Task

- For each k, how many methods called
- What is the pattern
- Explain

For RHanoi

If k = 1

Public List<String> solve (****) calls private void solve(****)

n!=0 first time so it runs solve again

n==0 so it returns

Then list.add(getKey(n, source, destination)); record the step just taken

Then it runs solve again to take the one ring from the intermediate to the destination.

Private Solve was called 3 times

If k = 2

Public List<String> solve (****) calls private void solve(****)

2!=0 so run solve again with n-1 (goal: source -> intermediate)

1!=0 –we've been here before +2 iterations to solve

Then list.add(getKey(n, source, destination)); record the step just taken

run solve again with n-1 (goal: intermediate -> destination)

1!=0 –we've been here before +2 iterations to solve

Private Solve was called 7 times

If k = 3

Public List<String> solve (****) calls private void solve(****)

3!=0 so run solve again with n-1 (goal: source -> intermediate)

2!=0 –we've been here before +6 iterations to solve

Then list.add(getKey(n, source, destination)); record the step just taken

run solve again with n-1 (goal: intermediate -> destination)

2!=0 –we've been here before +6 iterations to solve

Private Solve was called 15 times

Conclusion : for recursive Hanoi the formula is $k_0=1$, $k_n=2^{n+1}-1$

This makes sense after walking through my explanation above. It involves using repetitive solutions from previous steps to solve the current issue.

For DHanoi

If k = 1

Public List<String> solve (****) calls private void solve(****)

1!=0 first time so it runs solve again after it fails (sub != null) as there have been no previous steps recorded (goal: source -> intermediate)

n==0 so it returns

Then list.add(getKey(n, source, destination)); record the step just taken

Check if previous steps exist for n-1 again (fails) (goal: intermediate -> destination)

So run solve for n-1

n==0 so it returns

if steps don't exist- add them to the map

Private Solve was called 3 times

If k = 2

Public List<String> solve (****) calls private void solve(****)

2!=0 but doesn't fail (sub != null) this time as there are previous steps recorded (goal: source -> intermediate)

So it copies the list of steps leading up to this point from the map in a sublist then records them into the original list

Then list.add(getKey(n, source, destination)); record the step just taken

Check if previous steps exist for n-1 (goal: intermediate -> destination) (they do exist)

So it copies the list of steps leading up to this point from the map in a sublist then records them into the original list

if steps (key) don't exist- add them to the map

Private Solve will be called 1 time if k =1 has been run before. Else solve will be run 7 times as it needs to save steps into the map

Conclusion: for dynamic Hanoi the number of times private void solve(****) is called depends on if the map and list are populated with the previous k=1 to k = k-1 solutions. If so then solve will only be called once, except for when k =1. Else the program will require just as many recursive calls as Rhanoi to populate the map and list.
so $k_0=1$, $k_n=2^{n+1}-1$ or $k_0=1$, $k_1=2$, $k_n=1$