COSC 1437 – GE2

Part 1: This program is based on Chapter 8 Programming Challenge 8 (Search Benchmarks) on Page 488. Use the following file names for this part: C08PC18.CPP and C08PC18.EXE.

Write and test **five** separate functions, one for each of the following **five** algorithm steps. Make sure each function works correctly before trying Step **6**.

Algorithm Step **1**: Use the rand function to populate an int array passed as the first parameter with 1000 unique integers in the range 0 to 9999. The size of the array should be passed as the second parameter. Note that the array is an OUT parameter and the size is an IN parameter.

Algorithm Step **2**: Randomly generate an array index value from 0 to 999. Use this value as an index into the array passed in as the first parameter. Return the indexed array value using a return statement. The array is the first parameter (IN parameter) to the function. The size of the array is passed as the second parameter. The value returned from this function will be your search key for Step 3 and Step 5.

Algorithm Step **3**: Use a linear search to locate the key. Keep track of the number of comparisons needed to find the key. The array is the first parameter, the array size is the second parameter, and the search key is the third parameter. All parameters are IN parameters. Return the number of comparisons. On average the number of comparisons should be about 500.

Algorithm Step **4**: Sort the array in ascending order. The array is the first parameter (IN/OUT). The size of the array is the second parameter (IN).

Algorithm Step **5**: Use the binary search algorithm to find the key in the array. Keep track of the number of comparisons needed to find the key. The array is the first parameter, the array size is the second parameter, and the search key is the third parameter. All parameters are IN parameters. Return the number of comparisons. On average the number of comparisons should be about 9.

Step **6**: Write a main function. This function should seed the random number generator based on the system time. Then set up a loop that loops 1000 times. In the loop body, call each of the **five** functions, accumulating data on the number of comparisons for the linear search and the number of comparisons of the binary search.

When the loop ends, display a summary as follows:

After 1000 tests on 1000 element arrays, the linear search results were:
The minimum number of comparisons to find the key was: NNN
The maximum number of comparisons to find the key was NNN.
The average number of comparisons to find the key was: NNN.N

After 1000 tests on 1000 element arrays, the binary search results were:
The minimum number of comparisons to find the key was: NNN
The maximum number of comparisons to find the key was NNN.
The average number of comparisons to find the key was: NNN.N