CMPT 130: Lab Work Week 12

1. Given two arrays **A** and **B** with equal size **n**, write a C++ function that returns true if every element of A is found in B; otherwise returns false.

   In your function there will be an **if-statement** that compares an element in A with an element in B, how many times does, in the worst case, that **if-statement** get executed? Give your answer in terms of **n**. Give the complexity of your algorithm in Big-O notation. Classify your algorithm as linear, logarithmic, quadratic, or some other class of functions. Assume the arrays **A** and **B** are not sorted.

   n^2, O(n^2), quadratic

2. Repeat question 1 above but this time assume the array **A** is not sorted but the array **B** is sorted.

   binary, nlogn, O(nlogn)

3. Repeat question 1 above but this time assume the array **A** is sorted but the array **B** is not sorted.

   binary, nlogn, O(nlogn)

4. Repeat question 1 above but this time assume both the arrays **A** and **B** are sorted.
5. While it is important to analyze complexity of algorithms for the worst case scenarios, it is advisable for us also to know how to count the actual number of times a critical operation is performed for specific input.

   So consider the following array and answer the questions that follow:

A =							
	9	**2**	**3**	**5**	**2**	**7**	**6**

   a. How many times does the critical operation of the selection sort algorithm get executed for this specific array?

   b. How many times does the critical operation of the Bubble sort algorithm get executed for this specific array?

   c. How many times does the swapping operation of the elements of A[j] and A[j+1] in the bubble sort algorithm get executed for this specific array?

   d. How many times does the critical operation of the **modified** Bubble sort algorithm get executed for this specific array?

   all decreasing, no diff between modified and normal bubble sort

   e. How many times does the critical operation of the insertion sort algorithm get executed for this specific array?

   f. How many times does the moving of the element of A[j] to A[j+1] in the insertion sort get executed for this specific array?

   g. How many times does the critical operation of the count distinct elements of an array algorithm get executed for this specific array?

6. Guided by Q5 above, provide an example of an array where the critical operation of the insertion sort algorithm will be performed as many times as the worst case scenario.
7. Repeat Q6 above for selection sort algorithm.
8. Repeat Q6 above for bubble sort algorithm.
9. Repeat Q6 above for modified bubble sort algorithm.
10. Repeat Q6 above for counting distinct elements of any array algorithm.

11. Write a C++ function named `cstrSequentialSearch` that takes two arguments: a cstring **s** and a character **ch** and returns the index of **ch** in **s** if it is found or return -1 if it is not found. **Hint:-** All you need to do is to first determine the length of the cstring and then use sequential search algorithm.
12. Write a C++ function named **reverseSequentialSearch** that takes an array of integers, its size and a searchValue as arguments and returns the last index of the searchValue in the array. If the search value is not found, your function must return -1.
13. Write a C++ function named **reverseCstrSequentialSearch** that takes a cstring and a searchValue as arguments and returns the last index of the searchValue in the cstring. If the searchValue is not found, then your function must return -1.
14. Write a C++ function named **cstrBinarySearch** that takes four arguments: a sorted C-string **s**, a start index, last index, and a character **ch** and returns the index of **ch** in **s** if it is found or return -1 if it is not found. Use binary search algorithm.
15. **[Challenge:- This is beyond the scope of CMPT 130 but try it]** Given an array **A** of size **n**, we would like to fill the array with ***distinct*** random integer numbers in the range **[1, n]**. That is each element of the array must be assigned a random integer number in the specified range and at the end the elements of the array must be distinct.

    First of all, let us design an algorithm to fill the array.

    Step 1. Generate a random integer in the range [1, n] and put it in A[0]

    Step 2. In order to fill element A[k], for each index k=1,2,3,…,n-1; repeatedly generate random integer in the range [1, n] until you get one that is different from all the elements A[0], A[1], A[2], …., A[k-1]. Once you get such a random integer, put it in A[k] and proceed to fill A[k+1].

    Question:- Implement the algorithm as a function and write a test program in order to test your function. How many times does your algorithm generate random numbers? Give the worst case complexity of the algorithm i.e. the count of generating random numbers in Big-O notation.