

---

## Problem Set 4

*Reading:* Chapters 17, 21.1–21.3

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered in the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated.

Three-hole punch your paper on submissions.

You will often be called upon to “give an algorithm” to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct algorithms which are *which are described clearly*. Convolved and obtuse descriptions will receive low marks.

---

### Exercise 4-1. The Ski Rental Problem

A father decides to start taking his young daughter to go skiing once a week. The daughter may lose interest in the enterprise of skiing at any moment, so the  $k$ th week of skiing may be the last, for *any*  $k$ . Note that  $k$  is *unknown*.

The father now has to decide how to procure skis for his daughter for every weekly session (until she quits). One can *buy* skis at a one-time cost of  $B$  dollars, or *rent* skis at a weekly cost of  $R$  dollars. (Note that one can buy skis at any time—e.g., rent for two weeks, then buy.)

Give a 2-competitive algorithm for this problem—that is, give an online algorithm that incurs a total cost of at most twice the offline optimal (i.e., the optimal scheme if  $k$  is known).

---

### Problem 4-1. Queues as Stacks

Suppose we had code lying around that implemented a stack, and we now wanted to implement a queue. One way to do this is to use two stacks  $S_1$  and  $S_2$ . To insert into our queue, we push into

stack  $S_1$ . To remove from our queue we first check if  $S_2$  is empty, and if so, we “dump”  $S_1$  into  $S_2$  (that is, we pop each element from  $S_1$  and push it immediately onto  $S_2$ ). Then we pop from  $S_2$ .

For instance, if we execute  $\text{INSERT}(a)$ ,  $\text{INSERT}(b)$ ,  $\text{DELETE}()$ , the results are:

	$S_1 = [ ]$	$S_2 = [ ]$	
$\text{INSERT}(a)$	$S_1 = [a]$	$S_2 = [ ]$	
$\text{INSERT}(b)$	$S_1 = [b, a]$	$S_2 = [ ]$	
$\text{DELETE}()$	$S_1 = [ ]$	$S_2 = [a, b]$	“dump”
	$S_1 = [ ]$	$S_2 = [b]$	“pop” (returns $a$ )

Suppose each push and pop costs 1 unit of work, so that performing a dump when  $S_1$  has  $n$  elements costs  $2n$  units (since we do  $n$  pushes and  $n$  pops).

- Suppose that (starting from an empty queue) we do 3 insertions, then 2 removals, then 3 more insertions, and then 2 more removals. What is the total cost of these 10 operations, and how many elements are in each stack at the end?
- If a total of  $n$  insertions and  $n$  removals are done in some order, how large might the running time of one of the operations be (give an exact, non-asymptotic answer)? Give a sequence of operations that induces this behavior, and indicate which operation has the running time you specified.
- Suppose we perform an arbitrary sequence of insertions and removals, starting from an empty queue. What is the amortized cost of each operation? Give as tight (i.e., non-asymptotic) of an upper bound as you can. Use the accounting method to prove your answer. That is, charge  $\$x$  for insertion and  $\$y$  for deletion. What are  $x$  and  $y$ ? Prove your answer.
- Now we’ll analyze the structure using the potential method. For a queue  $Q$  implemented as stacks  $S_1$  and  $S_2$ , consider the potential function

$$\Phi(Q) = \text{number of elements in stack } S_1.$$

Use this potential function to analyze the amortized cost of insert and delete operations.

#### Problem 4-2. David Digs Donuts

Your TA David has two loves in life: (1) roaming around Massachusetts on his forest-green Cannondale R300 road bike, and (2) eating Boston Kreme donuts. One Sunday afternoon, he is biking along Main Street in Acton, and suddenly turns the corner onto Mass Ave. (Yes, that Mass Ave.) His growling stomach announces that it is time for a donut. Because Mass Ave has so many donut shops along it, David decides to find a shop somewhere along that street. He faces two obstacles in his quest to satisfy his hunger: first, he does not know whether the nearest donut shop is to his left or to his right (or how far away the nearest shop is); and second, when he goes riding his contact lenses dry out dramatically, blurring his vision, and he can’t see a donut shop until he is directly in front of it.

You may assume that all donut shops are at an integral distance (in feet) from the starting location.

- (a) Give an efficient (deterministic) algorithm for David to locate a donut shop on Mass Ave as quickly as possible. Your algorithm will be online in the sense that the location of the nearest donut shop is unknown until you actually find the shop. The algorithm should be  $O(1)$ -competitive: if the nearest donut shop is distance  $d$  away from David's starting point, the total distance that David has to bike before he gets his donut should be  $O(d)$ . (The optimal offline algorithm would require David to bike only distance  $d$ .)
- (b) Optimize the competitive ratio for your algorithm—that is, minimize the constant hidden by the  $O(\cdot)$  in the competitive ratio.
- (c) Suppose you flip a coin to decide whether to start moving to the left or to the right initially. Show that incorporating this step into your algorithm results in an improvement to the expected competitive ratio.