# Tower of Hanoi

| n | start | finish | temp |
|---|---|---|---|
| 3 | A | B | C |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          MoveSingleDisk(start, finish);
        } else {
          ->MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
} }
```

| n | start | finish | temp |
|---|---|---|---|
| 2 | A | C | B |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          MoveSingleDisk(start, finish);
        } else {
          ->MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
} }
```

| n | start | finish | temp |
|---|---|---|---|
| 1 | A | B | C |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          ->MoveSingleDisk(start, finish);
        } else {
          MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
} }
```

1

| n | start | finish | temp |
|---|---|---|---|
| 2 | A | C | B |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          MoveSingleDisk(start, finish);
        } else {
          MoveTower(n - 1, start, temp, finish);
          ->MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
} }
```

2

| n | start | finish | temp |
|---|---|---|---|
| 2 | A | C | B |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          MoveSingleDisk(start, finish);
        } else {
          MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          ->MoveTower(n - 1, temp, finish, start);
} }
```

| n | start | finish | temp |
|---|---|---|---|
| 1 | B | C | A |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          ->MoveSingleDisk(start, finish);
        } else {
          MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
} }
```

3

| n | start | finish | temp |
|---|---|---|---|
| 2 | A | C | B |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          MoveSingleDisk(start, finish);
        } else {
          MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
->) }
```

| n | start | finish | temp |
|---|---|---|---|
| 3 | A | B | C |

```
 void MoveTower(int n, char start, char finish, char
temp) {
   if (n == 1) {
     MoveSingleDisk(start, finish);
   } else {
     MoveTower(n - 1, start, temp, finish);
     ->MoveSingleDisk(start, finish);
     MoveTower(n - 1, temp, finish, start);
} }
```

4

| n | start | finish | temp |
|---|---|---|---|
| 3 | A | B | C |

```
 void MoveTower(int n, char start, char finish, char
temp) {
   if (n == 1) {
     MoveSingleDisk(start, finish);
   } else {
     MoveTower(n - 1, start, temp, finish);
     MoveSingleDisk(start, finish);
     ->MoveTower(n - 1, temp, finish, start);
} }
```

| n | start | finish | temp |
|---|---|---|---|
| 2 | C | B | A |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          MoveSingleDisk(start, finish);
        } else {
          ->MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
} }
```

| n | start | finish | temp |
|---|---|---|---|
| 1 | C | A | B |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          ->MoveSingleDisk(start, finish);
        } else {
          MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
} }
```

5

| n | start | finish | temp |
|---|---|---|---|
| 2 | C | B | A |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          MoveSingleDisk(start, finish);
        } else {
          MoveTower(n - 1, start, temp, finish);
          ->MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
} }
```

6

| n | start | finish | temp |
|---|---|---|---|
| 2 | C | B | A |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          MoveSingleDisk(start, finish);
        } else {
          MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          -> MoveTower(n - 1, temp, finish, start);
} }
```

| n | Start | Finish | Temp |
|---|---|---|---|
| 1 | A | B | C |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          ->MoveSingleDisk(start, finish);
        } else {
          MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
} }
```

7

| n | start | finish | temp |
|---|---|---|---|
| 2 | C | B | A |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          MoveSingleDisk(start, finish);
        } else {
          MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
->) }
```

| n | Start | Finish | Temp |
|---|---|---|---|
| 3 | A | B | C |

```
 void MoveTower(int n, char start, char finish, char temp) {
        if (n == 1) {
          MoveSingleDisk(start, finish);
        } else {
          MoveTower(n - 1, start, temp, finish);
          MoveSingleDisk(start, finish);
          MoveTower(n - 1, temp, finish, start);
->) }
```

# String Permutation

| n | prefix | rest | i | |
|---|---|---|---|---|
| 3 | | ABC | 0 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            ->RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 2 | A | BC | 0 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            ->RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | AB | C | 0 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            ->RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 0 | ABC | | 0 | ABC |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        ->print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | AB | C | 1 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    ->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 2 | A | BC | 1 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
          ->  RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | AC | B | 0 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            ->RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 0 | ACB | | 0 | ACB |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        ->print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | AC | B | 1 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    ->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 2 | A | BC | 2 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    ->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 3 | | ABC | 1 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
          -> RecursivePermute(newPrefix, newRest);
        }
->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 2 | B | AC | 0 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            -> RecursivePermute(newPrefix, newRest);
        }
->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | BA | C | 0 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            ->RecursivePermute(newPrefix, newRest);
        }
    ->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 0 | BAC | | 0 | BAC |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        ->print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | BA | C | 1 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    ->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 2 | B | AC | 1 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            ->RecursivePermute(newPrefix, newRest);
        }
    ->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | BC | A | 0 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            ->RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 0 | BCA | | 0 | BCA |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        ->print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | BC | A | 1 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    ->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 2 | B | AC | 2 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    ->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 3 | | ABC | 2 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            ->RecursivePermute(newPrefix, newRest);
        }
->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 2 | C | AB | 0 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
          ->  RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | CA | B | 0 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        ->print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            -> RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 0 | CAB | | 0 | CAB |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        ->print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | CA | B | 1 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    ->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 2 | C | AB | 1 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            ->RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | CB | A | 0 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            -> RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 0 | CBA | | 0 | CBA |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        ->print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 1 | CB | A | 1 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    ->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 2 | C | AB | 2 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
    ->i }
}
```

| n | prefix | rest | i | |
|---|---|---|---|---|
| 3 | | ABC | 3 | |

```
void RecursivePermute(string prefix, string rest) {
    if (rest == "") {
        print prefix;
    } else {
        for (int i = 0; i < n; i++) {
            string newPrefix = prefix + rest[i];
            string newRest = rest.substr(0, i) + rest.substr(i+1);
            RecursivePermute(newPrefix, newRest);
        }
->i }
}
```