COMPSCI 105

Principles of Computer Science

Recursion

Recursion

Recursion **(or self-reference) is the concept of defining a data structure or an algorithm in terms of itself.**

Data structure example:

- A list is either empty, or it consists of an element followed by a list.

Algorithm example - summing elements from 1 to n:

- If n is 1, then the answer is 1
- Otherwise, calculate the sum of the elements from 1 to n-1, and then add n.

Recursive definitions

Definitions should have

- Base case
- Recursive case

Examples

- Definition of natural numbers
  - base case: 1 is a natural number
  - recursive case: i is a natural number, if i-1 is a natural number
- Sum of the first n natural numbers
  - base case: sum(0) = 0
  - recursive case: sum(n) = n + sum(n-1)
- Calculating Factorial
  - base case: 0! = 1
  - recursive case: n! = n × (n-1)!

Calculating the sum from 0 .. n

Start with a non-recursive method

- We can test this method thoroughly
- We know that it works for values of n >= 0

```
public int sum(int n) {
	int result = 0;
	for (int i = 0; i <= n; i++) {
		result += i;
	}
	return result;
}
```

Calculating the sum from 0 .. n

Another non-recursive method

- Makes use of the previous method
- Simple and works

```
public int sum2(int n) {
	return n + sum(n-1);
}
```

Two methods that calculate the sum

- sum(int)
- sum2(int)

Recursive methods

Replace the call to sum() with a call to sum2()

```
public int sum2(int n) {
	return n + sum2(n-1);
}
```

Need to have a base case

```
public int sum2(int n) {
	if (n == 0)
		return 0;
	else
		return n + sum2(n-1);
}
```


Volunteers required

You will be given a pile of cards

If there is only one card in the pile then

- give it back

If there is more than one card in the pile then

- take the top card
- pass the rest of the cards to another person
- wait to be given a card back
- when you get a card back, compare it with the one you already have
- give the biggest card back to the person that passed you the pile

Code to find the maximum

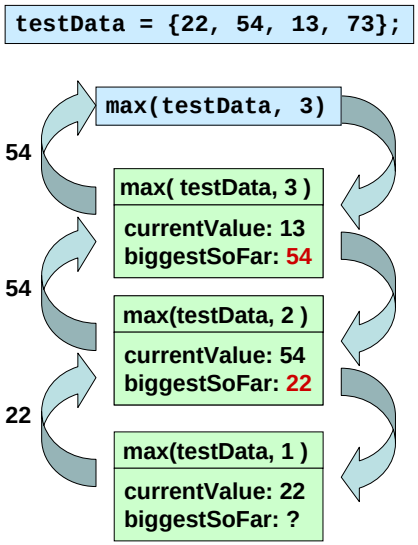
```
//data is an array containing a lot of int values
//n is the number of elements in the data set that we are
finding the maximum of.

public int max(int[] data, int n) {
	int currentValue = data[n-1];
	if (n <= 1)
		return currentValue;
	else {
		int biggestSoFar = max(data, n-1);
		return Math.max(currentValue, biggestSoFar);
	}
}
```

```
int[] testData = {22, 54, 13, 73};
System.out.println( max(testData, 3) );
```

Code trace

```
public int max(int[] data, int n) {
  int currentValue = data[n-1];
  if (n <= 1)
    return currentValue;
  else {
    int biggestSoFar = max(data, n-1);
    return Math.max( currentValue,
                     biggestSoFar  );
  }
}
```



Common mistakes

BAD recursion

- What is this missing?

```
public void bad() {
       System.out.println("very bad");
       bad();
}
```


Common mistakes

BAD recursion

- What is wrong here?

```
public int sum(int n) {
       if (n == 0)
              return 0;
       else
              return sum(n+1) – (n+1);
}
```


Summary

Recursion must:

- have a base case
- make progress towards that base case on each recursive step (by tackling a problem of a smaller size), so that it is guaranteed to reach the base case eventually

Activity

Remove all the diamonds from a pack of cards

- You will be passed some cards
- Remember who gave them to you

Base case:

- If you receive one card, then check the suit
- If it is a diamond, then give back nothing
- If it is any other suit, then give back the card

Recursive case:

- If you receive more than one card
- Split the cards into two groups
- Pass each group to someone else to remove the diamonds
- When you receive back the cards from two people, join them together and pass them back to the person that gave them to you

Code Example

Factorial definition:

- 0! = 1
- n! = n * (n-1) * (n-2) * (n-3) * ... * 3 * 2 * 1

Factorial definition:

- Base case: factorial(0) = 1
- Recursive case: factorial(n) = n * factorial(n-1)

Recursive Factorial

Code follows from the recursive definition:

```
public int factorial(int n) {
	if (n == 0)
		return 1;
	else
		return n * factorial(n-1);
}
```


Code Example

Printing all the integer values from n to 0

Base case:

- When n is 0, just print out 0

Recursive case:

- print out n, then print all the integer values from n-1 to 0

```
public void print(int n) {
	if (n == 0)
		System.out.println(0);
	else {
		System.out.println(n);
		print(n-1);
	}
}
```

Exercise

Given the following recursive method:

```
public void print(int n) {
	if (n == 0)
		System.out.println(0);
	else {
		System.out.println(n);
		print(n-1);
	}
}
```

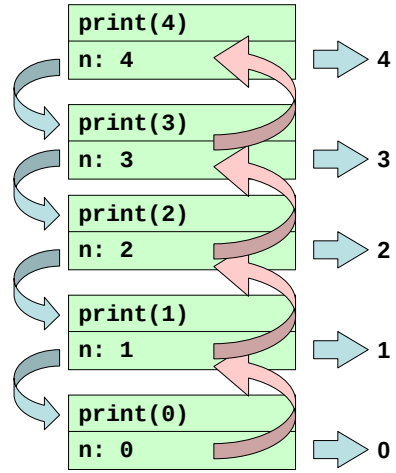
What is the output that results when the method is called using:

```
print(4);
```


Exercise - solution

Given the following recursive method:

```
public void print(int n) {
    if (n == 0)
        System.out.println(0);
    else {
        System.out.println(n);
        print(n-1);
    }
}
```



Thinking recursively

- **Four questions for constructing recursive solutions**
  - Can you define the problem in terms of a smaller problem of the same type?
  - Does each recursive call diminish the size of the problem?
  - What instance of the problem can serve as the base case?
  - As the problem size diminishes, will you reach this base case?

Fibonacci Numbers

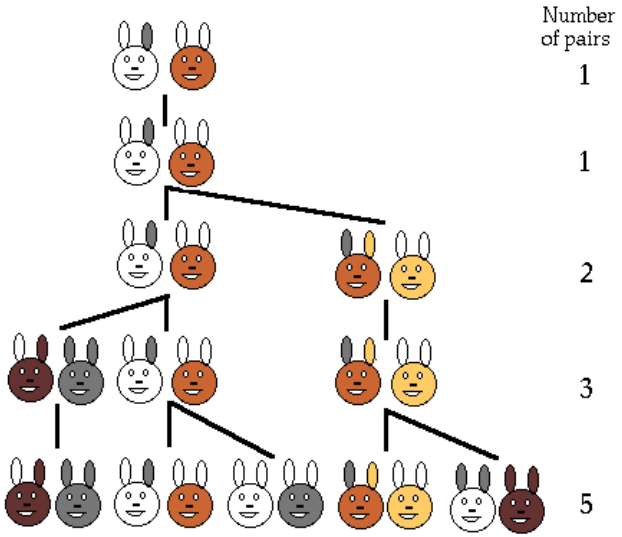
Leonard Fibonacci

- One of the most talented mathematicians of the middle ages
- 1202

Rabbits

- *Suppose a newly-born pair of rabbits, one male, one female, are put in a field.*
- *Rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits.*
- *Suppose that our rabbits never die and that the female always produces one new pair (one male, one female) every month from the second month on.*

Rabbit population growth



Fibonacci Sequence

Sequence of Fibonacci numbers

- 1 1 2 3 5 8 13 21 34 55 89 144 233 ...
- Each number is the sum of the previous two numbers

Base case:

- fibonacci(1) = 1
- fibonacci(2) = 1

Recursive case

- fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)

http://en.wikipedia.org/wiki/Golden_ratio

http://en.wikipedia.org/wiki/Fibonacci_number

Recursive method

Follows from the definition

```
public int fibonacci(int n) {
      if (n <= 2)
            return 1;
      else
            return fibonacci(n-1) + fibonacci(n-2);
}
```


Exercise

Write a recursive method that returns a String in reverse. For example, the method call:

```
reverse("Hello World");
```

would return the value:

```
dlroW olleH
```

Think recursively

- Can you define the problem in terms of a smaller problem of the same type?
- Does each recursive call diminish the size of the problem?
- What instance of the problem can serve as the base case?
- As the problem size diminishes, will you reach this base case? -

Exercise - solution

Think recursively

- Print out the last letter, then print out the rest of the String in reverse
- Each recursive call is smaller (since the string is smaller)
- A string of length 1 is the base case
- We will reach the base case since the string gets smaller each time

```
public static String reverse(String s) {
    if (s.length() == 1)
       return s;
    else {
       String last = "" + s.charAt(s.length()-1);
       String rest = reverse(s.substring(0, s.length()-1));
       return last + rest;
    }
}
```


Exercise - solution2

Think recursively

- Remove the first letter, reverse the rest of the String and then print the first letter.
- Each recursive call is smaller (since the string is smaller)
- A string of length 1 is the base case
- We will reach the base case since the string gets smaller each time

```
public static String reverseString(String s) {
    if (s.length() == 1)
       return s;
    else {
       String first = "" + s.charAt(0);
       String rest = reverseString(s.substring(1));
       return rest + first;
    }
}
```


Exercise

We define the concept of a *foo* number as follows.

- 0 is a foo number.
- If *i* is a foo number then *i*+5, and *i*+7 are also foo numbers.

Write a recursive method which will test if a given integer is a foo number.

```
public boolean isFoo(int n) {




}
```