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.

Homework solutions should be neatly written or typed and turned in through Gradescope by 11:59pm on the due date. No late homeworks will be accepted for any reason. 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. Only post about graded homework questions on Piazza if you suspect a typo in the assignment, or if you don’t understand what the question is asking you to do. Other questions are best addressed in office hours.

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, using 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.
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. This problem refers to the following two algorithms.

```
procedure SortA(a_1, a_2, \ldots, a_n: a list of real numbers with \( n \geq 1 \))
1. for i := 1 to \( n - 1 \)
2. \hspace{1em} item := a_i
3. \hspace{1em} location := i
4. for j := i + 1 to \( n \)
5. \hspace{2em} if a_j < item then
6. \hspace{3em} item := a_j
7. \hspace{3em} location := j
8. \hspace{2em} a_{location} := a_i
9. a_i := item
```  

```
procedure SortB(a_1, a_2, \ldots, a_n: a list of real numbers with \( n \geq 1 \))
1. for k := 1 to \( n - 1 \)
2. \hspace{1em} i := n - k + 1
3. \hspace{1em} item := a_i
4. \hspace{1em} location := i
5. for j := 1 to \( i - 1 \)
6. \hspace{2em} if a_j > item then
7. \hspace{3em} item := a_j
8. \hspace{3em} location := j
9. \hspace{2em} a_{location} := a_i
10. a_i := item
```  

(a) (4 points) State, but do not prove, a loop invariant that could be used to show SortA correctly solves the sorting problem.

(b) (4 points) State, but do not prove, a loop invariant that could be used to show SortB correctly solves the sorting problem.

2. This problem refers to the two algorithms SortA and SortB from problem 1 above.

(a) (2 points) How many comparisons does SortA perform on the input list 3, 2, 5, 1, 4? Make a list of all comparisons the algorithm performs. This list should include which elements are being compared in each comparison.

(b) (2 points) Find a different input list containing the numbers 1, 2, 3, 4, and 5 for which SortA does fewer comparisons than it does on the input list 3, 2, 5, 1, 4, or say why it is impossible to find such a list.
(c) (2 points) How many comparisons does SortB perform on the input list 3, 2, 5, 1, 4? Make a list of all comparisons the algorithm performs. This list should include which elements are being compared in each comparison.

(d) (2 points) Find a different input list containing the numbers 1, 2, 3, 4, and 5 for which SortB does fewer comparisons than it does on the input list 3, 2, 5, 1, 4, or say why it is impossible to find such a list.

(e) (2 points) For the input list n, n – 1, n – 2, . . ., 2, 1, how many comparisons does each algorithm do, in terms of n? Simplify your answer.

3. This problem refers to the following two sorting algorithms.

```
procedure SortC(a_1, a_2, . . . , a_n: a list of real numbers with n ≥ 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 SortD(a_1, a_2, . . . , a_n: a list of real numbers with n ≥ 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) How many comparisons does SortC perform on the input list 3, 1, 6, 5, 2, 4? Make a list of all comparisons the algorithm performs. This list should include which elements are being compared in each comparison.

(b) (2 points) How many comparisons does SortD perform on the input list 3, 1, 6, 5, 2, 4? Make a list of all comparisons the algorithm performs. This list should include which elements are being compared in each comparison.

(c) (2 points) Find an input list containing the numbers 1, 2, 3, 4, 5, and 6 for which SortC does the fewest possible number of comparisons (i.e. a best-case input).

(d) (2 points) Find an input list containing the numbers 1, 2, 3, 4, 5, and 6 for which SortD does the fewest possible number of comparisons (i.e. a best-case input).

(e) (2 points) Find an input list containing the numbers 1, 2, 3, 4, 5, and 6 for which SortC does the greatest possible number of comparisons (i.e. a worst-case input).

(f) (2 points) Find an input list containing the numbers 1, 2, 3, 4, 5, and 6 for which SortD does the greatest possible number of comparisons (i.e. a worst-case input).

4. 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. if $a_i > total$
5. Append $a_i$ to list $L$.
6. $total := total + a_i$
7. return $L$

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

**Loop Invariant:** After the $k$th iteration of the for loop, $total = a_1 + a_2 + \cdots + a_k$ and $L$ contains all elements from $a_1, a_2, \ldots, a_k$ 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.

5. This problem refers to the following search algorithm.

procedure ReverseSearch($x$: integer, $a_1, a_2, \ldots, a_n$: distinct integers)

1. $i := n$
2. while $(i \geq 1$ and $x \neq a_i)$
3. $i := i - 1$
4. return $i$

Use a loop invariant to prove that the ReverseSearch algorithm given above is correct, i.e., that it returns the location of the target value $x$ in the list, and returns 0 if the target is not present in the list. 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.