Homework 5 - Lists, Trees, and Tries

`LinkedList` vs `ArrayList` vs. `TreeSet`

Structure	Timing (s)	Memory (MB)
`ArrayList`	14.849	6.6804
`LinkedList`	18.8428	12.871496
`TreeSet`	15.9886	

Conclusion

I would say that yes, this does make sense. Since `ArrayList` has to store only the index on top of the data, it would naturally take up less storage than `LinkedList` which would have to store two pointers to the next and previous elements. The logic also follows for the ArrayList to take up less time for adding because whenever an `ArrayList` gets to over half capacity, it doubles. This means that it is significantly faster to add because the `ArrayList` creates new memory space in chunks. Finally, `TreeSet` would take only a little bit more storage since it has to store more pointers and nodes but the same amount of data. However, the adding happens almost instantly because adding for trees is `O(h)`.

`Trie`

Structure	Timing (s)	Memory (MB)
`Trie`	0.001	43.033528

Conclusion

I would say that this makes sense since a `trie` would have to store quite a few different nodes and the structured is mired by a lot of overhead. However, it is unbelievably fast because you only have to do work proportional to the length of the word, which means that it does not matter how many words are stored. The longest word in `dictionary.txt` is ~26 characters, which means that adding is basically constant. This is different because others structures depend on the information being small in order to be fast.