Term Project
CS 2210 – Fall 2021
Sam DeCook and Brooke Ackley

I. Requirements: Restate the problem specification, and any detailed requirements

Develop and ADT to make a (2, 4) search tree, storing ordered information. You only need to be able to store integers.

II. Design: How did you attack the problem? What choices did you make in your design, and why? Show class diagrams for more complex designs.

We attacked this by working on each of the three methods one at a time. While working on the `findElement()` method, we realized each method would need to be able to search for a node, so we made a `search()` method. We also made other helper methods, like `FFTGE()`, `WCIT()`, `fixOverflow()`, and `fixUnderflow()`, per Dr. Gallagher's recommendations.

III. Security Analysis: State the potential security vulnerabilities of your design. How could these vulnerabilities be exploited by an adversary? What would be the impact if the vulnerability is exploited?

There really aren't any security vulnerabilities, this ADT just stores data and moves it around. There aren't any opportunities to gain access because all users are allowed to do is store data.

IV. Implementation: Outline any interesting implementation details.

Our `search()` method is probably the most interesting implementation detail. `findElement()` needed a node where the element was found, `insertElement()` needed the node where the element would go, and `removeElement()` needed the node with the element to be deleted.

So `search()` would search for a key. If it found it, it would return the node the element was in (used for `findElement()` and `removeElement()`). If it didn't find an element, it would return the last node it searched through, which would be a leaf. This was used for `insertElement()` to find where to insert the item. This one method was really useful for making each of the methods shorter and much easier to understand.

V. Testing: Explain how you tested your program, enumerating the tests if possible. Explain why your test set was sufficient to believe that the software is working properly, i.e., what were the range of possibilities of errors that you were testing for.

We tested our program by inserting 10,000 random numbers into our tree and also into an array. We printed out the tree and made sure insert was working and that everything was hooked up. Then we used the array to remove all the elements. We printed out the last 30 removes to make sure it was working. Since we did this with 10,000 numbers, we are confident our code works.

VI. Summary/Conclusion: Present your results. Did it work properly? Are there any limitations? If it is an analysis-type project, this section may be significantly longer than for a simple implementation-type project.

Our code compiled and ran properly and produced the expected output!

VII. Lessons Learned: List any lessons learned. What might you have done differently if you were going to attack this again.

Listen to Dr. Gallagher's recordings. I (Sam) heard those helped a lot. We also would have tested our code for each method. We didn't do a lot of testing before we had finished remove, and that made debugging much harder.