HW File: (ALL CODE MUST BE IN C++)

All of these programs should have an accompanying “white paper” explaining what you did. They should be commented fully as well. Please record a video presentation explaining what you did in a concise and clear manner.

1. Test whether a vector or a list is faster by timing them via:
    a. Filling them with random numbers
    b. Filling them with random strings (use ascii)
    c. Doing the above with move semantics when filling with random strings
2. Test whether a recursive, iterative or linked-type binary search is faster by testing it on arrays of size 1 million, 10 million, and 100 million with:
    a. Arrays that are filled with random numbers
    b. Arrays that are filled with numbers that are backwards
3. Create a sort for 2d matrices. Do a bubble sort, insertion sort, and selection sort variation of this.
4. Compare the times it takes to sort a random array vs a linked list with a bubble sort.
5. Create a multi-level sort. For instance, for all selections of n > 10 you do sort X and within sort X, when you have a situation with n < 10 you do sort Y. Be creative. Time your sort against two “reasonably comparable” sorts (you may use libraries for the “reasonably comparable sorts”).
6. We have received a secret message encoded with a Vigenere cipher (wiki). We know that the key is 32 digits long and a brute force attack would take us a long time (see here). We do not know any “tricks” how to break a Vigenere cipher, but we do know how it works. Instead we are going to test random possible solutions and use a feature function to evaluate the possible solution. We will collect solutions that pass a certain threshold and mutate those solutions with some low probability, hopefully getting closer to the ‘true solution”. We will also use recombination (see here) , with some low probability, continually increasing our threshold as we narrow in on a solution. Create a program that does this, paying special attention to the data structure you wish to manage your possible solutions.
7. Create your own vector and list. Time it similar to question #1
8. Create a *templated class* that *effectively* finds all possibilities of a list of random numbers that adds to some s.
9. Create a random array of size k. Create a function to check if it is a deBruijn (https://en.wikipedia.org/wiki/De_Bruijn_sequence) sequence of B(n,k). If it is not randomly mutate each spot in the array (from a 0 to 1 or 1 to 0) with a 5% probability, keep doing this until you have found a deBruijn sequence. Do this with an array structure and a linked structure. To this 100 times for each, time it and compare your results.
10. Create a linked list filled with numbers 1 to k. Create a function that sorts the list so that first all odd numbers are listed and then all even numbers.