## 6.19:

**Answer:**

We will use dynamic programming.

Let the binary string $s$ have $n$ characters. Let $x'$ be the repetition of $x$ with $n$ characters. Let $y'$ be the repetition of $y$ with $n$ characters. We rephrase the problem as: is $s$ an interleaving of $x'$ and $y'$? This way we don't have to deal with more than one pass through $x'$ or $y'$ because they are both already as long as $s$. Let $s[j]$ be the $jth$ character in $s$ and $s[1:j]$ be the slice of the first $j$ characters in $s$. WLOG we define the same notation for $x'$ and $y'$.

It's given that if $s$ is an interleaving of $x'$ and $y'$, then the last character in $s$ comes from either $x'$ or $y'$. Therefore, by removing the last character, we can get a smaller recursive problem on $s[1:n-1]$ with the repetitions $x'$ and $y'$ (taking the prefix of whichever repetition contained the last character).

Subproblems:
Let $T(i,j)$ = yes if $s[1:i+j]$ is an interleaving of $x'[1:i]$ and $y'[1:j]$.

Recurrence:
Let $T(i,j)$ = yes iff ($T(i-1,j)$ = yes AND $s[i+j] = x'[i]$) OR ($T(i,j-1)$ = yes AND $s[i+j] = y'[j]$).

Algorithm:

```
1  Is-Interleaving(s, x', y', n)
2    Array M[0...n][0...n]
3    M[0][0] = yes
4    for i = 1, ..., n
5      for j = 1, ..., n
6        if i + j ⩽ n
7          if M[i − 1][j] = yes AND s[i + j] = x'[i]
8            M[i][j] = yes
9          if M[i][j − 1] = yes AND s[i + j] = y'[j]
10           M[i][j] = yes
11         if i + j = n AND M[i][j] = yes
12           return yes
13         M[i][j] = no
14   return no
```

Time complexity:
The outer for loop runs $n$ times and the inner for loop runs at most $n$ times, and the statements within the inner for loop run in constant time. Therefore, the total running time is $O(n * n * 1) = O(n^2)$.

5.2 We can modify the alg. for finding the # of inversions. The key is that we will "merge" twice, once for sorting and once for counting the significant inversions. This is because while the comparison for sorting and counting was the same in the original problem, it is different for this problem so the pointers advance differently.

Merge-and-Sort ( A, B )

Maintain a Current pointer into each list, initialized to point to the front elements

While both lists are nonempty

Let $a_i$ and $b_j$ be the elements pointed to by the Current pointer

Append the smaller of these two to the output list

Advance the Current pointer in the list from which the smaller element was selected

Once one list is empty, append the remainder of the other list to the output

Return the merged list

Merge-and-Count(A, C)

$n = |A| + |C|$

$k = \lfloor n/2 \rfloor$

$i = k$, $j = n$, $N_3 = 0$

if $A[i] > C[j]$

$N_3 \mathrel{+}= j - k$

if $i > 1$

$i--$

if $i = 1$

return $N_3$

else

if $j > k + 1$

$j--$

if $j = k + 1$

return $N_3$

Let $L = \{a_1, \ldots, a_n\}$

Sort-and-Count-Significant-Inversions(L)

$n = |L|$

if $n = 1$

return $N = 0$, $\{a_1\}$

$k = \lfloor n/2 \rfloor$, $A = \{a_1, \ldots, a_k\}$, $B = \{a_{k+1}, \ldots, a_n\}$

$N_1, A = b_1, \ldots, b_k$ = Sort-and-Count-Significant-Inversions(A)

$N_2, B = b_{k+1}, \ldots, b_n$ = Sort-and-Count-Significant-Inversions(B)

$C = \{2b_{k+1}, \ldots, 2b_n\}$

O(n) $N_3$ = Merge-and-Count(A, C)

O(n) return $N = N_1 + N_2 + N_3$, $b_1, \ldots, b_n$ = Merge-and-Sort(A, B)

Let $T(n)$ be the maximum # of steps Sort-And-Count-Significant-Inversions does on n numbers. It has 2 recursive calls and additionally has at most n steps for Merge-and-Count and at most n steps for Merge-and-Sort. So $T(n) \leq 2T(n/2) + 2n$. By theorem 5.2, $T(n)$ is bounded by $O(n \log n)$.

5.3 We can use a divide and conquer algorithm. The key is that if more than $n/2$ cards are equivalent in the whole set, then at least one of the recursive calls on the 1st and 2nd halves will have the same property. However, the converse is not true, so if either half has the property, then we must check if the majority card from that half is also a majority card in the whole set.

```
Has-Majority-Card(S)
    if |S| = 1
        return the one card
    if |S| = 2
        if the two cards are equivalent
            return either card
    n = |S|, c = NULL
    Let S1 be the set of the first ⌊n/2⌋ cards
    Let S2 be the set of the remaining cards
    if a card c1 is returned by Has-Majority-Card(S1)
        test c1 against all other cards in S1 ∪ S2
        if c1 is equivalent to more than n/2 cards in S1 ∪ S2
            c = c1
    if a card c2 is returned by Has-Majority-Card(S2)
        test c2 against all other cards in S1 ∪ S2
        if c2 is equivalent to more than n/2 cards in S1 ∪ S2
            c = c2
    return c
```

Let $T(n)$ be the maximum number of tests the algorithm does on a set of $n$ cards. It has 2 recursive calls and does at most $n + n = 2n$ tests outside of that. So $T(n) \leq 2T(n/2) + 2n$. By theorem 5.2, $T(n)$ is bounded by $O(n \log n)$.

5.5 We can use a divide and conquer algorithm. The keys are:

- Start by sorting the lines in order of increasing slope. Note that the 1st and last lines will always be visible.
- If two lines are visible, the region in which the line of smaller slope is uppermost lies to the left of the region in which the line of larger slope is uppermost.
- When we merge, we consider each intersection point and consider which half has the uppermost line at this point.
- We must determine the visible lines and corresponding intersection points when merging two subproblems in $O(n)$ time, in order to achieve $O(n \log n)$ total running time.

Let $A = \{L_1, \ldots, L_n\}$

```
Find-Vis-Lines (A)
    Sort A in order of increasing slope
    n = |A|
    R = {}, P = {}.
O(1)  if n ≤ 3
        L1 and L3 will always be visible, L2 will be visible iff it meets
        L1 to the left of where L3 meets L1
        add visible lines to R in order of increasing slope
        add intersection points to P in increasing order of x-coordinate
    m = ⌈n/2⌉
    Call Find-Vis-Lines (L1, ..., Lm), R1 = {Li1, ..., Lip}, P1 = {a1, ..., ap-1}
    Call Find-Vis-Lines (Lm+1, ..., Ln), R2 = {Lj1, ..., Ljq}, P2 = {b1, ..., bq-1}
O(n)  Call R, P = Merge (R1, P1, R2, P2)
    return R, P
```

Merge ( $R_1$, $P_1$, $R_2$, $P_2$ )

O(n) merge $P_1$ and $P_2$ in order of increasing x-coordinate into $c_1, \ldots, c_{p+q-2}$

O(n) for each k

Let $L_{i_s} \in R_1$ be the uppermost line in $R_1$ at $c_k$

Let $L_{j_t} \in R_2$ be the uppermost line in $R_2$ at $c_k$

Let $\ell$ be the smallest index for which $L_{j_t}$ lies above $L_{i_s}$ at $c_\ell$

Let $(x^*, y^*)$ be intersection point of $L_{i_s}, L_{j_t}$ so $c_{\ell-1} < x^* < c_\ell$

So $L_{i_s}$ uppermost left of $x^*$, $L_{j_t}$ uppermost right of $x^*$

return visible lines among $R_1 \cup R_2 = \{L_{i_1}, \ldots, L_{i_s}, L_{j_t}, \ldots L_{j_q}\}$,

intersection points $= \{a_{i_1}, \ldots, a_{i_{s-1}}, (x^*, y^*), b_{j_t}, \ldots b_{j_{q-1}}\}$

Let $T(n)$ be the maximum number of steps Find-Vis-Lines does on $n$ lines. It has 2 recursive calls and Merge does at most $n + n = 2n$ operations outside of that. So $T(n) \leq 2T(n/2) + 2n$. By theorem 5.2, $T(n)$ is bounded by $O(n \log n)$.