### A Possible Pitfall With Recursion

**Fibonacci numbers.** 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

\[
F(n) = \begin{cases} 
0 & \text{if } n = 0 \\
1 & \text{if } n = 1 \\
F(n-1) + F(n-2) & \text{otherwise}
\end{cases}
\]

### Recursion Challenge 1 (difficult but important)

**Q.** Is this an efficient way to compute \(F(50)\)?

```java
public static long F(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return F(n-1) + F(n-2);
}
```

**A.** No, no, no! This code is spectacularly inefficient.

### Recursion Challenge 2 (easy and also important)

**Q.** Is this a more efficient way to compute \(F(50)\)?

```java
public static long F(int n) {
    long[] F = new long[n+1];
    F[0] = 0;
    F[1] = 1;
    for (int i = 2; i <= n; i++)
        F[i] = F[i-1] + F[i-2];
    return F[n];
}
```

**A.** Yes. This code does it with 50 additions.

**Lesson.** Don’t use recursion to engage in exponential waste.

**Context.** This is a special case of an important programming technique known as **dynamic programming**.
Binomial Coefficients

Binomial coefficient. \( \binom{n}{k} \) = number of ways to choose \( k \) of \( n \) elements.

Ex. Number of possible 7-card poker hands = \( \binom{52}{7} \) = 2,598,960.

Ex. Probability of “quads” in Texas hold ’em:
\[
\binom{13}{4} \times \binom{48}{3} = \frac{224,848 \times 133,784,560}{151,756,560} \approx 0.594 : 1
\]

Binomial Coefficients: Sierpinski Triangle

Binomial coefficient. \( \binom{n}{k} \) = number of ways to choose \( k \) of \( n \) elements.

Sierpinski triangle. Color black the odd integers in Pascal’s triangle.

Binomial Coefficients: First Attempt

```java
public class SlowBinomial {
    // natural recursive implementation
    public static long binomial(long n, long k) {
        if (k == 0) return 1;
        if (n == 0) return 0;
        return binomial(n-1, k-1) + binomial(n-1, k);
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int K = Integer.parseInt(args[1]);
        StdOut.println(binomial(N, K));
    }
}
```

Pascal’s identity.
\[
\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}
\]

Pascal’s triangle.
Performance Challenge 3

Q. Is this an efficient way to compute binomial coefficients?
A. No, no, no! [same essential recomputation flaw as naïve Fibonacci]

Timing Experiments

Timing experiments: direct recursive solution.

<table>
<thead>
<tr>
<th>(2n, n)</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(26, 13)</td>
<td>0.46</td>
</tr>
<tr>
<td>(28, 14)</td>
<td>1.27</td>
</tr>
<tr>
<td>(30, 15)</td>
<td>4.30</td>
</tr>
<tr>
<td>(32, 16)</td>
<td>15.69</td>
</tr>
<tr>
<td>(34, 17)</td>
<td>57.40</td>
</tr>
<tr>
<td>(36, 18)</td>
<td>230.42</td>
</tr>
</tbody>
</table>

Q. Is running time linear, quadratic, cubic, exponential in N?

Performance Challenge 4

Let \( F(N) \) be running time to compute \( \text{binomial}(2N, N) \).

```
public static long binomial(long n, long k) {
    if (k == 0) return 1;
    if (n == 0) return 0;
    return binomial(n-1, k-1) + binomial(n-1, k);
}
```

Observation. \( F(N+1) / F(N) \) converges to about 4.

Q. What is order of growth of the running time?
A. Exponential: \( 4^N \). will not finish unless \( N \) is small.

Dynamic Programming

Key idea. Save solutions to subproblems to avoid recomputation.

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
1 & 1 & 2 & 3 \\
2 & 1 & 3 & 6 \\
3 & 1 & 4 & 10 \\
4 & 1 & 5 & 15 \\
5 & 1 & 6 & 20 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
1 & 1 & 1 & 1 \\
2 & 1 & 2 & 3 \\
3 & 1 & 3 & 6 \\
4 & 1 & 4 & 10 \\
5 & 1 & 5 & 15 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
1 & 1 & 1 & 1 \\
2 & 1 & 2 & 3 \\
3 & 1 & 3 & 6 \\
4 & 1 & 4 & 10 \\
5 & 1 & 5 & 15 \\
\end{array}
\]

\[\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}\]

\[20 = 10 + 10\]

Tradeoff. Trade (a little) memory for (a huge amount of) time.
Binomial Coefficients: Dynamic Programming

```java
public class Binomial {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int K = Integer.parseInt(args[1]);
        long[][] bin = new long[N+1][K+1];

        // base cases
        for (int k = 1; k <= K; k++) bin[0][k] = 0;
        for (int n = 0; n <= N; n++) bin[n][0] = 1;

        // bottom-up dynamic programming
        for (int n = 1; n <= N; n++)
            for (int k = 1; k <= K; k++)
                bin[n][k] = bin[n-1][k-1] + bin[n-1][k];

        // print results
        StdOut.println(bin[N][K]);
    }
}
```

Timing Experiments

Timing experiments for binomial coefficients via dynamic programming.

<table>
<thead>
<tr>
<th>(25, 5)</th>
<th>time 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(26, 13)</td>
<td>instant</td>
</tr>
<tr>
<td>(28, 14)</td>
<td>instant</td>
</tr>
<tr>
<td>(30, 15)</td>
<td>instant</td>
</tr>
<tr>
<td>(32, 16)</td>
<td>instant</td>
</tr>
<tr>
<td>(34, 17)</td>
<td>instant</td>
</tr>
<tr>
<td>(36, 18)</td>
<td>instant</td>
</tr>
</tbody>
</table>

Q. Is running time linear, quadratic, cubic, exponential in \( N^2 \)?

Performance Challenge 5

Let \( F(N) \) be running time to compute \( \text{binomial}(2N, N) \) using DP.

```java
for (int n = 1; n <= N; n++)
    for (int k = 1; k <= K; k++)
        bin[n][k] = bin[n-1][k-1] + bin[n-1][k];
```

Q. What is order of growth of the running time?

A. Quadratic: \( N^2 \)

Remark. There is a profound difference between \( 4^N \) and \( N^2 \).

Assignments

Assignment 5: DNA Sequence Alignment

Biology Review

A genetic sequence is a string of four-letter alphabet
- Adenine (A), Thymine (T), Guanine (G), Cytosine

A gene is a genetic sequence that contains the information needed to
construct a protein. All of your genes taken together are referred
to as the human genome.

Examples:
- A A C A G T T A C C
- T A A G G T C A

Edit Distance: when are the two genetic sequences "similar" enough?
- also useful for spell checking, plagiarism detection, computational
  linguistics, and speech recognition.
Examples: \( x = A A C A G T T A C C \) \( y = T A A G G T C A \)

Edit-distance: the minimum number of editing steps (measured as "cost")
required to go from one string to another.

<table>
<thead>
<tr>
<th>operation</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert a gap</td>
<td>2</td>
</tr>
<tr>
<td>align two characters that mismatch</td>
<td>1</td>
</tr>
<tr>
<td>align two characters that match</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|cccc}
  x & y & cost & x & y & cost \\
  \hline
  A & T & 1 & A & T & 1 \\
  A & A & 0 & A & A & 0 \\
  C & A & 1 & C & - & 2 \\
  A & 0 & 1 & A & A & 0 \\
  G & G & 0 & G & 0 & 0 \\
  T & T & 0 & T & 0 & 1 \\
  T & C & 1 & T & T & 0 \\
  A & A & 0 & A & - & 2 \\
  C & - & 2 & C & C & 0 \\
  C & - & 2 & C & A & 1 \\
  \hline
  & & & & & 8 \\
  & & & & & 7 \\
\end{array}
\]

How to calculate the edit distance?

- Recurse on the suffixes of the two strings \( x[0..M] \) and \( y[0..N] \)
- Notations: \( x[i] \) character \( i \) of the string \( x \).
  - \( x[i..M] \): the suffix of string \( x \) consisting of characters \( x[i] \) ... \( x[M-1] \)
  - \( y[j..N] \): the suffix of string \( y \) consisting of characters \( y[j] \) ... \( y[N-1] \)
  - \( \text{opt}[i][j] \) will store the edit distance between \( x[i..M] \) and \( y[j..N] \)

Calculating things backwards ...

- if we do not insert any gap,
  
  \[
  \text{opt}[i][j] = \text{opt}[i+1][j+1] \\
  \quad \text{if } x[i] = x[j] \\
  \text{opt}[i][j] = \text{opt}[i+1][j+1] + 1 \\
  \quad \text{if } x[i] \neq x[j] \\
  \]

- if we insert a gap btw \( x[i] \) and \( x[i+1] \),
  
  \[
  \text{opt}[i][j] = \text{opt}[i+1][j+1] + 2 \\
  \]

- if we insert a gap btw \( y[j] \) and \( y[j+1] \),
  
  \[
  \text{opt}[i][j] = \text{opt}[i+1][j+1] + 2 \\
  \]

Base case: \( \text{opt}[M][j] = 2(N-j) \) and \( \text{opt}[i][N] = 2(M-i) \)