CSCI 332 Design/Analysis of Algorithms
Exam #1 February 17, 2023 **DUE February 24, 2023**

Please type your answers. Diagrams and equations may be hand-drawn neatly and included in the typed text of your response.

1. *Consider the Spellchecker problem you wrote for this class. The program reads in a file, and then allows the user to search until done or until the program terminates. A user can also be another program that calls the spellchecker. Which of the four types of binary search trees would be the most efficient as the underlying data structure? Why? (30 points)*

   An AVL tree, while reading the date may take some time it only happens once, and then the only function being called is the search which is worst case log(n) and has the fastest look-up of the trees as its balancing is the strictest.

2. *Consider a spellchecker that runs in an editing environment like Microsoft Word. The dictionary is read into a BST structure when the editing program is launched. The spellcheck happens as the user types, and the program indicates an error if the word is misspelled. Further, "misspelled" words (such as names or industry-specific terms) may be added to the dictionary by the user. The word will be added to the current dictionary tree and to the file from which the dictionary words were initially read. Which of the four types of BST's would be most efficient and effective for this problem? Why? (30 points)*

   Probably a splay tree as they pull the words that are frequently used to the top which lowers the time it takes to search for them, this would be worse in edge cases, and adding a new word wouldn't be great, but it means the search time for common words that repeat endlessly (like: the, to, and, etc..) would have much faster search times.

3. *Of the four BST's we studied, which ones use null nodes in their algorithms? Briefly, how are they used? (10 points)*

   Red-black trees, they are used to mark the end of the tree, in a red-black tree all external/leaf nodes are black null nodes.

4. *Despite its shortcomings, the array can be a very efficient underlying data structure for an ADT. Why? Please provide an illustrative example. (5 points)*

   Because you can reach any point in the array in constant time making it very easy to find any stored value. For example in a list to get the item at an index you can just go to that point in the array compared to a reference based implementation where you need to loop through from the beginning of the list to reach the target index.

5. *In the context of algorithmic analysis, what is amortization? Provide an example. (5 points)*

   It is used on algorithms which have a rare worst case scenario which is less common to give a more accurate analysis of the algorithm. For example, in an array-based implementation of a list insert will take more time on the occasions that it needs to resize, but that is not common, it is much more common that it will be constant time, so amortization gives a result that reflects that.

6. *Consider the recursive factorial algorithm. Explain (in words and math) how to calculate the time complexity of the recursive factorial. Based on this, what might you conclude about recursive programs? Please cite your sources. (10 points)*

   To find the complexity of a recursive problem one must find a function describing the complexity relative to the previous step in the recursion, for the factorial definition this works like

   T(n) = T(n-1) + 3 (because there are three constant time functions in the algorithm)

   T(n) = T(n-2) + 6 (continuing)

   T(n) = T(n-3) + 9 (Clearly there is a pattern forming)

   T(n) = T(n-m) + 3m

   T(n) = T(n-n) + 3n

   T(n) = T(0) + 3n

   T(n) = 1 + 3n

   Time complexity is O(n). Not really sure what to conclude about recursive algorithms, my only thought is that recursive algorithms have a complexity equal to n to the power of the number of times the algorithm calls itself. Not really sure, it is hard to make a solid conclusion from one data point, but I believe the recursive Fibonacci algorithm is n^2 and includes two recursive calls so it lines up.

   (Source: https://stackoverflow.com/questions/2327244/complexity-of-recursive-factorial-program#:~:text=T(N)%20is%20directly%20proportional,complexity%20is%20O(N).)

7. *A busy retail store has two checkout stations and is considering adding a third. The checkout area is designed so that the customers wait in a single line until a checkout station becomes*

available. Then, the customer at the front of the line proceeds to the available station. Data stored about each customers includes arrival time (when they get in line) and departure time. Given this information, explain how you would design a simulation to determine how many more customers can be served with a third checkout. Your response should include an explanation and may include diagrams and pseudo code. (10 points)

Start with a queue that that mimics customers getting in line and then working their way up to the front. You enqueue the customer and all their items when they get into line, and dequeue from the queue when a checkout station becomes available. The most accurate way to simulate the rest would be to create a multithreaded checkout function that waits for a time based on the items stored in the customer object and allow for three threads at a time. Because multithreading is difficult and I don't remember how to do it, it would be much easier to do with an int[3] array and an update function, each loop of the update function decreases the value of the integers in the array and whenever an integer becomes zero it deques the first customer from the queue and sets the integer that just became zero to the number of items that the customer has. You can use the arrival time and departure time to measure average time spent in line.