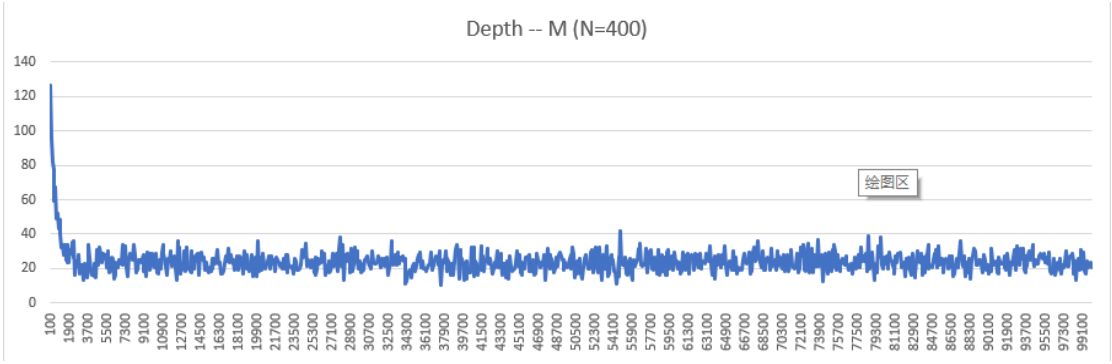
Submission Document

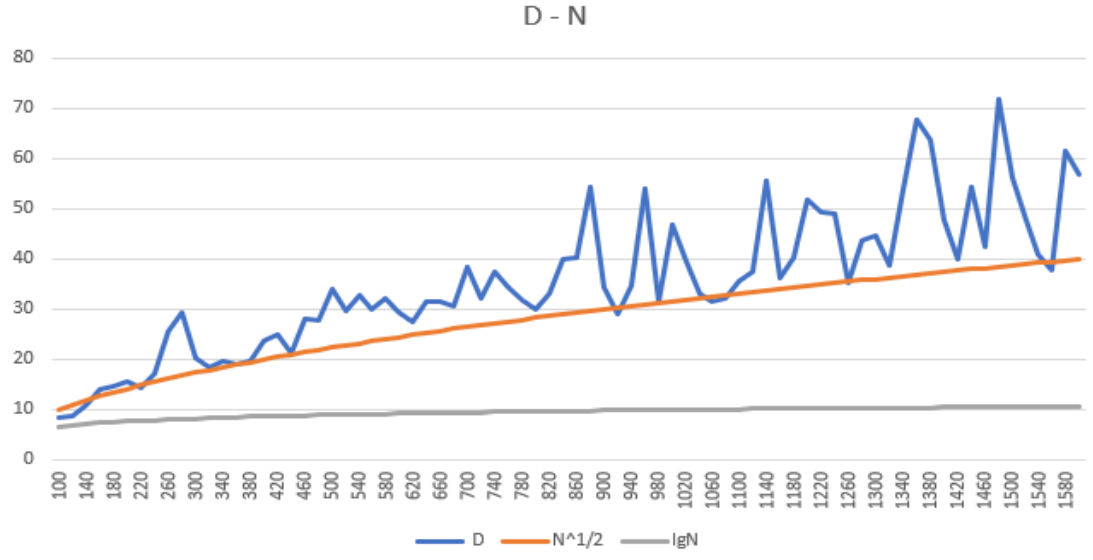
If we define M as the total count of deletion and insertion, N as the maximum value of elements that BST able to contain, after M (randomly delete or insert) operations, the final depth (D) of BST is about $\sqrt{N}$ instead of lgN. In this state, the complexity of each operation (not only deletion and insertion, but also searching) will be $O(\sqrt{N})$ (because of the max steps of each operation are equals to the depth of the tree). This conclusion is reasonable.

To prove this theory through experiment, first, I set N equals to 400. If D gets close to 20 ($20=\sqrt{400}$) as M grows, then I can assert that if M is large enough, a "final state" will reach. In this state, D approximately equals to $\sqrt{N}$. I use Test.java to do this experiment. Result is shown below. As the M grows from 100 to 100000, depth



decent from 120 to around 20, which is the same as my expectation.

After that, I choose a constant M = 100000, give a range of N to see the relation between D and N. I give sequence of M which starts at 100, grows by 20, ends at 1600. I get the result like this:



Obviously, the tendency of D acts more like $\sqrt{N}$ than lgN. There are several common points at D and $\sqrt{N}$, but lgN is much smaller than what we've got here.

Both of those two experiments are repeated 20 times and the D is the average value of them. In random cases, BST rarely reach the ideal lgN depth but much often reach $\sqrt{N}$ depth. There should be a more balance way to build the tree. Of course we have!