COMP3506/7505: Regular Exercise Set 6
Prepared by Yufei Tao

Problems marked with an asterisk may be difficult.

Problem 1. Prove that our algorithm for the “stack-with-array problem” takes \( O(n) \) time to process any sequence of \( n \) operations where each operation can be either a push or a pop.

Problem 2. Let \( S \) be a multi-set of \( n \) integers. Define the frequency of an integer \( x \) as the number of occurrences of \( x \) in \( S \). Design an algorithm to produce an array that sorts the distinct integers in \( S \) by frequency. Your algorithm must terminate in \( O(n) \) expected time. For example, suppose that \( S = \{75, 123, 65, 75, 9, 9, 65, 9, 93\} \). Then you should output \((123, 93, 65, 75, 9)\). Note that if two integers have the same frequency, their relative ordering is unimportant. For example, \((93, 123, 75, 65, 9)\) is another legal output.

Problem 3* (Textbook Exercise 17.3-7). Suppose that we want to implement the following two operations on a set \( S \) of integers (\( S \) is empty at the beginning):

- Insert(\( e \)): Add a new integer \( e \) into \( S \) (you are assured that \( e \) is not already in \( S \)).
- Delete-Half: Delete the \( \lceil |S|/2 \rceil \) smallest elements from \( S \).

Describe a data structure that consumes \( O(|S|) \) space, and supports each operation in \( O(\log |S|) \) time amortized.

For students in the training camp: improve the operation bound to \( O(1) \) amortized!

Problem 4. Recall that our algorithm for the “dynamic array problem” (i.e., insertion only) ensures that if the current set \( S \) has \( n \) elements, the array has a length of at most \( 2n \). Modify the algorithm to ensure that our array has length at most \( 1.5n \). You will still need to process an insertion in \( O(1) \) amortized time.

Problem 5. Let \( S \) be a set of \( n \) key-value pairs of the form \((k, v)\), where \( k \) is the key and \( v \) is the value. Preprocess \( S \) into a data structure so that the following queries can be answered efficiently. Given a pair \((q_k, q_v)\), a query

- Returns nothing if \( S \) contains no pair with key \( q_k \);
- Otherwise, it returns the number of pairs \((k, v) \in S\) such that \( k = q_k \) and \( v \leq q_v \).

Define the frequency of a key \( k \) as the number of pairs in \( S \) with key \( k \). Define \( f \) as the maximum frequency of all keys. Your structure must use \( O(n) \) space, and answer a query in \( O(\log f) \) expected time. Furthermore, it must be possible to construct the structure \( O(n \log f) \) time.

For example, suppose that \( S = \{(75, 35), (123, 6), (65, 32), (75, 22), (9, 1), (9, 10), (65, 74), (9, 8), (93, 23)\} \). Then, given \((63, 33)\), the query returns nothing. Given \((65, 33)\), the query returns 1. Given \((65, 2)\), the query returns 0. In this example, \( f = 3 \).