Testing the performance of different search algorithms

Daniel Armstrong ID:300406381

Throughout the first trimester of Comp102 we only looked at arrays and ArrayLists and with this simple iteration to find things we want to find. Now being in Comp103 we see more and more types of collections and wit this comes more search algorithms as we find that simple iterations is simply not good enough, its slow its costly and simply does not compare. The following short report shows how a few different types of searches, stack up, mainly circulating around the use of binary search.

To conduct this experiment, requires writing a simple program that implements ArrayStack, the program should be designed to read a text file, and with that input load the values into an ArrayStack, then compare them to the dictionary to see if the spelling is correct. The program should be written with all the methods one would like to see in array list, like add (Adds an item), remove (Removes an item), contain (Contains the item), size (Returns the number of items in collection) and find index (Returns a certain item if in the collections of items.), with this program implementing simple iterations and then modified to implement binary search to compare the two. (The code is attached.) (could use more detail.)

Data:

Measurment:	Loading/Checking	Unsorted (s)	Sorted (s)		Average time: (s)	Sorted (s)	Unsorted (s)
1	Loading:	110.036	107.124		Loading:	126.8756	120.6368
	Checking:	132.697	131.346		Checking:	136.472	132.4004
2	Loading:	110.822	109.551				
	Checking:	134.156	138.02				
3	Loading:	104.423	110.87				
	Checking:	128.387	134.911				
4	Loading:	106.85	173.535				
	Checking:	128.971	142.633				
5	Loading:	171.053	133.298				
	Checking:	137.791	135.45				
1	Loading:	114.893	0.354		Loading:	0.3286	115.1862
	Checking:	0.568	0.572		Checking:	0.5746	0.5472
2	Loading:	114.638	0.339				
	Checking:	0.556	0.59				
3	Loading:	114.646	0.319				
	Checking:	0.549	0.557				
4	Loading:	114.941	0.322				
	Checking:	0.546	0.6				
5	Loading:	116.813	0.309				
	Checking:	0.517	0.554				
1	Loading:	0.278	0.246		Loading:	0.2506	0.2554
	Checking:	0.301	0.358		Checking:	0.2804	0.2804
2	Loading:	0.267	0.242				
	Checking:	0.261	0.288				
3	Loading:	0.228	0.286				
	Checking:	0.329	0.282				
4	Loading:	0.248	0.224				
	Checking:	0.285	0.217				
5	Loading:	0.256	0.255				
	Checking:	0.226	0.377				

Results measured on: Wednesday the 9th of August.

Different processors and systems, may cause variety in results.

Data definitions:

Loading – the time it took the program to load the objects into the collections.

Checking – the time taken for the program to check the loaded data with data from a document.

Sorted – the document being compared to loaded data, in an ordered format

Unsorted – the document being compared to loaded data, but without ordered format

Discussion:

From what we can see with the results we can see that the use of a Hash set is far superior to the binary search and far better than the simple iterator. All do the same job, but the Hash set uses a lot less time and computational power to do it, which is the main notice, the principle of the iterator, is to search through every value and compare it to the value you have, this is why it's so extensive on the power, if you consider this example, there are points you are comparing 1 word with 206,000 other strings, on the other hand a binary search employs the use of high mid and low, by continually cutting the collection in half until it has only a singular value, making checking very efficient, there are other methods, using a hash set instead of an array set, allowed the addition of items, in a much faster fashion whether they were sorted or not, and still implemented binary search, it allowed the computational power to be cut dramatically, shown in the results and a massive save in time.

Another search algorithm to consider interpolation which uses the process of comparing the first value to an area in the list, there are drawbacks to using interpolation, yes it would make checking faster, faster than binary search is unknown, but it would still need a sorted data file, so an array set would pretty much be a must.

Looking at the above results. All seem pretty much coherent, some slight differences could arse purely in what the system is doing now, and how much memory is delegated where. In this case there are no significant outliers. But to make sure my results carried validity, I far tested my program, gathered data then averaged the results. We have two data files. A sorted and unsorted which changed the performance by a lot especially with the binary search in the sorted array set, which meant there was lest memory taken up when adding values to the array set.

One notable outlier, is a check reading which is of the other readings by 50+ seconds, which could be caused by a variety of system changes.

From what we can see in the results, its clear hash set is the way to go when storing variables. Pairing this with a binary search makes a far better storage and search algorithm team. Which uses less computational power and uses less time, than say simple iteration.