## *What's left??*

- ***Algorithms:*** Unambiguous instructions to accomplish some task.

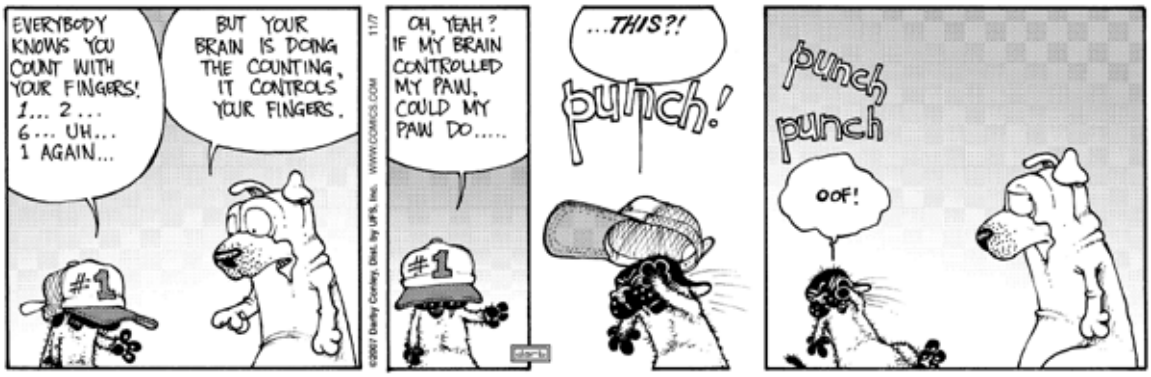
  To find my name in the phone book, scan trough the names in order until you find mine.

- ***Recursion:*** Induction for algorithms.

  To find my name in part of the the phone book, divide it in half, figure out which half contains my name, and find my name in that half.

- ***Counting:*** How many X's are there in Y?

  How many four-letter words can you make with the letters in ABRACADABRA?



© Darby Conley/Dist. by UFS, Inc.

- ***Big-O notation:*** How to talk about the running time of algorithms without knowing anything about the CPU speed, programming language, compiler, operating system, etc.

  Linear search takes $O(n)$ time; binary search takes $O(\log n)$ time.

- ***Solving recurrences:*** How fast is that recursive algorithm?

  If $T(n) \leq T(n/2) + O(1)$, then $T(n) = O(\log n)$.

## *Algorithms*

An ***algorithm*** is a set of unambiguous instructions describing how to transform some input into some output.

- The input and output are from some specified domains.
- Generality: The algorithm applies to many different inputs.
- Each step is precisely described.
- Each step can be performed in finite time.
- For any input, the algorithm finishes after a finite amount of time.

Richard Feynman's problem-solving "algorithm":

1. Write down the problem.
2. Think very hard.
3. Write down the answer.

What we need to know about algorithms:

- Correctness: Does it do what we want?
- Efficiency: How fast is it?

```
COLLATZ(x):
  while x > 1
      print x,
      if x is even
          x ← x/2
      else
          x ← 3x + 1
  print x
```

COLLATZ$(x)$ halts for all positive integers $x \leq 10^{18}$.

But nobody knows if COLLATZ$(x)$ halts for ***all*** positive integers $x$.

- COLLATZ$(1) \implies 1$
- COLLATZ$(2) \implies 2, 1$
- COLLATZ$(3) \implies 3, 10, 5, 16, 8, 4, 2, 1$
- COLLATZ$(7) \implies 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$
- COLLATZ$(27) \implies$

  $27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 618, 309, 928, 464, 232, 116, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$

***Is it correct?***

```
SELECTIONSORT(A[1..n]):
  for i ← 1 to n
      for j ← i + 1 to n
          if A[j] < A[i]
              tmp ← A[i]
              A[i] ← A[j]
              A[j] ← tmp
```

***How do we prove that an algorithm is correct?***

***What does "correct" actually mean?***

**Theorem:** SELECTIONSORT *reorders the elements of any input array* $A[1..n]$ *into increasing order:*

$$\boxed{A[1] \le A[2] \le A[3] \le \cdots \le A[n]}$$

*In other words,* SELECTIONSORT **sorts** *its input.*

**Lemma:** *For any input array* $A[1..n]$ *and any integer* $i$*, after the* $i$*th iteration of the outer loop,* $A[i]$ *is the smallest element of* $A[i..n]$*.*

```
SELECTIONSORT(A[1..n]):
  for i ← 1 to n
      for j ← i + 1 to n
          if A[j] < A[i]
              swap A[i] ↔ A[j]
```

**Lemma:** *For all input arrays* $A[1..n]$*,*
*for all* $i$ *and* $j$ *such that* $1 \le i < j \le n$*,*
*during the* $i$*th iteration of the outer loop,*
*after the* $j$*th iteration of the inner loop,*
$A[i]$ *is the smallest element of* $A[i..j]$*.*

**Proof:** Let $A[1..n]$ be an arbitrary input array.
Let $i$ and $j$ be arbitrary integers such that $1 \le i < j \le n$.
Consider the $i$th iteration of the outer loop.

Assume that for all $k < j$, after the $k$th iteration of the inner loop, $A[i]$ is the smallest element of $A[i..k]$.

There are two cases to consider: $j = i + 1$ and $j > i + 1$.

- Suppose $j = i + 1$.

  If $A[i] > A[i + 1]$ before the if statement,
  then the algorithm swaps $A[i]$ and $A[i + 1]$,
  so $A[i] < A[i + 1]$ afterwards.

  If $A[i] \le A[i + 1]$ before the if statement,
  then the algorithm does not swap $A[i]$ and $A[i + 1]$,
  so $A[i] \le A[i + 1]$ afterwards.

- Suppose $j \ge i + 2$.

  The inductive hypothesis implies that just *before* the $j$th iteration of the inner loop, $A[i] = \min A[i..j - 1]$.

  So in the $j$th iteration, the algorithm compares $\min A[i..j-1]$ with $A[j]$ and swaps the smaller of the two values into $A[i]$.

  So the new value of $A[i]$ is $\min\{\min A[i..j - 1], A[j]\} = \min A[i..j]$.

In both cases, we conclude that after the $j$th iteration of the inner loop, $A[i]$ is the smallest element of $A[i..j]$. □

```
SELECTIONSORT(A[1..n]):
  for i ← 1 to n
      for j ← i + 1 to n
          if A[j] < A[i]
              swap A[i] ↔ A[j]
```

**Lemma:** *For all $i$ such that $1 \leq i \leq n$, after the $i$th iteration of the outer loop, $A[1..i]$ contains the $i$ smallest elements of $A[1..n]$, in sorted order.*

**Proof:** Let $i$ be an arbitrary non-negative integer.

Assume for any positive integer $h \leq i$, after the $h$th iteration of the outer loop, $A[1..h]$ contains the $h$ smallest elements of $A[1..n]$, in sorted order.

- Suppose $i = 1$. The previous lemma implies that after the last iteration of the inner loop, $A[i] = A[1]$ is the smallest element in the array.
- Suppose $i \geq 2$.

  The inductive hypothesis implies that just before the $i$th iteration begins, $A[1..i-1]$ contains the $i-1$ smallest elements in sorted order.

  The previous lemma implies that the inner loop puts the next smallest element into $A[i]$.

□

## ***Loop invariants***

```
SELECTIONSORT(A[1..n]):
  for i ← 1 to n
      ⟨⟨A[1..i−1] is sorted⟩⟩
      for j ← i + 1 to n
          ⟨⟨A[i] is smallest in A[i..j−1]⟩⟩
          if A[j] < A[i]
              swap A[i] ↔ A[j]
          ⟨⟨A[i] is smallest in A[i..j]⟩⟩
      ⟨⟨A[i] is smallest in A[i..n]⟩⟩
      ⟨⟨A[1..i] is sorted⟩⟩
  ⟨⟨A[1..n] is sorted⟩⟩
```

## ***How long does it take?***

```
SELECTIONSORT(A[1..n]):
  for i ← 1 to n
      for j ← i + 1 to n
          if A[j] < A[i]
              swap A[i] ↔ A[j]
```

- What computer? What language? What compiler? What OS?
- How long does it take to compare $A[i]$ and $A[j]$? To swap $A[i]$ and $A[j]$? To maintain $i$ and $j$?
- How many times do we swap?
- What numbers are in the array $A[1..n]$?
- What is $n$?

## ***How long does it take?***

```
SELECTIONSORT(A[1..n]):
  for i ← 1 to n
      for j ← i + 1 to n
          if A[j] < A[i]
              swap A[i] ↔ A[j]
```

Let's count the number of comparisons ***as a function of*** $n$:

- 1 iteration of the inner loop: 1 comparison
- $i$th iteration of the outer loop: $\sum_{j=i+1}^{n} 1 = n - i$ comparisons
- All iterations of the outer loop:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n} (n - i) = \boxed{\frac{n(n-1)}{2} \text{ comparisons}}$$

- This doesn't depend on the input values in $A[1..n]$.

Is that better or worse than $n^2$? $\frac{n^2}{100} + 5n$? $1000n$? $n^{3/2}$?

What does this tell us about the actual running time of the algorithm?

***How long does it take?***

```
SELECTIONSORT(A[1..n]):
  for i ← 1 to n
      for j ← i + 1 to n
          if A[j] < A[i]
              swap A[i] ↔ A[j]
```

The computer science answer:

**SELECTIONSORT *runs in* $\Theta(n^2)$ *time.***

Next time we'll see what this actually means.