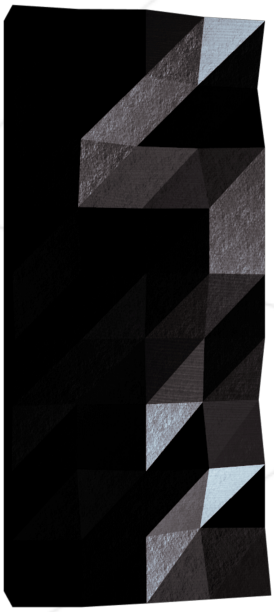

Starfleet

Interview

Staff 42 pedago@42.fr

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

Contents

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

Repetitions in array

II.1 Interview question

`Given a sorted array of integers, write a function that will return the number with the biggest number of repetitions.`

NOTE : in case of equity, return either one.

II.2 Acceptable answers (no constraint)

II.2.1 With a hash table

You can use a hash table to count the occurences of every numbers in the array. It would take O(n) time and O(n) space. This solution does not use the fact that the array is sorted. We can do better!

```
O(n) time , O(n) space
```

II.2.2 Linear count

This can be done in a single pass without using extra space. For each element of the array :

- If it is the same as the previous one, increment the counter.
- Otherwise, if the count is higher than the maximum number of repetitions, set it as the new maximum. Re-initialize the counter to zero.

```
O(n) time , O(1) space
```

Code:

```
int repetitions(int *arr, int n) {
        int res;
        int max_count;
        int current_count;

        if (n < 1)
                return (INT_MIN);
        res = arr[0];
        current_count = 1;
        max_count = 1;
        for (int i = 1; i <= n; i++) {
                if (i < n && arr[i] == arr[i - 1]) {
                        current_count++;
                } else {
                        if (current_count > max_count) {
                                max_count = current_count;
                                res = arr[i - 1];
                        }
                        current_count = 1;
                }
        }
        return (res);
}
```

II.3 Follow up question

Refine the solution to be more efficient.

II.3.1 Hints

- Do you really need to visit each and every element of the array ?
- Can you use a "divide and conquer" algorithm to maximize the number of elements that you do not need to visit ?

II.4 Best solution

II.4.1 Divide and conquer

The method is to check the occurences of the middle element and then divide the remaining parts into two arrays (in a "binary search" fashion). Repeat these steps for each subarrays until the length of the subarray is lower than the maximum number of repetitions.

In order to guarantee an optimal runtime, we use a "Breadh First Search" (BFS) algorithm using a queue.

EXAMPLE :

```
Input : {1, 1, 2, 5, 5, 5, 6, 9, 9, 9, 9}

Step 1 : {1, 1, 2}, {6, 9, 9, 9, 9}, res = 5, count = 3

Step 2 : {6, 9, 9, 9, 9}, res = 5, count = 3

Step 3 : {6}, res = 9, count = 4

Step 4 : res = 9, count = 4
```

```
O(n/k) time , O(n/k) space
where k is the length of the longest sequence
worst case O(n) time
```

code:

```
struct s_node *newNode(int start, int end) {
        struct s_node *node;

        if (!(node = (struct s_node *)malloc(sizeof(struct s_node))))
                return (NULL);
        node->start = start;
        node->end = end;
        return (node);
}

int repetitions(int *arr, int n) {
        int res;
        int max_count;
        int current_count;
        struct s_queue *queue;
        struct s_node *node;
        int mid, left, right;

        if (n < 1)
                return (INT_MIN);
        res = arr[0];
        max_count = 1;
        queue = queueInit();
        queueAdd(queue, newNode(0, n - 1));
        while(!queueIsEmpty(queue)) {
                node = queueRemove(queue);

                if (node->end - node->start < max_count) {
                        free(node);
                        continue;
                }

                mid = (node->start + node->end) / 2;
                current_count = 1;
                left = node->start;
                right = node->end;

                for (int i = mid; i > node->start; i--) {
                        if (arr[i] == arr[i - 1]) {
                                current_count++;
                        } else {
                                left = i - 1;
                                break ;
                        }
                }

                for (int i = mid; i < node->end; i++) {
                        if (arr[i] == arr[i + 1]) {
                                current_count++;
                        } else {
                                right = i + 1;
                                break ;
                        }
                }

                if (current_count > max_count) {
                        max_count = current_count;
                        res = arr[mid];
                }

                queueAdd(queue, newNode(node->start, left));
                queueAdd(queue, newNode(right, node->end));
                free(node);
        }
        return (res);
}
```