HW 3 is the continuation of your lab 2. You must upload **ALL** your code through Blackboard even if you have demo-ed part of your work to me during the lab in class (Please see the submission requirements).

Task 1 [50 pts] Sorted ArrayList

Please download the **HW/checkpointed_lab2_starting_files** as the starting files for this homework, although you are strongly recommended to finish your lab2 of implementing an unsorted ArrayList.

The ***ArrayList.cpp*** in **Checkpointed_lab2_starting_files** implements most of an unsorted arraylist functions, based on **labs/lab2_starting_files**, except the assignment operator overloading. Please implement a sorted array list by rewriting the insert/delete/retrieval functions inside ***ArrayList.cpp***, after you overload the assignment operator and finish ***data_management::updateRecord***.

- Feel free to reuse the code in the slides to complete your homework.
- The insert/retrieval/retrieval must follow the same interface and specifications as we discussed in the class and stated in the slides.
- You are recommended to implement the operator overloading first, and test it with some data to make sure it works before starting implementing sorted ArrayList.
- Besides implementing the ***data_management: updateRecord,*** Do you have to make any changes at the layer of data_management application when changing ***ArrayList.cpp*** from unsorted ArrayList to sorted ArrayList? Why or why not? Please state it at the last page. (see the grading sheet at the end of the document).
- At this point of time, the search should simply be implemented as linear search.

Task 2 [30 points] Search

Please reimplement the search of your ArrayList implementation using binary search, in addition to the linear search.

- Add an option to your console menu so that when users run a search, they can choose to do either **1) fast mode (based on binary search)** or **2) slow mode (based on linear search).**
- To facility your debugging, you should implement a profiling variable just like how we did with resize counter.
    - Add a member variable called "***stats_cmp***" to your sorted ArrayList to record how many comparisons have been done for the current run of the search.
    - ***stats_cmp*** is always reset to zero when a search starts.
    - ***stats_cmp*** is incremented when a compare in the search function is done.
    - ***stats_cmp*** is displayed when the search ends.
- To facility your experimenting, you should implement a timer to measure how much time is spent on each search. The timer should look like this:

```
clock_t startTime = clock();
list.RetrievalItem(…);
clock_t endTime = clock();
clock_t clockTicksTaken = endTime - startTime;
double timeInSeconds = clockTicksTaken / (long double) CLOCKS_PER_SEC;
```

- You should do a wide range of experiments based on searching "***Melissa Dunn***", a non-existent key word, to compare the performance of linear search vs. that of binary search and fill up the following table.

# of records	stats_cmp(Binary Search)	run time (Binary Search)	stats_cmp(Linear Search)	run time (Linear Search)
600				
1,250				
2,500				
5,000				
10,000				

Grading Criteria:

1) If your code does not compile, or compiles with any warnings, you will only receive up to 30% of full points.

2) Please see the grading sheet for the breakdowns of grades.

3) You will lose another 10% on your earned points if any memory-leak bug or dangling-pointer bug is found.

Submission Requirements:

1) At the end of this document, there is a grading sheet. You must fill-up the **ALL** fields of self-evaluation part of the table, as well as the experimental table and the short answer question.

- You must at least fill in comment "A" or "P" or "N" to indicate that you have completed all, part of, or nothing of the task respectively.
- You are encouraged to write down more information about your implementation, including but not limited to:
    - additional tests you have done.
    - corner cases your code might not handle.
    - better solution you come up with.
    - assumption you have made for your code.
- Even if you complete nothing on a task, if you mark "N", you will receive 15% for the question. If you do not answer, you will receive zero, and I will not review your solution for that task.

2) You must submit the grading sheet with your code, or you will receive 0 (see 3)

3) Please save the grading sheet to the same folder where your project folder is. Please keep the directory hierarchy of the project and then compress the project folder into a zip file and upload it through Blackboard.

This page should be filled and submitted with your project.

Task 1: Sorted ArrayList(10 points each)	**Self-Evaluation**	**Score (used by the instructor)**
ArrayList: Assignment Operator Overloading		
ArrayList: Insert		
ArrayList: Delete		
ArrayList: retrieval		
data_management::updateRecord		
Task 2 (70 points)		
Binary Search – console (5pts)		
Binary Search – function (25pts)		
Profiling—stat_cmp (10 pts)		
Timer (10 pts)		
Experiments(20 pts)		

# of records	stats_cmp(Binary Search)	run time (Binary Search)	stats_cmp(Linear Search)	run time (Linear Search)
600				
1,250				
2,500				
5,000				
10,000				

[short answer question]
Except implementing the ***data_management: updateRecord,*** Do you have to make any changes at the layer of data_management application when changing ***ArrayList.cpp*** from unsorted ArrayList to sorted ArrayList? Why or why not? Please state it here (5 points).

_____