Boggle Analysis

Lexicon Implementation	Iterate Time	Word Time	Prefix Time	Node Count	One Way Count
SimpleLexicon	0.063	0.043	0.121		
BinarySearchLexicon	0.053	0.002	0.113		
TrieLexicon	0.355	0.001	0.073	153759	70578
CompressedTrieLexicon	0.156	0.001	0.095	114921	31740

Based on this data, the TrieLexicon is "fastest" based on Word Time and Prefix Time. This is due to the fast that it runs in pseudo-constant time, O(w), running at the time of the longest chain in the trie. Meanwhile, SimpleLexicon should run near O(n) time, as it is iterates through each item one by one. BinarySearchLexicon should run near O(log(n)) time, as it uses binary search, effectively recursively splitting the lexicon in half each iteration. However, the Iteration Time for TrieLexicon is much slower because of its iterator method in which recursively calls and appends nodes finding each word, in effectively nested loops. Meanwhile, the other two methods only use simple iteration through the array.

The counts for the nodes and one-way counts show that the CompressedTrieLexicon saves space both in length and number of total nodes. However, it takes more time for word and prefix identification as a trade-off. They perform relatively similarly, both running faster than the other two in Word and Prefix Time.

BoardFirstAutoPlayer:

Lexicon	Time
SimpleLexicon	1.121
BinarySearchLexicon	0.632
TrieLexicon	0.403

LexiconFirstAutoPlayer:

Lexicon	Time
SimpleLexicon	20.111
BinarySearchLexicon	17.505
TrieLexicon	21.493

Based on the above data, BoardFirstAutoPlayer runs much faster than LexiconFirstAutoPlayer, using any of the lexicons. BoardFirstAutoPlayer only runs through possibilities of words/prefixes on the board, and stops as soon as there is no possibility of a prefix from a location. Thus, it has several times fewer iterations than LexiconFirstAutoPlayer. LexiconFirstAutoPlayer will iterate through each word in the lexicon in a loop, and for each iteration additionally call the GoodWordOnBoard class which recursively backtracks to find the word as BoardFirstAutoPlayer does. Using this data, I selected BoardFirstAutoPlayer and TrieLexicon as the fastest combination available to conduct the following experiments to find the maximum-scoring boards.

4 by 4 Boggle Board:

Count	Max	Time
1000	889	3.033
10000	889	24.166
50000	1011	121.463
100000	1011	246.048
1000000 (est.)		2400

Board found:

```
c  l  i  t
s  m  e  r
b  d  a  s
c  l  e  h
```

5 by 5 Boggle Board:

Count	Max	Time
1000	1301	8.751
10000	2120	69.141
50000	2120	394.907
100000	2120	688.006
1000000 (est.)		8000

Board found:

```
p  a  c  o  d
o  x  s  e  r
a  t  n  t  r
n  i  e  a  s
d  r  n  c  e
```

Based on the above data, the running time increases linearly with the count. As each number of boards doubles, the time doubles. This is because each count means the game is played entirely once with a new randomly generated board, and should thus be linear, assuming random boards will create an average run time that is uniform. The 5 by 5 board took much longer to solve, at about a ratio of 10:3.