COMP 53: Search and Sort Lab, part 1

Instructions: In this lab, we are going to review linear and binary search.

- 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 **27 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. Globally define an integer `steps` that is used to count the number of comparisons in the search.
4. 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 a function that receives two cities as input and returns true if name of the first city is smaller than name of the second city. Otherwise it returns false. We will use this function to sort the vector of `City` objects later on. ***(2 points)***.

(d) Define function `unsigned int cityLinearSearch(...)` that receives a vector of `City` objects as input along with a city name (a string). It does a linear search on the vector and if successful returns the population associated with the input city name. Otherwise, it returns 0. Moreover, you need to count the number of comparisons, by updating `steps`. ***(4 points)***.

(e) Define function `unsigned int cityBinarySearch(...)` that receives a vector of `City` objects as input along with a city name (a string). It does a binary search on the vector and if successful returns the population associated with the input city name. Otherwise, it returns 0. Moreover, you need to count the number of comparisons, by updating `steps`. ***(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) Sort `cityVector` based on names in ascending order, and print out the updated vector. *Hint*: Use `sort()` function ***(1 points)***.

(iv) Do a linear search on `cityVector` to find the population of San Francisco. Print the population along with the number of comparisons for this search ***(2 points)***.

(v) Do a linear search on `cityVector` to find the population of Boston. Print the population along with the number of comparisons for this search ***(2 points)***.

(vi) Do a binary search on `cityVector` to find the population of San Francisco. Print the population along with the number of comparisons for this search ***(2 points)***.

(vii) Do a binary search on `cityVector` to find the population of Boston. Print the population along with the number of comparisons for this search ***(2 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


Sorting cityVector based on names in ascending order:
Eugene: 200000
Las Vegas: 700000
Los Angeles: 4000000
Portland: 700000
Redding: 90000
Reno: 300000
```

```
Sacramento: 500000
San Diego: 1500000
San Francisco: 900000
Seattle: 750000
Stockton: 300000


Linear search for the population of San Francisco: 900000
Number of comparisons for this search: 9

Linear search for the population of Boston: 0
Number of comparisons for this search: 11

Binary search for the population of San Francisco: 900000
Number of comparisons for this search: 2

Binary search for the population of Boston: 0
Number of comparisons for this search: 3
```