Week 2. Problem set

1. In [CLRS, §16.1], a stack with an extra operation MULTIPOP is discussed. What is the total cost c of executing n of the stack operations PUSH, POP, and MULTIPOP, assuming that the stack begins with k objects? The answer must consist of exact lower and upper bounds given as formulae in terms of n, k (not asymptotic complexity!). Provide a brief justification (2–4 sentences). Formulas and equations without explanation will not be accepted.

2. A sequence of PUSH, POP, and CLEAN operations is performed on an initially empty stack. When an element is first PUSHed to the stack, it is marked as **new**. The CLEAN operations inspects all elements of the stack:

   - Every **new** element is marked as **old** (in constant time).
   - Every **old** element is removed from the stack (in constant time).

   Perform amortized time complexity analysis using the **accounting method** [CLRS, §16.2] for a sequence of PUSH, POP, and CLEAN operations performed on an initially empty stack:

   (a) Specify actual cost, amortized cost, and accumulated credit for each operation. Assume that n_i is the size of stack before operation and k_i is number of **old** elements in the stack.

Operation	Actual cost (c_i)	Amortized cost ($\widehat{c_i}$)	Credit	Justification
PUSH				
POP				
CLEAN				

   (b) Prove that the total amortized cost of a sequence of n operations provides an upper bound on the total actual cost of the sequence.

   (c) Write down the asymptotic complexity for a sequence of n operations.

3. Solve the previous exercise using the **potential method** [CLRS, §16.3]:

   (a) Define the potential function Φ on a stack; the potential function may depend on the size n of the stack and on the number k of **old** elements in the stack;

   (b) Compute $\Phi(D_i) - \Phi(D_{i-1})$ for each possible ith operation (PUSH, POP, CLEAN)

   (c) Compute amortized cost for CLEAN using your potential function;

   (d) Write down amortized asymptotic complexity for CLEAN.

4. **(+0.5% extra credit)** Show how to implement an improved version of CLEAN operation from the previous exercise, such that it works in $\Theta(k)$ (worst case) where k is the number of **old** elements in the stack:

   (a) Provide the pseudocode of CLEAN for an array based implementation of Stack

   (b) Provide the pseudocode of CLEAN for a linked list based implementation of Stack

   (c) Explain briefly for each implementation why it works in $\Theta(k)$

5. **(+1% extra credit)** A sorted collection of n integers is represented by a linked list of k sorted arrays. The arrays are of sizes $2^{b_0}, 2^{b_1}, \ldots, 2^{b_k}$, such that $b_0 < b_1 < \ldots < b_k$.

   To ADD an integer i in a sorted collection, we add a singleton array with i to the beginning of the list of arrays. Then, to ensure the invariant, we resize the arrays: going from smaller arrays to larger, if two smallest arrays have the same size, we MERGE them (using linear time merging as in MERGE-SORT) and repeat the process until the smallest array is unique.

   For example,

   - a collection $\{1, 2, 3, 4, 5, 8, 9, 10, 11, 12\}$ can be represented by a list of three arrays:

     $$\boxed{2} \to \boxed{1, 3} \to \boxed{4, 5, 6, 8, 9, 10, 11, 12}$$

   - inserting 7 in this collection, we first add a singleton array $\boxed{7} \to \boxed{2} \to \boxed{1, 3} \to \boxed{4, 5, 6, 8, 9, 10, 11, 12}$
   - then, since we have two arrays of size 1, we MERGE them $\boxed{2, 7} \to \boxed{1, 3} \to \boxed{4, 5, 6, 8, 9, 10, 11, 12}$
   - then, since we have two arrays of size 2, we MERGE them $\boxed{1, 2, 3, 7} \to \boxed{4, 5, 6, 8, 9, 10, 11, 12}$
   - now, since we have only one array of the smallest size, we stop.

   Perform amortised time complexity analysis using the **accounting method** for a sequence of n ADD operations performed on an initially empty sorted collection:

   (a) Specify actual cost, amortized cost, and accumulated credit for ADD[1]; the amortized cost may depend on the number n of operations in the sequence;

   (b) Prove that the total amortized cost of a sequence of n operations provides an upper bound on the total actual cost of the sequence.

   (c) Write down the asymptotic complexity for a sequence of n operations.

   (d) Write down amortized asymptotic complexity for ADD.

References

[CLRS] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C., 2022. *Introduction to algorithms, Fourth Edition*. MIT press.

[1] Hint: take inspiration in example for incrementing a binary counter example [CLRS, Section 16.2]; for each element in the collection there should be enough credit for all merge events for that element.