The purpose of Lab 1-C was to implement a recursive method that compared a hashed string to a text file containing 100 usernames, salt values, and hashed passwords.

The algorithm to convert a string into a hash string was given. It used sha256 hashing. Firstly, a method generated all possible permutations of a given set of characters (0 to 9), which fit the minimum and maximum criteria of characters given. I then had the program read the text file with all usernames and passwords and create an array where each item was each line in the file.

Before continuing on the main method, I solved the method that would actually compare the two, hashed strings. I found it easier to implement if-then statements and initialize the variable I outside the method in order to keep track of the number of lines that had been checked. I did this rather than using one or more loops (for or while), or a separate class object, which I found confusing. I had to separate each item in the array of the full text by each comma. I did this using a temporary array where item 0 was the username, item 1 was the salt value, and item 2 was the hashed password of a given line. I then created a new hashed password using the string provided and the salt value of the line together, using the sha256 method.

I checked for compatibility between the hashed strings using an if-else statement, where it would print the password and the username if it found them to be compatible with the original hashed password, or else it would do the recursive call, jumping to the next line. I later got rid of the else statement but kept the recursive call and added a base case where I used the variable i. The method runs through the whole list in the file before returning.

The method they returned to was what I developed next. It was where the actual string of passwords would be generated. I tried several itertools for this, including combinations, permutations, and combinations\_with\_replacement, but they all left out several possible password combinations, so I finally settled on .product(). The method generated all possible permutations of the items on the list given with a length of k. I set it up so that if k was less than the variable min, then the passwords would not be carried over to the comparison method. I had to modify the actual resulting strings from the product method because they were printing with brackets and commas. I used .join() for this.

Essentially, a string of k length would be generated and then compared to all possible hashed passwords. This causes the code to run a bit long as it consists of a recursive method inside another recursive method.

I ran into some trouble when two identical strings kept returning a false statement. I solved this by searching online. Someone mentioned that in the text file, the last item separated by a coma also contains a \n, although invisible. So I used the .rstrip() method to remove it before comparing the strings. Another issue was the runtime, which I marginally shortened by making a copy of the lines array and then deleting each item as it was paired up with a password, making the list shorter and shorter as the password combinations increased.

I then added my values for testing my code, including a minimum and maximum number of characters, and an array of possible characters. I added a print statement for when the code was finished.

This is a table of my testing input values and their results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **min** | **max** | **items** | **TERMINAL** | **Runtime (s)** |
| 3 | 7 | ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] | Password 017 found for User71  Password 113 found for User21  Password 144 found for User27  Password 185 found for User43  Password 294 found for User38  …  Password 9562452 found for User37  Password 9861315 found for User65  COMPARISON COMPLETE | 630.361368894577 |
| 3 | 7 | ['0', '1', '2', '3'] | Password 113 found for User21  Password 1130 found for User47  COMPARISON COMPLETE | 9.657283067703247 |
| 2 | 3 | ['0', '1', '2', '3'] | Password 113 found for User21  COMPARISON COMPLETE | 0.03473711013793945 |
| 2 | 4 | ['0', '1', '2', '3', ‘a’] | Password 113 found for User21  Password 1130 found for User47  COMPARISON COMPLETE | 0.2953658103942871 |
| 2 | 4 | ['a', 'b', 'c', 'd', 'e'] | COMPARISON COMPLETE | 0.3218400478363037 |
| 1 | 2 | ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] | COMPARISON COMPLETE | 0.041571855545043945 |
| 0 | 3 | ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] | Password 017 found for User71  …  Password 960 found for User15  COMPARISON COMPLETE | 0.39180898666381836 |
| -2 | 4 | ['0', '1', '2', '3'] | Password 113 found for User21  Password 1130 found for User47  COMPARISON COMPLETE | 0.14356184005737305 |
| -2 | -4 | ['0', '1', '2', '3'] | COMPARISON COMPLETE | 0.00022721290588378906 |

I mainly learned to solve things in parts rather than linearly. Getting stuck in one place may be solved by leaving it and getting back to it after solving another. I wasted time trying to finish my main method before fully finishing the comparisons method. I also realized I should check each piece of code I add with a test run, rather than waiting until I have everything. This way I can immediately solve each issue.

I made a last-minute change to the algorithm and added a method to track the runtime in seconds.

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

*import* itertools  
*import* hashlib  
*import* time  
  
*def* time\_in\_seconds(items, min, max):  
 start\_time = time.time()  
 main(items, min, max)  
 elapsed\_time = time.time() - start\_time  
 *print*(elapsed\_time)  
  
*# Transform string into hashed password  
def* hash\_with\_sha256(str):  
 hash\_object = hashlib.sha256(str.encode('utf-8'))  
 hex\_dig = hash\_object.hexdigest()  
 *return* hex\_dig  
  
*def* generateString(items, min, max, k, lines):  
 *if*(k > max):  
 *print*("Comparison Complete")  
 *return  
 else*:  
 *# Generate all possible permutations of given items* perms = list(itertools.product(items, repeat=k))  
 *for* j *in* perms:  
 *# .join() removes brackets and commas from perms* newStr = "".join(j)  
 *# Do not hash and compare strings smaller than min  
 if* len(newStr) >= min:  
 comparePass(lines, newStr, 0)  
 generateString(items, min, max, k+1, lines)  
  
*# Compare recursively hashed passwords  
def* comparePass(lines, str, i):  
 *if* i >= len(lines):  
 *return  
 # Split each line by each comma and save in temporary array length 3* temp = lines[i].split(',')  
 *# Transform salt value (temp[1]) and password into hashed password using sha256* codedPass = hash\_with\_sha256(str+temp[1])  
 *# Remove \n from split item to avoid errors* hashedPass = temp[2].rstrip("\n")  
 *# Print each password found with Username  
 if*(codedPass == hashedPass):  
 *print*("Password " + str + " found for " + temp[0])  
 *del* lines[i]  
 comparePass(lines, str, i+1)  
  
*def* main(items, min, max):  
  
 *# Read file and separate each line into an array  
 with* open('password\_file.txt', 'r') *as* textFile:  
 ogLines = textFile.readlines()  
 lines = ogLines  
  
 k = 0  
 generateString(items, min, max, k, lines)  
  
*#Password length limits*min = 3  
max = 7  
  
*# Characters allowed in password*items = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']  
  
time\_in\_seconds(items, min, max)