Week 1 D

> [!WARNING] Direct quote from solution notes contain words of venting

```
// ======== NOTES ========
/*  Subproblem:
 *
 *  We can construct the total number of prefixes included at each i
 *  the number of included s[i] and t[i] by checking for s[i] == 'b' and t[i] == 'a'
 *      ans += (ans_i = (s[i] == 'b') + (t[i] == 'a'))
 *  Each increment of i ans_i doubles
 *  ans_i is bounded by k
 *
 */
```

This is a counting problem, notice that the number of potential strings is exponent of 2 to the length. We could traverse from the front to the back of the two strings (by major lexicographical order), and then find the difference between two current pointer (i.e. if both s_i and t_i are `a`, then the number of generateable string from this position is determine by $i + 1$, and the number of potential strings so far between `s` and `t` from this character is 1, if $s_i < t_i$, that is `a` and `b`, then the number of potential strings upto this point is 2), and 0 if `s[i] = b`, `t[i] = a`. The recurrence relation above illustrates this logic.

However if we hit the upper bound of k strings for the total number of strings, we will have to fixate it to k. Note that the variable `ans_i` for accumulating each instances of i may still grow and will cause an overflow. Even if it is no longer used after exceeding `k` it can still cause exceptions if left unchecked.

The total runtime of this algorithm is $O(n)$ dependent purely on the length of the input `s` and `t`.