Comp 103 Assignment 04 Results
Henry Wylde

Experiment:
The experiment done was on the efficiency of three different type of Collection classes in Java: ArrayBag, SortedArrayBag and HashBag.
Tests were done for the methods of adding items to each collection, removing items and checking whether the collection contains the items. In total, three different tests were run for each set of data (3 sets of data at counts of 20,000, 40,000 and 80,000) in order to compute an average time in miliseconds for each bag for each different data set.
How the experiments worked to compute the times per method, is that it would determine how long it would take to call those methods for the data counts (eg. It would check how long it takes for an ArrayBag to add 20,000 items to it, or a HashBag to call contains on the 20,000 data sets).

Summary:

Average totals for 20,000:	ArrayBag: 3530
	SortedArrayBag: 614
	HashBag: 105
Average totals for 40,000:	ArrayBag: 14280
	SortedArrayBag: 2239
	HashBag: 35
Average totals for 80,000:	ArrayBag: 67935
	SortedArrayBag: 8355
	HashBag: 56

Discussion:

For an ArrayBag:	adding is an O(1) method
	removing is an O(n) method
	contains is an O(n) method.
For a SortedArray Bag:	adding is an O(n) method
	removing is an O(n) method
	contains is an O(log n) method
For a HashBag:	adding is an O(1) method
	removing is an O(1) method
	contains is an O(1) method

For an ArrayBag, the contains method was very costly, in each of the tests it was the highest costing method to call. With the SortedArrayBag having an O(log n) contains method using a binary search algorithm, this significantly reduced the time it took for SortedArrayBag to call contains() on an item. While the other methods for a SortedArrayBag are O(n) cost, this is the worst case and the adding / removing method vary a lot as is shown by in the 40,000 data set with the adding method range being about 350ms (1416 – 1064). This shows that judging the time taken for the SortedArrayBag on adding on these data sets is unreliable as it is very subject to variation.

It was interesting noting how the HashBag remained an extremely fast bag to use for all data sets. With it being barely more than 100ms total for any of them, it is easily the best option for fast random access, adding and removing.

Results Table:
(Note: times are in miliseconds)

Amount of Data	Tests	Method	Bags		
			ArrayBag	SortedArrayBa	HashBag
20000	Test 1 (14981 Distinct)	Adding	22	261	80
		Contains	2615	24	16
		Remove	1135	278	26
		Total	3772	563	122
	Test 2 (14970 Distinct)	Adding	20	352	58
		Contains	2077	26	29
		Remove	1302	300	24
		Total	3399	678	111
	Test 3 (14978 Distinct)	Adding	23	326	38
		Contains	2085	29	19
		Remove	1311	245	26
		Total	3419	600	83
	Average	Adding	22	313	59
		Contains	2259	26	21
		Remove	1249	274	25
		Total	3530	614	105
40000	Test 1 (23132 Distinct)	Adding	1	1416	10
		Contains	8207	35	18
		Remove	5544	1067	20
		Total	13752	2518	48
	Test 2 (23116 Distinct)	Adding	1	1131	6
		Contains	8737	42	5
		Remove	5787	953	5
		Total	14525	2126	16
	Test 3 (23224 Distinct)	Adding	2	1064	22
		Contains	8669	39	8
		Remove	5891	971	10
		Total	14562	2074	40
	Average	Adding	1	1204	13
		Contains	8538	39	10
		Remove	5741	997	12
		Total	14280	2239	35
80000	Test 1 (29909 Distinct)	Adding	24	4204	34
		Contains	35903	98	11
		Remove	31772	4195	16
		Total	67699	8497	61
	Test 2 (29877 Distinct)	Adding	3	4171	18
		Contains	36286	95	14
		Remove	31945	4115	16
		Total	68234	8381	48
	Test 3 (29904 Distinct)	Adding	2	4162	34
		Contains	36001	97	13
		Remove	31868	3929	13
		Total	67871	8188	60
	Average	Adding	10	4179	29
		Contains	36063	97	13
		Remove	31862	4080	15
		Total	67935	8355	56