CSC 480 Algorithms
Assignment 3
Aim:

Practice building and using Hashtable (chaining), QuickSort and HeapSort algorithm.
Constraints: Whitelisted libraries: “iostream”, “fstream”, “chrono”, “cstdlib”, “ctype” and “string”.

Part A:
Step 1:
Write a driver program `ass3A.cpp` to sort a random array of integers on the range 1...10000 using the index array version of Quicksort. The program should be able to handle up to 10000 numbers with the only input being the number of elements of the array to fill. To avoid the scan overrun problem place 10001 in the next entry in the array after the data to be sorted (and an extra entry in the index array as well). The output should be the sorted order of the array printed 10 numbers to the line tabbed across the page, but the actual data should not be rearranged. Incorporate the seeding of the random number generator by time in the main function.

Step 2:
An improvement to Quicksort, so that closer to equal partition sizes are achieved is to use a median estimate for the pivot value. At the same time the scan overrun problem can be avoided by rearranging the Left, Middle and Right indexes so that the values they reference are arranged with the middle value in the Left position and the largest value in the Right position.

Write the function

```
    void Sort3(int& A, int& B, int& C);
```

to perform the task of rearranging the indexes so that `X[A]<=X[B]<=X[C]` in the least number of comparisons. Place the function in `qsort.cpp`, adding the appropriate call to `Sort3` in `Partition` just before the assignment of `PValue`. Remove the extra items in the original and index array. Check to make sure that the program still works.

Step 3:
An interesting application of the partitioning process of Quicksort is to find the kth element of the sorted order of an array without necessarily sorting the entire array. (k ranges from 0 to one less than the size of the array.) Such a function could find the median house price of 100,000 selling prices without actually sorting all the 100,000. It turns out that the procedure is no longer required to be recursive as each step of the process only further partitions one of the resulting partitions. Here’s how it works.

```
start with the entire array 0...n-1
    repeat
        partition the portion of the array, with resulting pivot point P
        if P < k,
            the kth is in the right portion so set it to be the portion to be further partitioned
        if P > k,
            then the kth must be in the left portion so make it the portion of the array
    until P = k or partition is of length 1 (we’ve found the kth).
```

Write the function

```
void kthValue(int index[], int n, int k);
```

to find the `k`th entry (numbered from 0) in the array of `n` values indexed by `index`. At completion of the function we can guarantee that the kth value is at index `k`. Place it in the file `qsort.cpp` (and its prototype in the associated interface file). Write a driver program in `ass3B.cpp` which reads in `n`, the size of an array (up to 1000 values), generates that many random integers in the range 1...10000 into such an array, then repeatedly reads in a value of `k`, calls `kthValue`, and prints out the result, terminating when a value not in the range `0...n-1` is input.

Step 4:
One of the most useful ways of testing sorting algorithms is to use a large dataset. You will find a text file being Pride and Prejudice (Jane Austin) 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 "J. Austin" work. Please note that capitalization should not matter (i.e. "Darcy" and "darcy" 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.

Test whether the function works by augmenting your driver program from previous steps in `ass3C.cpp` to form a menu driver to test all tasks. A "zero" menu option request terminates the program.

Submit:
The files `makefile(should make multiple targets and defaults to ass3c) asss3A.cpp, qsort.cpp, qsort.h, heapsort.h, heapsort.cpp, ass3B.cpp, hashtable.h, hashtable.cpp`, and `ass3C.cpp` should be submitted.