Assignment 1.2 - Password cracking. (Net ID - rg3595)

1. The technique I used to obtain the passwords was using a password dictionary with the list of common passwords (https://www.kaggle.com/wjburns/common-password-list-rockyoutxt) to crack the passwords.
The given hash list for eharmony seemed to be the easiest to crack. Most of the passwords (for sha-1 and sha-256) that I cracked from the latter two lists (linkedin and formspring respectively) were salted. However, I did come across about 5-6 passwords in both of those text files that were not salted. I was able to verify this, assuming that there were no collisions involved here. I used salt of size 3 with all the passwords I've showed as cracked for sha1 and sha2. The function I've written for cracking passwords outputs the cracked passwords from different files (in order) while running the same program. The arguments passed to that function are relevant to that particular hash function and corresponding hash dump only. One trick that came in handy was using an empty string as the salt for MD5 that enabled me to write such a function. The program outputs the cracked passwords automatically into a file alongside the respective cracked hashes for each given hash dump. The key to speeding up my program came from choosing the right data structures for comparing my results. The number of passwords would have been significantly higher had I tried more variants with respect to placing the salt.

2. **Other techniques that I considered using** were brute force attack and reverse lookup using rainbow tables. Using brute force was like executing an exhaustive version of the dictionary attack. Along with brute forcing, using rainbow tables meant that the probability of cracking a password was lower when compared to the dictionary method, for the same duration of time. These techniques would probably yield bigger results but weren't feasible for me as they seemed to require a more powerful machine. Given more time and better hardware, it's a definite possibility.

3. **The passwords were stored** in the form of hashes. One can conclude the type of hashing function used on the password just by looking at the lengths of these hashes.
Eharmony hash dump contained MD5 hashes (length of 128 bits, meaning 16 bytes).
Linkedin hash dump contained SHA-1 hashes (length of 160 bits, meaning 20 bytes).
Formspring hash dump contained SHA-256 hashes (length of 256 bits, meaning 32 bytes).
The passwords from Linkedin and Formspring contained salt which was stored as a part of the password. **The difficulty levels** when it came to cracking each one of them were different. MD5 (for eharmony) seemed to be the easiest amongst all as these passwords were hashed directly with no salt. In contrast, linkedin (sha1) and formspring (sha256) are a bit more difficult as these lists contained salted passwords. For each password taken from the password dictionary file, every combination of the salt (for characters as shown in the code) was added and hashed and then compared. The running time grows significantly with the size of the dictionary file and the kind of the salt used while hashing these passwords. According to me, cracking these passwords came about in an increasing order of difficulty with MD5 at the lowest and sha-256 at the highest. One reason was the usage of salt and the second reason was the probability of cracking a password for a hash of that length (32 bytes vs 20 bytes vs 16 bytes).