# Solutions for Data Structures and Algorithms Spring 2023 — Problem Sets

By Dmitriy Okoneshnikov, B22-DSAI-04

January 30-31, 2023

## Week 2. Problem set

1. In [Cormen, Section 16.1], a total amortised time complexity is computed for a sequence of $n$ Increment operations starting from an initially zero counter. Compute the total amortised time complexity for a sequence of $n$ Increment operations starting from a non-zero counter with value $k$. No justification is required, provide your answer using big-Oh notation.

   **Answer.** $O(1)$

2. In [Cormen, Section 16.1], a stack with an extra operation Multipop is discussed. Provide an example of a sequence of Push, Pop, and Multipop operations on an initially empty stack, such that

   (a) the actual total cost of the sequence is 5,

   (b) the sequence contains one Pop, and one Multipop, and 3 Push operations, in some order,

   (c) Multipop($k$) must be used with $k \geq 2$.

   No justification is required for this exercise.

   **Answer.** Push, Multipop(2), Push, Push, Pop

3. Consider `StackQueue`, an implementation of the Queue ADT using a pair of stacks: a *front stack* and a *rear stack*:

   (a) A queue is empty when both stacks are empty.

   (b) To perform `offer(e)`, we `push(e)` into the rear stack.

   (c) To perform `poll()`, we `pop()` from the front stack if it is not empty. If the front stack is empty, we repeatedly `pop()` elements from the rear stack and `push` them onto the front stack, until the rear stack is empty. Finally, we `pop()` from the front stack, since it is no longer empty.

   Perform amortised time complexity analysis for a sequence of `offer(e)` and `poll()` operations performed on an initially empty `StackQueue`. You **must** apply either the accounting method or the potential method.

   Assume that the execution cost (time) of `push(e)`, `pop()`, `isEmpty()` for the underlying stack implementation is 1.

   **Solution.**

   Amortised time will be calculated using the accounting method.

From the problem statement we know that `offer` $\equiv$ `push` and `poll` $\equiv$ `pop` + `isEmpty` + `push`, so we need to find the amortized costs for the stack operations, and then use them to find the amortized costs for `offer` and `poll`.

Actual costs of the stack operations (from the problem statement) are the following:

| operation | $c_i$ |
|---|---|
| `push` | 1, |
| `pop` | 1, |
| `isEmpty` | 1, |

Let us assign the following amortized costs:

| operation | $\hat{c_i}$ |
|---|---|
| `push` | 5, |
| `pop` | 0, |
| `isEmpty` | 1, |

$\sum_{i=0}^{n} \hat{c_i} \geq \sum_{i=0}^{n} c_i$

$5 + 0 + 1 \geq 1 + 1 + 1$

$6 \geq 3 \Rightarrow$ true

Let us see how to pay for any sequence of queue operations by charging the amortized costs.

**Enqueue.**

(a) When a new element is enqueued, we use 1 unit of cost for the enqueue and we leave the other 4 units of cost for future.

**Dequeue.**

(a) When an element is dequeued, we use 1 unit of cost to check if the front stack is empty.

(b) Front stack is empty.

  i. On each iteration check if rear stack is empty. This costs 1 unit of cost, each element will have 3 units of cost left.

  ii. Pop elements from the rear stack using another 1 unit of cost, each element will have 2 units of cost left.

  iii. Push elements into the front stack using 1 unit of cost, each element will have 1 unit of cost left.

  iv. Pop the element from the front stack using the last 1 unit of cost that this element has.

(c) Front stack is not empty.

  i. Pop the element from the front stack using 1 unit of cost.

Therefore, a sequence of $n$ `offer` operations is $5 \cdot n = O(n)$ and a sequence of $n$ `poll` operations is $1 \cdot n = O(n)$ (we have to pay only for one `isEmpty` operation).

# References


[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms, Fourth Edition.* The MIT Press 2022