1. **(2 pts) Program description; purpose of the assignment.**

Throughout this assignment we constructed multiple different types of minimal priority queues using different data structures. We used binary heap, vector, and linked lists to construct the MPQs. After constructing the MPQs we use a CPU_Job class to mimic a processor. The files were read in from a text file and then outputted to another text file. The purpose of the assignment was to better our understanding of different data structures and the minimal priority queue and how to write fast algorithms that best adhere to design principles (such as not running super long).

2. **(2 pts) Instructions to compile and run your program; input and output specifications.**

To compile the program I used visual studio for most of the implementations except the linked list in which I used the TAMU servers, because the linked list function was running for over 45 minutes on my computer but only for around 10-15 minutes on the server. Through the header files for each data structure I included a main function that was commented out that has some test cases I used to test the data structure. In the actual main.cpp I included all 3 main functions (commented out) I used for the CPU_Job phase. To run the different main functions just remove the comments and you can enter the text file name to input the different text files. The output for the CPU_Job is to a text file named OutSize

3. **(5 pts) Programming style, and program organization and design: naming, indentation, whitespace, comments, declaration, variables and constants, expressions and operators, error handling and reporting, files organization, operators overloading. templates. Please refer to the PPP-style document.**

I used templates for all the functions so that you can enter any sort of data type, as specified in the instructions. Also I wrote all the different data structures in their own header files and included them in the main.cpp. For each header file I had a saved main function in the header file that consists of the test cases I used. Furthermore, I included comments on the Big-O of each function inside the program. I organized it into different header files so the code would be well structured and there would not be crowding. Furthermore, it would be easier to grade if I included my test cases inside each header file. I kept the comments to a minimum as not to crowd the program as the program is fairly simple and doesn't need to be overcrowded with unnecessary information. I kept an adequate amount of white space in each file as to offer easy visibility of different functions. There is no error handling in the program. The main.cpp file includes all the header files and different main functions corresponding to each data structure, depending on the data structure I wanted to test I just commented out the other ones and ran the one I needed to test.

4. **(5 pts) Discussion of the implementation and running time in the terms of big-O.**

For the vector data structure I had 3 templated functions is_emptyy(), remove_min(), and insert(). The is_emptyy() returns a bool value back and has a constant Big-O of O(1). The remove_min() function searches the vector for the smallest element notes the index of the element, removes the element from the vector, and returns the element, since the for loop runs for a N amount of times the Big-O notation is O(n). The remove_min() functions runs for a constant time of O(1), since it only pushes to the back of the vector. For the linked list data structure, I followed a similar style as the vector classes and had 3 same templated functions. The is_emptyy() and insert() both have a Big-O of O(1) since they contain 1 instruction each that runs for a constant time. The is_emptyy() has 1 compare statement and returns a bool and the insert() pushes to the front of the linked list. The remove_min() function uses a for loop with two iterators, one to run through the linked list and one to save the location of the lowest value. The first iterator is compared to a temporary value that is initially set to the beginning of the list, if the temporary value is greater than the temp value is changed to the iterator value and the second iterator value is updated to know which location the lowest value exists at. The Big-O notation is O(n) since it iterates through the entire list n amount of times. The binary heap data structures again has 3 functions but the implementation of the is_emptyy() function is the same except the function name is is_empty(). The reason for that is because my IDE was telling me is_empty() was too ambiguous for the vector and linked list structures but not for the binary heap. The binary heap class consists of 7 functions total and 1 default constructor. 3 of the functions are used to access the left, right and parent element of different elements all have a constant time O(1). I used a vector to implement the binary heap. Furthermore, one of the functions shows the heap and is called show_heap() which has a O(n) complexity and displays the heap. The insert() function pushes a element to the back of the vector and then organizes the binary heap to maintain a minimal binary heap, the while loop compares the child to its parent and swaps places if necessary, the Big-O notation is O(logn), since it iterates by n/2 (since the parent is (n-1)/2). The remove_min() function runs in O(n) time and the implementation is similiar to the previous data structures, where a for loop runs through and compares each element to the parent and switches them if necessary. In the CPU_job header file I implemented the operators <, > for the CPU_job class and both have a constant run time of O(1).

5. **(6 pts) Presenting the testing results, use a table to present timings for each implementation (9 timings in total).**

Inputs	Vector	Linked List	Binary Heap
1000	178 ms	100 ms	135 ms
10,000	10304 ms	8710 ms	6,687 ms
100,000	968973 ms	932,750 ms	605,025 ms

The vector and binary heap timings were done on my computer, and the linked list timing was done on the TAMU servers, because my computer was running into problems with large input sizes for the linked list. The TAMU server timings for the vector and binary heap would be faster if I did run them on the server, but due to time constraints I was unable to test everything on the server.