15-122 : Principles of Imperative Computation

Summer 1 2012

Assignment 3

(Theory Part)

Due: Wednesday, June 06, 2012 in class

Name: _____

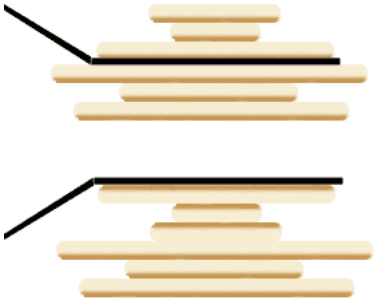
Andrew ID: _____

Recitation: _____

The written portion of this week's homework will give you some practice working with sorting algorithms, stacks and queue data structures and performing some asymptotic analysis tasks. You can either type up your solutions or write them *neatly* by hand, and you should submit your work in class on the due date just before lecture or recitation begins. Please remember to *staple* your written homework before submission.

Question	Points	Score
1	12	
2	8	
Total:	20	

1. **Pancake sort.** Pancake sort is a sorting algorithm that works as follows. Suppose you have a stack of pancakes that you would like to sort from smallest to largest such that largest pancake is at the bottom of the stack after sorting. However, the only operation allowed on this data structure is flipping (aka making a sub stack up side down). Only the sub stacks that *contain the top element* can be flipped and we assume flip is an $O(1)$ operation. As an example, here is what happens when the top 3 pancakes are flipped upside down. The first image shows the original pancake stack before flipping the top 3 pancakes. The second image shows the pancake stack after flipping the top 3 pancakes.



Here is a possible algorithm for sorting the pancakes.

```
struct pancake {
    int size;
    string flavor;
};
typedef struct pancake* pancake;

void pancakesort(pancake[] A, int n){
  int i=n;
  while (i>1) {
    int max = findmax(A,0,i);
   //findmax(A,0,i) finds the index of the max size pancake in A[0..i)
   flip(A,0,max); // flip will reverse the array A[0..max+1)
   flip(A,0,i-1); // flip will reverse the array A[0..i)
   i=i-1;
  }
}
```

(4) (a) What is the worst case asymptotic complexity of your algorithm using big O notation in terms of n where n is the number of pancakes? Explain your answer.

Solution: $T(n) = O($

(4) (b) Rewrite the above code and add all annotations (pre-conditions, loop invariants, assertions and post-conditions). You can use functions used in class lectures to write your annotations.

Solution:

(4) (c) Prove (by induction) that your above code correctly sorts the pancakes from smallest to largest.

Solution:

2. **Stacks and Queues.**

Consider the following interfaces for `queue` and `stack` that accept elements of the type `elem`:

```
queue queue_new();
bool queue_empty(queue Q);
void enqueue(elem e, queue Q);
elem dequeue(queue Q);

stack stack_new();
bool stack_empty(stack S);
void push(elem e, stack S);
elem pop(stack S);
```

(4) (a) Write a function `reverse(queue Q)` for reversing the order of a given queue (with `elem` types) using a stack. After the operations, a given queue must remain in the original order. Include proper annotations in your code. What is the worst case run-time complexity of this function if there are n elements in the queue?

Solution:

(4) (b) Write a function `is_equal(stack S1, stack S2)` that takes two stacks (of the same size with `elem` types) and returns true if both stacks contain the same elements and the elements are in the same order. That is, the stacks are identical. Otherwise, it must return false. You may assume that the function bool `is_equal(elem e1, elem e2)` returns true if e1 is "equal" to e2. After the operations, both stacks must remain in the original order and you are allowed to use additional stacks or queues (but no other data structures) as needed. What would be the worst case asymptotic complexity of your code if each stack contains n elements each?

Solution: