Computer Science 112
Data Structures

Lecture 17:
Heaps
Review of recursion

While Sakai is down

- **slides, java, etc. at**
  http://www.cs.rutgers.edu/~lou/112/s15

Review: Hash Tables

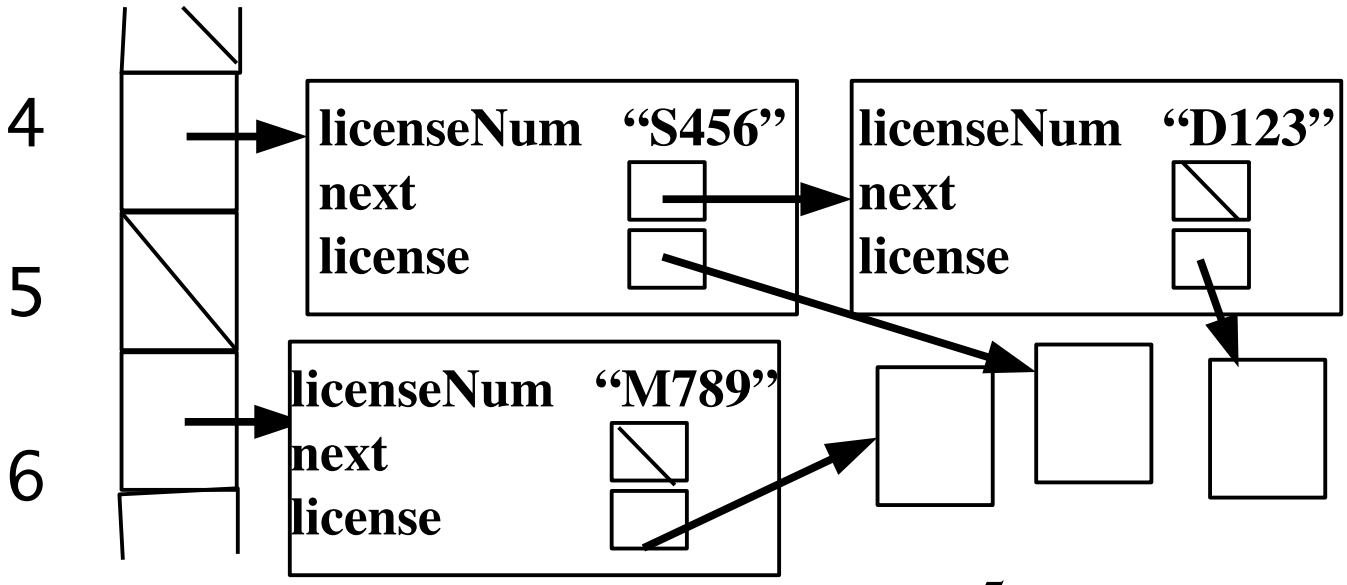
- **implement a mapping keys → values**
- **hash(key) → index into array of linked listst, every index equally likely**

Review: Hash Tables

- **Implement a mapping from keys to values**
- **Operations:**
  - **get(key): return value stored for given key**
  - **put(key, value): add / change value for given key**
  - **remove(key): remove any value stored for key**
- **All in average O(1)**

Hash Function

- A hash function turns a key into an array index
  - equal keys → same hash value
  - every hash value should be equally likely
- Hash value is an index into an array of linked lists



Get(key)

index = hash(key)

search list at table[index] for key

return value from object found

If no object found for key, return null

- **O(L) where L is list length**

put(key, value)

index = hash(key)

search list at table[index] for key

if found, change value

else insert key, value in list

- **O(L) where L is list length**

remove(key)

index = hash(key)

search list at table[index] for key

if found, delete object from list

- **O(L) where L is list length**

Load Factor

- **Two meanings for "load factor"**
  - **current fullness of table:**
    number of entries / size of table
  - **threshold:**

    when current load factor > threshold
    rehash into larger array
- **Average L = load factor**
- **We keep load factor < threshold, so**

  O(L) = O(1)

Built-in Hashing in Java

- **The class java.util.HashMap<K, V>**
  - **Note: generic with two class parameters:**
    - **K: class of keys**
    - **V: class of values**
  - **E.g. NetID => Student**
    java.util.HashMap<NetID, Student>
  - **See JDK API**
  - **See Driver.java, UseDriverMap.java**

New: Priority Queues

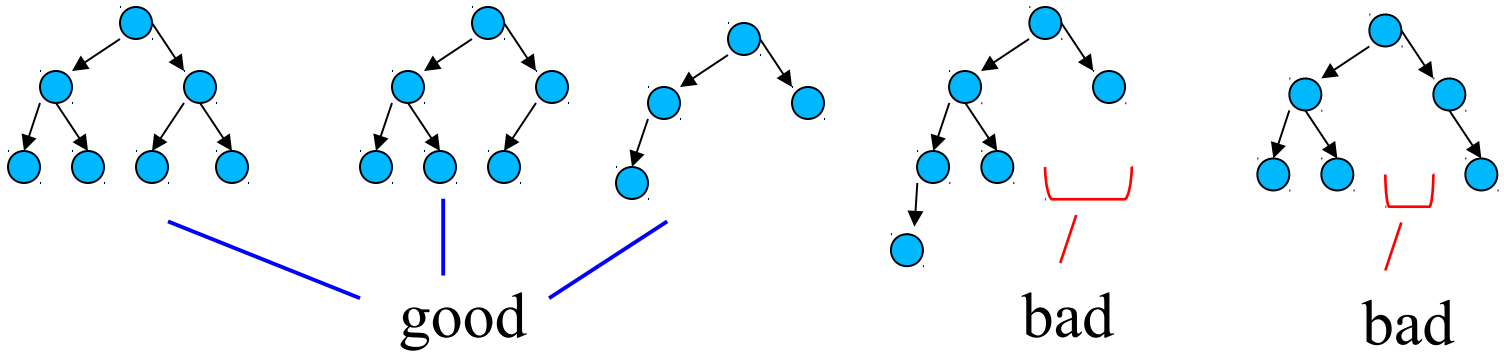
- **Each data item has a priority**
- **Add items to queue in any order**
- **Remove items in priority order**
  - **add A:5, B:3, C:6**
  - **remove C**
  - **add D:8**
  - **remove D, remove A**

Implement as an array

- **Unsorted array:**
  - **Insert one item O(1)**
  - **Remove one item O(n)**
- **Sorted array:**
  - **Insert one item: O(n)**
  - **Remove one item: O(1)**

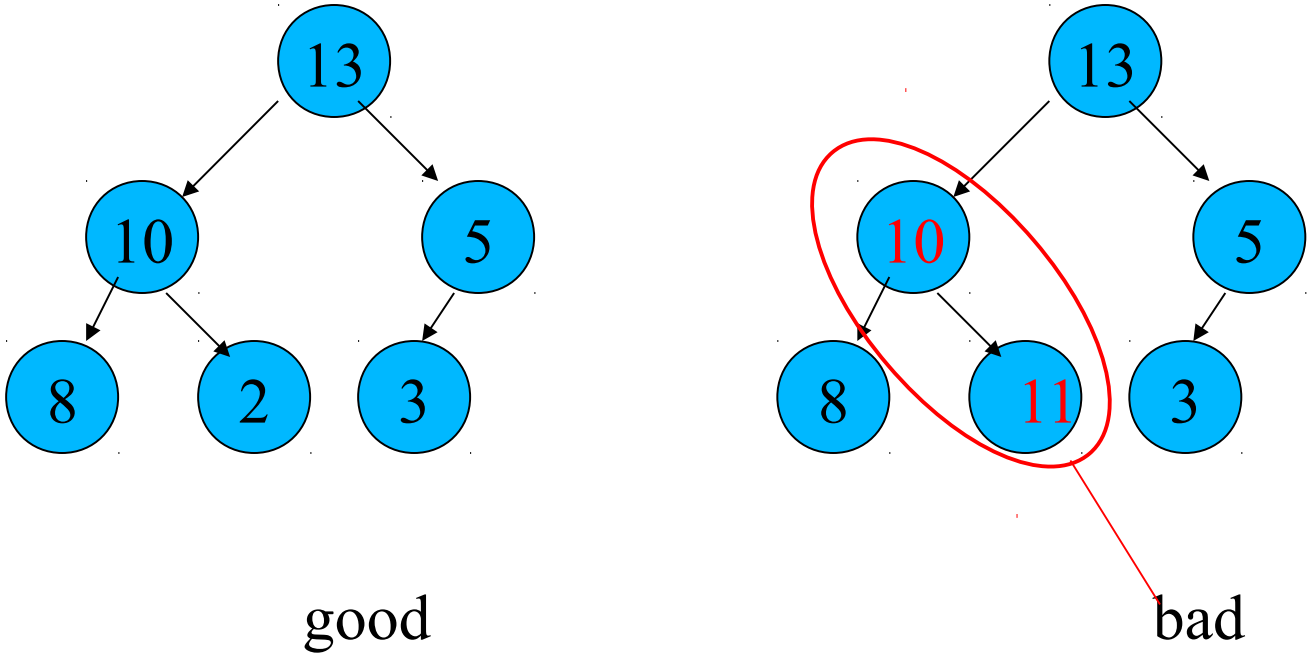
Heap

- **A heap is a way to implement a priority queue with O(log n) complexity**
- **A heap is a complete binary tree**
  - **all levels except maybe the last are full**
  - **last level filled from left to right**



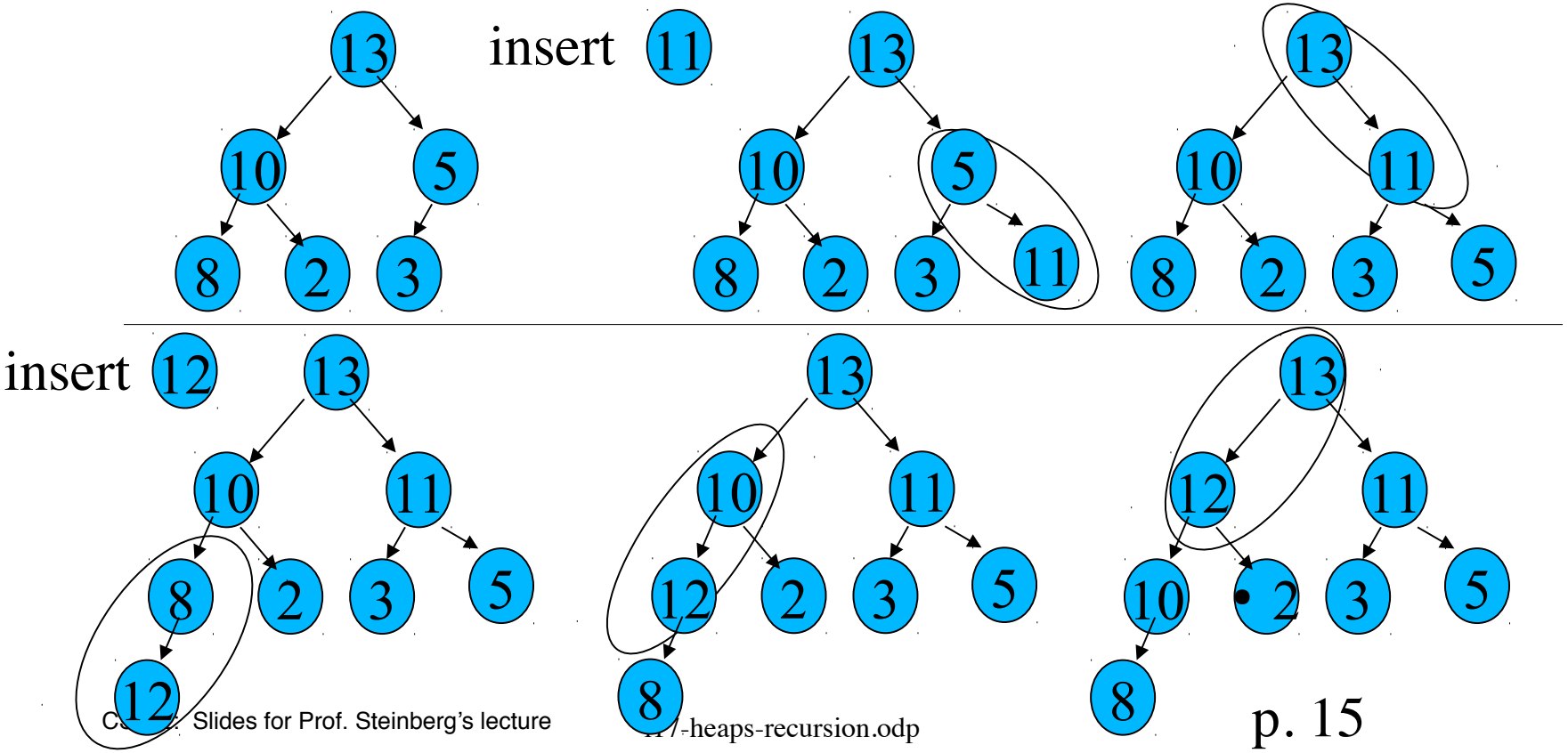
Heap

- **The number at a node is greater than the number at any descendant**



Heap Insert

- **Add node at end of last level**
- **Move up restoring order (*filter up*)**

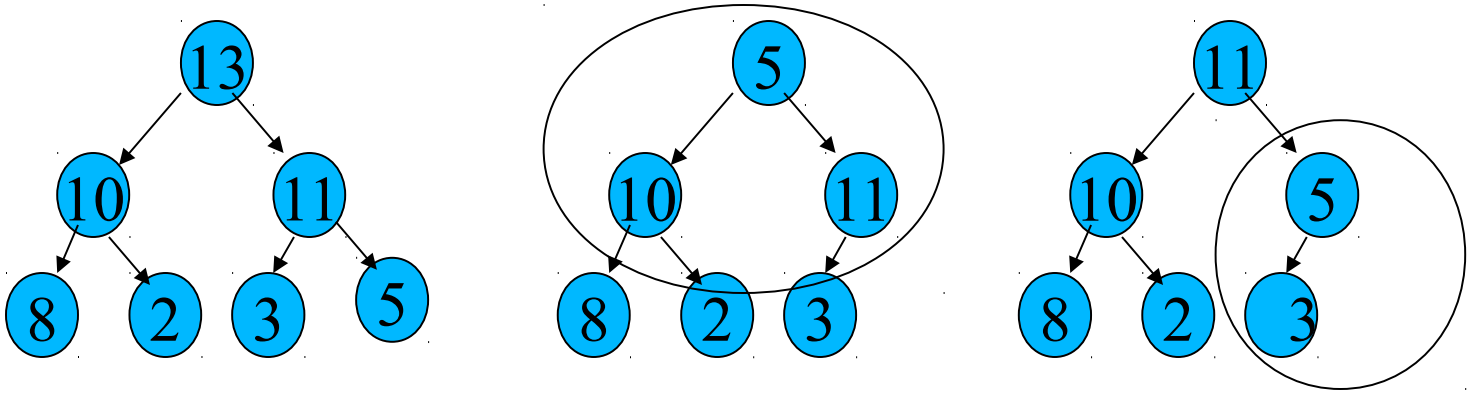


Big O of insertion

- **O(H) where H is height of while tree**
  = O(log(n)) where n is number of nodes

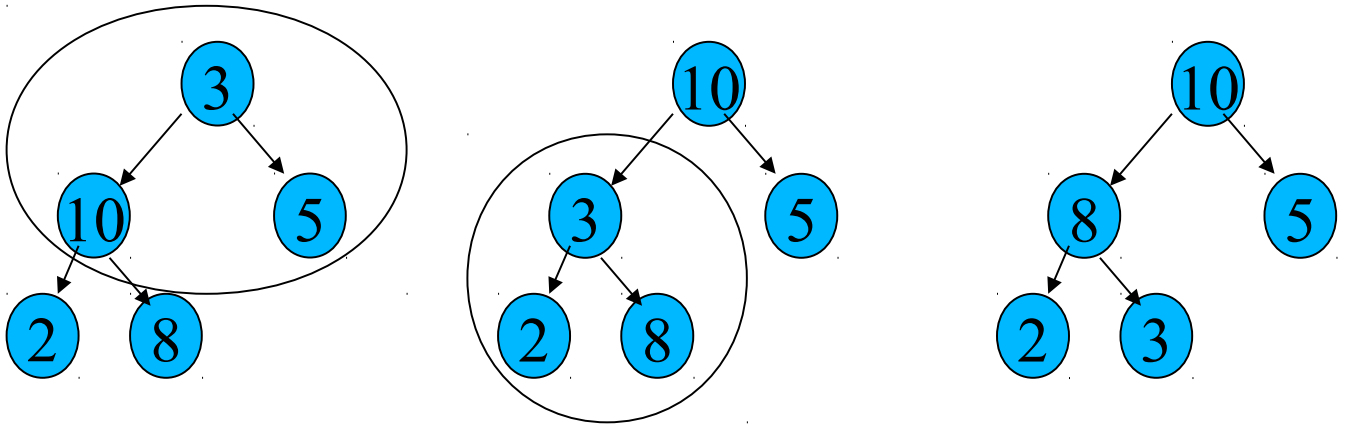
Heap Deletion

- **Copy out data at root**
- **Delete last node on last row & put data in root**
- **Move down restoring order (*filter down*)**



Heap Deletion

- **Compare current node and two children**
  - **if current node largest, stop**
  - **if left child is largest swap current and left**
  - **similar if right child is largest**

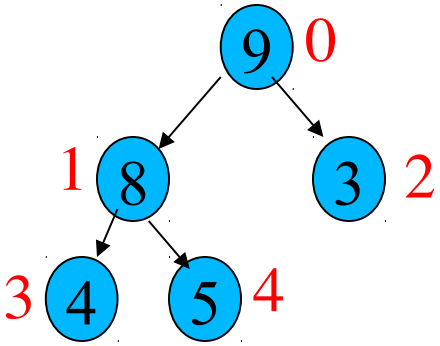


Big O of deletion

- **O(H) where H is height of whole tree**
  = O(log(n)) where n is number of nodes

Heap Representation

- **Store heap in an array**
  - **For node at index j, children are at 2j+1 and 2j+2**
  - **Root at index 0**

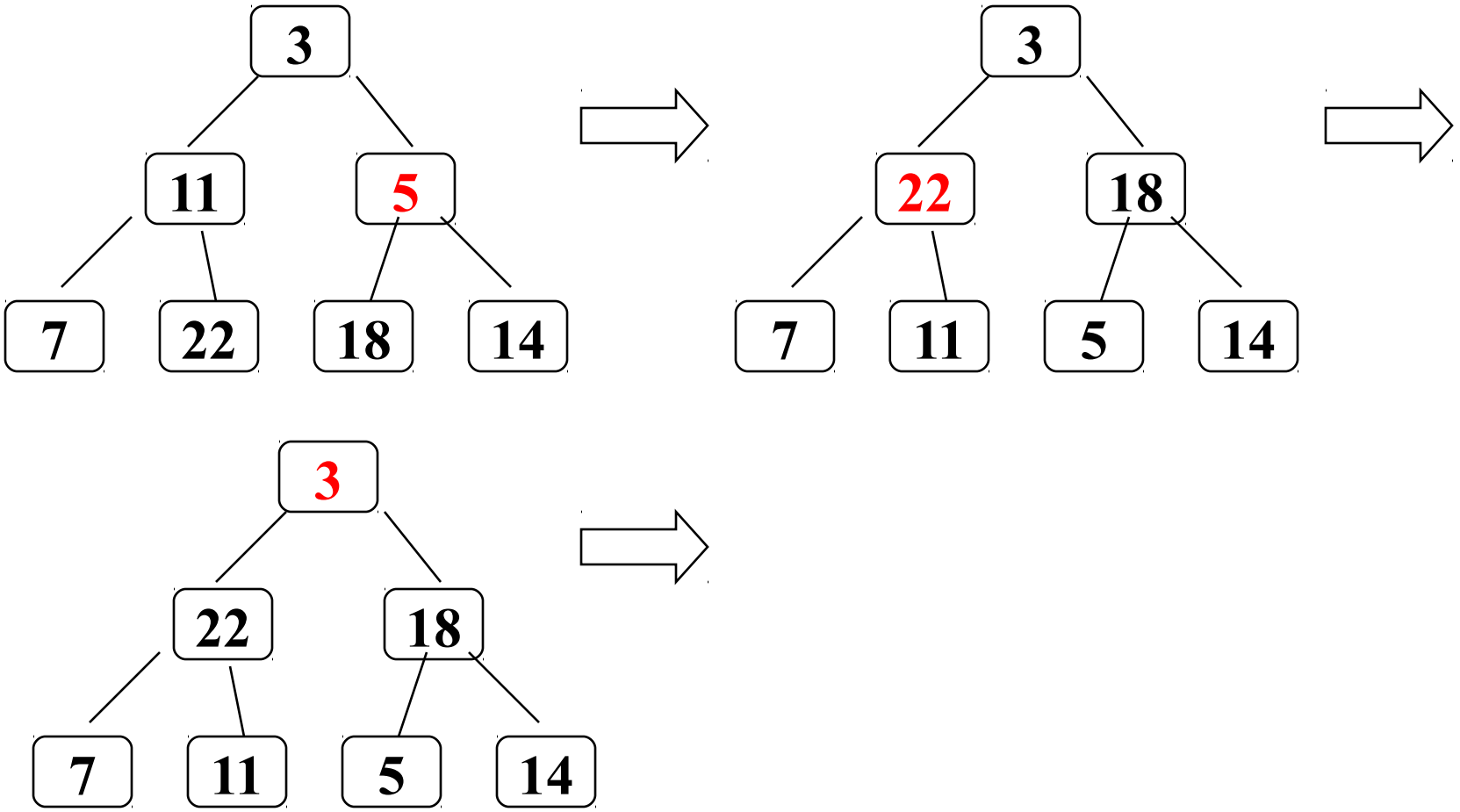


0	1	2	3	4
9	8	3	4	5

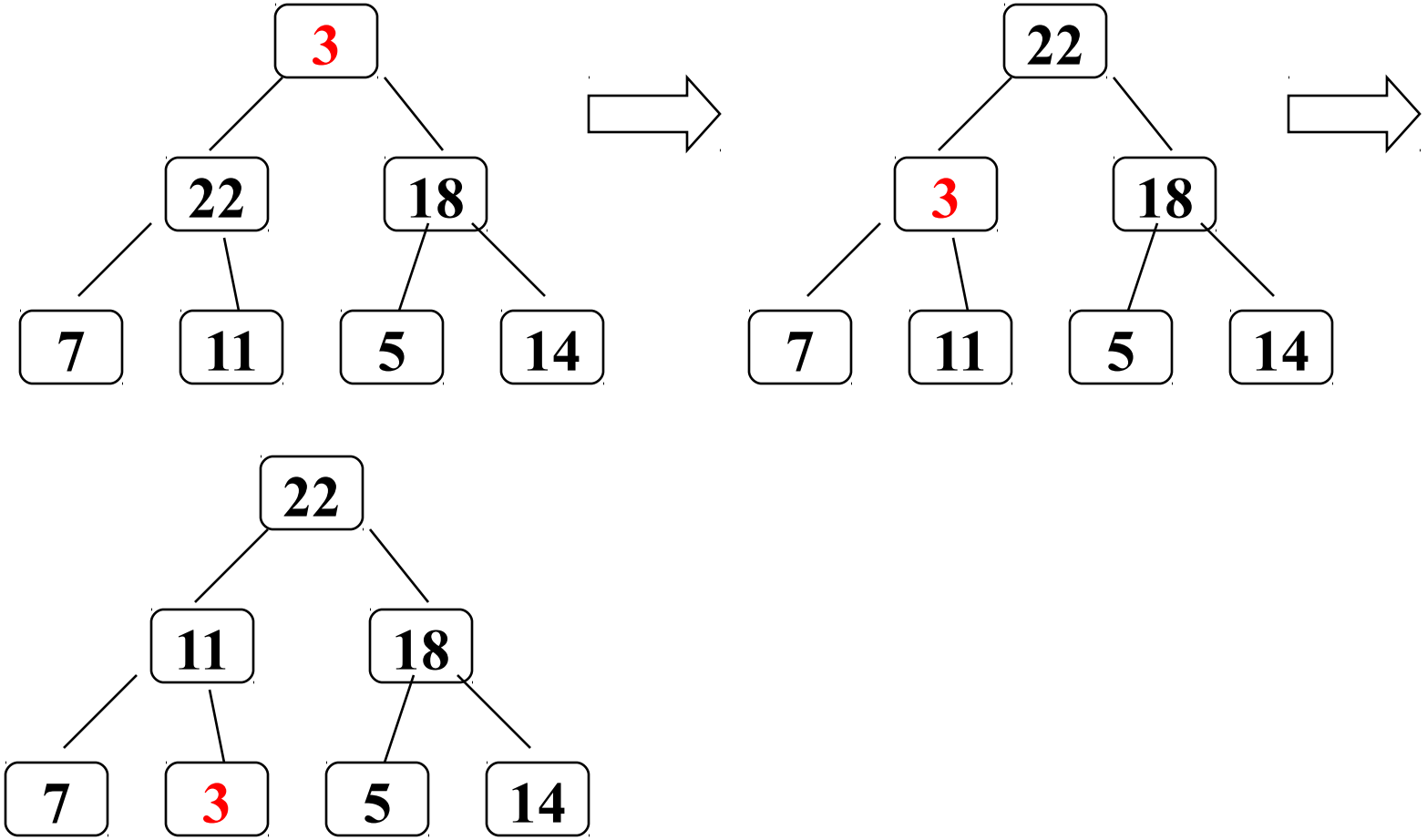
Building a Heap from an Array

- **Go from last non-leaf to index 0**
  - **At each node, do filter-down**

Building a Heap from an Array



Building a Heap from an Array



Building a Heap from an Array

- **Work at a node is O(h) where h is height of subtree rooted at that node**
- **In a complete binary tree, majority of nodes close to bottom, so adds up to O(n)**

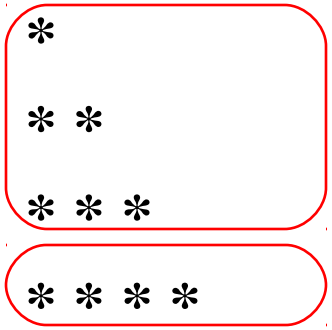
Review: Recursion

- **Recursion is a way of looking at a problem**
- **EG problem: print a pattern like**

```
*
* *
* * *
* * * *
```

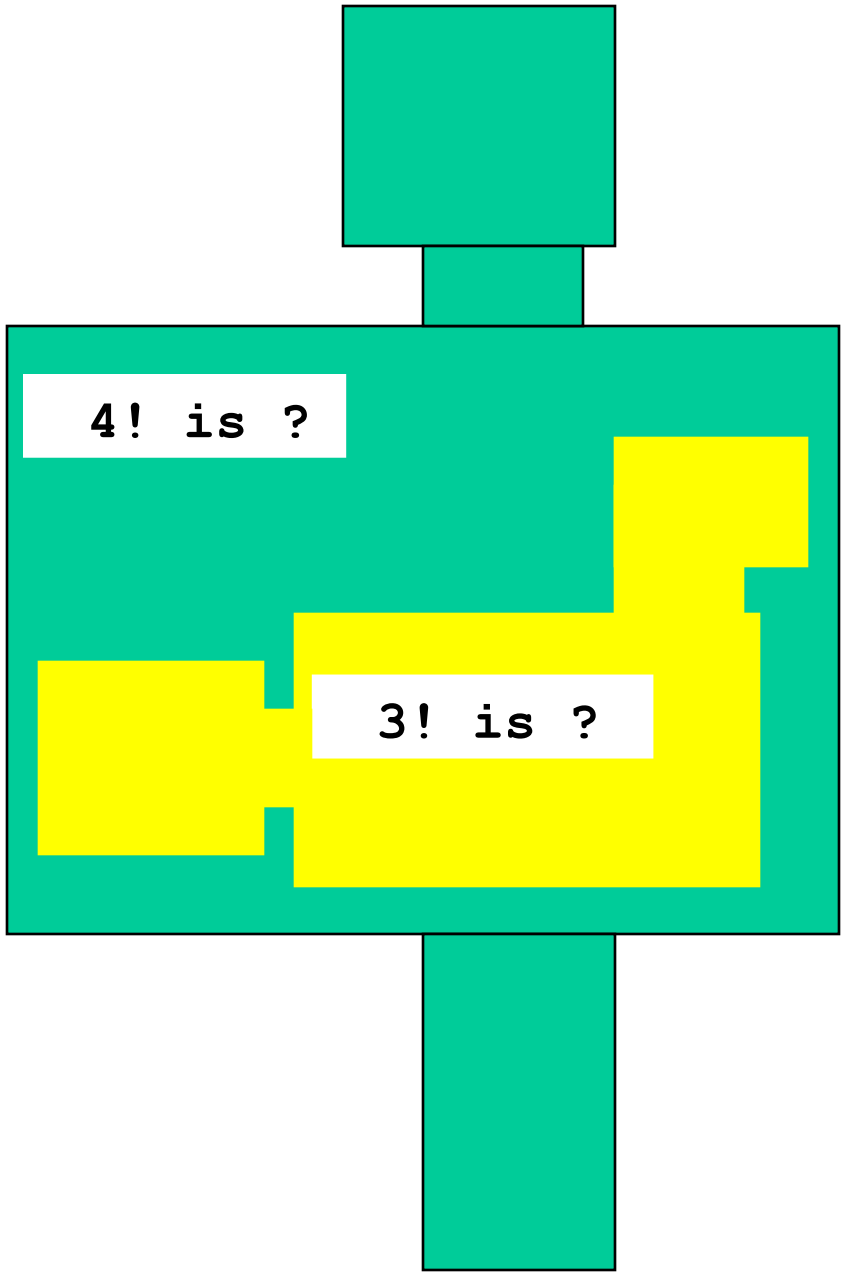
Recursion

- **Recursive view**



- **A size 4 triangle is**
  - **A size 3 triangle, followed by**
  - **A line of length 4**

- **Seeing recursion**
  - **look at a problem as if it were "pregnant":**
    - **inside it is a smaller version of the same problem**



Recursive Definitions in Math

Factorial in math

1! = 1 ← **Base case**

n! = n * (n-1)! ← **Recursive case**

Recursive Definitions in Java

Factorial in Java

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

Palindromes

- **A string is a palindrome if**
  - **first and last characters are the same, and**

    racecar
  - **rest of string without first and last is a palindrome**

    aceca
- **or if its length is 0 or 1**

  e

  See Palindrome,java

Integer Powers

How many multiplies does it take to calculate 3^8?

3*3	**=**	**9**	**3*3 = 9**
9*3	**=**	**27**	**9*9 = 81**
27*3	**=**	**81**	**81*81 = 6561**
…			
2187*3 = 6561			**3 *s**
7 *s			

Integer Powers

$x^y =$

1	$(y = 0)$
x	$(y = 1)$
$(x^{y/2})^2$	$(y \text{ even})$
$(x^{\lfloor y/2 \rfloor})^2 * x$	$(y \text{ odd})$

See Power.java

How Does Recursion Work?

```
static void triangle(int n){
   if (n==1){
      printNStars(1);
   } else {
      triangle(n-1);
      printNStars(n);
   }
}
```

Method Call

When method A calls method B

- **A's invocation record is put on hold**
  - **a new invocation record is created for B**
  - **B runs from beginning to end using this new invocation record**
  - **the new invocation record is destroyed**
- **A's old invocation record is reactivated**
- **A continues running from where it left off**

Method Call

When method A calls method ~~B~~ A

- **A's invocation record is put on hold**
  - **a new invocation record is created for ~~B~~ A**
  - **~~B~~ A runs from beginning to end using this new invocation record**
  - **the new invocation record is destroyed**
- **A's old invocation record is reactivated**
- **A continues running from where it left off**

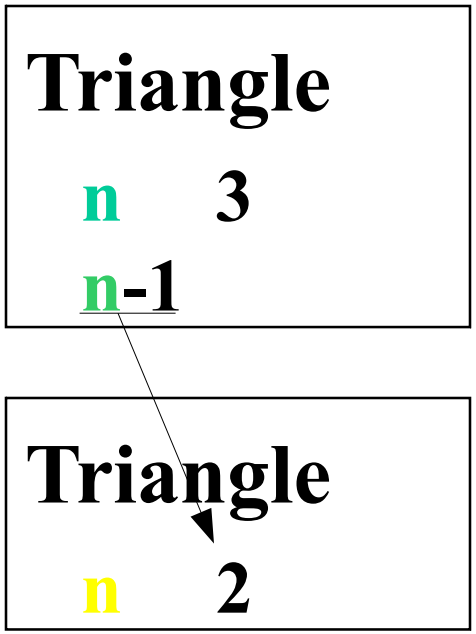
How Does Recursion Work?

Triangle

n 3

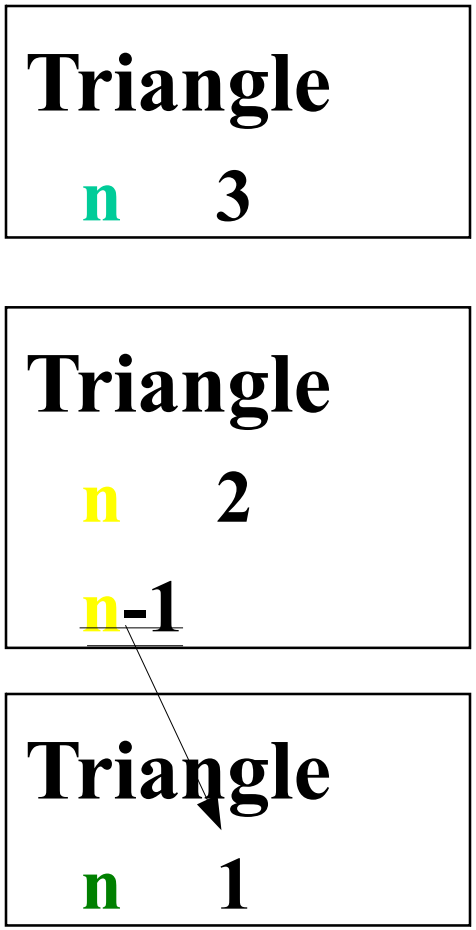
```
static void triangle(n ){
   if …
    triangle(n-1)
    printNStars(n);}
```

How Does Recursion Work?



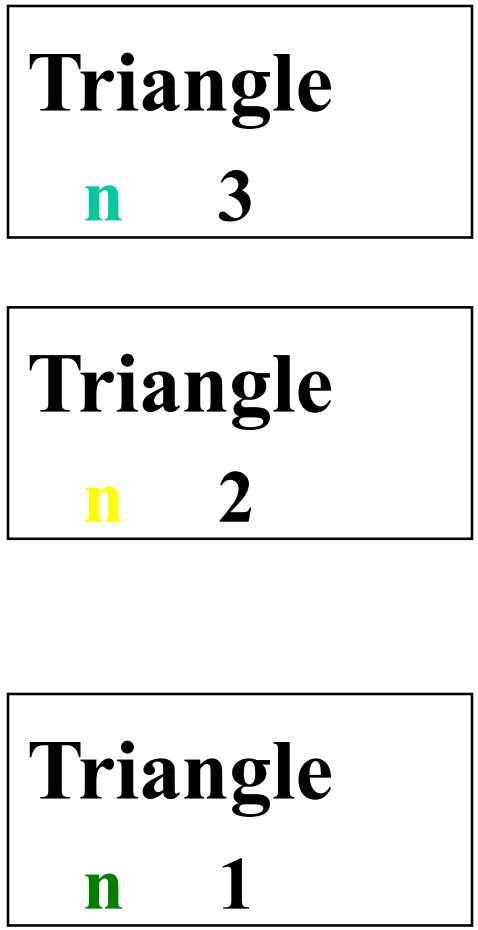

```
static void triangle(n ){
   if …
    triangle(n-1)
    printNStars(n);}

static void triangle(n ){
   if …
    triangle(n-1)
    printNStars(n);}
```

How Does Recursion Work?



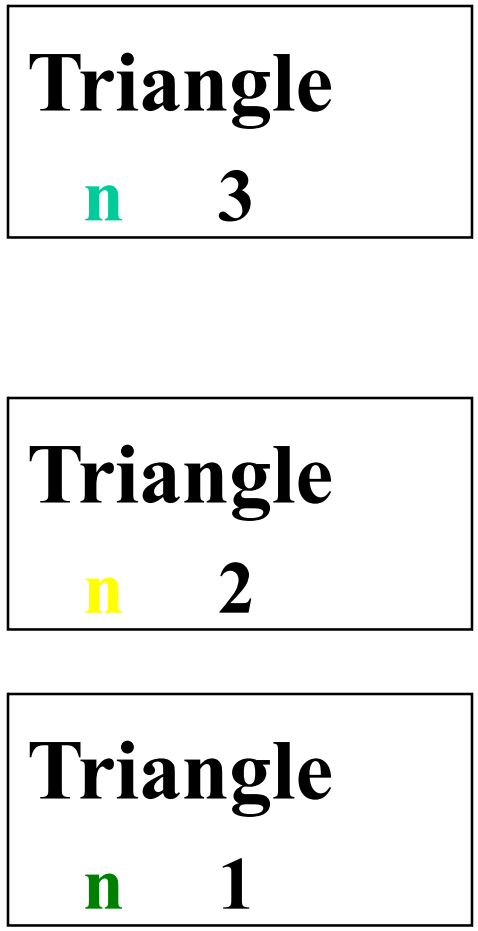

```
static void triangle(n ){
  if …
   triangle(n-1)
   printNStars(n);}

static void triangle(n ){
  if …
   triangle(n-1)
   printNStars(n);}

static void triangle(n ){
  if …
   printNStars(1)
```

How Does Recursion Work?




```
static void triangle(n ){
  if …
   triangle(n-1)                    *
   printNStars(n);}
static void triangle(n ){
  if …
   triangle(n-1)
   printNStars(n);}
static void triangle(n ){
  if …
   printNStars(1)
```

How Does Recursion Work?



How Does Recursion Work?

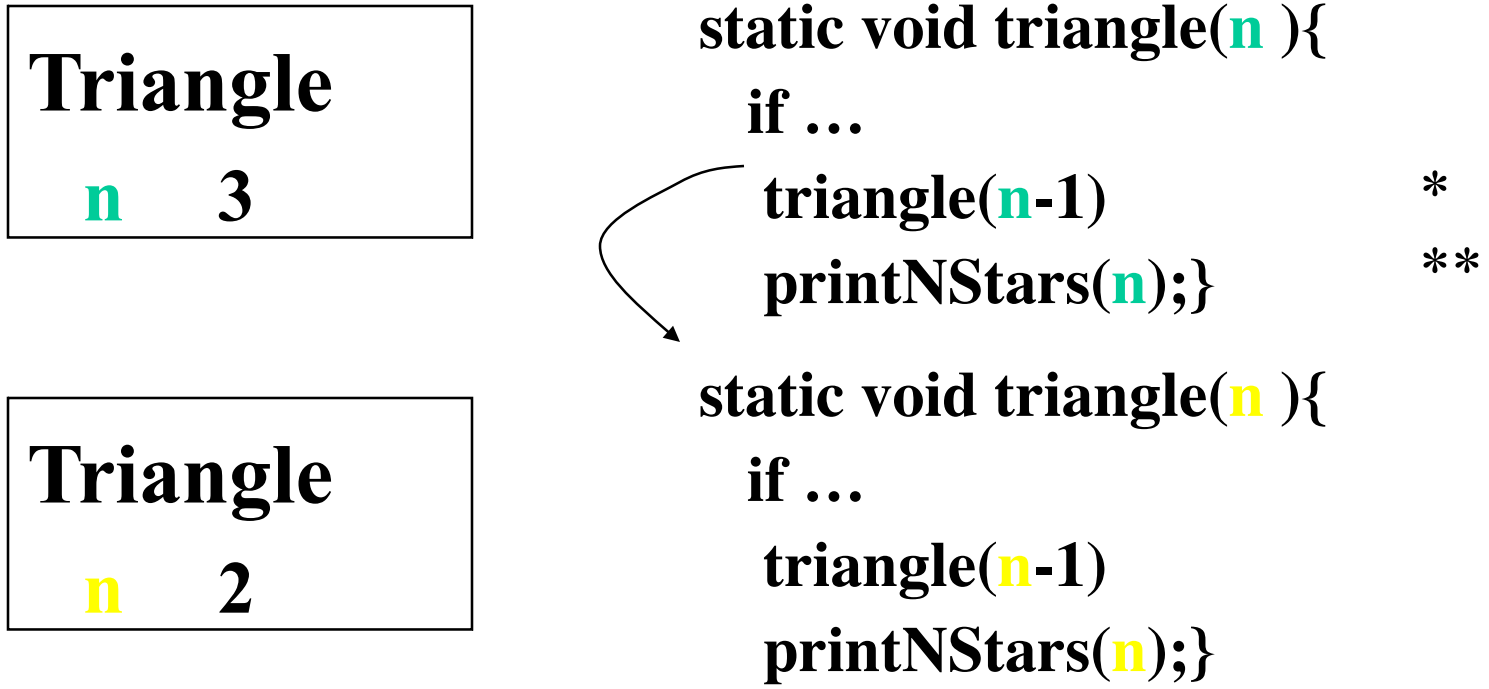
Triangle
n 3

```
static void triangle(n ){
  if …
   triangle(n-1)                    *
   printNStars(n);}
```

Triangle
n 2

```
static void triangle(n ){
  if …
   triangle(n-1)
   printNStars(n);}
```

How Does Recursion Work?

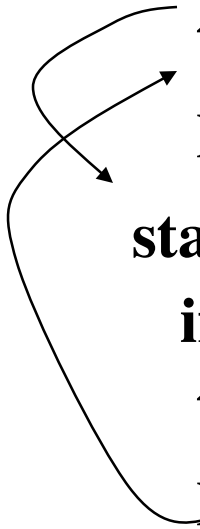


How Does Recursion Work?

Triangle
n 3

Triangle
n 2

```
static void triangle(n ){
  if …
   triangle(n-1)                    *
   printNStars(n);}                 **

static void triangle(n ){
  if …
   triangle(n-1)
   printNStars(n);}
```



How Does Recursion Work?

Triangle

n 3

```
static void triangle(n ){
   if …
    triangle(n-1)              *
    printNStars(n);}           **
```

How Does Recursion Work?

Triangle

n 3

```
static void triangle(n ){
   if …
    triangle(n-1)
    printNStars(n);}
```

```
*
**
***
```

How Does Recursion Work?

```
*
**
***
```

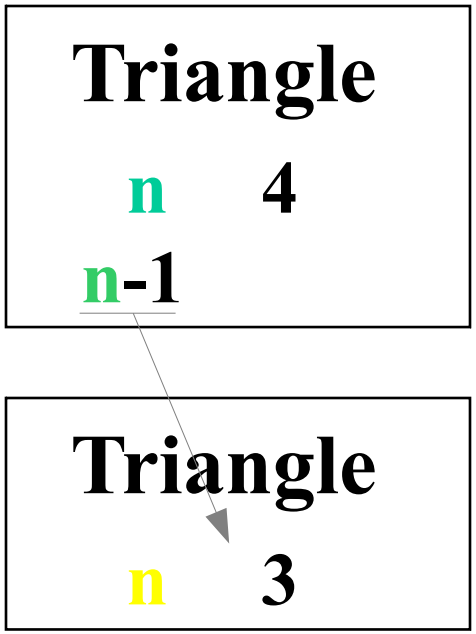
How Does Recursion Work?

Triangle

n 4

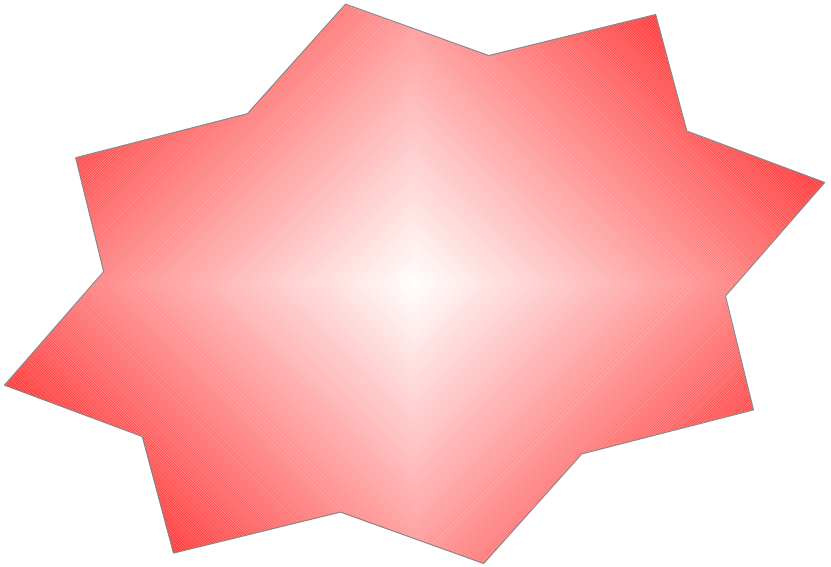
```
static void triangle(n ){
  if …
   triangle(n-1)
   printNStars(n);}
```

How Does Recursion Work?

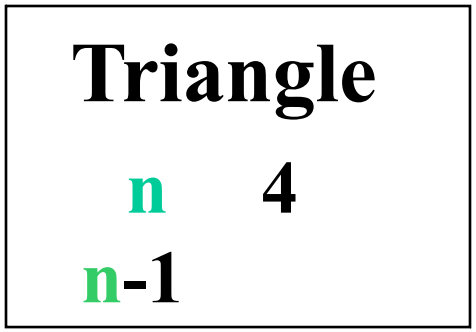



```
static void triangle(n ){
   if …
    triangle(n-1)
    printNStars(n);}

static void triangle(n ){
    …
}
```

Whoosh!!



How Does Recursion Work?




```
static void triangle(n ){
   if …
    triangle(n-1)
    printNStars(n);}

static void triangle(n ){
    …
}
```

```
*
**
***
```

How Does Recursion Work?

Triangle

n 4

n-1

```
static void triangle(n ){
   if …
    triangle(n-1)
    printNStars(n);}

static void triangle(n ){
    …
}
```

```
*
**
***
```

How Does Recursion Work?

Triangle

n 4

n-1

```
static void triangle(n ){
  if …
   triangle(n-1)
   printNStars(n);}
```

```
*
**
***
****
```

How Does Recursion Work?

*

**

***

****

Writing recursive methods

- **Write recursive code to print a downTriangle of size n**
  - **Assume you have printNStars(int n)**

****
***
**
*

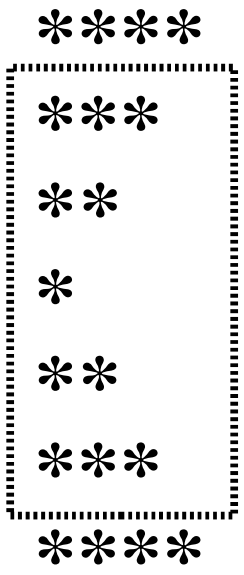
Writing recursive methods

- **Write recursive code to print a bowtie of size n**
  - **Assume you have printNStars(int n)**

```
****
***
**
*
**
***
****
```

Writing recursive methods

- **Write recursive code to print a bowtie of size n**
  - **Assume you have printNStars(int n)**



See Stars.java

Writing recursive methods

- **Code to produce a hill. E.g. hill(2, 4)**

```
**
***
****
***
**
```

See Stars.java

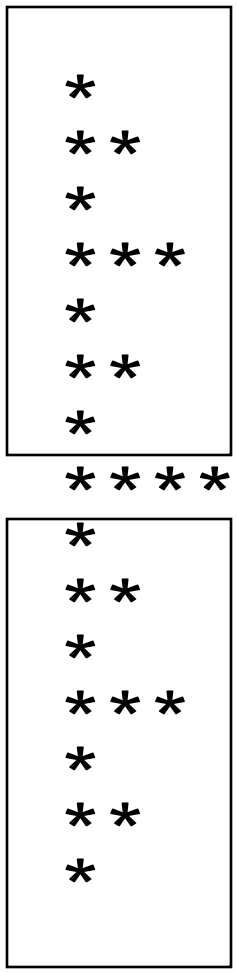
What does sumSq do?

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

Ruler Pattern

```
*
**
*
***
*
**
*
****
*
**
*
***
*
**
*
```

Ruler Pattern



- **Smaller problem appears twice!**
- **To do ruler n:**
  - **do ruler n-1**
  - **print n *s**
  - **do ruler n-1**
- **See Stars.java**

Tower of Hanoi

- **Rules:**
  - **3 stacks of disks**
  - **Move one disk at a time:**
    - **Take top disk off some stack**
    - **Put it on the top of another stack**
  - **Must not place larger disk on smaller**

  - **See TowersOfHanoi.java**