COMP 250 - Homework #3 Solution Mathieu Blanchette

Question 1 (45 points)

```
   // Pseudocode of recursive method
    // Algorithm mergeSort(A, start, stop)
    // if (start<stop) {
    //     mid = (start+stop)/2
    //     mergeSort(A, start, mid)
    //     mergeSort(A, mid+1, stop)
    //     merge(A, start, mid, stop)
    // }

    static void mergeSort(int A[]) {

        ProgramFrame current=new ProgramFrame(A, 0,A.length-1);
        callStack=new Stack<ProgramFrame>();
        callStack.push(current);

        while (!callStack.empty()) {

          System.out.println(callStack.peek());

            if (callStack.peek().PC==4) {
                callStack.pop();
                callStack.peek().PC++;
            }

            // Base case, start = stop: Simply return, i.e. pop frame from stack and
increase PC by one.
            if (callStack.peek().start==callStack.peek().stop) {
                callStack.pop();
                callStack.peek().PC++;
            }

            callStack.peek().mid=(callStack.peek().start+callStack.peek().stop)/2;
            if (callStack.peek().PC==1) {
                current=new ProgramFrame(callStack.peek().A,
callStack.peek().start,callStack.peek().mid);
                callStack.push(current);
                continue;
            }

            if (callStack.peek().PC==2) {
                current=new ProgramFrame(callStack.peek().A,
callStack.peek().mid+1,callStack.peek().stop);
                callStack.push(current);
                continue;
            }

            if (callStack.peek().PC==3) {
                callStack.peek().mid=(callStack.peek().start+callStack.peek().stop)/2;
                merge(callStack.peek().A , callStack.peek().start,
callStack.peek().mid,callStack.peek().stop);
                callStack.pop();
                if (!callStack.empty()) callStack.peek().PC++;
                continue;
            }
        }
    }
```

Question 2. (15 points)

Consider the following recurrence:

$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 * T(n-1) + n & \text{if } n > 1 \end{cases}$

a) (10 points) Obtain an explicit formula for the following recurrence using one of the techniques seen in class. In the process of doing so, you will possibly come across the summation $\Sigma_{i=1..n} (i * 2^i)$, which can be simplified as $2 * (1 + 2^n * (n-1))$.

$T(n) = 2 * T(n-1) + n$
$= 2 * (2 * T(n-2) + n-1) + n = 4\ T(n-2) + 3\ n - 2$
$= 4\ (2*T(n-3) + n-2) + 3\ n - 2 = 8\ T(n-3) + 7n - 2 - 8$
$= 8\ (2* T(n-4) + n - 3) + 7n - 2 - 8 = 16\ T(n-4) + 15n - 2 - 8 - 24$
…
(after k substitutions)
$= 2^k\ T(n-k) + (2^k - 1)\ n - (\Sigma_{i=1...k-1}\ i * 2^i)$
$= 2^k\ T(n-k) + (2^k - 1)\ n - 2 * (1 + 2^{k-1} * (k-2))$

When k = n-1, the recursion stops, and we get
$T(n) = 2^{n-1}\ T(1) + (2^{n-1} - 1)\ n - 2 * (1 + 2^{n-2} * (n-3))$
$= 2^{n-1} + (2^{n-1} - 1)\ n - 2 * (1 + 2^{n-2} * (n-3))$
$= 2^{n+1} - n - 2$

Informal check: Based on the recursive formula, we get:
$T(1) = 1$
$T(2) = 2* T(1) + 2 = 4$
$T(3) = 2* T(2) + 3 = 11$
$T(4) = 2 * T(3) + 4 = 26$
$T(5) = 2* T(4) + 5 = 57$

Based on our explicit formula, we get:
$T(1) = 2^0 + (2^0 - 1)\ 1 - 2 * (1 + 2^{-1} * (1-3)) = 1 - 0 - 2 * (1 - 1) = 1$
$T(2) = 2 + (2 - 1) * 2 - 2\ (1 + 1*-1) = 4$
$T(3) = 4 + (4-1)*3 - 2\ (1 + 2* 0) = 11$
$T(4) = 8 + (8-1)*4 - 2\ (1 + 4*1) = 26$
$T(5) = 16 + (16-1)*5 - 2 * (1 + 8*2) = 57$
So everything looks good…

b) (5 points) Prove your result using induction.

Let P(n) be the prosition: $T(n) = 2^{n+1} - n - 2$.
We show that P(n) holds for any value of n >= 1 .

Base case: If n=1, T(1) = 1 (by definition), and $2^{n+1} - n - 2 = 2^{1+1} - 1 - 2 = 1$

Induction step: Assume that P(k) is true for some value k>=1, i.e.
$T(k) = 2^{k+1} - k - 2$.
We need to show that this implies that P(k+1) is true, i.e.
$T(k+1) = 2^{k+2} - (k+1) - 2$.

$T(k+1) = 2\ T(k) + k + 1$ (by definition of T()).
$= 2\ (2^{k+1} - k - 2) + k + 1$
$= 2^{k+2} - k - 3$
$= 2^{k+2} - (k+1) - 2$

Thus, P(k) implies P(k+1), so, by the principle of induction, P(n) will be true for any value of n >= 1.

Question 3 (15 points)

Consider the following "magic" trick. You have a deck of *n* cards, labeled 1,2,...,*n* (but not necessarily in that order).
You then repeat the following process until no cards are left: (i) Show to the public the card on the top of the deck, and remove it from the deck, and (ii) Take the next card from the top of the deck and place it at the bottom of the deck, without showing it. Your goal is to have previously ordered the cards in the deck so that the cards shown to the public are in increasing order: 1, 2, ..., *n*. For example, if *n*=5, then starting from the arrangement 1,5,2,4,3 would work: 1,5,2,4,3 -> 2,4,3,5 -> 3,5,4 -> 4,5 -> 5

Question: Write an algorithm that prints the appropriate initial ordering for any given number *n* of cards.
Algorithm orderCards(n)
Input: An integer n
Output: Prints the correct card ordering.

Here's the simplest solution, which proceed backward to reverse the series of operations required to produce the right output of cards:

```
LinkedList deck <- new LinkedList
For i = n downto 1 do {
        Deck.rotateRight()      // brings to the front the card at the back of the deck
        Deck.addFirst(i)
}
```

Here, the rotateRight method simply removes the last element from the list and places it in front on the list.

Question 4. (15 points)

We have seen it class the precise meaning of the notation $f(n) \in O(g(n))$, essentially meaning that $f(n)$ grows at most as fast as a constant times $g(n)$. A similar notation can be used to say that $f(n)$ grows at least as fast a constant times $g(n)$, using the Omega notation:
$f(n) \in \Omega(g(n))$ if and only if $g(n) \in O(f(n))$.
Finally, we can express the fact that $f(n)$ grows exactly as fast as a constant times $g(n)$ using the Theta notation:
$f(n) \in \Theta(g(n))$ if and only if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.

Question: Prove that $\log(n!) \in \Theta(n \log(n))$

First, we show that $\log(n!) \in O(n \log n)$. For this, we need to find c and N such that for all $n \geq N$, $\log(n!) \leq c\, n \log(n)$. This is the easy part, as c=1 and N = 1 will work:

$\log(n!) = \log (n * (n-1) * (n-2) ...* 2 * 1)$
$= \log(n) + \log (n-1) + \log(n-2) + ... + \log(2) + \log(1)$
$\leq \log(n) + \log(n) + \log(n) + ... + \log(n) + \log(n)$
$= n \log(n)$
$= c\ n \log(n)$

Thus, $\log(n!) \in O(n \log n)$.

Now, to prove that $n \log n \in O(\log(n!))$, we need to find c and N such that for all $n \geq N$, $n \log n \leq c \log(n!)$. This time, it will be easier to start with the right side of the inequality. Let's not pick c and N yet and see what we'd need.

$\log(n!) = \underbrace{\log(n) + \log (n-1) + \log(n-2) + ... + \log(n/2 +1)}_{\geq\ n/2 \log(n/2)} + \underbrace{\log(n/2) + ...+ \log(2) +\log(1)}_{n/2 \log(1)}$

$= n/2 \log(n/2)$
$= n/2 (\log(n) - 1)$
$\geq n/2 (\log(n) - ½ \log(n))$ if $n \geq 4$
$= 1/4\ n \log(n)$

Thus, for N=4, c = 4, we get that $n \log(n) \leq 4\ n \log(n)$ for all $n \geq N$