**Out of Lab 4 – Analysis**

Phase II:

2. To examine and analyze the outliers we used an algorithm to calculate the variance of the data for each parameter (ie, ArrayList Add vs Vector Add) and then using the proper variance for each set of data, then calculated numbers that exceeded 1 standard deviation. The results are attached and formatted to include the run number, the position of the value stored in our data array (this will be constant), and the value itself. In terms of add time, all the collections seem to have fairly consistent outlier data such that the culmination of runs has between 2 and 5 outliers with a mostly equal distribution between larger and smaller outliers. The only worthy conclusion that can be drawn from the Search outlier results is that on average there were more outliers for all the data sets, generally between 3-6, which is statistically slightly higher than the number of outliers should be outside 1 standard deviation (normally there would be around 68-70% of values within 1 standard deviation). Looking at outliers for sort times, all of the Hash collection variants (except for HashTable, which had 6 outliers) have very consistent data with only 1-2 outliers with the rest of the collections averaging between 3 and 4 outliers.

3. **Best Add Time:**

-ArrayList

-Vector

-HashMap

**Best Search Time:**

-HashSet

-Linked HashSet

-TreeSet

**Best Remove Time:**

-Linked HashSet

-HashMap

-Linked HashMap

**Best Over All Time:**

-HashSet

-TreeSet

-Linked HashSet

**Worst Add Time:**

-Linked HashSet

-Linked List

-TreeMap

**Worst Search Time:**

**-**TreeMap

-HashMap

-HashTable

**Worst Remove Time:**

**-**Linked List

-ArrayList

-Vector

**Worst Over All Time:**

**-**TreeMap

-HashMap

-HashTable

4. – Analysis of Times:

Best Add Time – ArrayList and Vectors are certainly two obvious contenders for best add time as they are both backed by arrays and adding elements only require the addition of an element to the end of the array which is either O(1) or O(n) (being the worst case). HashMap is also a good contender because values can simply be added in any order to the map without any sort of order or iterator sorting the values as they’re entered.

Best Search Time – In terms of search performance, HashSet is great because the constant time O(1) performance is often faster for the basic operations than O(log(n)) time which while is very good, will not be as good as constant time. This is why TreeSet and Linked HashSet are also part of the efficient-elite search collections.

Best Remove Time – Unsurprisingly, several of the same characteristic structures rank in the best remove time as best search time, as removal and searching frequently go hand in hand for several collections. Linked HashMap and Linked HashSet are very efficient at removal because they maintain insertion order, which can be used to remove elements very quickly. HashMap similarly has a constant remove time of O(1), since the collection is not ordered and does not have to be iterated through to remove an element.

Best Overall Time – HashSet, TreeSet, and Linked HashSet had the best overall performance because the general performance times for HashSet tend to be O(1), for TreeSet it is O(log(n)) and for Linked HashSet it’s somewhere in between but tends to be closer to constant time. These collections simply perform the basic operations faster than most other collection options.

Worst Add Time – It makes sense that Linked HashSet has a slow insertion speed being that it maintains the insertion order. Linked List is rocking about O(n) insertion time with the focal point being that as the list increases the insertion speed will slow.

Worst Search Time -It’s no surprise that TreeMap is the worst collection when it comes to searching (guaranteed log(n) wooohooo), it’s basically like sequentially searching through a giant array. Every element must be checked until the correct one is found. HashMaps are similarly bad in that they are not created with or maintained in any order, so you’ll probably have to look at most of the elements before you find the one you’re looking for. Hash Tables often suffer from O(n) worst case time complexity especially when it comes to searches because too many items can be hashed into the same key.

Worst Remove Time – It makes sense that ArrayList, Vectors, and Linked List all have the worst remove times of all the collections as you must iterate through each collection element by element until the correct item is found and can be removed from the list. The main point being that none of these items possess direct links to elements, which then relies solely on the position of the desired item being close to the front of the list.

Worst Over All Time – Because we searched through so many numbers (21,474

), the inefficiencies of TreeMap, HashMap, and HashTable, the overall time greatly reflected the result.

5. How the position of top 3 overall collections would change if searches are doubled. Hypothesis and conclusion.

HashSet, Linked HashSet, and TreeSet should be at the top in speed.

ACTUAL (Best 3 from very best):

HashSet

TreeSet

Linked HashSet

TreeMaps, HashMaps, and Hash Tables should be at the bottom in speed.

ACTUAL (Worst 3 from very worst):

TreeMap

HashMap

HashTable

6. How the position of top 3 overall collections would change if searches are halved. Hypothesis and conclusion.

HashSet, Linked HashSet, and TreeSet should be at the top in speed.

ACTUAL (Best 3 from very best):

HashSet

TreeSet

Linked HashSet

TreeMaps, HashMaps, and Hash Tables should be at the bottom in speed.

ACTUAL (Worst 3 from very worst):

TreeMap

HashMap

HashTable