Data Structures - Assignment no. 1, November 10, 2008

Remarks:

- **Write both your name and your ID number very clearly on the top of the exercise. Write your exercises in pen, or in clearly visible pencil. Please write *very* clearly.**
- **Recall that 80% of the theoretical exercises must be submitted. The exercises can and must be worked on and submitted alone.**
- **Give correctness and complexity proofs for every algorithm you write.**
- **For every question where you are required to write pseudo-code, also explain your solution in words.**

1. Which of the following statements are true and which are false? Only give an answer, you do not have to explain:

   (a) $40 \log n = O(8 \log n)$.
   (b) $2^n/300 = O(n)$.
   (c) $40n + 6 = O(2^n - 50)$.
   (d) $40n + 6 = O(\log n + 17)$.

2. A *deque* (double-ended queue) is an ADT (Abstract Data Type) that supports the following operations:

   (a) Push(x,D): Inserts x at the head of the deque.
   (b) Pop(D): Removes the element at the head of the deque and returns it.
   (c) Inject(x,D): Inserts x at the tail of the deque.
   (d) Eject(D): Removes the element at the tail of the deque and returns it.
   (e) Size(D): Returns the number of elements in the deque.
   (f) Empty?(D): Returns true if and only if the deque is empty
   (g) Make-deque(): Generates an empty deque

   Give a description, not in pseudo-code, of an implementation of this ADT that uses an array of fixed size N. Define precisely the representation of your data structure. Then describe in pseudo-code the implementations of $Push(x, D)$, $Size(D)$, and $Eject(D)$. You can assume that we never try to $Pop(D)$ or $Eject(D)$ when D is an empty deque, and that the size of the deque never exceeds N. All operations should take $O(1)$ time. Note that in order to avoid the situation where the deque is almost empty but you can't insert new elements, you'd probably have the use kind of "circular array". Note that the size of the deque is bounded by N, so you don't have to use the "doubling" method.

3. In the lecture you learned the "doubling" method that allows to implement a stack using an array without placing a limit on the size of the stack, such that the amortized complexity of each operation is $O(1)$. The method is that every time the array gets full, a new array is allocated whose size is twice the size of the old array, and the old array is copied to the new array.

   (a) Suppose we change the implementation so that the size of the new array is 4/3 times the size of the old array. What is the time complexity of a sequence of m operations on the worst-case? Prove[1] your answer.

   (b) Suppose we change the implementation so that the size of the new array is 10 **plus** the size of the old array. What is the time complexity of a sequence of m operations on the worst-case? Prove your answer.

4. (a) Implement, in pseudo-code, a trenary (base-3) counter: You are given an infinite array such that each cell can only hold the digits 0,1,2, and you want the counter to support the operation *increment()* that increases the value of the counter by 1.

   (b) Suppose you start from a counter initialized to 0 and you perform m *increment*() operations. What is their total cost? What is the amortized time complexity of increment? Prove your answer.

5. Let L be a singly-linked list that consists of n elements. Each element contains a pointer "next". Someone possibly made a mistake, and the final element of the list, instead of having a "null" pointer, might be pointing to an element inside the list. What does the following pseudo-code do? What is its worst-case time complexity?

   INPUT: x, which is a pointer to the first element of the list
   $y \leftarrow x.next$
   IF ($y == NULL$) THEN RETURN "true"
   WHILE ($y \neq x$)
   $y \leftarrow y.next$
   IF ($y == NULL$) THEN RETURN "true"
   $y \leftarrow y.next$
   IF ($y == NULL$) THEN RETURN "true"
   $x \leftarrow x.next$
   END WHILE
   RETURN "false"

[1] Here and in the future – we don't care which method you use to prove the claim, as long as it is correct! Specifically, you can use a bank, the potential method, or any other method you like.