

# Pseudocode

Set  $i = 0$

While  $i < 10$

1. print [ $i$  "bottles of beer on a stone"]
2. print [ $i$  "bottles of beer"]
3. print ["if one of those bottles should  
happen to clone"]
4. print [ $i+1$  "bottles of beer on a stone"]
5. INCREMENT  $i$  ( $i = i+1$  or " $i++$ ")

# Pseudocode

GCD (a, b)

While remainder r is not 0

1. find q and r so that  $a = bq + r$

2. set  $a = b$

    set  $b = r$

    continue ....

Output a

# stepping through code

- for understanding
- for debugging

GCD (a, b)

While remainder r is not 0

1. find q and r so that  
 $a = bq + r$

2. set a = b  
set b = r

continue ....

Output a

	a	b	q	r
Initial values	42	15	?	?
after step 1	42	15	2	12
after step 2	15	12	2	12
step 1 (2nd time)	15	12	1	3
step 2 (2nd time)	12	3	1	3
step 1 (3rd time)	12	3	4	0
step 2 (3rd time)	3	0	4	0

# Binary Search

Assume sorted data  $M(0) \dots M(n)$

Binsearch ( $x$ )

put left, right fingers on 0 and  $n$

WHILE  $x$  not equal  $M(\text{left finger})$

set midpoint halfway between fingers

IF  $M(\text{midpoint}) < x$

THEN move left finger to midpoint

ELSE move right finger to midpoint

END WHILE

# Binary Search

Assume sorted data  $M(0) \dots M(n)$

Binsearch (x)

left = 0

right = n

midpoint = (left + right) / 2

WHILE x not equal M(midpoint)

    midpoint = (left + right) / 2

    IF  $M(\text{midpoint}) < x$  THEN left = midpoint

        ELSE right = midpoint

END WHILE

# stepping through binary search

	left	right	midpoint		ARRAY M( )
Initial values	0	16	?	searching for 33	M(0) = 3
compute midpt	0	16	8		M(1) = 4
move left	8	16	8		M(2) = 6
compute midpt	8	16	12		M(3) = 21
move right	8	12	12		M(4) = 24
compute midpt	8	12	10		M(5) = 27
move right	8	10	10		M(6) = 28
compute midpt	8	10	9		M(7) = 30
				M(8) = 31	
				M(9) = 33	
				M(10) = 39	
				M(11) = 44	
				M(12) = 45	
				M(13) = 48	
				M(14) = 50	
				M(15) = 55	
				M(16) = 57	

# Bubble Sort

Put left, right fingers on  $M(0)$ ,  $M(1)$

IF  $M(\text{left}) > M(\text{right})$  THEN swap them

Move left, right fingers to right

IF right finger =  $n$

THEN done with this pass, start again

BUT next time only go up to  $n-1$

Etc., for passes ending at  $n-2$ ,  $n-3$ , etc.

# Bubble Sort

```
FOR max = n-1 downto 1
  FOR left = 1 upto max
    right = left + 1
    IF M(left) > M(right) THEN swap them
```

**LET'S UNROLL THIS CODE  
TO MAKE SURE WE UNDERSTAND**

# Bubble Sort

```
FOR max = n-1 downto 1
```

```
  FOR left = 1 upto max
```

```
    right = left + 1
```

```
    IF M(left) > M(right) THEN swap them
```

```
Let max = n-1
```

```
  blah blah blah
```

```
Let max = n-2
```

```
  blah blah blah
```

```
... etc
```

```
Let max = 2
```

```
  blah blah blah
```

```
Let max = 1
```

```
  blah blah blah
```

There are  $n-1$  phases  
During a phase, what happens?

A nested FOR loop is executed

The nested loop moves from 1 to max  
and does a swap if necessary

Let's look at an example with  $n = 4$

# Bubble Sort

*FOR max = n-1 downto 1*

FOR left = 1 upto max

right = left + 1

IF M(left) > M(right) THEN swap them

*max = 3*

IF M(1) > M(2) swap them

IF M(2) > M(3) swap them

IF M(3) > M(4) swap them

*max = 2*

IF M(1) > M(2) swap them

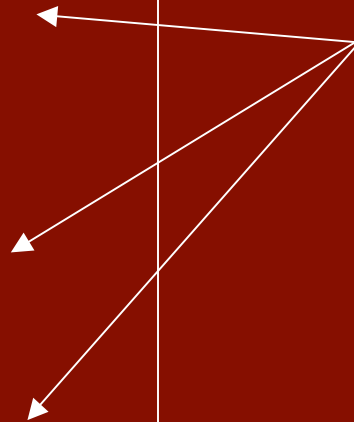
IF M(2) > M(3) swap them

*max = 1*

IF M(1) > M(2) swap them

Now we can see the effect of the inner loop for each "version" of the outer loop...

...and the overall effect of the nested loops



# Running time

- Count steps
- Don't get bogged down in detail
- Iteration is the biggest contributor
- Bubble Sort:  $(n-1) + (n-2) + \dots + 2 + 1$   
=  $(n/2) * n$   
=  $n^2/2$

# Insertion Sort

Insert first

Insert next into proper place

Insert next into proper place

etc.

# Insertion Sort

```
FOR i = 1 to n          /* Insert M(i) into correct place in new list T()
{
  j = 1                /* Start looking at position T(1)
  WHILE M(i) < T(j)
    { j = j+1
    }
                        /* AHA! M(i) belongs right after T(j)

  slide T(j+1), T(j+2), ... over to T(j+2), T(j+3)...
  T(j+1) = M(i)
}
```

where *slide* is a separate "procedure" that we have written.

# Running Times

$$\begin{aligned}\text{Insertion Sort: } & 1 + 2 + 3 + 4 \dots + n-1 \\ & = (n/2) * (n) \\ & = n^2/2\end{aligned}$$

## Quick Sort:

- depends on how lucky you are with "pivot"
- complex analysis shows on average, you are lucky
- running time on average can be shown to be about  $n \log n$

# Running Times

## Binary Search

- At each iteration, range is cut in half
- Total number of iterations to find an item among  $N$  elements is  $y$  such that
  - “cut  $N$  in half  $y$  times to get down to 1”
- Yes, this is just  $(\text{base } 2) \log N$
- Searching among  $2^{300}$  elements takes only 300 steps  
(this exceeds the number of particles in the universe!!)

# Running time matters

