Previously …

Analysis of Algorithms
running times (mathematical models)
big O notation
examples
Today ...

Recursion
  definition
  backtracking
  examples
Recursion

Function that calls \textit{itself}

often used when iterative solution is not simple
Recursion

Function that calls itself

often used when iterative solution is not simple

Structure

base case

solution of a trivial case

recursive call(s)

divide problem into smaller instances!
Base Case

stops the recursion and solves a simple case
Base Case

Base Case

*stops* the recursion and solves a simple case

Infinite recursion!

make sure recursive algorithm reaches base case

stack overflow
int power(int x, int n) {
    if (n == 0) {
        return 1;
    } else {
        return x * power(x, n-1);
    }
}
int power(int x, int n) {
    if (n == 0) {
        return 1;
    } else {
        return x * power(x, n-1);
    }
}
unsigned long int factorial(unsigned long int n) {
    return (n < 2) ? 1 : n * factorial(n-1);
}
unsigned long int factorial(unsigned long int n) {
    return (n < 2) ? 1 : n * factorial(n-1);
}

int gcd(int x, int y) {
    return (y == 0) ? x : gcd(y, x % y);
}
Recursive Graphics

From https://en.wikipedia.org/wiki/H_tree
Recursive Graphics

From https://en.wikipedia.org/wiki/H_tree

From http://www.oftenpaper.net/sierpinski.htm
Recursive Backtracking
Permutations

Consider finding all permutations of an input string

From https://en.wikipedia.org/wiki/Derangement
Consider finding all permutations of an input string $n!$. 

From https://en.wikipedia.org/wiki/Derangement
Permutations

Exhaustively consider all possibilities

“abcd”

“a” + all permutations of “bcd”

“b” + all permutations of “acd”

“c” + all permutations of “abd”

“d” + all permutations of “abc”
Permutations

Exhaustively consider all possibilities

“abcd”

“a” + all permutations of “bcd”
“b” + all permutations of “acd”
“c” + all permutations of “abd”
“d” + all permutations of “abc”

\[ n^* (n-1)! \]
void permutations(char *str, int i, int n) {
    if (i == n) {
        std::cout << str << std::endl;
    } else {
        for (int j = i; j <= n; j++) {
            swap(str+i, str+j);
            permutations(str, i+1, n);
            swap(str+i, str+j);
        }
    }
}
Backtracking

Brute force algorithms
test all possibilities (expensive)
but ... easy to implement
Backtracking

Brute force algorithms
test all possibilities (expensive)
but … easy to implement

Recursive Backtracking

recursive — solve smaller versions of the problem
backtrack when necessary (dead ends)
Backtracking

```cpp
bool solve(configuration) {
    if no more choices {
        return True
    }
    for all available choices {
        try one choice
        recursively solve after making choice
        if ok return True
        else unmake choice (backtrack)
    }
    return False // no more choices
}
```
How many placements?

\[ \binom{n}{k} \]

64 choose 8

From https://en.wikipedia.org/wiki/Eight_queens_puzzle
N-Queens

// call algorithm with col 0
if N queens placed, return True
for all available positions in this column
    try placing a queen safely
    recursively solve for next column
    if success, return True
else, remove queen and try next row
return False // can’t place queen in this col
From https://en.wikipedia.org/wiki/Backtracking
Sudoku

find col, row of first unassigned cell
if all cells assigned, return True
for digits 1 to 9
  try placing a digit safely
  recursively solve next unassigned cell
  if success, return True
else, remove digit and try another
return False // can’t place digit in this cell
Maze

Find a path connecting two cells