Joseph Pennington (2912079)

EECS 565 Project 1 Report

**Program Output (Passwords & Times)**

**1)** Cipher Text: MSOKKJCOSXOEEKDTOSLGFWCMCHSUSGX
Key Length: 2        firstWordLength: 6

Plain Text: CAESARSWIFEMUSTBEABOVESUSPICION
Key: KS
Time: 0.21900725364685059 seconds

**2)** Cipher Text:
PSPDYLOAFSGFREQKKPOERNIYVSDZSUOVGXSRRIPWERDIPCFSDIQZIASEJVCG
XAYBGYXFPSREKFMEXEBIYDGFKREOWGXEQSXSKXGYRRRVMEKFFIPIWJSKF
DJMBGCC
Key Length: 3        firstWordLength: 7

Plain Text:
FORTUNEWHICHHASAGREATDEALOFPOWERINOTHERMATTERSBUTESPECIAL
LYINWARCANBRINGABOUTGREATCHANGESINASITUATIONTHROUGHVERYSLI
GHTFORCES
Key: KEY
Time: 8.548305034637451 seconds

**3)** Cipher Text: MTZHZEOQKASVBDOWMWMKMNYIIHVWPEXJA
Key Length: 4        firstWordLength: 10

Plain Text: EXPERIENCEISTHETEACHEROFALLTHINGS
Key: IWKD
Time: 205.35800290107727 seconds

**4)** Cipher Text: SQLIMXEEKSXMDOSBITOTYVECRDXSCRURZYPOHRG
Key Length: 5        firstWordLength: 11

Plain Text: IMAGINATIONISMOREIMPORTANTTHANKNOWLEDGE
Key: KELCE
Time: 3950.9282610416412 seconds (~65 minutes)

**5)** Cipher Text:
LDWMEKPOPSWNOAVBIDHIPCEWAETYRVOAUPSINOVDIEDHCDSELHCCPVHRPOHZUSERSFS
Key Length: 6 firstWordLength: 9

Plain Text:
EDUCATIONISWHATREMAINSAFTERONEHASFORGOTTENWHATONEHASLEARNEDINSCHOOL
Key: HACKER
Time: 166556.8705496788 seconds (~46.25 hours)

**6)** Cipher Text: VVVLZWWPBWHZDKBTXLDCGOTGTGRWAQWZSDHEMXLBELUMO
Key Length: 7 firstWordLength: 13

Did not attempt. See explanation below.

**Discussion**

In general, the brute force Vigenere cipher cracker I developed was most efficient when the key length was less than or equal to 4. Once the key length increased, the time needed to test every possible key increased exponentially. For example, testing a key length of 5 only required about an hour to fully test each key. However, to test a key length of 6, the program took nearly 2 days. This was why I decided not to attempt the last cipher text. From timing my program, it could complete 100,000 keys per minute. By doing a simple calculation, the program would take upwards up 56 days to test all $26^7$ keys.

To decrypt cipher texts with larger keys, program efficiency is extremely important. However, looking at my results, it is easy to see that my program was not the most efficient. Program performance can be the result of many different factors. One of the largest ways that program efficiency can increase is by using the appropriate data structure. My password cracker utilized lists to handle most of the data which may not have been the best data structure due to the large amount of data that needed to be stored and repeatedly indexed. Another large factor of program efficiency is with the overall algorithm and logic. Essentially, my program used a triple-nested for loop to generate a key, decrypt the first word, and check if that word was in the dictionary. For loops, in general, are very time consuming, and it would be wise to avoid them when possible. Lastly, computing resources and the programming language used can affect the efficiency. I created my project using Python, which is less efficient than C++, while also only having the available computing resources on my laptop. If I were to use the School of Engineering's computing resources, the program may have run faster.

Overall, brute force password cracking is effective when the key length is low. On the other hand, this can be offset if one uses an efficient algorithm with adequate computing resources.