For this assignment, you will use C++ to implement one sorting algorithm, and to compare it to two standard sorting routines.

**Step 1: Implementation (40 points)**
Write a program that sorts an array of integers. Specifically, your program should do these things, in this order:

1. First, read a string from standard input (that is, from `cin`), indicating which sorting algorithm to use. This string will be one of these three choices: selection, qsort, or stl.

2. Second, read a list of positive integers, one on each line, from standard input. The last line will have a 0, which indicates the end of input, but should not be included in the sorting. Store these input integers in a `vector<int>`.

3. Next, sort that vector, using one of three different sorting functions. For each algorithm, keep track of the number of key comparisons made by the algorithm in a global variable of type `long long`. (The code snippets below use the name `comparisons` for this variable.)

3a. If the first line of input was selection, then use selection sort, which you should implement yourself. Details may be found in your notes from the lecture on September 27, or in Chapter 3 of your textbook.

3b. If the first line of input was qsort, then use the `qsort` function, which has been around the C language for a very long time (since at least 1984), and is still available in modern C++ programs. To use it:
   - Include the declarations of the C standard library functions:
     ```c
     #include <stdlib.h>
     ```
   - Define a comparison function that will be called by `qsort`:
     ```c
     int qsortCompare (const void *a, const void *b) {
         comparisons++;
         if (*(int*)a < *(int*)b) return -1;
         if (*(int*)a == *(int*)b) return 0;
         return 1;
     }
     ```
     This function is called each time `qsort` needs to compare two array elements. In it, we increment our comparison count and return -1, 0, or +1, based on the relationship between the two integers that `qsort` wants to compare.
   - Call the function, passing a pointer to the first element, the number of elements, the number of bytes in each element, and a pointer to the compare function:
qsort(&A[0]), A.size(), sizeof(int), qsortCompare);

3c. If the first line of the input was stl, then use the sort function, which is included as part of the C++ Standard Template Library (STL). To use it:

- Include the declarations of the C++ STL algorithms library:
  
  ```
  #include <algorithm>
  ```

- Import the std namespace, to avoid writing std:: over and over:
  
  ```
  using namespace std;
  ```

- Define a comparison function that will be called by sort:
  
  ```
  bool stlSortCompare (int i, int j) {
    comparisons++;
    return i<j;
  }
  ```

  This function is called each time sort needs to compare two array elements. Its form is similar to the comparison for qsort, but the parameters and return values are slightly different.

- Call the function, passing iterators for the start and end of the vector, along with a pointer to the comparison function.
  
  ```
  sort(A.begin(), A.end(), stlSortCompare);
  ```

4. After the sorting is done, print out the elements of the sorted array on standard output (cout), one per line.

5. Print one final line that says Comparisons: , followed by the number of comparisons made by the sorting algorithm.

An example input and the corresponding correct output is shown in Figure 1. Your code should be contained in a single C++ source file. I will compile this program using this command line:

```
  g++ -Wall -std=c++11 yourfile.cpp
```

Note that, aside from the snippets above, no “starter code” is available; you should write the entire program from scratch. For calibration purposes, my solution took about 70 lines of non-comment C++ code.

**Step 2: Analysis (10 points)**

1. Use your program from Step 1 to execute all three sorting algorithms on randomly-created inputs of these sizes:

   \[ n = 200, 400, 600, \ldots, 6000. \]

   The course website has a program called `generate_sort.sh` that can generate random input files for you; see the comments in that file for details.

   You will need to run your program a total of \(3 \cdot 30 = 90\) times. You can do this by hand (which might be tedious) or by writing a script (which is recommended, but not strictly necessary). Record the number of comparisons for each algorithm and for each input size.
<table>
<thead>
<tr>
<th>Input:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>qsort</code></td>
<td>12</td>
</tr>
<tr>
<td>23</td>
<td>21</td>
</tr>
<tr>
<td>4123</td>
<td>23</td>
</tr>
<tr>
<td>412</td>
<td>24</td>
</tr>
<tr>
<td>434</td>
<td>43</td>
</tr>
<tr>
<td>1234</td>
<td>43</td>
</tr>
<tr>
<td>21</td>
<td>43</td>
</tr>
<tr>
<td>43</td>
<td>213</td>
</tr>
<tr>
<td>2143</td>
<td>213</td>
</tr>
<tr>
<td>213</td>
<td>214</td>
</tr>
<tr>
<td>4124</td>
<td>412</td>
</tr>
<tr>
<td>2143</td>
<td>412</td>
</tr>
<tr>
<td>213</td>
<td>412</td>
</tr>
<tr>
<td>412</td>
<td>434</td>
</tr>
<tr>
<td>43</td>
<td>1234</td>
</tr>
<tr>
<td>214</td>
<td>2143</td>
</tr>
<tr>
<td>12</td>
<td>2143</td>
</tr>
<tr>
<td>412</td>
<td>4123</td>
</tr>
<tr>
<td>43</td>
<td>4124</td>
</tr>
<tr>
<td>24</td>
<td>Comparisons: 61</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: An example input and the correct output.

2. Use your favorite plotting software to create two line graphs, each showing the number of comparisons for each algorithm as a function of the input size. The first graph should show all three algorithms; the second should show only `qsort` and `stl`. Figure 2 shows examples of the format of the plots, with fake data. Your graphs will look different, because your data will be real. (The need for two separate graphs comes from the fact that if things are working correctly, the comparison count for selection sort will grow much faster. Omitting it in the second graph will make any differences between `qsort` and STL’s `sort` easier to see.)

3. Write one paragraph of English text answering these questions: Does your data seem consistent with our analysis of selection sort? What order of growth (as a function of $n$, expressed in $\Theta$ notation) do you observe for `qsort` and the STL `sort` function? Which of these three would you recommend for real use?

**What to Submit**

1. You should submit, using the department’s dropbox website, your single C++ source file.

2. You should submit, either on paper or by email with PDF attachment, your line graphs and your answers to the questions above.
Figure 2: An example of the format for the two required plots. Your data will be different, but the plot should have these axes and these data series.