Recursion

When to use it and when not to use it

Basics of Recursion

- Recursion uses a method
- Within that method a call is made to the same method
- Somewhere within the method, the loop must be exited

Disadvantages

- 3 major disadvantages
- Slow algorithms
- e.g. recursive Fibonacci:

```
int fib (int n)
{
  if (n <= 2) return 1;
  else return fib (n - 1) + fib (n - 2);
}
```

Disadvantages Cont.

- Procedure call overhead: 30% slower
- Not much time lost though

- Data on stack (stack overflow)
- Not much of a problem these days
- Watch out for using more than the limit

Disadvantages: Summary

- The first one is the one to watch out for (slow algorithms)
- The other two can almost be ignored (procedure call overheads and data on stack)

What to Use Instead

- If iteration is easy to program, use it
- If dynamic programming works, use it instead

Why Use Recursion?

- Easy to program
- Easy to debug
- Shorter than dynamic programming

n Queens Problem

- Place n queens on an n x n chess board so that no queen is attacked by another queen
- Find out the total number of configurations

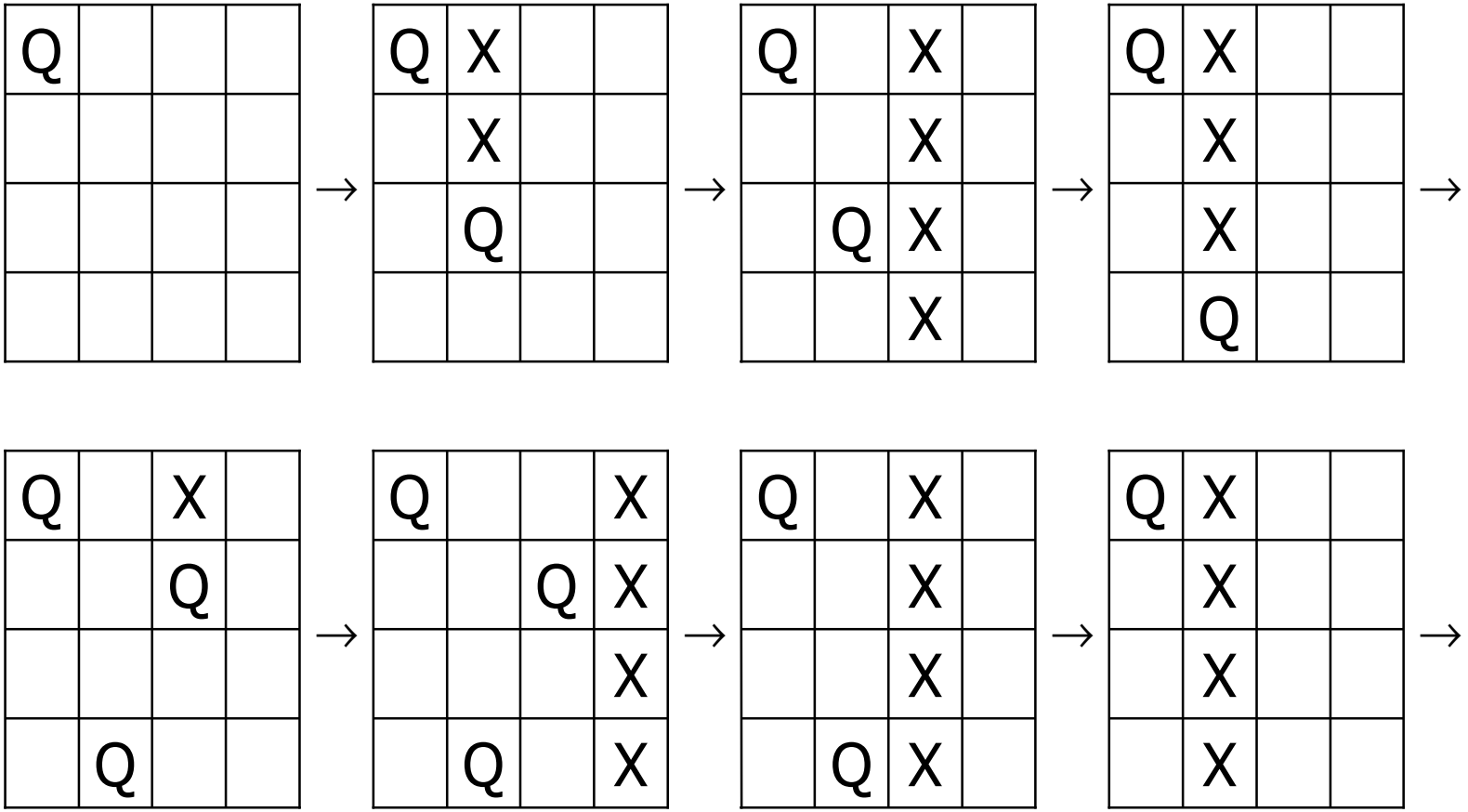
nQueensSolution

- The most obvious solution is to recursively add queens, trying all possible placements
- It is easy to see that there can only be one queen per column
- At each step, you know the position of the queens for the first i columns

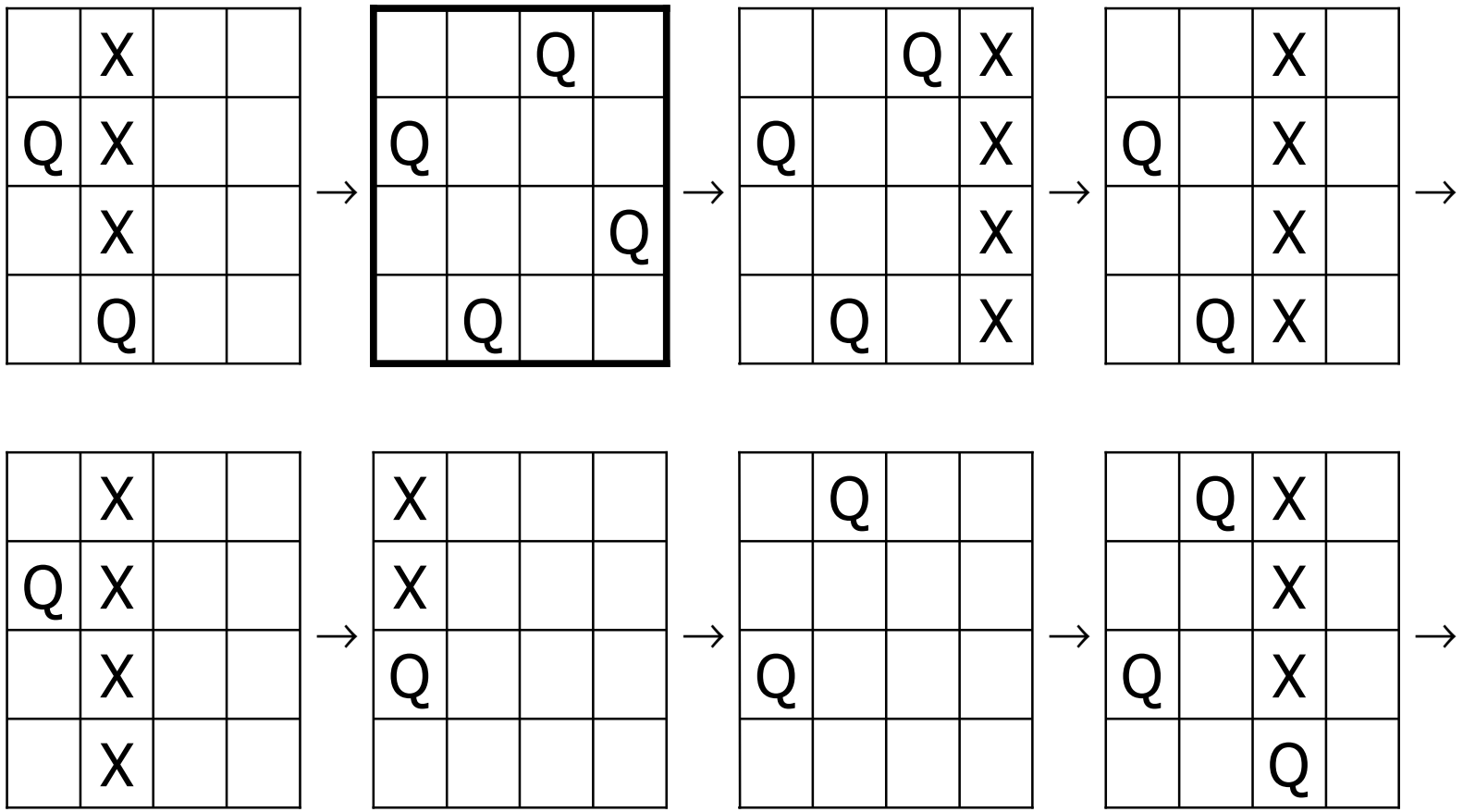
nQueensSolutionCount

- You try to place a queen in the i+1 column
- If successful, you move on to the next step, trying to place a queen in the i+2 column

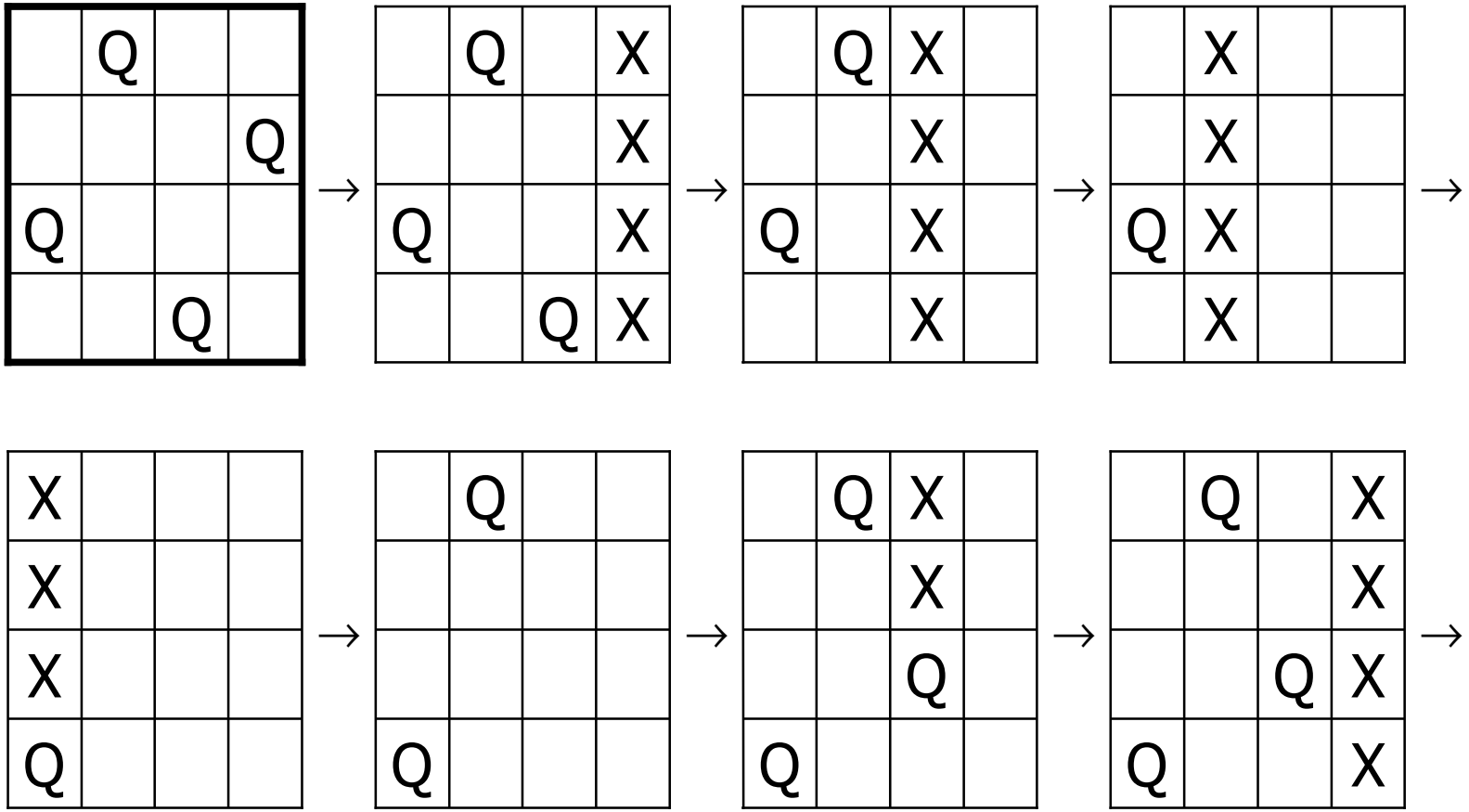
nQueensSolutionCount



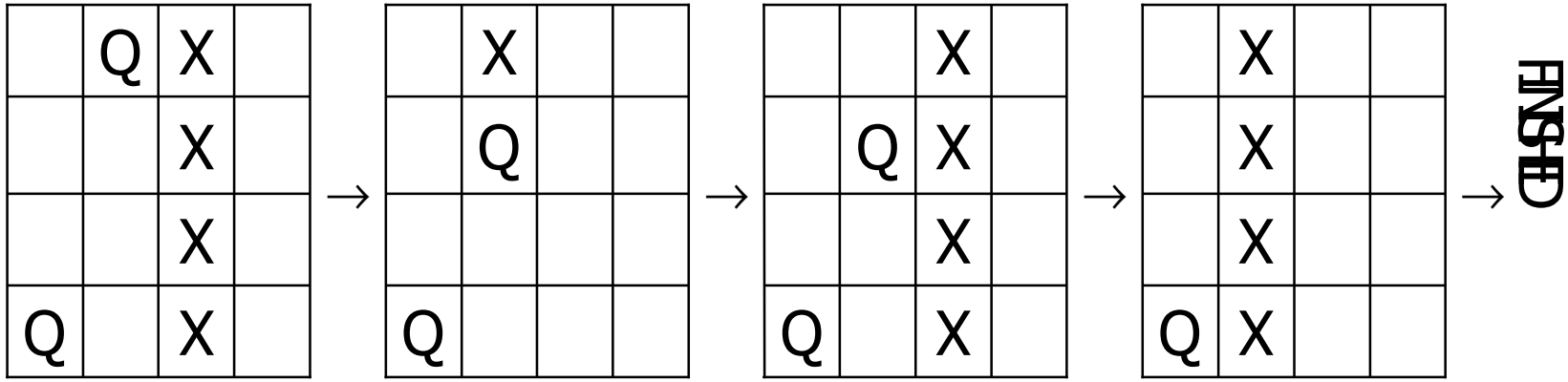
nQueensSolutionCount



nQueens Solution Count



nQueensSolutionCount



nQueens Solution Count

```
void search (int col)
{
  if (col == n)
  {
    count++;
    return;
  }
  for (int row = 0; row < n; row++)
    if (not_attacked (row, col))
    {
      board [col] = row;
      search (col + 1);
    }
}
```

nQueensSolutionCount

- The `not_attacked` method simply checks to see if the queen can be placed in a particular block

nQueens Analysis

- Since the number of possibilities is small, recursion is quick enough in this case
- For a 10x10 board it runs for 0.1 s
- This solution is also an example of depth first search
- In questions similar to this, use recursion, since it is easier to program

Little Flower Shop IOI'99

- You want to arrange the window of your flower shop in a most pleasant way. You have F bunches of flowers, each being of a different kind and at least as many vases (V) ordered in a row.
- The bunches are moveable and are uniquely identified by integers between 1 and F. These id-numbers have a significance. They determine the required order of appearance of the flower bunches in the row of vases. The bunch i must be on the left of bunch j whenever i < j.
- Excess vases will be left empty. A vase can only hold one bunch of flowers.

Little Flower Shop Cost

- Each vase has a distinct characteristic (just like flowers do). Hence, putting a bunch of flowers in a vase results in a certain aesthetic value. These values are presented in the following table:

FLOWERS	VASES					
	1	**2**	**3**	**4**	**5**
1	7	23	-5	24	16
2	5	21	-4	10	23
3	-21	5	-4	-20	20

Flow Solution 1

- We could use recursion:
- ```
  int flo (int f, int v)
  {
    if (f == F)
      return 0;
    int m = 0;
    for (; v <= V - F + f ; v++)
      m = max (m, grid [f][v] +
           flo (f + 1, v));
    return m;
  }
  ```
  ```

Flower Solution 1 Analysis

- In the initial problem, the case numbers must also be output
- It is far too slow
- It tries every solution to find the maximum

Flower Solution 2

- The first solution was too slow
- This time we'll try dynamic programing
- Start from the second last row
- Scan each row from left to right
- Add the maximum value in each block

Flower Solution 2 Cont.

```
int m;
for (int i = F - 2; i >= 0; i--)
{
  m = -100;
  for (int j = V - F + i; j >= i;j--)
  {
    m = max (m, grid [i + 1][j + 1]);
    grid [i][j] += m;
  }
```

Flower Solution 2 Cont.

- ```
  int answer = -1000000;
  for (int i = 0; i < F; i++)
    ans = max (ans, grid [0][i]);
  ```
  ```

Flow Solution 2 Cont.

- The solution is illustrated in the tables on the right
- The top table is before
- The bottom table is after
- Yellow numbers were changed

	1	2	3	4	5
1	7	23	-5	24	16
2	5	21	-4	10	23
3	-21	5	-4	-20	20

	1	2	3	4	5
1	41	53	25	24	16
2	5	41	16	30	23
3	-21	5	-4	-20	20

Flower Solution 2 Analysis

- Only the necessary calculations are done
- Much faster algorithm (0.05s, quicker than 2s allowed)
- To determine the base number is easy

Flow Analysis

- In this case, recursion is not fast enough
- As always, if dynamic programming works, use it
- Not all problems can be solved with dynamic programming since it uses more memory

Conclusion

- When recursion runs within the time limit, use it
- If it doesn't, try using dynamic programming or another technique
- If you can't do it any other way, use recursion to get partial marks

Conclusion Cont.

- Remember, many of the techniques use recursion, so it is important to know well
- Most graph theory (shortest paths, etc.) uses recursion