Graded Assignment 2

Due date: April 9, at 20:00

This is an individual assignment. You may discuss it with others, but your formulations, your code, and all the required material must be written on your own. In any case, you must acknowledge the sources used and clearly mention any help received from colleagues.

Exercise 1. *(30% grade)*

Implement a data structure in Python that stores a sequence of numbers S and that supports the following operations: `print_all(S)` prints all elements in order; `append(S,a)` adds an element at the end; `pop(S)` removes the last element; and `find_min(S)` returns the minimal value in the sequence. Apart from `print_all(S)` that runs in $\mathcal{O}(n)$ time, all other operations must run in constant time.
Can your data structure also support a `delete_min()` algorithm that removes one of the minimal elements in constant time? Briefly justify your answer.

Exercise 2. *(50% grade)*

Given an array A of n numbers, we are interested in counting or printing all the numbers in a given interval $[a, b)$. For a given input array, we want to run these algorithms several times over different intervals. We therefore consider ways to pre-process the array into a specialized data structure X that would speed up the counting and printing algorithms.

Question 1: implement algorithms `preprocess(A)`, `range_count(X,a,b)`, and `range_search(X,a,b)`, in Python, such that `X = preprocess(A)` runs in $\mathcal{O}(n \log n)$ time and produces a data structure X such that `range_count(X,a,b)` runs in $\mathcal{O}(\log n)$ time, and `range_search(X,a,b)` runs in $\mathcal{O}(\log n + k)$ time where k is number of elements in the given $[a, b)$ interval.

Question 2: Consider a special case in which all elements are natural numbers between 0 and m. Implement a new data structure and corresponding algorithms such that `X = preprocess(A)` runs in $\mathcal{O}(n + m)$ time, and `range_count(X,a,b)` runs in constant time.

Exercise 3. *(20% grade)*

Implement an algorithm that transforms a min-heap H into a max-heap in-place.