Ryan Filgas
CS202
P3 Efficiency Writeup
Karla Fant

For this project I felt the design of the classes was more organized than the previous projects. Using hierarchies effectively allowed me to reduce the mountains of wrapper functions that plagued earlier implementations. For example, I derived my base class from a node, and had the node manage base class pointers. By working in this way my base class had direct access to node functions, and I didn't need to deal with the inefficiencies of excessive dynamic casting or extra dereferencing in every single node while traversing the data structure. Using operator overloading made function calls a much easier process as well.

In terms of the design being object oriented, the project performed well in some areas, and not as well in others. On the brighter side, I was able to reduce the amount of code significantly using the concepts of OOD. On the darker side, the data structure had getters of course. The part that may have been a violation of the ADT construct is that my client program is derived from the data structure. In this case it meant there was a lot less code to write, but if this were for a client the leg work would have been needed to abstract the data structure more.

I made a few major changes to my design along the way. The purpose of every change was to reduce the amount of work needed. In this project I feel that stepping back to look at design thoughtfully helped me exceedingly more than rushing through the coding process. The first change I made was to derive my data structure from a node, and the second was to derive the menu side of the program from the data structure itself. This provided a lot of flexibility in the implementation and it also allowed for changes on the fly if I decided to move a function from my menu class into the data structure if it made more sense there. Working this way meant the only code I had to change was the scope of the function. From there I was able to copy paste the prototype from one class into the public section of the other.

Efficiency wise I made good decisions and bad. Good decisions were using hierarchical structures to increase efficiency through less dereferencing when traversing the data structure. Implementing a red/black tree was also an efficiency gain, however if I more thoughtfully applied my traversal algorithms, I would be able to take advantage of this even more. My application of a node as a class seemed clunky to me. It involved using a getter to grab the pointer, using that one to get the next etc. It does not seem like a very smooth process, and I would hope there are more efficient implementations out there.

As far as the solution goes, I would have liked to add more functionality. When looking through posts, you want to search, look through thumbnails, and click for more details. To implement this in a terminal, I would have needed to make a function that searches and remembers in the tree where it is or fetches the results into another structure to be sifted through one at a time. I would have wanted to implement a place where the user can store favorites. I also would have implemented more search functionality to allow the user to really narrow down their perfect apartment based on the other data members.

Ryan Filgas
CS202
P3 Efficiency Writeup
Karla Fant

For my implementation I realized a lot of things could have reduced the amount of work and code. I should have used vectors more liberally to avoid long argument lists and explored other constructs to help delegate the work. As far the data structure goes a red/black tree is a good way to increase efficiency. The drawback of this data structure is that it sorts items by a single variable; this does not work well if something needs to be retrieved by username. The best solution to this would be multiple structures sharing common data. Wrapping up, I feel this project was the closest thing to OOD that I have implemented this term and It’s the first one where I feel the concepts of OOD made the code simpler and saved a lot of time.