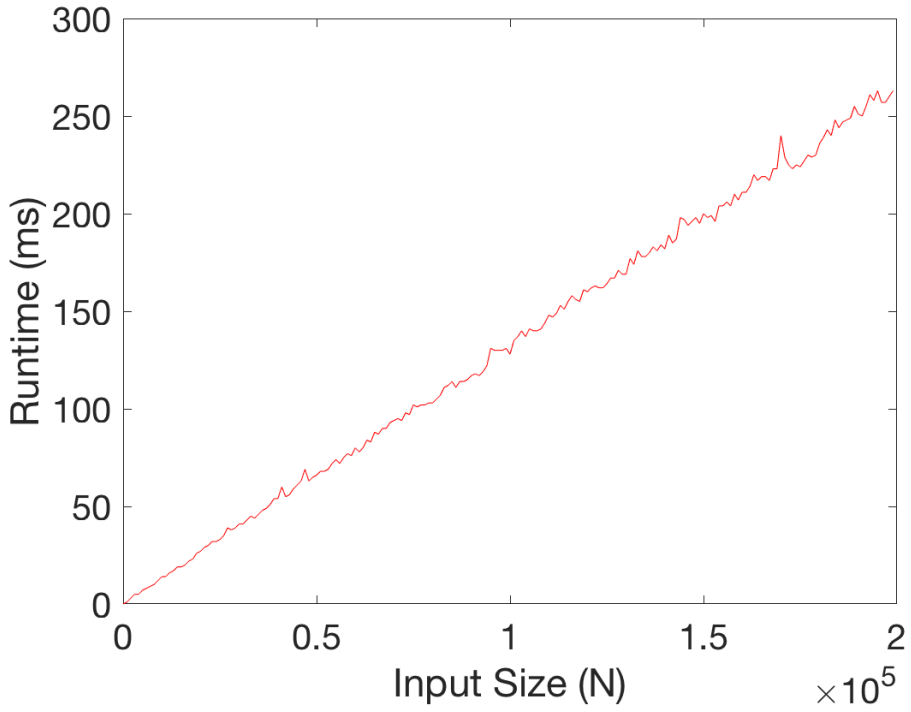
Experiment 1

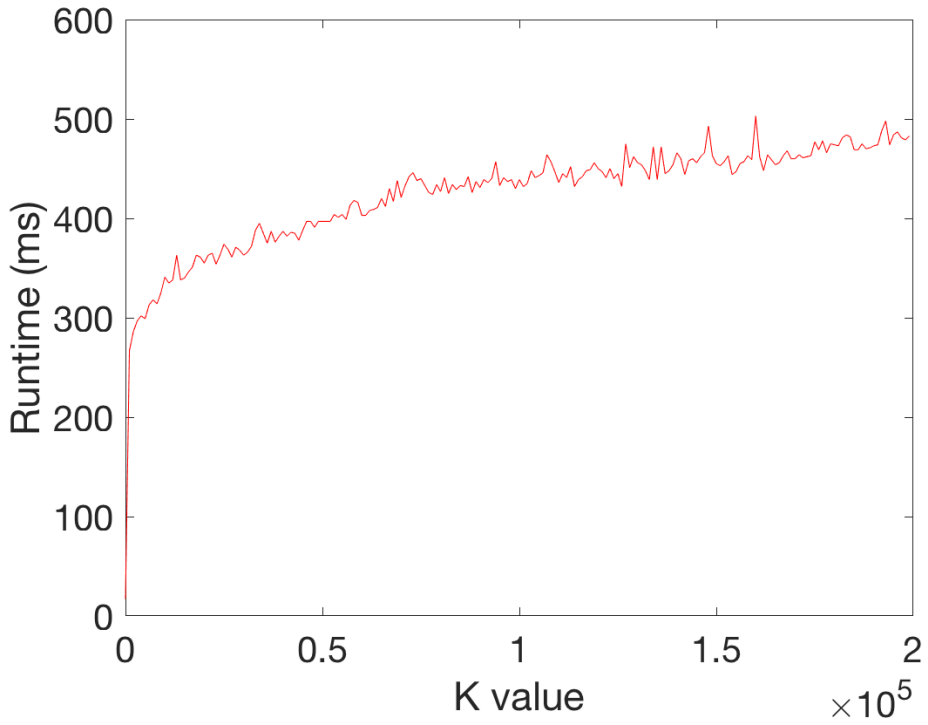
1. This experiment seems to be repeatedly timing the runtime of topKsort on lists of various sizes between 0 and 200,00 with a set value of K (500).
    a. I expect the runtime to grow linearly with the increase in the input size, as the algorithm of topKsort was designed to run in O(n log k) time
2. Plot



3. My prediction seems correct: runtime increases relatively linearly with an increase in the input (list) size. I expect this because the topKsort does n inserts/removes into a heap of size k, which take log k operation each, and thus overall this will run in O(n log k) time. Thus, we see the runtime scales linearly with the input size. If it did not scale linearly, I would know I had a bug in my code. If it did not scale linearly, there would be no benefit in using this algorithm over an algorithm that simply sorts the entire list.

Experiment 2

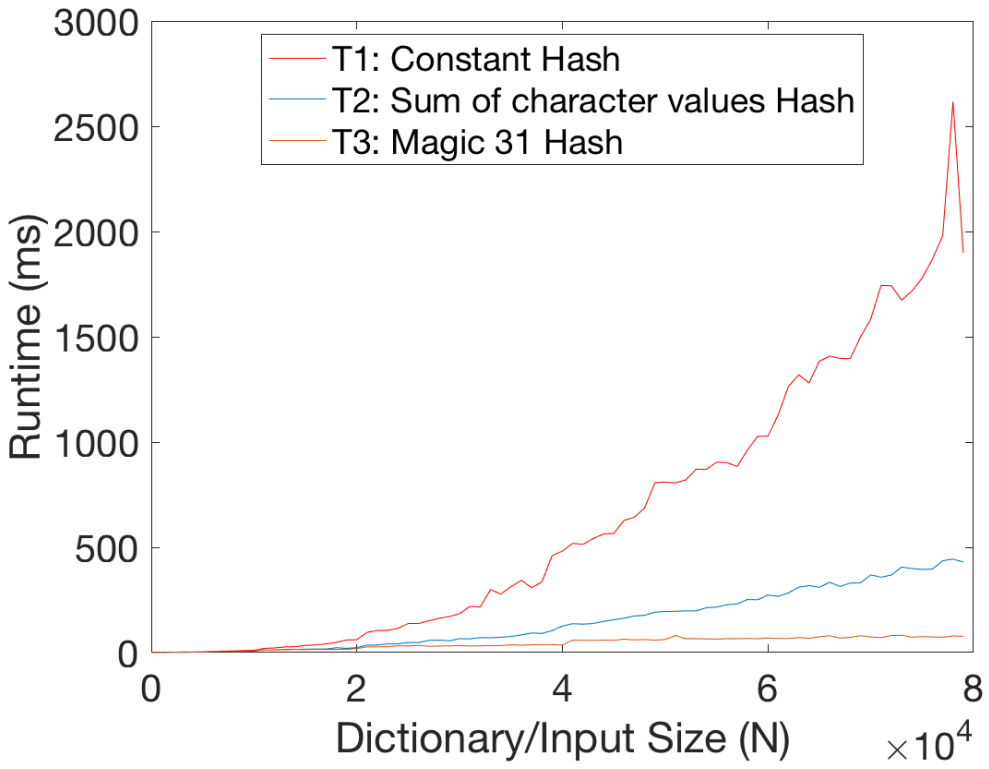
1. This experiment seems to be repeatedly timing the runtime of topKsort on a list of size 200,000 with various values of K (from 0 to 200,000).
    a. I expect the runtime to grow logarithmically with the increase in K, as the algorithm of topKsort was designed to run in O(n log k) time.
2. Plot



3. My prediction appears correct: there seems to be a logarithmic relationship between the K value in topKsort and the runtime (as long as K <= list length). This is because the topKsort algorithm requires n inserts/removes into a heap of size k, requiring log k operations each time. Overall, this makes the runtime complexity O(n log k) and we can see runtime asymptotically scales logarithmically with K. The increase in runtime slows down because as the binary heap gets taller, its layers get more and more nodes. Thus, it can store more new nodes before having to build another layer. Since we only have to traverse through each level once, not each node, once the heap is large this allows us to increase the number of elements in heap immensely without drastically changing the run time. Thus, extra cost of traversing the heap slows.

Experiment 3

1. This experiment seems to be repeatedly timing the runtime of the dictionary.put method of our ChainedHashDictionary at various dictionary sizes. Test 1, 2, and 3 all utilize different hashing methods, so this experiment is likely testing the effectiveness of these various hashing methods (i.e. which is the best at avoiding collisions).
    a. I expect the constant hash (test 1) to be the slowest as it results in many collisions due to its low variability element to element. I expect the sum of characters values hash to perform better as it can avoid collisions by adding more variability in the hash between separate elements. Magic 31 hash will probably perform the best because it is magic.
2. Plot



3. My prediction seems correct: the constant hash performs much worse than the other two, and the magic 31 hash performs the best. The tests appear to run in nonlinear time with respect to the input size. In the worst case, the chainedHashDictionary has all its elements in one bucket, and has to traverse every key-value pair to see if the key is already present. This is only an O(N) traversal operation at worst (if key is unique), but because the test calls N of these .put(), it makes the test runtime increase with O(N^2). So the Hash/Put operations are linear, but the N calls of the from the test makes the overall test nonlinear.

- **Questions about the project:**
  1. What is your name and your partner's name?

     Zachary McNulty (Partner: Mahir Bathija)

  2. How was the project? Did you enjoy it?

     I thought the algorithm design aspect of topKsort was cool. It was a nice challenge, and I liked how it focused more on understanding how the code was working and implementing an efficient algorithm than simply writing a lot of code. Once you developed the algorithm, the rest was easy. I thought the ArrayHeap was pretty mundane for the most part, but the percolate down algorithm had some interesting edge cases to consider. I did not really enjoy having to write my own testing code, although it is probably a good skill to learn.

  3. Do you have any suggestions for improving the project?

     I was a bit lost on how to decide certain facets of the testing programs, specifically the stress tests. I never really knew what good benchmarks for the timeout times or input sizes were. I wish this section gave more advice/guidance on tools for choosing appropriate values.

- **Questions about your partner:**
  1. How did you feel about pair programming this assignment?

     It did not work too well on this project. Because there was so little code to write, most of the time was spent on algorithm design. It was pretty hard to work together on this process, besides helping troubleshoot ideas.

  2. How was your partnership?

     It was alright. I seem to be doing most of the hard work, and I try to help him learn along the way. Now that we started pair programming, it is a bit easier to guide him and help him catch his mistakes. It still can be a bit frustrating at times and I kind of wish I could just work alone.