#include <iostream>  
#include <vector>  
#include <ctime>  
#include <cassert>  
#include <fstream>  
#include <string>  
  
using namespace std;  
  
// MergeSort function declarations  
void mergeSort(vector<string> &a, vector<string> &tmp, int left, int right);  
  
void mergeSortedLists(vector<string> &a, vector<string> &tmp, int leftPos, int rightPos, int rightEnd);  
  
// BubbleSort function declaration  
void swap(string \*a, string \*b);  
  
void bubbleSort(string \*a, int n);  
  
int main(int argc char\* argv[]) {  
  
// the command line argument should be the follow  
// BubbleSort.exe CSC112\_Project\_3\_NBA\_Dataset\_Frank.csv bubble.txt merge.txt  
  
 ifstream inFile;  
 ofstream outFileMerge;  
 ofstream outFileBubble;  
 string tmp; // string that will store unused information  
 // open input and output files here  
 inFile.open("CSC112\_Project\_3\_NBA\_Dataset\_Frank.csv"); // open the data from NBA dataset  
 outFileBubble.open("bubble.txt"); // create a new file that stores the sorted list by using bubbleSort  
 outFileMerge.open("merge.txt"); // create a new file that stores the sorted list by using mergeSort  
 // check if these input file and output file opened successfully  
 if (!inFile.is\_open()) {  
 cout << " Could not open NBA dataset" << endl;  
 return 1; // something bad happened , return non‐zero  
 }  
 if (!outFileMerge.is\_open()) {  
 cout << "Could not open file merge.txt." << endl;  
 return 1;  
 }  
 if (!outFileBubble.is\_open()) {  
 cout << "Could not open file bubble.txt." << endl;  
 return 1;  
 }  
  
 // get input: vector length which is the total number of entries in your data set  
 // Dr. Alqahtani said in office hour that to ignore the instruction and hard code the length of your dataset  
 vector<string> v(4551); // vector to be sorted  
 vector<string> t(4551); // temporary workspace  
  
 // define a string pointer dynamically allocate an array of strings  
  
 string \*stringPtr = new string[4551]; // declare a dynamically allocated array with size of dataset  
  
  
 // populate and print input  
 cout << "Unsorted:" << endl;  
  
 getline(inFile, tmp);  
 for (int i = 0; i < 4550; i++) {  
 getline(inFile, tmp, ',');  
 v.at(i) = tmp;  
 getline(inFile, tmp, ',');  
 getline(inFile, tmp, ',');  
 getline(inFile, tmp, ',');  
 getline(inFile, tmp, ',');  
 getline(inFile, tmp, ',');  
 getline(inFile, tmp);  
 }  
 for (int j = 0; j < 4551; j++) {  
 cout << v.at(j) << endl;  
 }  
 cout << endl;  
  
 // copy the list of strings from vector to array  
 for (int k = 0; k < 4551; k++) {  
 stringPtr[k] = v.at(k);  
 }  
  
 clock\_t start\_mergeSort = clock();  
 // sort vector using mergeSort  
 mergeSort(v, t, 0, v.size() - 1);  
 clock\_t end\_mergeSort = clock();  
  
 // check output, make sure vector is sorted after mergeSort  
 for (int i = 1; i < v.size(); i++) {  
 assert(v.at(i - 1) <= v.at(i));  
 }  
  
  
 clock\_t start\_bubbleSort = clock();  
 // sort array using bubbleSort  
 bubbleSort(stringPtr, 4551);  
 clock\_t end\_bubbleSort = clock();  
  
 // check output, make sure array is sorted after bubbleSort  
 for (int i = 1; i < v.size(); i++) {  
 assert(stringPtr[i - 1] <= stringPtr[i]);  
 }  
  
 // print sorted vector after mergeSort into "merge.txt"  
 cout << "Sorted:" << endl;  
 for (int i = 0; i < v.size(); i++) {  
 outFileMerge << v.at(i) << endl;  
 }  
 cout << endl;  
  
 // print sorted array after bubbleSort into file "bubble.txt"  
 for (int i = 0; i < v.size(); i++) {  
 outFileBubble << stringPtr[i] << endl;  
 }  
 cout << endl;  
  
  
  
 // print elapsed time for mergeSort and bubbleSort  
 double elapsed\_mergeSort = double(end\_mergeSort - start\_mergeSort) / CLOCKS\_PER\_SEC;  
 double elapsed\_bubbleSort = double(end\_bubbleSort - start\_bubbleSort) / CLOCKS\_PER\_SEC;  
  
 cout << elapsed\_mergeSort << " " << elapsed\_bubbleSort << endl;  
  
 // close all related input and output file stream  
 inFile.close();  
 outFileMerge.close();  
 outFileBubble.close();  
  
 // delete the dynamically allocated string pointer array  
 delete[] stringPtr; // free the memory of the array  
 stringPtr = NULL;  
  
  
 //*FIXME refer to 16.16 zybook to improve the structure* //*FIXME refer to 10.5.2 to add eof and fail statement* // *FIXME input output file 都要check 是否Open* return 0;  
}  
  
void mergeSortedLists(vector<string>& a, vector<string>& tmp, int leftPos, int rightPos, int rightEnd) {  
 int leftEnd = rightPos - 1;  
 int tempPos = leftPos;  
 int numElements = rightEnd - leftPos + 1;  
  
 while (leftPos <= leftEnd && rightPos <= rightEnd) {  
 if (a[leftPos] <= a[rightPos]) {  
 tmp[tempPos++] = a[leftPos++];  
 } else {  
 tmp[tempPos++] = a[rightPos++];  
 }  
  
 }  
  
 while (leftPos <= leftEnd) {  
 tmp[tempPos++] = a[leftPos++];  
 }  
 while (rightPos <= rightEnd) {  
 tmp[tempPos++] = a[rightPos++];  
 }  
  
 for (int i = 0; i < numElements; i++, --rightEnd) {  
 a[rightEnd] = tmp[rightEnd];  
 }  
}  
  
void mergeSort(vector<string>& a, vector<string>& tmp, int left, int right) {  
 if (left < right) {  
 int center = ( left + right ) / 2;  
 mergeSort(a, tmp, left, center);  
 mergeSort(a, tmp, center + 1, right);  
 mergeSortedLists(a, tmp, left, center + 1, right);  
 }  
}  
  
// Swap function  
void swap(string \*a, string \*b) {  
 string tmp;  
 tmp = \*a;  
 \*a = \*b;  
 \*b = tmp;  
}  
  
// BubbleSort function  
void bubbleSort(string \*a, int n) {  
 int i;  
 int j;  
 for (i = 0; i < n - 1; i++) {  
 for (j = 0; j < n - i - 1; j++) {  
 if (\*(a + j) > \*(a + j + 1)) {  
 swap(\*(a + j), \*(a + j + 1));  
 }  
 }  
 }  
}