# Programming Assignment #3
**Incentive Date: Mar 10th 2024**
**Due Date: Mar 20th 2024**
**Priority Queues & Heaps**

**Description:**
For this programming assignment, you will implement a **Priority Queue** based on a **min heap** in three different ways. You will **build three different implementations** of this Priority Queue. The first implementation will use an Unsorted Array (UnsortedPriorityQueue.h), the second implementation will use a Sorted array (SortedPriorityQueue.h), and the final implementation will be based on an array representation of a binary heap (PriorityQueueHeap.h).

**[Advisory]:**
- **As you have been tasked with creating the PriorityQueueHeap class in LE5, you may simply copy the implemented code for this class from LE5 to PA3. Note that the LE5 PriorityQueueHeap class took in a comparison object function to allow for both min and max heaps. In this PA, we will only be implementing a min heap so you may have to make slight changes during comparison of the parent and the child for the PriorityQueueHeap class**
  - **You will have to directly compare for a min heap rather than relying on the compare function.**
- **You can resize (by doubling the size) the array dynamically for all three implementations when the number of elements in the array is equal to the capacity of the array.**

**starter.zip (Starter Code):**
The following is a brief description of the starter code provided to you:
- **AbstractPriorityQueue.h**
  - Declares the common priority queue methods such as **pq_insert**, **pq_delete**, **pq_size**, **pq_get_min** as pure virtual which will be overridden by the three child classes.
  - **No changes to this file is necessary**
- **PriorityQueueHeap.h**
  - Implements the min-heap based priority queue using an array based representation of a binary heap.
- **UnsortedPriorityQueue.h & SortedPriorityQueue.h**
  - The UnsortedPriorityQueue class uses an unsorted array to implement the priority queue methods for a min heap.
  - The SortedPriorityQueue class uses a sorted array to implement the priority queue methods for a min heap. At each insert and delete operation, the array must remain sorted.
- **main.cpp**
  - You can use this file to instantiate PriorityQueueHeap, UnsortedPriorityQueue, and SortedPriorityQueue objects and test your code.

**Contract (TASK):**

Your task is to implement the methods declared in the PriorityQueueHeap, UnsortedPriorityQueue, and SortedPriorityQueue classes. For each of these classes, you have been provided with non-functional definitions of these methods in their respective header files. **You must implement these method definitions in the given header files**, by replacing the current non-functional definitions with your functional implementations. **Do not create additional cpp files to define these functions, make sure that they are part of the header files.**

**[Advisory]:**

- You may choose **10** to be the initial capacity of the array when allocating memory in the constructor
- You must throw a **std::out_of_range** exception when you come across invalid calls to the pq_get_min and pq_delete methods

**Deliverables and Submissions:**

In a zip folder named using the format **<FirstName>-<LastName>-<UIN>-<PA3>.zip** you must submit the following file(s) to Canvas:

- **PriorityQueueHeap.h** (containing the functional method definitions of the PriorityQueueHeap class)
- **UnsortedPriorityQueue.h** (containing the functional method definitions of the UnsortedPriorityQueue class)
- **SortedPriorityQueue.h** (containing the functional method definitions of the SortedPriorityQueue class)
- **PA3_Report.pdf** (containing your report - details mentioned below)

**Do not use angular brackets in the name of the submission folder.**

**Grading:**
**This assignment is worth 80 points.** The 80 points will be split up between Coding (45 points) and a Report (35 points).

**Coding (45 points)**
Each method of the three classes will be tested using an automatic testing infrastructure. You will be provided with the testing infrastructure which you can use to evaluate your implementations.

The grading rubric is as follows:

- Each of the following methods will be tested out for each of the three classes:
    - pq_insert: 7 points
    - pq_delete: 7 points
    - pq_delete and pq_get_min exceptions: 1 point

In case of any memory leaks in your program, we will subtract 5 points.

**Report (35 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 (5 points)***: In this section, you should describe the objective of this assignment.
2. ***Theoretical Analysis (5 points)***: 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 (10 points)***: In this section, you should provide a description of your experimental setup, which includes but is not limited to
   - i. How did you generate the test inputs?
   - ii. What input sizes did you test? Why? **(You must test the runtime on at least a 100,000 elements as the largest data point)**
4. ***Experimental Results (15 points)***: 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.
   - iii. 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.
   - iv. 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.
   - v. Provide a discussion of your results, which includes but is not limited to:
     1. Which of the three Priority Queue implementations performs the best? Does it depend on the input?
     2. To what extent does the theoretical analysis agree with the experimental results? Attempt to understand and explain any discrepancies you note.

**testInfrastructure.zip (Test Methodology):**
In order to familiarize you with the auto-grading environment on our end we are providing you with an automated test infrastructure that can be used to check the correctness of your code. The testing infrastructure is constructed as follows:

- **pq_test.cpp:** This cpp file defines all the test functions that have been mentioned in the grading rubric above. It also runs these tests and outputs your final score.
- **test.py:** This file compiles pq_test.cpp in order to produce the executable. It then runs this executable which will indicate your score (as well as the test cases you have passed or failed). It will also run valgrind on the executable file to indicate the presence of memory leaks if they exist.

**Note: There is a possibility that one or more of your methods causes a segmentation fault which in turn causes the test script to crash when calling that particular method. In such a case, we will not be able to calculate your score for the assignment and you will be assigned a default grade of 0. After the grades are posted, you will be given 1 week to rectify your error(s) and resubmit the assignment. You will be subject to a 25% penalty on the grade obtained from the resulting resubmission of the assignment.**

**How to test your program natively (run your own test cases):**
You can use main.cpp provided in the starter.zip folder to instantiate PriorityQueueHeap, UnsortedPriorityQueue, SortedPriorityQueue objects. You can then test the functionality of your methods by calling these methods and checking whether what they return is what you expect.
To compile your own tests in main.cpp, **you can use the makefile provided within starter.zip to compile your program.** In your terminal:

- Run **make** on the terminal (this will create an executable named main)
- Next, run **./main** on your terminal to run your native tests
- Finally run **valgrind --leak-check=full ./main** to check whether there are memory leaks.

**How to use testInfrastructure:**
To check your score using testInfrastructure, you can follow either of the two options mentioned below:

- **Option 1:** Move your PriorityQueueHeap.h, UnsortedPriorityQueue.h, SortedPriorityQueue.h files into the testInfrastructure directory. After this, run **python3 test.py**
- **Option 2:** Move your PriorityQueueHeap.h, UnsortedPriorityQueue.h, SortedPriorityQueue.h files into the testInfrastructure directory. **Ensure that the makefile in this directory is the one given to you originally in the testInfrastructure.zip folder and not in the starter.zip folder.** In your terminal:
  - Run **make** in order to compile the program (which will compile pq_test.cpp and create an executable named pq_test).
  - Next, run **./pq_test** on your terminal to actually run the tests and output your score.
  - Finally run **valgrind --leak-check=full ./pq_test** to check whether there are memory leaks.

**Please also make sure that you run the test script a few times (3 - 4 times should be sufficient) to ensure that your code is consistently passing test cases in different random runs of the test script.**

## (Optional) PA3 Extension Assignment:

**This optional extension is not for a grade. However, you are encouraged to engage with it in order to learn how a priority queue is used in the real world. This may come in handy for the SA.**

From this PA onward, we will be introducing an "application" component of the data structure being implemented. For this particular extension in this PA, we will be exploring how priority queues are used for scheduling in CPUs.

Processes are assigned priorities based on estimated execution times **(lower execution times have higher priorities)**. The algorithm selects and executes the process with the highest priority first, optimizing turnaround times and resource utilization. The processes may be initiated dynamically, and the CPU must make real-time decisions on which process to schedule when it is free.

**Note that you will need your PriorityQueueHeap class from LE5 to implement this assignment. You can also alternatively use the STL priority queue.**

In the PA3-Extension.zip file you've been given a starter file called CPU_Scheduler.cpp. The CPU_Scheduler.cpp file contains the following:

- **Process** class
    - The process class contains an id and a defined processing time
- **CPU_Scheduler** class
    - This process contains a private PriorityQueueHeap member object that the CPU will use to schedule processes.
    - The CPU_Scheduler class exposes a public facing method: **void scheduleProcess(const Process& process)** that can be called to schedule a process anytime during the CPU's execution. The CPU must determine when to schedule the process based on the aforementioned priority on lower processing times.
- **main()**
    - The main() function of the cpp file will instantiate a thread that runs the CPU Scheduler (using the runScheduler() method). This will allow us to concurrently instantiate and schedule new Processes at different points in time during the CPU Scheduler's execution.

**Task:** Your task is to use the priority queue within the CPU Scheduler class to optimally schedule processes in real time. To do this, you must define the comparison function of the Process class to allow the priority queue to function correctly, as well as implement the logic in the runScheduler() and scheduleProcess() methods in the CPU_Scheduler class. You can test your implementation using the sample set of processes instantiated at various points in main().