Instructions to the Reader

- All of my code has been written in python (3.8.0 64-bit) and I have tested it on 3.7x and 2.7x (anaconda) as well on a windows10 64-bit machine. Below are some technical specifications I have listed to install python dependencies that I have used in some parts of my program and I have mentioned the programs which use them beside
    - Pip install NumPy (3dgraphplot.py, obst.py, avg_OBST.py)
    - Pip install pandas (obst.py)
    - Pip install matplotlib(avg_OBST.py,3dgraphplot.py)
    - Using MPL toolkits which come with Matplotlib by import(3dgraphplot.py)
- The only main run needed to get the results are:
    - final_run.py (gives scores and averages on 10 runs- **only python** with no dependencies is sufficient to run this)
    - 3dgraphplot.py (***BEST*** to run this code and get all the results from the 4 codes and a plotted on a 3d graph for better understanding). But needs installation of NumPy, matplotlib and successful importation of mpl_toolkits.
- Below is a table with all the programs and a brief description what they do:

Req_Func.py	Generates random permutation index, returns the 1000 nodes, returns the access sequence (10000 long) to all the programs in my folder
obst.py	Calculates the cost for static optimal BST using the frequencies generated by Req_Func.py
avg_OBST.py	Plots a graph based on the values received by obst.py over 10 runs and calculates average
splay.py	Runs the splay search on 1000 nodes with access sequence 10000 times and returns the cost
mtr.py	Performs the move to root operation on the nodes and returns the cost function
dmt.py	Performs the dynamic monotone tree search and returns the cost function
final_run.py	Gets all the data from the above four runs and calculates averages and competitive ratios
3dgraphplot.py	Uses the final_run.py and generates a multiple 2d planes on a 3d library

- I have included multiple comments in all of my code, to understand the run better simply uncomment the code and it will give visual pointers on how the code executes (ex: print(root.val), print (“->"), etc)
- I’d also like to point out that my static OBST program takes a lot of time to compile and run (10-15 min per run) since I was working with n-dimensional arrays and that on imported libraries(NumPy and pandas data frame(df)) and the fact that it is a n^3 runtime algorithm over 1000 nodes. So, I felt it was better to run them on my system 10 times and store the values in a txt file (OBST_Results.py) and use a program to calculate the average and deviation across all runs. It only strayed over (700 cost) i.e. 83000 to 83700 hence the results wouldn’t vary by much. I also attached a file (OBST values.pdf) which contains all my 10 runs for proof so you wouldn’t have to go through it. But the code does work perfectly and you can run it any ways