Lec 16

Recursion
[3142, 5798, 6550, 8914]
[3142, 5736, 6550, 8914]
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;
}
What gets printed

```java
public void hello(int x) {
    System.out.println("hi " + x);
    hello(x+1);
}
```

What happens if we call `hello(0);`

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
5 * 4!
5 * 4 * 3!
5 \times 4 \times 3 \times 2!
5 * 4 * 3 * 2 * 1!
5 * 4 * 3 * 2 * 1 * 0!
5 * 4 * 3 * 2 * 1 * 1
public int fact(int x) {  
    if(x <= 0)  
        __________;  
    else  
        __________;  
}
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)

```java
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
[vfgoe early]

[adr]
What gets printed

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
Binary search

• You have an array of increasing timestamps
  – <1000232, 1000241, 1001323...>

• And matching array of wall power readings
  – <150.23, 149.44, 180.39, ...>

• Given a timestamp, find the nearest recorded
timestamp/power reading
  – Solve with loops
  – Solve with recursion
public int find(int n, int low, int high) {
    int middle = (low+high)/2;
    if(array[middle] == n)
        return middle; //base case
    if(array[middle] > n)
        return find(n, low, middle);
    else
        return find(n, middle, high);
}