Recursion

- Reading: Savitch, Chapter 11

Objectives

- To introduce the concept of recursion
- To understand how recursive methods work
- To compare recursion with iteration

Recursion

- The phenomenon that a method calls itself is named as recursion.

Example

```
//Selection Sort
public static void sort (int a[], int n, int start)
{
    int min = index_of_minimum_element (a, start);
    swap (a[start], a[min]);
    sort (a, n, start +1);
}
```

Recursion

- Common problem solving method
- Applicable when the problem solution can be defined in terms of smaller problems of exactly the same type
- Example - printing a string backwards

```
method reverse(s, n)
   print the last character of s
   reverse(s, n-1)
```

Recursion

- A recursive algorithm contains two parts

  (1) a recursive step

  (2) a terminating case

Examples of recursion

- Factorial

  n! = 1 * 2 * ... * (n-1) * n

  the recursive step:

  n! = (n-1)! * n

  the terminating case:

  1! = 1

Examples of recursion

- Power

  $a^n = a * a * ... * a$ (totally n times)

  the recursive step:

  $a^n = a^{n-1}\ a$

  Or: power(a,n) = power(a,n-1) * a

  the terminating case:

  power(a, 0) = 1

Writing recursive algorithms

- Express the solution to the problem in terms of solutions to one or more smaller problems of exactly the same type
- Always remember to include a terminating case

Class exercise

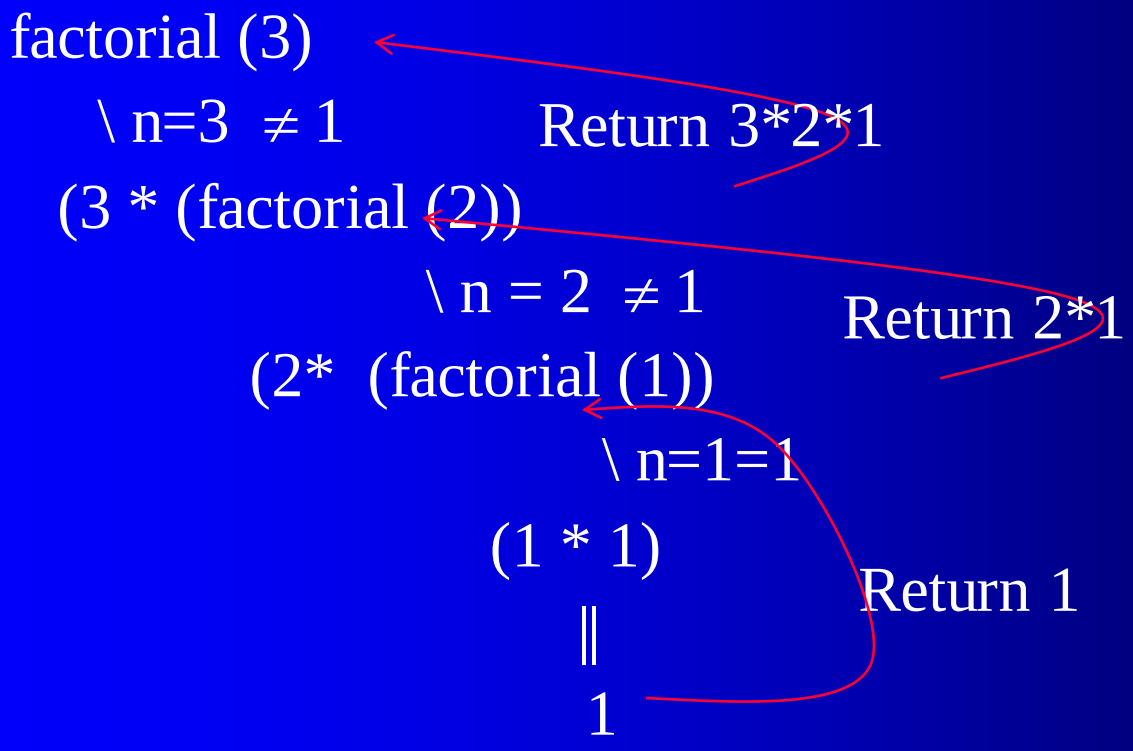
- Write a recursive version of the power and the factorial methods
- Remember to include a terminating case
- Reminder: power(a,n) = power(a,n-1) * a

and factorial(n) = factorial(n-1) * n

```
//Power.java
public class Power {
   public static void main(String[] args) {
    System.out.println(power(6, 4));
   }
   public static double power(double x, int n) {
          if (n == 0)
                 return 1;
          else
                 return power(x, n -1) * x;
   }
}
```

```
//Factorial.java
public class Factorial {
  public static void main(String[] args) {
    System.out.println(factorial (3));
   }
   public static int factorial(int n) {
    if (n == 1)
          return 1;
    else
          return factorial(n-1) * n;
   }
}
```

How recursion works



The call of factorial (3) cannot have a value until the value of factorial (2) is back, factorial (2) will wait for factorial(1) …

How recursion works

- JAVA String reversal method

```
public static void reverse(String s, int n) {
    if (n != 0) {
        System.out.print(s.charAt(n-1));
        reverse(s,n-1);
    }
}
```

How recursion works (ctd)

- Using the reverse method

```
public static void main(String [] args)
{
    String str = “bin”;
    System.out.print("The reverse of ‘“ + str + “’ is “);
    reverse (str, str.length());
}
```

How recursion works (ctd)

- Using the reverse method

```
public static void main(String [] args)
{
    String str = “bin”;
    System.out.print("The reverse of ‘“ + str + “’ is “);
    reverse (str, str.length());
}
```

Output:

How recursion works (ctd)

- Using the reverse method

```
public static void main(String [] args)
{
    String str = “bin”;
    System.out.print("The reverse of “‘ + str + “’ is “);
    reverse (str, str.length());
}
```

Output:

How recursion works (ctd)

- Using the reverse method

```
public static void main(String [] args)
{
    String str = “bin”;
    System.out.print("The reverse of “ + str + “ is “);
    reverse (str, str.length());
}
```

Output:

How recursion works (ctd)

- Using the reverse method

```
public static void main(String [] args)
{
    String str = “bin”;
    System.out.print("The reverse of ‘“ + str + “’ is “);
    reverse (str, str.length());
}
```

Output:

```
The reverse of ‘bin’ is ‘
```

How recursion works (ctd)

- Using the reverse method

```
public static void main(String [] args)
{
    String str = “bin”;
    System.out.print("The reverse of ‘“ + str + “’ is “);
    reverse (str, str.length());
}
```

Output:

```
The reverse of ‘bin’ is
```

How recursion works (ctd)

- Tracing the call to reverse

```
// s = “bin”, n = 3

if (n != 0)
{
    System.out.print(s.charAt(n-1));
    reverse(s, n-1);
}
```

How recursion works (ctd)

- Tracing the call to reverse

```
// s = “bin”, n = 3

if (n != 0)
{
   System.out.print(s.charAt(n-1));
   reverse(s,n-1);
}
```

Output

How recursion works (ctd)

- Tracing the call to reverse

```
// s = “bin”, n = 3
if (n != 0)
{
    System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

How recursion works (ctd)

- Tracing the call to reverse

```
// s = “bin”, n = 3

if (n != 0)
{
   System.out.print(s.charAt(n-1));
   reverse(s,n-1);
}
```

Output

How recursion works (ctd)

- Tracing the call to reverse

```
// s = “bin”, n = 3

if (n != 0)
{
    System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

n

How recursion works (ctd)

- Tracing the call to reverse

```
// s = “bin”, n = 3

if (n != 0)
{
   System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

n

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
```

```
// s = "bin",  n = 2
if (n != 0)
{
   System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

n

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
    // s = "bin",  n = 2
    if (n != 0)
    {
       System.out.print(s.charAt(n-1));
        reverse(s,n-1);
    }
```

Output

n

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
```

```
// s = "bin",  n = 2
if (n != 0)
{
    System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

n

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
```

```
// s = "bin",  n = 2
if (n != 0)
{
   System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

ni

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
```

```
// s = "bin",  n = 2
if (n != 0)
{
   System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

ni

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
```

```
// s = "bin",  n = 2
if (
{
}
```

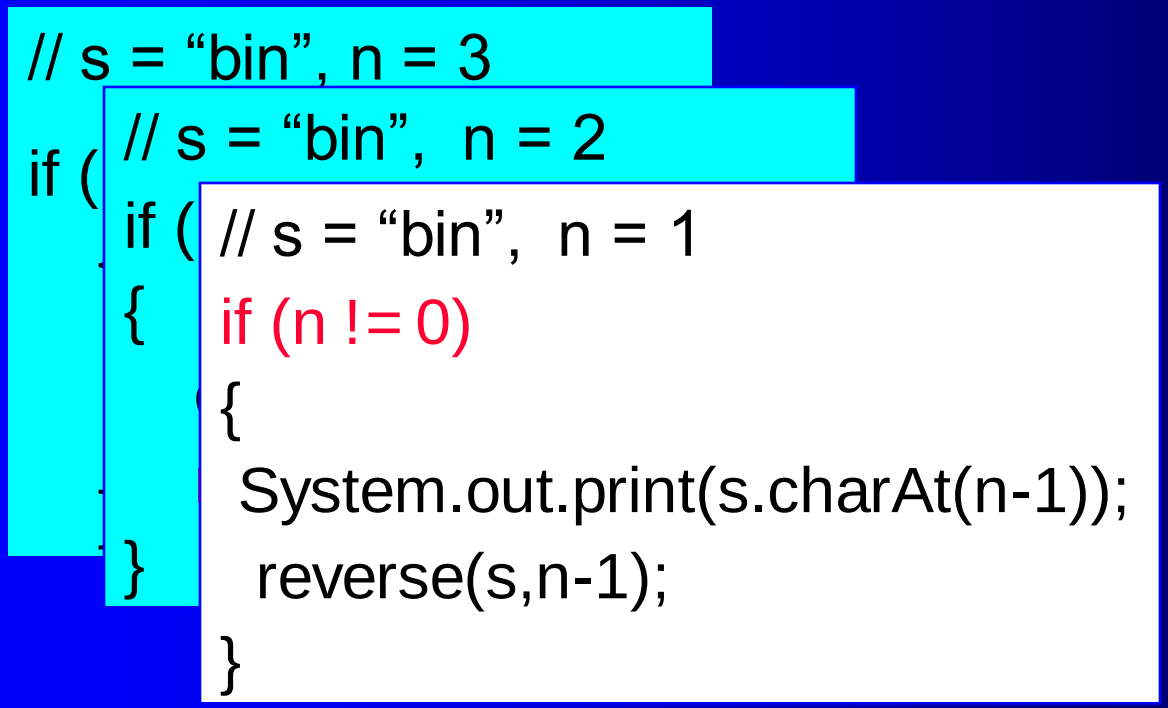
```
// s = "bin",  n = 1
if (n != 0)
{
 System.out.print(s.charAt(n-1));
  reverse(s,n-1);
}
```

Output

ni

How recursion works (ctd)

- Tracing the call to reverse



Output

ni

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
    // s = "bin",  n = 2
    if (
    {
        // s = "bin",  n = 1
        if (n != 0)
        {
         System.out.print(s.charAt(n-1));
          reverse(s,n-1);
        }
    }
```

Output

ni

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
    // s = "bin",  n = 2
    if (
    {
        // s = "bin",  n = 1
        if (n != 0)
        {
        System.out.print(s.charAt(n-1));
        reverse(s,n-1);
        }
    }
```

Output

nib

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
```

```
// s = "bin",  n = 2
if (
{
}
```

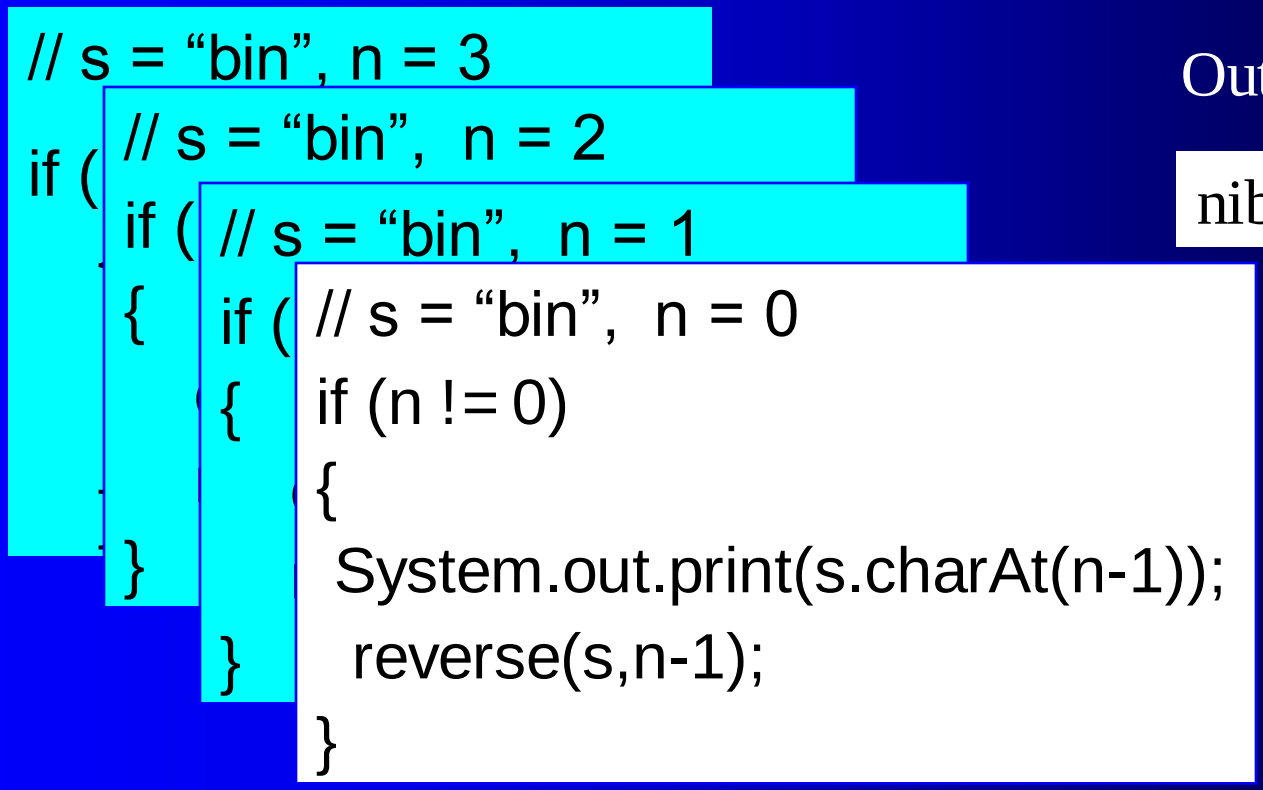
```
// s = "bin",  n = 1
if (n != 0)
{
 System.out.print(s.charAt(n-1));
 reverse(s,n-1);
}
```

Output

nib

How recursion works (ctd)

- Tracing the call to reverse

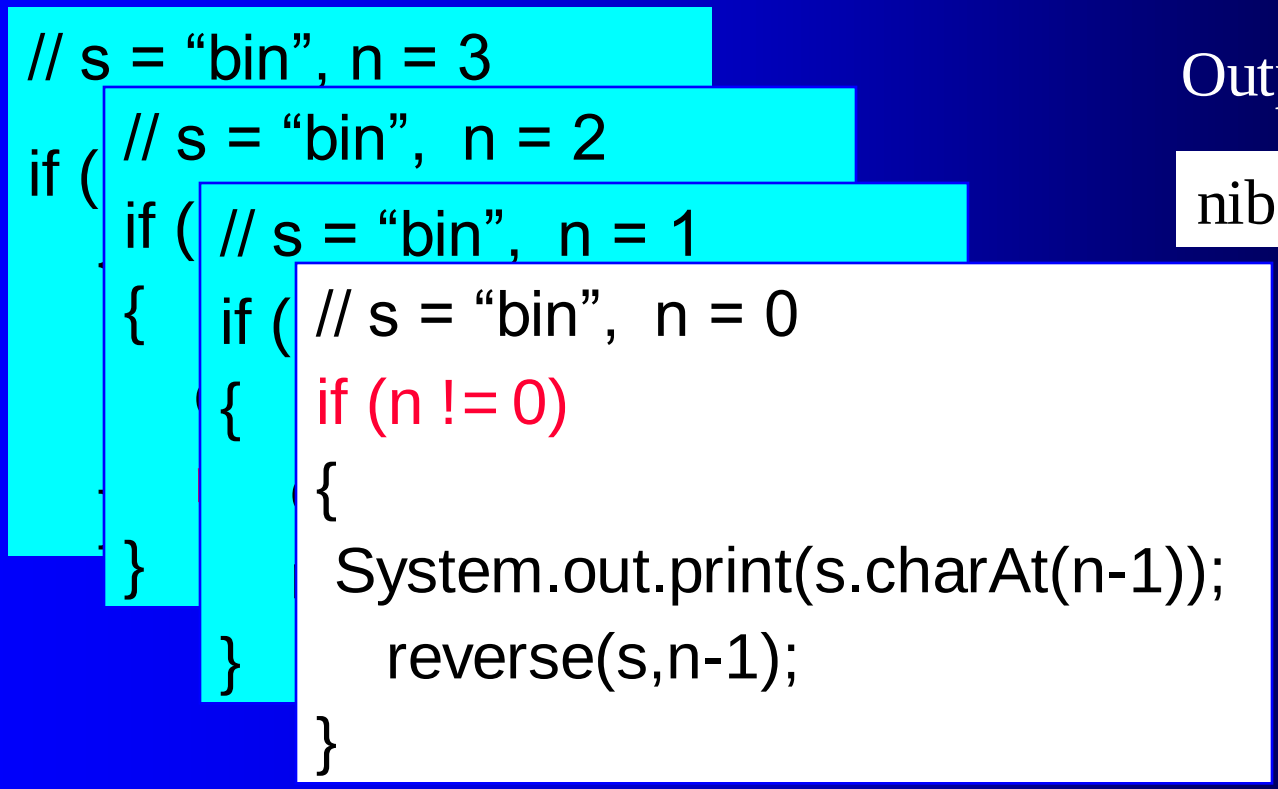


Output

nib

How recursion works (ctd)

- Tracing the call to reverse

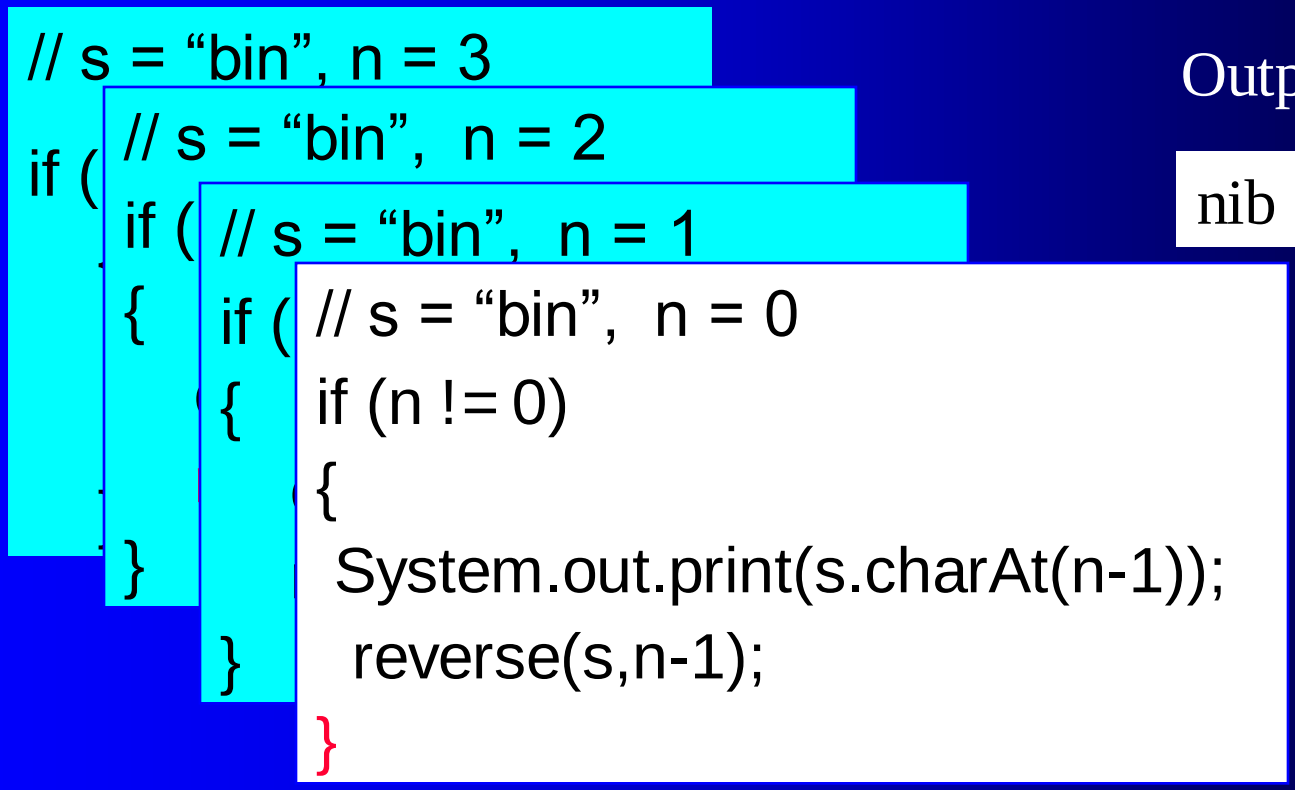


Output

nib

How recursion works (ctd)

- Tracing the call to reverse



Output

nib

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
```

```
// s = "bin",  n = 2
if (
{
}
```

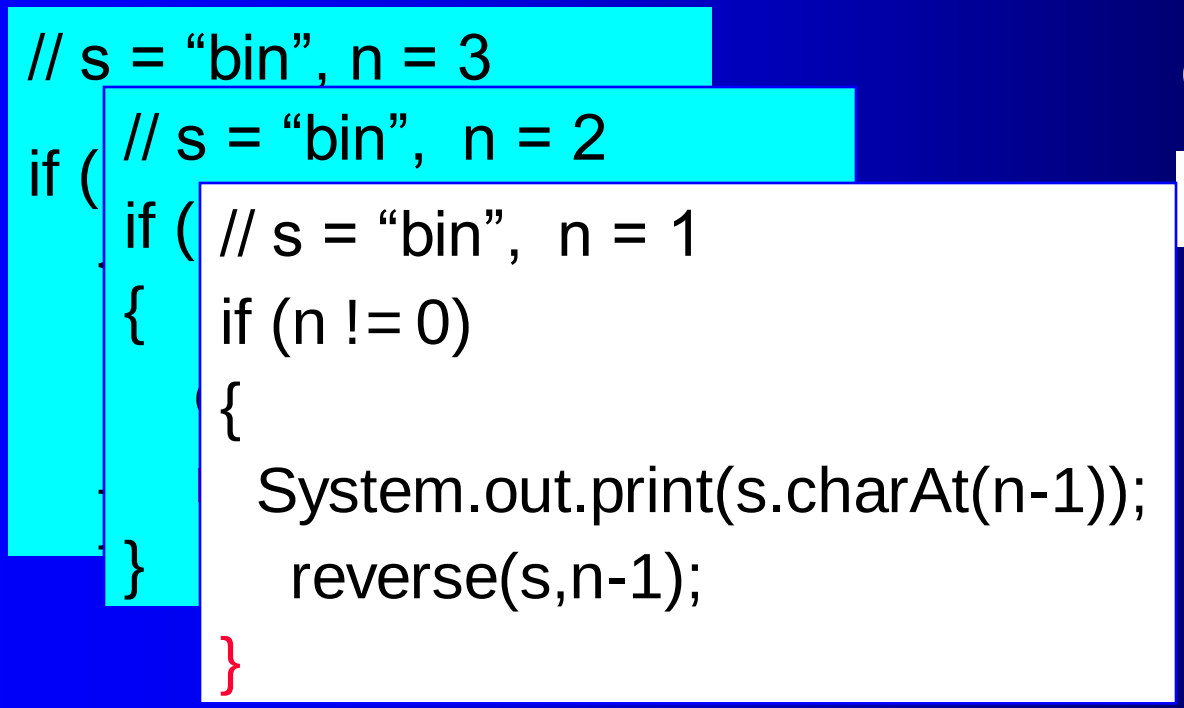
```
// s = "bin",  n = 1
if (n != 0)
{
 System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

nib

How recursion works (ctd)

- Tracing the call to reverse



Output

nib

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
```

```
// s = "bin",  n = 2
if (n != 0)
{
   System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

nib

How recursion works (ctd)

- Tracing the call to reverse

```
// s = "bin", n = 3
if (
```

```
// s = "bin",  n = 2
if (n != 0)
{
   System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

nib

How recursion works (ctd)

- Tracing the call to reverse

```
// s = “bin”, n = 3

if (n != 0)
{
    System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

nib

How recursion works (ctd)

- Tracing the call to reverse

```
// s = “bin”, n = 3

if (n != 0)
{
    System.out.print(s.charAt(n-1));
    reverse(s,n-1);
}
```

Output

nib

How recursion works (ctd)

- Using the reverse method

```
public static void main(String[] args)
{
    String str = “bin”;
    System.out.print("The reverse of ‘“ + str + “’ is “);
    reverse (str, str.length());
}
```

Output:

```
The reverse of ‘bin’ is nib
```

Class exercise - do as homework

- Determine the purpose of the following recursive method by tracing a call to it with the arguments s = “bin” and n = 3.

```
public static void mystery(String s, int n) {
    if (n == 1)
        System.out.println(s.charAt(0));
    else {
        mystery (s, n-1);
        System.out.println(s.charAt(n-1));
    }
}
```

Recursion v iteration

- Recursive factorial method

```
public static int factorial(int n) {
        if (n == 1)
                return 1;
        else
                return n*factorial(n-1);
  }
```

Recursion v iteration (ctd)

- Iterative factorial method

```
public static int factorial(int n) {
    // assume n >= 0
    int product = 1;
    for (int i = 2; i <= n; i++) {
        product = i*product;
    }
    return product;
}
```

Iteration v recursion (ctd)

- Iterative programs usually
  - run faster
  - use less storage
- Recursion often provides
  - simpler, more elegant solutions
  - code that is easier to understand