**Project2 – Worst-Case Complexity Analysis**

**CIS-350 SUMMER 2021**

**with Dr. Jinhua Guo**

**Demetrius Johnson**

**14 June 2021**

* **bool isPrime( int n )**
  + O(N)
  + The for loop that is dependent upon n (where n is number being checked as prime) will go less than n, say n/a, where a is some real number; thus loop will iterate as n approach to infinity up to n times in the worst case.
* **int nextPrime( int n )**
  + O(N)
  + This function contains a for loop that in the worst case will execute up to n/a times, where a is some real number; thus it simplifies to n as n approaches infinity.
* **int hash1( const string & key )**
  + O(N)
  + Polynomial hash function with highest degree of 1; n is the size of the string input into the hash function.
* **int hash1( int key )** 
  + O(1)
  + Simply returns the current integer; thus constant time.
* **bool contains(const HashedObj& x) const**
  + O(1)
  + Simply calls a function one time and returns; constant time.
* **void makeEmpty()**
  + O(N)
  + One for-loop; will execute up to n times, where n is the size of hash table vector.
* **bool insert(const HashedObj& x)**
  + O(1)
  + Calls a few functions which have higher degree; but this function itself is constant time.
* **bool remove(const HashedObj& x)**
  + O(1)
  + Calls a few functions which have higher degree; but this function itself is constant time.
* **//statistic functions**
* **float getLoadFactor(void)** 
  + O(N)
  + Simply returns a value; constant time.
* **int numActiveBuckets(void)**
  + O(N)
  + Simply returns a value; constant time.
* **int hashTableSize(void)**
  + O(N)
  + Simply returns a value; constant time.
* **int total\_InsertionCollisions(void)**
  + O(N)
  + Simply returns a value; constant time.
* **float avg\_InsertionCollisions(void)** 
  + O(N)
  + Simply returns a value; constant time.
* **int longest\_collision\_chain(void) { return longest\_collisionChain; }** 
  + O(N)
  + Simply returns a value; constant time.
* **bool isActive( int currentPos ) const**
  + O(N)
  + Simply returns a value after making one comparison; constant time.
* **int findPos( const HashedObj & x ) const**
  + O(N)
  + One while loop; it will iterate up to approximately n times in the worst, where n is the size of the hash table.
* **int findPos\_insert(const HashedObj& x)**
  + O(N)
  + Same as above function; just with a few variables to track statistics; other than that the functions are exactly the same.
* **void rehash( )**
  + O(N)
  + The first for loop executes up to 2\*n times, the next for loop which is not a nested for loop will execute up to 2\*n times as well, where n is the original size of the hash table passed in; thus 2\*n + 2\*n = 4\*n, which simplifies to a complexity of n, the highest power.
* **int myhash( const HashedObj & x )**
  + O(1)
  + Executes a few simple commands and calls a few functions.
* **void clear\_nonAlpha(string& word);**
  + O(N)
  + There is only one for loop which executes up to n times, where n is the size of the word passed into the function.
* **void spellChecker(const string& word, const int& lineNumber, const HashTable<string>& dictionary)**
  + O(N)
  + The highest amount of nested for loops is two: but the inner for loop only executes at a constant time and is not dependent on n (the size of a word) as the outer loop is.
* **int main(int argc, char \*\*argv )**
  + O(N^2)
  + Contains one for loop, and contains a while loop with another while loop nested inside.
  + Let n represent the number of lines to be checked in a file in the while loops; every iteration has to pass in a string, and then in the inner loop that string has to be parsed into smaller streams.

Here is a sidenote mainly for myself for future reference for how to run a program.exe with command line arguments:



