More on the Cost of Computation
Examples from Last Time

- add m n (on naturals): \( m \) recursive calls
- append l_m l_n: \( m \) recursive calls
- Cross l_m l_n: \( m^n \) recursive calls
- map f l_n: \( n \) recursive calls, and calls to f
- find-root: \( n \log n \) recursive calls
- insert l_n: upto \( n \) recursive calls to insert
- insert-sort l_n: upto \( n^2/2 \) calls to insert
- maxi l_n: ? (Homework problem)
Order of Complexity

- Consider three algorithms
  - CostA(n) = 2*n^3 + n^2 + 1
  - CostB(n) = 3*n^2 + 10
  - CostC(n) = 2^n

- Which one is better?
Order of Complexity

Consider three algorithms

- CostA(n) = 2*n^3 + n^2 + 1
- CostB(n) = 3*n^2 + 10
- CostC(n) = 2^n

One with smaller dominator is best

Can we formalize this notion?
Order of Complexity

- We'll say that "CostZ is order $f(n)$" if
  - CostZ(n+offset) < factor * $f(n+offset)$
- More concisely: "CostZ is $O(f(n))$"
- Examples:
  - CostA(n) = 2*n^3 + n^2 + 1  CostA is $O(n^3)$
  - CostB(n) = 3*n^2 + 10  CostB is $O(n^2)$
  - CostC(n) = $2^n$  CostC is $O(2^n)$
- Essentially in the theory and practice of "Algorithms", and in "Complexity theory".
Famous "Complexity Classes"

- $O(1)$ constant-time (head, tail)
- $O(\log n)$ logarithmic (binary search)
- $O(n)$ linear (vector multiplication)
- $O(n \cdot \log n)$ "n log n" (sorting)
- $O(n^2)$ quadratic (matrix addition)
- $O(n^3)$ cubic (matrix multiplication)
- $n^{O(1)}$ polynomial (…many! …)
- $2^{O(n)}$ exponential (guess password)
- Are there more?
Plug for the lab: Vectors

- Speaking of cost...
  - Anything ever bother you about lists?
- Vectors
  - (build-vector N f)
  - (vector-ref V i)
  - (vector-length V)
  - (vector? e)
- Home work problem on vectors
Say we want to "integrate" a sequence:

- Given: [1,1,0,2,1]
- Return: [1,2,2,4,5]

How can we implement this?

How fast is our program?

Can we do any better?