Problem Set 3
IE 496 – Computational Methods in Optimization
Dr. Ralphs

1. Implement the API of the Python list class, but with a linked list as the underlying data structure. You will need two classes: one for a "node" in the linked list and the other for the linked list itself. Develop a rigorous set of tests for your linked list class and do an empirical comparison to Python's native list class. Generate graphs comparing running times for various operations, such as using the list as a stack.

2. Consider the following algorithm for multiplying 2×2 matrices. To compute the product

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}, \tag{1}$$

   we first compute the following products:

$$\begin{aligned} m_1 &= (a_{12} - a_{22})(b_{21} + b_{22}) && (2)\\ m_2 &= (a_{11} + a_{22})(b_{11} + b_{22}) && (3)\\ m_3 &= (a_{11} - a_{21})(b_{11} + b_{12}) && (4)\\ m_4 &= (a_{11} + a_{12})b_{22} && (5)\\ m_5 &= a_{11}(b_{12} - b_{22}) && (6)\\ m_6 &= a_{22}(b_{21} - b_{11}) && (7)\\ m_7 &= (a_{21} + a_{22})b_{11}. && (8) \end{aligned}$$

   Then compute the entries in the product matrix with

$$\begin{aligned} c_{11} &= m_1 + m_2 - m_4 + m_6 && (9)\\ c_{12} &= m_4 + m_5 && (10)\\ c_{21} &= m_6 + m_7 && (11)\\ c_{22} &= m_2 - m_3 + m_5 - m_7. && (12) \end{aligned}$$

   With this technique, we compute the product with seven multiplications and 18 additions rather than the usual eight multiplications and four additions. Although this does not result in a savings for the case of 2×2 matrices, the technique can be generalized to improve up the asymptotic running time of the naive algorithm, which requires $O(n^3)$ operations.

   (a) Generalize the above method to derive a recursive method for multiplying two $n \times n$ matrices. You may assume n is a power of 2.

   (b) Write the recursion for the running time of your method, solve it, and show that the running time of this method is asymptotically better than that of the naive algorithm.

(c) Propose a parallel version of this algorithm and analyze its running time using the PRAM model of computation. Can the number of processors be chosen so as to make this algorithm cost-optimal? Would the running time be affected if the underlying architecture had more restricted communication, such as with a mesh?

3. Implement a parallel version of merge sort using OpenMP and analyze it empirically. Compare your results to your theoretical analysis from Homework 2. Your program should be capable of utilizing a variable number of threads and a variable input list size.

4. Suppose we have a graph with n vertices that is represented by a set of n bit vectors v^i, where $v^i_j = 1$ if there is an edge from vertex i to vertex j. Find an $O(n)$ algorithm to determine the vector p^1 for which $p^1_j = 1$ if there is a path from vertex 1 to vertex j. The operations you may use are bitwise logical operations on bit vectors, arithmetic operations on integers, instructions which set the value of a particular bit of a particular vector, and an instruction that assigns the value j to an integer variable a if the left-most "1" in a vector v is in position j or sets $a = 0$ if v consists of all zeros.