CSC 225: Assignment 1 Part B, Written Exercises

Due at the beginning of class on Thursday September 28

Instructions for all assignments:

1. Draw boxes for your marks on the top of the first page of your submission. Place a 0 in the corresponding box for any questions you omit. For this assignment:

Question	1	2	3	4	5	6
Marks	0	0	0	0	0	0

2. Questions should be **in order**. Show your work unless otherwise stated.
3. Please put your name (last name underlined) and student number on all submissions.

Learning objectives

Recurrences are used throughout CSC 225 as a tool for analyzing time and space usage of algorithms. Induction is used for proofs of correctness and is also a mathematical foundation for recursive algorithms. This assignment also reviews basic linked list functionality.

1. [15] Consider the following recurrence relation defined only for $n = 2^k$ for integers $k \geq 1$: $T(2) = 5$, and for $n \geq 4$, $T(n) = 2n + T(n/2)$.

   Three students were working together in a study group and came up with three different answers for this recurrence:

   (a) $T(n) = 2n\log_2(n) - 3$

   (b) $T(n) = n\log_2(n) + n + 1$

   (c) $T(n) = 4n - 3$

   Determine which solution (if any) is correct by trying to prove that each one is correct by induction. If you cannot complete the induction proof, explain what goes wrong.

2. [15] The recurrence you should solve for this question is only defined when $n = 2^k$ for some integer k where $k \geq 3$. The base case is $T(8) = 7$. When $k \geq 4$ the recurrence is given by $T(n) = n + 2T(n/2)$. Solve this recurrence relation using repeated substitution. To show your work when using repeated substitution you should number your steps so that you have:
   Step 0: The original recurrence for T(n).
   Hint: The algebra is easier if you use
   Step 0: $T(n) = T(2^k) = 2^k + 2\,T(2^{k-1})$.
   Step 1: The formula for T(n) after one substitution into the RHS.
   Step 2: The formula for T(n) after two substitutions into the RHS.

Then determine the general pattern for Step i:
Step i: The formula for T(n) after i substitutions into the RHS.
Next determine at which step i the base case appears on the RHS of the formula, say at some step f. Then use your template for Step f to get a closed formula for the recurrence relation.

3. A student counted up the number of statements executed in a program developed for sorting *n* integers and came up with a recurrence of the form:
$T(n) = a * n + b + T(n-1)$, and $T(1) = s$ where a, b and s are constants.

   (a) [10] Use the method of repeated substitution to solve this recurrence.
   Your solution should be a closed formula: sums or expressions with T in the formula are not permitted.

   (b) [10] Prove your answer to (a) is correct by induction.

4. The purpose of this question is to show that as long as a, b and s from question 3 are constants satisfying $a > 0$, and $b, s \geq 0$, the final answer for the recurrence always has the same Big Oh time complexity. This rationalizes making the recurrences we solve for time complexities as simple as possible when doing algorithm analysis. For example, $T(n) = n + T(n-1)$, and $T(1) = 1$ is a simpler recurrence giving the same time complexity ($a = 1$, $b = 0$, and $s = 1$).

   (a) [10] Assume that $a > 0$, and $b, s \geq 0$. Find constants n_0 and c such that for all $n \geq n_0$, your answer from question **3** is at most $c\ n^2$. Hint: you will have to express your constant c as some function of the constants a, b and s.

   (b) [5] Assume that $a > 0$, and $b, s \geq 0$. Find constants n_0 and d such that for all $n \geq n_0$, your answer from question **3** is at least $d\ n^2$ for some constant $d > 0$.

5. Consider the *ReadRear* Java method given on the last page of this assignment.

   (a) [10] Draw pictures that show the data structure each time a checkpoint is reached for the problems of sizes one, two, three and four specified as input as follows (same format as for assignment #1 Part B: the first integer *n_digit* is the number of items in the list and this is followed by *n_digit* integers)
   1 9
   2 7 8
   3 6 5 4
   4 1 3 0 2

   and indicate for each instance, the number of times the marked statement is executed while creating the linked list.

   (b) [5] A *loop invariant* is a statement about a loop used to prove properties about a loop. Fill in a correct function for $f(k)$ in the following loop invariant and then prove that your loop invariant is correct by induction.

Loop invariant: At checkpoint #2 of the *kth* iteration of the while loop, the *current* pointer will be pointing to cell number $f(k)$ on the linked list where the cells are numbered starting with cell one.

(c) [5] Set up a recurrence which counts the number of times that the statement with the comment **// Statement to count.** is executed for a given value of n and justify your formula.

(d) [5] Solve your recurrence from (c) to get a closed formula.

6. [10] Prove by induction that your closed formula from 5(c) is the number of times that the given statement is executed for a problem of size n. Note: if your proof is simply a proof that you have a correct closed formula for your recurrence from 5(c), then you will not get any marks for this question. Your proof must refer back to the program in order to be correct. The lines of the code are numbered in order to make it easier for you to refer to them in your proof.

The Java code for this question:

```
 1  public void readRear(Scanner in)
 2  {     ListNode tmp, current; int data; int i;
 3        n= readInteger(in);
 4        start=null; rear=null;
 5        for (i=0; i < n; i++)
 6        {
 7              data= readInteger(in);
 8              tmp= new ListNode(data, null);
 9              if (i==0) { start=tmp; }
10               else
11               {
12                    // Checkpoint 1.
13                    current= start;
14                    while (current.next != null)
15                    {
15                          current= current.next;  // Statement to count.
16                          // Checkpoint 2.
18                    }
19                    current.next= tmp;
20               }
21               rear= tmp;
22        }
23        // Checkpoint 3.
24  }
```