Logan Habner, Edmund Trinh
CSE 373
Project 3: Group Writeup

1. The formulas are as follows:

$$parent(i) = \frac{i-1}{4}$$
$$child(i,j) = 4 * i + j + 1$$

2. We refactored the redundancy by checking the smallest found child with every other child from left to right. Beginning with the furthest left child as the “smallest”, we compare it to the next node to the right. The index of the smaller one is then kept as the “smallest”. Once this loop goes through all nodes, we will have the index of the smallest child.
   The only challenge that we had was with retrieving the first child as the comparison would fail due to the child function trying to access an index outside of the bounds of the array. We simply included a check for the index in the child function that calls the heap resizing function we made if necessary.

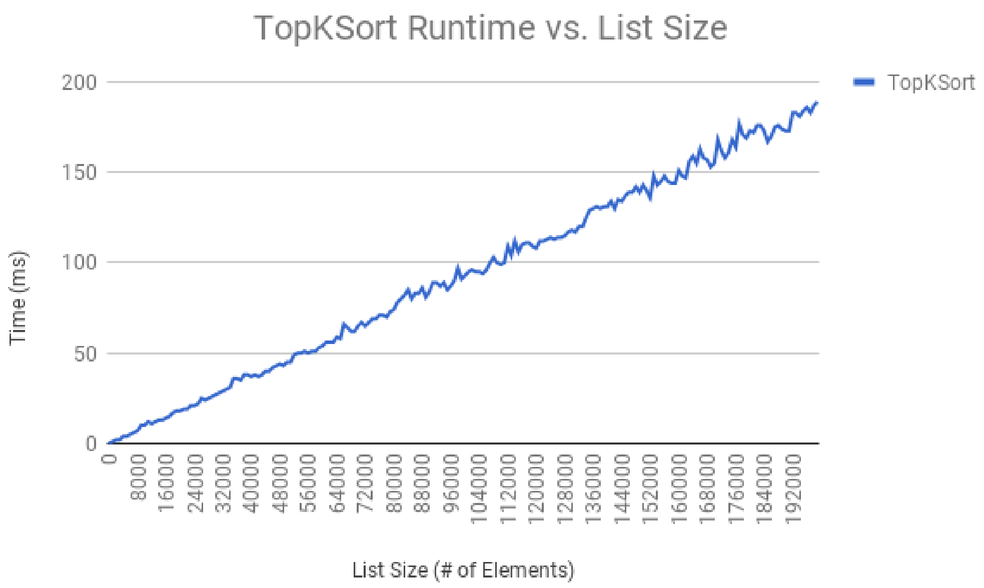
3.
   a. Experiment 1
      1. This experiment creates a list of list sizes from 0 to 200,000, with a stepping of 1000. It then iterates over this list, creating a list from 0 to the current list size, with a stepping of 1, and runs the topKSort function for a K of 500 ten times on the list with a stepping of 1 and records the time taken. The experiment runs this process 5 times.
      2. If MAX_LIST_SIZE = n, and STEP = s, then one trial in the experiment has a worst case runtime of:

$$O\left(\frac{n}{s} * n * \log(k)\right)$$

         The n*log(k) comes from the runtime of the topKSort, which inserts n elements into a heap with log(k) insertion time. The n/s comes from the size of the listSizes, which the experiment uses each element from. The value of the element currently used is represented as the n in the n*log(k) of topKSort. Since the maximum for the listSize is MAX_LIST_SIZE and the n in n*log(k) is based on the current list size, n can be considered the same variable.
         Since the n/s is fixed as that n is the maximum n, and k is fixed at 500, then the actual runtime should be O(n).

Logan Habner, Edmund Trinh
CSE 373
Project 3: Group Writeup

3.

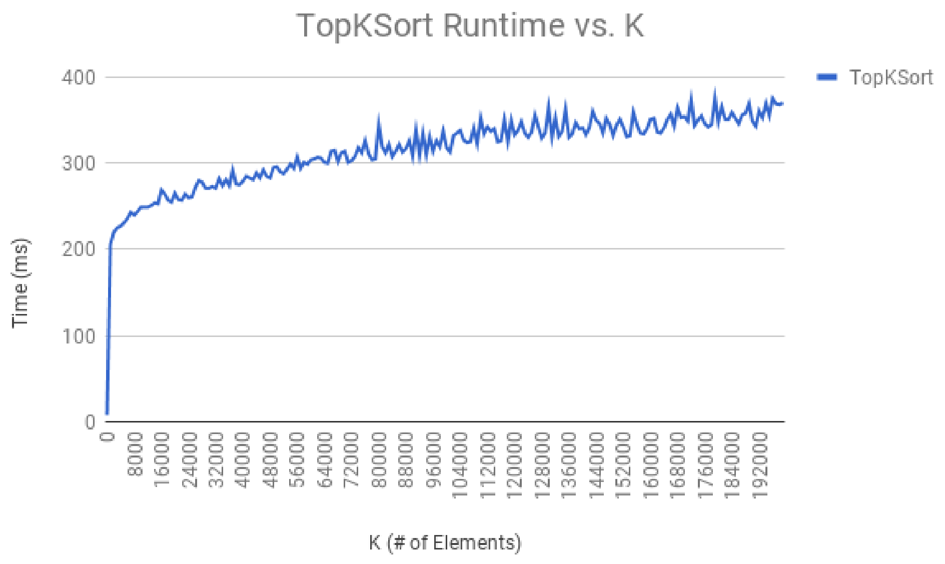


4. Since k is fixed at 500, and the max n and s are fixed, the bound of the function becomes O(n) instead of what we wrote. Our answer is correct for the worst case, but the actual runtime is O(n), as the graph shows (with a little variation).

b. Experiment 2
   1. This experiment creates a list of sizes of k from 0 to 200,000, with a stepping of 1000. It then iterates over this list, and for each element, the experiment creates a list from 0 to 200,000 with a stepping of 1 and runs topKSort 10 times with the current value from the valuesOfK list and records how long it took for the 10 times. The experiment runs this trial 5 times.
   2. If MAX_LIST_SIZE = n, and STEP = s, then one trial in the experiment has a worst-case runtime of:
   $$O\left(\frac{n}{s} * n * \log(n)\right)$$
   The n*log(n) comes from the runtime of the topKSort when k approaches n (log(n) would usually be log(k)), which inserts n elements into a heap with log(k) insertion time. The n/s comes from the size of the valuesOfK, which the experiment uses each element from. The value of the element currently used is represented as the n in the n*log(k) of topKSort. Since the maximum for the listSize is MAX_LIST_SIZE and the n in n*log(k) is based on the current list size, n can be considered the same variable.
   Since the first and second n are the maximum n, and s is fixed, the actual runtime is O(log(n)).

Logan Habner, Edmund Trinh
CSE 373
Project 3: Group Writeup

3.



4. Because the first and second n are the maximum and s is fixed, the runtime displayed in the graph of the results is O(log(n)). This is the same as in our hypothesis.

c. Experiment 3
   1. This experiment tests putting random arrays of chars into a ChainedHashDictionary. The experiment uses a MAX_DICTIONARY_SIZE of 80000, with a stepping of 1000, to create a list of dictionary sizes. This list is iterated over and the tests are run for each element. The experiment tests three different ways of implementing hash codes to test the Dictionary, with the time to finish all insertions (put()) recorded. All of these are run 5 times.
   2. The first test hash code returns a constant that is the addition of the value of the first 4 chars in the array, so it should make the ChainedHashDictionary run around O(n) runtime.
      The second test hash code returns the addition of the values of all chars in the array, so it should make the ChainedHashDictionary run around O(n).
      The third test hash code returns an output that multiplies itself by 31 and adds the value of the current char for each char in the array, so it should make the ChainedHashDictionary run in around O(1) runtime.
      If MAX_DICTIONARY_SIZE = n, and step = s, then:
      Test 1 runtime:

$$O\left(\frac{n}{s} * n\right)$$

Logan Habner, Edmund Trinh
CSE 373
Project 3: Group Writeup

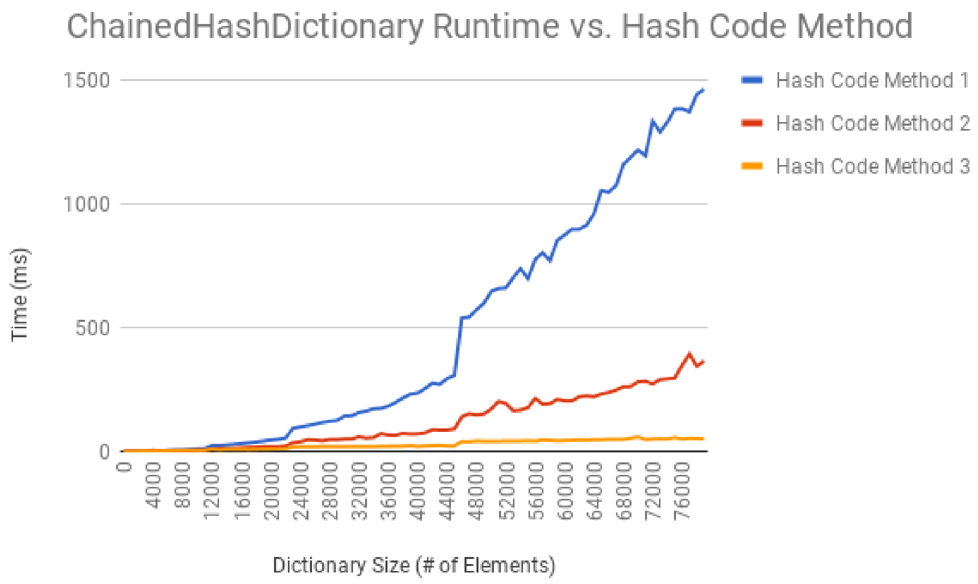
Test 2 runtime:

$$O\left(\frac{n}{s} * \mathrm{n}\right)$$

Test 3 runtime:

$$O\left(\frac{n}{s}\right)$$

3.



4. Since s is fixed in all tests, the runtime of the tests are as follows, based on the graph:
   Test 1: O(n^2)
   Test 2: O(n^2)
   Test 3: O(n)

   Test 1 and 2 both have the same big-O bound, but they are a constant factor different in actual runtime, so Test 1 takes more time. The results of the tests confirm our hypotheses, with the caveat that s is fixed, making the big-O bound slightly different.