CSCI 340 **Computer Assignment 2** **Spring 2018**
(10 points)

STL algorithms

Similar to Assignment 1, you are to implement two search algorithms (*linear search* and *binary search*) in C++ on randomly generated integers stored in vectors. In this assignment, you will use routines from STL <algorithm> to implement these algorithms.

Assignment2.cc is provided to you at the directory:
`/home/turing/mhou/public/csci340spring2018.`

In this file, the main function is already implemented. You are required to implement the following functions:

- `void genRndNums ( vector < int >& v, int seed )` : This routine generates random integers in the range `[ LOW = 1, HIGH = 100 ]` and puts them in vector `v`. Initializes the random number generator (RNG) by calling the function `srand()` with the seed value `seed`, and generates random integers by calling the function `rand()`. To use `srand` and `rand`, you need the header `<cstdlib>`. The vector `v` is already allocated with space. Use vector's member function to get the size of the vector.

- `bool linearSearch ( const vector < int >& inputVec, int x )` : A linear search algorithm, where `x` is the searched item in vector `inputVec.` It simply starts searching for `x` from the beginning of vector `v` to the end, but it stops searching when there is a match. If the search is successful, it returns `true`; otherwise, it returns `false`. To implement this routine, simply call the `find()` function from the STL `<algorithm>`.

- `bool binarySearch ( const vector < int >& inputVec, int x )` : A binary search algorithm, where `x` is the searched item in vector `inputVec`. If the search is successful, it returns `true`; otherwise, it returns `false`. To implement this routine, simply call the `binary_search()` function from the STL `<algorithm>`.

- `int search ( const vector < int >& inputVec, const vector < int >& searchVec, bool ( *p ) ( const vector < int >&, int ) )` : A generic search algorithm – takes a pointer to the search routine `p( )`, and then it calls `p( )` for each element of vector `searchVec` in vector `inputVec`. It computes the total number of successful searches and returns that value to the `main()` routine as an input argument to the print routine `printStat()`, which is used to print out the final statistics for a search algorithm.

- `void sortVector ( vector < int >& inputVec )` : A sort algorithm to sort the elements of vector `inputVec` in ascending order. To

implement this routine, simply call the `sort()` function from the STL `<algorithm>`.

- `void printStat ( int totalSucCnt, int vec_size )` : Prints the percent of successful searches as floating-point numbers on `stdout`, where `totalSucCnt` is the total number of successful comparisons and `vec_size` is the size of the test vector.

- `void print_vec ( const vector < int >& vec )`: This routine displays the contents of vector `vec` on standard output, printing exactly `NO_ITEMS = 10` numbers on a single line, except perhaps the last line. The sorted numbers need to be properly aligned on the output. For each printed number, allocate `ITEM_W = 6` spaces on standard output. You can re-use the implementation of this routine from Assignment 1, but remember to change the values of related constants.

Programming Notes:

- Include any necessary headers and add necessary global constants.

- You are not allowed to use any I/O functions from the C library, such as `scanf` or `printf`. Instead, use the I/O functions from the C++ library, such as `cin` or `cout`.

- You need to use correct implementation of `print_vec` from the first assignment. Please seek help from the TAs if necessary.

- Execute the `srand ( )` function *only once* before generating the first random integer with the given seed value `SEED`. The `rand ( )` function generates a random integer in the range [`0, RAND_MAX`], where the constant value `RAND_MAX` is the largest random integer returned by the `rand ( )` function and its value is system dependent. To normalize the return value to a value in the range [`LOW, HIGH`], execute:

  `rand ( ) % ( HIGH – LOW + 1 ) + LOW`.

- In the final version of your assignment, you are not supposed to change existing code, including the main method, provided to you in the original source file assginment2.cc.

- To compile the source file, execute "`g++ -Wall assignment2.cc –o assignment2.exe`". This will create the executable file `assignment2.exe`. To test your program, execute "`./assignment2.exe &> assignment2.out`", which will put the output (including any error messages) in file `assignment2.out`. You can find the correct output of this program in file `assignment2.out` in the directory shown in the last page.

- Add documentation to your source file.

- Prepare your Makefile so that the TA only needs to invoke the command "`make`" to compile your source file and produce the executable file `assignment2.exe`.

- When your program is ready, submit your source file and Makefile to your TA by following the Assignment Submission Instructions.