1. A sparse matrix is a matrix containing a lot of zero elements. We can use linked lists to store only the non-zero elements of a sparse matrix to save memory space in a computer.

(a) (7 pts.) Design a data structure for an n by n sparse matrix A. Define such data structure using C or a pseudo code and name it "matrix". (Each element in A is a floating point value.)

(b) (8 pts.) Write a sub-routine named **`float aij(matrix A, int i, int j)`** to return the element of a_{ij}, where a_{ij} is the element at the i-th row and j-th column in sparse matrix A.

(c) (20 pts.) Let det A denote the determinant of an n by n matrix A. The determinant can be determined recursively in the following way. If n is 1, then det A is equivalent to the only element in the matrix. For n larger than 1, $\det A = \sum_{k=1}^{n} (-1)^{1+k} a_{1k} \det S_k$, where S_k is the sub-matrix of A resulting from the deletion of row 1 and column k. For example,

$$\det\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} = (-1)^{1+1}\times 1\times\det\begin{bmatrix}5&6\\8&9\end{bmatrix} + (-1)^{1+2}\times 2\times\det\begin{bmatrix}4&6\\7&9\end{bmatrix} + (-1)^{1+3}\times 3\times\det\begin{bmatrix}4&5\\7&8\end{bmatrix}.$$

Write a sub-routine called **`float determinant(matrix A)`** that uses the recursive formula $\det A = \sum_{k=1}^{n} (-1)^{1+k} a_{1k} \det S_k$ to obtain the determinant of the input sparse matrix A.

2. (15 pts.) Write a sub-routine, **`void path(int G[N][N], int s, int d)`**, in C language to find a path in an undirected graph. Parameter G is a two-dimensional integer array that is the adjacency matrix of the input graph. The constant value N is the number of vertices in the graph. In array G, the element $G[i][j]$ is 1 if there is an edge between vertex i and vertex j; otherwise, the value of $G[i][j]$ is 0. The parameter s is the ID of the source vertex and d is the ID of the destination vertex. Your program has to show the node ID along the path from the source to the destination.

3. Consider a maximum heap which is defined by an integer array $H[N]$ where N is the size of the heap array. For the questions below, use C language or pseudo code to complete the tasks.

(a) (15 pts.) Write a function to convert the input array into a max heap. The function prototype is given as **`void createHeap(int [], int)`** where the first parameter is the heap array and the second parameter is the size of the array.

(b) (10 pts.) Including the above function, write a function to find the k^{th} largest element in the heap. The function prototype is given as **`int kthElement(int [], int, int)`** where the first parameter is the heap array, the second parameter is the size of the array, and the third parameter is the k.

(c) (10 pts.) Analyze the function in (a) and determine its time complexity in the Big-O notation. Be sure to show the details of your analysis along with proper calculations.

4. Consider a hash and answer the following.

(a) (9 pts.) Complete the resulting hash table (list size 9) below after inserting the following sequence of data from left to right using the following hash function and the probing method.

- ✓ Keys: 152, 214, 314, 504, 600
- ✓ Address = H(key) = key % list_size
- ✓ Probing function: linear probing

Table[0]	
Table[1]	
Table[2]	
Table[3]	
Table[4]	
Table[5]	
Table[6]	
Table[7]	
Table[8]	

(b) (3 pts.) Describe another probing function for above to reduce the number of collisions. Show also your resulting table and explain why it contains fewer collisions.

(c) (3 pts.) Describe another hash function which may result in fewer collisions. Show also your resulting table and explain why it contains fewer collisions.