Thinking Recursively
GTGTC Info Session

• "Are you passionate about computer science, education or both? Do you want to encourage young women interested in learning coding skills?

• Come to the Girls Teaching Girls to Code Info Session! We're going to be discussing our organization, what we do, and how you can get involved.

• We welcome all levels of experience. Learn about Code Camp, our biggest event, as well as other opportunities throughout the year.

• Hope to see you there!

Women's Community Center (Fire Truck House)  
Wednesday, January 25 7-8 PM
Recursive Problem-Solving

if (problem is sufficiently simple) {
  Directly solve the problem.
  Return the solution.
}
else {
  Split the problem up into one or more smaller problems with the same structure as the original.
  Solve each of those smaller problems.
  Combine the results to get the overall solution.
  Return the overall solution.
}
int digitalRootOf(int value);
int sumOfDigitsOf(int value);

int sumOfDigitsOf(int value) {
    if (value < 10) {
        return value;
    } else {
        return sumOfDigitsOf(value / 10) + (value % 10);
    }
}

int digitalRootOf(int value) {
    if (value < 10) {
        return value;
    } else {
        return digitalRootOf(sumOfDigitsOf(value));
    }
}
string reverseOf(const string& text) {
    if (text == "") {
        return "";
    } else {
        return reverseOf(text.substr(1)) + text[0];
    }
}
int bestCoverageFor(const Vector<int>& populations) {
    if (populations.size() == 0) {
        return 0;
    } else if (populations.size() == 1) {
        return populations[0];
    } else {
        Vector<int> allButFirst = tailOf(populations);
        Vector<int> allButFirstTwo = tailOf(allButFirst);

        int withFirst = populations[0] +
                        bestCoverageFor(allButFirstTwo);
        int withoutFirst = bestCoverageFor(allButFirst);

        return max(withFirst, withoutFirst);
    }
}
What's Going On?

- Recursion solves a problem by continuously simplifying the problem until it becomes simple enough to be solved directly.
- The **recursive step** makes the problem slightly simpler.
- The **base case** is what ultimately makes the problem solvable – it guarantees that when the problem is sufficiently simple, we can just solve it directly.
Recursive Problem-Solving

if (problem is sufficiently simple) {
    Directly solve the problem.
    Return the solution.
}
else {
    Split the problem up into one or more smaller problems with the same structure as the original.
    Solve each of those smaller problems.
    Combine the results to get the overall solution.
    Return the overall solution.
}
The Towers of Hanoi Problem
Towers of Hanoi
Towers of Hanoi

Move this tower...
Towers of Hanoi

Move this tower...

...to this spindle.
Towers of Hanoi

Move this tower...

...to this spindle.
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi

A

B

C
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Towers of Hanoi
Solving the Towers of Hanoi

Thanks to Grant Sanderson for the animation idea.
Solving the Towers of Hanoi

Thanks to Grant Sanderson for the animation idea.
Solving the Towers of Hanoi

Thanks to Grant Sanderson for the animation idea.
Solving the Towers of Hanoi

Thanks to Grant Sanderson for the animation idea.
Solving the Towers of Hanoi

Thanks to Grant Sanderson for the animation idea.
Solving the Towers of Hanoi

Thanks to Grant Sanderson for the animation idea.
Solving the Towers of Hanoi

**Step One:** Move the four smaller disks from Spindle A to Spindle B.

**Step Two:** Move the blue disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

**Step One:** Move the four smaller disks from Spindle A to Spindle B.

**Step Two:** Move the blue disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

**Step One:** Move the four smaller disks from Spindle A to Spindle B.

**Step Two:** Move the blue disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Step One: Move the four smaller disks from Spindle A to Spindle B.

Step Two: Move the blue disk from Spindle A to Spindle C.

Step Three: Move the four smaller disks from Spindle B to Spindle C.
Step One: Move the four smaller disks from Spindle $A$ to Spindle $B$.

Step Two: Move the blue disk from Spindle $A$ to Spindle $C$.
Solving the Towers of Hanoi

**Step One:** Move the four smaller disks from Spindle A to Spindle B.

**Step Two:** Move the blue disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

**Step One:** Move the four smaller disks from Spindle A to Spindle B.

**Step Two:** Move the blue disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi

A

B

C
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi

**Step One:** Move the three smaller disks from Spindle A to Spindle C.

**Step Two:** Move the blue disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

**Step One:** Move the three smaller disks from Spindle A to Spindle C.

**Step Two:** Move the blue disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

Step One: Move the three smaller disks from Spindle A to Spindle C.

Step Two: Move the blue disk from Spindle A to Spindle C.

Step Three: Move the four smaller disks from Spindle B to Spindle C.
Step One: Move the three smaller disks from Spindle A to Spindle C.
Solving the Towers of Hanoi

**Step One:** Move the three smaller disks from Spindle A to Spindle C.

**Step Two:** Move the green disk from Spindle A to Spindle B.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

**Step One:** Move the three smaller disks from Spindle A to Spindle C.

**Step Two:** Move the green disk from Spindle A to Spindle B.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Step One: Move the three smaller disks from Spindle A to Spindle C.
Step Two: Move the green disk from Spindle A to Spindle B.
Step Three: Move the three smaller disks from Spindle C to Spindle B.
Solving the Towers of Hanoi

A     B     C
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Step One: Move the two smaller disks from Spindle A to Spindle B.
Solving the Towers of Hanoi

**Step One:** Move the two smaller disks from Spindle A to Spindle B.

**Step Two:** Move the blue disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

Step One: Move the two smaller disks from Spindle A to Spindle B.

Step Two: Move the blue disk from Spindle A to Spindle C.

Step Three: Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

**Step One:** Move the two smaller disks from Spindle A to Spindle B.

**Step Two:** Move the blue disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
**Step One:** Move the two smaller disks from Spindle A to Spindle B.

**Step Two:** Move the yellow disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

**Step One:** Move the two smaller disks from Spindle A to Spindle B.

**Step Two:** Move the yellow disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

Step One: Move the two smaller disks from Spindle A to Spindle B.
Step Two: Move the yellow disk from Spindle A to Spindle C.
Step Three: Move the two smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi

A

B

C
Solving the Towers of Hanoi
Solving the Towers of Hanoi

A

B

C
Solving the Towers of Hanoi

**Step One:** Move the smallest disk from Spindle A to Spindle C.

**Step Two:** Move the blue disk from Spindle A to Spindle C.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

Step One: Move the smallest disk from Spindle A to Spindle C.

Step Two: Move the blue disk from Spindle A to Spindle C.

Step Three: Move the four smaller disks from Spindle B to Spindle C.
Step One: Move the smallest disk from Spindle A to Spindle C.

Step Two: Move the blue disk from Spindle A to Spindle C.

Step Three: Move the four smaller disks from Spindle B to Spindle C.
Step One: Move the smallest disk from Spindle A to Spindle C.

Step Two: Move the blue disk from Spindle A to Spindle C.

Step Three: Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

**Step One:** Move the smallest disk from Spindle A to Spindle C.

**Step Two:** Move the orange disk from Spindle A to Spindle B.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

**Step One:** Move the smallest disk from Spindle A to Spindle C.

**Step Two:** Move the orange disk from Spindle A to Spindle B.

**Step Three:** Move the four smaller disks from Spindle B to Spindle C.
Solving the Towers of Hanoi

Step One: Move the smallest disk from Spindle A to Spindle C.

Step Two: Move the orange disk from Spindle A to Spindle B.

Step Three: Move the smallest disk from Spindle C to Spindle B.
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi
Solving the Towers of Hanoi

**Only Step:** Move the smallest disk from Spindle A to Spindle C.
To move the 5-disk tower from Spindle A to Spindle C:

Step One: Move the four smaller disks from Spindle A to Spindle B.
Step Two: Move the blue disk from Spindle A to Spindle C.
Step Three: Move the four smaller disks from Spindle B to Spindle C.

To move the 4-disk tower from Spindle A to Spindle B:

Step One: Move the three smaller disks from Spindle A to Spindle C.
Step Two: Move the green disk from Spindle A to Spindle B.
Step Three: Move the three smaller disks from Spindle C to Spindle B.

To move the 3-disk tower from Spindle A to Spindle C:

Step One: Move the two smaller disks from Spindle A to Spindle B.
Step Two: Move the yellow disk from Spindle A to Spindle C.
Step Three: Move the two smaller disks from Spindle B to Spindle C.

To move the 2-disk tower from Spindle A to Spindle B:

Step One: Move the smallest disk from Spindle A to Spindle C.
Step Two: Move the orange disk from Spindle A to Spindle B.
Step Three: Move the smallest disk from Spindle C to Spindle B.

To move the 1-disk tower from Spindle A to Spindle C:

Only Step: Move the smallest disk from Spindle A to Spindle C.
**To move the 1-disk tower from Spindle A to Spindle C:**

*Only Step:* Move the smallest disk from Spindle A to Spindle C.

**To move an n-disk tower from Spindle A to Spindle C:**

*Step One:* Move the \((n-1)\) smaller disks from Spindle A to Spindle B.

*Step Two:* Move the \(n\)th disk from Spindle A to Spindle C.

*Step Three:* Move the \((n-1)\) smaller disks from Spindle B to Spindle C.
```c
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

A

C

A

B

C
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}
```
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```
```c
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}
```
```c
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

Given the tower of three disks with sizes 3, 2, and 1, the diagram shows the initial state with disks on platform A. The algorithm will move the disks to platform C, maintaining the规则 that a larger disk cannot be placed on top of a smaller one. The sequence of moves will be as follows:

1. Move the smallest disk from A to B.
2. Move the middle-sized disk from A to C.
3. Move the smallest disk from B to C.
4. Move the middle-sized disk from A to B.
5. Move the largest disk from B to C.
6. Move the smallest disk from A to B.
7. Move the middle-sized disk from B to A.
8. Move the smallest disk from B to C.
9. Move the middle-sized disk from A to C.
10. Move the largest disk from B to A.
11. Move the smallest disk from A to C.
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

n 2 from a to b temp c
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}
```

```c
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```c
int main() {
    moveTower(3, 'a', 'c', 'b);
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

### Diagram

```
\[\begin{array}{c}
\text{A} & \text{B} & \text{C} \\
\text{B} & \text{B} & \text{B} \\
\text{A} & \text{A} & \text{A} \\
\text{A} & \text{A} & \text{A} \\
\text{A} & \text{A} & \text{A} \\
\end{array}\]
```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

void moveSingleDisk(char from, char to) {
    // Code to move a single disk from one peg to another
}
```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

<table>
<thead>
<tr>
<th>n</th>
<th>from</th>
<th>to</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>
```c
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

n 2  from  a  to  b  temp  c
```c
// Move a tower of 3 disks from A to C
moveTower(3, 'A', 'C', 'B');

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

n 1 from c  to b  temp a
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

The diagram shows the initial state with disks on tower A, and the final state with the single disk moved from A to B.
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

The diagram shows the state of the towers at each step of the Tower of Hanoi puzzle with 3 disks. The initial position is with all disks on tower A, and the goal is to move them to tower C, with intermediate moves on tower B.

- **n = 1**: Move disk from C to B.
- **n = 2**: Move disk from A to C, then move disk from B to A, and finally move disk from C to A.
- **n = 3**: Move disk from B to A, then move disk from C to B, move disk from A to C, move disk from B to C, move disk from A to B, move disk from C to A, and finally move disk from B to A.

The diagram illustrates the recursive nature of the solution, moving disks one at a time, while ensuring no two disks overlap.
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

void moveSingleDisk(char from, char to) {
    // Function to move a single disk
}
```
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

Diagram: A

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
</tr>
</tbody>
</table>
```c
int main() {
    void moveTower(int n, char from, char to, char temp) {
        if (n == 1) {
            moveSingleDisk(from, to);
        } else {
            moveTower(n - 1, from, temp, to);
            moveSingleDisk(from, to);
            moveTower(n - 1, temp, to, from);
        }
    }
    moveTower(3, 'a', 'c', 'b');
    return 0;
}
```
```c
int main() {
    int n = 3;
    char from = 'a';
    char to = 'c';
    char temp = 'b';

    moveTower(n, from, to, temp);
}
```

```c
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

```c
void moveSingleDisk(char from, char to) {
    // Implementation
}
```
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

The diagram illustrates the movement of disks from column A to column C using column B as a temporary holder. The disks are represented by different colors: brown, green, and blue. The initial position shows three disks on column A, and the final position shows all disks on column C, with some moves indicated by the movement of the disks.
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
int main() {

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

n 2 from b to c temp a
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

**Diagram:**
```
A  C  B
n  2  from b to c temp a
```
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

**Diagram:**

- **n**: 2
- **from**: b
- **to**: c
- **temp**: a

**Columns:**

- **A**: 2 disks
- **B**: 1 disk
- **C**: 1 disk
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

In the diagram, the blocks represent disks with the current state as follows:
- **A**: Block
- **B**: Yellow square
- **C**: Green square
- **A**: Blue square

The code snippet describes the Towers of Hanoi problem, where the goal is to move all disks from the starting peg to the destination peg, following the rules:
1. Only one disk can be moved at a time.
2. A disk can only be moved if it is the top disk of a stack.
3. A disk cannot be placed on top of a smaller disk.

The diagram shows the initial state with the disks on peg A and the destination peg is C.

The moveTower function is recursive, solving the problem by breaking it down into smaller subproblems. The base case is when `n` is 1, in which case a single disk is moved directly from the start to the destination. For larger `n`, the function recursively moves `n-1` disks from start to temp, then moves the `n`th disk from start to destination, and finally moves the `n-1` disks from temp to destination.
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
# Tower of Hanoi

The Tower of Hanoi is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
3. No disk may be placed on top of a smaller disk.

## Recursive Solution

The solution to the Tower of Hanoi problem is a classic example of a recursive algorithm. The function `moveTower` is defined as follows:

```c
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

This function works as follows:

1. If there is only one disk (`n == 1`), move it directly from the source rod to the destination rod.
2. If there are more than one disk (`n > 1`), recursive calls are made to move `n-1` disks from the source rod to the auxiliary rod, then move the remaining disk from the source rod to the destination rod, and finally move the `n-1` disks back from the auxiliary rod to the destination rod.

## Example

Consider the case where there are 3 disks (`n = 3`) and the initial configuration is (A, B, C) with disks on top of each other on rod A, and the destination is rod B. The moves are as follows:

1. Move disk 1 from A to C.
2. Move disk 2 from A to B.
3. Move disk 1 from C to B.
4. Move disk 3 from A to C.
5. Move disk 1 from B to A.
6. Move disk 2 from B to C.
7. Move disk 1 from A to C.

At this point, all disks are on rod C, and the puzzle is solved.

---

**Diagram**

- **A**: Initial position
- **B**: Intermediate position
- **C**: Destination position

<table>
<thead>
<tr>
<th>Move</th>
<th>From</th>
<th>To</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

The diagram illustrates the disk movement from tower A to tower C via tower B. Initially, there are two disks on tower B, and the goal is to move them to tower C. The movement follows the recursive nature of the moveTower function, where disks are moved one at a time, ensuring they always remain in ascending order on any given tower.
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

Diagram:
```
A
| n 2 | from b | to c | temp a |
```

A: Yellow
B: Green
C: Blue

Initial state:
- A: 2 disks
- B: Empty
- C: Empty

Move tower from B to C:
- Move 1 disk from B to C
- Move 1 disk from A to B
- Move 1 disk from C to B

Final state:
- A: Empty
- B: 1 disk
- C: 2 disks
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

```
<table>
<thead>
<tr>
<th>n</th>
<th>from</th>
<th>to</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>b</td>
<td>c</td>
<td>a</td>
</tr>
</tbody>
</table>
```
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

---

### Diagram

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>1</td>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>from</td>
<td>a</td>
<td>to</td>
<td>temp</td>
</tr>
</tbody>
</table>

The diagram shows the initial state with a single disk on A. In the first step, move the disk from A to C. Then, move the disk from B to A. Finally, move the disk from C to B, completing the tower transfer.
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}

void moveSingleDisk(char from, char to) {
    // Function to move a single disk from one tower to another
}
```
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

The diagram illustrates the movement of disks, starting with a tower of three disks. The function `moveTower` is called with parameters `n = 3`, `from = 'a'`, `to = 'c'`, and `temp = 'b'`. The diagram shows the state of the towers after each recursive call, with the disks represented by colored rectangles.
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

Diagram:
```
1

A

B

C
```
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

Diagram: A to B to C
```c
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}

void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    }
    else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}
Emergent Behavior

- Even though each function call does very little work, the overall behavior of the function is to solve the Towers of Hanoi.
- It's often tricky to think recursively because of this emergent behavior:
  - No one function call solves the entire problem.
  - Each function does only a small amount of work on its own and delegates the rest.
Writing Recursive Functions

- Although it is good to be able to trace through a set of recursive calls to understand how they work, you will need to build up an intuition for recursion to use it effectively.

- You will need to learn to trust that your recursive calls – which are to the function that you are currently writing! – will indeed work correctly.
  - Eric Roberts calls this the “Recursive Leap of Faith.”

- Everyone can learn to think recursively. If this seems confusing now, don't panic! You'll start picking this up as we continue forward.
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
void moveTower(int n, char from, char to, char temp) {
    if (n == 0) {
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
void moveTower(int n, char from, char to, char temp) {
    if (n > 0) {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```java
void moveTower(int n, char from, char to, char temp) {
    if (n > 0) {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```
Picking a Base Case

- When choosing base cases, you should always try to pick the absolute smallest case possible.
- The simplest case is often so simple that it appears silly.
  - Solve Towers of Hanoi with no disks.
  - Add up no numbers.
  - Reverse an empty string.
- This is a skill you'll build up with practice.
Parking Randomly

Car has length one
Parking Randomly

- Given a curb of length five, how many cars, on average, can park on the curb?
- We can get an approximate value through random simulation:
  - Simulate random parking a large number of times.
  - Output the average number of cars that could park.
- **Question**: How do we simulate parking cars on the curb?
Parking Randomly
Parking Randomly
Parking Randomly

\[ x \quad x + 1 \]
Parking Randomly

0 \quad x \quad x + 1 \quad 5
Parking Randomly

Place cars randomly in these ranges!
Parking Randomly

```java
int parkRandomly(double low, double high) {
    if (high - low < 1.0) {
        return 0;
    } else {
        double position = randomReal(low, high - 1.0);
        drawCarAt(position);

        return 1 + parkRandomly(low, position) +
        parkRandomly(position + 1.0, high);
    }
}
```
The Parking Ratio

• The average number of cars that can be parked in a range of width $w$ for sufficiently large $w$ is approximately $0.7475972 \times w$

• The constant $0.7475972\ldots$ is called Rényi's Parking Constant.

• For more details, visit this link!
So What?

- The beauty of our algorithm is the following recursive insight:
  
  *Split a problem into smaller, independent pieces and solve each piece separately.*

- Many problems can be solved this way.
Next Time

- **Graphical Recursion**
  - How do you draw a self-similar object?
- **Exhaustive Recursion**
  - How do you generate all objects of some type?