1) (10 pts) ANL (Algorithm Analysis)

Consider the following problem:

Given two input values, ***n*** and ***k***, determine the number of strings of length ***n***, which only contains A's and B's, that have a run of ***k*** or more consecutive B's.

One algorithm to solve the problem is as follows:

Recursively generate each possible string of ***n*** A's and B's. These can be generated in alphabetical order, never storing more than 1 of the strings at the same time.

For each string generated, loop through the string from left to right, keeping a running tally of the current number of B's. (For example, with the string ABBABBBAAB, the running counter would update as follows 0 → 1 → 2 → 0 → 1 → 2 → 3 → 0 → 0 → 1.) If this running tally ever equals or exceeds ***k***, add 1 to a global counter storing the final result. For simplicities sake, assume that the loop completes going through the whole string before 1 is potentially added to the global counter.

With proof, determine the Big-Oh runtime of this algorithm in terms of the input parameter, ***n***.

The recursive algorithm which generates each possible combination of ***n*** letters runs itself in $O(2^n)$ time. For each letter, there are 2 choices, and we pair up each possible choice at each slot, so to get the total number of combinations we multiply 2 by itself ***n*** times to get 2^n combinations. Though it's a little difficult to prove (no proof is required for full credit here), although sometimes we change more than 1 letter between combinations, over the course of the whole algorithm, the total number of letter changes does not exceed 2^{n+1}, which is a constant times 2^n, meaning that the original run time of $O(2^n)$ to generate each combination is accurate.

For each combination generated, the algorithm described runs a simple loop through all the letters of the string. Since there are ***n*** letters and only a constant amount of work is done for each letter (either adding one to our running tally or setting it to 0), $O(n)$ time is spent on each combination.

It follows that the overall running time of the algorithm is $O(n2^n)$.

Grading: 5 pts for arguing that the total number of combinations is 2^n.
3 pts for arguing that evaluating each combination takes $O(n)$ time.
2 pts for concluding that the total run time is $O(n2^n)$

Give partial as necessary – some credit can be given if parts of the analysis are accurate. (For example, award 2 pts out of 5 for the incorrect conclusion that there are n! valid strings of length n.)