

## lab 13 Fundamental sorting: Quick and Merge Sort

---

**Instructions:** Implement Quick and Merge sort for your Array List class. Then implement one of the fundamental  $O(n \log_2 n)$  sorting algorithm (Quick or Merge) for a (singly or doubly) Linked List.

```
1
2 template <class T>
3 class Array {
4     private:
5         /* You fill out the private contents. */
6
7     public:
8         ...
9
10        /* Runs a quick sort algorithm on the array.
11         * The array shall be ordered from least to greatest
12         */
13        void qsort();
14
15        /* Runs a merge sort algorithm on the array.
16         * The array shall be ordered from least to greatest
17         */
18        void msort();
19
20        /* Runs the sort routing you believe is the best. */
21        void sort();
22        ...
23 };
24
25
26 /* SLL = Singly Linked List */
27 template<class T>
28 class SLList {
29     ...
30     public:
31     ...
32         /* Sort the linked list. You may use any  $O(n \log n)$  sort algorithm you
33         * wish.
34         */
35         void sort();
36     ...
37 };
```

**Write some test cases:**

Create some test cases, using cxxtestgen, that you believe would cover all aspects of your code.

**Part 2: Performance**

Generate a graph to compare the performance of bubble sort, selection sort, insertion sort, and the sort you chose for a Singly Linked List. Your graph should have data size on the x axis and time on the y axis. Make sure to label each graph line! Please turn in as a .pdf!

**Memory Management:**

Now that are using new, we must ensure that there is a corresponding delete to free the memory. Ensure there are no memory leaks in your code! Please run Valgrind on your tests to ensure no memory leaks!

**How to turn in:**

Turn in via GitHub. Ensure the file(s) are in your directory and then:

- `$ git add <files>`
- `$ git commit`
- `$ git push`

**Due Date:** October 13, 2017 2359

**Teamwork:** No teamwork, your work must be your own.