# Recursion

Mitsu Ogihara

Department of Computer Science
University of Miami

# Table of Contents

# Outline

1. Recursion
   - What is a recursion?
   - A classic example of recursion: the factorial
2. More examples
   - GCD
   - The Tower of Hanoi
   - Binary Search

# Recursion

We say that a problem $Q$ can be "recursively solved" if $Q$ exhibits the following property.

- Each problem instance $x$ of $Q$ can be measured for its size, where the size value is a nonnegative integer.
- All problem instances $x$ of $Q$ having small size (for example, size 0) can be quickly solved $\cdots$ **trivial cases**
- Each nontrivial case $x$ of $Q$ can be solved by obtaining an answer to another instance $y$, each having smaller size than $x$, and then applying some computing to the obtained answer

# Recursive Solution General Structure

```
some_type_T solve(some_type_E x) {
  if (x is trivial) {
    return easy solution of x;
  }
  generate a smaller instance y;
  T = solve( y );
  Calculate and return the solution for x;
}
```

# Recursive Solution General Structure

```
some_type_T solve(some_type_E x) {
  if (x is trivial) {
    return easy solution of x;
  }
  generate a smaller instance y;
  T = solve( y );
  Calculate and return the solution for x;
}
```

A curious feature of a recursive method is that a call to the method itself appears in it

# Outline

# A Classic Example: the Factorial

- Recall that the factorial of a positive integer $n$, denoted by $n!$, is the product of all integers from 1 to $n$
- By definition for all integers $n <= 1$, $n! = 1$
- For each $n > 1$, $n! = (n-1)! * n$

# A Classic Example: the Factorial

- Recall that the factorial of a positive integer $n$, denoted by $n!$, is the product of all integers from 1 to $n$
- By definition for all integers $n <= 1$, $n! = 1$
- For each $n > 1$, $n! = (n-1)! * n$
- This suggests the following solution:
  - If $n <= 1$, return 1;
  - If $n >= 2$, compute $(n-1)!$, multiply it by $n$, and then return the product

# FactorialRecursive.java

```
// compute factorial recursively
public class FactorialRecursive {
  //-- recursive method for computing the factorial
  public static long compute( int n ) {
    // returnValue is the value to be returned
    long returnValue;
    // the trival case
    if ( n <= 1 ) {
      returnValue = 1;
    }
    // the recursive case
    else {
      returnValue = compute( n - 1 ) * n;
    }
    // print the result
    System.out.printf( "n=%-3d n!=%30d%n", n, returnValue );
    // return the value computed
    return returnValue;
  }
```

This `compute` is the method for computing the factorial

# FactorialRecursive.java

```java
// compute factorial recursively
public class FactorialRecursive {
  //-- recursive method for computing the factorial
  public static long compute( int n ) {
    // returnValue is the value to be returned
    long returnValue;
    // the trival case
    if ( n <= 1 ) {
      returnValue = 1;
    }
    // the recursive case
    else {
      returnValue = compute( n - 1 ) * n;
    }
    // print the result
    System.out.printf( "n=%-3d n!=%30d%n", n, returnValue );
    // return the value computed
    return returnValue;
  }
```

Here we use the 64-bit `long` instead of the 32-bit `int` so as to be able to compute a large value

# FactorialRecursive.java

```java
// compute factorial recursively
public class FactorialRecursive {
  //-- recursive method for computing the factorial
  public static long compute( int n ) {
    // returnValue is the value to be returned
    long returnValue;
    // the trival case
    if ( n <= 1 ) {
      returnValue = 1;
    }
    // the recursive case
    else {
      returnValue = compute( n - 1 ) * n;
    }
    // print the result
    System.out.printf( "n=%-3d n!=%30d%n", n, returnValue );
    // return the value computed
    return returnValue;
  }
```

The trivial case $n <= 1$: 1 is stored in `returnValue`

# FactorialRecursive.java

```java
// compute factorial recursively
public class FactorialRecursive {
  //-- recursive method for computing the factorial
  public static long compute( int n ) {
    // returnValue is the value to be returned
    long returnValue;
    // the trival case
    if ( n <= 1 ) {
      returnValue = 1;
    }
    // the recursive case
    else {
      returnValue = compute( n - 1 ) * n;
    }
    // print the result
    System.out.printf( "n=%-3d n!=%30d%n", n, returnValue );
    // return the value computed
    return returnValue;
  }
```

The recursive case $n > 1$; the value of `compute(n-1)` is multiplied by `n` and then stored in `returnValue`

# FactorialRecursive.java

```java
// compute factorial recursively
public class FactorialRecursive {
  //-- recursive method for computing the factorial
  public static long compute( int n ) {
    // returnValue is the value to be returned
    long returnValue;
    // the trival case
    if ( n <= 1 ) {
      returnValue = 1;
    }
    // the recursive case
    else {
      returnValue = compute( n - 1 ) * n;
    }
    // print the result
    System.out.printf( "n=%-3d n!=%30d%n", n, returnValue );
    // return the value computed
    return returnValue;
  }
```

The value calculated is printed and then returned

# FactorialRecursive.java

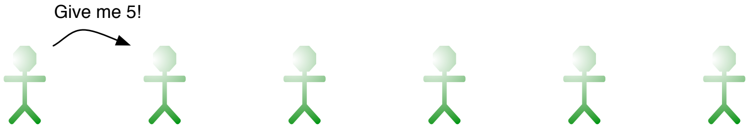
```java
  //-- main method
  public static void main( String[] args ) {
    int number;
    String response;
    do {
      number = Tools.getInt( "Enter value between 0 and 30: " );
      if ( number >= 0 && number <= 30 ) {
        compute( number );
      }
      response = Tools.getString( "Try again? (y/n): " );
    } while ( response.startsWith( "y" ) );
  }
}
```

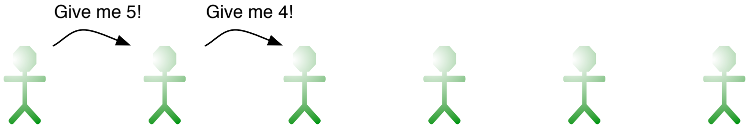
Receive input from the user and compute the factorial

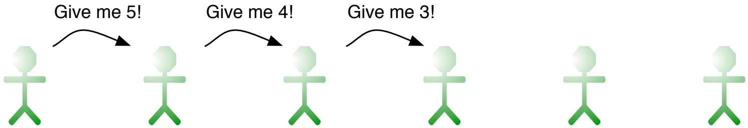
# FactorialRecursive.java

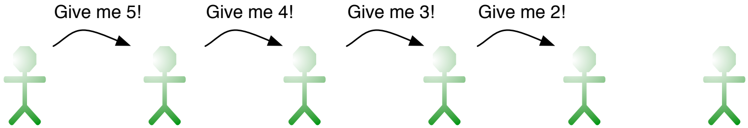
```java
  //-- main method
  public static void main( String[] args ) {
    int number;
    String response;
    do {
      number = Tools.getInt( "Enter value between 0 and 30: " );
      if ( number >= 0 && number <= 30 ) {
        compute( number );
      }
      response = Tools.getString( "Try again? (y/n): " );
    } while ( response.startsWith( "y" ) );
  }
}
```

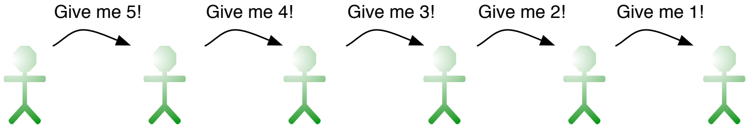
Ask the user whether he/she wants to continue

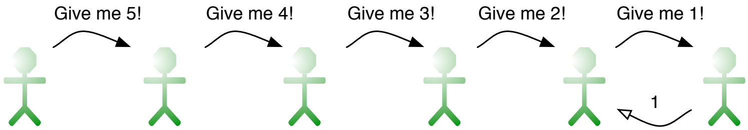

Give me 5!

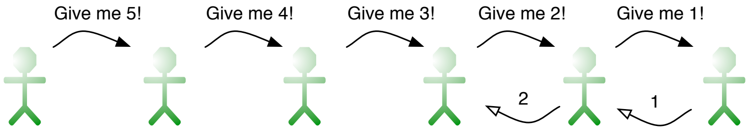

Give me 5!
Give me 4!

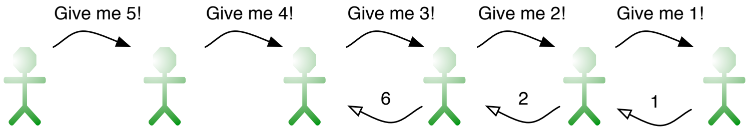

Give me 5!
Give me 4!
Give me 3!

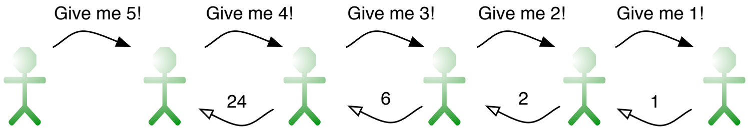

Give me 5!
Give me 4!
Give me 3!
Give me 2!

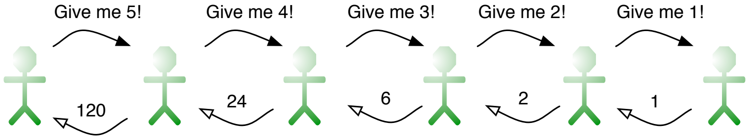

Give me 5!
Give me 4!
Give me 3!
Give me 2!
Give me 1!

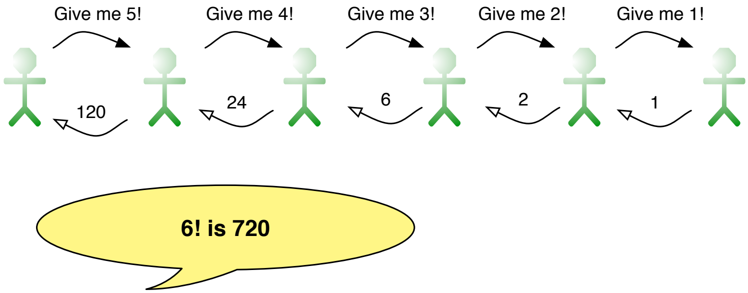

Give me 5!
Give me 4!
Give me 3!
Give me 2!
Give me 1!
1


Give me 5!
Give me 4!
Give me 3!
Give me 2!
Give me 1!
2
1


Give me 5!
Give me 4!
Give me 3!
Give me 2!
Give me 1!
6
2
1


Give me 5!
Give me 4!
Give me 3!
Give me 2!
Give me 1!
24
6
2
1


Give me 5!
Give me 4!
Give me 3!
Give me 2!
Give me 1!
120
24
6
2
1


Give me 5!
Give me 4!
Give me 3!
Give me 2!
Give me 1!
120
24
6
2
1
6! is 720

# Table of Contents

# Outline

# GCD

- The **greatest common divisor (GCD)** of two integers is the largest positive integer that divides both numbers, for example:
  - GCD(20, 35) = 5
  - GCD(14, 5) = 1
  - GCD(-121, -143) = 11
  - GCD(0, 191) = 191

# Euclid's Algorithm

A famous algorithm by Euclid to compute the GCD of nonnegative integers $n$ and $m$

- While $n > 0$ repeat the following:
  - Set $r$ to $m\%n$
  - Replace $m$ with $n$
  - Replace $n$ with $r$
- When $n$ becomes 0, $m$ is the GCD

# Euclid's Algorithm

A famous algorithm by Euclid to compute the GCD of nonnegative integers $n$ and $m$

- While $n > 0$ repeat the following:
  - Set $r$ to $m\%n$
  - Replace $m$ with $n$
  - Replace $n$ with $r$
- When $n$ becomes 0, $m$ is the GCD

Examples:

- $(14, 35) \to (35, 14) \to (14, 7) \to (7, 0)$ and so the GCD is 7

# Euclid's Algorithm

A famous algorithm by Euclid to compute the GCD of nonnegative integers $n$ and $m$

- While $n > 0$ repeat the following:
  - Set $r$ to $m\%n$
  - Replace $m$ with $n$
  - Replace $n$ with $r$
- When $n$ becomes 0, $m$ is the GCD

Examples:

- $(14, 35) \rightarrow (35, 14) \rightarrow (14, 7) \rightarrow (7, 0)$ and so the GCD is 7
- $(10458, 2505) \rightarrow (2505, 438) \rightarrow (438, 315) \rightarrow (315, 123) \rightarrow (123, 69) \rightarrow (69, 54) \rightarrow (54, 15) \rightarrow (15, 9) \rightarrow (9, 6) \rightarrow (6, 3) \rightarrow (3, 0)$ and so the GCD is 3

# GCDRecursive.java

```java
import java.util.Scanner;
public class GCDRecursive {
  public static int GCD( int a, int b ) {
    System.out.println( "a = " + a + ", b = " + b );
    if ( b == 0 ) {
      return a;
    }
    return GCD( b, a % b );
  }
  public static void main( String[] args ) {
    Scanner console = new Scanner( System.in );
    System.out.print( "Enter two numbers: " );
    int m = console.nextInt();
    int n = console.nextInt();
    if ( m >= 0 && n >= 0 ) {
      int g = GCD( m, n );
      System.out.printf( "The GCD of %d and %d is %d%n", m, n, g );
    }
  }
}
```

Method header.

# GCDRecursive.java

```
import java.util.Scanner;
public class GCDRecursive {
  public static int GCD( int a, int b ) {
    System.out.println( "a = " + a + ", b = " + b );
    if ( b == 0 ) {
      return a;
    }
    return GCD( b, a % b );
  }
  public static void main( String[] args ) {
    Scanner console = new Scanner( System.in );
    System.out.print( "Enter two numbers: " );
    int m = console.nextInt();
    int n = console.nextInt();
    if ( m >= 0 && n >= 0 ) {
      int g = GCD( m, n );
      System.out.printf( "The GCD of %d and %d is %d%n", m, n, g );
    }
  }
}
```

Print the two numbers.

# GCDRecursive.java

```java
import java.util.Scanner;
public class GCDRecursive {
  public static int GCD( int a, int b ) {
    System.out.println( "a = " + a + ", b = " + b );
    if ( b == 0 ) {
      return a;
    }
    return GCD( b, a % b );
  }
  public static void main( String[] args ) {
    Scanner console = new Scanner( System.in );
    System.out.print( "Enter two numbers: " );
    int m = console.nextInt();
    int n = console.nextInt();
    if ( m >= 0 && n >= 0 ) {
      int g = GCD( m, n );
      System.out.printf( "The GCD of %d and %d is %d%n", m, n, g );
    }
  }
}
```

Base case: b==0

# GCDRecursive.java

```java
import java.util.Scanner;
public class GCDRecursive {
  public static int GCD( int a, int b ) {
    System.out.println( "a = " + a + ", b = " + b );
    if ( b == 0 ) {
      return a;
    }
    return GCD( b, a % b );
  }
  public static void main( String[] args ) {
    Scanner console = new Scanner( System.in );
    System.out.print( "Enter two numbers: " );
    int m = console.nextInt();
    int n = console.nextInt();
    if ( m >= 0 && n >= 0 ) {
      int g = GCD( m, n );
      System.out.printf( "The GCD of %d and %d is %d%n", m, n, g );
    }
  }
}
```

Recusive case. Make a recursive call

# GCDRecursive.java

```java
import java.util.Scanner;
public class GCDRecursive {
  public static int GCD( int a, int b ) {
    System.out.println( "a = " + a + ", b = " + b );
    if ( b == 0 ) {
      return a;
    }
    return GCD( b, a % b );
  }
  public static void main( String[] args ) {
    Scanner console = new Scanner( System.in );
    System.out.print( "Enter two numbers: " );
    int m = console.nextInt();
    int n = console.nextInt();
    if ( m >= 0 && n >= 0 ) {
      int g = GCD( m, n );
      System.out.printf( "The GCD of %d and %d is %d%n", m, n, g );
    }
  }
}
```

Receive two numbers from the user.

# GCDRecursive.java

```java
import java.util.Scanner;
public class GCDRecursive {
  public static int GCD( int a, int b ) {
    System.out.println( "a = " + a + ", b = " + b );
    if ( b == 0 ) {
      return a;
    }
    return GCD( b, a % b );
  }
  public static void main( String[] args ) {
    Scanner console = new Scanner( System.in );
    System.out.print( "Enter two numbers: " );
    int m = console.nextInt();
    int n = console.nextInt();
    if ( m >= 0 && n >= 0 ) {
      int g = GCD( m, n );
      System.out.printf( "The GCD of %d and %d is %d%n", m, n, g );
    }
  }
}
```

Execute GCD and report the result

# Outline

# Tower of Hanoi

- The legend says there is a temple in a remote place in Hanoi, where the monks are counting the time till the end of the world

# Tower of Hanoi

- The legend says there is a temple in a remote place in Hanoi, where the monks are counting the time till the end of the world
- There are 64 golden disks of all distinct diameters and there are three polls that the three disks surround

# Tower of Hanoi

- The legend says there is a temple in a remote place in Hanoi, where the monks are counting the time till the end of the world
- There are 64 golden disks of all distinct diameters and there are three polls that the three disks surround
- The disks were originally surrounding one pole, in the decreasing order of diameter

# Tower of Hanoi

- The legend says there is a temple in a remote place in Hanoi, where the monks are counting the time till the end of the world
- There are 64 golden disks of all distinct diameters and there are three polls that the three disks surround
- The disks were originally surrounding one pole, in the decreasing order of diameter
- The monks are moving the disks to another pole under the following condition:
    - A disk of larger diameter cannot be placed on a disk of smaller diameter

# Tower of Hanoi

- The legend says there is a temple in a remote place in Hanoi, where the monks are counting the time till the end of the world
- There are 64 golden disks of all distinct diameters and there are three polls that the three disks surround
- The disks were originally surrounding one pole, in the decreasing order of diameter
- The monks are moving the disks to another pole under the following condition:
    - A disk of larger diameter cannot be placed on a disk of smaller diameter
- The world ends when the monks complete the task of moving the disks

# The Four-disk Version

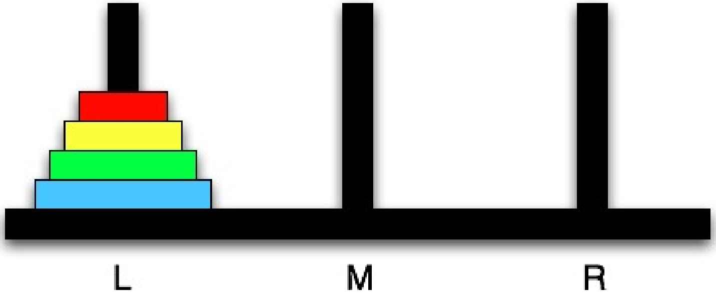
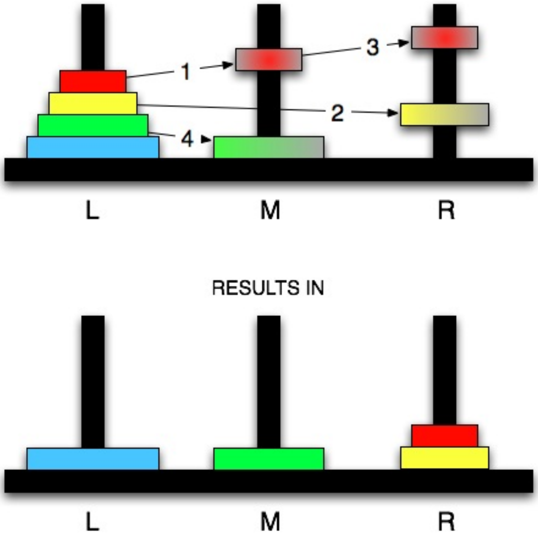



Figure out how the task can be accomplished and how many steps will be needed

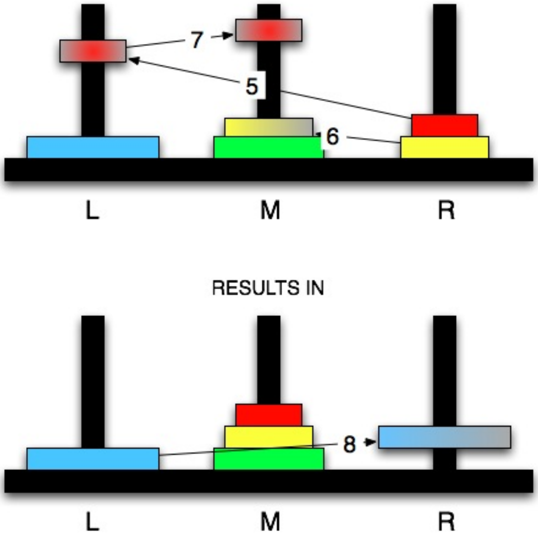
# Solution Idea

- Suppose N disks are to move from a pole X to a pole Y
- Call the remaining pole Z
- If N is one, simply move the unique disk from X to Y
- If N is greater than one,
  - move the top N-1 disks from X to Z,
  - move the Nth disk from X to Y, and then
  - move the top N-1 disks from Z to Y

# Solution idea



# Solution idea



# Hanoi.java

```
import java.util. * ;
public class Hanoi {
  static String[] names = { "L", "M", "R" };
  static void solve( int number, int fromPole, int toPole ) {
    if ( number == 1 ) {
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
    }
    else {
      int remainder = 3 - fromPole - toPole;
      solve( number - 1, fromPole, remainder );
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
      solve( number - 1, remainder, toPole );
    }
  }
  public static void main( String[] args ) {
    solve( Integer.parseInt( args[ 0 ] ), 0, 1 );
  }
}
```

Names of the poles

# Hanoi.java

```java
import java.util. * ;
public class Hanoi {
  static String[] names = { "L", "M", "R" };
  static void solve( int number, int fromPole, int toPole ) {
    if ( number == 1 ) {
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
    }
    else {
      int remainder = 3 - fromPole - toPole;
      solve( number - 1, fromPole, remainder );
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
      solve( number - 1, remainder, toPole );
    }
  }
  public static void main( String[] args ) {
    solve( Integer.parseInt( args[ 0 ] ), 0, 1 );
  }
}
```

The Hanoi solver: the parameters are the number of disks to move, the starting pole, the target pole

# Hanoi.java

```java
import java.util. * ;
public class Hanoi {
  static String[] names = { "L", "M", "R" };
  static void solve( int number, int fromPole, int toPole ) {
    if ( number == 1 ) {
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
    }
    else {
      int remainder = 3 - fromPole - toPole;
      solve( number - 1, fromPole, remainder );
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
      solve( number - 1, remainder, toPole );
    }
  }
  public static void main( String[] args ) {
    solve( Integer.parseInt( args[ 0 ] ), 0, 1 );
  }
}
```

The base case

# Hanoi.java

```
import java.util. * ;
public class Hanoi {
  static String[] names = { "L", "M", "R" };
  static void solve( int number, int fromPole, int toPole ) {
    if ( number == 1 ) {
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
    }
    else {
      int remainder = 3 - fromPole - toPole;
      solve( number - 1, fromPole, remainder );
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
      solve( number - 1, remainder, toPole );
    }
  }
  public static void main( String[] args ) {
    solve( Integer.parseInt( args[ 0 ] ), 0, 1 );
  }
}
```

The recursive case: first obtain the remaining pole

# Hanoi.java

```java
import java.util. * ;
public class Hanoi {
  static String[] names = { "L", "M", "R" };
  static void solve( int number, int fromPole, int toPole ) {
    if ( number == 1 ) {
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
    }
    else {
      int remainder = 3 - fromPole - toPole;
      solve( number - 1, fromPole, remainder );
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
      solve( number - 1, remainder, toPole );
    }
  }
  public static void main( String[] args ) {
    solve( Integer.parseInt( args[ 0 ] ), 0, 1 );
  }
}
```

Move all but the last to the remainder

# Hanoi.java

```
import java.util. * ;
public class Hanoi {
  static String[] names = { "L", "M", "R" };
  static void solve( int number, int fromPole, int toPole ) {
    if ( number == 1 ) {
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
    }
    else {
      int remainder = 3 - fromPole - toPole;
      solve( number - 1, fromPole, remainder );
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
      solve( number - 1, remainder, toPole );
    }
  }
  public static void main( String[] args ) {
    solve( Integer.parseInt( args[ 0 ] ), 0, 1 );
  }
}
```

Move the last to the target

# Hanoi.java

```java
import java.util. * ;
public class Hanoi {
  static String[] names = { "L", "M", "R" };
  static void solve( int number, int fromPole, int toPole ) {
    if ( number == 1 ) {
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
    }
    else {
      int remainder = 3 - fromPole - toPole;
      solve( number - 1, fromPole, remainder );
      System.out.printf( "Move %d from %s to %s%n", number,
          names[ fromPole ], names[ toPole ] );
      solve( number - 1, remainder, toPole );
    }
  }
  public static void main( String[] args ) {
    solve( Integer.parseInt( args[ 0 ] ), 0, 1 );
  }
}
```

Move the disks located at the remainder to the target

# The number of steps

- $N = 1$: 1
- $N >= 2$: $2 * \texttt{the} - \texttt{no} - \texttt{steps} - \texttt{for} - (N-1) + 1$
- This gives: $2^N - 1$

# The number of steps

- $N = 1$: 1
- $N >= 2$: $2 * \mathtt{the-no-steps-for}{-}(N-1)+1$
- This gives: $2^N - 1$

If $N = 64$, this is $18,446,744,073,709,551,615$
If a disk can be moved in a second, **this will be 585 billion years!**

# Outline

# Finding a Key in a Sorted Array of Integers

The sequential search for the key requires examination of all keys in the worst case scenario

Do the following for a better search:

# Finding a Key in a Sorted Array of Integers

The sequential search for the key requires examination of all keys in the worst case scenario

Do the following for a better search:

- Use two integers `start` and `end` to specify the range $[start, end - 1]$ in which the key is searched for

# Finding a Key in a Sorted Array of Integers

The sequential search for the key requires examination of all keys in the worst case scenario
Do the following for a better search:

- Use two integers `start` and `end` to specify the range $[start, end - 1]$ in which the key is searched for
- Examine the halfway point, say `mid`, between the two end points

# Finding a Key in a Sorted Array of Integers

The sequential search for the key requires examination of all keys in the worst case scenario
Do the following for a better search:

- Use two integers `start` and `end` to specify the range $[start, end - 1]$ in which the key is searched for
- Examine the halfway point, say `mid`, between the two end points
  - If the element is the key you are looking for, the search is over

# Finding a Key in a Sorted Array of Integers

The sequential search for the key requires examination of all keys in the worst case scenario
Do the following for a better search:

- Use two integers `start` and `end` to specify the range $[start, end-1]$ in which the key is searched for
- Examine the halfway point, say `mid`, between the two end points
  - If the element is the key you are looking for, the search is over
  - If the element is greater than the key, update `end` with `mid`-1

# Finding a Key in a Sorted Array of Integers

The sequential search for the key requires examination of all keys in the worst case scenario

Do the following for a better search:

- Use two integers `start` and `end` to specify the range $[start, end-1]$ in which the key is searched for
- Examine the halfway point, say `mid`, between the two end points
  - If the element is the key you are looking for, the search is over
  - If the element is greater than the key, update `end` with `mid`-1
  - If the element is smaller than the key, update `start` with `mid`+1

# Finding a Key in a Sorted Array of Integers

The sequential search for the key requires examination of all keys in the worst case scenario
Do the following for a better search:

- Use two integers `start` and `end` to specify the range $[start, end - 1]$ in which the key is searched for
- Examine the halfway point, say `mid`, between the two end points
  - If the element is the key you are looking for, the search is over
  - If the element is greater than the key, update `end` with `mid`-1
  - If the element is smaller than the key, update `start` with `mid`+1
- Eventually, either you find the key or `start` becomes equal to `end`, that is when you know that the key does not appear in the array

# Binary Search Code (print array)

```
public static final int WIDTH = 8;
public static void printArray( int[] nums, int start, int end ) {
  for ( int index = 0; index < nums.length; index ++ ) {
    if ( index % WIDTH == 0 ) {
      System.out.printf( "%3d: ", index );
    }
    if ( index < start || index >= end ) {
      System.out.print( "...." );
    }
    else {
      System.out.printf( "%4d", nums[ index ] );
    }
    if ( index == nums.length - 1
        || ( index % WIDTH ) == WIDTH - 1 ) {
      System.out.println();
    }
    else {
      System.out.print( " " );
    }
  }
}
```

The width parameter ... the number of elements per line

# Binary Search Code (print array)

```
public static final int WIDTH = 8;
public static void printArray( int[] nums, int start, int end ) {
  for ( int index = 0; index < nums.length; index ++ ) {
    if ( index % WIDTH == 0 ) {
      System.out.printf( "%3d: ", index );
    }
    if ( index < start || index >= end ) {
      System.out.print( "...." );
    }
    else {
      System.out.printf( "%4d", nums[ index ] );
    }
    if ( index == nums.length - 1
        || ( index % WIDTH ) == WIDTH - 1 ) {
      System.out.println();
    }
    else {
      System.out.print( " " );
    }
  }
}
```

The method takes as parameters a sorted array and a range

# Binary Search Code (print array)

```
  public static final int WIDTH = 8;
  public static void printArray( int[] nums, int start, int end ) {
    for ( int index = 0; index < nums.length; index ++ ) {
      if ( index % WIDTH == 0 ) {
        System.out.printf( "%3d: ", index );
      }
      if ( index < start || index >= end ) {
        System.out.print( "...." );
      }
      else {
        System.out.printf( "%4d", nums[ index ] );
      }
      if ( index == nums.length - 1
          || ( index % WIDTH ) == WIDTH - 1 ) {
        System.out.println();
      }
      else {
        System.out.print( " " );
      }
    }
  }
```

At the beginning of line, print the index

# Binary Search Code (print array)

```
public static final int WIDTH = 8;
public static void printArray( int[] nums, int start, int end ) {
  for ( int index = 0; index < nums.length; index ++ ) {
    if ( index % WIDTH == 0 ) {
      System.out.printf( "%3d: ", index );
    }
    if ( index < start || index >= end ) {
      System.out.print( "...." );
    }
    else {
      System.out.printf( "%4d", nums[ index ] );
    }
    if ( index == nums.length - 1
        || ( index % WIDTH ) == WIDTH - 1 ) {
      System.out.println();
    }
    else {
      System.out.print( " " );
    }
  }
}
```

Substitute the numbers outside the range with `"...."`

# Binary Search Code (print array)

```
public static final int WIDTH = 8;
public static void printArray( int[] nums, int start, int end ) {
  for ( int index = 0; index < nums.length; index ++ ) {
    if ( index % WIDTH == 0 ) {
      System.out.printf( "%3d: ", index );
    }
    if ( index < start || index >= end ) {
      System.out.print( "...." );
    }
    else {
      System.out.printf( "%4d", nums[ index ] );
    }
    if ( index == nums.length - 1
        || ( index % WIDTH ) == WIDTH - 1 ) {
      System.out.println();
    }
    else {
      System.out.print( " " );
    }
  }
}
```

Print the numbers inside the range in four letters

# Binary Search Code (print array)

```
public static final int WIDTH = 8;
public static void printArray( int[] nums, int start, int end ) {
  for ( int index = 0; index < nums.length; index ++ ) {
    if ( index % WIDTH == 0 ) {
      System.out.printf( "%3d: ", index );
    }
    if ( index < start || index >= end ) {
      System.out.print( "...." );
    }
    else {
      System.out.printf( "%4d", nums[ index ] );
    }
    if ( index == nums.length - 1
        || ( index % WIDTH ) == WIDTH - 1 ) {
      System.out.println();
    }
    else {
      System.out.print( " " );
    }
  }
}
```

Punctuation. Either space or newline

# Binary Search Code (hit return)

```
  public static void next( String message ) {
    System.out.print( w + "......" + HIT RETURN );
    ( new Scanner( System.in ) ).nextLine();
  }
```

Method for receiving any string from the user.
Print the string given as the parameter and then a prompt.
Then receive a new line using the direct creation of a new Scanner object and attaching to the object the nextline method

# Binary Search Code (find method)

```
  public static int find( int[] numbers, int key ) {
    return find( numbers, key, 0, numbers.length );
  }
```

The search method. Takes an array and the key to search. Return a position at the key is located.

# Binary Search Code (find method)

```
public static int find( int[] numbers, int key ) {
  return find( numbers, key, 0, numbers.length );
}
```

Call the method (via overloading) by specifying the search range.

# Binary Search Code (find method, full)

```
public static int find( int[] numbers, int key, int start, int end ) {
  System.out.println();
  printArray( numbers, start, end );
  if ( start >= end ) {
    next( "SEARCH RANGE EMPTY" );
    return - 1;
  }
  int mid = ( start + end ) / 2;
  System.out.printf( "MID=%d,value=%d, ", mid, numbers[ mid ] );
  if ( numbers[ mid ] == key ) {
    next( "MATCH!" );
    return mid;
  }
  else if ( numbers[ mid ] > key ) {
    next( "SEARCH LEFT!" );
    return find( numbers, key, start, mid );
  }
  else {
    next( "SEARCH RIGHT!" );
    return find( numbers, key, mid + 1, end );
  }
}
```

The header.

# Binary Search Code (find method, full)

```java
public static int find( int[] numbers, int key, int start, int end ) {
  System.out.println();
  printArray( numbers, start, end );
  if ( start >= end ) {
    next( "SEARCH RANGE EMPTY" );
    return - 1;
  }
  int mid = ( start + end ) / 2;
  System.out.printf( "MID=%d,value=%d, ", mid, numbers[ mid ] );
  if ( numbers[ mid ] == key ) {
    next( "MATCH!" );
    return mid;
  }
  else if ( numbers[ mid ] > key ) {
    next( "SEARCH LEFT!" );
    return find( numbers, key, start, mid );
  }
  else {
    next( "SEARCH RIGHT!" );
    return find( numbers, key, mid + 1, end );
  }
}
```

Print one line and print the array

# Binary Search Code (find method, full)

```java
public static int find( int[] numbers, int key, int start, int end ) {
  System.out.println();
  printArray( numbers, start, end );
  if ( start >= end ) {
    next( "SEARCH RANGE EMPTY" );
    return - 1;
  }
  int mid = ( start + end ) / 2;
  System.out.printf( "MID=%d,value=%d, ", mid, numbers[ mid ] );
  if ( numbers[ mid ] == key ) {
    next( "MATCH!" );
    return mid;
  }
  else if ( numbers[ mid ] > key ) {
    next( "SEARCH LEFT!" );
    return find( numbers, key, start, mid );
  }
  else {
    next( "SEARCH RIGHT!" );
    return find( numbers, key, mid + 1, end );
  }
}
```

BASE CASE: If the range has size 0, key was not found. Return $-1$

# Binary Search Code (find method, full)

```java
public static int find( int[] numbers, int key, int start, int end ) {
  System.out.println();
  printArray( numbers, start, end );
  if ( start >= end ) {
    next( "SEARCH RANGE EMPTY" );
    return - 1;
  }
  int mid = ( start + end ) / 2;
  System.out.printf( "MID=%d,value=%d, ", mid, numbers[ mid ] );
  if ( numbers[ mid ] == key ) {
    next( "MATCH!" );
    return mid;
  }
  else if ( numbers[ mid ] > key ) {
    next( "SEARCH LEFT!" );
    return find( numbers, key, start, mid );
  }
  else {
    next( "SEARCH RIGHT!" );
    return find( numbers, key, mid + 1, end );
  }
}
```

Choose the middle position. Print the position and the value.

# Binary Search Code (find method, full)

```java
  public static int find( int[] numbers, int key, int start, int end ) {
    System.out.println();
    printArray( numbers, start, end );
    if ( start >= end ) {
      next( "SEARCH RANGE EMPTY" );
      return - 1;
    }
    int mid = ( start + end ) / 2;
    System.out.printf( "MID=%d,value=%d, ", mid, numbers[ mid ] );
    if ( numbers[ mid ] == key ) {
      next( "MATCH!" );
      return mid;
    }
    else if ( numbers[ mid ] > key ) {
      next( "SEARCH LEFT!" );
      return find( numbers, key, start, mid );
    }
    else {
      next( "SEARCH RIGHT!" );
      return find( numbers, key, mid + 1, end );
    }
  }
```

If the key matches, return the position.

# Binary Search Code (find method, full)

```
public static int find( int[] numbers, int key, int start, int end ) {
  System.out.println();
  printArray( numbers, start, end );
  if ( start >= end ) {
    next( "SEARCH RANGE EMPTY" );
    return - 1;
  }
  int mid = ( start + end ) / 2;
  System.out.printf( "MID=%d,value=%d, ", mid, numbers[ mid ] );
  if ( numbers[ mid ] == key ) {
    next( "MATCH!" );
    return mid;
  }
  else if ( numbers[ mid ] > key ) {
    next( "SEARCH LEFT!" );
    return find( numbers, key, start, mid );
  }
  else {
    next( "SEARCH RIGHT!" );
    return find( numbers, key, mid + 1, end );
  }
}
```

If the key is smaller, choose left.

# Binary Search Code (find method, full)

```java
public static int find( int[] numbers, int key, int start, int end ) {
  System.out.println();
  printArray( numbers, start, end );
  if ( start >= end ) {
    next( "SEARCH RANGE EMPTY" );
    return - 1;
  }
  int mid = ( start + end ) / 2;
  System.out.printf( "MID=%d,value=%d, ", mid, numbers[ mid ] );
  if ( numbers[ mid ] == key ) {
    next( "MATCH!" );
    return mid;
  }
  else if ( numbers[ mid ] > key ) {
    next( "SEARCH LEFT!" );
    return find( numbers, key, start, mid );
  }
  else {
    next( "SEARCH RIGHT!" );
    return find( numbers, key, mid + 1, end );
  }
}
```

If the key is larger, choose right.

# Binary Search Code (main part1)

```
public static final int MAXIMUM = 1000;

public static void main( String[] args ) {
  Scanner console = new Scanner( System.in );

  System.out.print( "Enter size: " );
  int size = console.nextInt();

  int[] numbers = new int[ size ];
  for ( int index = 0; index < size; index ++ ) {
    numbers[ index ] = (int)( Math.random() * MAXIMUM );
  }
  Arrays.sort( numbers );

  printArray( numbers, 0, numbers.length );
```

The maximum value for the entries.

# Binary Search Code (main part1)

```
public static final int MAXIMUM = 1000;

public static void main( String[] args ) {
  Scanner console = new Scanner( System.in );

  System.out.print( "Enter size: " );
  int size = console.nextInt();

  int[] numbers = new int[ size ];
  for ( int index = 0; index < size; index ++ ) {
    numbers[ index ] = (int)( Math.random() * MAXIMUM );
  }
  Arrays.sort( numbers );

  printArray( numbers, 0, numbers.length );
```

Have the user select the array size.

# Binary Search Code (main part1)

```
public static final int MAXIMUM = 1000;

public static void main( String[] args ) {
  Scanner console = new Scanner( System.in );

  System.out.print( "Enter size: " );
  int size = console.nextInt();

  int[] numbers = new int[ size ];
  for ( int index = 0; index < size; index ++ ) {
    numbers[ index ] = (int)( Math.random() * MAXIMUM );
  }
  Arrays.sort( numbers );

  printArray( numbers, 0, numbers.length );
```

Randomly choose elements in the array. Then sort.

# Binary Search Code (main part1)

```
public static final int MAXIMUM = 1000;

public static void main( String[] args ) {
  Scanner console = new Scanner( System.in );

  System.out.print( "Enter size: " );
  int size = console.nextInt();

  int[] numbers = new int[ size ];
  for ( int index = 0; index < size; index ++ ) {
    numbers[ index ] = (int)( Math.random() * MAXIMUM );
  }
  Arrays.sort( numbers );

  printArray( numbers, 0, numbers.length );
```

Print the array.

# Binary Search Code (main part2)

```
    int key = 0;
    while ( key >= 0 ) {
      System.out.print( "Enter key to search (negative to quit): " );
      key = console.nextInt();
      int val = find( numbers, key );
      if ( val < 0 ) {
        System.out.println( key + " was not found" );
      }
      else {
        System.out.println( key + " was found at " + val );
      }
    }
  }
```

Receive key to search from the user.

# Binary Search Code (main part2)

```
    int key = 0;
    while ( key >= 0 ) {
      System.out.print( "Enter key to search (negative to quit): " );
      key = console.nextInt();
      int val = find( numbers, key );
      if ( val < 0 ) {
        System.out.println( key + " was not found" );
      }
      else {
        System.out.println( key + " was found at " + val );
      }
    }
  }
```

- Execute search and receive the result.

# Binary Search Code (main part2)

```
    int key = 0;
    while ( key >= 0 ) {
      System.out.print( "Enter key to search (negative to quit): " );
      key = console.nextInt();
      int val = find( numbers, key );
      if ( val < 0 ) {
        System.out.println( key + " was not found" );
      }
      else {
        System.out.println( key + " was found at " + val );
      }
    }
  }
```

- Report the result by examining the value returned.

# The End