Assignment 6 - Algorithms Primer

Problem 1.

a) Write `public static int search(String[] arr, String s)` which returns the first index of `arr` which is s. If s is not found in the array, the method should return -1. For example, `search({"a", "b", "c", "b"}, "b")` should return 1, while `search({"a", "b", "c", "b"}, "z")` should return -1. Once your method finds s, it should return and not check any more indices.

b) What is the best case scenario when you run search? In other words, what is the minimum number of elements your method has to check and when does that occur? What is the worst case? How many elements of arr does your method have to check in this case?

c) What if we were guaranteed that `arr` was *sorted*, i.e. the strings are in alphabetical order. Can you think of any better ways to look through the array?

Problem 2.

a) Write a `public static int minIndex(int[] arr)` method that returns the index of the smallest element in the input array `arr`. For example, `minIndex({3, 4, 1, 5, 2, 9})` should return `2` since the minimum element in the array is at index 2 in the array. If there are multiple minimum elements, you can return any index where that element is stored.

Hint: As you loop through the array, you will need to store temporary variable for the current minimum, and its index. Initialize these to the 0-th element of the array before you start iterating.

b) Modify your code above to write `public static int minIndexStartingAt(int[] arr, int startIndex)` that finds the minimum element of `arr` stored between index `startIndex` and the end of the array. For example, `minIndexStartingAt({3,4,1,5,2,9}, 3)` returns `4`, since 2 is the minimum element starting at index `3`. `startIndex` should be inclusive, i.e. the element stored at `startIndex` could be the minimum.

c) On the last homework assignment, we wrote `public static void swap(int[] arr, int a, int b)` which swaps the elements at index a and index b in the array. We can use that method in conjunction with `minIndexStartingAt` to sort an array so that it's elements are stored in the array in ascending order.

To do so, we can first find the index of the minimum element of the array, and then swap it into the first position. Then we find the minimum of the rest of the array, and swap it into the second position. We repeat this until we've sorted the entire array. This algorithm is called *selection sort*.

Write `public static void selectionSort(int[] arr)` that sorts `arr` using the algorithm described above.

Problem 3. Imagine you had a complicated Boolean expression that contained many different Boolean variables. If you wanted to know if that Boolean expression could ever evaluate to `true`, you might try to use *brute-force search*, where you try every possible *assignment* of the variables (an *assignment* is a specific setting of the variables to `true` and `false`).

For example, you might start by making every variable `false` and then evaluate the expression. If it was `false`, you might then try making the first variable `true` and the rest `false`, and re-evaluate. Once we've tried making every variable `true` while the rest are `false`, we would try making 2 variables `true` at once while the rest are false. We repeat until we try making every variable `true` at once.

If there were only 2 Boolean variables, how many assignments would you have to check? What about 3 variables? What if there are n variables?