Alexander Dean

COS 285

27 September 2017

Report: Program 4

The purpose of program four was to implement code that would locate the first index in an array after a matching value, code that would insert a value properly into the AddendumList, and code that would merge the last two levels of arrays.

For the findIndexAfter() method, code from findFirstIndex() was recycled. Instead of narrowing down on the first index of a matching item, the code located the last index of a matching item, and returned the index after this item. The Big-O order of this method is approximate O(log(n) + k), where k is the number of matching items in the list. In the worst case, the method will return the middle index of the array (if the array is all equivalent items). This means that the Big-O order would be O(log(N) + N/2). This means that the worst case looks through half of the items in the array. Originally, a different binary search was implemented that performed two comparisons per method call (the method is recursive): one on the element at the middle index, and one on the element directly after. While this method also returned the correct result, it was substantially slower on the item set given in the test cases. It is my belief that this may be faster when 10000 items are used, for it should be, in the worst case, O(2log(N)).

The merge1Level() method builds off of the findIndexAfter() method. For each element in the last array, the program finds the merge index in the second to last array. It then adds all elements in the second to last array to the newly created array, then adds the element that was compared in the method to the newly created array. It does this for all elements in the last array, then adds any remaining elements from the second to last array to the new array, stably merging both the L2 arrays. In the worst case, this code performs the findIndexAfter() method for every element in the last array, which would be N/2 elements large. This means that there are (N/2)log(N) comparisons at most. It also must iterate through each loop, which means N items are added. The final order of this method could be approximated by O(N + (N/2)log(N)), which can further be stated as simply O(Nlog(N)).

The add() method builds off of both of the previous methods. It finds the index in the L2 array that it needs to insert the item into. It shifts everything in that array down that is larger than that index, and inserts the item. If the array is the same size or greater than the previous array, it will perform the merge1Level() method until this is no longer the case. In the worst case, the sizes of the L2 Arrays in the addendum list are as follows: N/2, N/4, N/8, N/16…8, 4. This means it will have to perform the merge log2(N/2) – 2 times. The subtraction of 2 comes from the fact that the minimum array size is 4, and log2(4) = 2. If the worst case of the merge1Level() method is O(Nlog(N)), this means that the Big-O order in the worst case of the add() method is approximately O(Nlog2(N)).

To grow the L1 array, the policy of increasing the array by 50% was used. It was determined that this would happen less and less as items were added, so it was unnecessary to double the L1 array each time.

The requested printouts of the Report tab, as well as the printout of test45.txt can be found following this report. At this time, there are no known bugs with the code.