INSTRUCTIONS

REQUIRED READING Rosen Sections 3.1 and 5.5.

Homework questions will be similar to your future exam questions. The homework will not be graded, however, you are highly recommended to practice using these questions for better preparations of the exams.

KEY CONCEPTS Analyzing algorithms, Correctness proofs for iterative algorithms with Loop Invariants, Time analysis for algorithms (number of comparisons), Time complexity (Big O and its relatives).

1. Prove the following formula using induction for all \( n \geq 0 \). Be clear about what your base case, your induction hypothesis, and inductive step are (label them).

\[
\sum_{i=0}^{n} 2^i = 2^{i+1} - 1
\]

2. Suppose you meet someone that is supposedly very generous. They have unlimited money and would like to offer you some under two conditions: 1) they keep their money in packages of $4 and $7 only for convenience 2) while they will always offer you at least $20, they will consider it very rude if you do not take the full amount they offer and take the money back as a result. Are they really generous i.e. is it always possible to take the amount they offer using only their packages of $4 and $7? Prove your answer.
3. This problem refers to the following sorting algorithm.

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

1. for $i := 1$ to $n - 1$
2. \hspace{1em} for $j := 1$ to $n - i$
3. \hspace{2em} if $a_j > a_{j+1}$ then
4. \hspace{3em} interchange $a_j$ and $a_{j+1}$

State and prove the loop invariant for BubbleSort.

4. This problem refers to the following sorting algorithm.

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

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

State and prove the loop invariant for InsertionSort.

5. (a) Change the pseudocode of MinSort to sort only the sub-sequence of elements at positions (starting with 1) of the form $3k + 1$ for $k \in \mathbb{N}$. For example, $[11, 3, 3, 8, 5, 9, 0] \mapsto [0, 3, 3, 8, 5, 9, 11]$.

(b) What is the time complexity in big-O notation of the resulting algorithm?

6. Give the number of comparisons that will be performed by each sorting algorithm if the input array of length $n$ happens to be of the form $[1, 2, \ldots, n-3, n-2, n, n-1]$ (i.e., sorted except for the last two elements). On the real exam, you would be given pseudocode for the algorithms, though it is a very good idea to be comfortable with how the algorithms work to save time on the exam. For now, you can refer to the slides for pseudocode.

(a) MinSort
(b) BubbleSort
(c) InsertionSort

7. Some more properties of MinSort:

(a) While we often talk about the number of comparisons an algorithm make, it is useful to know the number of assignments ($\leftarrow$ or $\Leftarrow$) made. What is the worst-case number of assignments MinSort will make? State and justify the general form of an input leading to this.

(b) Can we improve the running time in certain cases for MinSort? Explain how and calculate the worst-case time this might add to the algorithm.

(c) As stated in lecture, lower (and upper bounds) are usually not required to be “tight” e.g. for any non-trivial algorithm a lower bound is 1. Perform the lower bound analysis on Slide 91 using $n/3$ and come up with a new lower bound for MinSort. Run your lower bound on the input $[5, 3, 8, 1, 4, 9]$ and show your calculations.
8. This problem refers to the following two algorithms.

**procedure** SortA\((a_1, a_2, \ldots, a_n): \text{a list of real numbers with } n \geq 1\)

1. \textbf{for} \(i := 1\) to \(n - 1\)
2. \hspace{1em} \textit{item} := \(a_i\)
3. \hspace{1em} \textit{location} := \(i\)
4. \textbf{for} \(j := i + 1\) to \(n\)
5. \hspace{1em} \textbf{if} \(a_j < \textit{item}\) \textbf{then}
6. \hspace{2em} \textit{item} := \(a_j\)
7. \hspace{2em} \textit{location} := \(j\)
8. \hspace{1em} \(a_{\textit{location}} := a_i\)
9. \(a_i := \textit{item}\)

**procedure** SortB\((a_1, a_2, \ldots, a_n): \text{a list of real numbers with } n \geq 1\)

1. \textbf{for} \(k := 1\) to \(n - 1\)
2. \hspace{1em} \(i := n - k + 1\)
3. \hspace{1em} \textit{item} := \(a_i\)
4. \hspace{1em} \textit{location} := \(i\)
5. \textbf{for} \(j := 1\) to \(i - 1\)
6. \hspace{1em} \textbf{if} \(a_j > \textit{item}\) \textbf{then}
7. \hspace{2em} \textit{item} := \(a_j\)
8. \hspace{2em} \textit{location} := \(j\)
9. \hspace{1em} \(a_{\textit{location}} := a_i\)
10. \(a_i := \textit{item}\)

What is different about the two algorithms and what is the same? Justify all your answers by referring specifically to the pseudocode. You can illustrate your answers with examples.

9. The next problem refers to the three sorting algorithms displayed on page 4.

Suppose we start with the list of natural numbers 2, 7, 5, 6, 2, 4 and run one of these algorithms to sort it. We stop the program after 3 iterations of the outer loop, and we see that the list now looks like 2, 5, 6, 7, 2, 4.

(a) Which algorithm(s) could have been used to sort the list?
(b) For each algorithm given in part (a), how many comparisons between list elements were performed by the algorithm at the time we stopped the program?

Justify all your answers by referring specifically to the pseudocode.
procedure InsertionSortA($a_1, a_2, \ldots, a_n$: 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 InsertionSortB($a_1, a_2, \ldots, a_n$: a list of real numbers with $n \geq 2$)

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

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

1. for $j := 2$ to $n$
2. $i := 1$
3. $b := j$
4. while $i < b$
5. $c := \lfloor (i + b)/2 \rfloor$
6. if $a_j > a_c$ then $i := c + 1$
7. else $b := c$
8. $m := a_j$
9. for $k := 0$ to $j - i - 1$
10. $a_{j-k} := a_{j-k-1}$
11. $a_i := m$
10. When finding the maximum and minimum values in a list, as with sorting a list, we measure the cost as the number of comparisons between list elements.

(a) Suppose you want to find the maximum value in an arbitrary list of size $n$ (with integer $n \geq 1$), doing as few comparisons as possible. What is the minimum number of comparisons that you must perform to guarantee that you find the maximum value in any list of size $n$? Explain why a smaller number of comparisons will not be enough to ensure that the maximum is found.

(b) Similarly, suppose you want to find the minimum value in an arbitrary list of size $n$, doing as few comparisons as possible. What is the minimum number of comparisons that you must perform to guarantee that you find the minimum value in any list of size $n$?

11. This problem refers to the following search algorithm.

\begin{verbatim}
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
\end{verbatim}

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.

12. In the following problem, we are given a list $A = a_1, \ldots, a_n$ of salaries of employees at our firm and two integers $L$ and $H$ with $0 \leq L \leq H$. We wish to compute the average salary of employees who earn between $L$ and $H$ (inclusive), and the number of such employees. If there are no employees in the range, we say that 0 is the average salary. This is an iterative algorithm which takes as input $A, L,$ and $H$ and returns an ordered pair $(avg, N)$ where $avg$ is the average salary of employees in the range, and $N$ is the number of employees in the range.

\begin{verbatim}
procedure AverageInRange(A : list of n integers, L, H : integers with 0 \leq L \leq H)
1. sum := 0
2. N := 0
3. for i := 1 to n
4. if L \leq a_i \leq H then
5. sum := sum + a_i
6. N +=
7. if N = 0 then
8. return (0, 0)
9. return (sum/N, N)
\end{verbatim}

(a) State and prove a loop invariant for AverageInRange.

(b) Describe the running time of this algorithm in $\Theta$ notation, assuming that comparisons and arithmetic operations take constant time. Justify your answer.
13. Suppose we are adding two \( n \)-digit integers, using the usual algorithm learned in grade school. The primary operation here is the number of single-digit additions that must be performed. For example, to add 48 plus 34, we would do three single-digit additions:

1. In the ones place, add \( 8 + 4 = 12 \).
2. In the tens place, add \( 4 + 3 = 7 \).
3. In the tens place, add \( 7 + 1 = 8 \).

(a) If \( n = 5 \), give an example of two \( n \)-digit numbers that would be a best-case input to the addition algorithm, in the sense that they would cause the fewest single-digit additions possible.

(b) In the best case, how many single-digit additions does this algorithm make when adding two \( n \)-digit numbers?

(c) In the best case, when adding two \( n \)-digit numbers, describe the number of single-digit additions in \( \Theta \) notation.

(d) If \( n = 5 \), give an example of two \( n \)-digit numbers that would be a worst-case input to the addition algorithm, in the sense that they would cause the most single-digit additions possible.

(e) In the worst case, how many single-digit additions does this algorithm make when adding two \( n \)-digit numbers?

(f) In the worst case, when adding two \( n \)-digit numbers, describe the number of single-digit additions in \( \Theta \) notation.

14. For each part, answer True or False, and give a short explanation for your answer. All logarithms are base 2.

(a) \( 2n^2 + 3n \in O(n^2) \).

(b) \( n \log n \in O(n^2) \).

(c) \( 24n^4 + 1200n^2 \in O(n^3) \).

(d) \( 17n^3 + 18n^2 + 5 \in \omega(n^4) \).

(e) \( 17n^3 + 18n^2 + 5 \in \omega(n^3) \).

(f) \( 17n^3 + 18n^2 + 5 \in \omega(n^2) \).

(g) \( \sqrt{n^3} \in O(n^2) \).

(h) \( \log(n) \in \Omega(\log(\log(n))) \).

(i) \( \log((n!)^2) \in O(n^2) \).

(j) \( 1^2 + 2^2 + 3^2 + \ldots + n^2 \in \Omega(n^3) \).

(k) \( n^2 + \log(n^2) + \log(10^{10} \cdot n^{10}) \in \Theta(n^2) \).

(l) \( \sqrt{2^n} \in O(1.5^n) \).

(m) \( 2n^3 \in o(n^3) \).

(n) \( 2n^3 \in O(n^3) \).