Lec 24

Recursion
[3142, 5798, 6550, 8914]
[3142, 5738, 6550, 8914]
[3, 14, 2, 5, 7, 3, 6, 0, 5, 3, 0, 8, 9, 1, 4]
How would you code this?

- Method that returns TRUE if any element in the array is odd
public boolean anyOdd(int[] array) {
    for(int i = 0; i < array.length; i++) {
        if(array[i] % 2 == 1) return true;
    }
    return false;
}
public void hello(int x) {
    System.out.println("hi "+x);
    hello(x+1);
}

A) Compiler error
B) hi 0
C) hi 0, hi 1, hi 2, ... hi <forever>
D) hi 0, hi 1, ... hi <some big number>, until crash
E) - None of the above
Recursion: Step 1

• Solve one step, one part of the problem
  – e.g. is the first number in the array even

• Leave rest of the problem for future steps
  – e.g. the rest of the array
Get to a smaller problem

```java
public boolean anyEven(int[] array) {
    if (array[0] % 2 == 0) return true;
    else {
        int[] smaller = Arrays.copyOfRange(array, 1, array.length);
        return anyEven(smaller);
    }
}
```
5 * 4!
5 * 4 * 3!
5 * 4 * 3 * 2!

5 * 4 * 3 * 2 * 1!
5 * 4 * 3 * 2 * 1 * 0!
5 * 4 * 3 * 2 * 1 * 1
Recursion: Step 2

• Know when to stop! Known as the base case
  – When the array only has one element
  – $0!$
What are the missing lines to calculate \( n! \)?

```java
public int fact(int x) {
    if (x <= 0)
        ______________;
    else
        ______________;
}
```

```java
int answer = fact(5);
System.out.println(answer);
```

A) return 0;
   fact(x);
B) return 1;
   return x * fact(x-1);
C) return 1;
   return (x-1) * fact(x-1);
D) return 0;
   return x * fact(x-1);
E) None of the above
3 parts to recursion

• Know when to stop (called the base case).
• Decide how to take one step.
• Break the journey down into that step plus a smaller journey.
The trickiest code tracing question yet: What does this code do? (when we call it with the number 5)

```java
public static void foo(int x)
{
    if (x>1)
        foo(x-1);
    System.out.println(x);
}
```

A) Prints out 5
B) Prints out the numbers from 5 down to 1
C) Prints out the numbers from 1 up to 5
D) I have NO IDEA! How can you call foo from inside foo!?!?!
public static void foo(int x) {
    if (x>1)
    foo(x-1);
    System.out.println(x);
}
The trickiest code tracing question yet:
What does this code do?
(when we call it with the number 5)

public static void foo(int x)
{
    System.out.println(x);
    if (x>1)
        foo(x-1);
}

A) Prints out 5
B) Prints out the numbers from 5 down to 1
C) Prints out the numbers from 1 up to 5
D) I have NO IDEA! How can you call foo from inside foo?!?!
public static void foo(int x) {
    System.out.println(x);
    if (x>1)
        foo(x-1);
}
Recursion: Why?

• That’s HOW it works.
• Why do you use it?
  – Perfect for problems where there is an obvious answer for some small problem and all larger problems build from smaller problems
• There are iterative (loop based) solutions for every problem solvable with recursion. Use whichever is simpler...
  – Although there may be performance implications of each
Permutations
[vfgoeeraly]

[adr]
public void permute(String perm) {
    if(perm.length() == letters.length)
        System.out.println(perm);
    else {
        for(int i =0; i < letters.length; i++) {
            if(perm.contains(letters[i]))
                continue;
            permute(perm+letters[i]);
        }
    }
}


What happens if we call
char[] letters = {'A','B'};
permute(""');

A) Infinite recursion
B) AA, AB, BA, BB
C) AB, BA
D) AA, AB, BB, BA
E) A, AA, AB, B, BA, BB