**For Task-2**

* We know the load factor is = (number of items in the table)/table size.
* Increasing table size (while number of items is half or less than half of the table size) decreases the load factor - i.e. when the table is relatively empty - the chance of a collision is small and the operations are `almost' constant time. When the load factor is high, the operations can even degrade to linear time (if collisions are resolved in an unsophisticated way).
* Also increasing table size allows more options to be available in the table by doing mod operations in our hash function which is

value = (ord(key[i]) + a\*value) % self. Table\_size

* As more positions can be obtained from the hash function, there is less chances of primary clustering to be formed as less collisions will occur. Also when less collisions occur, there are lesser clusters formed in our hash table and we do not need to probe many array positions when a collision occurs. Probe chain length are smaller. We are able to find an empty spot faster when inserting and this reduces time to insert each element which hence reduces overall time to insert all elements. Hence run-time is lesser when table size is generally increased.
* Generally, for values of b, it is seen that changing values of b doesn’t affect the run-time that much for each of the three files.
* However, as our universal hash function uses b to make the hash function pseudo random. hence arithmetic operation a = a \* b % Table\_size is computationally expensive when both b and table size is very large and will definitely take time to be processed by CPU. Hence this may increase run time.
* Also, b will make the run time better if it is a large prime but not too large as it will help to generate values of “a” which will hence have lesser common factors with the table size
* For b=250726 and table size = 250727, it takes a lot of time to calculate total time. After analyzing the hash values that are generated from the hash function they are:



* All the hash values are between 97 and 122. So this shows that due to poor choice of value of b (in combination of the choice of table size value), the hash function returns table positions in a very short range in such a big hash table. This will cause keys hashing to the same position to collide very very frequently causing primary clusters to grow in size very fast. Hence increasing length of probe chains. Hence affecting insertion time of each (key, value) pair into the table. That is why insertion takes very long time for this value of b = 250726 and value of table size = 250727.
* The English\_small file has less collisions compared to the other two files due to less items to be inserted into the hash functions, so the hash function is called less times and due to fewer items, there are lesser collisions. Hence runtime is lower compared to the English\_large and the French file.

**For task-3**

* As can be seen from the previous analysis and from the graphs, it is seen that as collisions increases, time taken for insertion increases due to increase in primary clustering which increases the probe chain length causing more time for each (key, value) pair to find an empty spot.
* Also as discussed earlier, as table size is increased number of collisions decreases due to the mod behavior of our hash function and also because our load factor decreases as number of items are kept constant for each file.
* Also changing value of b and keeping table size constant doesn’t affect the number of collisions that much. But as discussed earlier, poor choice of b in combination with the table size may cause significant increase in collisions and run time may increase drastically.

**For task 4 and task 5**

Comparing collisions of separate chaining with quadratic and linear probing, it is seen that collisions for separate chaining are very less compared to the other two. This is as separate chaining has no primary clustering and secondary clustering. So one key when hashed will not occupy places of other keys as in separate chaining, so we do not have to probe. Once collision occurs, we just add it to the end of the link list. This prevents occupying other index positions of other keys which hence reduces collisions significantly.

Comparing collisions of quadratic and linear probing, it is seen that collisions are almost the same. Although quadratic probing eliminates primary clustering, it has the problem of secondary clustering. This leads elements with the same hash value forming chains and increasing run time if chains become longer in length. **“But the major problem in practice is that secondary clustering prevents us from guaranteeing an insert if table is >= half full.”**

At one time, it is seen that even if the table size is increased, all the collisions techniques tend to have the same number of collisions. This is because it doesn’t matter when table size is very high, because at that point whatever resolving collisions technique you use, the hash function will generally return array positions that would be very far apart, so different resolving collisions methods wont matter.

**Additional Research**

Although collisions for separate chaining are significantly less compared to linear and quadratic probing, it is seen that time taken for insertion is generally more for separate chaining although the collisions are significantly less. In practice, linear probing is typically significantly faster than chaining. This is primarily due to locality of reference, since the accesses performed in linear probing tend to be closer in memory than the accesses performed in chained hashing.