# CSCE 221 Cover Page
# Programming Assignment #3

First Name: Last Name: UIN:

**Any assignment turned in without a fully completed coverpage will receive ZERO POINTS.**
Please list all below all sources (people, books, webpages, etc) consulted regarding this assignment:

| CSCE 221 Students | Other People | Printed Material | Web Material (URL) | Other |
|---|---|---|---|---|
| 1. | 1. | 1. | 1. | 1. |
| 2. | 2. | 2. | 2. | 2. |
| 3. | 3. | 3. | 3. | 3. |
| 4. | 4. | 4. | 4. | 4. |
| 5. | 5. | 5. | 5. | 5. |

Recall that University Regulations, Section 42, define scholastic dishonesty to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion. Please consult the Aggie Honor System Office for additional information regarding academic misconduct – it is your responsibility to understand what constitutes academic misconduct and to ensure that you do not commit it.

I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received nor given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment.

Today's Date:

Printed Name (in lieu of a signature):

# Priority Queues and Heaps

Due: March 21, 2018 11:59pm

**Description**:
For this programming assignment, you will implement a Priority Queue. Your Priority Queue can grow dynamically as in the Stack assignment (although you should use a method with amortized cost O(1) for insertion) or you may make a Priority Queue large enough to fit all of your data ahead of time. You will build three different implementations of the Priority Queue. Two of these implementations will be based on a List. The first list implementation, UnsortedPQ, should use an unsorted list to store the data. The second list implementation, SortedPQ, should use a sorted list. The third implementation, HeapPQ, should be an array-based implementation of a Heap.

We will test your implementations using a set of numbers in a text file named "numbers.txt". Each line of the file will contain a number. The first number will be which Priority Queue to use (0=UnsortedPQ, 1=SortedPQ, 2=HeapPQ). The second number will be the number of remaining elements in the file, *n*. The first two numbers of the file (queue type and *n*) will NOT be inserted into your Priority Queue. However, the remaining *n* numbers will be inserted. You should then remove all *n* numbers and print them to the screen. Here is an example numbers.txt file.

Using your three implementations, you will also time how long it takes to insert n random numbers and the total time to insert n random numbers and remove n random numbers. You may use a random number generating as opposed to the file input above. Time how long insertion takes using the StopWatch class (or the class on Piazza that is not Visual Studio-specific) provided. You should measure the total time at fixed intervals. You will then show a graph of the total insertion time versus the number of items inserted for each implementation. You will then time how long insertion and removal takes by inserting n numbers and removing n numbers for each implementation. You will then graph the sorting time versus the number of elements. This process will allow you to see how your choice of implementation can affect performance.

**Coding Portion** (50 Points):

- Start with the following template: PriorityQueue.h, and create three versions of the Priority Queue (using templates for the type of object) filling in all of the member functions.
- Be sure to test the correctness of your algorithms and implementations (we will).
- Your code will be graded based on whether or not it compiles, runs, produces the expected output, produces correct output, whether or not your experimental setup is correct, and your coding style (does the code follow proper indentation/style and comments).

**Report** (50 Points):
You will write a brief report that includes theoretical analysis, a description of your experiments, and discussion of your results. At a minimum, your report should include the following sections:

1. *Introduction*. In this section, you should describe the objective of this assignment.
2. *Theoretical Analysis*. In this section, you should provide an analysis of the complexity of an insert operation and the complexity of the sort (inserting all of the items and then removing all of the items). Describe the advantages and disadvantages of the three strategies. What is the complexity of an insert (on average) for the different implementations? What is the complexity of the sort (on average) for the different implementations?
3. *Experimental Setup*. In this section, you should provide a description of your experimental setup, which includes but is not limited to
    a. Machine specification

b. How did you generate the test inputs? What input sizes did you test? Why? What data structures did you use for your List implementation of the Priority Queue?
c. How many times did you repeat each experiment?

4. *Experimental Results*. In this section, you should compare the performance (running time) of the insert() operation and the sort in the three different implementations to one another and to their theoretical complexity.
    a. Make a plot showing the running time (y-axis) vs. the number of insert operations (x-axis). You must use some electronic tool (matlab, gnuplot, excel, …) to create the plot – hand-written plots will NOT be accepted.
    b. Make a plot showing the running time (y-axis) vs. the number of items inserted and removed (x-axis). You must use some electronic tool (matlab, gnuplot, excel, …) to create the plot – hand-written plots will NOT be accepted.
    c. Provide a discussion of your results, which includes but is not limited to:
        i. Which of the three Priority Queue implementations performs the best? Does it depend on the input?
        ii. To what extent does the theoretical analysis agree with the experimental results? Attempt to understand and explain any discrepancies you note.