This homework is due Wednesday, November 21, at the start of class. The questions are drawn from the material in the lectures and Chapter 17 of the text on amortized analysis.

The homework is worth a total of 100 points. When questions with several parts do not specify the points for each part, each part has equal weight.

Remember to (a) put your name at the top of every page, (b) start each problem on a new page, (c) write on just one side of a page and do not use scrap paper, (d) put your answers in the correct order, and (e) staple your pages together. If you can’t solve a problem, state this, and write only what you know to be correct. Neatness and conciseness count.

(1) **(Stack with Backup)** (10 points) Suppose you want to support a stack that has the operations Push, Pop, and Multipop as discussed in class, as well as the new operation

- **Backup(S)**, which writes a copy of the entire contents of stack $S$ to a file for archiving. (Backup does not alter $S$.)

Suppose that the size of the stack never exceeds $k$, and that Backup is called after every $k$ operations on the stack.

Show that under these conditions, Push, Pop, Multipop, and Backup all take $O(1)$ amortized time, independent of $k$. Use the accounting method for your analysis.

(2) **(Simulating a queue using stacks)** (30 points) Show how to implement the queue data structure by using two stacks, so that the amortized time for queue operations in the stack-based implementation matches their worst case time in a standard queue implementation. More specifically, show how to implement the operations

- **Put($x$, Q)**, which adds element $x$ to the rear of queue $Q$, and
- **Get(Q)**, which removes the element $x$ on the front of queue $Q$ and returns $x$,

so that both operations run in $O(1)$ amortized time. Use the potential function method for your analysis.

(3) **(bonus) (Stack with Multipush)** (10 points) Suppose you want to support a stack that has the operations Push, Pop, and Multipop as discussed in class, as well as the new operation

- **Multipush(A, k, S)**, which pushes all elements in the array $A[1:k]$ onto stack $S$.

This is equivalent to doing $\text{Push}(A[k], S)$, $\text{Push}(A[k-1], S)$, $\ldots$, $\text{Push}(A[1], S)$.

Can Push, Pop, Multipop, and Multipush all be supported in $O(1)$ amortized time per operation?

(Note: If you feel this can be achieved, give an amortized analysis demonstrating it. If not, give an argument showing it is impossible.)

(4) **(Constant amortized time DeleteMin)** (20 points) Show that, by an appropriate choice of a potential function, the standard implementation of the implicit heap used in heap sort takes $O(1)$ amortized time for a DeleteMin, and $O(\log n)$ amortized time for an Insert.

(Note: In your solution, (a) specify how you concretely measure the real time for these two operations, (b) specify your potential function for the heap, and (c) analyze the amortized time for both operations using the potential function method. Implicit heaps are described in Section 6.5 of the text.)
(5) **(Deleting the larger half)** (40 points) Design a data structure that supports the following two operations on a set $S$ of integers:

- **Insert**$(x, S)$, which inserts element $x$ into set $S$, and
- **DeleteLargerHalf**$(S)$, which deletes the largest $\lceil |S|/2 \rceil$ elements from $S$.

Show how to implement this data structure so both operations take $O(1)$ amortized time. Use the *accounting method* for your analysis.

Note that Problem (3) is a *bonus* question, and is *not* required.