More Recursion

Tiziana Ligorio
tligorio@hunter.cuny.edu

Today's Plan

Recursion Review

8 Qeens Problem

Permutations

Combinations

Announcements and Syllabus Check

Types of Recursion

Reverse String:

- single recursive call
- Base case: stop => no return value

Dictionary:

- split problem into halves but solve only 1
- Base case: stop => no return value

Fractal Tree:

- split problem into halves and solve both
- Base case: stop => no return value

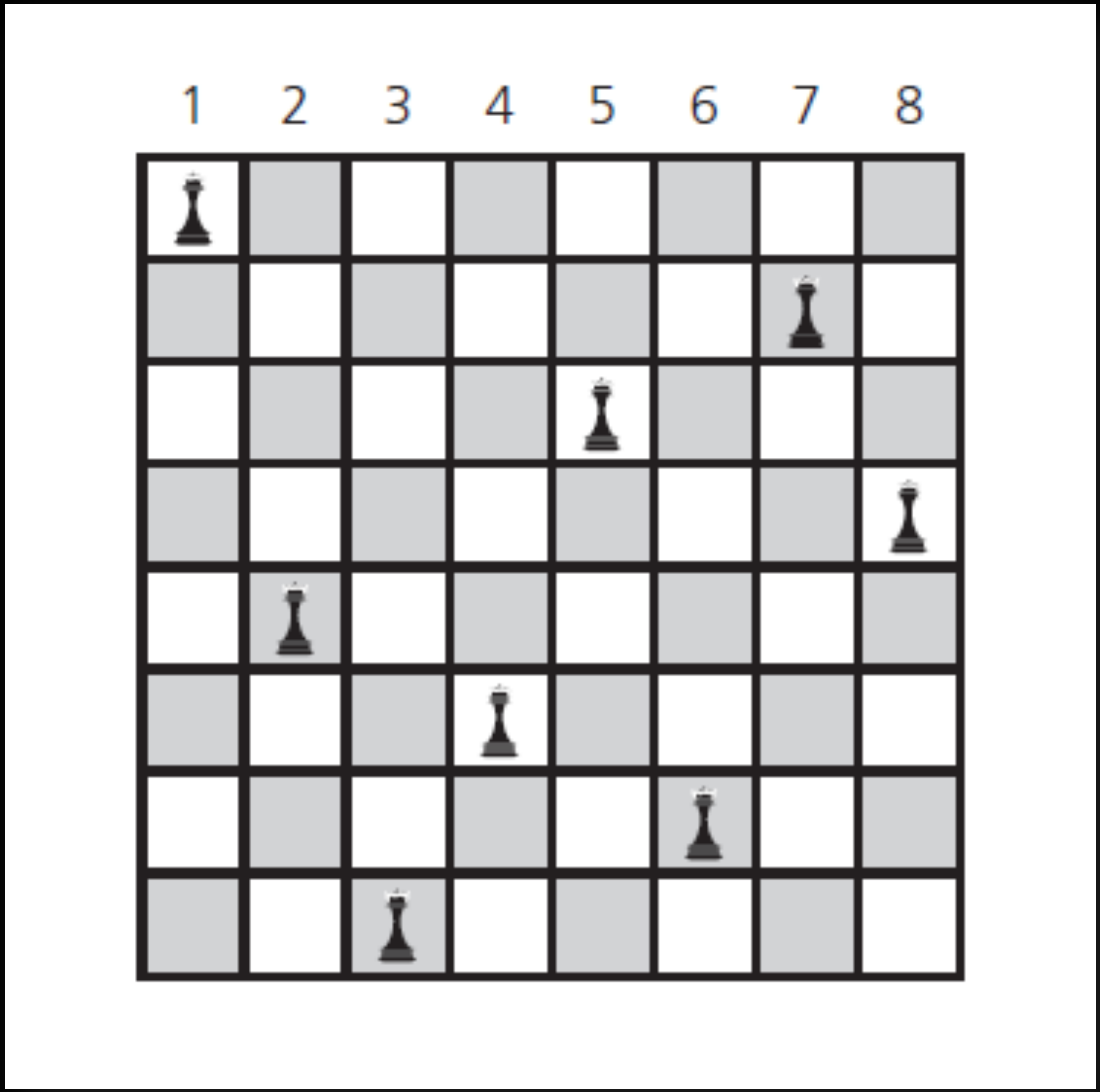
Factorial:

- single recursive call
- Base case: return a value for computation in each recursive call

Recursive Searching and Sorting

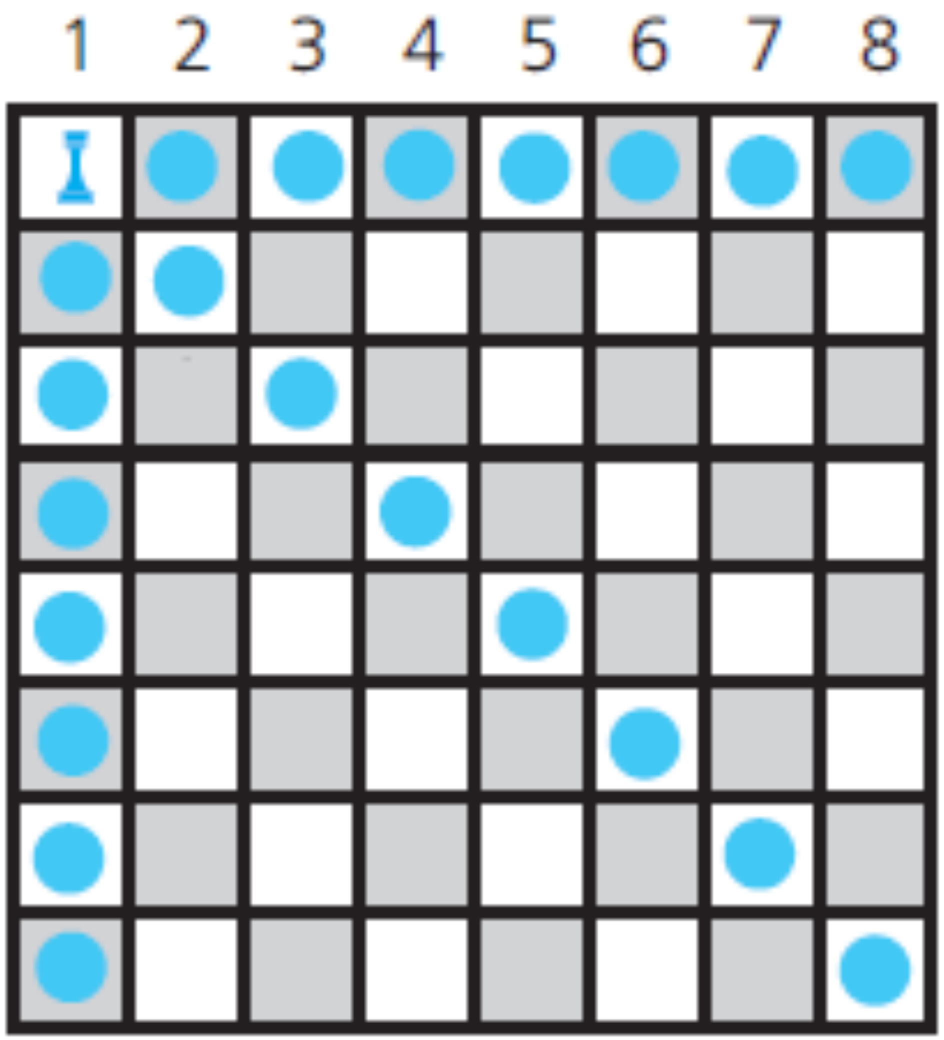
Binary Search	$O(\log n)$	$\Omega(1)$
MergeSort	$O(n \log n)$	$\Omega(n \log n)$
QuickSort	$O(n^2)$	$\Omega(n \log n)$

The Eight Queens Problem



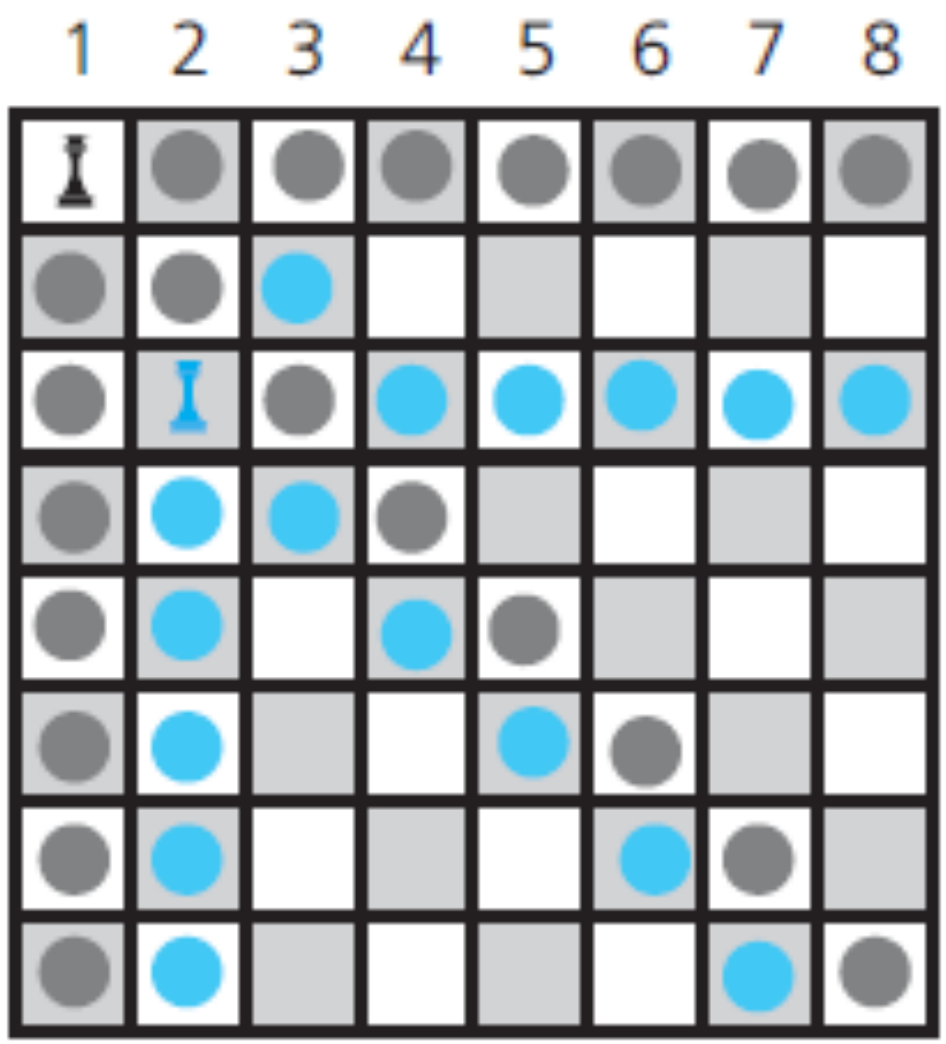
Place 8 Queens on the board s.t. no queen is on the same row, column or diagonal

The Eight Queens Problem



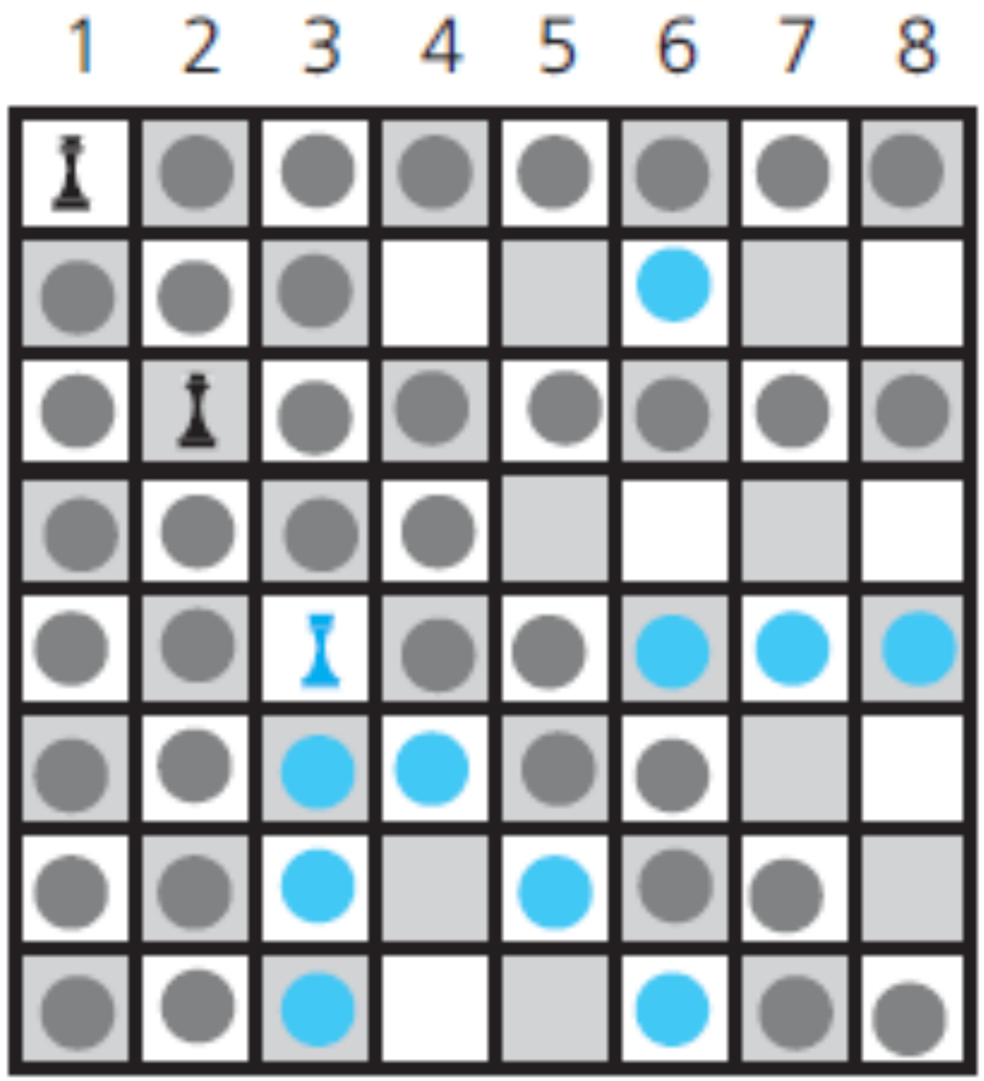
(a) The first queen in column 1

The Eight Queens Problem



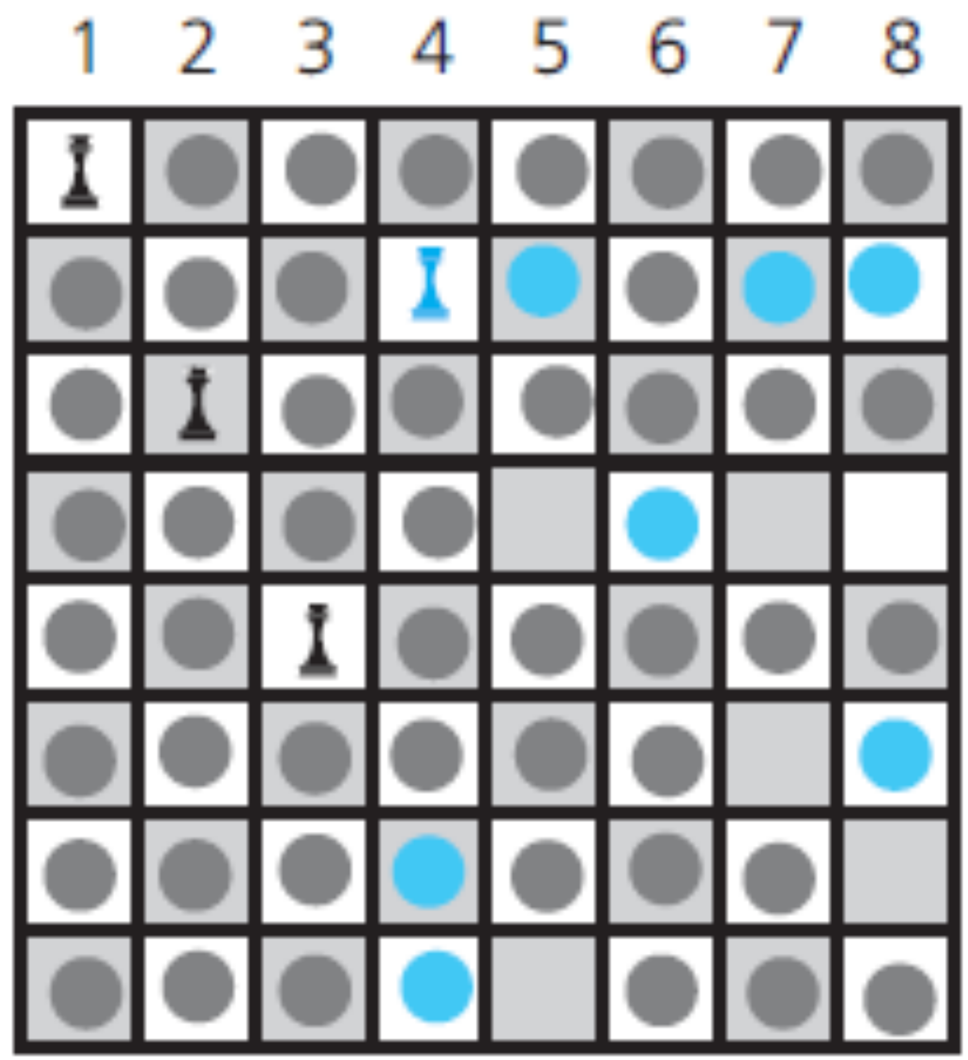
(b) The second queen in column 2

The Eight Queens Problem



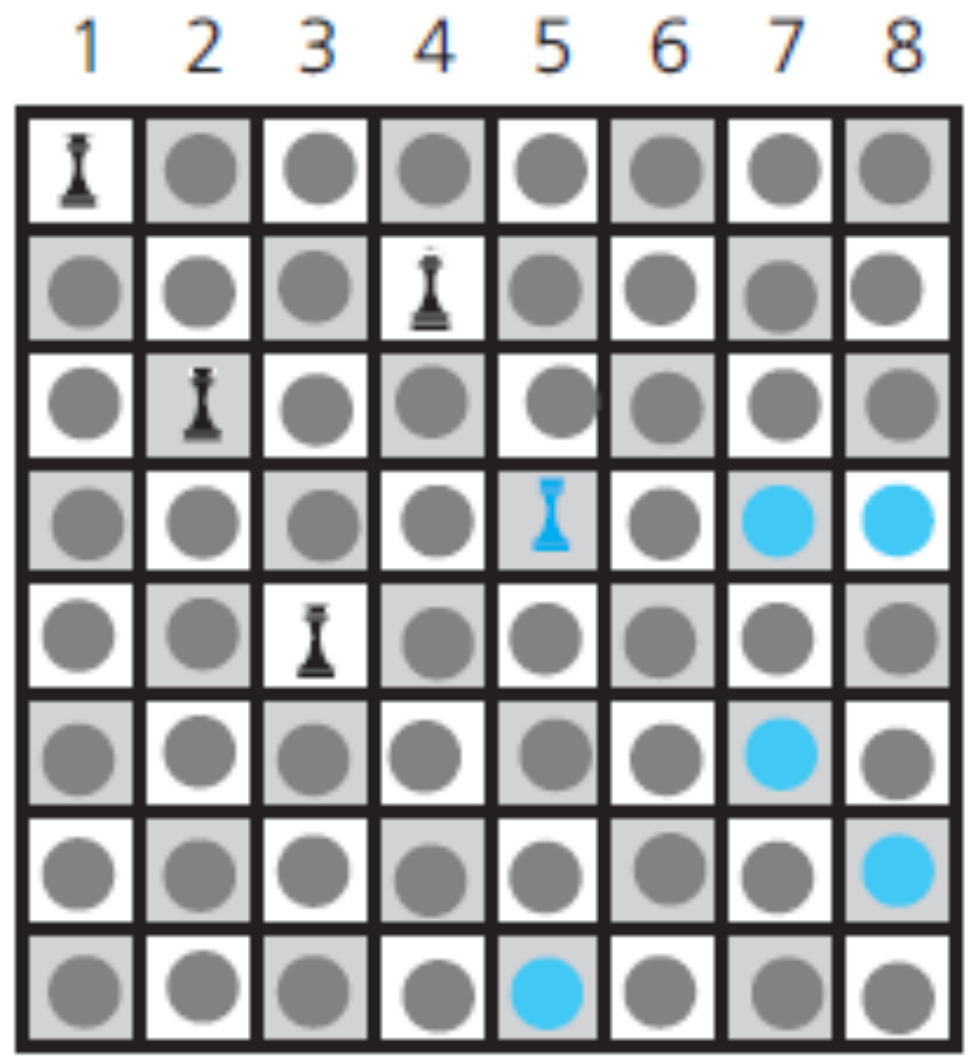
(c) The third queen in column 3

The Eight Queens Problem



(d) The fourth queen in column 4

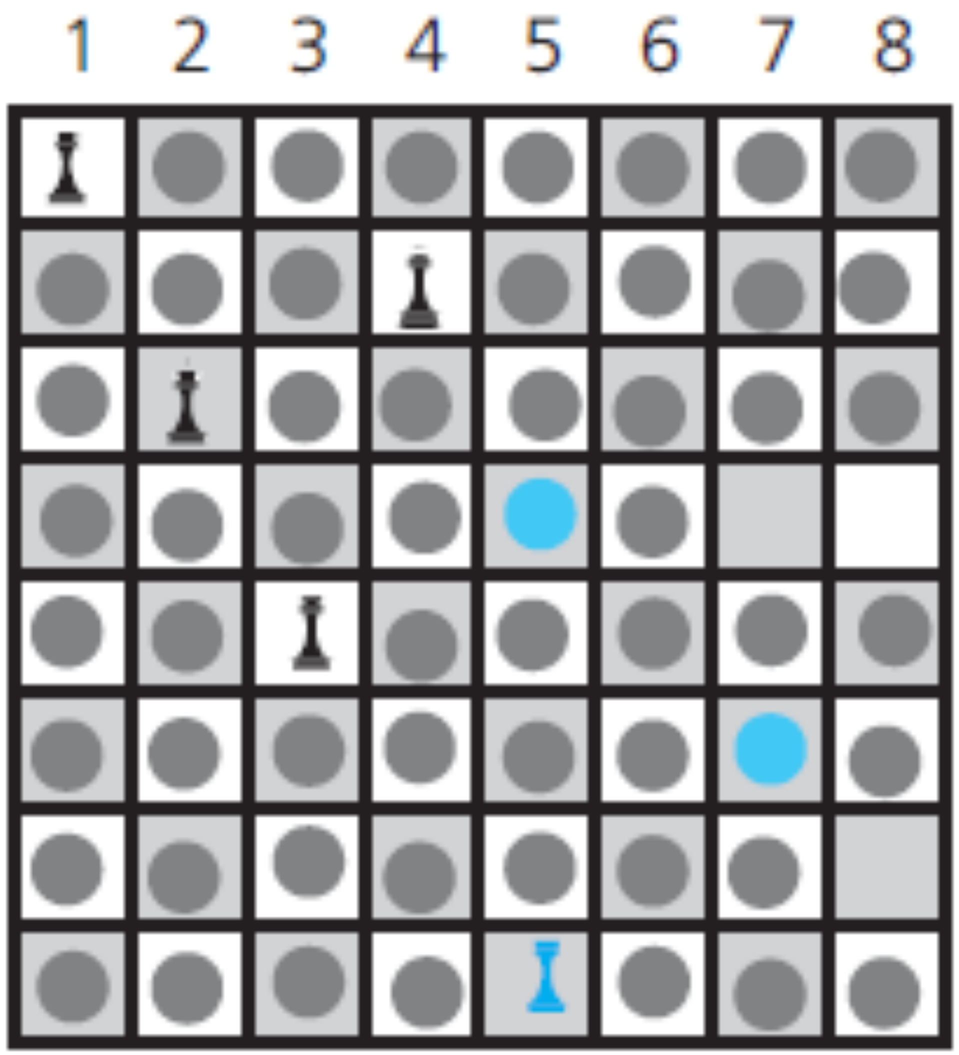
The Eight Queens Problem



(e) The five queens can attack all of column 6

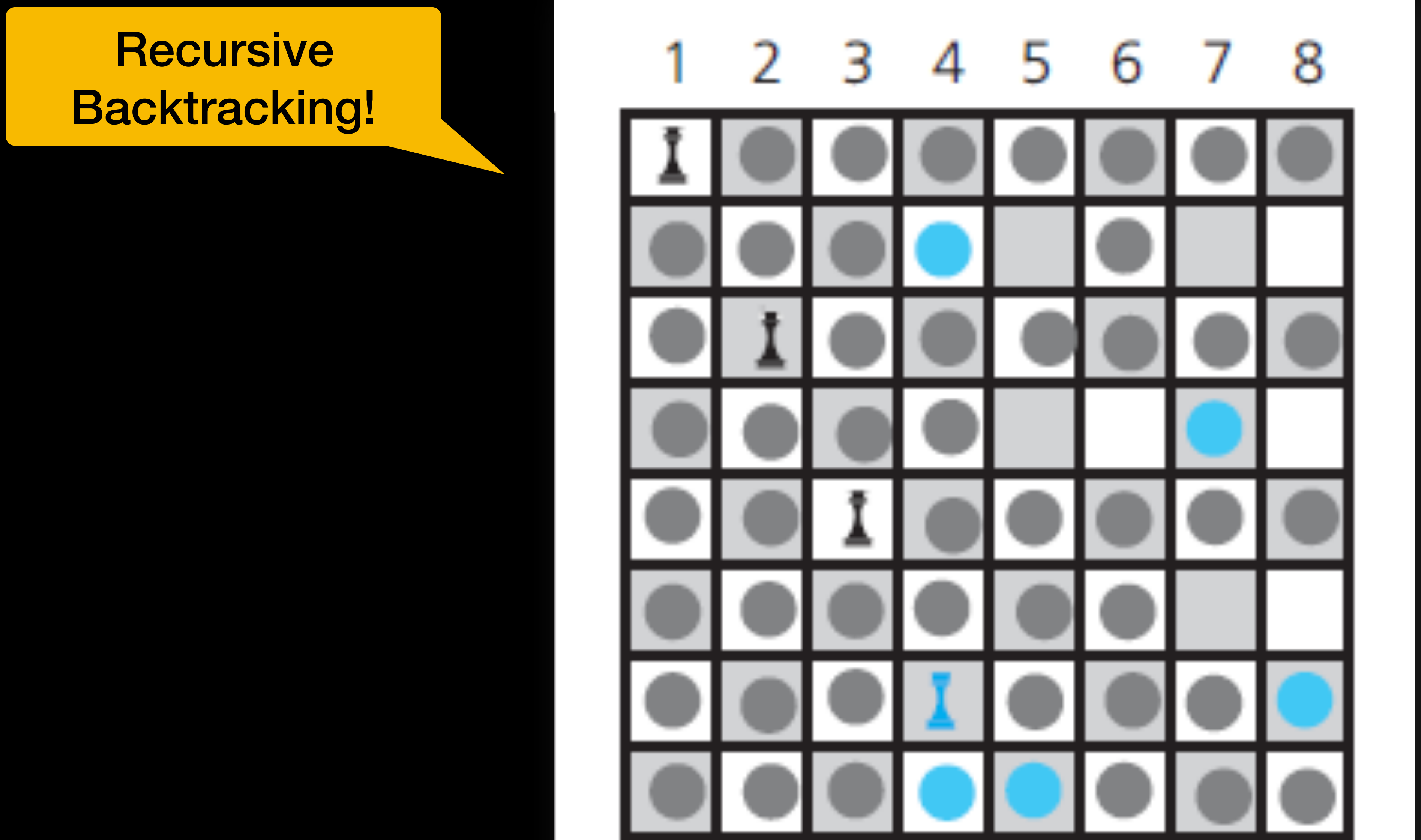
The Eight Queens Problem

Recursive Backtracking!



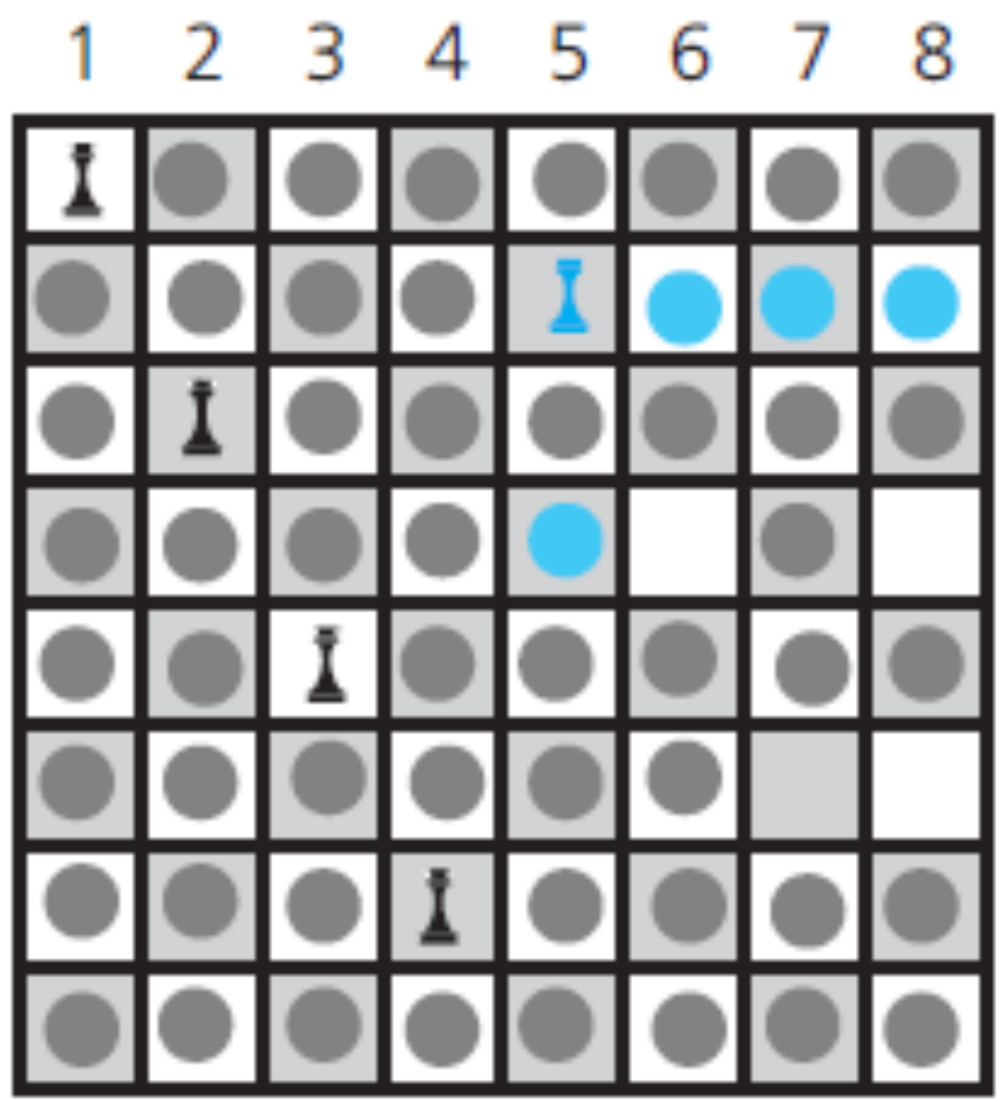
(f) Backtracking to column 5 to try another square for the queen

The Eight Queens Problem



(g) Backtracking to column 4 to try another square for the queen

The Eight Queens Problem



(h) Considering column 5 again

The Eight Queens Problem

```
bool placeQueens(board, column)
{
    if(column > BOARD_SIZE)
        return true; //Problem is solved!
    else{
        while(there are safe squares in this column)
        {
            place queen in next safe  square;
            if(placeQueen(board, column+1)) //recursively look forward
                return true; //queen safely placed
        }
        return false; //recursive backtracking
}
```

Find Permutations

A	B	C	D

A	B	C	D
A	B	D	C
A	C	B	D
A	C	D	B
A	D	B	C
A	D	C	B

B	A	C	D
B	A	D	C
B	C	A	D
B	C	D	A
B	D	A	C
B	D	C	A

C	A	B	D
C	A	D	B
C	B	A	D
C	B	D	A
C	D	A	B
C	D	B	A

D	A	B	C
D	A	C	B
D	B	A	C
D	B	C	A
D	C	A	B
D	C	B	A

Find Permutations

A	B	C	D

A	B	C	D	B	A	C	D	C	A	B	D	D	A	B	C
A	B	D	C	B	A	D	C	C	A	D	B	D	A	C	B
A	C	B	D	B	C	A	D	C	B	A	D	D	B	A	C
A	C	D	B	B	C	D	A	C	B	D	A	D	B	C	A
A	D	B	C	B	D	A	C	C	D	A	B	D	C	A	B
A	D	C	B	B	D	C	A	C	D	B	A	D	C	B	A

Find Permutations

A	B	C	D

A	B	C	D
A	B	D	C
A	C	B	D
A	C	D	B
A	D	B	C
A	D	C	B

B	A	C	D
B	A	D	C
B	C	A	D
B	C	D	A
B	D	A	C
B	D	C	A

C	A	B	D
C	A	D	B
C	B	A	D
C	B	D	A
C	D	A	B
C	D	B	A

D	A	B	C
D	A	C	B
D	B	A	C
D	B	C	A
D	C	A	B
D	C	B	A

Find Permutations

A	B	C	D

A	B	C	D
A	B	D	C
A	C	B	D
A	C	D	B
A	D	B	C
A	D	C	B

B	A	C	D
B	A	D	C
B	C	A	D
B	C	D	A
B	D	A	C
B	D	C	A

C	A	B	D
C	A	D	B
C	B	A	D
C	B	D	A
C	D	A	B
C	D	B	A

D	A	B	C
D	A	C	B
D	B	A	C
D	B	C	A
D	C	A	B
D	C	B	A

Find Permutations

A	B	C	D

A	B	C	D
A	B	D	C
A	C	B	D
A	C	D	B
A	D	B	C
A	D	C	B

B	A	C	D
B	A	D	C
B	C	A	D
B	C	D	A
B	D	A	C
B	D	C	A

C	A	B	D
C	A	D	B
C	B	A	D
C	B	D	A
C	D	A	B
C	D	B	A

D	A	B	C
D	A	C	B
D	B	A	C
D	B	C	A
D	C	A	B
D	C	B	A

Find Permutations

A	B	C	D

A	B	C	D
A	B	D	C
A	C	B	D
A	C	D	B
A	D	B	C
A	D	C	B

B	A	C	D
B	A	D	C
B	C	A	D
B	C	D	A
B	D	A	C
B	D	C	A

C	A	B	D
C	A	D	B
C	B	A	D
C	B	D	A
C	D	A	B
C	D	B	A

D	A	B	C
D	A	C	B
D	B	A	C
D	B	C	A
D	C	A	B
D	C	B	A

Find Permutations

A	B	C	D

A	B	C	D
A	B	D	C
A	C	B	D
A	C	D	B
A	D	B	C
A	D	C	B

B	A	C	D
B	A	D	C
B	C	A	D
B	C	D	A
B	D	A	C
B	D	C	A

C	A	B	D
C	A	D	B
C	B	A	D
C	B	D	A
C	D	A	B
C	D	B	A

D	A	B	C
D	A	C	B
D	B	A	C
D	B	C	A
D	C	A	B
D	C	B	A

Find Permutations

A	B	C	D

A	B	C	D
A	B	D	C
A	C	B	D
A	C	D	B
A	D	B	C
A	D	C	B

B	A	C	D
B	A	D	C
B	C	A	D
B	C	D	A
B	D	A	C
B	D	C	A

C	A	B	D
C	A	D	B
C	B	A	D
C	B	D	A
C	D	A	B
C	D	B	A

D	A	B	C
D	A	C	B
D	B	A	C
D	B	C	A
D	C	A	B
D	C	B	A

Find Permutations

A	B	C	D

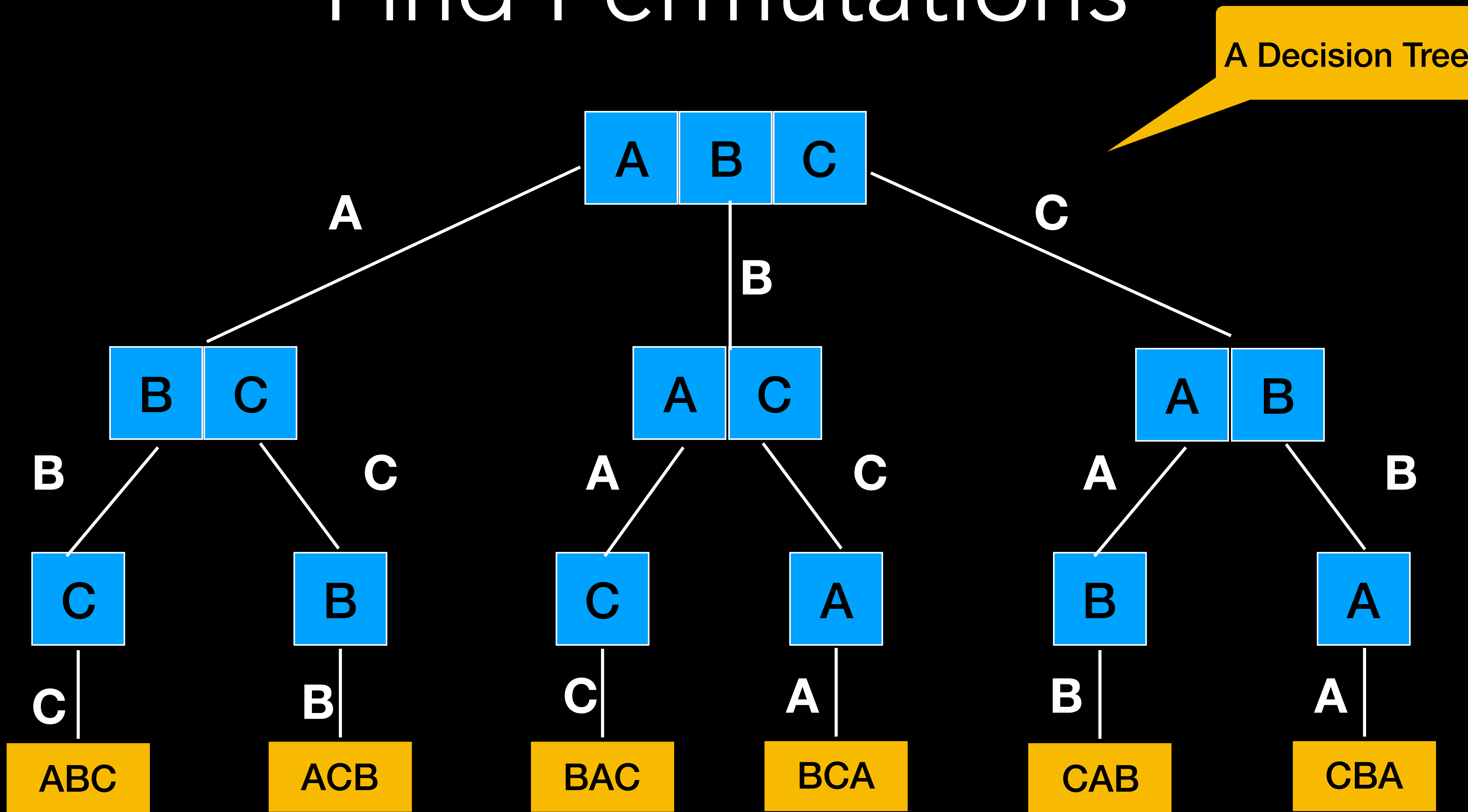
A	B	C	D
A	B	D	C
A	C	B	D
A	C	D	B
A	D	B	C
A	D	C	B

B	A	C	D
B	A	D	C
B	C	A	D
B	C	D	A
B	D	A	C
B	D	C	A

C	A	B	D
C	A	D	B
C	B	A	D
C	B	D	A
C	D	A	B
C	D	B	A

D	A	B	C
D	A	C	B
D	B	A	C
D	B	C	A
D	C	A	B
D	C	B	A

Find Permutations



```
vector<string> generatePermutations(string word)
{
    vector<string> result;
    if (word.length() == 0)
    {
        // The empty string has only itself as a permutation
        result.push_back(word);
        return result;
    }
    for (int i = 0; i < word.length(); i++)
    {
        // The word without the ith letter
        string shorter_word = word.substr(0, i) + word.substr(i + 1);
        vector<string> shorter_permutations =
                                       generatePermutations(shorter_word);

        // Add the ith letter to the front of all permutations
        // of the shorter word
        for (int j = 0; j < shorter_permutations.size(); j++)
        {
            string longer_word = word[i] + shorter_permutations[j];
            result.push_back(longer_word);
        }
    }
    return result;
}
```

Exam Drill:
Analyze the worst-case time complexity of this algorithm
O(?)

```
vector<string> generatePermutations(string word)
{
    vector<string> result;
    if (word.length() == 0)
    {
        // The empty string has only itself as a permutation
        result.push_back(word);
        return result;
    }
    for (int i = 0; i < word.length(); i++)
    {
        // The word without the ith letter
        string shorter_word = word.substr(0, i) + word.substr(i + 1);
        vector<string> shorter_permutations =
                                    generatePermutations(shorter_word);

        // Add the ith letter to the front of all permutations
        // of the shorter word
        for (int j = 0; j < shorter_permutations.size(); j++)
        {
            string longer_word = word[i] + shorter_permutations[j];
            result.push_back(longer_word);
        }
    }
    return result;
}
```

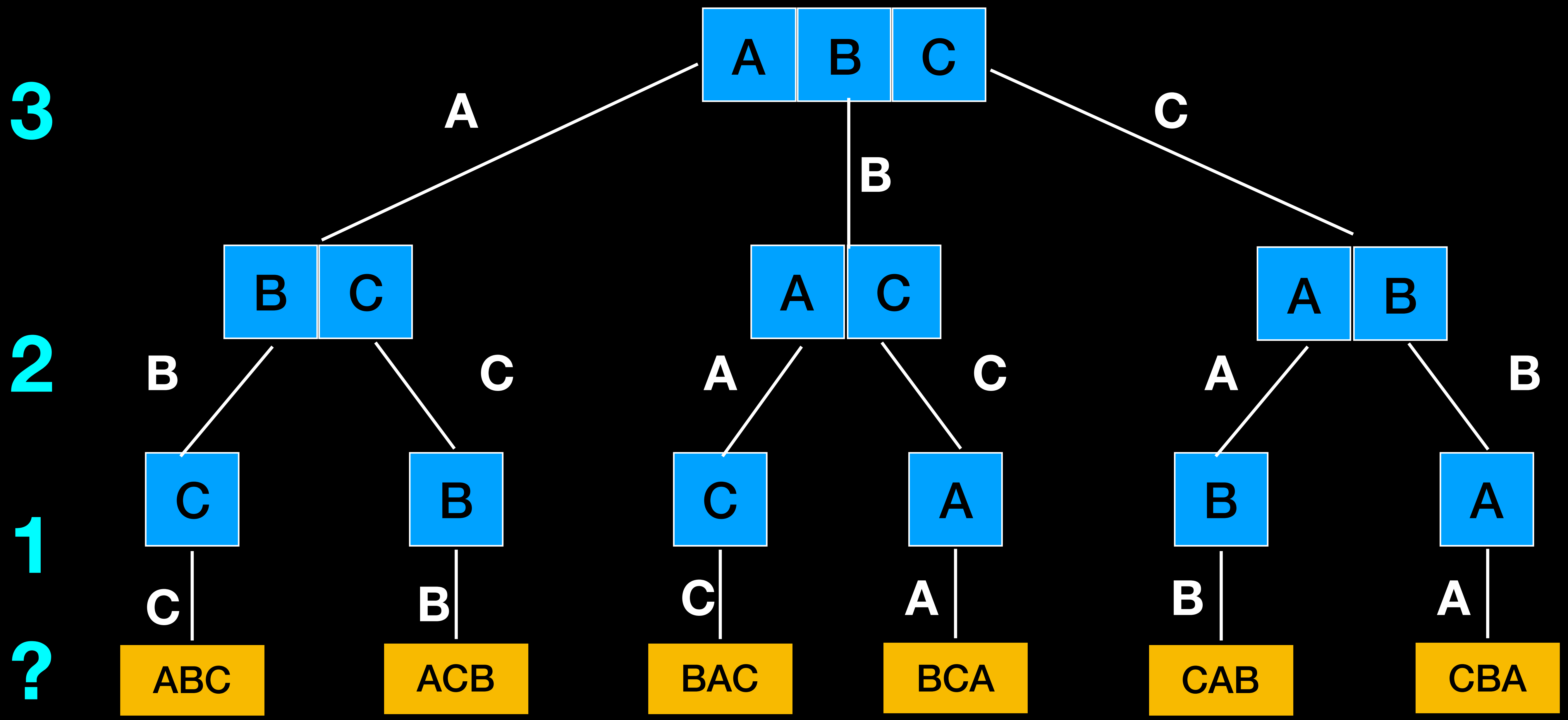
```
vector<string> generatePermutations(string word)
{
    vector<string> result;
k   if (word.length() == 0)
    {
        // The empty string has only itself as a permutation
        result.push_back(word);
        return result;
    }
n   for (int i = 0; i < word.length(); i++)
    {
        // The word without the ith letter
        string shorter_word = word.substr(0, i) + word.substr(i + 1);
        vector<string> shorter_permutations =
                                    generatePermutations(shorter_word);

        // Add the ith letter to the front of all permutations
        // of the shorter word
        for (int j = 0; j < shorter_permutations.size(); j++)
        {
            string longer_word = word[i] + shorter_permutations[j];
            result.push_back(longer_word);
        }
    }
    return result;
}
```

```
vector<string> generatePermutations(string word)
{
    vector<string> result;
k   if (word.length() == 0)
    {
        // The empty string has only itself as a permutation
        result.push_back(word);
        return result;
    }
n   for (int i = 0; i < word.length(); i++)
    {
        // The word without the ith letter
      k string shorter_word = word.substr(0, i) + word.substr(i + 1);
        vector<string> shorter_permutations =
                                  generatePermutations(shorter_word);

        // Add the ith letter to the front of all permutations
        // of the shorter word
      n for (int j = 0; j < shorter_permutations.size(); j++)
        {
          k string longer_word = word[i] + shorter_permutations[j];
            result.push_back(longer_word);
        }
    }
    return result;
}
```

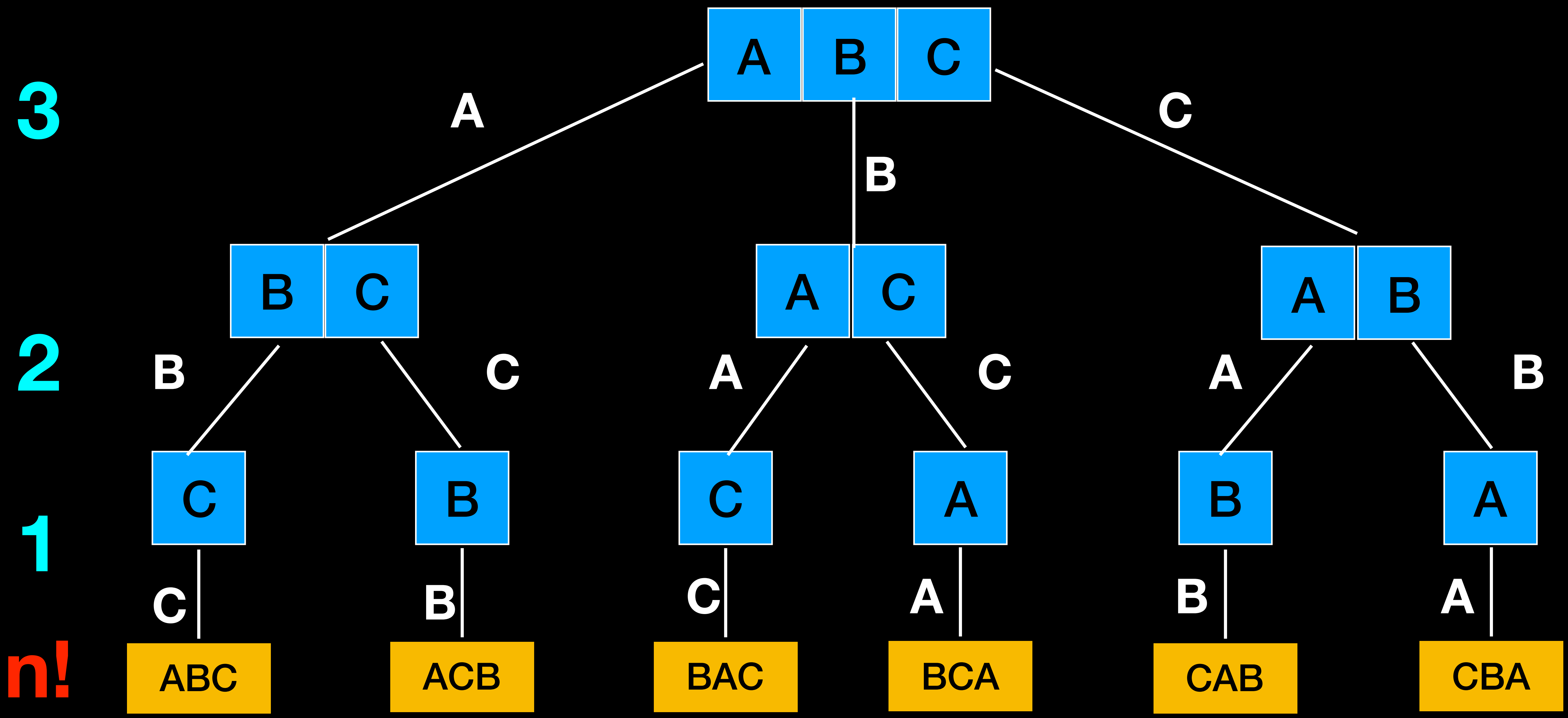
```
vector<string> generatePermutations(string word)
{
    vector<string> result;
    if (word.length() == 0)
    {
        // The empty string has only itself as a permutation
        result.push_back(word);
        return result;
    }
    for (int i = 0; i < word.length(); i++)
    {
        // The word without the ith letter
        string shorter_word = word.substr(0, i) + word.substr(i + 1);
        vector<string> shorter_permutations =
                                generatePermutations(shorter_word);

        // Add the ith letter to the front of all permutations
        // of the shorter word
        for (int j = 0; j < shorter_permutations.size(); j++)
        {
            string longer_word = word[i] + shorter_permutations[j];
            result.push_back(longer_word);
        }
    }
    return result;
}
```

k n k ? n k

Find Permutations



Find Permutations



```
vector<string> generatePermutations(string word)
{
      vector<string> result;
k     if (word.length() == 0)
      {
          // The empty string has only itself as a permutation
          result.push_back(word);
          return result;
      }
n     for (int i = 0; i < word.length(); i++)
      {
          // The word without the ith letter
   k      string shorter_word = word.substr(0, i) + word.substr(i + 1);
          vector<string> shorter_permutations =
                            n!    generatePermutations(shorter_word);

          // Add the ith letter to the front of all permutations
          // of the shorter word
   n      for (int j = 0; j < shorter_permutations.size(); j++)
          {
     k        string longer_word = word[i] + shorter_permutations[j];
              result.push_back(longer_word);
          }
      }
      return result;
}
```

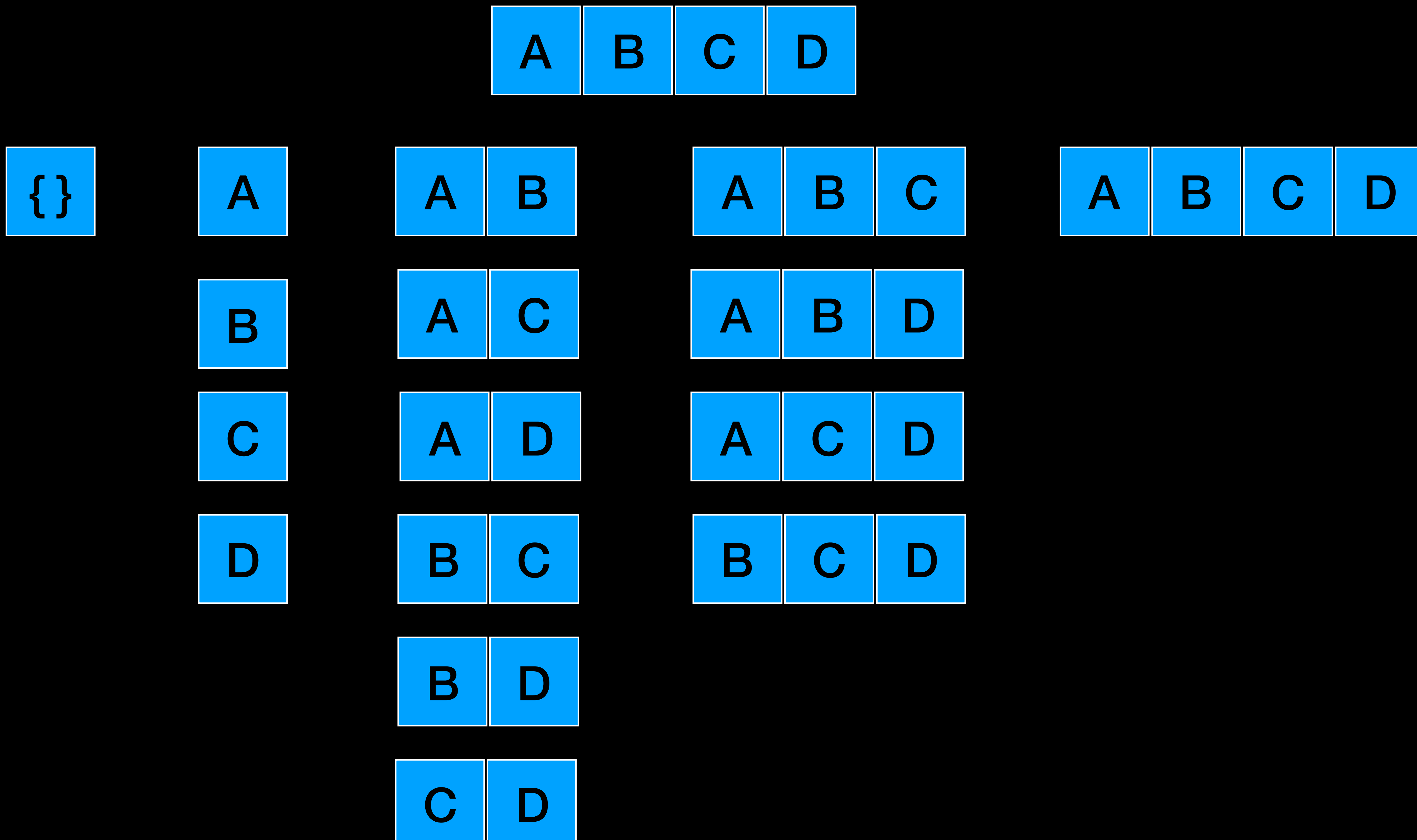
$$T(n) = O(\, n!\, n^2 \,)$$

Recursive Decision Tree

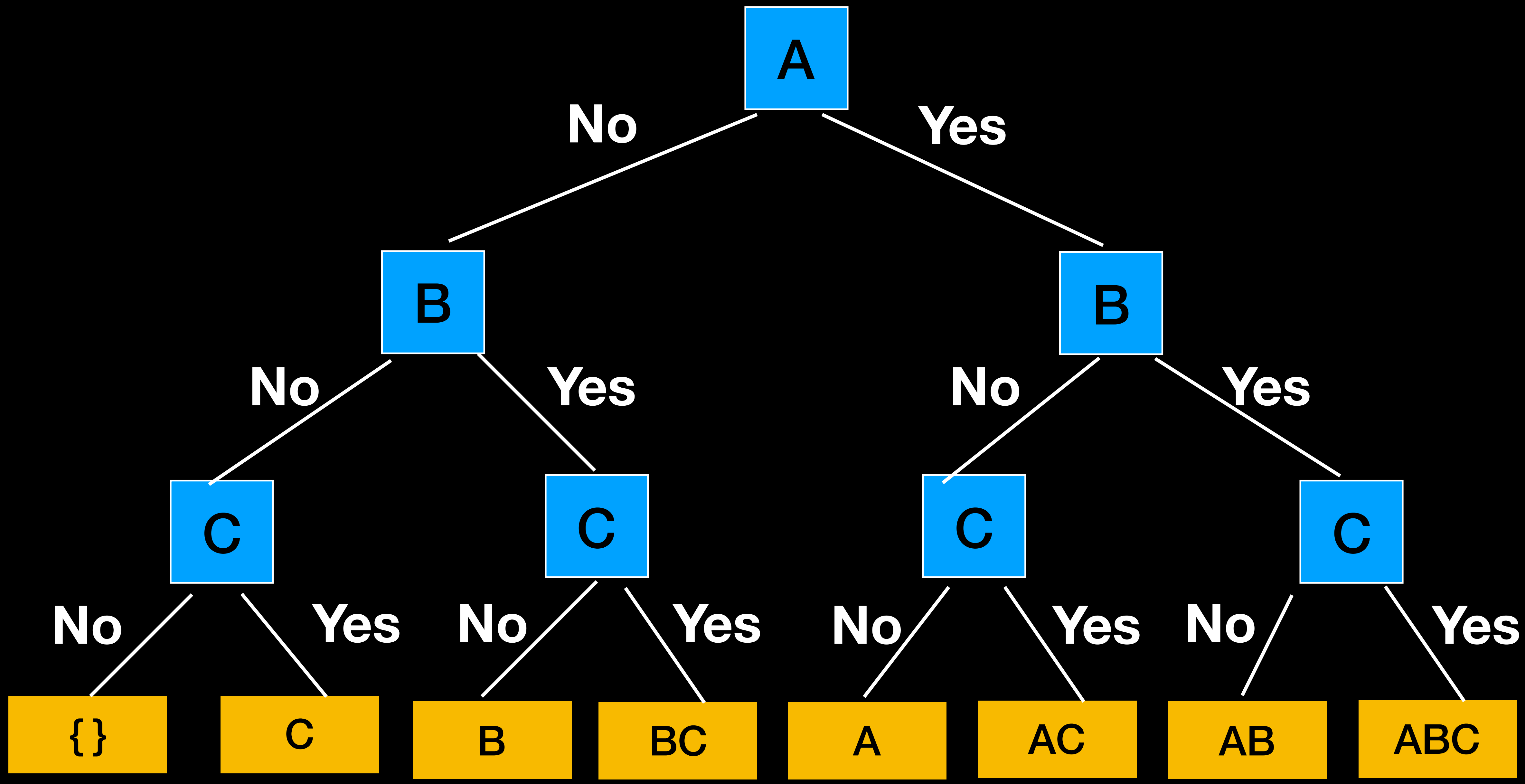
```
void exploreFrom(current_state, decisions_made) {
    if (all decisions have been made) { //base case
        output the result of the decisions we've made;
    } else {
        for (each decision we can make) {
            exploreFrom(result of making that decision,
                        decisions_made + this_decision);
        }
    }
}
```

Generally, if you can express a problem solution with a decision tree you can translate it into a recursive algorithm

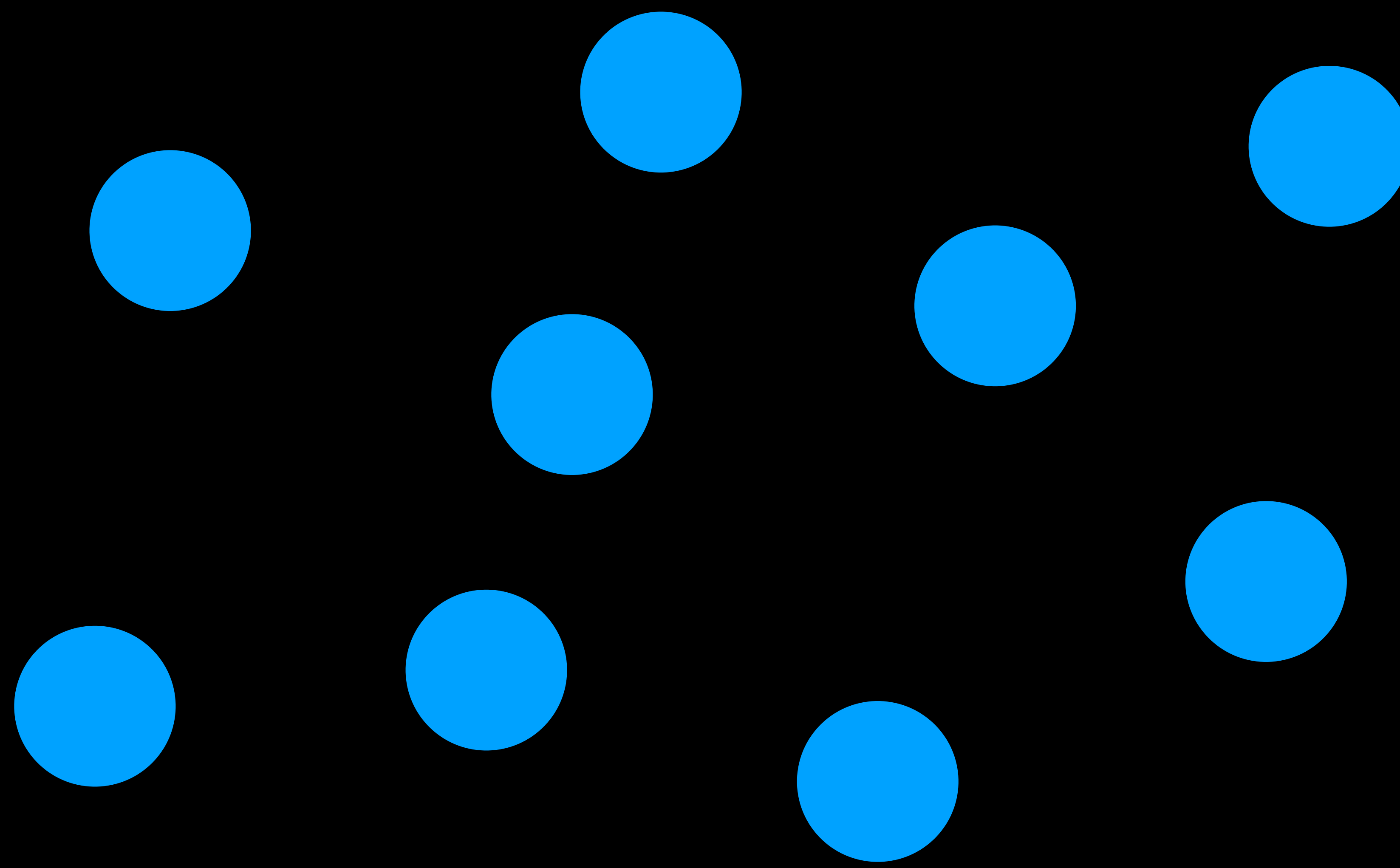
Find Combinations



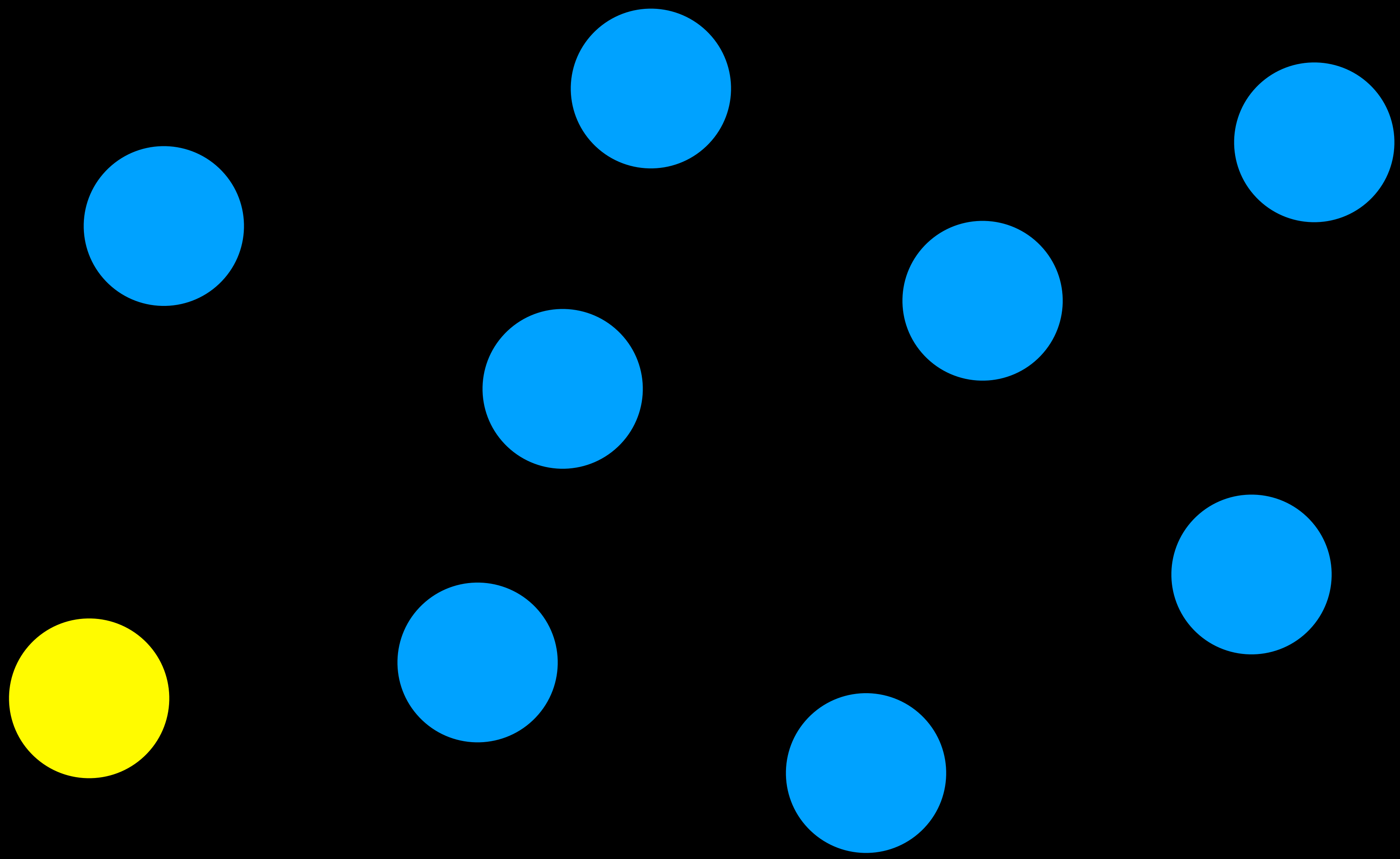
Find All Combinations



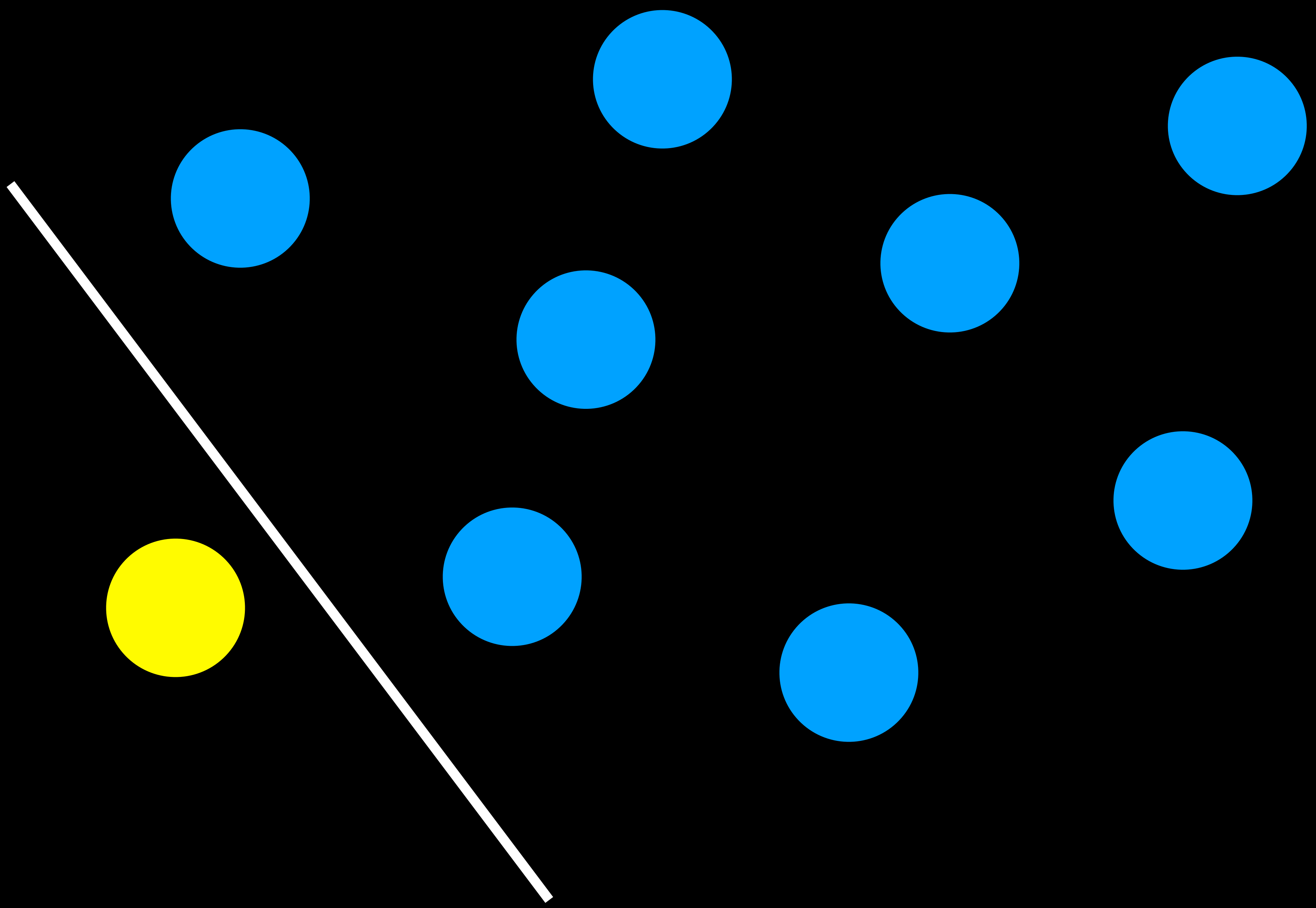
Find Combinations (n choose k)



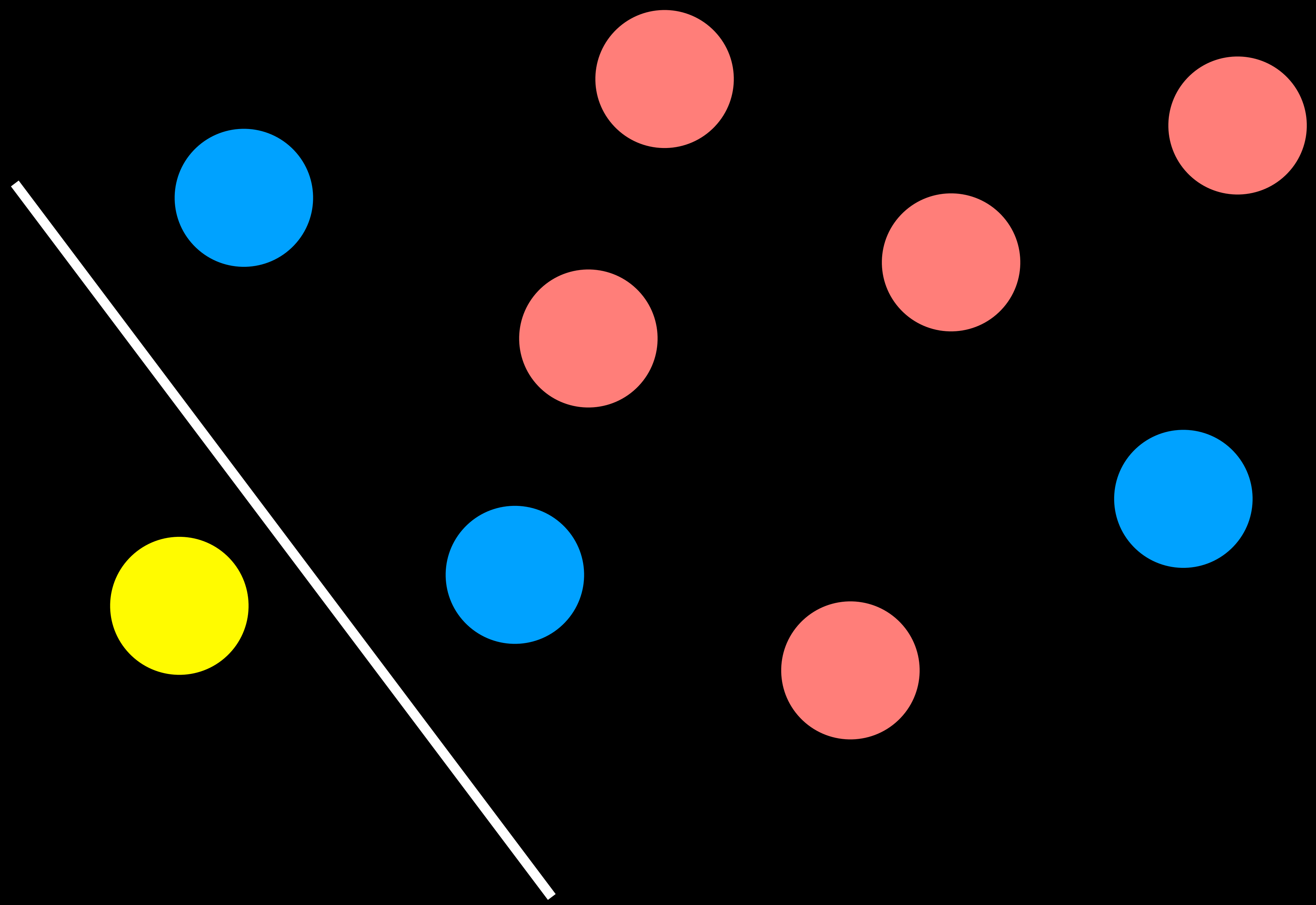
Find Combinations (n choose k)



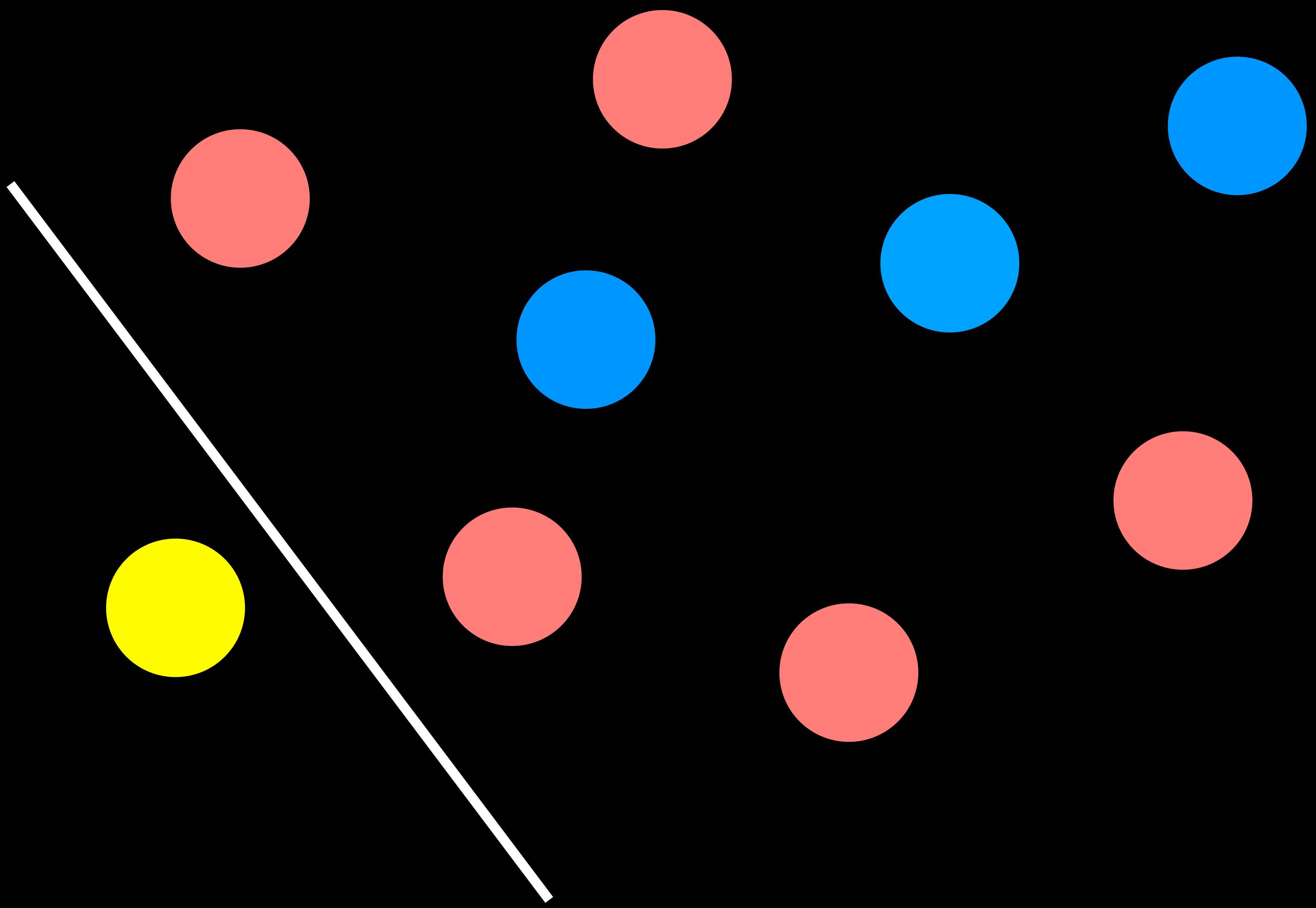
Find Combinations (n choose k)



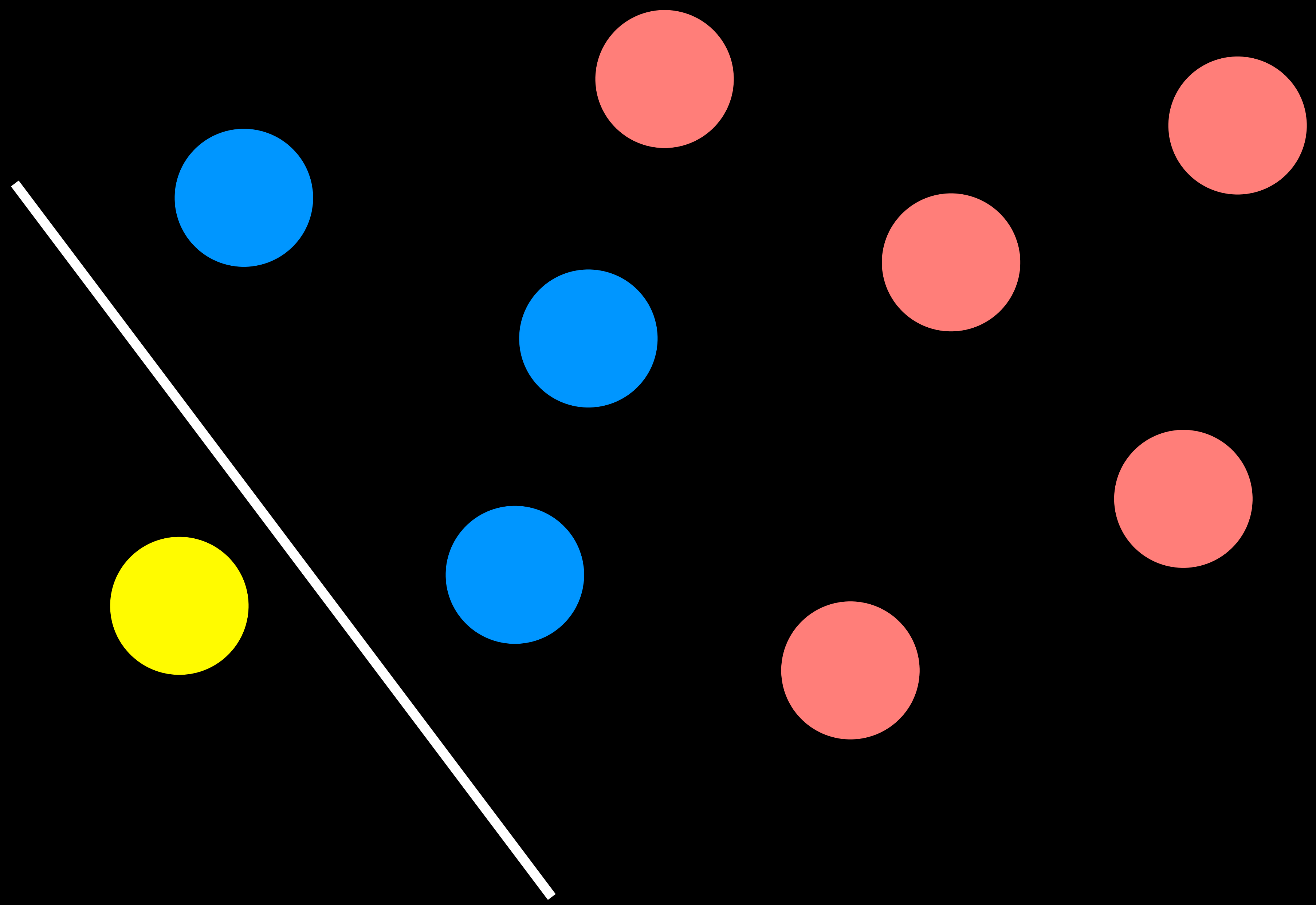
Find Combinations (n choose k)



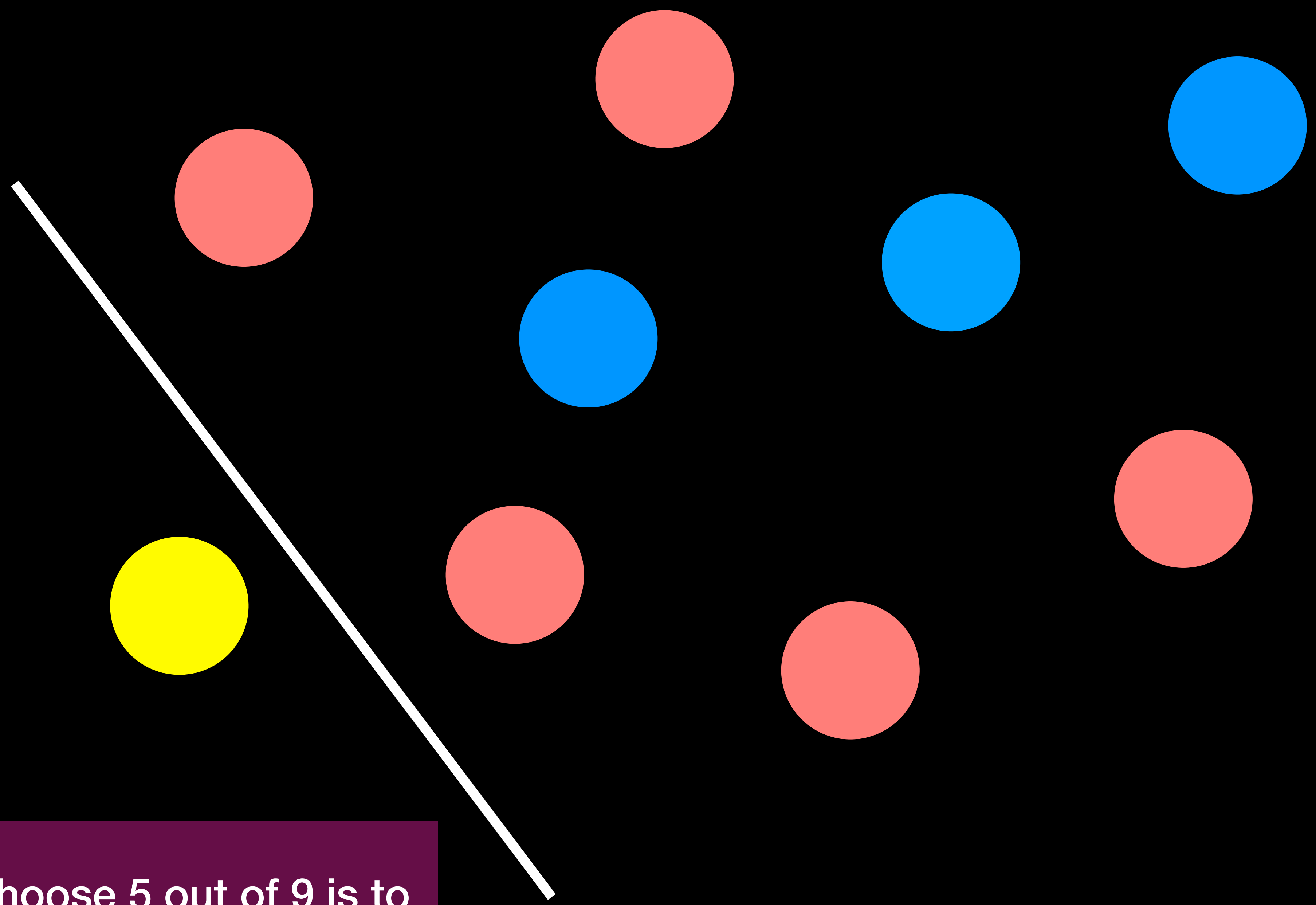
Find Combinations (n choose k)



Find Combinations (n choose k)

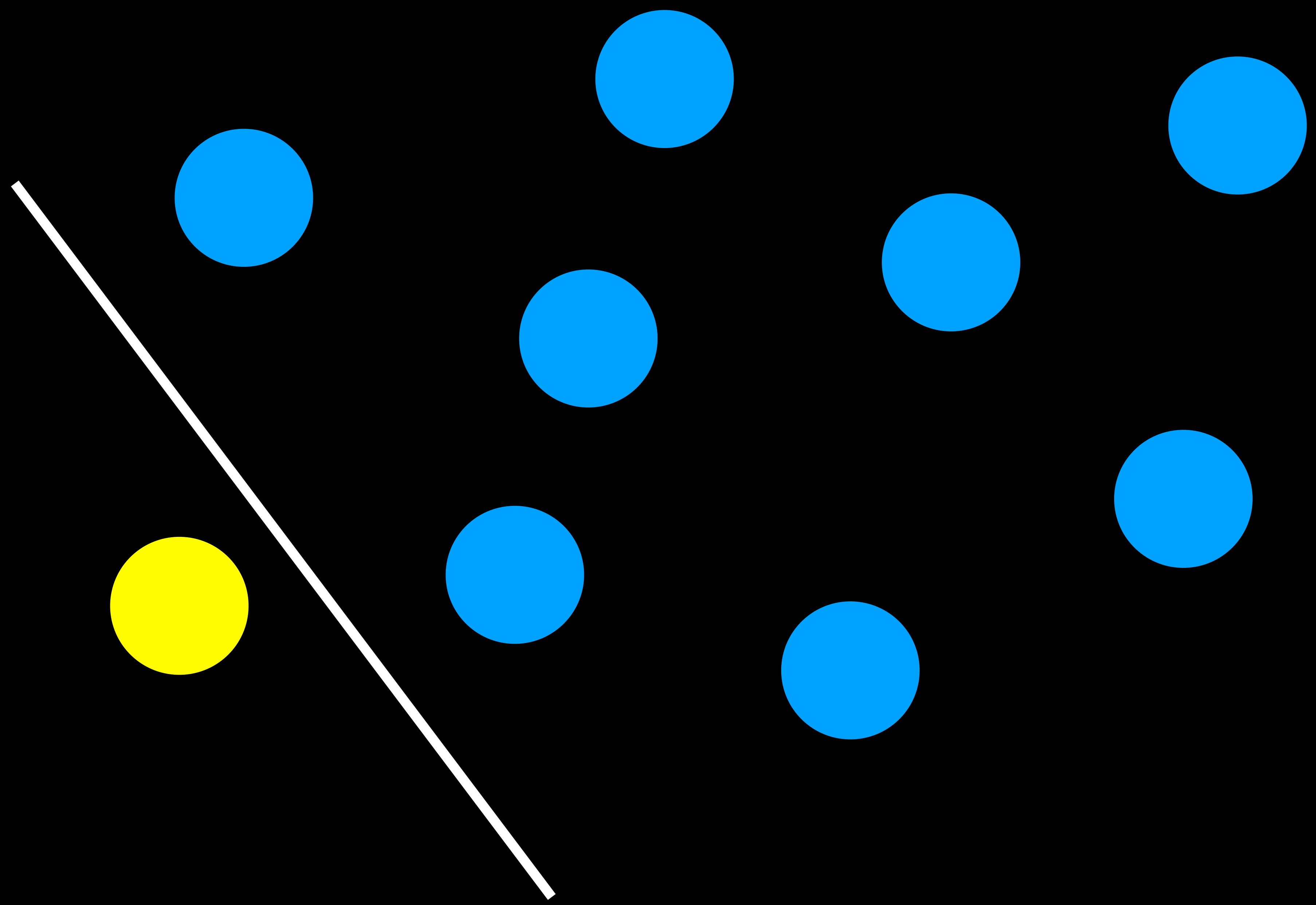


Find Combinations (n choose k)

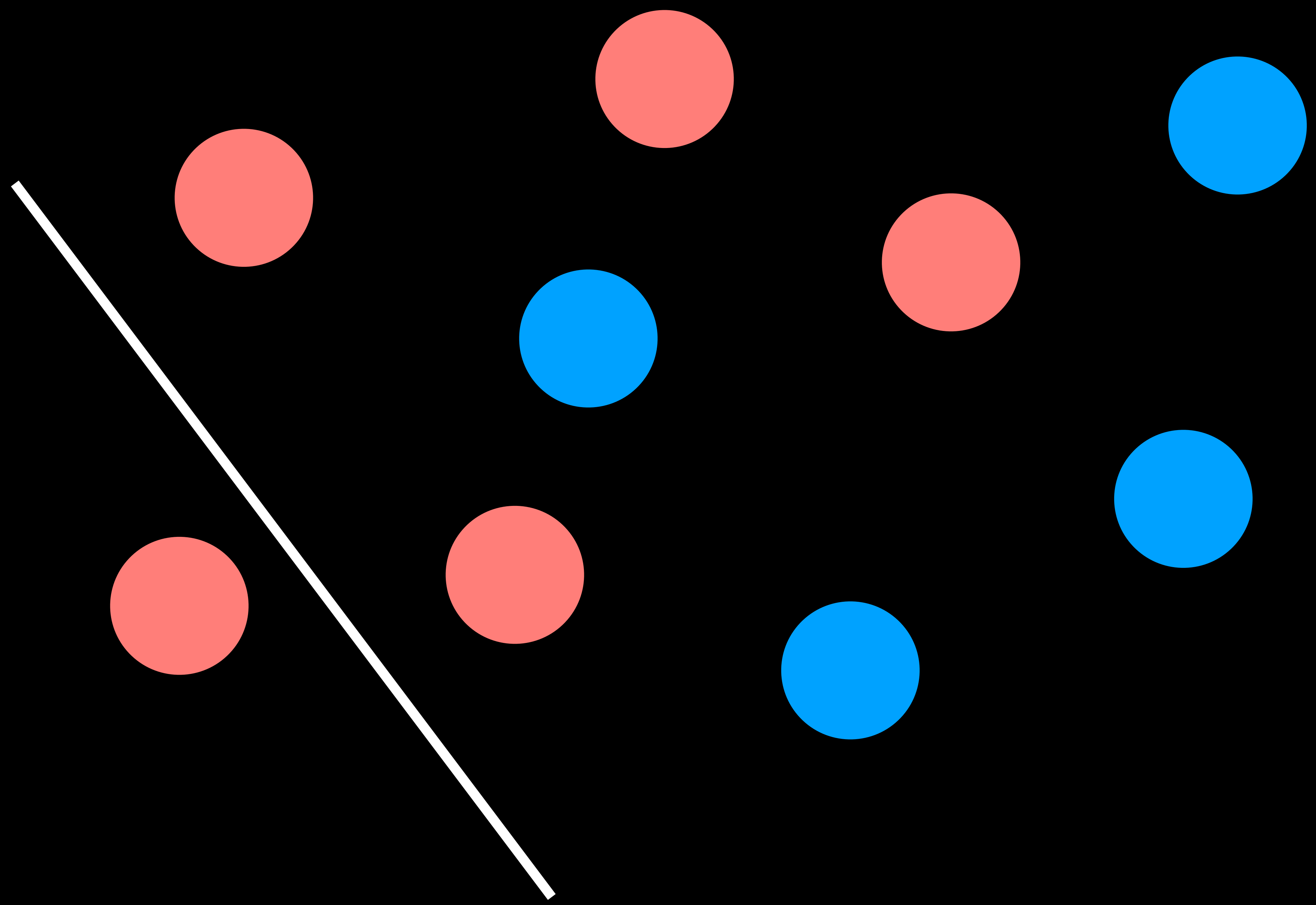


One way to choose 5 out of 9 is to
Exclude 1 and choose 5 out of 8

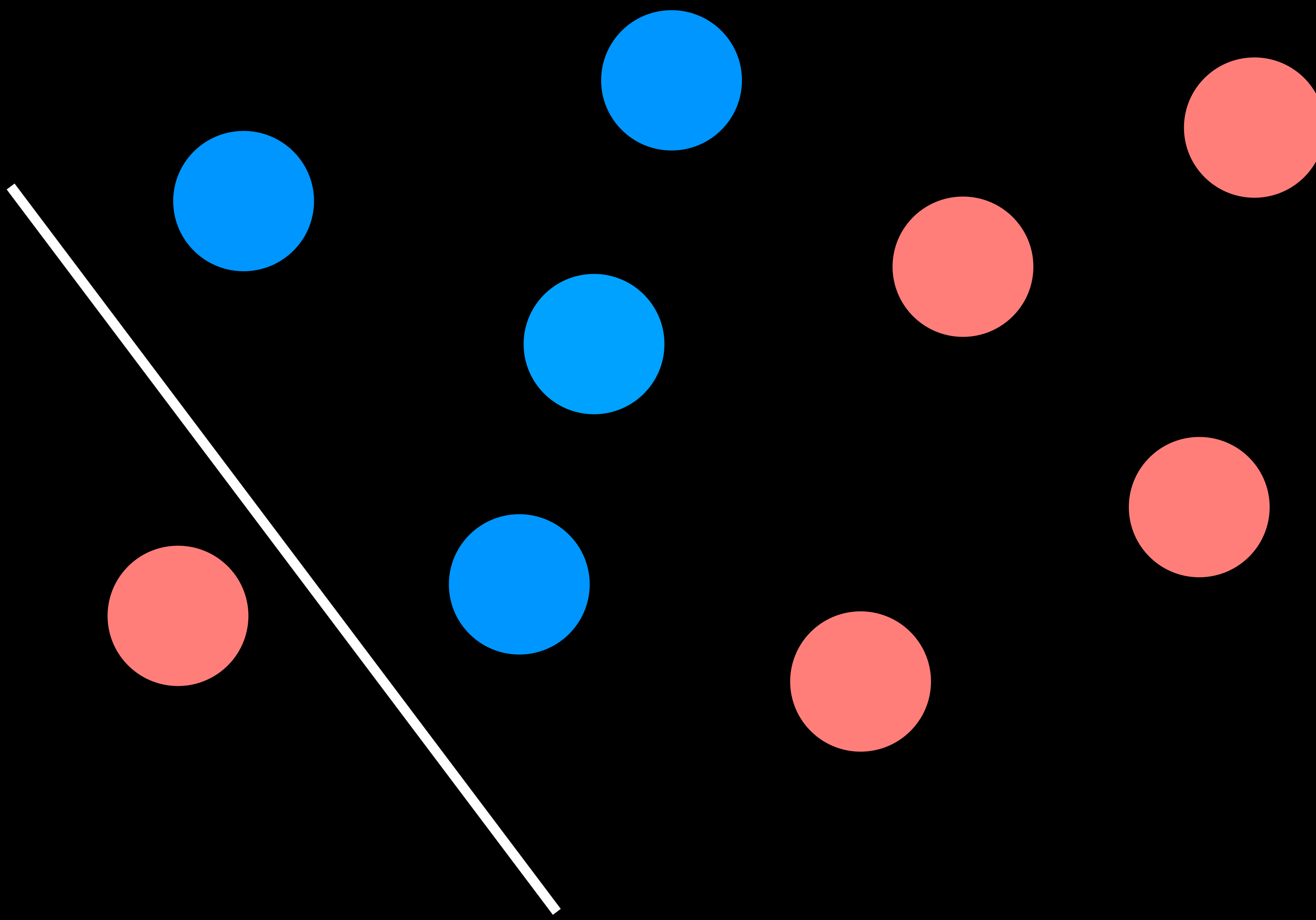
Find Combinations (n choose k)



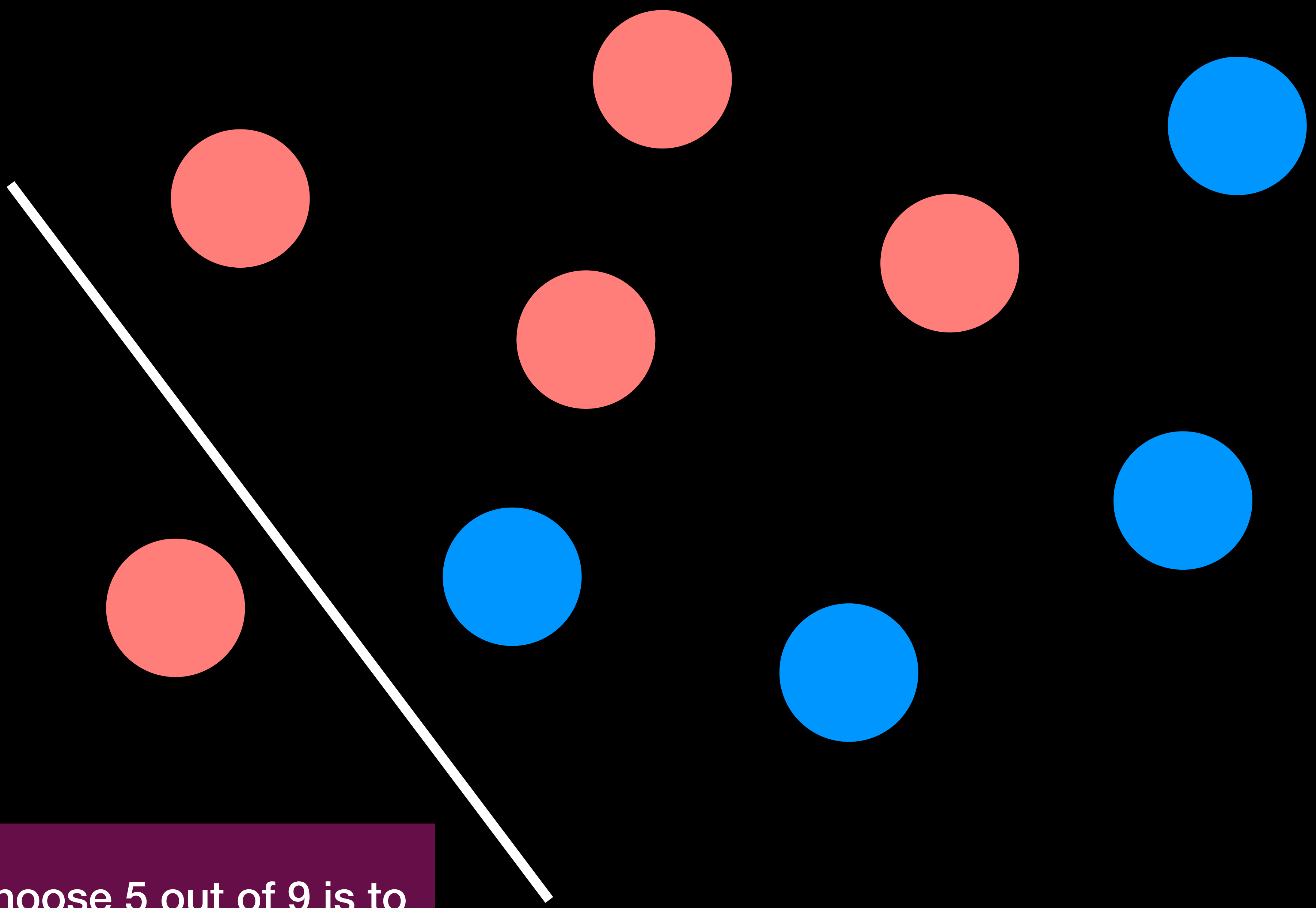
Find Combinations (n choose k)



Find Combinations (n choose k)



Find Combinations (n choose k)



One way to choose 5 out of 9 is to
Include 1 and choose 4 out of 8

Find Combinations

```
chooseK(sequence,  k)
{
    if (k == 0) or (sequence.length() == k)
        return sequence; //basecase

    combinations_k = chooseK(sequence_without_first, k);
    combinations_k-1 = chooseK(sequence_without_first, k-1);
    append first element to combinations_k-1;
    return combinations_k + combinations_k-1;

}
```

Find Combinations

Analysis

```
chooseK(sequence,  k)
{
    if (k == 0) or (sequence.length() == k)
        return sequence; //basecase

    combinations_k = chooseK(sequence_without_first, k);
    combinations_k-1 = chooseK(sequence_without_first, k-1);
    append first element to combinations_k-1;
    return combinations_k + combinations_k-1;

}
```

Find Combinations

```
chooseK(sequence,  k)
{
    if (k == 0) or (sequence.length() == k)
        return sequence; //basecase

    combinations_k = chooseK(sequence_without_first, k);
    combinations_k-1 = chooseK(sequence_without_first, k-1);
    append first element to combinations_k-1;
    return combinations_k + combinations_k-1;

}
```

T(n-1, k)

T(n-1, k-1)

$$T(n, k) = T(n-1, k) + T(n-1, k-1) + c$$

$$T(n, k) = \boxed{T(n-1, k)} + \boxed{T(n-1, k-1)} + c$$

$$T(n, k) = \boxed{T(n-1, k)} + \boxed{T(n-1, k-1)} + c$$

$$T(n, k) = \boxed{T(n-2, k) + T(n-2, k - 1)}$$

$$T(n, k) = \boxed{T(n-1, k)} + \boxed{T(n-1, k-1)} + c$$

$$T(n, k) = \boxed{T(n-2, k) + T(n-2, k - 1)} + \boxed{T(n-2, k-1) + T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-1, k)} + \boxed{T(n-1, k-1)} + c$$

$$T(n, k) = \boxed{T(n-2, k) + T(n-2, k - 1)} + \boxed{T(n-2, k-1) + T(n-2, k-2)} + 2c$$

$$T(n, k) = T(n-2, k) + 2T(n-2, k - 1) + T(n-2, k-2) + 2c$$

$$T(n, k) = \boxed{T(n-1, k)} + \boxed{T(n-1, k-1)} + c$$

$$T(n, k) = \boxed{T(n-2, k) + T(n-2, k - 1)} + \boxed{T(n-2, k-1) + T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-2, k)} + \boxed{2T(n-2, k - 1)} + \boxed{T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-1, k)} + \boxed{T(n-1, k-1)} + c$$

$$T(n, k) = \boxed{T(n-2, k) + T(n-2, k-1)} + \boxed{T(n-2, k-1) + T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-2, k)} + \boxed{2T(n-2, k-1)} + \boxed{T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-3, k) + T(n-3, k-1)}$$

$$T(n, k) = \boxed{T(n-1, k)} + \boxed{T(n-1, k-1)} + c$$

$$T(n, k) = \boxed{T(n-2, k) + T(n-2, k-1)} + \boxed{T(n-2, k-1) + T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-2, k)} + \boxed{2T(n-2, k-1)} + \boxed{T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-3, k) + T(n-3, k-1)} + \boxed{2T(n-3, k-1) + 2T(n-3, k-2)} +$$

$$T(n, k) = \boxed{T(n-1, k)} + \boxed{T(n-1, k-1)} + c$$

$$T(n, k) = \boxed{T(n-2, k) + T(n-2, k - 1)} + \boxed{T(n-2, k-1) + T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-2, k)} + \boxed{2T(n-2, k - 1)} + \boxed{T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-3, k) + T(n-3, k-1)} + \boxed{2T(n-3, k - 1) + 2T(n-3, k - 2)} + \boxed{T(n-3, k-2) + T(n-3, k-3)} + 3c$$

$$T(n, k) = \boxed{T(n-1, k)} + \boxed{T(n-1, k-1)} + c$$

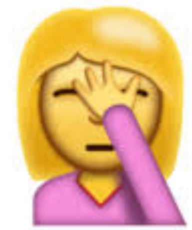
$$T(n, k) = \boxed{T(n-2, k) + T(n-2, k - 1)} + \boxed{T(n-2, k-1) + T(n-2, k-2)} + 2c$$

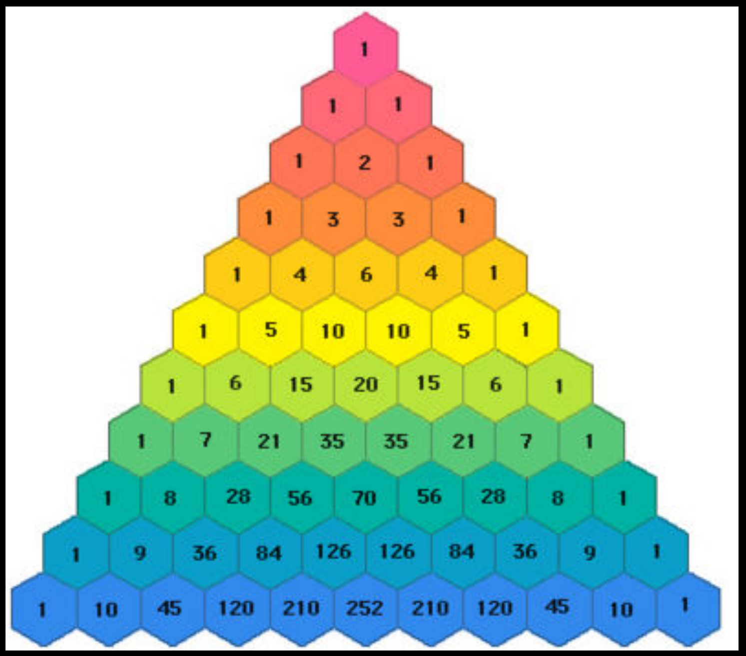
$$T(n, k) = \boxed{T(n-2, k)} + \boxed{2T(n-2, k - 1)} + \boxed{T(n-2, k-2)} + 2c$$

$$\begin{aligned} T(n, k) = {} & \boxed{T(n-3, k) + T(n-3, k-1)} + \boxed{2T(n-3, k - 1) + 2T(n-3, k - 2)} + \\ & \boxed{T(n-3, k-2) + T(n-3, k-3)} + 3c \end{aligned}$$

$$T(n, k) = T(n-3, k) + 3T(n-3, k - 1) + 3T(n-3, k - 2) + T(n-3, k-3) + 3c$$

Of Course!!!
It's nChoosek !!!





$$T(n, k) = \boxed{T(n-1, k)} + \boxed{T(n-1, k-1)} + c$$

$$T(n, k) = \boxed{T(n-2, k) + T(n-2, k - 1)} + \boxed{T(n-2, k-1) + T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-2, k)} + \boxed{2T(n-2, k - 1)} + \boxed{T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-3, k) + T(n-3, k-1)} + \boxed{2T(n-3, k - 1) + 2T(n-3, k - 2)} + \boxed{T(n-3, k-2) + T(n-3, k-3)} + 3c$$

$$T(n, k) = T(n-3, k) + 3T(n-3, k - 1) + 3T(n-3, k - 2) + T(n-3, k-3) + 3c$$

Binomial Coefficients

But we are looking for Big-O!!!

$$T(n, k) = \boxed{T(n-1, k)} + \boxed{T(n-1, k-1)} + c$$

$$T(n, k) = \boxed{T(n-2, k) + T(n-2, k-1)} + \boxed{T(n-2, k-1) + T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-2, k)} + \boxed{2T(n-2, k-1)} + \boxed{T(n-2, k-2)} + 2c$$

$$T(n, k) = \boxed{T(n-3, k) + T(n-3, k-1)} + \boxed{2T(n-3, k-1) + 2T(n-3, k-2)} + \boxed{T(n-3, k-2) + T(n-3, k-3)} + 3c$$

$$T(n, k) = T(n-3, k) + 3T(n-3, k-1) + 3T(n-3, k-2) + T(n-3, k-3) + 3c$$

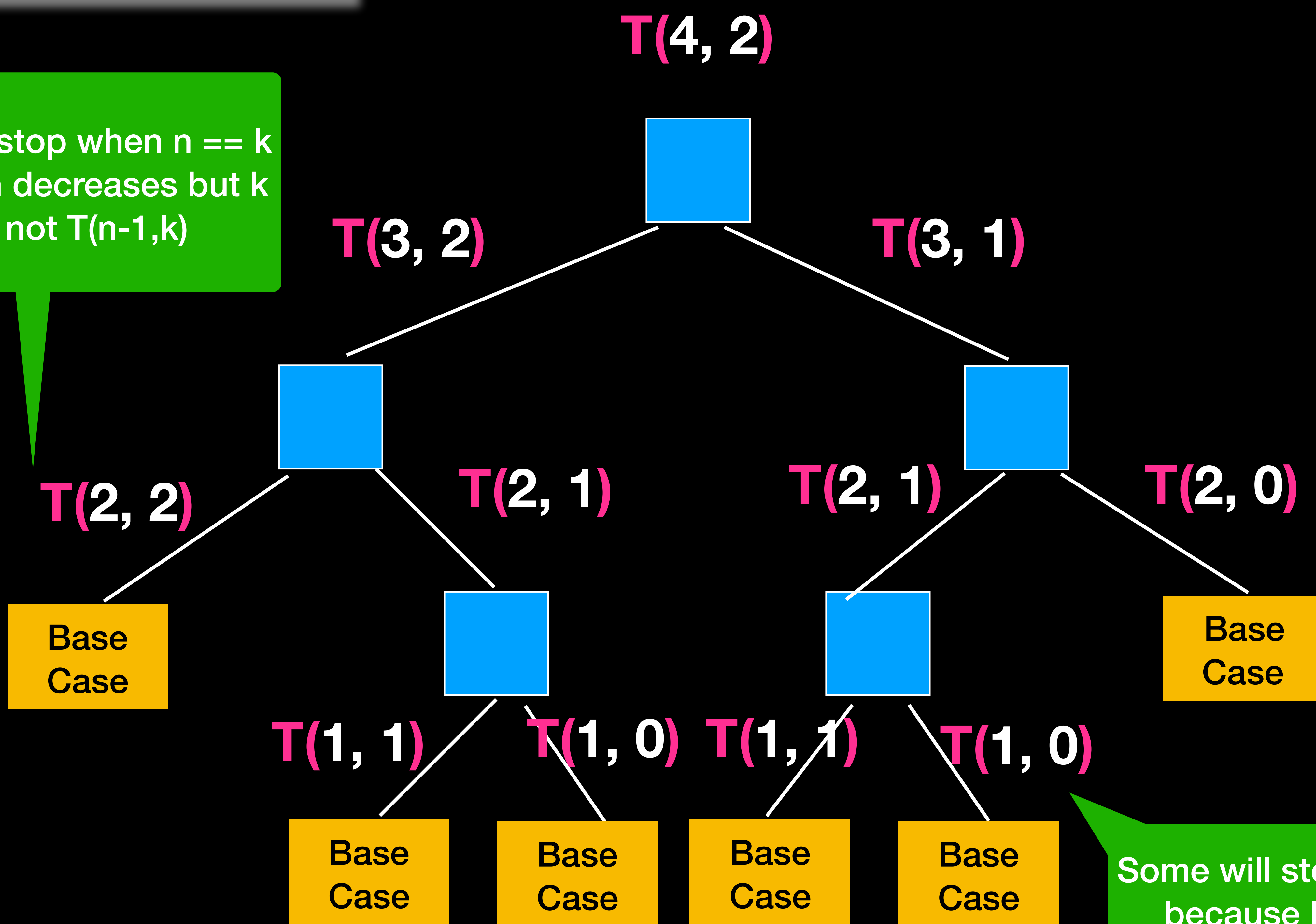
Some will stop when n == 0

All +
We are looking for
dominant term

Some will stop when k == n

We know n > k

Some will stop when n == k because n decreases but k does not T(n-1,k)

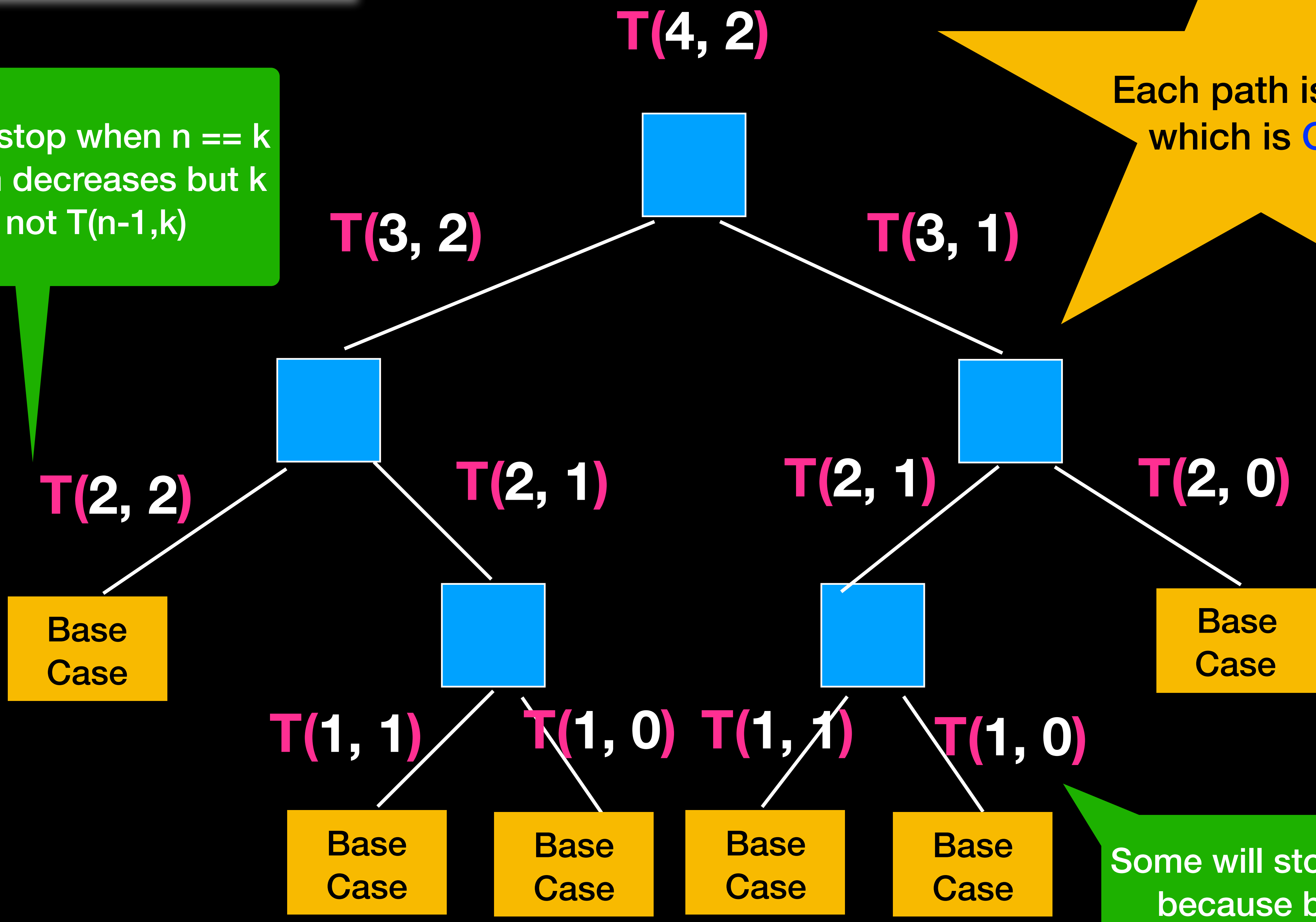


Some will stop when k == 0 because both n and k decrease but k < n T(n-1,k-1)

We know n > k

Some will stop when n == k because n decreases but k does not T(n-1,k)

Each path is O(k) which is O(n)



Some will stop when k == 0 because both n and k decrease but k < n T(n-1,k-1)

Find Combinations

```
chooseK(sequence,  k)
{
    if (k == 0) or (sequence.length() == k)
        return sequence; //basecase

    combinations_k = chooseK(sequence_without_first, k);
    combinations_k-1 = chooseK(sequence_without_first, k-1);
    append first element to combinations_k-1;
    return combinations_k + combinations_k-1;

}
```

T(n-1, k)

T(n-1, k-1)

$$T(n, k) = c_1 O(n) + c_2 O(n) + O(1)$$

Find Combinations

```
chooseK(sequence,  k)
{
    if (k == 0) or (sequence.length() == k)
        return sequence; //basecase

    combinations_k = chooseK(sequence_without_first, k);
    combinations_k-1 = chooseK(sequence_without_first, k-1);
    append first element to combinations_k-1;
    return combinations_k + combinations_k-1;

}
```

$$T(n) = O(\ n\ )$$

Count Combinations

```
int countCombinations(int n, int k)
{
    if ( (k == 0) || (k == n))
        return 1;
    else
        return countCombinations(n-1, k-1) +
                          countCombinations((n-1), k);
}
```