# CS 7345 Sorting Library- Single Threaded API Documentation:

a. **Overview:** This library allows for the creation of a user-specified vector and the running of several sorting algorithms on any vector of numbers passed into it. The vector can be specified to be in reverse order (from how it was created), completely random-shuffled (if the specified array was not already in that order), and duplicated (80% duplicated, which tests edge cases of many sorting algorithms and is thus also typically used as a benchmark).
    i. **User Input:** The array can be of any length (at least, any before the hardware/browser completely runs out of memory to supply) and **must contain only integers.**
    ii. **Sorting Algorithms:** Bubble, Merge, Insertion

b. **Endpoints:** All functions within the JS port of the library are called through a defined "Algo" object. This allows for single-file instantiation and readability of commands. The commands that can be called are divided into two families- *Algo.Vector*, which encompasses all functions that create and manipulate the target vector to be sorted- and *Algo.Sort*, which encompasses the previously mentioned sorting algorithms -Bubble, Merge, and Insertion.
    i. ***Algo.Vector:***
        1. *Algo.Vector.Init(int):* creates an empty VectorInt, the emscripten-bound equivalent of a C++ std::vector<int>, which can then be populated using the C++ push_back(data) method or other C++-centric means (with some changes implemented in Embind, such as VectorInt.get(i) instead of indexing std::vector<int>[i] and VectorInt.resize() instead of std::<vector>int.reserve()) and used within the library's other functions.
            a. The argument "size" passed into the method is optional. If none is supplied, an empty VectorInt is returned. If a size is supplied, a VectorInt populated with a default value of ascending numbers (ie: {1,2,3,… n}) is returned.
            b. **In order to use the other functions in this library, a VectorInt MUST first be created using this function, or the resulting object will not properly be bound/typed.**
                i. *Example Call:*
                    1. `var arr = Algo.Vector.Init();`
                    2. `var arr = Algo.Vector.Init(3);`
                ii. *Output:*
                    1. Returns an empty VectorInt object (in this case, "arr" now contains an empty VectorInt)
                    2. Returns a VectorInt object of size 3 populated with values {1,2,3}.
        2. *Algo.Vector.Shuffle(VectorInt):* shuffles the inputted vector with the Fisher-Yates Shuffle.
            a. *Example Call:*

```
var random = Algo.Vector.Shuffle(arr);
```

- **b.** *Output:*
  - **i.** If we assume that "arr" contains a VectorInt holding {1,2,3,4,5}, one possible output of the function is a VectorInt holding {1,4,3,2,5}.

**3.** *Algo.Vector.Reverse(VectorInt):* reverses the order of every element in the inputted vector.

- **a.** *Example Call:*
  - **i.** `var random = Algo.Vector.Reverse(arr);`
- **b.** *Output:*
  - **i.** If we assume that "arr" contains a VectorInt holding {1,2,3,4,5}, this function will output a VectorInt holding {5,4,3,2,1}.

**4.** *Algo.Vector.Dupe(VectorInt):* duplicates 20% of the contents of the vector across the entire vector, such that each item is repeated roughly vector.size()/5 times.

- **a.** *Example Call:*
  - **i.** `var random = Algo.Vector.Dupe(arr);`
- **b.** *Output:*
  - **i.** If we assume that "arr" contains a VectorInt holding {1,2,3,4,5,6,7,8,9,10}, this function will output a VectorInt holding {1,2,2,2,2,2,6,6,7,7} (or equivalent).

**ii.** ***Algo.Sort:***

**1.** *Algo.Sort.Bubble(VectorInt):* Runs the Bubble Sort algorithm on the inputted vector and returns the sorted result.

- **a.** *Example Call:*
  - **i.** `var bubble = Algo.Sort.Bubble(arr);`
- **b.** *Output:*
  - **i.** If we assume that "arr" contains a VectorInt holding {2,1,3,5,4}, this function will output a VectorInt holding {1,2,3,4,5}.

**2.** *Algo.Sort.Merge(VectorInt):* Runs the Bubble Sort algorithm on the inputted vector and returns the sorted result.

- **a.** *Example Call:*
  - **i.** `var merge = Algo.Sort.Merge(arr);`
- **b.** *Output:*
  - **i.** If we assume that "arr" contains a VectorInt holding {2,1,3,5,4}, this function will output a VectorInt holding {1,2,3,4,5}.

**3.** *Algo.Sort.Insertion(VectorInt):* Runs the Bubble Sort algorithm on the inputted vector and returns the sorted result.

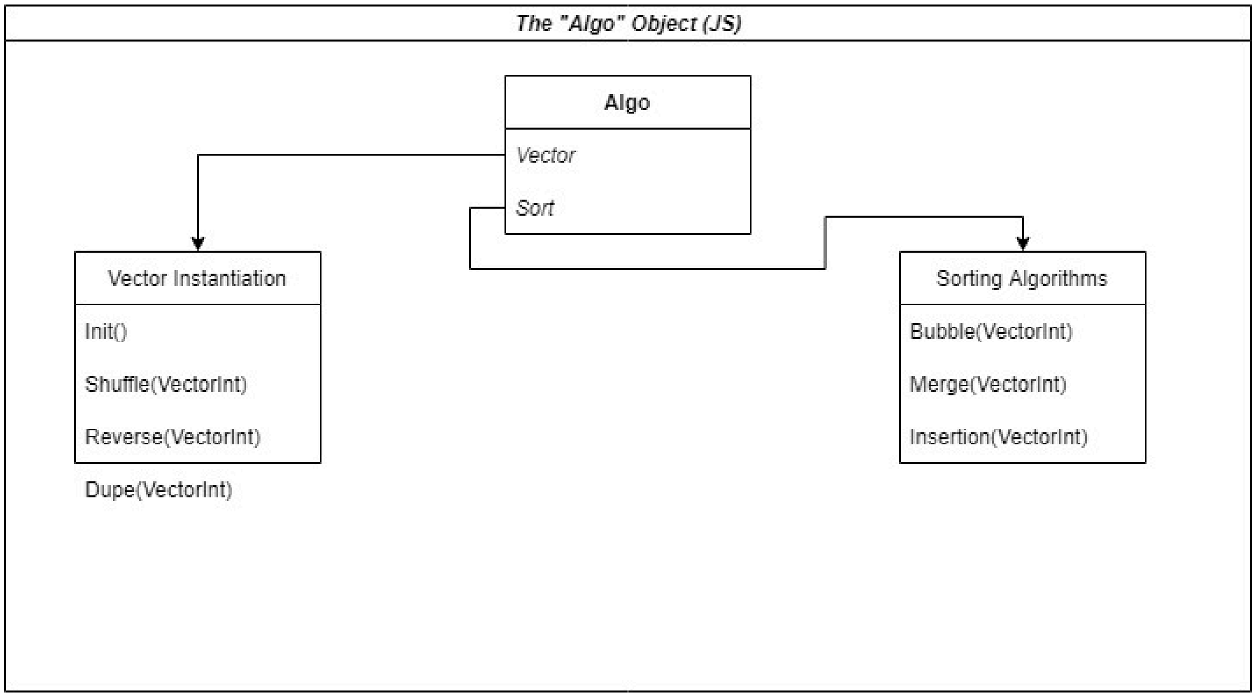
- **a.** *Example Call:*
  - **i.** `var insertion = Algo.Sort.Insertion(arr);`
- **b.** *Output:*

i. If we assume that “arr” contains a VectorInt holding {2,1,3,5,4}, this function will output a VectorInt holding {1,2,3,4,5}.

c. **Design Patterns:**
   i. **Strategy Design Pattern:** This library utilizes the Strategy Design pattern, allowing the main program, through the use of the “Algo” object, to seamlessly select the algorithm that is to be run at runtime (as opposed to hard-coding it). In C++, this “algorithm switching” (depending on desired input/output) is done through the use of function pointers that the Algo object keeps track of; while in the JS library, this same effect is achieved by allowing each function to be callable members fields of the Algo object.
      1. **Pros:**
         a. This strategy pattern allows the library to be easily expanded and extended (another emscripten-bound c++ sorting method could very easily extended to Javascript as a callable “Algo.Sort.myNewSortingFunction”), as all sorting algorithms (strategies) take the same input (a single VectorInt) and yield the same output (a single, sorted VectorInt).
         b. All algorithms are bound to the Algo object, so the only instantiations required once the library is loaded are the Algo object (which is automatically done in ‘library.js’) and the VectorInt object that will be sorted (which is also called through the Algo object).
      2. **Cons:**
         a. Like all Strategy patterns, the user must be aware of the entire definition of the Algo object to use it (to use a Strategy, the user must know it exists).

d. **Other Design Decisions:**
   i. **Use of Vectors:** Unlike arrays, which must be manually resized to change their size, vectors are able to be quickly imported to any language and are commonly used. Because the runtimes of all a vector’s factory operations are universally understood, it can very easily serve as a base for numeric operations. As vectors are also a templated type, the library, itself, can also be very easily templated (even for completely-user-defined classes, provided operations such as the equality operator are also defined for those classes).
   ii. **Default Value in GetVector(size):**
      1. If size is supplied, a default ascending numeric scheme is supplied as a test case for each sorting method- testing how an algorithm will act if *the given vector doesn’t need to be sorted.*

e. **Class Diagram/UML (JS):**



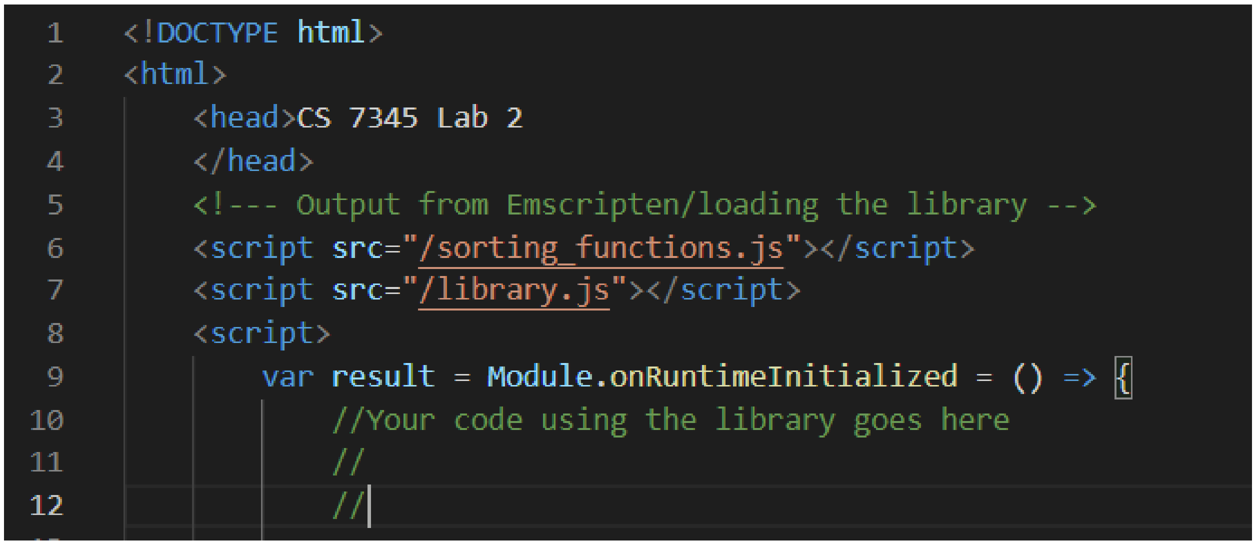
f. **Dependencies:**

   i. *sorting_functions.cpp:* The file containing the C++ functions that will be bound by Embind to callable attributes that can be accessed using Javascript.

   ii. *sorting_functions.js:* This file contains the generating bindings from running Embind on the "sorting_functions.cpp" file. It must be included for any calls to the Algo object. **This file must also be loaded before "library.js" is loaded, as it relies on the definitions found in this file.**

   iii. *library.js:* This file contains the definition for the Algo object and acts as a wrapper from the sorting function library contained in sorting_functions.js to. These files were split (as opposed to simply adding the definition of the Algo object to sorting_functions.js) so that compilation using emcc's binding commands do not erase the Algo object's definition.

   iv. **Note:** Because some browsers asynchronously load data**, all uses of the library must be made AFTER the module completely loads. This can be achieved by wrapping calls to the library in the Module.onRuntimeInitialized function, as seen below.**



```
<!DOCTYPE html>
<html>
    <head>CS 7345 Lab 2
    </head>
    <!--- Output from Emscripten/loading the library -->
    <script src="/sorting_functions.js"></script>
    <script src="/library.js"></script>
    <script>
        var result = Module.onRuntimeInitialized = () => {
            //Your code using the library goes here
            //
            //
```