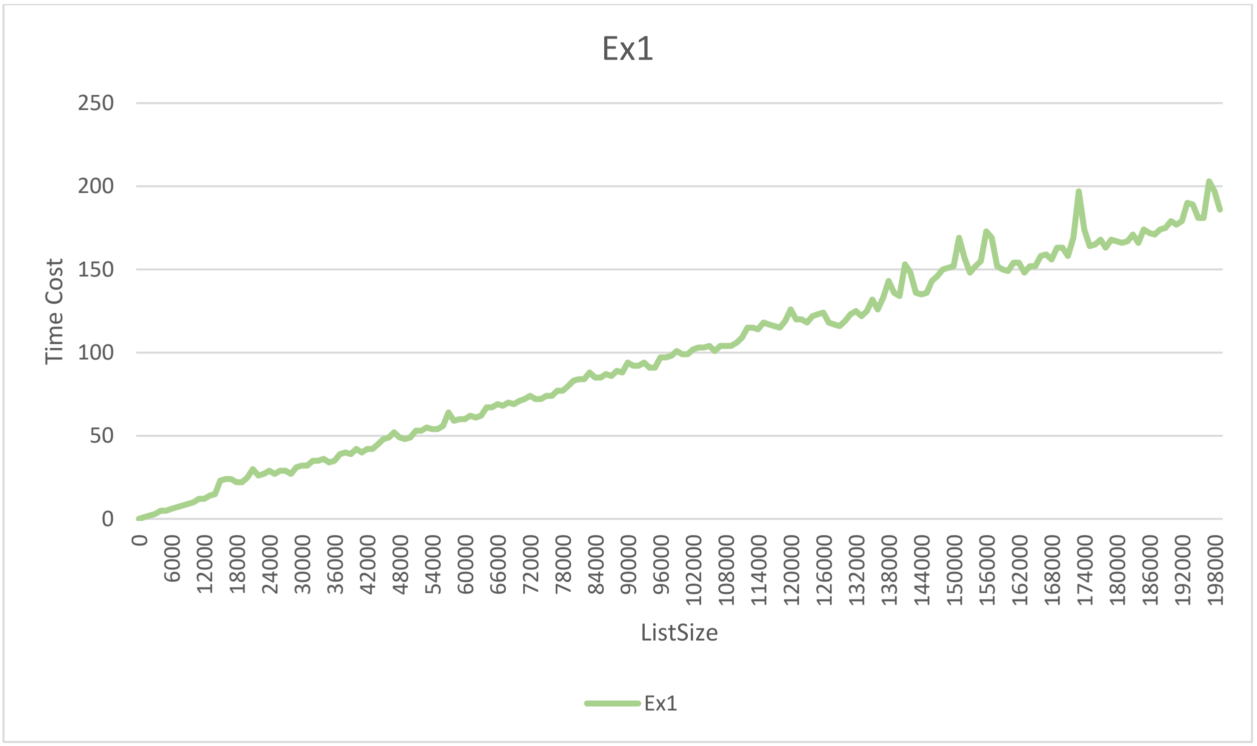
Experiment#1

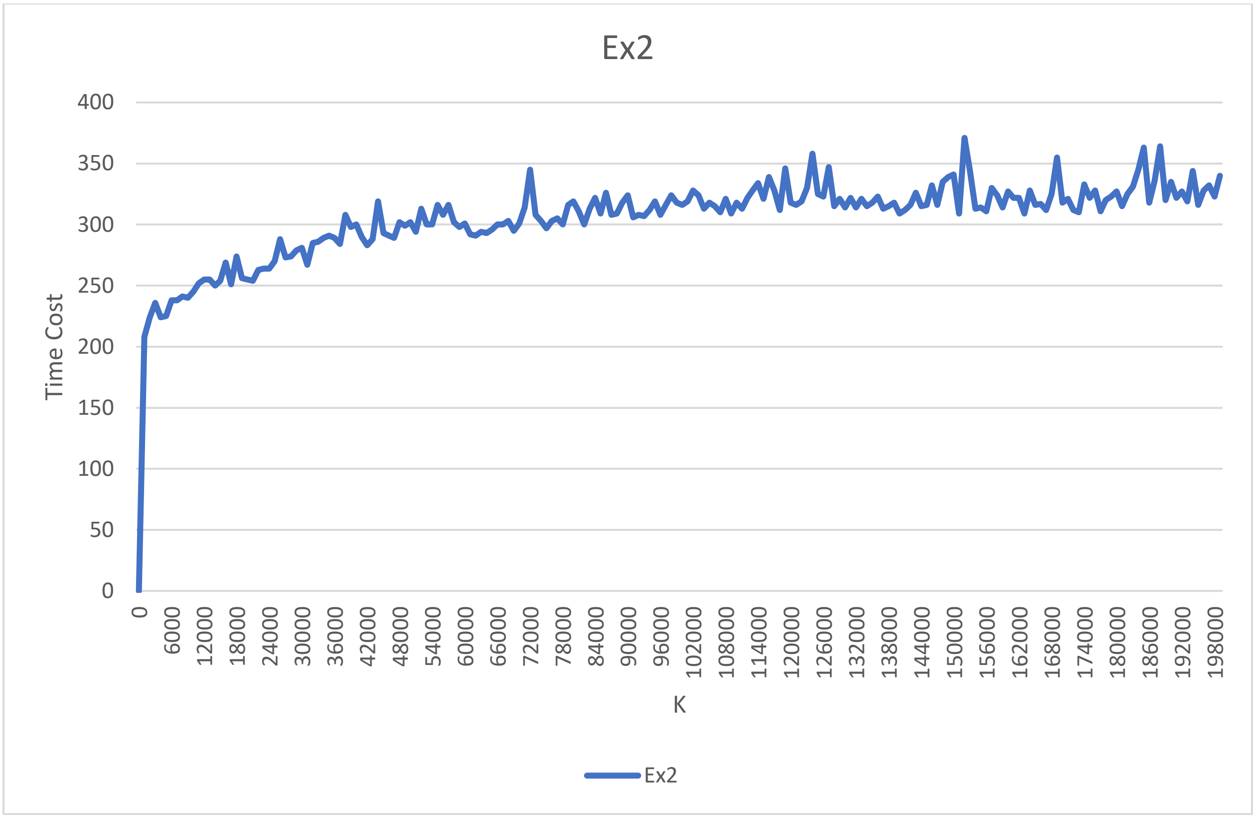
1. This experiment tests the time cost of topKSort() searching K = 500 items in different lists with increasing list size n.
2. Result:



3. My prediction is to have a graph showing a nearly linear shape. The result agrees with my prediction. The topKSort() needs to loop through the whole list in each run, and when item with large value is found, the item will be inserted into the heap, which takes log(K), therefore, totally, it is about O(nlog(K)), and O(n) in this case, because log(K) is a constant, so O(nlog(K)) can be considered as O(n), which should show a linear shape(as list size grows, the time cost grows linearly).

Experiment#2

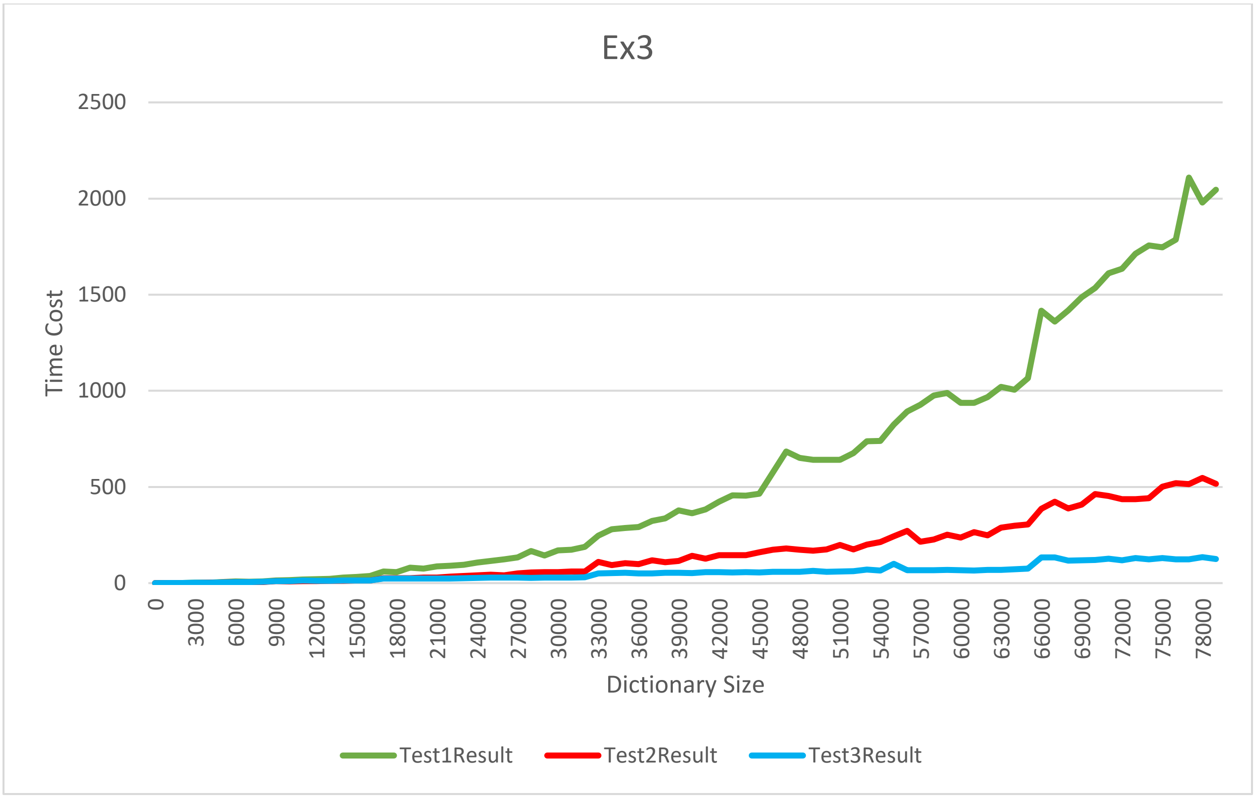
1. This experiment tests the time cost of topKSort() searching different number of items(changing K) in List with specified length n = 200000.
2. Result:



3. My prediction is to see a graph showing O(log(K)). The result agrees with my prediction. This is experiment is also testing topKSort(), so it is about O(nlog(K)) as explained in experiment#1, and because n = 200000 is a constant, so O(nlog(K)) can be considered as O(log(K)), which should be a shape of log function(as K growing, the time cost is going to be a constant.)

Experiment#3

1. This experiment is testing the time cost of put() in ChainedHashDictionary by using three different hash code.
2. Result:



3. I predict that the result from test 1 should be least efficient, then test2 and test 3 is the best in efficiency. The result agrees with my guess. For test 1, the hash code is the sum of the first four char in the string, which means the range is specified(a range of 4 * 26), therefore, the collision will more and more likely to happen when item to put is increasing. The test 2 is like test 1, whose hash code in within a range, but the range for test2(a range of numOfChars * 26) is much larger than that of test 1, so there will be less collision than test 1. Test 3 is the java implementation which creates hash within an extremely wide range(recursively multiplied by 31 and sum with the chars traversed), which means the collision is very unlikely to happen, so, this is pretty close to the best case O(1). Among three tests, test 1 is most close to the worst case, O(n). The test 2 is better than test 1 but worse than test 3. And because there is an outer for each loop in each test, so the first two tests look like O(n^2) and test 3 looks like O(n).