Administrative

Team Name: XGW
Team Members: Christian Walk, Isha Gupta, Jenny Xie
Link to Video: https://youtu.be/yp0SVRM2eG0

Extended and Refined Proposal

Problem: The problem we are trying to solve is creating and having a safe password. Some people use too common or too simple of a password and their online accounts are therefore not as secure. Our project aims to check the strength of people's inputted password by comparing it against a list of most commonly used passwords. We also implemented other special features such as the generation of a random password, adding other weak passwords not on our list, and a toggle button where the user can decide which data structure to search through.

Motivation: This is a problem because data breaches are often occurring, so a user needs to create a secure and unique password that cannot be easily hacked. If they are using a common password, like 123456, then it is likely that their account will be compromised, so our project will help the user generate a unique password for their account to ensure that the user's personal information is kept secure.

Features Implemented: Generating a strong password, checking a user inputted password against the map or tree which can be toggled depending on the user. Allows the user to add a weak password into the map and tree, and retrieve the number of accounts associated for a password.

Description of Data: We will be using a public data set [https://wiki.skullsecurity.org/Passwords] that contains ~14 million passwords that have been used by ~32 million accounts. However, we will only be using 100,000 of the unique passwords.

Tools/Languages/APIs/Libraries used: We used C++, Visual studio, and Clion. We use STL library: string, map, iostream, fstream.

Data Structures/Algorithms implemented: We implemented a binary AVL tree, that contains ~100,000 unique passwords from our data set. To do this, the ~100,000 unique passwords will be sorted by ASCII value (string compare() function) into the AVL tree. We also implemented a binary search function to search into this tree for a specific password.

Data Structures/Algorithms used: We used a map from STL library that stores the password as the key and number of times that it has been used to be the value. We also use the find function for the map.

Distribution of Responsibility and Roles:

For this project, Ishita used a map that stored the number of times a password has been used as the value and the password data as the key which came from a data file. All of us worked

together to insert each password from the map into a binary sort tree, specifically an AVL tree based on the ASCII values of each type of character as well as length of passwords where a shorter password was considered “less” than a longer password. Ishita was responsible for creating additional features such as the generation of a random strong password for the user. Christian took care of creating the AVL tree data structure and implementing rotational methods to keep the tree balanced everytime we add to it and implementing the insertion method. Jenny worked on the search algorithm to effectively iterate through the AVL tree when comparing an inputted password to the weak password data. We tested our individual portions but then tested them all together at certain checkpoints. Jenny and Ishita worked on the user interface and calling the proper functions based on user input. We all worked on the documentation for this project and calculated the time complexity. We all also worked together to create the video for the program.

Analysis

Changes Made:

We decided to include more options in our Password Menu. The new features that we added are being able to toggle searching for a password in the map or tree and adding a weak password to our databases of passwords. We wanted these new features because they seemed relevant and useful to the related topic of having a strong password.

Complexity Analysis of major function/features:

Function/Feature	**Complexity Analysis in Big O (worst case)**
Search() (tree implementation)	Time: **O(1)** Space: **O(h)** where h is the height of the tree
getBalanceFactor()	**O(1)** calculates and returns balance factor
insertNode()	Time: **O(1)** reassigning pointers Space:**O(h)** where h is height of tree because that is the max amount of nodes it will go through to insert
rotateLeftLeft()	**O(1)** reassigning pointers
rotateRightRight()	**O(1)** reassigning pointers
balance()	**O(1)** calls rotate methods but those are O(1)
Generating a Strong Password (in main)	**O(s + c)**, s is a list of all special characters c is combined total of inputted characters for each category
Checking the strength of a password	Time:**O(log n)** Calls search function of map Time: **O(1)** calls tree search function Space:**O(h)**, height of the tree (recursive)

Add a weak password to the database	Time:**O(log n)** inserts password into map Time: **O(1)** inserts into tree Space: **O(h)**, height of tree (recursive)
find() (of map)	**O(log n)** where n is the number of elements in the map
Get number of accounts associated for a password	Time: **O(log n)** searches through map by calling find function
Toggle data structure	**O(1) -** changes boolean
Exit	**O(1)** exits out of while loop
Entering data from file into map	**O(n*m),**n is the number of passwords and m is the length of characters in an input line(password length and number of accounts used length)

Reflection

Overall Experience:

Good learning experience especially working in a group on a coding project compared to the normal project of working on your own. The experience would have been better had we been able to meet up as a group and work on it together in person rather than through zoom calls and messaging. This experience entailed a segment of creativity which added to the overall experience and thus our project became more of a personal project where we implemented our own standards of how the program should work and look like.

Challenges:

Due to not seeing each other in person it was difficult to always be on the same page. Also estimating the amount of work each section takes is quite difficult. The guidelines were rather vague and therefore there was a lot of room for creativity but also quite difficult to nail down all the specific requirements.

If we started over, would there be any changes?

We would take more time to evaluate each section of the program and the time it would take to develop them so that the roles could be more evenly distributed. Implement a different data structure to compare complexities with. We would meet up more often and check progress/ check our different parts of code together.

What have you learned through this process:

We have learned that communication is critical when working in groups, especially with comments in the code so that other people reading it can easily understand the functionality. Working on a group project is different from working individually and a better reflection of real

computer science careers. When working with larger data files or big data it is important to really think through and look at the process because time complexity can really add up. Ensuring that you have a well configured and consistent data stream is important so that your program can handle each type of data.

References

https://wiki.skullsecurity.org/Passwords
This website consists of 14,341,564 unique passwords, used in 32,603,388 accounts. We will be using 100,000 of these unique passwords to determine whether a user's choice of password is strong enough and tough to crack. Each of these passwords will have a certain value associated with it that corresponds to the number of accounts that uses the password.