CMP SCI 5732 Project 3

Jamie Harris

For both of these tasks, I used the sha256 function from the `hashlib` python library. This function makes it very easy to get digests in both binary and hex forms.

Task 1

The most important specs of my system as it relates to this project are my RAM of 32GB and my CPU clock speed of 3.7 GHz. For this task, I created two files filled with random bytes: one 32 byte file, and one 2 kilobyte file. I then hashed each file 10,000 times, tracking the amount of time each hash took in order to get an average time.

hashCollision.py

```
from hashlib import sha256
from time import time_ns
from math import floor
import random

def generate_files():
    with open(file="small_file", mode="w+b") as small_file:
        small_file.write(random.randbytes(32))
    small_file.close()

    with open(file="large_file", mode="w+b") as large_file:
        large_file.write(random.randbytes(2048))
    large_file.close()

def hash_files():
```

```
    with open(file="small_file", mode="rb") as small_file:
        small_file_content = small_file.read()

    with open(file="large_file", mode="rb") as large_file:
            large_file_content = large_file.read()
    small_hash_total_time = 0
    for _ in range(0, 10000):
        start_time = floor(time_ns() / 1000)
        sha256(small_file_content)
        end_time = floor(time_ns() / 1000)
        small_hash_total_time += (end_time - start_time)
    small_file.close()
    small_hash_avg_time = small_hash_total_time / 100

    large_hash_total_time = 0
    for _ in range(0, 10000):
        start_time = floor(time_ns() / 1000)
        sha256(large_file_content)
        end_time = floor(time_ns() / 1000)
        large_hash_total_time += (end_time - start_time)
    large_file.close()
    large_hash_avg_time = large_hash_total_time / 100

    print(f"Small Hash took an average of
{small_hash_avg_time} microseconds")
    print(f"Large Hash took an average of
{large_hash_avg_time} microseconds")

generate_files()
hash_files()
```

Results:

Hashes on 32 byte file: 10000 hashes with average hash time of 0.048 milliseconds

Hashes on 2 kilobyte file: 10000 hashes with average hash time of 0.149 milliseconds

Task 2

For this task, I created an address bit string that is 32 bytes in length. I started it using the `bytes` constructor in python, which gives an `n` byte-long string of 0's. I then created an initial hash and got the hex digest from the hash. Since my birthday is May 14, we would look for the address starting with 05141997. For the first loop, I only check to see if the hex string (minus the initial "0x") starts with 0. I started a timer to see how long this took. After this step was done, I did the same, but checked if the hex string started with 05.
I ran this program multiple times to get an approximate range of how long it might take to get 1 character and 2 characters.

vanityHash.py

```
from hashlib import sha256
from time import time_ns
from math import floor

def create_vanity():
    # Initialize 32 bytes of 0
    address = int.from_bytes(bytes(32))

    address_hash = sha256(address.to_bytes(length=32))
    # Hex strings start with 0x
    hash_str = address_hash.hexdigest()[2:]

    start_time = floor(time_ns() / 1000)

    while hash_str[0] != "0":
        address += 1
        address_hash = sha256(address.to_bytes(length=32))
        # Hex strings start with 0x
        hash_str = address_hash.hexdigest()[2:]
```

```
    end_time = floor(time_ns() / 1000)

    print(f"Time to generate 1 digit of vanity address:
{end_time - start_time} microseconds")
    print(f"---------------")
    start_time = floor(time_ns() / 1000)

    while hash_str[0:2] != "05":
        address += 1
        address_hash = sha256(address.to_bytes(length=32))
        # Hex strings start with 0x
        hash_str = address_hash.hexdigest()[2:]

    end_time = floor(time_ns() / 1000)

    print(f"Time to generate 2 digit of vanity address:
{end_time - start_time} microseconds")

create_vanity()
```

Results

Matching first digit ("0") -> Between 30-50 microseconds

Matching first 2 digits ("05") -> Between 100-200 microseconds

It takes approximately 3-4 times longer to match 2 digits than to match one digit. Given this, we can make an estimate that to match the first 3 digits ("051"), it would take approximately 300-800 microseconds, and matching the first 4 digits ("0514") could between 1000 and 4000 microseconds.