**CSC 480 Algorithms**

Assignment 3 Due October 24th 11:00 PM

## Aim:

Practice building and using Hashtable (chaining), QuickSort and HeapSort algorithm.
**Whitelisted libraries**: “iostream”, “fstream”, “chrono”, “cstdlib”, “ctype” and “string”.
**Greylisted** (restricted use library): list, **whereby the following members are blacklisted:**

**{**

- **Operations:**
- **splice**
- **Transfer elements from list to list (public member function )**

- **remove**
- **Remove elements with specific value (public member function )**

- **remove_if**
- **Remove elements fulfilling condition (public member function template )**

- **unique**
- **Remove duplicate values (public member function )**

- **merge**
- **Merge sorted lists (public member function )**

- **sort**
- **Sort elements in container (public member function )**

- **reverse**
- **Reverse the order of elements (public member function )**

- **Observers:**
- **get_allocator**
- **Get allocator (public member function )**

- **Non-member function overloads:**
- **relational operators (list)**
- **Relational operators for list (function ) [all]**

- **swap (list)**
- **Exchanges the contents of two lists (function template )**

**}**

## Part A:

One of the most useful ways of testing sorting algorithms is to use a large dataset. You will find a text file being "A Scandal In Bohemia.txt" (Sir Arthur Conan Doyle) develop a suitable hashtable with open hashing (chaining) or closed hashing (double hashing) to store its contents, once stored use heap sort to display a list of word occurrences from highest to lowest and vice versa of the 150 most and least repeated words (and their count) in the "Conan Doyle's" work. Please note that capitalization should not matter (i.e. "Watson" and "watson" should be counted as the same word).

Here are some things to consider:
(a) What occupancy ratio should you use?
Experiment with values such as 50%, 70%, 80%. And report runtime in clock-ticks and seconds

(b) How do you set up a hash table of variable size?

(c) What hash function?
Experiment, but a simple function such as `f(r)=r%hsize` should be your initial attempt.

(d) What if there are collisions in the table?
Describe the collision resolution you implemented.

(e) Do you need an interface file (a ".h" file) for these functions?

(f) Write a function to prompt a user for a word and display the number of occurrences of this word in the text.

(g) A function to output a list of the 150 **least frequently occurring words** in the text (using **HeapSort**).

(h) A function to output a list of the 150 **most frequently occurring words** in the text (using **HeapSort**).

(i) A function to output the number of sentences in the text.

(j) The heapSort you develop should be able to be re-used to sort and text input, as you design this for the above specs keep reusability in mind. Create a second main file in ass3j.cpp with appropriate entry in your makefile when built will render a program that expect a command line argument being an input text file and an output text file name. The program will read the data in the input file (strings or cstrings (you decide)) and write to the output file the sorted contents of the input. Input casing not assured output strings should all be lower case strings.

Test whether the function works by augmenting your driver program from previous steps in `ass3j.cpp` to form a menu driver to test all tasks. A "zero" menu option request terminates the program.

## Submit:

The files `makefile and all code files` should be submitted via D2L.