Roman Vasilyev

Prof Ross

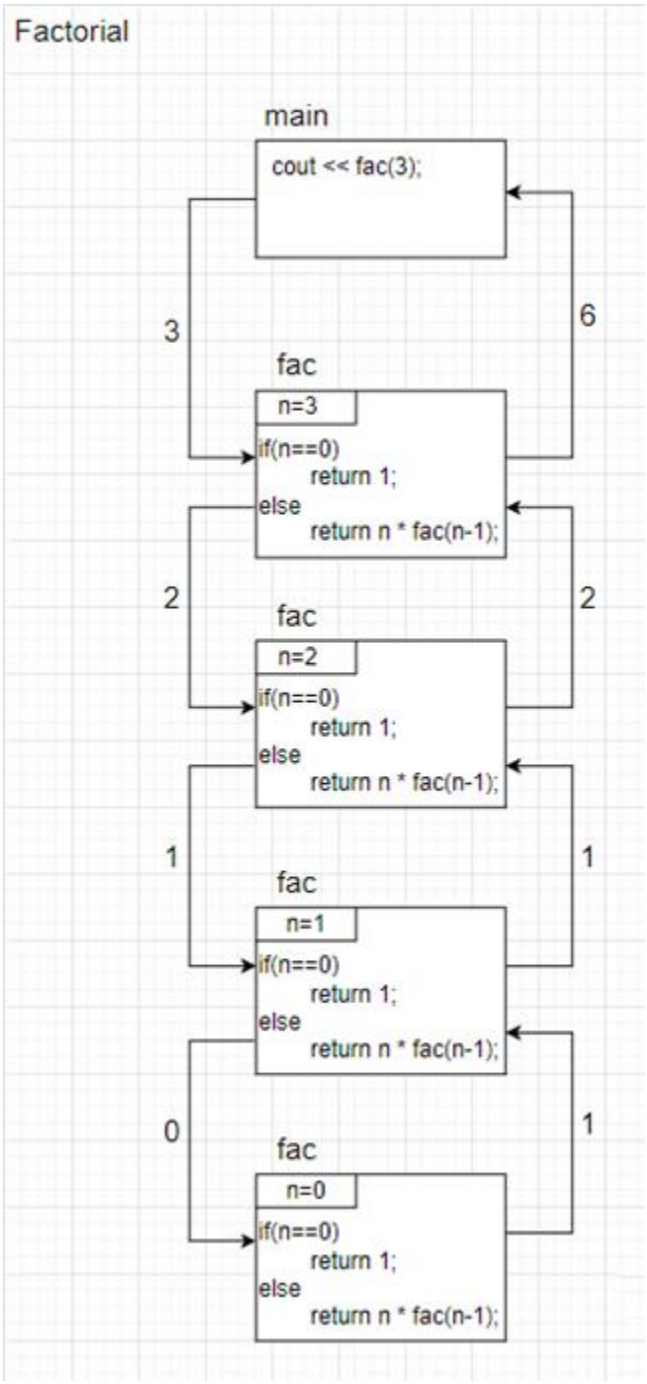
2/1/2024

Week 2

Recursion

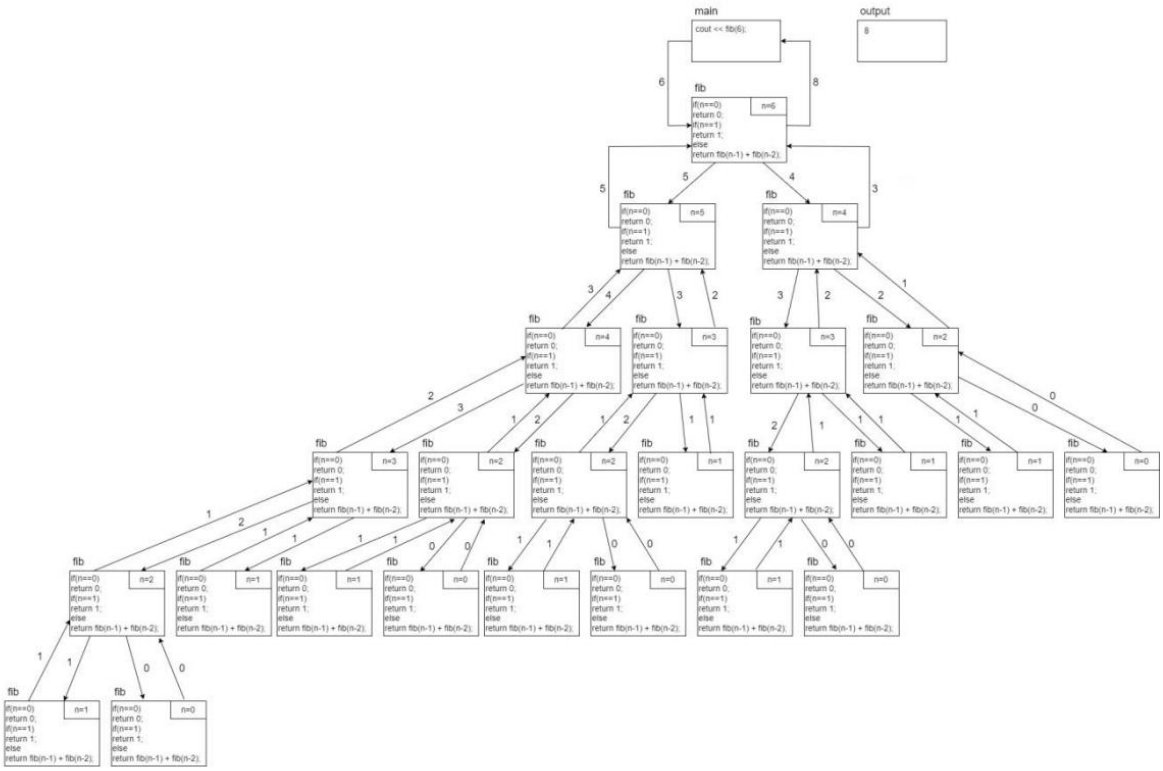
Factorial

The factorial sequence has a complexity of Big O(n) because the graph moves linearly which means all you have to do is add 1 to n to the find number of boxes.



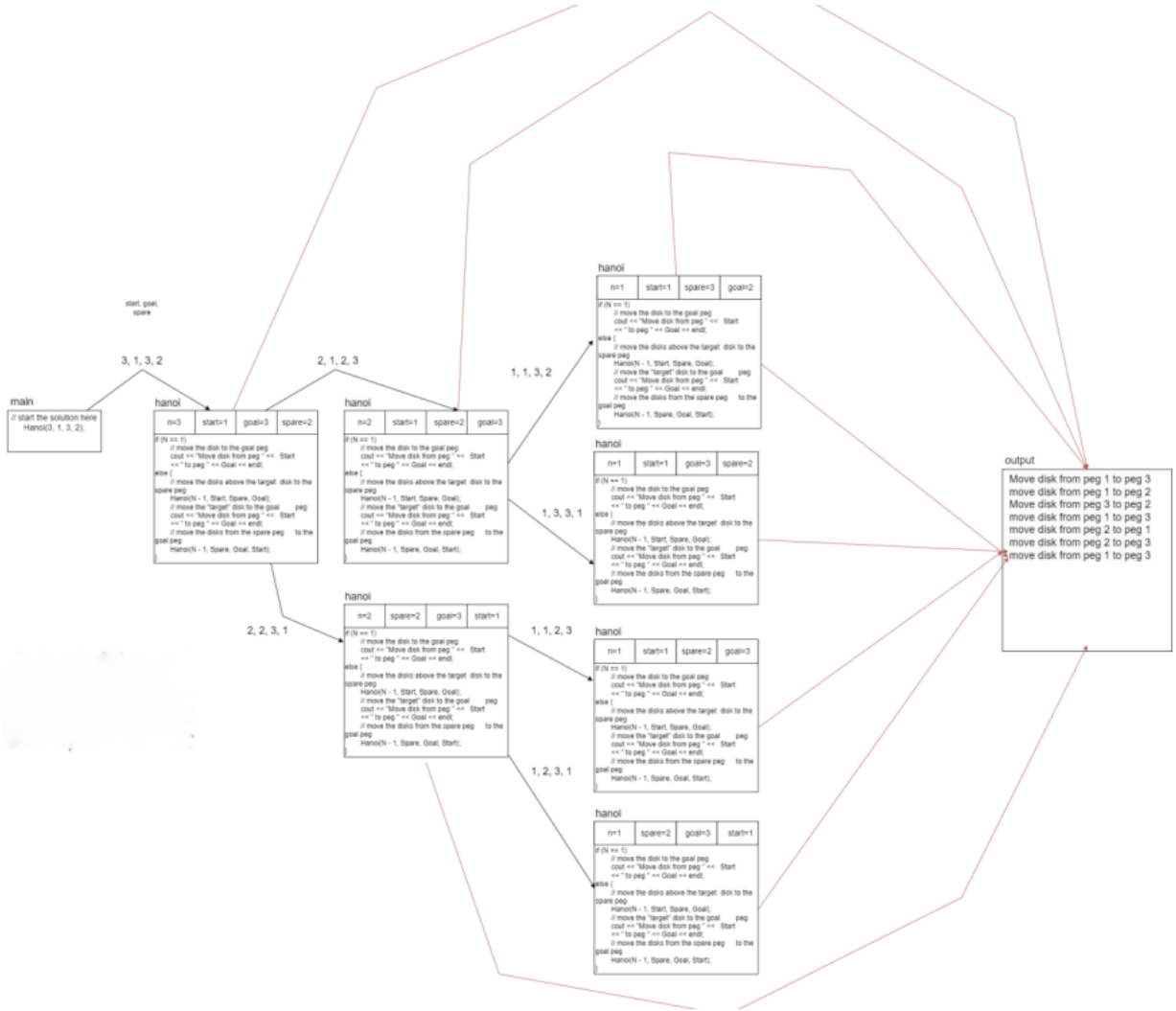
Fibonacci

The Fibonacci sequence has a time complexity of Big O(2^n) because each number in the sequence is added up as the sum of the last two numbers. As a result, a recursive pattern is formed.



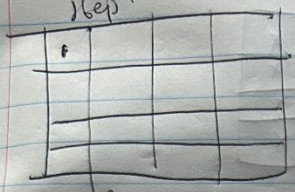
Hanoi

Hanoi has a sequence of Big O (2^n). This is because every time the function is called, it creates two additional smaller functions. This makes the formula 2^n-1.


start, goal, spare
3, 1, 3, 2
2, 1, 2, 3
1, 1, 3, 2
1, 3, 3, 1
2, 2, 3, 1
1, 1, 2, 3
1, 2, 3, 1
main
// start the solution here
Hanoi(3, 1, 3, 2);
hanoi
n=3 start=1 goal=3 spare=2
hanoi
n=2 start=1 spare=2 goal=3
hanoi
n=2 spare=2 goal=3 start=1
hanoi
n=1 start=1 spare=3 goal=2
hanoi
n=1 start=1 goal=3 spare=2
hanoi
n=1 start=1 spare=2 goal=3
hanoi
n=1 spare=2 goal=3 start=1
output
Move disk from peg 1 to peg 3
move disk from peg 1 to peg 2
Move disk from peg 3 to peg 2
move disk from peg 1 to peg 3
move disk from peg 2 to peg 1
move disk from peg 2 to peg 3
move disk from peg 1 to peg 3

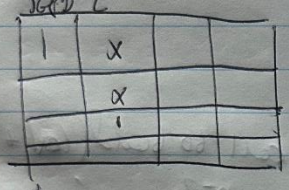
4 Queens

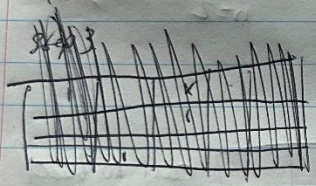
Step 1

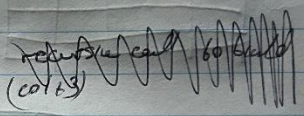


place queen to first block and recursive call.

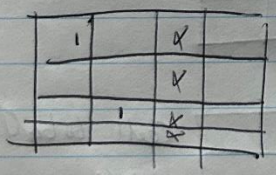
Step 2



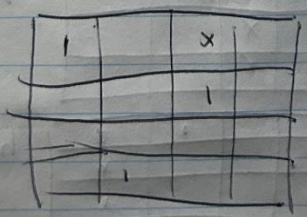




Step 3

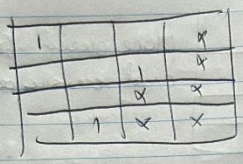


Step 4



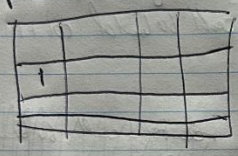
recursive call to backtrack (col+3)

Step 5



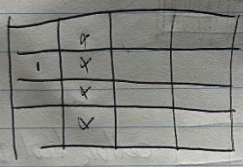
collar returns false

Step 6



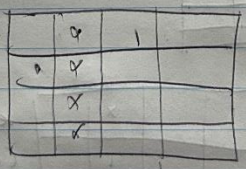
call to board(col + 1)
n=2

Step 7.



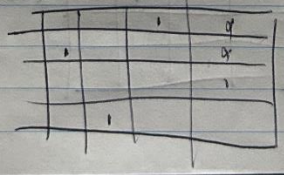
call to board
(col+2) i.e (4=3)

Step 8

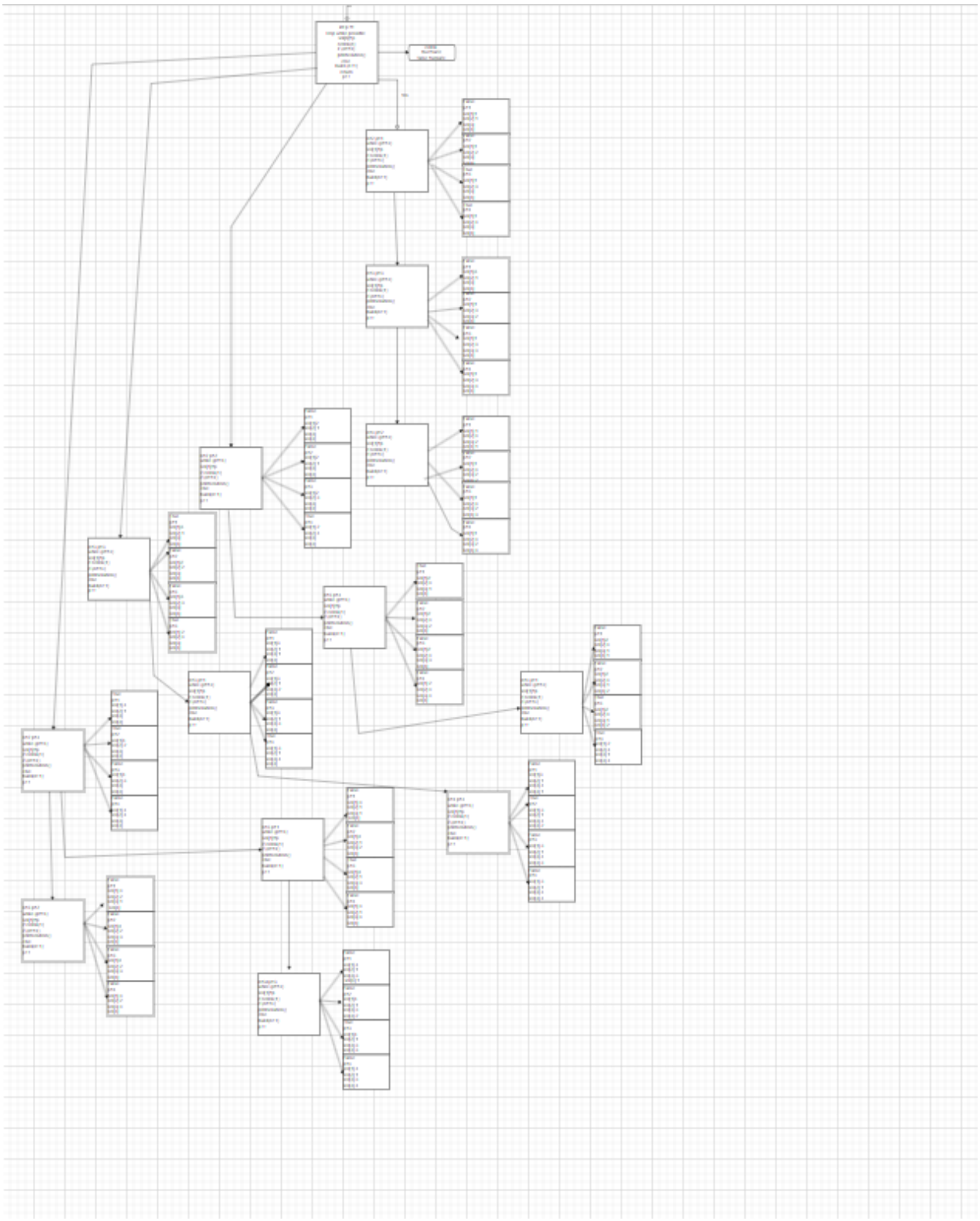


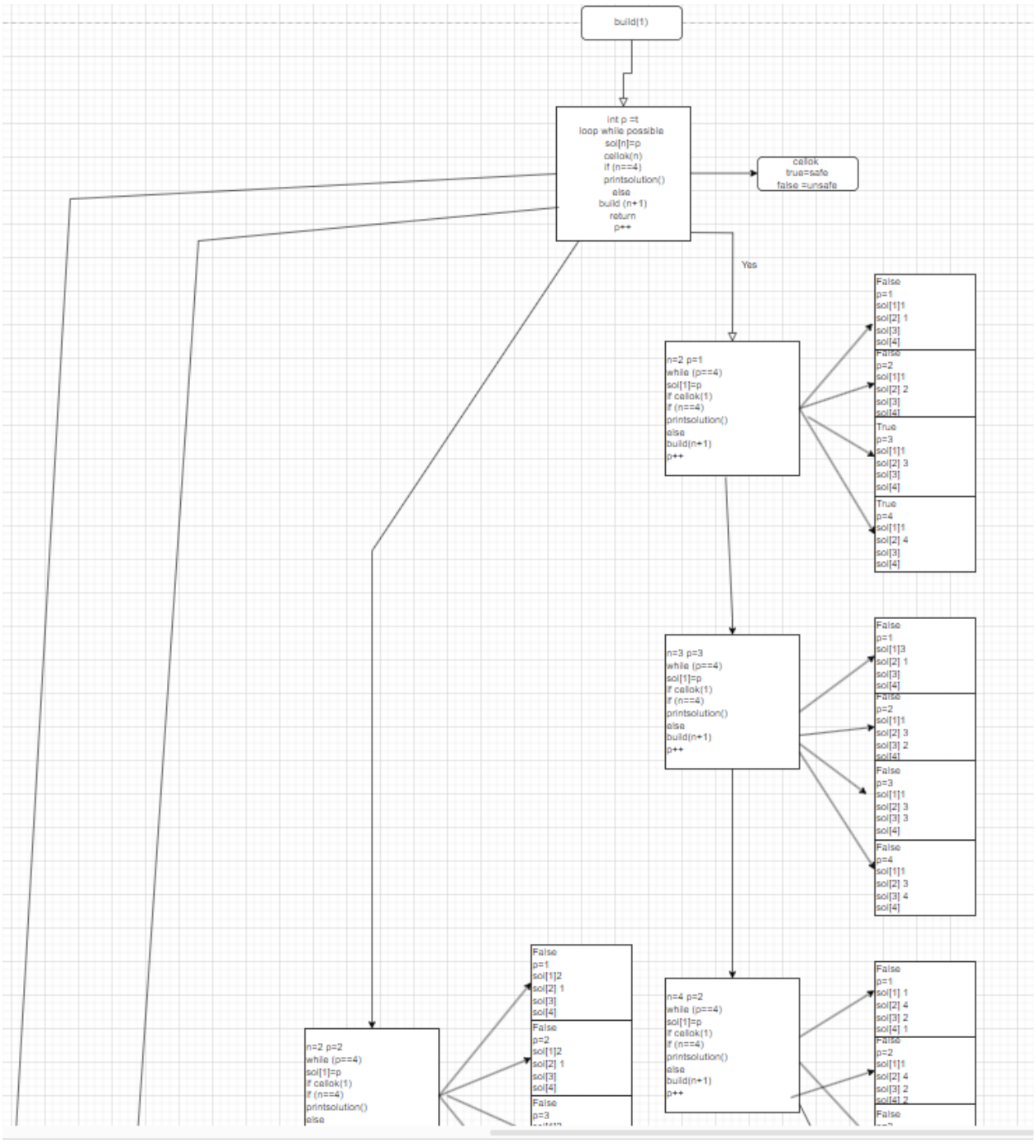
call to board (col+3)
n=4

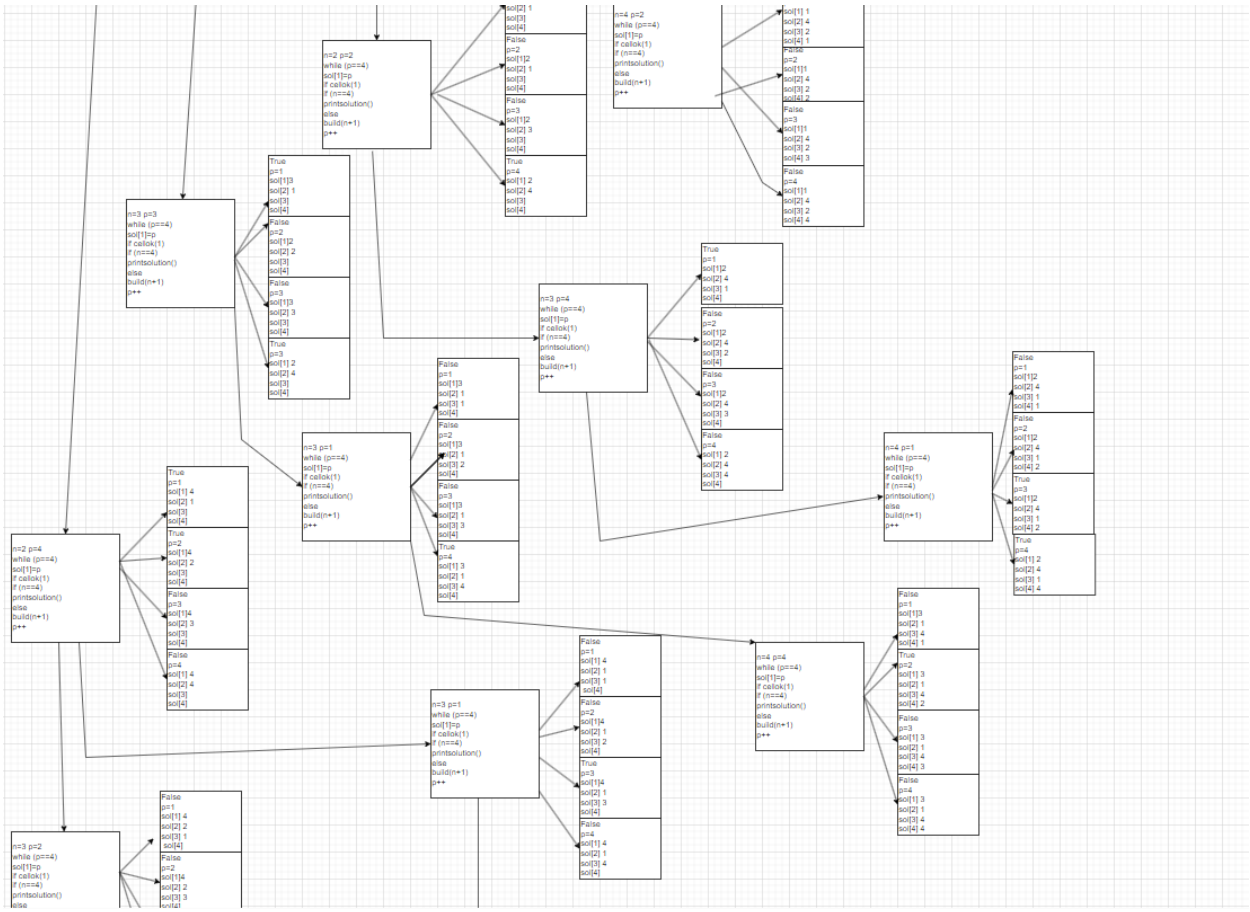
Step 9

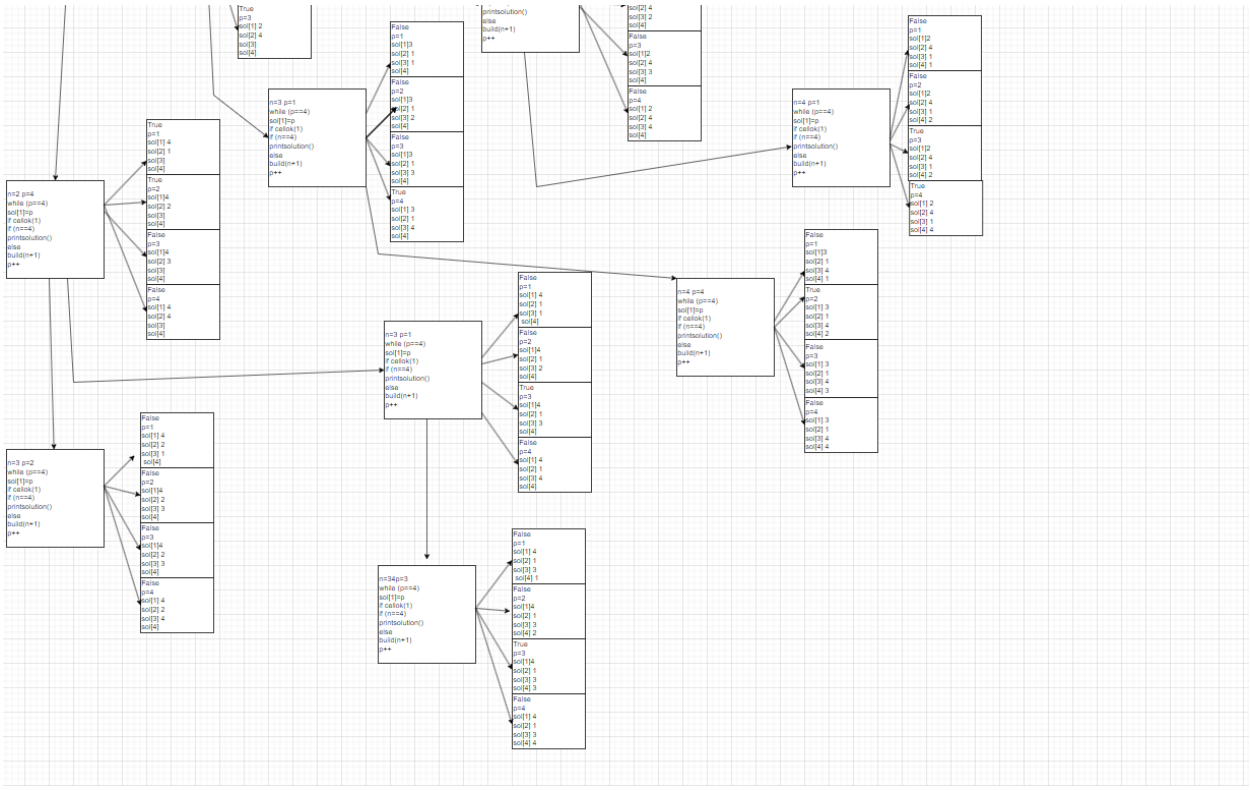


here n=4




build(1)
int p =t
loop while possible
sol[n]=p
callok(n)
if (n==4)
printsolution()
else
build (n+1)
return
p++
callok
true=safe
false =unsafe
Yes
n=2 p=1
while (p==4)
sol[1]=p
if callok(1)
if (n==4)
printsolution()
else
build(n+1)
p++
False
p=1
sol[1]1
sol[2] 1
sol[3]
sol[4]
False
p=2
sol[1]1
sol[2] 2
sol[3]
sol[4]
True
p=3
sol[1]1
sol[2] 3
sol[3]
sol[4]
True
p=4
sol[1]1
sol[2] 4
sol[3]
sol[4]
n=3 p=3
while (p==4)
sol[1]=p
if callok(1)
if (n==4)
printsolution()
else
build(n+1)
p++
False
p=1
sol[1]3
sol[2] 1
sol[3]
sol[4]
False
p=2
sol[1]1
sol[2] 3
sol[3] 2
sol[4]
False
p=3
sol[1]1
sol[2] 3
sol[3] 3
sol[4]
False
p=4
sol[1]1
sol[2] 3
sol[3] 4
sol[4]
n=2 p=2
while (p==4)
sol[1]=p
if callok(1)
if (n==4)
printsolution()
else
False
p=1
sol[1]2
sol[2] 1
sol[3]
sol[4]
False
p=2
sol[1]2
sol[2] 1
sol[3]
sol[4]
False
p=3
n=4 p=2
while (p==4)
sol[1]=p
if callok(1)
if (n==4)
printsolution()
else
build(n+1)
p++
False
p=1
sol[1] 1
sol[2] 4
sol[3] 2
sol[4] 1
False
p=2
sol[1]1
sol[2] 4
sol[3] 2
sol[4] 2
False


n=2 p=2
while (p==4)
sol[1]=p
if calok(1)
if (n==4)
printsolution()
else
build(n+1)
p++
n=3 p=3
n=3 p=1
n=3 p=4
n=4 p=2
n=2 p=4
n=3 p=2
n=4 p=4
n=4 p=1
True
False


n=2 p=4
n=3 p=1
n=4 p=4
n=3 p=2
n=3 p=3
n=34p=3
True
False

The queens sequence has a big O(n!). The algorithm backtracks to find all possible solutions of formations of queens on the boat.