CIS 303 Analysis of Algorithms
Fall 2020

Trevor Butler
Assignment 01

09/11/2020

Problem

Using the given implementation of *KthLargest.java*, along with *TestKthLargest.java* and *Time-TestKthLargest.java* derive a solution that will find the Kth largest element in an array. For example if the array is of size ten and already sorted, if **k** were equal 5 the answer would be 4 (**k**-1). Data should also be gathered regarding the amount of time it takes to find **k** in the array.*TimeTestKthLargest.java* will be useful to determine this as a *data.text* file should be produced with these values listed.

Hypothesis

Time is a major concern when developing a program which is why it always need to be looked at and addressed before development. While analyzing this problem of finding the Kth largest value of array size n, time is on the table. When a searching through array of size n, the larger the value of n, the longer the run time will be (n^x = run time increases). The same concept should apply to a larger value of k as it will need to cycle through the array more times to determine the value . This will add run time in either milliseconds or nanoseconds which does not seem like a large amount until this and other time-consuming tasks are done repeatedly.

Methods

1. Your implementation (code).

```
while (iterator < k)
{
    for (int x = 0; x < a.length; x++)
    {
        if (a[x] > checkValue)
        {
            for (int y = 0; x < k; x++)
            {
                if (a[x] != sorted[y])
                {
                    checkValue = a[x];
                }
            }
        }
    }

    //set values in array
    sorted[iterator] = checkValue;
    iterator++;
}
```

2. General approach.

   The initial approach to this problem was to create a new array of size k where the old array elements would be evaluated to see if they were smaller or larger than the highest element. Once that was determined, the values would be stored in the new array of size k.

3. Data structures.

   The code that was implemented dealt strongly with arrays and comparing values. For these experiments it seemed best to use these in order to keep elements in check as well as easy to access. Being that the main array could not be sorted itself, a second array of size k was introduced to store larger values.

4. Experiment details.

   (a) Size of N: n = 10^5

   (b) Case(s) tested (best, worst, random, etc.): Random (array filled with random values)

   (c) Number of iterations/copies of the experiment: 101 (Starts at 1 then iterates every 1000)

   (d) Metrics: Run time in milliseconds

Results and Discussion

After implementing the code and running a test there was a lot of data to take it. Testing for the Kth element took place as well as determining how long each Kth element took to find, this was recorded in milliseconds. Figure 1 below shows the results as K increased and how it affected the time. The test was ran from 1 to 100000, with an interval of a thousand (1, 1000, 2000, 3000, etc.). Despite assuming that there would be an exponential growth as K increase, the graph showed otherwise. As K increased so did the time, which was expected but not in the sporadic way shown as the graph bounced up and down throughout nearly the entire graph. After K was about 81,000 the graph/data seemed to even out and become less sporadic with times staying between 5100-7100ms.

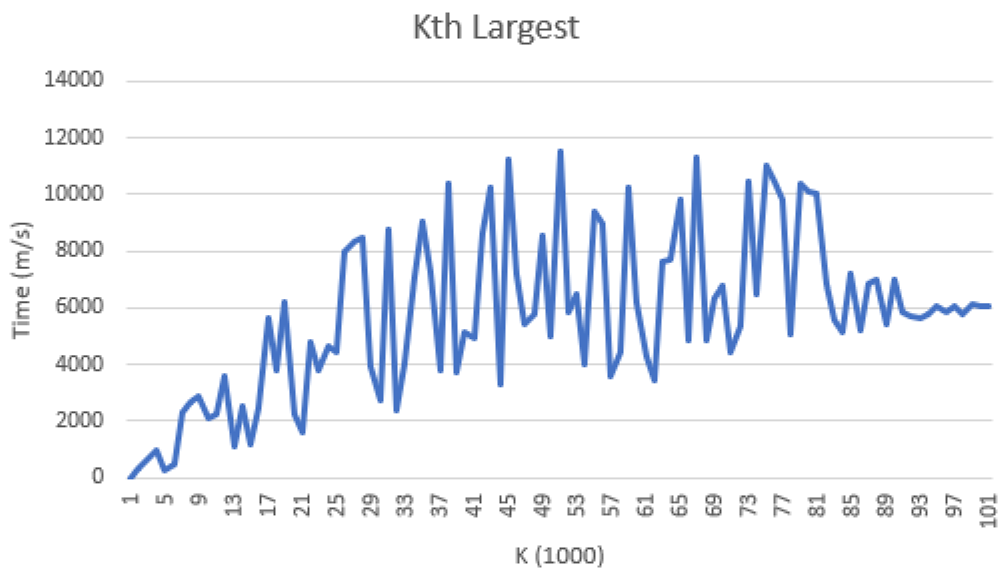

Figure 1: Kth Largest Graph

Conclusion

Although my hypothesis seemed logical and showed similarities to the results, the overall results did not support my hypothesis. The graph shown is not exponential but time does increase as K becomes larger. If I were to do this experiment again I would like to see a few tests among a few different variables. I would like to see how the results vary on different machines with different components (AMD/Intel CPUs), I would also like to see the results with different K values. When testing I also tried 50000 which never broke above 3000ms throughout the entire process. With these new variables, it would be intriguing to see how/if the data varied.