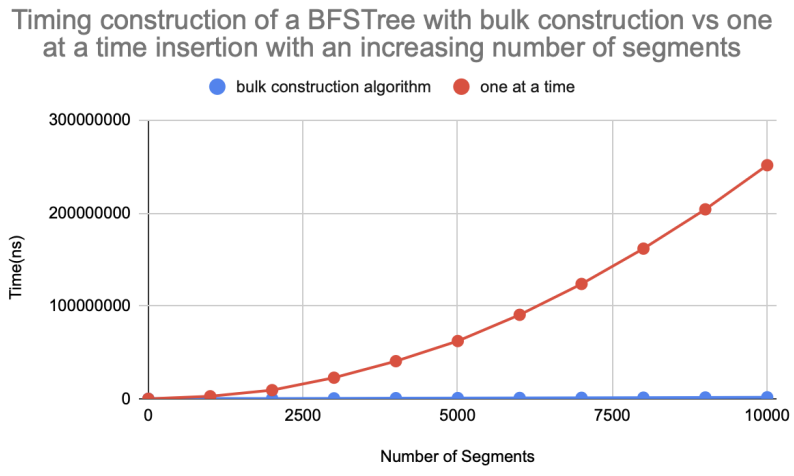
Aiden Pratt Analysis Doc Assignment 09

Construction

Perform an experiment to compare the time to construct a tree with N segments using a worst case input using the "bulk construction" constructor compared with inserting the segments one at a time into an initially empty tree. As an example, consider a bunch of vertical segments. What is the "worst case" order to insert them? Plot the runtimes of the 2 methods for building the tree. Do your experiments match the Big O growth rates you expected?

For tree data structures, the “worst case” for insertion typically occurs when the items are added in an already sorted order that does not allow the tree to be balanced. If these vertical items are sorted by their x index and then inserted in that order, it will result in the tree being heavily skewed one way.

N	optimized	one at a time insert
1	13413.76	177.97
1001	374860.41	2710997.1
2001	298352.48	9303185.84
3001	448779.58	2.27E+07
4001	664774.18	4.05E+07
5001	768074.56	6.22E+07
6001	908933.35	9.04E+07
7001	1071307.95	1.24E+08
8001	1239472.05	1.62E+08
9001	1434877.87	2.04E+08
10001	1624997.46	2.51E+08



Experiment 1 data.

This experiment compares the “bulk construction” algorithm I wrote to build the tree while adding N elements with inserting N elements into the tree one at a time with the insert method. It greatly displays that the one at a time construction is very slow as the number of elements increases. This represents a big O time complexity of O(N^2) as it is a worse case for adding every segment into the tree and must do it N times. The “bulk construction” algorithm displays a big O time complexity of (logN). This is because it does not need to check each element within the tree while adding the elements. The algorithm recursively looks at the segments and splits them when necessary, then creates separate arrayLists for the left and right side of the tree.

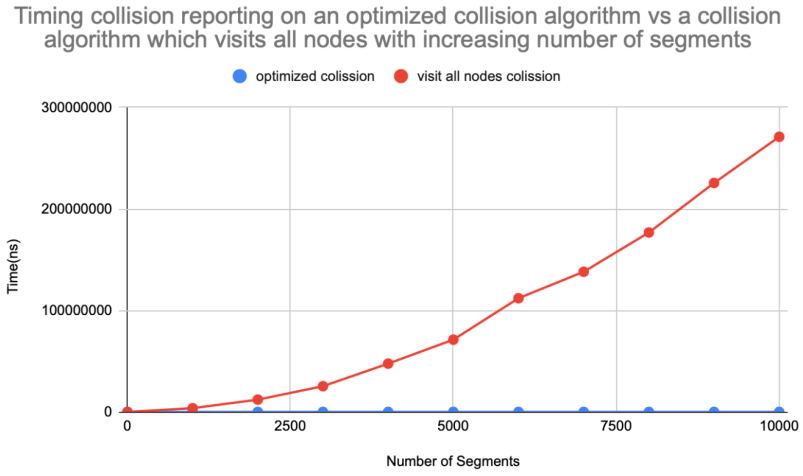
Collision Detection

Design an conduct an experiment to determine the effectiveness of the collision detection algorithm you implemented. Compare the optimized `collision` method you implemented with the following approach:

```
query = // pick a segment to test with, you decide how
boolean collisionFound = false;
tree.traverseFarToNear(x, y, //they don't matter
(segment) -> {
      if(segment.intersects(query)){
            collisionFound = true;
      }
}
```

which will visit all nodes. Does our optimized collision detection routine run in the big O you expected? Be sure to describe the details of your experiment

N	optimized colission	visit all nodes colission
1	872.08	4327.46
1001	369.5	3706152.07
2001	400.33	1.21E+07
3001	354.21	2.54E+07
4001	85.81	4.76E+07
5001	85.85	7.12E+07
6001	92.92	1.12E+08
7001	181.34	1.38E+08
8001	165.41	1.77E+08
9001	665.87	2.26E+08
10001	517.48	2.71E+08



Experiment 2 data.

This experiment compares the optimized collision detection algorithm I wrote with a worse collision detection algorithm which visits all the nodes. The collision algorithm I wrote is shown to consistently find collisions much quicker than the algorithm that must visit every node. My optimized collision algorithm displays a big O time complexity of O(1). This is because the algorithm checks with a parameter “query” segment for collisions, then checks if there is an intersection with this segment. The algorithm which must visit all nodes displays a big O time complexity of O(N^2). This is because it must visit every leaf node in the tree and do this N times.