CSCI 2270, Fall 2014
LAB 4, DYNAMIC SORTED INTEGER ARRAY OF ITEMS
Due by Moodle Saturday, September 20th, 11:55 pm

Purposes: Practice building a container class that is based on a sorted array.

You will be given a header file, which does not need to change; the implementation file, which needs to be completed, and a small test file to get you started (but which is different from the complete version of the test file we'll use).

You are making a dynamically sized array, which has 4 variables, defined in the file dynamic_size_array_sorted.h:

```
struct int_array {
    int* data;
    unsigned int count;
    unsigned int capacity;
    static const unsigned int DEFAULT_CAPACITY = 20;
};
```

These include the same variables as in lab 3, but now you must maintain your integers in sorted order.

On your VM, please make a directory for lab 4 and copy ALL the files for the C++ integer array from the moodle site into that directory. As before, *you only need to change the dynamic_size_array_sorted.cpp file for this assignment*.

Most of your lab 3 code should work here with no changes; the only changes you must make are to **`add()`** and **`remove()`**, which now have to maintain a sorted order. You can paste your other files in from lab 3.

Function 1: **`add(int_array& arr, const int& payload)`**

Next, we have to edit **`add(int_array& arr, const int& payload)`** to get items into the array. For that, we should:

1. check right away whether the integer array is full; if so, we'll call the **`resize(int_array& arr)`** function to double our **`data`** array size before continuing.
2. Once the integer array has room, we should put the new item in the slot at the correct index in **`data`** and increase **`count`** by 1. I recommend using a loop to do this; it works very much like the inner loop in insertion sort from the reading.

Run the test code again and find out if your array is sorted when you add numbers to its front, middle, or end.

Function 2: `remove(int_array& arr, const int& target)`

The `remove(int_array& arr, const int& target)` code is the next part to change. This code removes one instance of the target from the array and adjusts the array to avoid gaps and stay in sorted order.

1. If the target is not in the array, this function does nothing except return `false`.
2. If the target is in the array, then removing an item involves finding its slot i in the `data` array, and then copying each item after this slot i to the slot just before it, so we overwrite the item and shift down to cover the gap. Finally, we decrease `count` by 1 to account for the removed item and return true.

The trick here is to run a loop that does the opposite of the loop you put into the new add.

Upload your dynamic_size_array_unsorted.cpp file to the moodle assignment link for Lab 4 and make sure it is really there before you call it done.

Note: depending on your background, this may seem easy, or it may seem impenetrably difficult. Start work early, and ask lots of questions if you have them; it helps. If you get stuck, take a break; remember not to just spin your wheels. I expect to see a fair number of you in office hours, LA help hours, and recitation this week. Don't be shy!