 [count : 1]
Element 4 [count : 2]
```

II.2 Acceptable answers (no constraint)

II.2.1 Brute force solution

- For each element from 0 to (n - 1), count the number of occurrences of the element in the array.

```
O(n^2) time , O(1) space, where n is the number of elements in the array
```

Code:

```
void countOccurrences(int *arr, int n) {
        int count;

        for (int i = 0; i < n; i++) {
                count = 0;
                for (int j = 0; j < n; j++) {
                        if (arr[i] == arr[j])
                                count++;
                }
                printf("Element %d [count : %d]\n", i, count);
        }
}
```

II.2.2 Sort the array

- Sort the array (which can be done in-place in O(nlogn) time).
- Go through the sorted array and count the number of repeated elements. This is done in O(n) time, but since you have to sort the array first, the algorithm is still in O(nlogn).

```
O(nlogn) time , O(1) space, where n is the number of elements in the array
```

Code:

```
void countOccurrences(int *arr, int n) {
        int k, count;

        sort(arr); // sorting takes O(nlogn) time
        k = 0;
        count = 0;
        for (int i = 0; i < n; i++) {
                if (k < n && arr[k] == i) {
                        count++;
                        k++;
                } else {
                        printf("Element %d [count : %d]\n", i, count);
                        count = 0;
                }
        }
}
```

II.2.3 Solution with hash table

- Create a hash table with n elements (hash index from 0 to n-1).
- For each element in the array, increment the corresponding cell in the hash table.

```
O(n) time , O(n) space, where n is the number of elements in the array
```

Code:

```
void countOccurrences(int *arr, int n) {
        int hashTable[n] = {0};

        for (int i = 0; i < n; i++) {
                hashTable[i]++;
        }
        for (int i = 0; i < n; i++) {
                printf("Element %d [count : %d]\n", i, hashTable[i]);
        }
}
```

II.3 Follow up question

Provide a solution in-place with the best possible runtime.

Your algorithm must takes `O(1) space` apart from the input array and `O(n) time`.

II.3.1 Hints

- What is the particularity of this array?
- Can you use the array as a hashtable, knowing the range of its elements?

II.4 Best solution

II.4.1 Counting directly in the array

Use the fact that every elements in the array is in the range from 0 to (n - 1). You can use the array as a hash table and count the occurrences by adding the array size (n), and still be able to access the element in the cell with a modulo n.

This is done in 2 scans of the array :

- For each element in the array, add n to the cell with the corresponding index.
- Then, divide each element value by n to get the frequency of occurrence.

```
O(n) time , O(1) space, where n is the number of elements in the array
```

code:

```
void countOccurrences(int *arr, int n) {
        for (int i = 0; i < n; i++) {
                arr[arr[i] % n] += n;
        }
        for (int i = 0; i < n; i++) {
                printf("Element %d [count : %d]\n", i, arr[i] / n);
        }
}
```