Data Structures

Assignment 2

Jan-Willem van Bremen 500779265
Hugo de Groot 234567890

12-10-2018

Content

Introduction

We have worked on the backend of an application that keeps track of high-scores for games. The current problem is that the current implementation works for up to roughly 100 to 1000 players but when real data is fed into the application the high-score tables grinds to a hold or is very slow at best.
We are tasked to come up with a better, more efficient, implementation of the high-score sorting system. We used three different approaches and their performance differences. We used the insertion-sort, buckets-sort and the priority queue.

Using insertion-sort

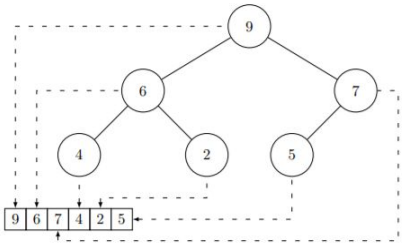
The idea behind insertion sort is that you take the list you want to sort. Then the program takes the first element of the list and saves it in a variable. The next step is to loop through the list until it finds a place where the highscore is bigger than the one that you're trying to find a new spot for. If it finds one then it switches the two numbers so the higher one is further on in the list. After that it repeats this process, but with the next number, until the entire list is sorted.

Using buckets-sort

The buckets-sort can be very fast because of how elements are ordered into buckets. We used arrayLists as buckets to store values as the index. That means more memory is required to run it at the cost of execution time.
The worst case efficiency of the buckets-sort is O(n2), and the best case is O(n+k) time where 'n' is the number of elements and 'k' is the number of buckets.

Priority queue

The priority queue is based on a heap where elements are sorted according to their priority. We used arrayLists to house the elements but you can visualize them to a tree structure as seen below. Each element can be a parent of a maximum of two (smaller) child elements.



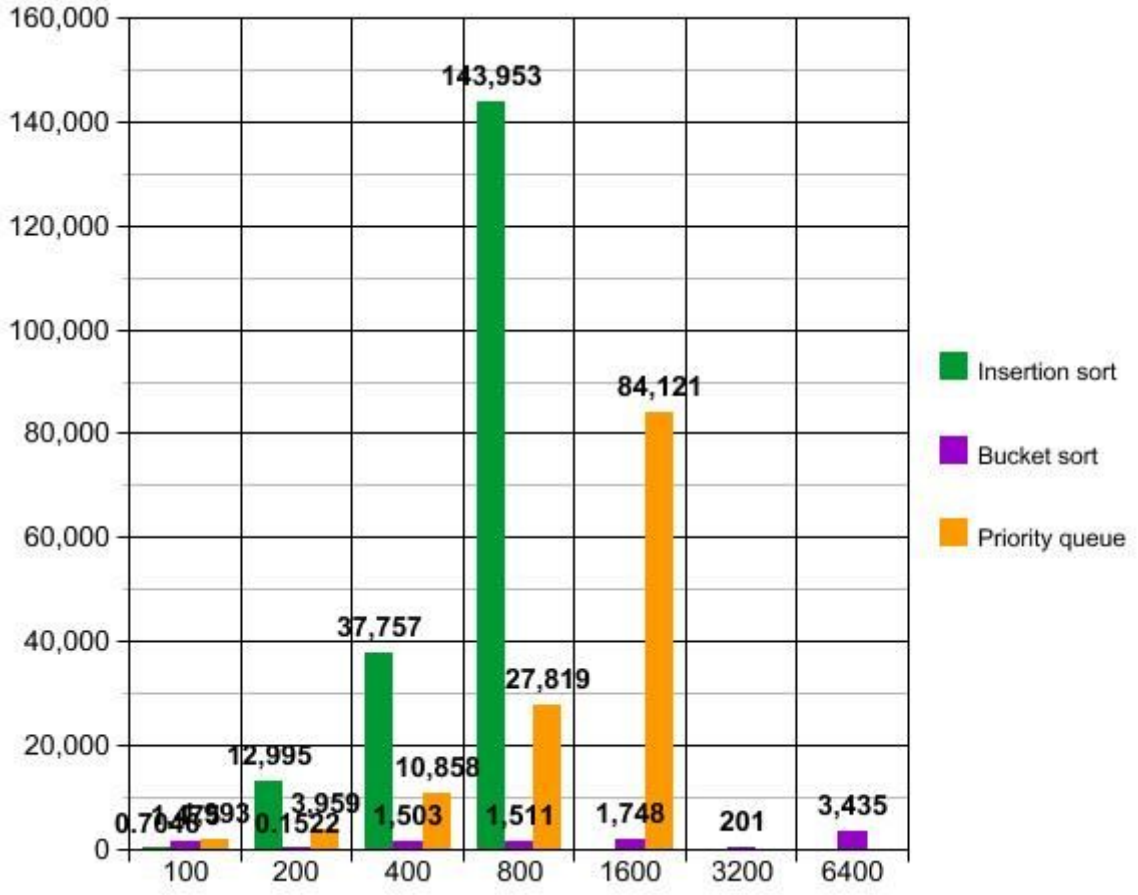
As new elements are added to the top of the list, they undergo a sinking treatment where they are compared to their children, and if they are smaller they will become their children. That way new elements are automatically sorted in the heap.

The efficiency of the priority queue is O(log(n)).

Performance Tests

To determine the performance of each sorting approach we have conducted tests using the -Xint JVM argument to ensure stable test results. In these tests we added various numbers of elements using the different methods and measured the execution time of each of them. In the table below we have displayed the results for each approach:

	100	200	400	800	1600	3200	6400
Insertion sort	0.7046	1,2995	3,7757	14,3953			
Bucket sort	0,1475	0.1522	0,1503	0,1511	0,1748	0,201	0,3435
Priority queue	0,1993	0,3959	1,0858	2,7819	8,4121		



Advice to sister

According to our development and testing we have concluded the best sorting approach to use is bucket sort because this method is the fastest, even when you add a lot of elements to the list that you need to sort. This is what we expected. The bucket sorting method is a steady method for sorting lists with a lot of elements.