**Overview**

The focus of these problems will be working with information extracted from a municipal government data feed containing bids submitted for auction of property. All materials for this assignment can be found in the Supporting Materials section below. The data set is provided in two comma-separated files:

1. eBid\_Monthly\_Sales.csv (larger set of 12,023 bids)
2. eBid\_Monthly\_Sales\_Dec\_2016.csv (smaller set of 76 bids)

This assignment is designed to explore sorting algorithms by implementing both a selection sort and quicksort of a vector of bids loaded from a CSV file.

We provide a starter console program that uses a menu to enable testing of the sort logic you will complete. It also allows you to pass in the path to the bids CSV file to be loaded, enabling you to try both files. In this version, the following menu is presented when the program is run:

**Menu:**

* 1. **Load Bids**
  2. **Display All Bids**
  3. **Selection Sort All Bids**
  4. **QuickSort All Bids**
  5. **Exit**

**Enter choice:**

The VectorSorting.cpp program is partially completed. It contains empty methods representing the programming interface used to interact with the linked list. You will need to add logic to the methods to implement the necessary behavior. Here are the methods in VectorSorting.cpp that you must complete:

void selectionSort(vector& bids)   
  
void quickSort(vector& bids, int begin, int end)  
int partition(vector& bids, int begin, int end)

**Prompt**

You will need to perform the following steps to complete this activity:

**Setup:** Begin by creating a new C++ project with a project type of "Hello World C++ Project".

1. Name the project “VectorSorting”. Remember to pick the correct compiler in **Toolchains** and click **Finish**. This will create a simple VectorSorting.cpp source file under the **/src** directory.
2. Download the starter program files and copy them to the project’s **/src** directory, replacing the existing auto-generated ones. Remember to right-click on the project in the Project Explorer pane on the left and **Refresh** the project so it adds all the new files to the **src** folder underneath.
3. Because this activity uses C++ 11 features, you may need to add the **-std=c++11** compiler switch to the miscellaneous settings.

**Task 1:** Implement the selection sort algorithm:

1. Code the selection sort logic using “bid.title” as the sort field.
2. Invoke the selectionSort() method from the main() method including collecting and reporting timing results.

**Task 2:** Implement the quicksort algorithm:

1. Code the quicksort logic using “bid.title” as the sort field.
2. Invoke the quickSort() method from the main() method including collecting and reporting timing results.

Here is sample output from running the completed program:

> ./VectorSorting ~/Downloads/eBid\_Monthly\_Sales.csv  
> VectorSorting.exe Downloads\eBid\_Monthly\_Sales.csv

**Loading bids from CSV and performing a selection sort:**

|  |  |  |
| --- | --- | --- |
| **Example Input** | **Choice: 1** | **Choice: 3** |
| **Display** | Menu: 1. Load Bids 2. Display All Bids 3. Selection Sort All Bids 4. QuickSort All Bids 9. Exit Enter choice: 1 | Menu: 1. Load Bids 2. Display All Bids 3. Selection Sort All Bids 4. QuickSort All Bids 9. Exit Enter choice: 3 |
| **Output** | Loading CSV file eBid\_Monthly\_Sales.csv 12023 bids read time: 12023 clock ticks time: 0.173945 seconds | 12023 bids sorted time: 10623604 clock ticks time: 10.6236 seconds |

**Loading bids from CSV and performing a quicksort:**

|  |  |  |
| --- | --- | --- |
| **Example Input** | **Choice: 1** | **Choice: 4** |
| **Display** | Menu: 1. Load Bids 2. Display All Bids 3. Selection Sort All Bids 4. QuickSort All Bids 9. Exit Enter choice: 1 | Menu: 1. Load Bids 2. Display All Bids 3. Selection Sort All Bids 4. QuickSort All Bids 9. Exit Enter choice: 4 |
| **Output** | Loading CSV file eBid\_Monthly\_Sales.csv 12023 bids read time: 174985 clock ticks time: 0.174985 seconds | 12023 bids sorted time: 47964 clock ticks time: 0.047964 seconds |

Note the dramatic difference in the time it takes to sort over 12,000 records - 10.6 seconds versus 4 *hundredths* of a second!

Your assignment submission must address the following rubric criteria:

* **Code Reflection:** A brief explanation of the code and its purpose, and a brief discussion of your experience in developing it, including any issues that you encountered while completing the exercise and what approaches you took to solve them
* **Pseudocode or Flowchart:** A pseudocode or flowchart description of the code that is clear and understandable and captures accurate logic to translate to the programming language
* **Specifications and Correctness:** Source code must meet its specifications and behave as desired. Correct code produces the correct output as defined by the data and problem; however, you should also produce fully functioning code (with no errors) that aligns with as many of the specifications as possible. You should write your code in such a way that the submitted file executes, even if it does not produce the correct output. You will be given credit for partially correct output that can be viewed and seen to be partially correct.
* **Annotation / Documentation:** All code should also be well-commented. This is a practiced art that requires striking a balance between commenting everything, which adds a great deal of unneeded noise to the code, and commenting nothing. Well-annotated code requires you to:
  + Explain the purpose of lines or sections of your code, detailing the approach and method you took to achieve a specific task in the code.
  + Document any section of code that is producing errors or incorrect results.
* **Modular and Reusable:** Programmers should develop code that is modular and reusable. If it contains functionality and responsibility in distinct methods, code is more flexible and maintainable. Your code should adhere to the single responsibility principle—classes and methods should do only one job. If you can replace a method with another that uses a different technique or implementation without impacting (having to refactor or rewrite) other parts of your code, then you have succeeded in creating modular methods.
* **Readability:** Code needs to be readable to a knowledgeable programmer. In this course, readable code requires:
  + Consistent, appropriate whitespace (blank lines, spaces) and indentation to separate distinct parts of the code and operations
  + Explicit, consistent variable names, which should clearly indicate the data they hold and be formatted consistently: for example, numOrders (camelCase) or item\_cost (underscored)
  + Organized structure and clear design that separates components with different responsibilities or grouping-related code into blocks

**Guidelines for Submission**

To complete this assignment, submit the **CPP code files** and a **code reflection and associated pseudocode or flowchart** written portion. Your written portion should be 1–2 paragraphs in length.

**Supporting Materials**

[**CS 300 Vector Sorting Assignment Student Files.zip**](https://learn.snhu.edu/content/enforced/1160013-CS-300-H7580-OL-TRAD-UG.22EW1/course_documents/CS%20300%20Vector%20Sorting%20Assignment%20Student%20Files.zip?_&d2lSessionVal=4GmVs2XzthMaf89KjcY7ioycY&ou=1160013)  
Download this zipped file folder to begin your assignment. The data sets you will use in this assignment are provided in these comma-separated files:

* eBid\_Monthly\_Sales.csv (larger set of 12,023 bids)
* eBid\_Monthly\_Sales\_Dec\_2016.csv (smaller set of 77 bids)
* VectorSorting.cpp program, which is a partially completed program that you can use as a starting point for the assignment

| **Criteria** | **Proficient (100%)** | **Needs Improvement (70%)** | **Not Evident  (0%)** | **Value** |
| --- | --- | --- | --- | --- |
| **Code Reflection** | Describes purpose of code, techniques implemented to solve problem, challenges encountered, and approaches to overcome the challenges | Lacks details in code purpose, techniques implemented, or challenges encountered | Does not explain purpose of code, techniques used, or challenges encountered | 25 |
| **Pseudocode or Flowchart** | Pseudocode or flowchart is clear and understandable and captures accurate logic to translate to the programming language | Pseudocode or flowchart has errors or omissions that affect its clarity or understandability, or the logic to translate to the programming language is inaccurate or incomplete | Pseudocode or flowchart does not contain the logic to translate to the programming language | 10 |
| **Specifications and Correctness: Algorithm** | All algorithm specifications are met completely and function in all cases | Details of the specifications are violated, or program often exhibits incorrect behavior | Program only functions correctly in very limited cases or not at all | 20 |
| **Specifications and Correctness: Data Structure** | All data structure specifications are met completely and function in all cases | Details of the specifications are violated, or program often exhibits incorrect behavior | Program only functions correctly in very limited cases or not at all | 20 |
| **Annotation / Documentation** | Code annotations explain and facilitate navigation of the code | Comments provide little assistance with understanding the code | Code annotations do not explain the code or do not facilitate navigation of code, or code is not fully or logically annotated | 10 |
| **Modular and Reusable** | Methods are limited in scope and responsibility, and both algorithms and data structures are implemented in such a way that they can be reused in other programs | Methods have errors in scope or responsibility, or algorithms or data structure are overly tied to the specific program | No attempt was made to develop modular or reusable code | 10 |
| **Readability** | Code follows proper syntax and demonstrates deliberate attention spacing, whitespace, and variable naming | Code contains variations from established syntax and conventions | Code contains significant variations from established syntax and conventions | 5 |
| Total: | | | | 100% |

Pseudocode for partition:

Pivot = middle element of the vector

While (low is not exceed the high):

Increase the low until pivot < bid[low]

decrease the high value until pivot > bids[high]

if low is less or equal to high swap the element

increase the low by one and decrease the high by one

return the high

quicksort:

set mid = 0

if begin is greater or ewaul to end return

mid = partion(bids,begin ,end)

recursive call to quicksort with begin and mid

recursive call to quicksort with mid+1 and end

Selection sort:  
vector

Size: size of the vecttor

For pos = 0 to size -1 of the vector:

Set min = pos

For j = pos+! to size

If min is greater then current bid title:

Set min = j

If(min != pos)

Swap the position of j and pos