CSE 21 Winter 2017
Homework 1
Due: Tuesday, January 17 at 11:59pm

INSTRUCTIONS

Homework should be done in groups of one to three people. You are free to change group members at any time throughout the quarter. Problems should be solved together, not divided up between partners. A single representative of your group should submit your work through Gradescope. Submissions must be received by 11:59pm on the due date, and there are no exceptions to this rule.

You will be able to look at your scanned work before submitting it. Please ensure that your submission is legible (neatly written and not too faint) or your homework may not be graded.

Students should consult their textbook, class notes, lecture slides, instructors, TAs, and tutors when they need help with homework. Students should not look for answers to homework problems in other texts or sources, including the internet. You may ask questions about the homework in office hours, but not on Piazza.

Your assignments in this class will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should always explain how you arrived at your conclusions and justify your answers with mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

For questions that require pseudocode, you can follow the same format as the textbook, or you can write pseudocode in your own style, as long as you specify what your notation means. For example, are you using “=” to mean assignment or to check equality? You are welcome to use any algorithm from class as a subroutine in your pseudocode. For example, if you want to sort list A using InsertionSort, you can call InsertionSort(A) instead of writing out the pseudocode for InsertionSort.

REQUIRED READING Rosen Sections 3.1 and 5.5.

KEY CONCEPTS Sorting algorithms, including selection (min) sort, insertion sort, and bubble sort; loop invariants and correctness proofs; searching algorithms; counting comparisons; best and worst case.
Note: For this assignment, the word “comparison” refers only to comparisons involving list elements. For example, if $a_i$ and $a_j$ are list elements in a list of length $n$, the code if $a_i < a_j$ performs one comparison. Similarly, the code if $a_i < 5$ performs one comparison. However, we would say the code if $i < n$ performs no comparisons because it is not making a comparison involving a list element.

1. For this problem, consider the following sorting algorithms

**procedure** SortA($a_1, a_2, \ldots, a_n$: a list of real numbers with $n \geq 2$)

1. for $j := 2$ to $n$
2. $i := 1$
3. while $a_j > a_i$
4. $i := i + 1$
5. $m := a_j$
6. for $k := 0$ to $j - i - 1$
7. $a_{j-k} := a_{j-k-1}$
8. $a_i := m$

**procedure** SortB($a_1, a_2, \ldots, a_n$: a list of real numbers with $n \geq 2$)

1. for $j := 2$ to $n$
2. $i := j$
3. while ($i > 1$ AND $a_j < a_{i-1}$)
4. $i := i - 1$
5. $m := a_j$
6. for $k := 0$ to $j - i - 1$
7. $a_{j-k} := a_{j-k-1}$
8. $a_i := m$

(a) (2 points) Trace the behavior of SortA on the input list 3, 2, 5, 6, 4, 1 by filling in the following table. Each row corresponds to the completion of one iteration of the outermost loop. When listing the comparisons done in each iteration, say which two elements are being compared in each comparison (example: 3 vs. 5).

<table>
<thead>
<tr>
<th>After the...</th>
<th>What the list looks like</th>
<th>Comparisons done in this iteration</th>
<th>Total number of comparisons done so far</th>
</tr>
</thead>
<tbody>
<tr>
<td>0th iteration</td>
<td>3, 2, 5, 6, 4, 1</td>
<td>none</td>
<td>0</td>
</tr>
<tr>
<td>1st iteration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd iteration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(b) (2 points) Trace the behavior of SortB on the input list 3, 2, 5, 6, 4, 1 by filling in the following table. Each row corresponds to the completion of one iteration of the outermost loop. When listing the comparisons done in each iteration, say which two elements are being compared in each comparison (example: 3 vs. 5).

<table>
<thead>
<tr>
<th>After the...</th>
<th>What the list looks like</th>
<th>Comparisons done in this iteration</th>
<th>Total number of comparisons done so far</th>
</tr>
</thead>
<tbody>
<tr>
<td>0th iteration</td>
<td>3, 2, 5, 6, 4, 1</td>
<td>none</td>
<td>0</td>
</tr>
<tr>
<td>1st iteration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd iteration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) (3 points) The same loop invariant can be used to prove the correctness of both SortA and SortB. State, but do not prove, this invariant.

**Loop Invariant:** After the $t^{th}$ iteration of the outer loop...

(d) (3 points) Suppose your above loop invariant is true for all values of $t \geq 0$ (you do not need to include the proof). How can you show SortA is correct based on this loop invariant being true? How can you show SortB is correct based on this loop invariant being true?

2. Consider the following algorithm

**procedure** Loops($n$: a positive integer)

1. for $i := 1$ to $n$
2. for $j := i$ to $n$
3. for $k := 1$ to $j$
4. print $(i, j, k)$

(a) (3 points) Write what the algorithm prints when $n = 3$.
(b) (3 points) How many times does the print routine get called when $n = 3$?
(c) (4 points) Describe (in words) a rule to decide, if $(i_1, j_1, k_1)$ and $(i_2, j_2, k_2)$ have both been printed for some $n$, which triplet was printed first.

3. In this problem, we are given a sequence $a_1, a_2, \ldots, a_n$ of integers and we want to return a list of all terms in the sequence that are greater than the sum of all previous terms of the sequence. For example, on an input sequence of 1, 4, 6, 3, 2, 20, the output should be the list 1, 4, 6, 20. The following algorithm solves this problem.

**procedure** PartialSums($a_1, a_2, \ldots, a_n$: a sequence of integers with $n \geq 1$)

1. total := 0
2. Initialize an empty list $L$. 

3. for $i := 1$ to $n$
4. \hspace{1em} if $a_i > \text{total}$
5. \hspace{2em} Append $a_i$ to list $L$.
6. \hspace{2em} $\text{total} := \text{total} + a_i$
7. \hspace{1em} return $L$

(a) (6 points) Prove the following loop invariant by induction on the number of loop iterations:

**Loop Invariant:** After the $t^{th}$ iteration of the for loop, $\text{total} = a_1 + a_2 + \cdots + a_t$ and $L$ contains all elements from $a_1, a_2, \ldots, a_t$ that are greater than the sum of all previous terms of the sequence.

(b) (4 points) Use the loop invariant to prove that the algorithm is correct, i.e., that it returns a list of all terms in the sequence that are greater than the sum of all previous terms of the sequence.

4. On the gameshow *The Price is Right*, contestants guess the price of an item, and the winner is the person who guesses closest to the actual price without going over the actual price. For example, if the price of a refrigerator is $1529$ and the contestants guess $1000, 1530, 1200,$ and $1600$, then the person who guessed $1200$ will win the refrigerator. If all guesses are too high, nobody wins. The following algorithm takes as input a list of guesses and an actual price, and determines the guess that is closest to the actual price without going over. All inputs are positive values. The algorithm returns 0 if all guesses are too high and there is no winner.

**procedure** ClosestWithoutGoingOver($a_1, a_2, \ldots, a_n$: list of guesses, $p$: price, all positive)

1. $\text{best} := 0$
2. for $i := 1$ to $n$
3. \hspace{1em} if $(a_i \leq p \text{ AND } a_i > \text{best})$
4. \hspace{2em} $\text{best} := a_i$
5. return $\text{best}$

Use a loop invariant to prove that the ClosestWithoutGoingOver algorithm given above is correct, i.e., that it returns the guess in the list that is closest to the price $p$ without going over, and returns 0 if all guesses in the list are too high. Be sure to include all the parts of the proof:

(a) (3 points) State the loop invariant precisely.
(b) (4 points) Prove the loop invariant by induction on the number of loop iterations.
(c) (3 points) Use the loop invariant to prove that the algorithm is correct as defined above.

5. (a) (3 points) Design an algorithm that finds the index of the last even integer in a given list of integers or returns 0 if the list does not have any even integer: (Start your pseudocode like this:)

**procedure** LastEven($a_1, a_2, \ldots, a_n$: list of integers) ......

(b) (2 points) State the loop invariant precisely.
(c) (3 points) Prove the loop invariant by induction on the number of loop iterations.
(d) (2 points) Use the loop invariant to prove that the algorithm is correct as defined above.