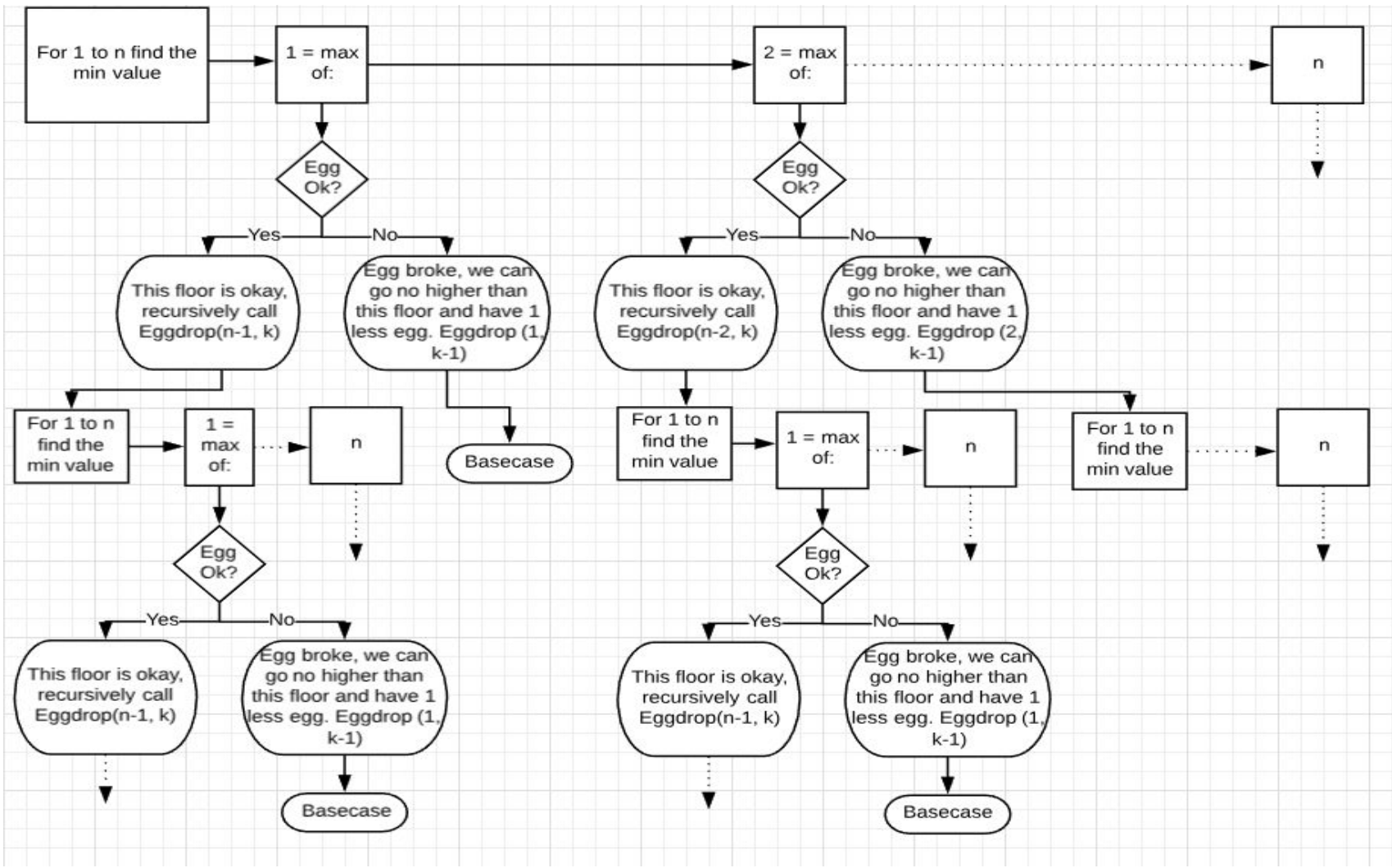
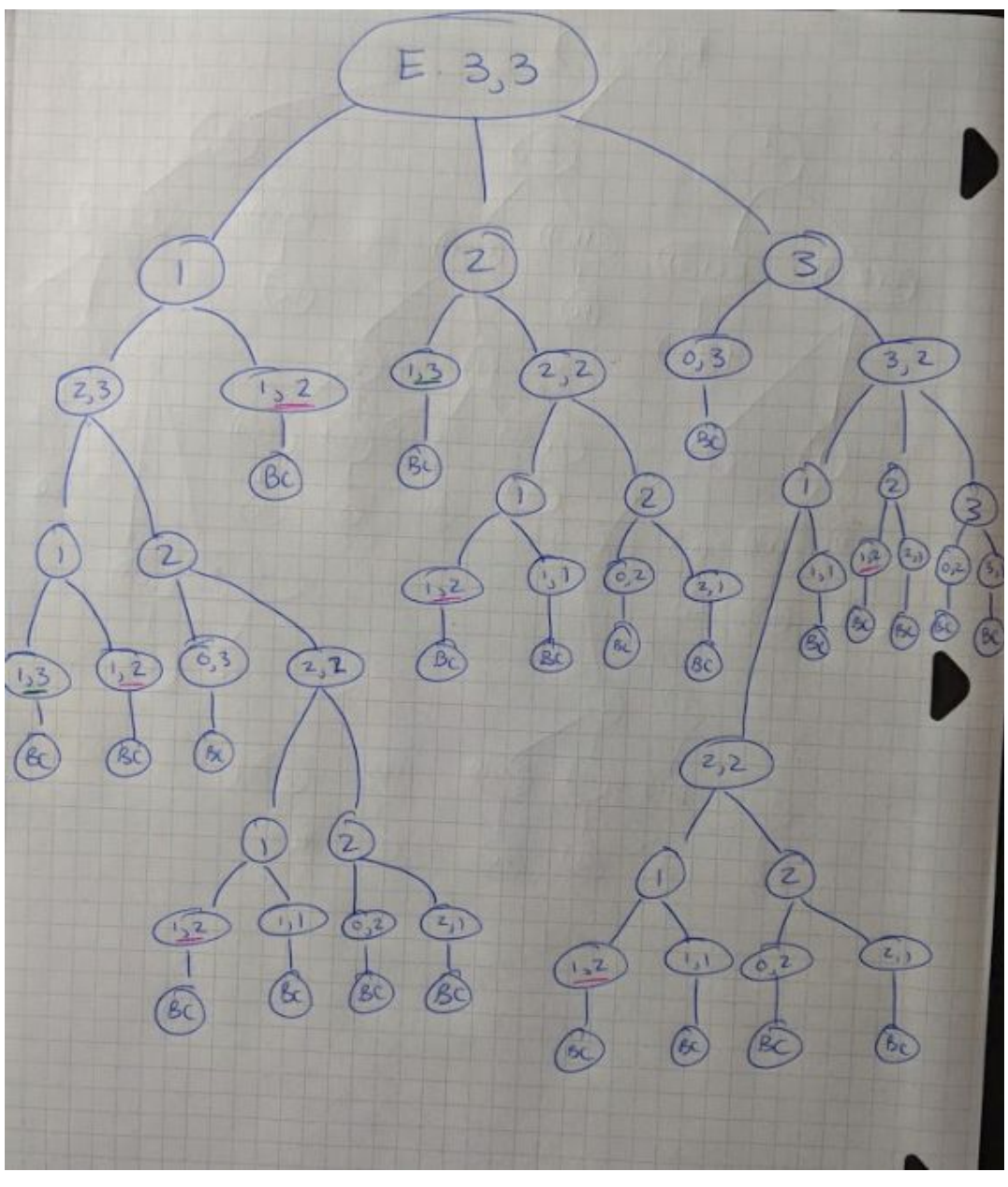


I made the diagram above to help me understand the flow of the E(n,k) algorithm, and how it would find the min number of trials required for a n-story tall building. For 1 to n it is checking to find what the maximum value is given both possibilities: the egg breaks or the egg doesn't break. For each possibility it then calls E(n,k) recursively after adjusting that n and k values and it repeats the for loop to find the min value for that given possibility. It does this until a basecase is reached and the maximum number of trials for that iteration is established. The original function call then returns the minimum value selected from these maximum values.

1. Using the recurrence, give a small example demonstrating why the Egg Drop problem is a good candidate for dynamic programming.



Dynamic programming is useful when a problem has: a recursive structure, more than one recursive call, and the calls overlap. As shown to the left all three of these criteria apply to the E(n,k) algorithm considering that each iteration from 1 to n generates two recursive calls, which themselves iterate through 1 to n generating recursive calls, and so on until a basecase is reached. Therefore any non-trivial call to E(n,k) will cause the problems to overlap several times over (some of which I've underlined in green and pink)

The problem also exhibits optimal substructure as it makes one decision by solving smaller versions of the problem until it finds the basecases and then combines the solutions.

2. What implementation would you recommend for a dynamic programming algorithm for the Egg Drop problem? Briefly justify your answer.

	1	2	3	...	k
0	n	n	n	n	n
1	n	n	n	n	n
2	n	NIL	NIL	NIL	NIL
...	n	NIL	NIL	NIL	NIL
n	n	NIL	NIL	NIL	NIL

A memoized solution could implement a two dimensional array to track the return values for each n, k pairing.

When the problem calls for a recursive call it would first check to see if a value already exists for that n, k pairing. If so, it would use that value in constant time rather than having to recursively find the value.

When the value does not exist the program would insert the value into the array to reduce runtime for the following calls.

3. Give pseudocode for a naïve recursive algorithm (i.e., non-dynamic programming algorithm) for $E(n, k)$.

Input:	n: an n-story building
Input:	k: a number of eggs
Output:	worst case number of trials to find floor egg will break from
1	Algorithm: EggDrop
2	**If** k = 1 or n = 0 or n = 1 **then**
3	**return** n
4	min = n
5	**for** i = 1 to n **do**
6	check = **max**(EggDrop[n-i, k], EggDrop[i, k-1])
7	**if** check < min **then**
8	min = check
9	**return** min

4. Give pseudocode for a memoized dynamic programming algorithm for $E(n, k)$.

Input:	n, k: building stories and number of eggs
Output:	two dimensional array
1	Algorithm: EggDropArr
2	dropresults = Array(n, k)
3	Initialize all cells of dropresults to NIL
4	Initialize all cells of dropresults (*, 1) and (0, *) and (1, *) to n
5	return dropresults(n, k)

Input:	dropresults(n, k), n, k
Output:	worst case number of trials to find floor egg will break from
1	Algorithm: EggDrop
2	min = n
3	**for** i = 1 to n **do**
4	**if** dropresults(n-i, k) != NIL **then**
5	chk1 = dropresults(n-i, k)
6	**else**
7	chk1 = dropresults(n-i, k) = EggDrop[n-i, k]
8	**if** dropresults (i, k-1) != NIL **then**
9	chk2 = dropresults(i, k-1)
10	**else**
11	chk2 = dropresults(i, k-1) = EggDrop[i, k-1]
12	check = **max**(chk1, chk2)
13	**if** check < min **then**
14	min = check
15	**return** min