COSC 2436 Homework #2

Overview:

You are asked to Create a C++ program to read a large number of random **long integers** from a file and then determine the quartile values **without sorting the numbers**. Your program should read in the number of values and then load the values into a dynamically allocated array. Once all the values have been read, it should find the first-quartile, the median (middle) value and the third-quartile in the list. Your program should write the quartile and median values into the output file, one per line. **Your program will be evaluated for its runtime.**

Input files

- The input files contain a line with the number of values in the file followed by one value per line.

- The first line is a positive integer that indicates the number of values the file contains. You will use this to dynamically allocate the array to hold the numbers in the file. Don't forget to release the memory when you are done using it.

- The subsequent lines contain **long integer** values, one per. You should read and store these values into the array.

Output files

- Output the first-quartile, median or middle value, and third-quartile in the array.

- For debugging, you may wish to output the time required to extract the values and consider the time complexity of your solution.

Example

input1.txt	**output1.txt**
10 5777523539921853504 3549292889148046380 3599698824668116388 5357772020071635406 9005878083635208240 4989597678449481749 994344248487496566 5995831931848466816 1862423402299864599 3311781361602019704	3311781361602019704 3599698824668116388 5777523539921853504

To determine the quartile values, we consider a sorted array and pick out the middle value, the value in the middle of the bottom half, and the value in the middle of the top half. The first quartile is the number that would be in the sorted array at the index $i_1 = \left\lfloor \frac{n-1}{4} \right\rfloor$ where n is the number of elements in the array. The median would be at index $i_2 = \left\lfloor \frac{n-1}{2} \right\rfloor$ in the sorted array. The third quartile is the number that would be at index $i_3 = \left\lfloor \frac{3n}{4} \right\rfloor$ in the sorted array.

This example, input1.txt, should execute in about 0.5ms, input2.txt should require about 50ms and input3.txt about 500ms, depending on the system and load.

While it would be tempting to sort the entire array, it is not necessary to do so and takes more time than finding these quartile values. We would like to be clever (lazy) in solving this problem. We know it is possible to find the largest number or smallest number in an unsorted array in **O(n)** time. If we had to do k such numbers, that is **O(*k*n)**, so for most ***k*** values this is simply **O(n)**. Unfortunately, the median is the middle number, so we would need **n/2** values, making this an **O(n^2)** problem. We know we could sort the numbers in **O(log n)** time, but we would like a faster way to find the solution. In fact, we can find the values in **O(n)** time.

To solve this problem, we realize that we can quickly partition the numbers using the same method provided by ***Quicksort***. The partition step provides a way to place a single value, the pivot, into its sorted position and then move the other elements into their correct set relative to this pivot. There are several options for selecting the pivot, but **for consistency we will always select the right-most value possible: the element at the right edge of the current subset.**

There are also several options for reconstructing the partitions, once the pivot is selected. Different implementations will provide the same quartile and median values but different numbers of partition steps, which could make debugging a challenge. We would like to use a partition based on an increasing-order sort so that the set left of the pivot is less than the pivot and the set to the right of the pivot is greater than the pivot. In addition, when we re-partition the values, we will move from the left-most index up to the pivot position and from the right most index down to the pivot to perform any swaps needed. We then swap the pivot into its final position.

Reminder

- Turn in your lab assignment to our Linux server, follow the link here for more instructions.
- Make sure to only have one (1) .cpp file with the main() function in your working directory, otherwise your program will fail the grading script.
- Create a folder under your root directory, name the folder hw2 (case sensitive), copy your .cpp and .h files (and only .h and .cpp files) to the folder (ArgumentManager.h is also needed).
- Only include the necessary files (.cpp and .h files) in your working directory in your final submission.
- To test your program, you may wish to copy the input files and answer files onto the server and run your program. Do not include any output files and after verifying that the code passes the tests, **delete any output*.txt files**.

Please reach out to me or the TAs for any clarifications or typos.