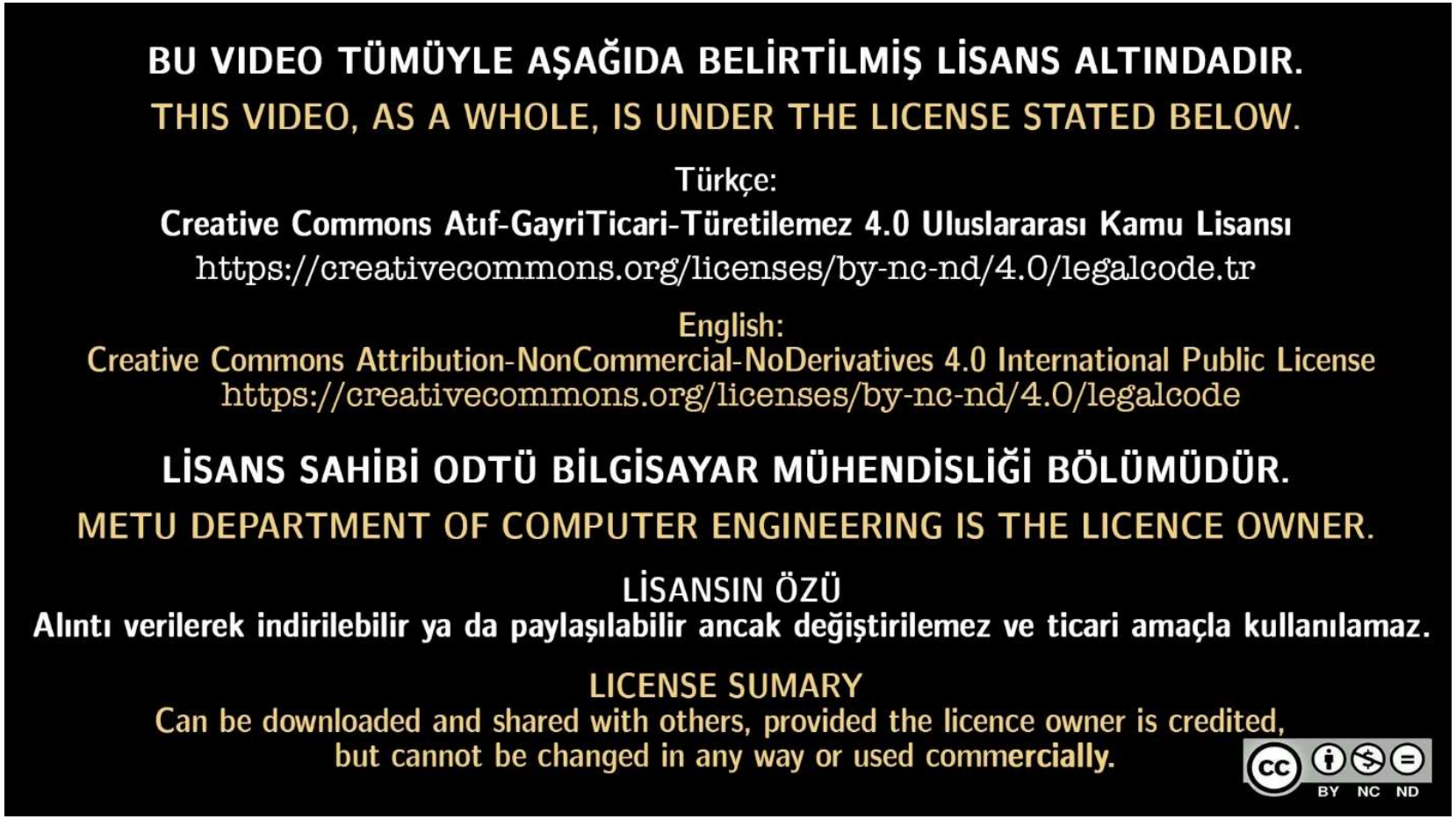
Warning



Recursion (Revisited)

Dr. Ismail Sengor ALTINGOVDE
METU

Recursion

- Recursion is an extremely powerful problem-solving technique
  - Breaks a problem in smaller identical problems
  - An alternative to iteration, which involves loops
- A sequential search is iterative
  - Starts at the beginning of the collection
  - Looks at every item in the collection in order
- A binary search is recursive
  - Repeatedly halves the collection and determines which half could contain the item
  - Uses a divide and conquer strategy

Recursive Solutions

- Facts about a recursive solution
  - A recursive method calls itself
  - Each recursive call solves an identical, but smaller problem
  - A test for the base case enables the recursive calls to stop
    - Base case: a known case in a recursive definition
  - Eventually, one of the smaller problems must be the base case

Recursive Solutions

- Four questions for construction of recursive solutions
  - How can you define the problem in terms of a smaller problem of the same type?
  - How does each recursive call diminish the size of the problem?
  - What instance of the problem can serve as the base case?
  - As the problem size diminishes, will you reach this base case?

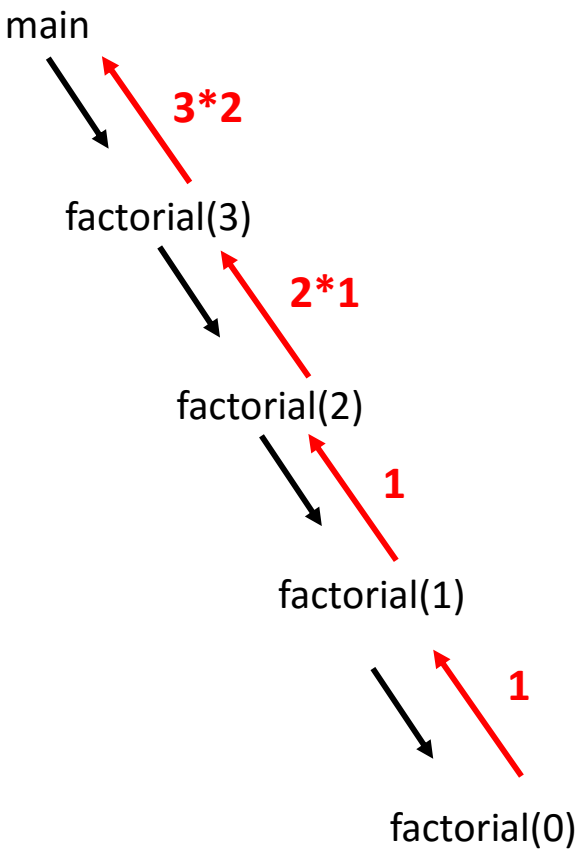
Recursive Functions

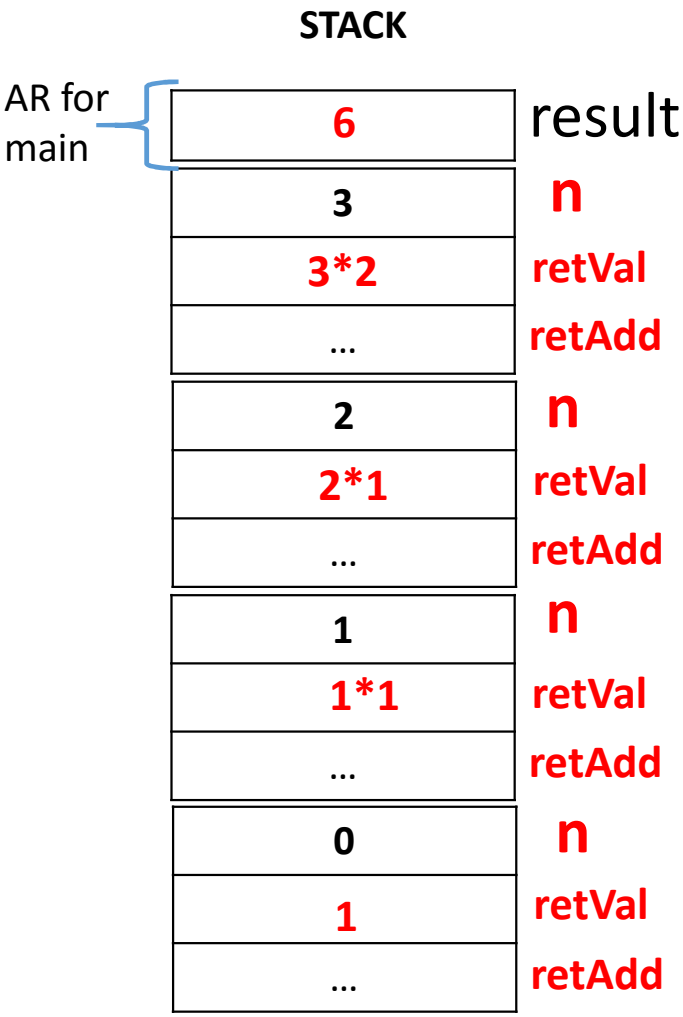
- A function may **directly** or **indirectly** call itself.

```
int factorial(int n)
{ if (n == 0)
    return 1;
  else
    return n*factorial(n -1); }
```

```
int main(void)
{ int result;
  result = factorial(3);
}
```

What happens?



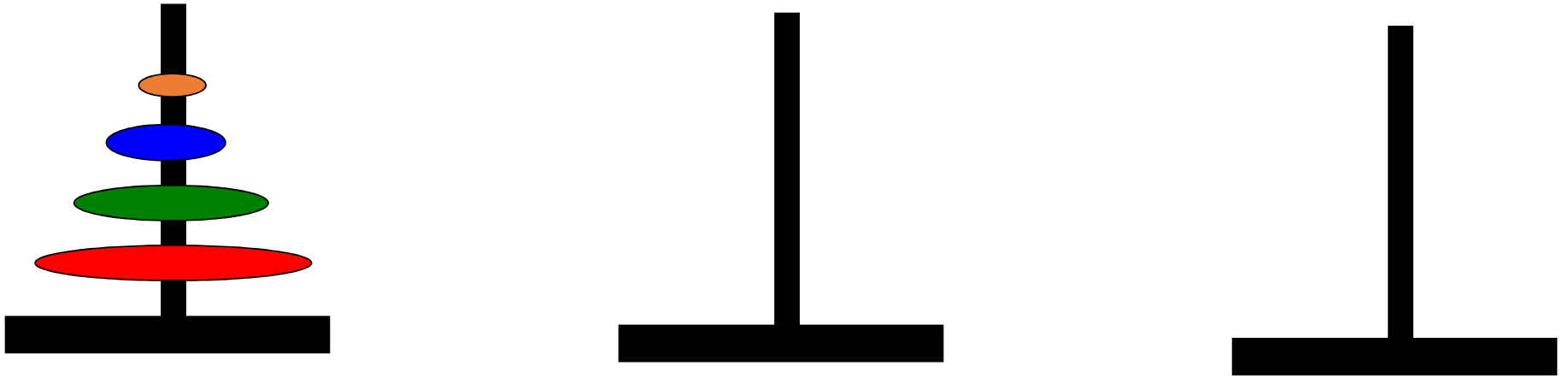


Looks expensive!

Towers of Hanoi
N-Queens
Coin Problem

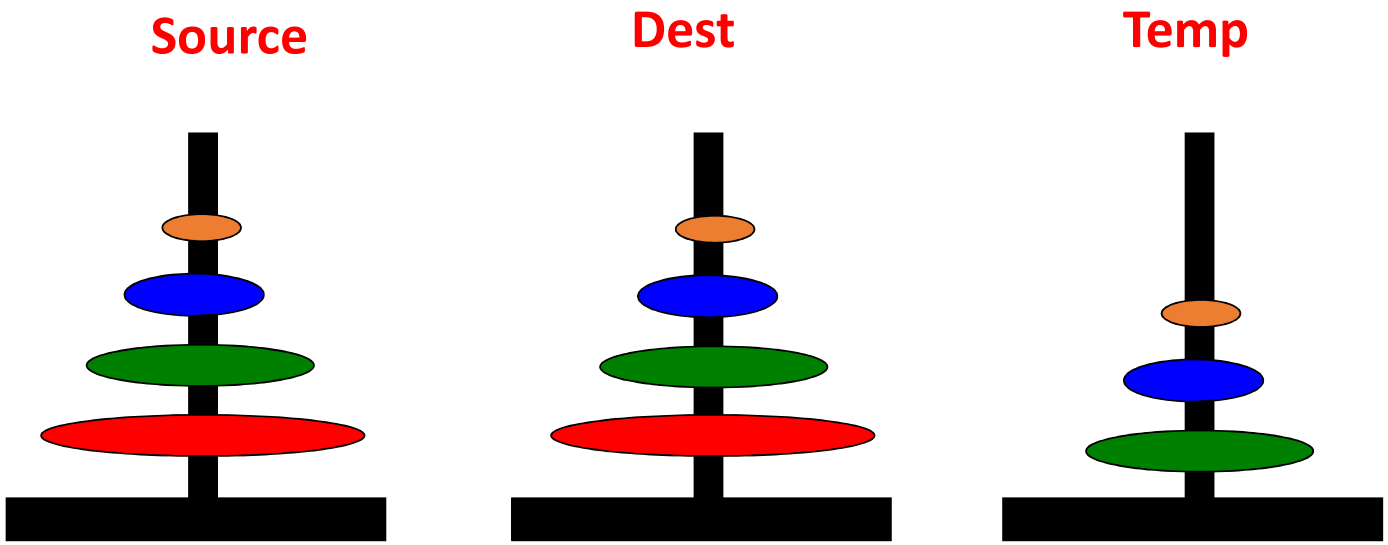
Problem

- Move disks from one peg (source) to another peg (destination), using a third peg as temporary
- RULES:
  - Exactly one disk is moved at a time
  - A larger disk cant be placed above a smaller disk



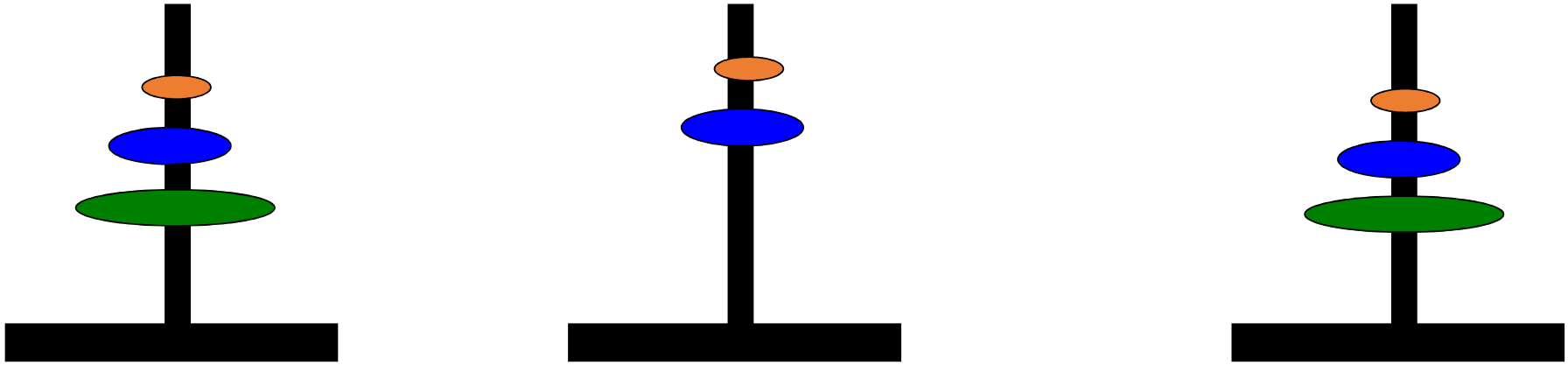
Problem: Move 4 disks from peg1 to peg-2

Involves moving 3 disks from **peg-1** to **peg- 3**, and then
moving 3 disk from **peg-3** to **peg-2**



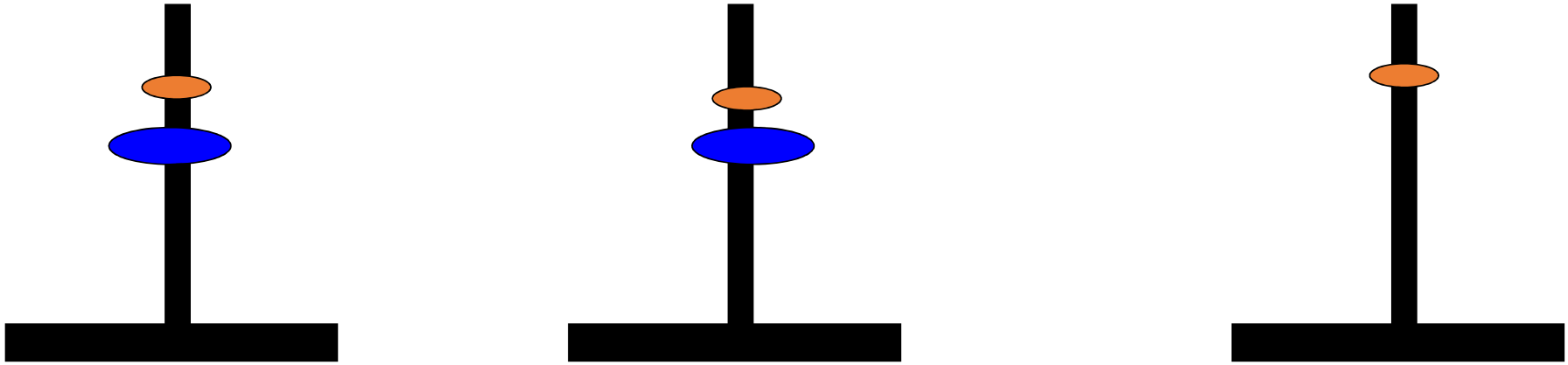
Smaller problem:
Move 3 disks from peg-1 to peg-3

Involves moving 2 disks from **peg-1** to **peg- 2**, and then
moving 2 disks from **peg-2** to **peg-3**



Smaller problem:
Move 2 disks from peg-1 to peg-2

Involves moving 1 disks from **peg-1** to **peg- 3**, and then
moving 1 disks from **peg-3** to **peg-2**



```
/* move n disks from peg x (source) to peg y (dest), z is temporary */
void move(int n, int x, int y, int z)
{  if (n== 1)
      printf("move top disk on %d to %d\n", x, y);
   else
   {  /* move n-1 disks from: x to z  */
      move(n-1 , x , z , y);
      printf("move top disk on %d to %d\n", x, y);
      /* move n-1 disks from  z to  y */
      move(n-1  , z , y   x );
   } }
```

move(3, 1, 2, 3)

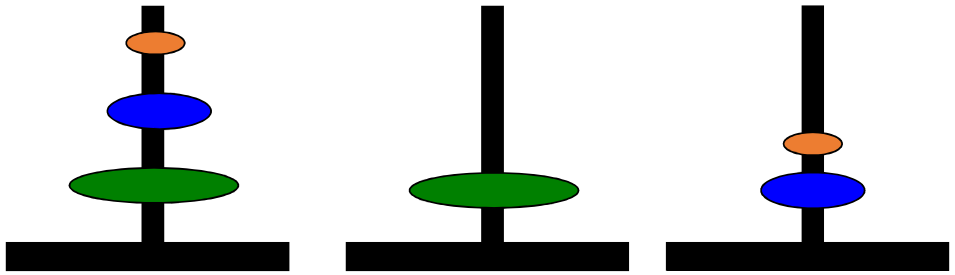
move(2, 1, 3, 2) printf(move top 1 to 2) move(2, 3, 2, 1)

move(1, 1, 2, 3) printf(move top 1 to 3) move(1, 2, 3, 1)

printf(move top 1 to 2)

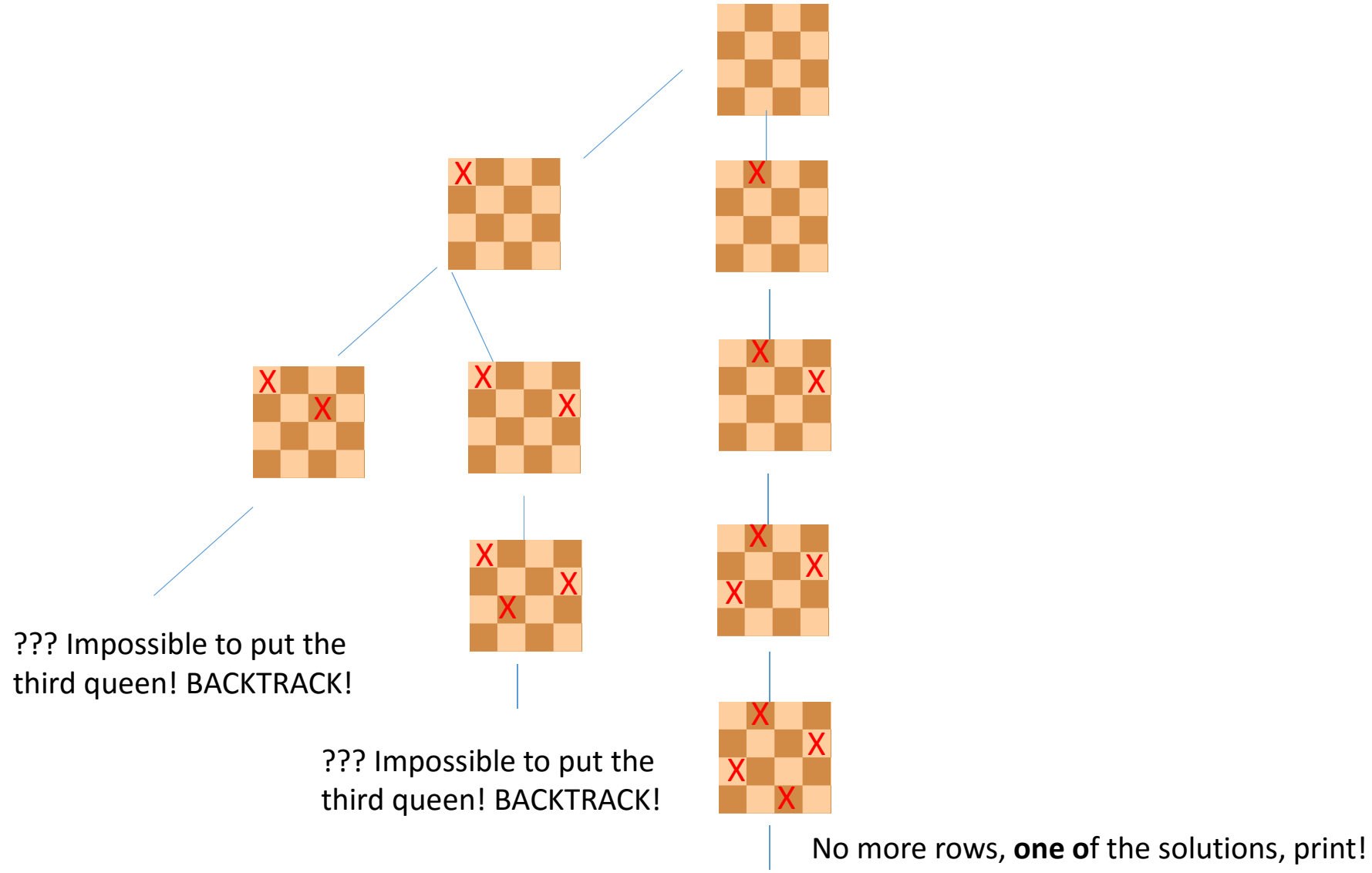
printf(move top 2 to 3)

```
{  if (n== 1)
     printf("move top disk on %d to %d\n", x, y);
   else
   {  move(  n-1 ,  x ,  z ,  y );
      printf("move top disk on %d to %d\n", x, y);
      move(  n-1,  z ,  y ,  x );
   } }
```



N-queens problem

- Place n queens on a nxn chess board, so that no two queens can attacak each other (i.e., no two queens are in same row, column, or diagonal).
- Solution with Backtracking:
  - Build a solution step by step
  - If at some step, it becomes clear that the current path can not lead solution, go back to previous step anf make another decision
- For N-queens:
  - Tentatively place a queen at a row
  - Recurse, if ok, done
  - If fails, undo tentative placement, try another one


??? Impossible to put the third queen! BACKTRACK!
??? Impossible to put the third queen! BACKTRACK!
No more rows, **one o**f the solutions, print!

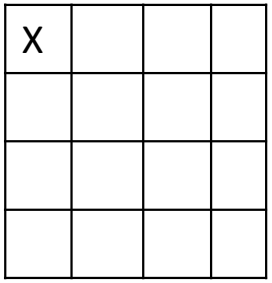
Pseudocode (not C syntax!!!)

```
RecNQueens(Q[n], r)
if r == n+1
    print Q
else   for j ← 1 to n
            legal ← true
            for i ← 1 to r-1
                if (Q[i] == j) or (Q[i] == j+r-i) or (Q[i] == j-r+i)
                    legal ← false
            if legal
                Q[r] ← j
                rec(Q[n], r+1)
```

Rec(Q[4], 1)

```
/* n=4, r=1 */
for j ← 1 to n
  legal ← true
  for i ← 1 to r-1
      if (Q[i] == j) or (Q[i] == j+r-i)
                     or (Q[i] == j-r+i)
  if legal
      Q[1] ← 1
      Rec(Q[4], 2)
```

j =1



RecQ[4], 2)

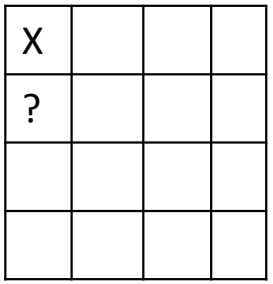
```
/* n=4, r=2 */
for j ← 1 to n
  legal ← true
  for i ← 1 to r -1
      if (Q[1] == 1) or (Q[1] == 1+2-1) or (Q[1] == 1-2+1)
      if (Q[1] == 2) or (Q[1] == 2+2-1) or (Q[1] == 2-2+1)
      if (Q[1] == 3) or (Q[1] == 3+2-1) or (Q[1] == 3-2+1)
  if legal
      Q[2] ← 3
      Rec(Q[4],3)
```

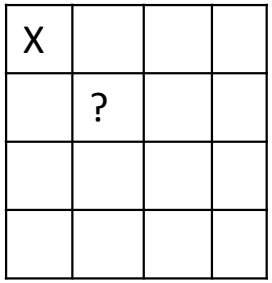
j=1 j=2 j=3

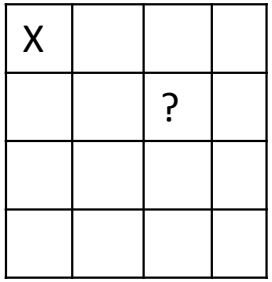
i=1

legal ← false

legal ← false







Coin Change Problem

- Input: An amount of money M, in cents
- Output: Smallest number of coins that adds up to M
  - Quarters (25c): q
  - Dimes (10c): d
  - Nickels (5c): n
  - Pennies (1c): p
  - Or, in general, c_1, c_2, …, c_d (*d* possible denominations)

Coin Change Problem

- Assume you have 1, 3, 5 cent coins
- Change 6 cent
- Possibilities:
  - 1 x 5, 1 x1: total 2
  - 2 x 3: total 2
  - 3 x 1, 1 x 3 total:4
  - 6 x 1 : total 6

Recursive solution

c= {1, 3, 5}

minNoCoins(m, c) = min

minNoCoins(m - 1) + 1
minNoCoins(m - 3) + 1
minNoCoins(m - 5) + 1

Warning