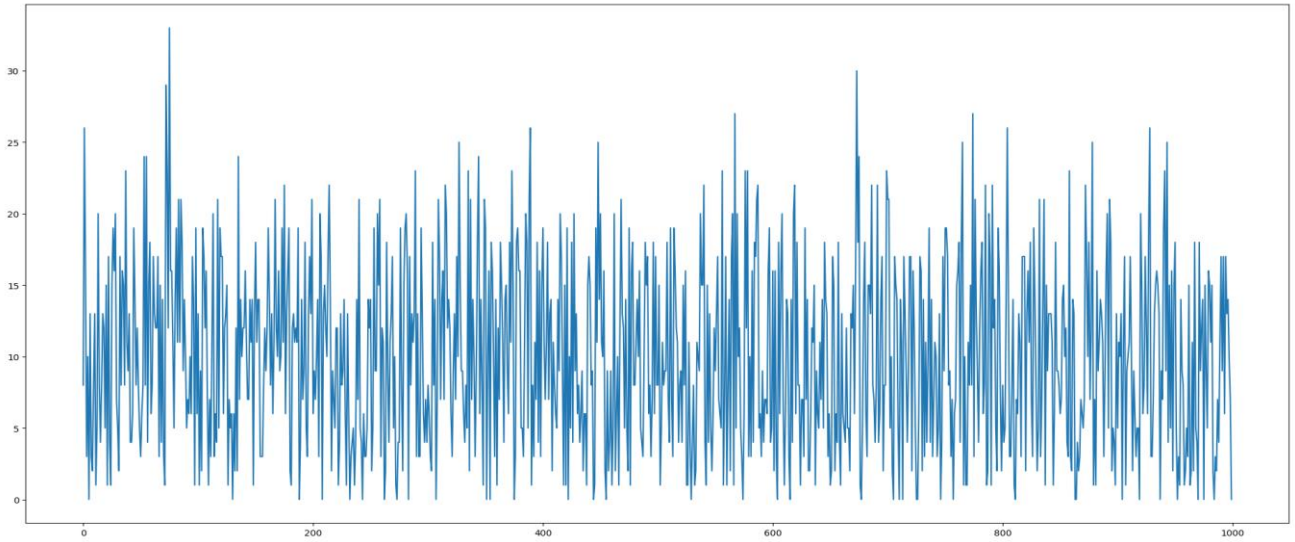
Empirical Score of The Static Competitive Ratio For "Self-Organizing Binary Search Trees"

Venkat Sai Charan Bandi

- Defining the costs for each of the runs were specified in the Paper and state that for each search the cost is the height of the tree +1 to access, and the paid exchanges (Rotation) cost 1. Since there are no insertions and each of our runs are already starting on a randomly generated BST. I used these to measure the costs in my code.
- To make the distributions accurate, all the frequencies generated are run through a random permutation generator as mentioned in the assignment. When I plotted all the frequencies on a visualizer, the below Figure1 was charted. Looking at it, we can see it was very random and equally distributed across so we can assume the scores are not affected by other variables other than the algorithm itself.

FIGURE 1 FREQUENCY DISTRIBUTION OVER 1000 NODES



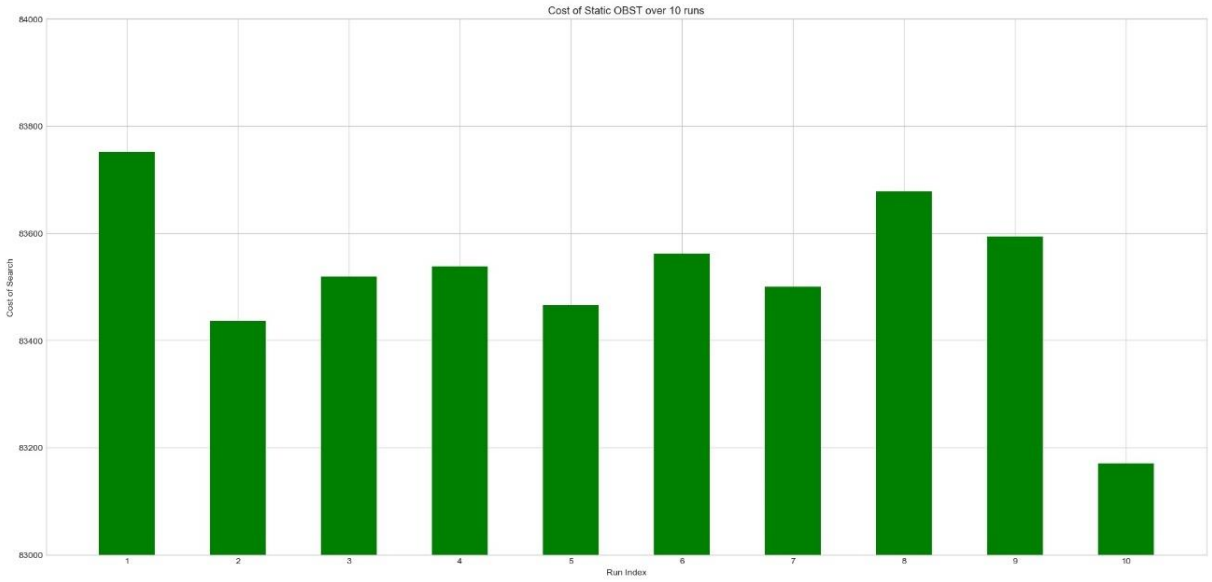
- To calculate the competitive formula, in the paper factor c is called the competitive ratio:

$$C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a.$$

Where the C_{OPT} is the cost for the optimal Offline algorithm in our cast the static OBST.
And C_A is the cost of the on-line algorithm.

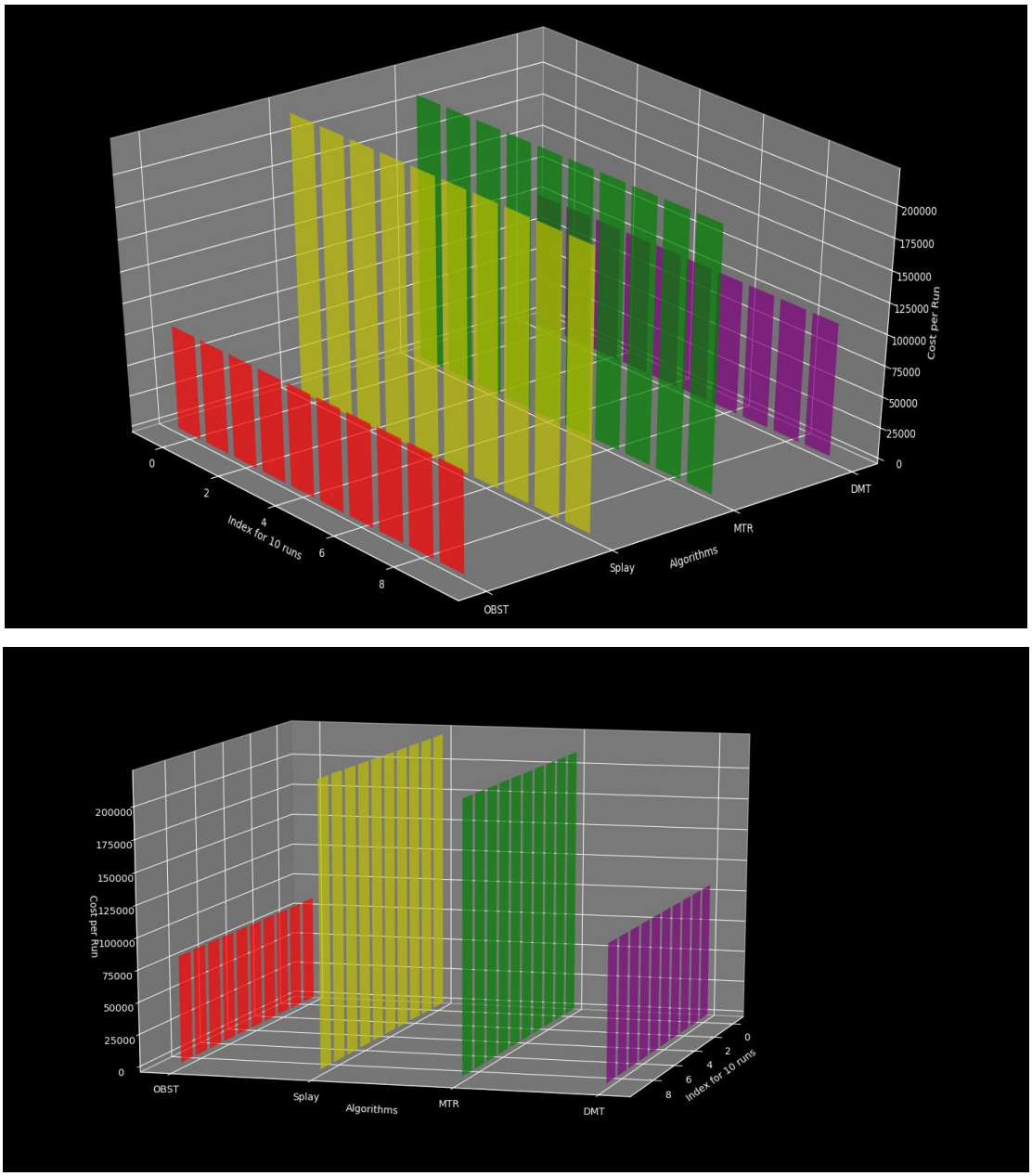
- To start, we fix the cost of the Static OBST and calculate the average, the figure 2 plots that values. The program avg_OBST.py retrieves this and an average of (83521), now we can start with the on-line algorithms to get the score

FIGURE 2 COSTS OF OBST (10 RUNS)

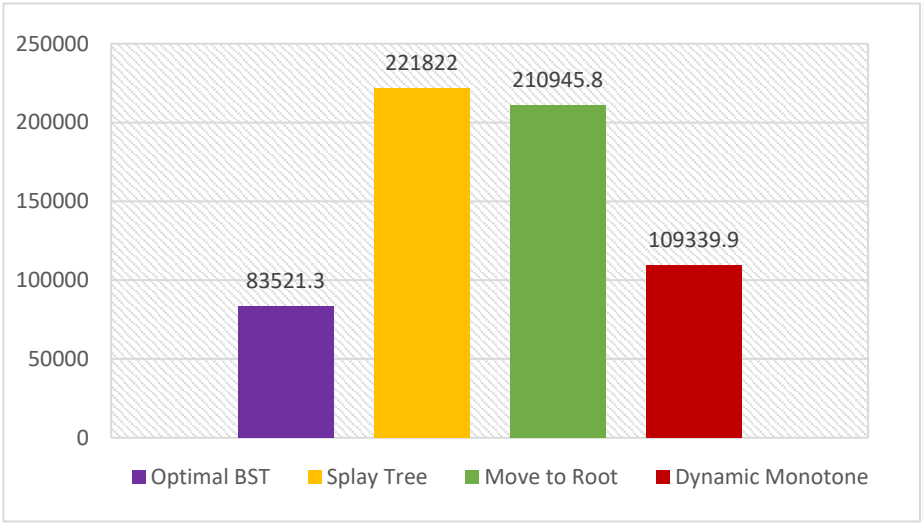


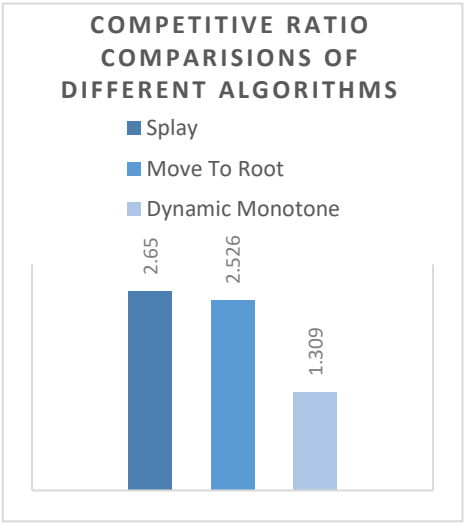
- Upon coding the other three online algorithms and using a visualizer to plot all the 10 * 4 runs to chart the multiple values on a 3d plane, I got the graph below. Running the 3dgraphplot.py can give a better view and angles to better understanding

FIGURE 3 COSTS OF ALL THE RUNS 3D ZYZ AXIS



- Averages of all the algorithms and calculating the competitive ratios, below:





- A example run of the final code giving all results:

```
*** Costs of 10 Runs ***

Optimal BST costs are:                [83751, 83436, 83519, 83538, 83465, 83562, 83500, 83678, 83594, 83170]
Splay costs are:                      [221760, 221800, 221792, 221808, 221810, 221900, 221832, 221796, 221848, 221874]
Move to root costs are:               [210890, 211138, 210892, 210830, 210878, 211032, 210862, 210886, 211058, 210992]
Dynamic Monotone Tree costs are:      [109202, 109435, 109345, 109384, 109367, 109376, 109344, 109341, 109202, 109403]


** Averages of each Algorithm ***

Optimal BST :                 83521.3
Splay :                       221822.0
Move to Root :                210945.8
Dynamic Monotone Tree :       109339.9


*** Static Competitive Ratio ***

Splay Tree :            2.656
Move to Root :          2.526
Dynamic Monotone :      1.309
```

Memoryless Self-Organizing BST's (Splay & Move-To-Root)

- After observing the data from the programs, we can make some practical implications on theoretically published data.
- The paper given contains some theorems about the self-organizing BST's:
- **Theorem 30:** The move to root heuristic is O(1) – distributive competitive as there is constant always in the algorithm and is always more or less in the same cost. As observed from my 10 costs it barely deviates within the range and always takes the same cost to run. Hence, we can say it is an O(1) distributive competitive
- **Theorem 31:** We didn't use sequential access of data as it randomized completely
- **Theorem 38:** The splay tree is O(1) static competitive as from the "dynamic optimality" conjecture, the splay tree does at most a constant factor more work than any Dynamic Binary Search Tree, as we can observe from the costs, it can be proved true.

State Based Algorithm (Dynamic Monotone Tree)

- As we can see from the results the Dynamic monotone tree dominates the search and rotations by almost half cost. The competitive ratio is along the ranges of 1.3 -1.4
- This when compared to the memoryless BST's we can easily call it more efficient in terms of cost when using frequencies and multiple accesses to the nodes of probabilistic randomness.
- As we are not calculating runtimes only the costs. We use more memory in the state based by saving counter values but that does result in less rotations and more efficient searches.
- Although the range of costs in the dynamic monotone is very fluctuant as compare to the constant running Memoryless BST's it always is better by half in terms of cost.
- Although the tree does badly in worst case scenario being all equal distributions it does well probabilistically in Omega (n/log n) hence is (n) competitive when compared to the (1) competitive BST's