**CS201 Homework 5**

**Alp Bölükbaşı – S015662**

The main objective of the homework is, conducting an experiment by drawing a comparison between 4 sorting algorithms’ runtimes.

**Experimental Setup**

The sample sizes that used in this experiment are:

N= 1000, 10000, 100000, 500000, 1000000

The algorithms that used in this experiment are:

* AlgorithmSortAll
  + It does a simple insertion sorting operation to the given input. It’s the slowest algorithm among all other algorithms. In worst case (time complexity), it works in O(N^2).
* AlgorithmSortK
  + It also does an insertion sorting operation by kth input and compares kth with next inputs. In worst case, it works in O(N^2).
* AlgorithmSortHeap
  + It sorts by a Binary minheap tree-based structure and does a sorting operation by referencing the minimum element on the binary tree. Repeats the same process for each remaining element. In worst case, it works in O(NlogN).
* AlgorithmSortQuick
  + It picks an element as “pivot” (usually pivot is median) and does many partitions around the pivot. This is the fastest known generic sorting algorithm. In worst case, it works in O(N^2) with a very low probability. Algorithm time is usually O(N).

**Results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Algorithm runtimes (seconds) | | | |
| Sample Size (N) | sortAll | sortK | sortHeap | sortQuick |
| 1000 | 0.003 | 0.003 | 0.003 | 0.002 |
| 10000 | 0.082 | 0.062 | 0.029 | 0.028 |
| 100000 | 5.605 | 3.502 | 0.306 | 0.29 |
| 500000 | 144.185 | 88.843 | 1.59 | 1.462 |
| 1000000 | 547.617 | 349.048 | 3.149 | 2.851 |

Each result was generated and re-ran a few times manually. The observed results that are stated above are the average measurements. All results have measured in my laptop with Intel i7 7th Gen and licensed Windows 10.

It is crystal clear that, sortAll is way slower than sortK. sortHeap and sortQuick is way quicker than other two. To make a better observation, we must make a comparison in between sortQuick and sortHeap. *(numbers on x-axis represents sample sizes: 1000, 10000, 100000, 500000, 1000000.)*

sortQuick is faster but their runtimes are close to each other.

**Discussion**

My interpretation before the experiment was almost correct and I was foreknown about the correct runtime order.

What I’ve mistaken about experiment was, even though these four algorithms’ runtimes are different, I’ve encountered with almost same speeds when N (=sample size) is small. I was surprised about that. As a result, we can observe and interpret that, insertion sort can be efficient in small sample sizes.

Even though both AlgorithmSortAll and AlgorithmSortK have same time complexity case rate: O(N^2), AlgorithmSortK is way faster. Because AlgorithmSortK does insertion operations only around kth index. So that means, AlgorithmSortK does not encounter with unnecessary data and thus, not processes them.

When we look at the second chart, we can see that AlgorithmSortHeap and AlgorithmSortQuick have very similar runtimes. We must look at bigger N values in order to analyze the exact relation. Quicksorting is faster because it is more cache-efficient than Heapsort. In Quicksort, the inner loops have a smaller body. It almost doesn't do unnecessary element swaps (swaps are time consuming). But in Heapsort, even if your all data is ordered, you “absolutely” are going to swap elements in order to fill the array. Secondly, it’s faster in terms of their O-Notations. Yes, Quicksort’s worst case, O(N^2) but it’s a very low probability to come across with. In my implementation, the median element selected as “pivot” and thus, it works in O(N). So, we did not encounter the worst case. But we can clearly say that there’s a colossal difference between first two algorithms: SortAll, SortK and last two algorithms: SortHeap, SortQuick.