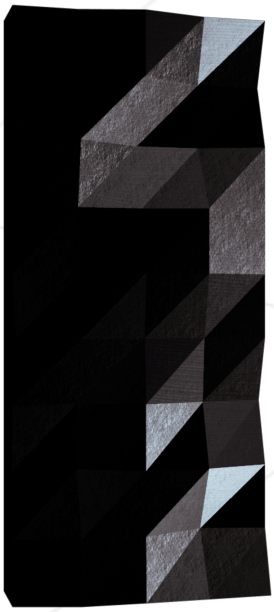

Starfleet

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

Sort stack

II.1 Interview question

`Write a program to sort a stack such that the smallest items are on the top. You can use an additional temporary stack, but you may not copy the elements into any other data structure (such as an array). The stack supports the following operations: push, pop, peek, and isEmpty.`

II.1.1 Hints

- One way of sorting the stack is to iterate through the stack and insert each element into a new stack in sorted order.
- Imagine your secondary stack is sorted. Can you insert elements into it in sorted order? You might need some extra storage. What could you use for extra storage?
- Keep the secondary stack in sorted order, with the biggest elements on the top. Use the primary stack for additional storage.

II.2 Best solution

II.2.1 Insert in order

One approach is to implement a rudimentary sorting algorithm : search through the entire stack to find the minimum element and then push that onto a new stack. Then, find the new minimum element and repeat. Unfortunately, this requires two additional stacks. We can do better!

Rather than searching for the minimum repeatedly, we can sort stack1 by inserting each element from stack1 in order into stack2:

- Pop element from stack1 and store it into a temporary variable tmp.
- Push all elements of stack2 greater than tmp into stack1.
- Push tmp into stack2.

Repeat these steps until stack1 is empty, and then push all the elements back into stack1.

```
O(n^2) time , O(1) space
```

code:

```
struct s_node {
        int           value;
        struct s_node *next;
};

struct s_stack {
        struct s_node *top;
};

struct s_stack *init(void);

int pop(struct s_stack *stack);

void push(struct s_stack *stack, int value);

int peek(struct s_stack *stack);

int isEmpty(struct s_stack *stack);


void sort(struct s_stack *stack) {
        struct s_stack *stack2 = init();
        int tmp;

        while (!isEmpty(stack)) {
                // insert each element in stack
                // in sorted order into stack2
                tmp = pop(stack);
                while (!isEmpty(stack2) && peek(stack2) > tmp) {
                        push(stack, pop(stack2));
                }
                push(stack2, tmp);
        }

        //copy the elements from stak2 back into stack
        while(!isEmpty(stack2)) {
                push(stack, pop(stack2));
        }
}
```