Homework 3: Stacks and Queues

Due: Friday, February 14 at 5pm on Canvas

Concepts: linear data structures, stacks, queues, amortized analysis

1. (6 points) Explain how a queue can be implemented using two stacks. Then, prove that your stack-based queue has the property that any sequence of n operations (where an operation is either an `enqueue` or `dequeue`) takes a total of O(n) time resulting in amortized O(1) time for each of these operations. You may use any of the amortized analysis techniques that we discussed in class.

2. (6 points) A deque (a double-ended queue) is an abstract data type that generalizes a queue. It supports the following operations

   (a) `pop_back:` remove the element at the back of the deque.

   (b) `pop_front:` remove the element at the front of the deque.

   (c) `push_back:` add a given element to the back of the deque.

   (d) `push_front:` add a given element to the front of the deque.

   We can implement a deque using three stacks A, B, and C. Stack A will correspond to the front of the deque, stack B will correspond to the back of the deque, and stack C will be a helper stack, which we will use to ensure that stack A and B are balanced. Whenever we want to push an element x to the front of the deque, we push it onto stack A. Similarly, whenever we want to push an element to the back of the deque, we push it onto to stack B.

   If A is not empty, we can pop an element from the front of the deque by popping off stack A. However, if A is empty and B is non-empty, we first transfer half the elements off of B and push them onto A, which reverses the order. Second, we pop the remaining half of elements on B and add them to C, again reversing the order. Third, we pop the elements off of A and add them to B, putting them back into the original order. Last, we swap the labels of stacks A and C. Thus, the items on the new stack A are the front half of the deque from front to middle, and the items on stack B are the back half of the deque from back to middle. See below. Similarly, if B is not empty, we pop an element from the back of the deque by popping off of stack B. Otherwise, we use C to split the elements between the two stacks as before.

$$A = [\ ], B = [4,3,2,1], C = [\ ]$$
$$A = [3,4], B = [2,1], C = [\ ]$$
$$A = [3,4], B = [\ ], C = [1,2]$$
$$A = [\ ], B = [4,3], C = [1,2]$$
$$C = [\ ], B = [4,3], A = [1,2]$$

   Prove that any sequence of n operations (where an operation is either a pop_front, pop_back, push_front, or push_back) takes a total of O(n) time resulting in amortized O(1) time for each of these operations. You should use an amortized analysis technique different from what you used in problem 1.

3. (12 points) Construct a data structure that implements the following operations:

   (a) `enqueue`: Adds an element to the set.

   (b) `dequeue`: Removes and returns the oldest element in the set.

   (c) `find_min`: Returns the minimum value in the set.

   This is a queue with the added operation of finding the minimum element. Prove the correctness of your data structure and analyze the complexity using amortized analysis. You should use an amortized analysis technique different from what you used in problems 1 and 2. (**Hint:** You may want to use two data structures – one to keep track of all the elements and one to keep track of the minimum.)