A CADBURY EGG HAS ABOUT 20g OF SUGAR.

(25g OUTSIDE THE US.)

"ONE CADBURY EGG" IS A NICE UNIT OF SUGAR CONTENT.

ONE 12oz. CAN OF SODA HAS ABOUT TWO CADBURY EGGS WORTH OF SUGAR.

ONE 20oz. BOTTLE HAS THREE.

ONE CADBURY EGG IS ENOUGH TO MAKE ME FEEL KINDA GROSS.

NOW WHEN I SEE COKE OR SNAPPLE OR NESTEA OR WHATEVER, I IMAGINE DRINKING A COUPLE DISSOLVED CADBURY EGGS.

WOW. HUH.

SO THE TAKEAWAY IS... I CAN EAT CADBURY EGGS BY THE HANDFUL ALL SEASON AND FEEL NO WORSE ABOUT IT THAN I DO ABOUT SODA?

THIS IS *AWESOME!*

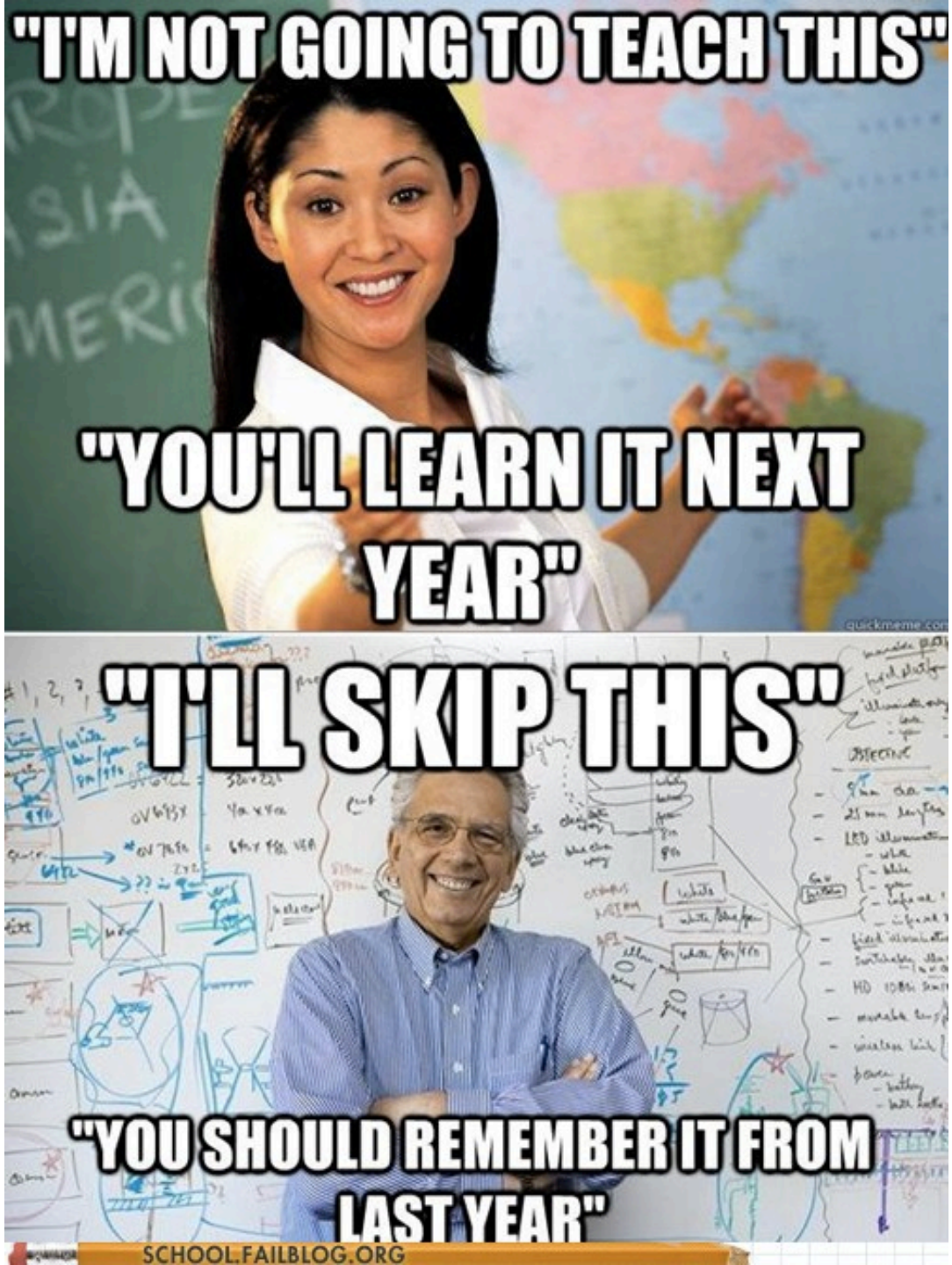
THAT'S NOT REALLY—

SIGH

- For today: Chapter 9
- Last week's exercises

- This week: Sorting
- Next week: Recursion

- Project 2: Will be due on May 12 (day of exam)


"I'M NOT GOING TO TEACH THIS"
"YOU'LL LEARN IT NEXT YEAR"
"I'LL SKIP THIS"
"YOU SHOULD REMEMBER IT FROM LAST YEAR"
SCHOOL.FAILBLOG.ORG

Sorting

- Will develop algorithms for arrays of int
  - easy to change for other types of elements
- Resulting order will be nondecreasing
  - easy to change for nonincreasing

Selection Sort

- Simple way to think about sorting
  - Find smallest element and put in first available slot
  - Find second smallest; put in next slot
  - Find third smallest; put in next slot
- Want to use as little extra space as possible
  - perform sort in place
  - move data around in array; no need for second array

Swapping

- Many algorithms require ability to interchange values of two variables
- to sort in place, will need to swap values in array[i] and array[j]

```
private void swap( int[] array, int i, int j, ) {
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}
```

Why is temp needed?

Iterative Selection Sort

- Keep track of slotToFill
  - initially 0
  - then 1, 2, etc.
- Find index of smallest element in segment of array not yet sorted
- Swap smallest element with element at index slotToFill
- Value of slotToFill during final execution of for loop is array.length-2. Why is that?

Operation of Selection Sort

Array of 5 ints

index: 0 1 2 3 4

elts[index]: 48 62 38 51 15

- slotToFill is 0
  - Find index of smallest in elts[0..4]
  - smallest in elts[4]
  - swap elts[4] with elts[0]

index:	0	1	2	3	4
elts[index]:	*15*	62	38	51	48

- slotToFill is 1
  - Find index of smallest in elts[1..4]
  - smallest in elts[2]
  - swap elts[2] with elts[1]

index:	0	1	2	3	4
elts[index]:	*15*	*38*	62	51	48

- slotToFill is 2
  - Find index of smallest in elts[2..4]
  - smallest in elts[4]
  - swap elts[4] with elts[2]

index: 0 1 2 3 4

elts[index]: *15 38 48* 51 62

- slotToFill is 3
  - Find index of smallest in elts[3..4]
  - smallest in elts[3]
  - swap elts[3] with elts[3]

index: 0 1 2 3 4

elts[index]: *15 38 48 51* 62

When slotToFill is incremented again, it will be equal to 4, which is array.length-1, so the for loop will terminate and elts is now in order

index: 0 1 2 3 4

elts[index]: 15 38 48 51 62

Sorting out Sorting

Complexity of Selection Sort

- Calculate number of comparisons needed to sort an array
- For array of n elements
  - n-1 comparisons to find smallest
  - then n-2 comparisons to find smallest of remaining n-1 items
  - then n-3 comparisons

    .

    .

    .
  - 1 comparison to find smallest of 2 elements

Total = (n-1) + (n-2) +.....+1

- Note: average case is same as worst case

Summing the Sequence

An observation:

Total = (n-1) + (n-2) + ... + 1

Total = 1 + 2 + ... + (n-1)

2(Total) = n + n + ... + n

i.e., (n-1) copies of n

since 2(Total) = (n-1)n

$$\text{Total} = \frac{(n-1)n}{2}$$

Bubble Sort

Consider the following psudocode. First, explain in English how this code works. Then, write this method in Java.

```
procedure bubbleSort( int [] A )
  repeat
    swapped = false
    for i = 1 to length(A) - 1 inclusive do:
      if A[i-1] > A[i] then
        swap( A[i-1], A[i] )
        swapped = true
      end if
    end for
  until not swapped
end procedure
```