Justin Bahr
NUID: 001592707

High Performance Computing
Homework 2 - Question 4

a. Reverse Cuthill-McKee SpMV reordering can be used for symmetrically sparse square matrices. This recording algorithm is based on a breadth-first search where the matrix is viewed like a graph's adjacency matrix. It results in a reduced bandwidth since nonzero elements are closer to the main diagonal (and each other), which provides a smoother access pattern and more compact storage in a CSR format. Note that degree refers to the number of edges a vertex of the graph representation has. Pseudocode below:
    1. Input a square nxn matrix A.
    2. Create an empty queue Q.
    3. Create an empty list R.
    4. Create an array of all false values V of size n.
    5. While (length(R) < n) // not all vertices of graph A have been appended to R
        a. Find the vertex p with minimum degree and add it to R.
        b. Mark p as visited and add p to Q.
        c. While (length(Q) > 0) // Q is not empty
            i. Dequeue first element c from Q.
            ii. For each unvisited edge n neighbor of c (sorted by increases degree)
                - Add n to R.
                - Mark n as visited and add it to Q.
    6. Reverse the order of R (swap(R[i], R[n-i-1]) for i = 0 to n/2).
    7. Output R.

Approximate Minimum Degree SpMV reordering can be used for symmetric square matrices. This reordering algorithm is based on selecting the vertex with the smallest degree (fewest connections) in the elimination graph. The elimination graph represents the matrix as vertices (rows) connected by edges wherever a nonzero occurs above the diagonal. When a vertex is eliminated, its neighbors form a clique, creating fill-in. The algorithm minimizes this by choosing low-degree vertices first. It is effective before LU/Cholesky factorization as it creates a sparse triangular factor. Pseudocode below:

1. Input a square nxn matrix A.
2. Create:
    - degree[n]: Array to store the degree of each node
    - visited[]: Array to track visited nodes
    - p[]: Permutation vector
3. For i from 0 to n-1:
    a. degree[i] = number of nonzeros values in row i
    b. adjList[i] = list of nodes adjacent to node i
4. For i from 0 to n-1:
    a. v = node with the minimum degree[] value not in visited[]
    b. Add v to visited[]
    c. p[i] = v
    d. Update degree[] value of nodes adjacent to v
    e. For each neighbor u of v:
        i. For each neighbor w of u:
            - If w isn't a neighbor of v, increment degree[u] and degree[w] by 1.
5. Return permutation vector p.

b. One hardware platform used in this paper is Naples with an AMD Epyc 7601 CPU. It's memory hierarchy is as follows:
    - L1 Instruction Cache: 64 KB, 4-way set associative, per core
    - L1 Data Cache: 32 KB, 8-way set associative, per core, write back
    - L2 Cache: 512 KB, 8-way set associative, per core, write back
    - L3 Cache: 64 MB, 16-way set associative, per socket, write back