# CSE250 Fall 2016
# Assignment A2 – Sorted Array

Due: 10/09/2016, 11:59PM

Last updated: *2016-09-26 12:53*

*Objectives*

- Implement first practical data structure.
- Practice working with C++ classes.
- Practice working with partially provided code.

*Introduction*

In many practical applications we want to store our objects in some sorted order. While there are specialized data structures designed specifically for that purpose, having an ordered collection of items with $O(1)$ access guarantee is really tempting. This is the challenge of the current assignment.

Your task is to extend partially implemented, and provided to you, `sorted_sc_array` class, such that it stores all elements in ascending order and automatically manages memory. Your implementation has to be efficient and you have to work under certain constraints. Specifically, you cannot use any external or standard C++ headers except of `<algorithm>`.

*Instructions*

1. Create directory `A2` where you will place your code.
2. Download `A2-handout.tar` from:
   http://www.jzola.org/courses/2016/Fall/CSE250/A/A2-handout.tar.
3. Untar handout, and move `a2.hpp` and `a2.cpp` to your `A2` directory.
4. Implement missing elements in `a2.hpp` taking into account the following:
   (a) `a2.hpp` is the only file you are allowed to modify. `a2.cpp` is a simple test program demonstrating functionality that your implementation of `sorted_sc_array` must support. You must use it to perform the most basic initial testing of your code. If your code fails to work with `a2.cpp` you will not be graded. You are not allowed to make any changes to `a2.cpp`.
   (b) Analyze carefully `a2.hpp` and `a2.cpp` to better understand `sorted_sc_array`. This class implements sorted array of `signed char`. You can safely assume that `signed char` has range $[-128, 127]$ – this might be helpful when designing really efficient implementation. The class already provides attributes to store pointer and size of the array, you should use them. It also provides methods `data()` and `size()`. These are primary access methods.
   (c) Missing parts of the code that you are expected to implement are marked with `IMPLEMENT ME`. In short, you have to implement copy constructor and assignment operator and

method `insert()`. Copy constructor and assignment operator must provide deep-copy functionality, and must ensure that there are no memory leaks.

(d) Method `insert()` is the workhorse of the class. Its purpose is to take a `signed char` and insert it into the array while ensuring that at the end the array is sorted in the ascending order. This is the only modifying method. **Each time new element is inserted to the array, once `insert` completes the array must be in sorted state.**

(e) **You can add, remove or modify code in `sorted_sc_array` without limits as long as it correctly compiles and executes with the provided `a2.cpp` and the only included header is `<algorithm>`**. To compile `a2.cpp` with your `a2.hpp` you should use basic compiler setting, e.g.: `g++ -std=c++11 -O2 a2.cpp -o a2`. If your code satisfies the most basic requirements for grading, you should see `pass` printed when executing `a2`.

(f) Your implementation must be efficient. You can expect millions of elements inserted and your class should be able to handle them in seconds.

(g) You should make no assumptions about the distribution of the inserted elements and the order in which elements will be inserted.

(h) You should be careful with how you utilize memory. If you are too excessive in allocating memory, your code will exhaust available resources and your code will be terminated and not graded.

*Submission*

1. Remove your binary code and other unrelated files (e.g. your test files).
2. Create a tarball with your `A2` folder.
3. Follow to https://autograder.cse.buffalo.edu and submit `A2.tar` for grading.
4. You have unlimited number of submissions, however, any submission after the deadline will have 50% points deducted.

*Grading*

- 10pt: The test `a2.cpp` compiles with your `a2.hpp`, runs and has no memory leaks.
- 90pt: If you pass the initial test, there will be nine benchmark tests. You will get 10pt for each correctly completed test (i.e. your implementation performs as expected, no memory leaks, and runs in reasonable time limits).
- If your code is **extremely** inefficient, autograder will terminate your code and you will receive 0pt.

*Remarks*

- Make sure that all file and directory names are exactly as instructed. Otherwise the grading system will miss your submission and you will get 0pt.
- Make sure that you start working on the assignment early! If you wait to the very last moment, you may expect delays from autograder (overloaded by other students who like you waited with their submission).
- Focus on finding the best performance algorithm and not low-level optimizations – the autograder architecture is unpredictable :-)