Homework #2

Overview:

You are asked to Create a C++ program to generate a large number of random **long integers** as specified by the user and then determine the **median** of those numbers **without sorting them**. Your program should read in the seed for the random number generator, and a count of the total number of values to randomly generate. It should generate and store the random values in a dynamically allocated array. Once all the values have been generated, it should find the median (middle) value in the list. For an even number of entries, we will round down the index number, rather than average the two middle entries. Your program should write the median value in the output file with an end-line. **Your program will be evaluated for its runtime.**

Input files

- The input files contain two lines that specify the parameters for the execution of your program.

- The first line provides an integer that you will use to seed the random number generator (using the command srand()). This will ensure that you generate the same "random" sequence of numbers for testing.

- The second line provides, as an integer, the number of values you should randomly generate. Using this value, you should dynamically allocate an array of **long integers**. Don't forget to release the memory when you are done using it.

Output files

- Output the median or middle value in the array. For even numbers, we will round the size down, rather than averaging the two middle values.

- On the next line, output the number of partition steps required to find this value.

- 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**
123 10	4502939041817604120 1

This example should execute in about 0.2ms, input2.txt should require about 20ms and input3.txt about 200ms, depending on the system.

While it would be tempting to sort the entire array, it is not necessary to do so and takes more time than finding the median value. 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 would like a faster way to find the median value.

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.

In the above example, the median value (4502939041817604120) was found in just one (1) partition step. 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 in place. Different implementations will provide the same median values but different numbers of partition steps. 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 and from the right most index down to the pivot to perform any swaps needed. We then swap the pivot into position.

5. 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 all your .cpp and .h 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 outputs files and after verifying that the code passes the tests, **delete any output*.txt files**.

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