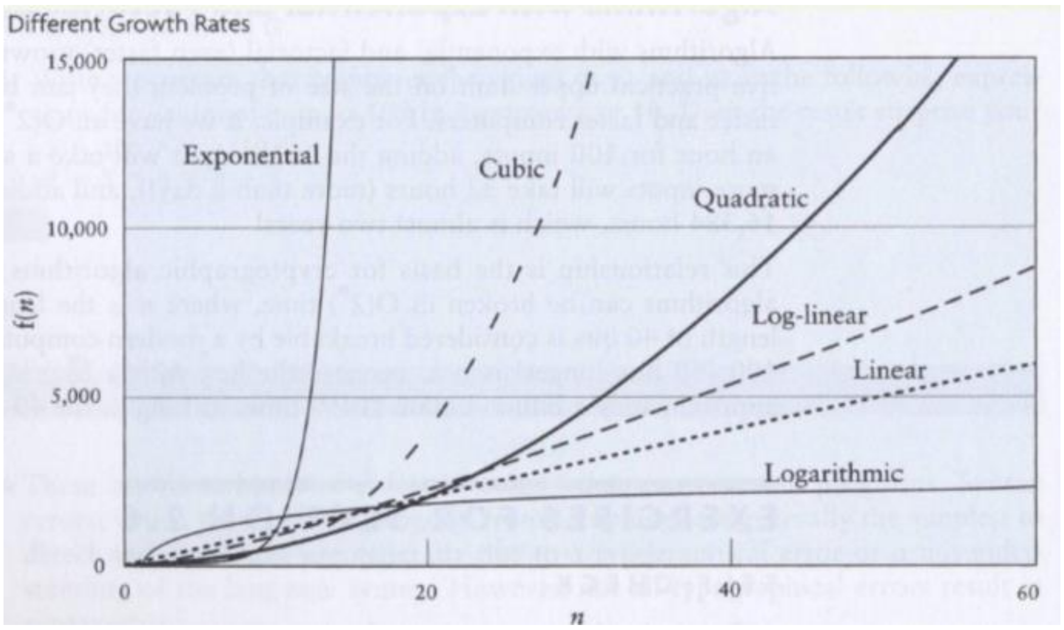
Name: Gareth Quirke
Student number: X00108966
Title: Analysis Report – AC CA3
Date: April 2017

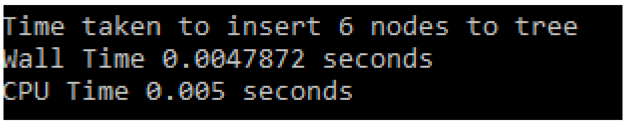
Inserting a new node containing city information

Can this be implemented with reasonable efficiency using a BST?

This operation can be performed in O(logN) time. From timing the processes in the main class I could see that this closely matched the growth rate of the wall clock. Below is a diagram taken from computational theory that shows this on a chart.



Logarithmic is the way this function is implemented. The times recorded below effectively show that this can be implemented efficiently with a BST.

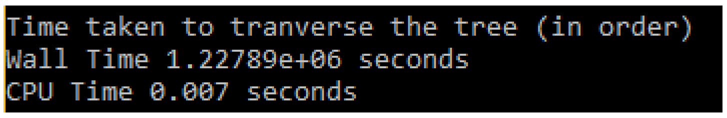


```
Time taken to insert 6 nodes to tree
Wall Time 0.0047872 seconds
CPU Time 0.005 seconds
```

Traversing the tree in order by coordinates

Can this be implemented with reasonable efficiency using a BST?

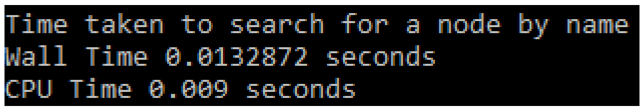
This function can be implemented in linear time O(n). This grows a little faster than adding a new node to the database but it is still efficient. Each node is visited once in the tree.


```
Time taken to tranverse the tree (in order)
Wall Time 1.22789e+06 seconds
CPU Time 0.007 seconds
```

Searching for a record in the database

Can this be implemented with reasonable efficiency using a BST?

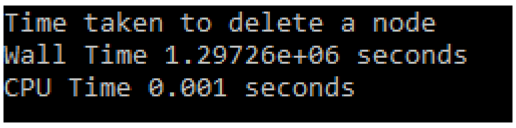
This is another function which can be completed in O(logN) time which is efficient. As we can see from the image below not much CPU time is needed to find a record.


```
Time taken to search for a node by name
Wall Time 0.0132872 seconds
CPU Time 0.009 seconds
```

Deleting a record from the database

Can this be implemented with reasonable efficiency using a BST?

Again, this is O(logN) time and how quickly the node is found & deleted depends on the height of the tree.


```
Time taken to delete a node
Wall Time 1.29726e+06 seconds
CPU Time 0.001 seconds
```

Additional Analysis

Can the database system be made more efficient by using one or more additional BSTs to organize the records by location?

I believe the database can be made more efficient with additional BST in place. The different BST would represent parts of the four hemispheres on earth: northern, southern, eastern and western. As I mentioned above, the time elapsed for executing functions where a node must be found first before an operation can occur greatly increases when the height of the tree is increased. A BST of the entire western hemisphere for example would be too large to maintain and the performance would be effected massively. Particularly in the search function for example if we look at the query we pass in we can narrow down to one region amongst an assortment of BSTs and find the record much faster.

Can any additional or different algorithms/data structures be used, if so what are the advantages and disadvantages?

I did some research into various ways to make a BST more efficient and the most common result was to create a balanced tree by using the DSW algorithm (Day-Stout-Warren) or by implementing an AVL tree.

A BST is balanced when the height is O(log n). A balanced tree ensures that the worst-case time complexity will always be O(log n). If we want to compare a standard unbalanced BST to an AVL – the self-balancing BST here are a few key points to consider:

- Each node has a balance factor meaning the AVL will consume more memory.
- Operations may take a little longer with an AVL as the balance factor needs to be maintained.
- An unbalanced BST has a worst-case scenario of linear.

Reference:
Svick's answer on the following question on stackoverflow
http://stackoverflow.com/questions/14670770/binary-search-tree-over-avl-tree

AVL Trees, MIT OpenCourse
https://www.youtube.com/watch?v=FNeL18KsWPc