COMP 53: Search and Sort Lab, part 5

Instructions: In this lab, we are going to review merge sort.

- Get into groups of **at most two people** to accomplish this lab.
- At the top of your source code files list the group members as a comment.
- Each member of the group must individually submit the lab in Canvas.
- This lab includes **18 points** in aggregate. The details are given in the following.

1 `city.h`

Use `city.h` from the previous lab without any modifications.

2 `main.cpp`

In `main.cpp` do the following step by step:

1. Globally define array `cityArray[]` consisting of cities with the following details:
   (a) Los Angeles with population of 4 million
   (b) San Diego with population of 1.5 million
   (c) San Francisco with population of 900 thousand
   (d) Sacramento with population of 500 thousand
   (e) Stockton with the population of 300 thousand
   (f) Redding with the population of 90 thousand
   (g) Las Vegas with the population of 700 thousand
   (h) Reno with the population of 300 thousand
   (i) Portland with the population of 700 thousand
   (j) Seattle with the population of 750 thousand
   (k) Eugene with the population of 200 thousand
2. Globally define a vector of `City` objects, without initial values. Call it `cityVector` ***(1 points)***.
3. Pass vectors to these functions as *reference*, and define them as *constant* if the functions are not allowed to modify them.
   (a) Define function `void initVector(...)` that receives a vector of `City` objects, an array of elements of type `City` as a second input, and an integer as its third input. The third input represents the number of elements in the input array. Initialize the input queue with the elements existing in the input array ***(2 points)***.
   (b) Define function `void printCityVector(...)` that receives a vector of `City` objects as input and prints the elements within the vector. *Hint*: You can use range-based `for` loops ***(2 points)***.
   (c) Define function `int mergeCityVector(...)` that receives a vector of `City` objects as input, along with three integers as indexes that represent the lower bound, the division point of the vector into two halves, and the upper bound within the vector. It merges the two halves of the vector (assuming they are sorted) according to the city populations ***(5 points)***.

(d) Define function `void cityMergeSort(...)` that receives a vector of `City` objects as input, along with two integers as indexes that represent the lower and upper boundaries within the vector. It does merge sort on the vector of `City` objects according to the city populations (by invoking the `mergeCityVector()` function on sorted vectors ***(5 points)***.

In `main()` function do the following step by step, using the functions defined above:

(i) Initialize `cityVector` according to array `cityArray[]` using the function defined above ***(1 points)***.

(ii) Print out the entries of `cityVector`, using the appropriate function defined above ***(1 points)***.

(iii) Do merge sort on `cityVector` and print out the updated vector. ***(1 points)***.

The output of the program may look like the following:

```
Initializing cityVector with cityArray[]:
Los Angeles: 4000000
San Diego: 1500000
San Francisco: 900000
Sacramento: 500000
Stockton: 300000
Redding: 90000
Las Vegas: 700000
Reno: 300000
Portland: 700000
Seattle: 750000
Eugene: 200000


Merge sort on cityVector:
Redding: 90000
Eugene: 200000
Stockton: 300000
Reno: 300000
Sacramento: 500000
Las Vegas: 700000
Portland: 700000
Seattle: 750000
San Francisco: 900000
San Diego: 1500000
Los Angeles: 4000000
```