Priority Queue Sort Experiment

Emil Lynegaard

September 24, 2017

1 Experiment

For this experiment, the programming language Go will be used. The experiment will set out to compare sorting of a list of numbers with a priority queue and with a call to a library sort function. Since Go does not ship with a priority queue directly, an implementation thereof from their documentation of their heap package[1] will be used. The sort function that will be used is Go's `sort.Ints`[2].

Measurement: For the comparison, the experiment will be ran 100 times for both the priority queue and for the library sort function. The input will be of size `5.000.000` and the two experiments will be ran independently on the same machine with minimal process interference.

Assumption: Due to the nature of a priority queue, it has additional housekeeping to do when we inserting and deleting items from it. Therefore we will claim that the library sorting function, will sort and return our input faster than the priority queue.

2 Results

Based on our results shown in Figure 1, we can with a certainty of 95% say that for permuting a list `A` to give a result `B` such that `B` is sorted[3], the priority queue implementation has at most a 0.69071% chance of being faster or equally fast as the library function.[4]. This is given the following:

$$(1 - p_H)^{100} \leq .05$$
$$p_H \geq 1 - .05^{1/100}$$
$$p_H \geq 691/10.000$$

From Figure 1 we may also observe that the standard deviation for both the library function and for the priority queue is relatively small, while the distance between the two methodologies remain very large, indicating that the actual chance of the priority queue outperforming the library is even smaller estimated.

[1]`https://golang.org/pkg/container/heap/`

[2]`https://golang.org/pkg/sort/#Ints`

[3]Sorted in this case meaning B[i] $\leq$ B[i+1] for any i.

[4]What a bold statement. Is this even remotely close to the intended experiment?

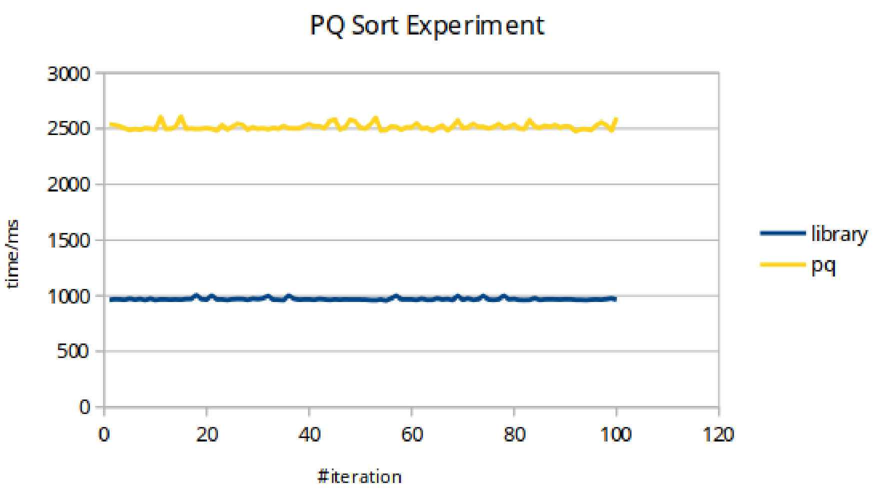

Figure 1: Graph plotting the running times of PQ-sort and Library-sort.

3 Bonus question

The priority queue will never be faster at sorting an array than a sensible library function. This is due to the overhead of maintaining the advantageous data structure that most priority queues will have. However, if one was fine with keeping the data in the priority queue, and had to repeatedly insert new items into the data structure, a priority queue would vastly outperform a basic array. For the priority queue we would get insertions of $O(\log n)$ in contrast to $O(n)$ with a naive array implementation..