Homework 3

1.

We search for the substring "ABX" in the string "ABCDABCDABCDABXD" Successful matches are in bold and unsuccessful are in regular text.

A	B	C	D	A	B	C	D	A	B	C	D	A	B	X	D
A	**B**	X													
	A														
		A													
			A												
				A	**B**	X									
					A										
						A									
							A								
								A	**B**	X					
									A						
										A					
											A				
												A	**B**	**X**	

We see that there are 9 successful and 12 unsuccessful matches.

2. Section 3.2 #8 (a)

ALGORITHM *CountSubstrings*$(T[0 \dots n])$

```
count ← 0
for i ← 0 to n − 1 do
    if T[i] = 'A' then
        for j ← i to n do
            if T[j] = 'B' then count ← count + 1
            end if
        end for
    end if
end for
```

In the best case, we have a string with no A's in it, and we never enter the second for loop. Thus, we would only iterate through the string once, so

$$C_B(n) \in \Theta(n)$$

In the worst case, we would have a string of just A's, so we would enter the second loop every single time. Thus, we have

$$\begin{aligned} C_W(n) &= \sum_{i=0}^{n-1} \sum_{j=i}^{n} 1 \\ &= \sum_{i=0}^{n-1} (n - i + 1) \\ &= (n+1) + (n+0) + \cdots + 1 \\ &= \frac{(n+2)(n+1)}{2} \end{aligned}$$

So,

$$C_W(n) \in \Theta(n^2)$$

3. Section 3.4 #8

If we have an array of n elements, then we can generate a permutation of the array and then check if that permutation is ordered. WE will always have to make $n-1$ comparisons, and at worst, we'll have to check $n!$ permutations. Thus, we'll have to make at most $(n-1)n!$ comparisons. So the efficiency of the worst case is

$$C_W(n) \in O((n+1)!)$$

4. Section 4.1 #7