Homework – Splay Experiment
Diellor Hoxhaj

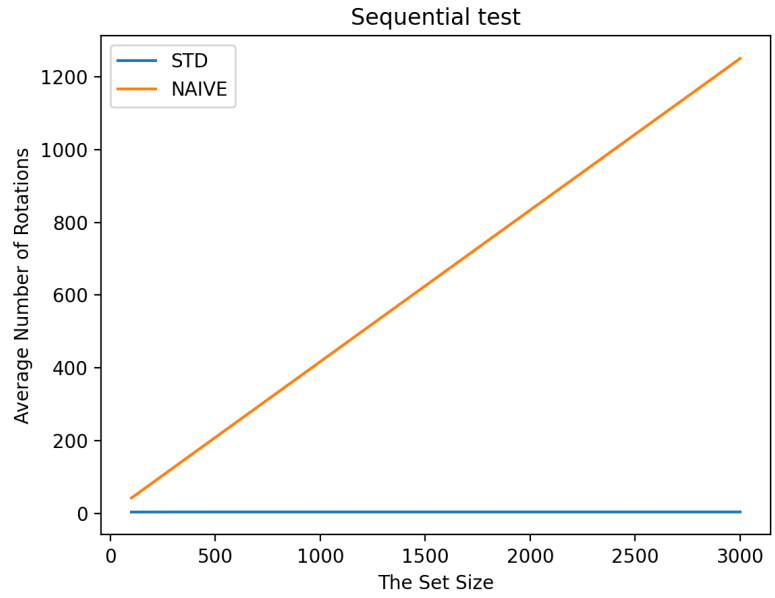

Figure 1- Sequential Test Graph

As we can see from the figure - 1, there is a huge difference between naive and std (using double rotations) solution.

Since Sequential test inserts elements sequentially, this will tree becomes inbalanced and have a height of **n** with **n** being the numbers of values inserted. As a result, the **amortised** Complexity of insertion in naive and standard (std) implementation will be **Ω** (n).

The difference in AVG number of rotations is because using naive splay strategy (only single rotations) we cannot have better amortized complexity then **Ω**(n). -
When finding elements in sequential order, we sequentially splay all the nodes of a path, while using single rotations to get the element to the root the result would look like a linked list again. It's a two linked list a the left side is turning into the original linked list. Therefore, the average depth of the node splayed is approximately n/2 and so the amortized complexity cannot be better than **Ω**(n).

For the find operation in standard implementation, we perform a normal BST find followed by a splay operation on the node found (or the leaf node last encountered, if the key was not found). We can charge the cost of going down the tree to the splay operation. Thus the amortized cost of find is O(log n). We can see the differences in the graph between standard implementation (O log n cost in finding) and naive implementation (O n in finding).

We also proofed this in the lecture, in std implementation the amortized cost will be **Ω**(logn), based on the this theorem that we proved in lecture we can prove this:

Theorem: The amortized cost of Splay(x) is at most $3 \cdot (r\,0\,(x) - r(x)) + 1$, where r(x) is the rank of the node x before the operation and r 0 (x) the rank after it.

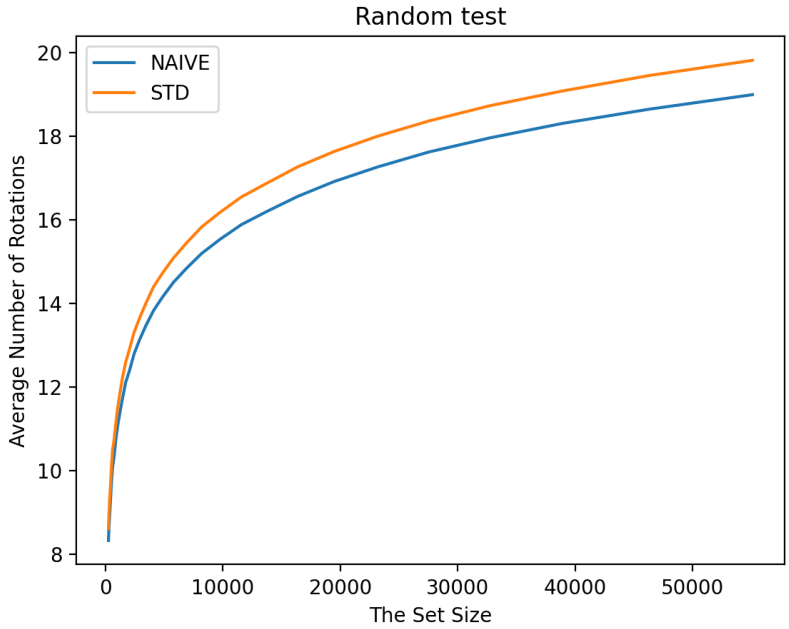

Figure 2 - Random Test

In Figure - 2 for finding 5n random elements, the amortized cost will be O (n). We have similar number of rotation in average. The logarithmic number of number of rotation is because in any unbalanced subtree we rotate elements from larger side. In native implementation the trees balance themselves that's why they perform a bit better.

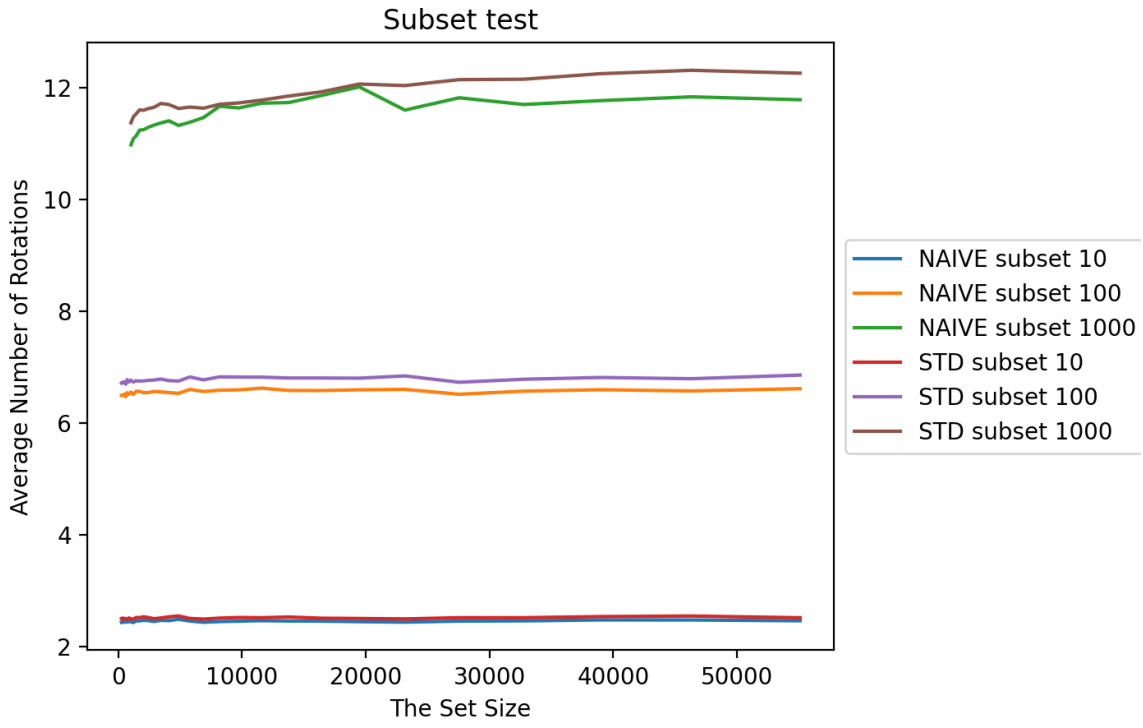

Figure 3 - Subset Test