sit with anyone that looks friendly

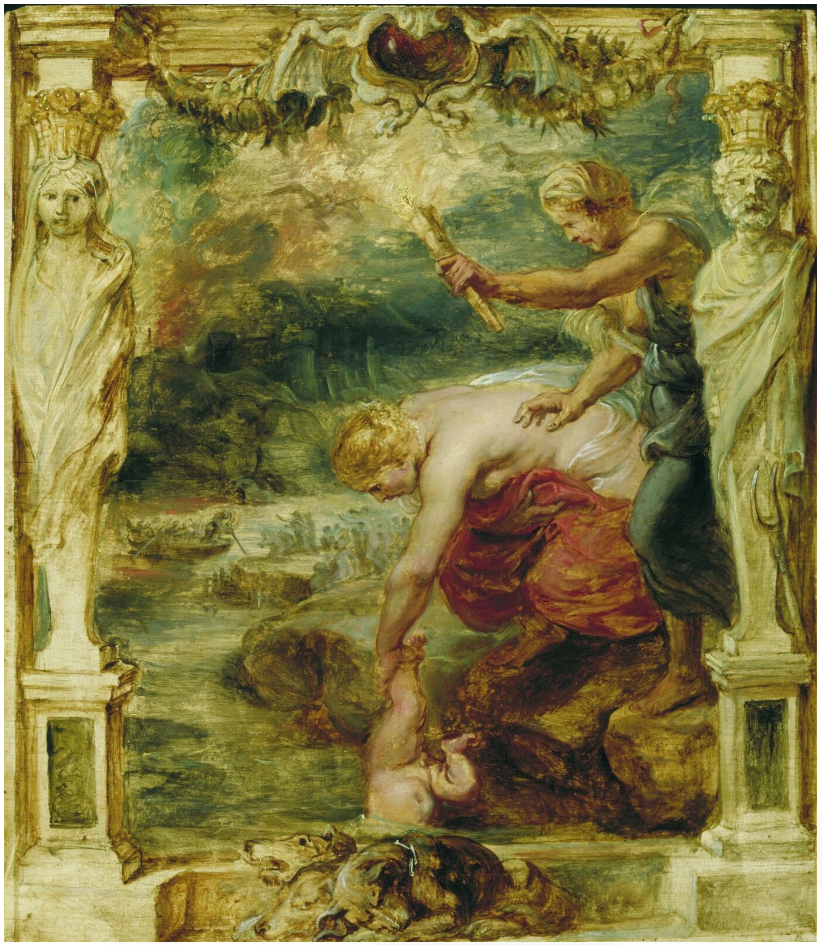
CMPS 12B/M
Introduction to Data Structures

- Instructor: Nathan Whitehead

Achille's Heel

- Goddess Thetis dips her baby in river Styx in Hades
  - Achilles becomes invulnerable except on his heel where held
- What is merge sort's heel?
- Number of comparisons
  - **O(n log(n))** good
- Space required
  - Needs extra **n** workspace

bad



More Visualizations

http://www.sorting-algorithms.com/

Moral of Story

tradeoffs

- There is not one perfect sort that is optimal for all problems
- Best sort depends on
  - which operations are expensive
  - how the data is ordered coming in
    - usually not entirely random (random is rare)
  - how easy it is to get working
    - buggy sort is no use to anyone

Why not just use the library sort built-in?

- **YES, YOU SHOULD!**
- But not always possible
  - API might not match way you want to sort
  - Structure might not all fit into memory
  - Might be badly optimized for your actual data ordering
  - There might be *many* sorts in libraries, how to choose?

Sorting Pictures in Flash Storage

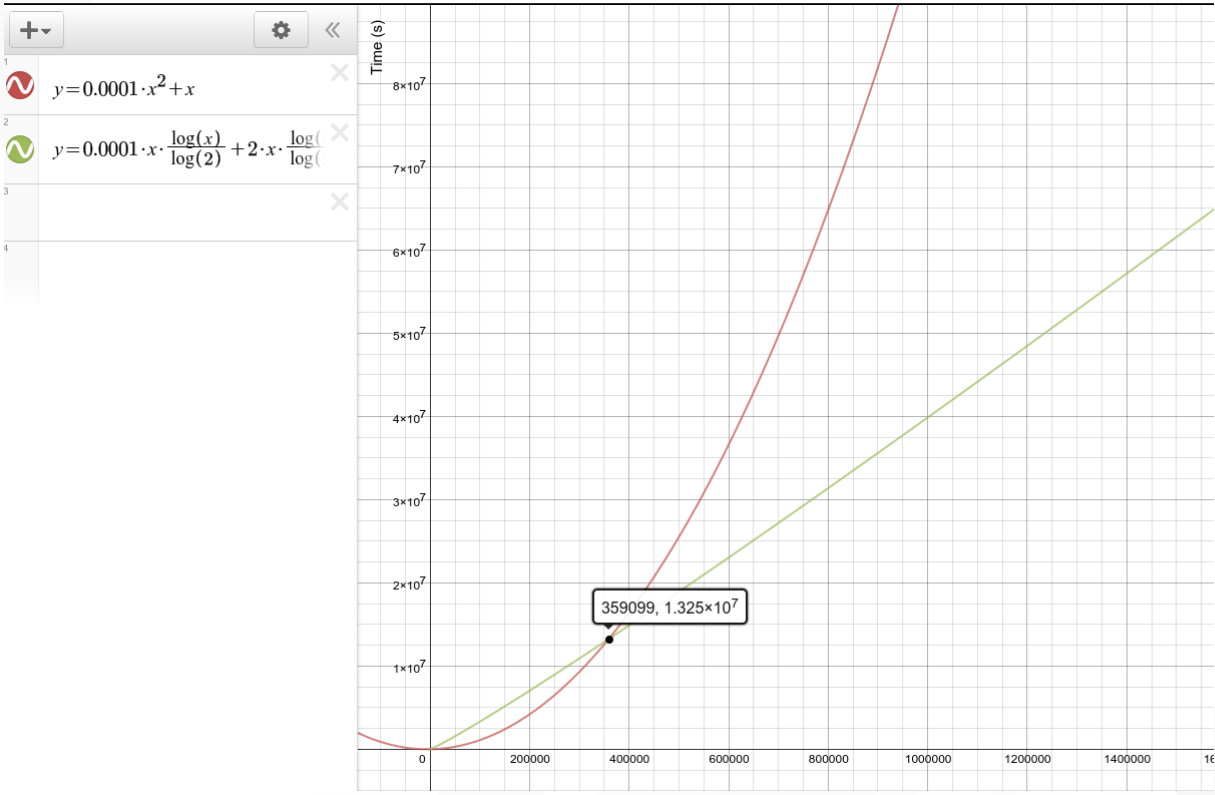
- **Quiz problem 3** – more details
- Suppose
  - Comparisons take 0.1 ms (read from cache)
  - Writing one picture to flash takes 1 s (slow!)
- *Merge sort* versus *Selection sort*
  - Merge sort has better big O...
  - Selection sort is linear in swaps...

- About how big does n have to be before merge sort wins?

Times

- Selection sort
  - Total comparisons
    - About n^2
  - Total moves/swaps
    - Exactly n
- Merge sort
  - Total comparisons
    - About n $\log_2(n)$
  - Total moves/swaps
    - About 2 n $\log_2(n)$
- Crossover
  - n=360000, time=1.3e7 seconds = 150 days

https://www.desmos.com/calculator/kyks5ivrlj



Adding up Numbers

- Let's add up numbers using code
- Given n, find sum of numbers 1 through n
- OK, we can do this...

Imperative (i.e. normal)

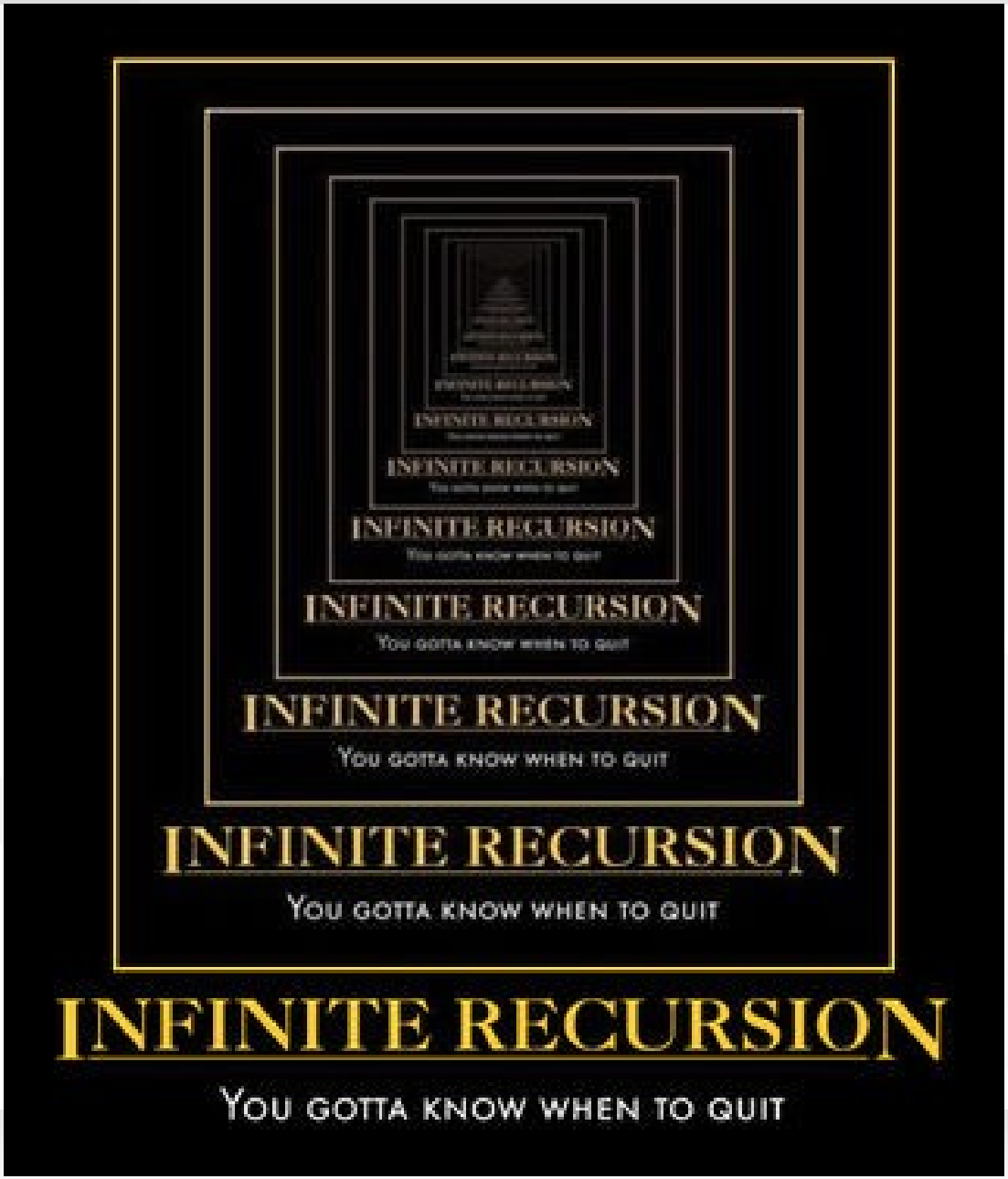
```
public static int sum(int n) {
    int total = 0;
    for(int i = 1; i <= n; i++) {
        total += i;
    }
    return total;
}
```

Recursive Version

- Want to write function sum
- What if we could ask the genie for solutions to any smaller problem?
  - How would that help?
  - How could we use smaller solutions to build bigger solutions?
- Suppose we know sum(n-1)
  - How does that help us get sum(n)?

sum(n) = sum(n-1) + n

Knowing When to Quit



Recursive Sum Facts

sum(n) = sum(n-1) + n

- When to quit?
  - How about sum(1) = 1

If n > 1 then
  sum(n) = sum(n-1) + n
else
  sum(1) = 1

Recursive Sum Code

```java
public static int sum(int n) {
    if (n > 1) {
        return sum(n - 1) + n;
    } else {
        return 1;
    }
}
```

sum(4)
=(sum(3) + 4)
=((sum(2) + 3) + 4)
=(((sum(1) + 2) + 3) + 4)
=(((1+2)+3)+4)
=((3+3)+4)
=(6+4)
=10

The Smart Way...

```java
public static int sum(int n) {
    return n * (n + 1) / 2;
}
```

math for the win!

When to use recursion?

- It takes memory to keep track of where you are in recursion
  - Cheap but not free
- It takes time to do function calls
  - Cheap but not free
- Makes many problems *conceptually simpler*
  - Small overhead is worth simpler code!

Stacks and Queues

- An Englishman, even if he is alone, forms an orderly queue of one.

  -*George Mikes*

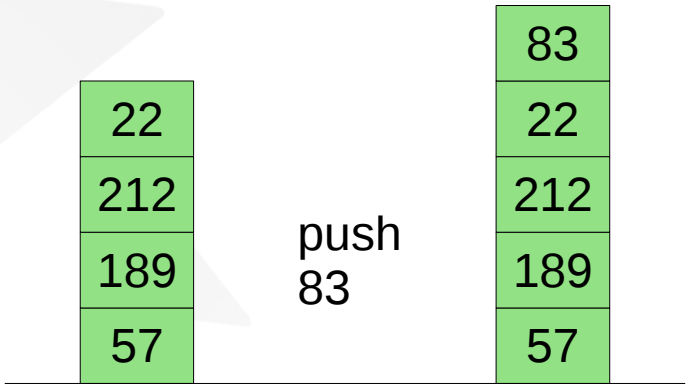
Different Kind of Structure

- Stacks and queues restrict access
  - Less capabilities compared to arrays
  - More suited to programming tasks than data tasks

interface matters

Stacks

- Only have access to top of stack
  - *Push* – add an element to top
  - *Pop* – remove top element
  - *Peek* – look at top without removing



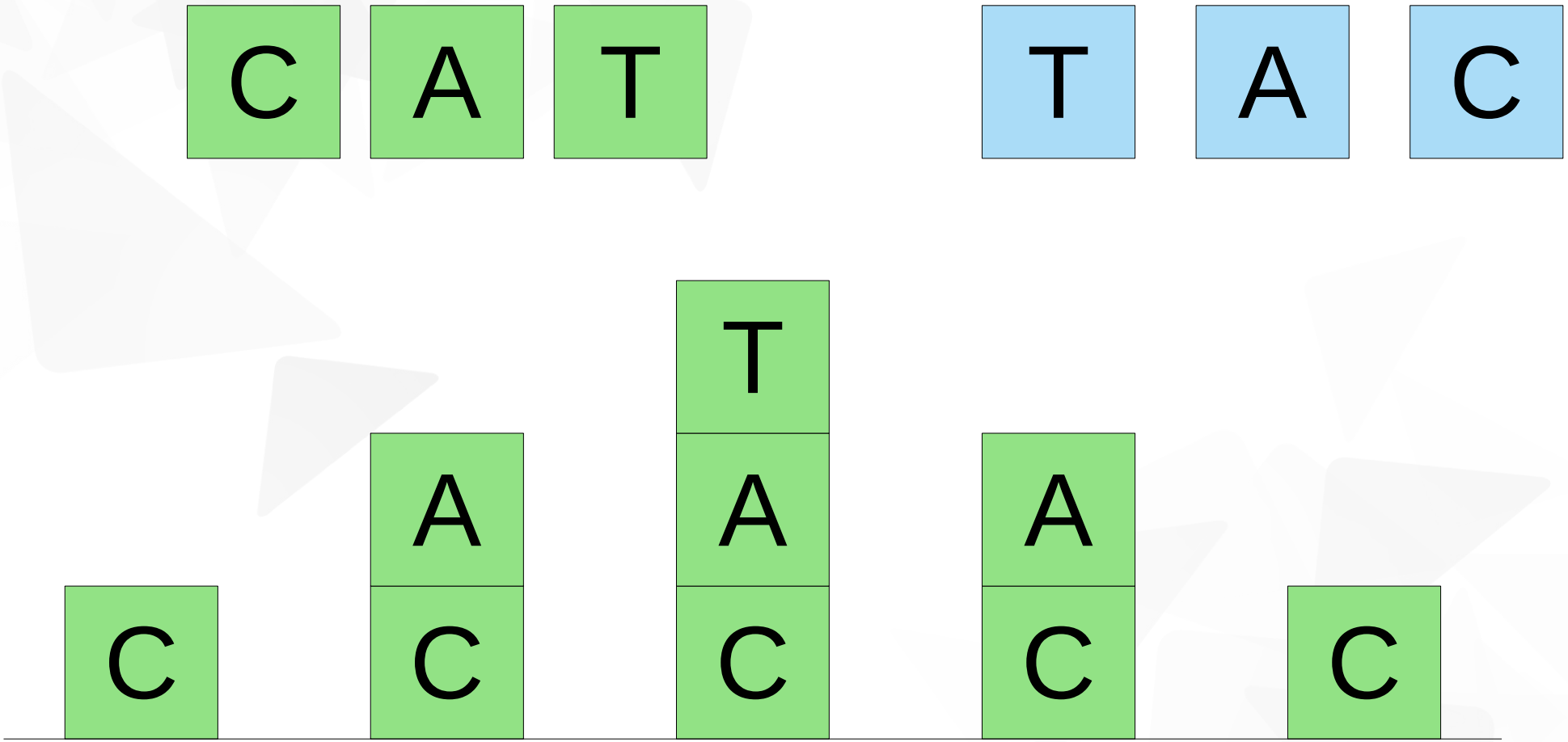
Stack Code

- Examples/Chap04/Stack/stack.java

Reversing Items

- Suppose you want to process items in reverse order
- We've solved this
  - Put all the elements in an array
  - Do a loop, swapping elements to reverse the array
  - Process elements in the array
- Easier
  - Put all elements in an array
  - Loop through array in reverse order

Reversing with a Stack



Matching Parentheses and Brackets

a	[	b	(	a	)	]

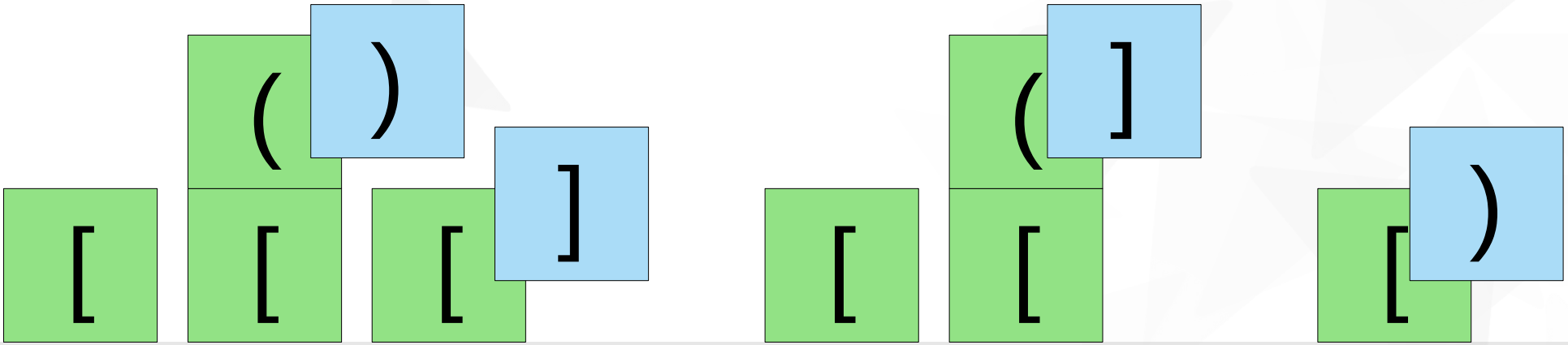
✓

a	[	b	(	a	]	]

✗

a	[	b	)	a	(	]

✗



Bracket Checker Code

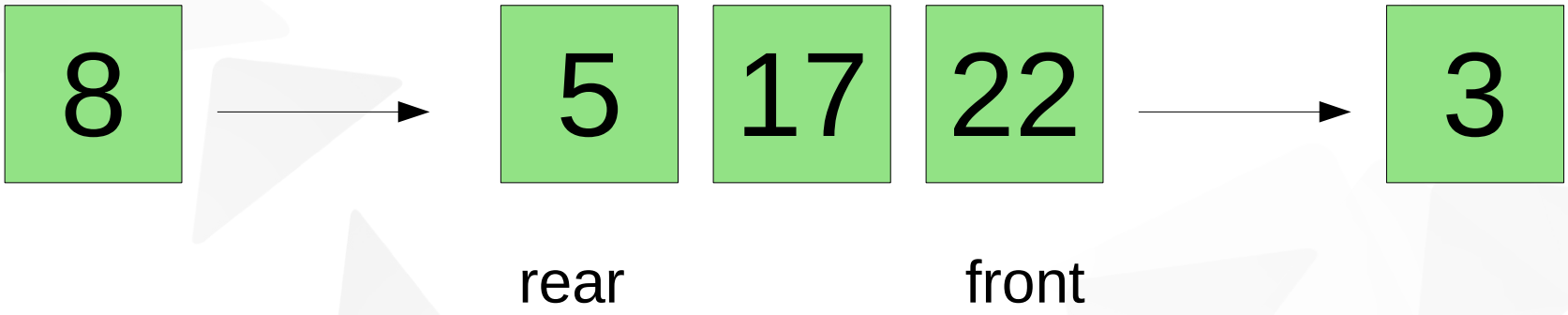
- Examples/Chap04/Brackets/brackets.java

Stack Efficiency

- How efficient are stacks?
  - Push, pop, peek
    - No comparisons needed
    - One array access
    - Doesn't depend on how many elements in stack
    - O(1) time for all operations

Queue

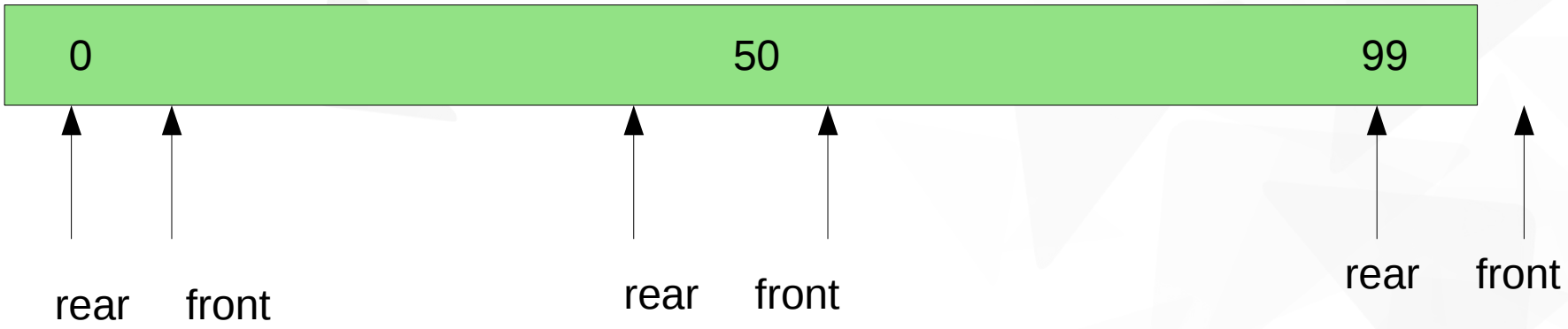
- Waiting in line
  - Arrive at the rear
  - Leave at the front
  - FIFO (first-in-first-out)



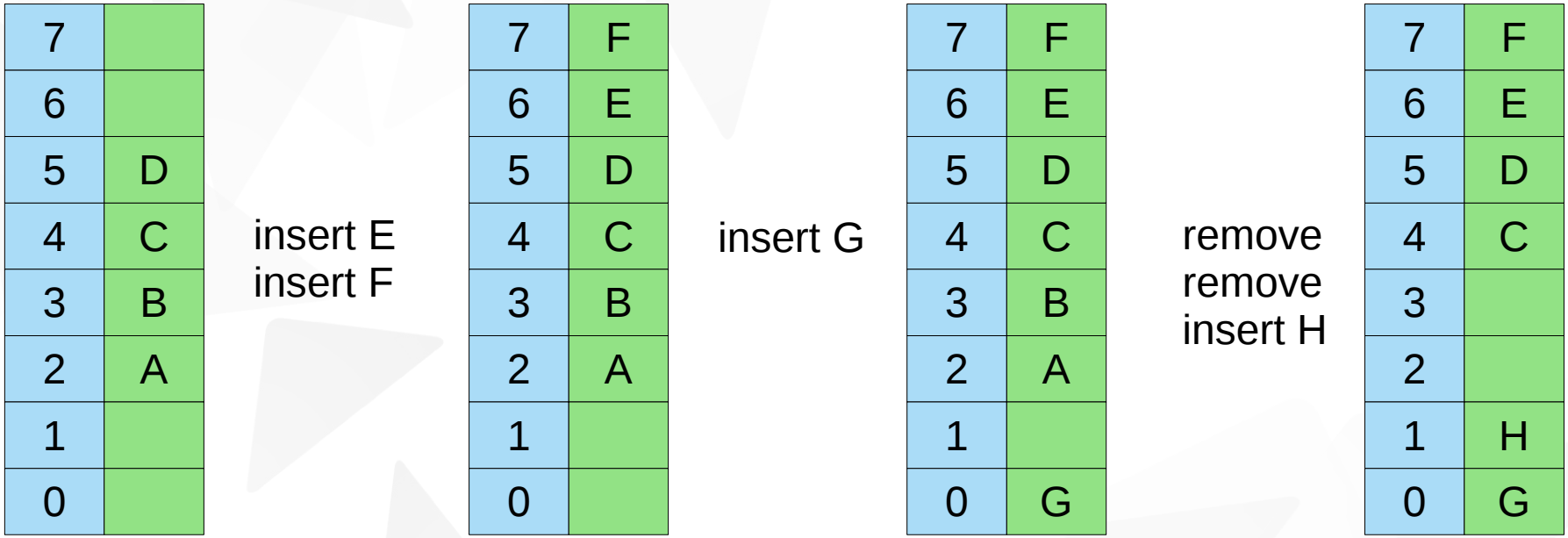
incredibly useful for keeping track of events, work to do, etc.

Queue

- Methods
  - *Insert* – add to rear of queue
  - *Remove* – remove from front of queue
  - *Peek* – look at front of queue
- Implementation
  - Maybe an array?
  - Front and rear keep increasing over time



Circular Buffer



wraparound

Circular Queue Code

- Examples/Chap04/Queue/queue.java

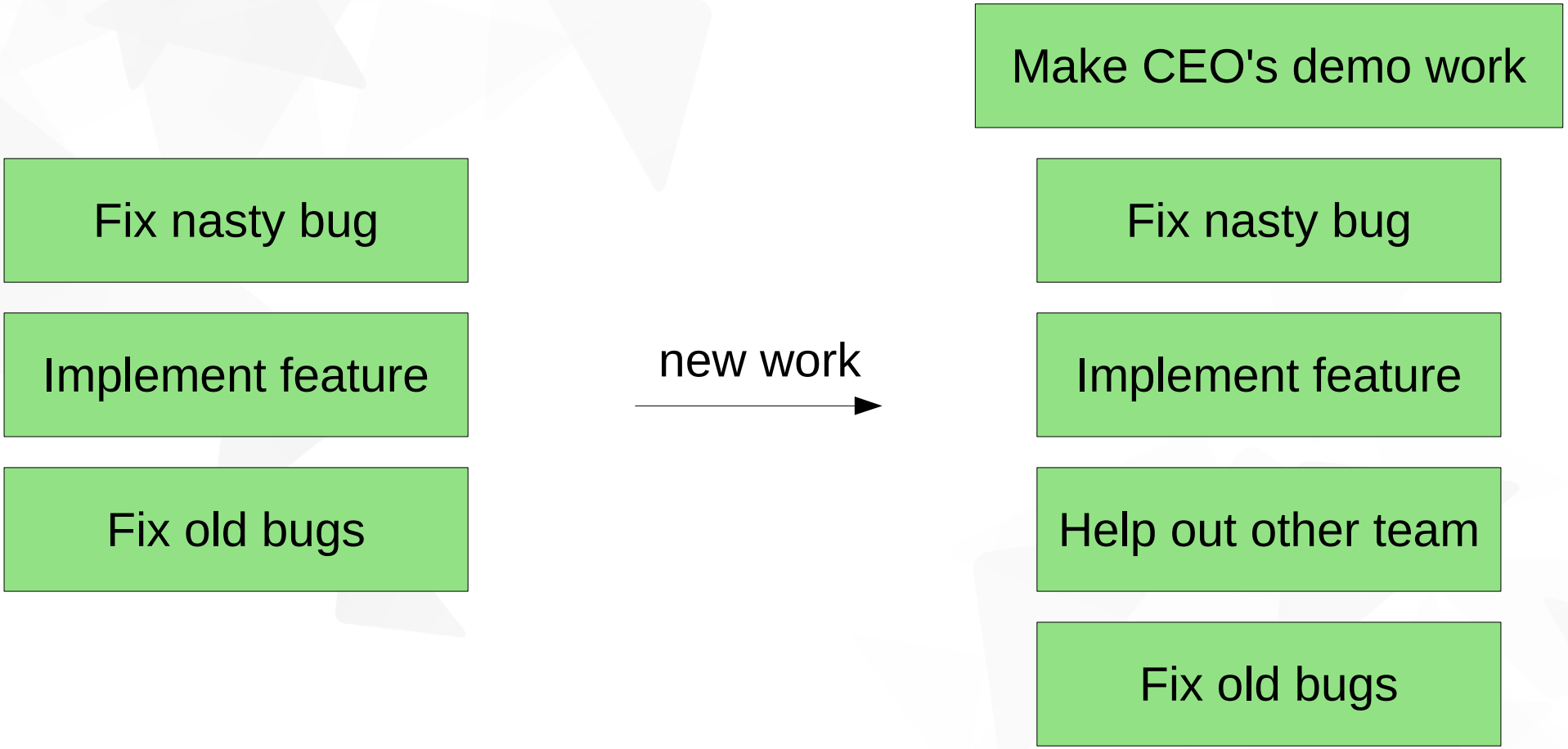
Queue Efficiency

- How efficient are circular queues?
  - Insert
    - No comparisons
    - One array access
    - O(1)
  - Remove
    - No comparisons
    - One array access
    - O(1)

Deque

- With circular techniques we can insert and remove from front or rear of buffer
  - Left and right indexes
  - insertLeft, removeLeft, insertRight, removeRight
  - insertRight and removeRight == stack
  - insertLeft and removeRight == queue
- Deques are generalization of stacks and queues
  - Commonly provided by libraries
  - Usually a stack or queue is what problem needs

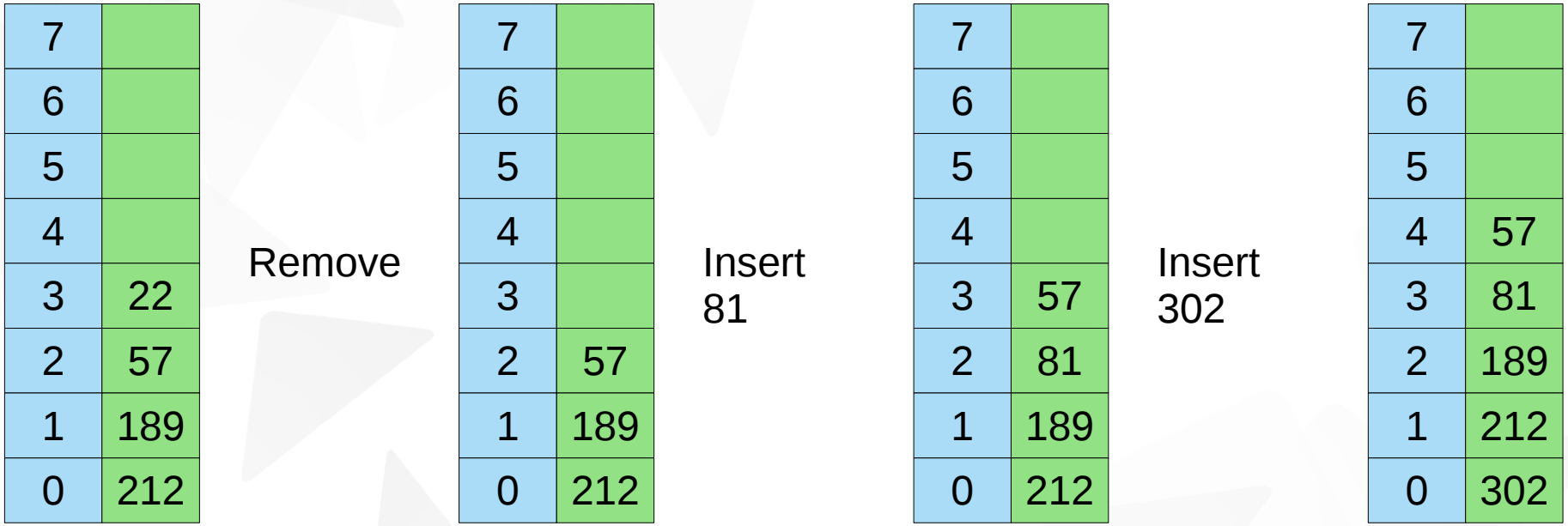
Priority Queue



Priority Queue

- Items go into queue
- Items come out of queue lowest first
  - Not FIFO
- One implementation
  - Keep sorted array
  - *Remove* – take out element in front (it is smallest)
  - *Insert* – find proper place to put new item, make room for it

Priority Queue



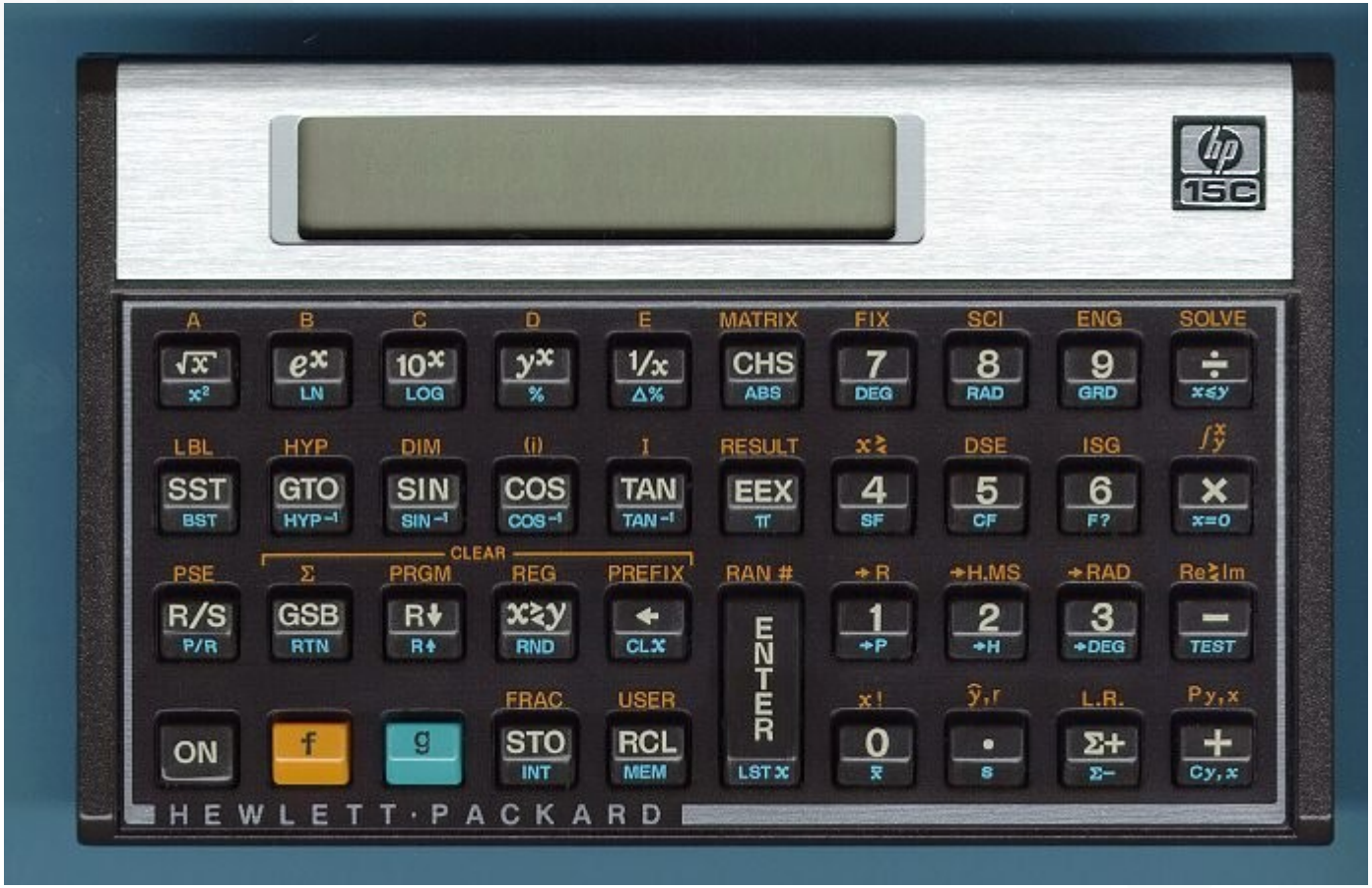
Efficiency of Priority Queue

- Our first implementation
  - Remove
    - No comparisons
    - No copying
    - One array access
    - O(1)
  - Insert
    - Find where to put it
    - Shift elements out of the way
    - Worst case requires shifting *n* elements
    - O(*n*) comparisons and copies

there is another way

we'll see later

Parsing Arithmetic Expressions



http://hp15c.com/

The End