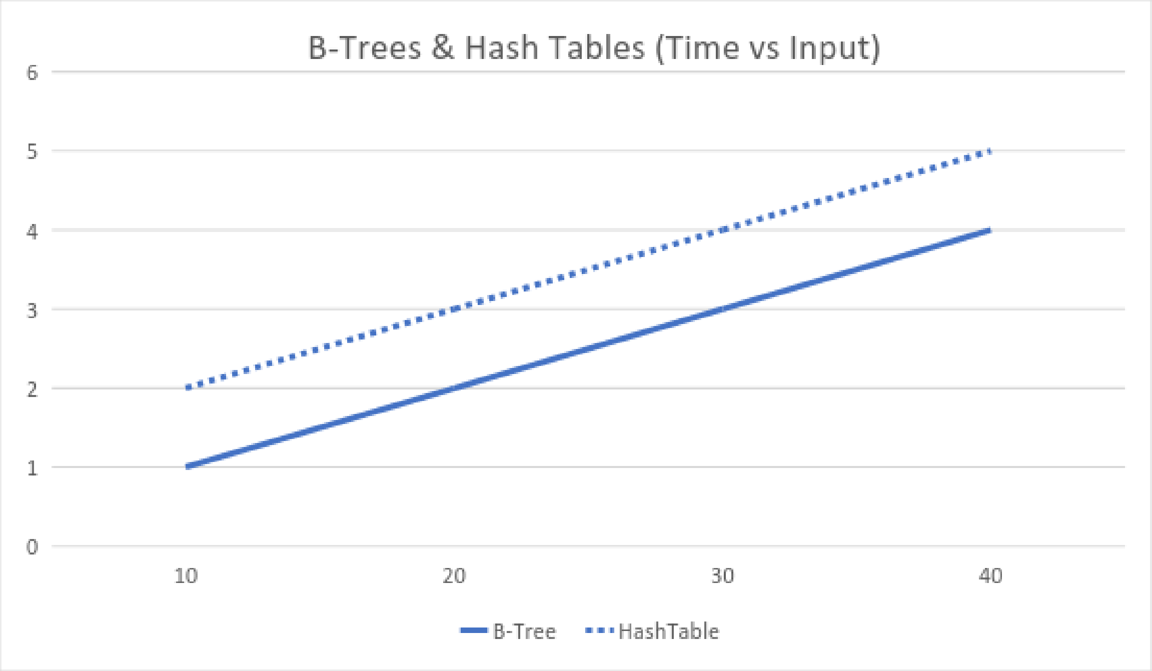
B-Trees vs. HashTables

Construct input files for B-Trees and HashTables to demonstrate that a B-Tree is asymptotically better than a HashTable. To do this, we expect you to show trends. You might consider fitting a curve to your results. Explain your intuition on why your results are what they are.

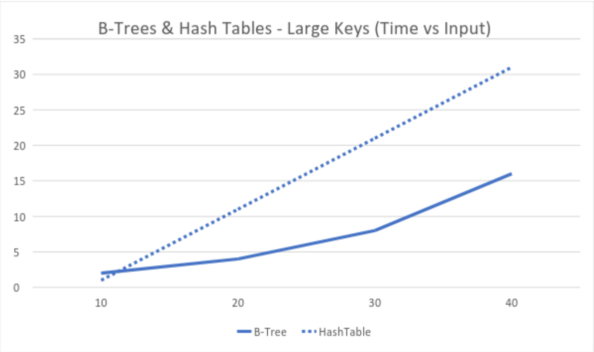
Bad Answer #1:

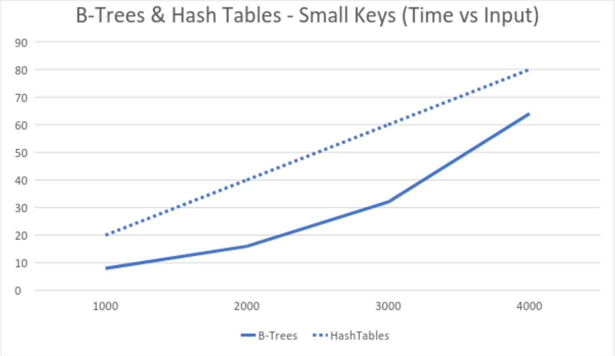


As you can see here, the B-Tree clearly has better run time at many different inputs. The graphs show that both run times increase linearly, but the B-Tree always has a slightly lower run time, so it's definitely the better data structure. All of our inputs are in the experiments file. We used different types of values: integers, strings and objects containing (x,y) coordinate pairs. We thought it was very interesting that both run times looked pretty linearly, we would have expected more differences, but data is data! For practical use, we don't see that there would be much of a difference in using either one of these since HashTable's run time is only slightly worse than B-Tree's.

What's wrong?

Bad Answer #2:





Our B-Tree did have better run time, as shown in the graph above. We think this is due to the fact that the B-Tree is paging to disk efficiently, but the HashTable is having to page to disk far more often. We tried two different kinds of input, one with very large keys with a lower M and L, and another with smaller keys and a larger M and L. We found that in both cases, the B-Tree performed better. This surprised us as HashTables usually work quite well with small values, but clearly if there are enough of them, having to page to disk multiple times really does make a difference. We did try with small numbers of small sized input as well, and as expected the HashTable performed better (we did not include this graph, we just did it out of curiosity). This made sense to us as the whole advantage of a B-Tree is more efficient disk look-ups, so the overhead in the B-Tree makes it perform worse than a HashTable with small inputs. Up to this point we had wondered why we don't hear more about B-Trees as they seem like such a better data structure to use, but now we see that there are only specific cases where it is really advantageous to use a B-Tree. (We know you said this in class, but now we've seen it in the data!) We think this sort of data structure would be good for things like large file systems on servers, or perhaps an application like GoogleDocs where a company is storing a lot of files within a lot of nested directories.

What's wrong?