4] Pseudocode: * Careful some lines may not be O(1)
eg A[n] = 0 // initialize all n to 0
$\in \Theta(n)$

5] A number a (base 10)
- requires $\sim \log a$ bits to represent if in binary

$8_{10} = 1000_2$
$\log a + 1$

a * b takes $O((\log a)(\log b))$ ~ school method
$O((\log a)(\log b)^{0.59})$ ~ efficient mult
- see later

Often set a word size (real world: 64 bits)
- elementary word size ops are then O(1) constant
  - add/sub, compare, read/write a word,

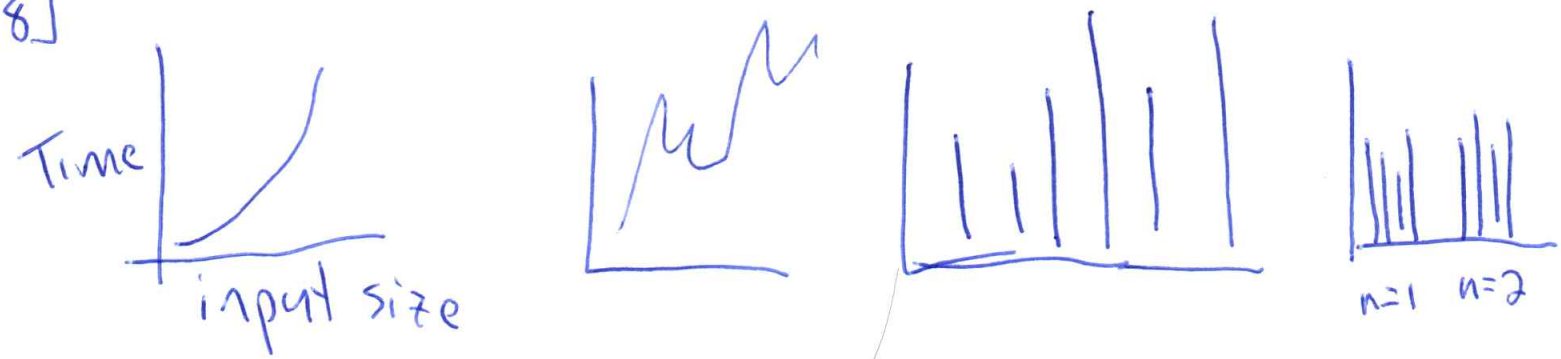
Numbers may still get big:

```
Product(A, n)
  S <- 1
  For i = 1 to n
    S = S * A[n]
```

what if A stores INT_MAX, what does C/C++ overflow O(n) do?

what if we allows to have as much space as needed? $O(n^2)$

8] Runtime



9] Random Algs => expected runtimes

$f(n) = 7n^2 + 13n + 27 \in O(n^2)$

$10^{100} n \in O(n)$

$2^{n+1} \in O(2^n)$

$(n+1)! \in O(n!)$? X $\in O(n+1)!$

$\log(n!) \in \Theta(n \log n)$

$(n-1)! \in O(n!)$? yes but also $o(n!)$ (little)
its $O((n-1)!)$

Challenge: If you have a RAM model and allow unlimited # bits for each memory location and change 1 for arithmetic and shifts, how can you "cheat" and sort in $O(n)$ steps!

Hint: Do all n^2 pairwise comparisons in 1 subtraction

Suppose Alg A is $O(n^2)$
Alg B is $O(n \log n)$

Which is faster?

- This is like $A \leq 5$, $y \leq 10$ which is smaller?

=> To compare Algs, need tight bounds

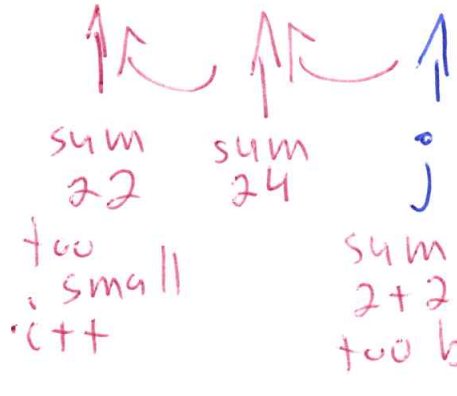
$\Theta(n^2)$ vs $\Theta(n \log n)$ better

2-Sum Alg 3

- sum too big j--
- sum too small i++

(sorted) Target: 23

A: 2 3 5 11 12 20 22 24



3-Sum

Faster algs?

For many years NO!

but now 2014, 2017.