CS 218, Fall 2016  Homework 2  
Posted: October 5th, 2016  

- The solution of this assignment has to be typed (LaTeX works great)
- Write on the first page your full name with upper-case LAST name, assignment number, and student ID
- You are expected to work on this assignment on your own. Include the following on first page “I certify that this submission represents my own original work” (date, signature) and sign it
- Anything that you submit that comes from “external sources” (a friend, a web page, a book, etc.) must be acknowledged and will be graded accordingly.
- Written assignments have to be submitted on paper at the beginning of the class on the due date. No late assignment will be accepted
- Use pseudocode/python (or English) to describe your algorithms; in addition to pseudocode/python, the algorithm has to be explained in English
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in class, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your algorithms

Problem 1. (30 points)  
Using the Master Theorem, give an asymptotic tight bound for $T(n)$ defined by the following recurrence relation

$$T(n) = \begin{cases} 
2 & n = 2 \\
4T \left(\sqrt{n}\right) + \log^2 n & n > 2 
\end{cases}$$

You will need to apply an appropriate substitution to apply the Theorem.

Problem 2. (30 points)  
We have seen in class that the procedure MERGE in the Mergesort algorithm takes two sorted arrays of size $n$ and produces one fully sorted array of size $2n$ in $O(n)$ time. Use the decision tree method to prove a $2n - o(n)$ lower bound\(^1\) for the problem of merging two sorted arrays, each containing $n$ items.

\(^1\)The little-oh notation is used here to denote an upper bound that is not asymptotically tight. Formally, we say that $f(n) \in o(g(n))$ if for any positive constant $c$ we can find a constant $n_0$ such that $o \leq f(n) < cg(n)$ for all $n \geq n_0$.  

---

\(\text{\textcopyright 2016 by the Department of Computer Science, University of California, Santa Barbara. All rights reserved.} \)
Problem 3. (40 points)

Show how to implement a queue using two stacks $S_1$ and $S_2$ so that the amortized cost of each operation on the queue is $O(1)$.

1. Give the pseudocode for the Enqueue($x$) operation and the Dequeue() operation (you can omit error checking for underflow and overflow of the stacks).

2. Use the accounting method to charge each operation a constant amortized cost and prove that a sequence of $n$ Enqueue and Dequeue cost $O(n)$ time overall.