1. (a) Based on your proposed sparse matrix data structure from Homework 2, implement a sparse matrix class in C++. Your class should allow for the user to choose whether the matrix is to be stored in sparse or dense format.

(b) Override the \( \star \) operator so that two instances of your matrix class can be multiplied as in Matlab.

(c) Do some computational experiments comparing the empirical running time for multiplying random matrices of different densities stored both in sparse and dense formats. Use Matlab as a baseline for comparison.

2. Propose a sparse matrix class in Python. You don’t have to actually code it for this homework, but discuss how it would be implemented and research how you would override the \( \star \) operator in this case.

3. Consider the following algorithm for multiplying 2\( \times \)2 matrices. To compute the product
\[
\begin{bmatrix}
c_{11} & c_{12} \\
c_{21} & c_{22}
\end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix},
\]
we first compute the following products:

\[
\begin{align*}
m_1 &= (a_{12} - a_{22})(b_{21} + b_{22}) \\ m_2 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ m_3 &= (a_{11} - a_{21})(b_{11} + b_{12}) \\ m_4 &= (a_{11} + a_{12})b_{22} \\ m_5 &= a_{11}(b_{12} - b_{22}) \\ m_6 &= a_{22}(b_{21} - b_{11}) \\ m_7 &= (a_{21} + a_{22})b_{11}.
\end{align*}
\]

Then compute the entries in the product matrix with
\[
\begin{align*}
c_{11} &= m_1 + m_2 - m_4 + m_6 \\ c_{12} &= m_4 + m_5 \\ c_{21} &= m_6 + m_7 \\ c_{22} &= m_2 - m_3 + m_5 - m_7.
\end{align*}
\]

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\( \times \)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 

4. Solve each of the following recurrences, assuming that $T(n)$ is constant for sufficiently 
small values of $n$. Explain how you got your answer.

   (a) $T(n) = 2T(n/2) + n^3$
   (b) $T(n) = 2T(n/4) + \sqrt{n}$
   (c) $T(n) = T(\sqrt{n})$
   (d) $T(n) = 4T(n/2) + n^2 \sqrt{n}$

5. Suppose a list $A$ contains only integers from 0 to $2^i$ for some given positive integer 
$i$. Explain how to do binary search to find out if a given integer appears in the list 
using only bit operations. This method would be useful when $i$ is very large and 
we cannot assume that the basic operations involving the numbers in the array are 
constant-time. In this case, the only operation we can use to access information about 
the numbers in the array in constant-time is “fetch the $j^{th}$ bit of $A[i]$,” which takes 
constant time. What is the running time of your binary search algorithm in terms of 
the number of such operations?