Which of the following are correct?

A. Vectors can “grow”  
B. Arrays can “grow”  
C. They both can’t “grow”  
D. They both can “grow”  
E. Whatever
Administrative Details

• Lab 2
  • Only 8 more to go!

• Lab 3
  • This is a partner lab; you get to work in groups of 2.
  • Please complete PRE-LAB before lab
Agenda

- Measuring Growth (Big-O)
  - Recursion
Measuring Computational Cost

- How can we measure the amount of time needed to run a program?

- Compute $t$ of seconds
  - Difference in hardware
  - Input $x$
- Count # of operations
- Express in terms of input size
  $\Rightarrow$ “$n$” in the expression
Measuring Computational Cost

Consider these two code fragments...

1. Finding an element
   for (int i=0; i < arr.length; i++)
       if (arr[i] == x) return true;
   return false;

   \(\sim n\)
   \(= O(n)\)

2. Finding a pair of duplicate items
   for (int i=0; i < arr.length; i++)
       for (int j=0; j < arr.length; j++)
           if (i != j && arr[i] == arr[j]) return true;
   return false;

   \(\sim n^2\)
   \(= O(n^2)\)
Asymptotic Analysis (Big-O Analysis)

- A function $f(n)$ is $O(g(n))$ if and only if there exist positive constants $c$ and $n_0$ such that $|f(n)| \leq c \cdot g(n)$ for all $n \geq n_0$

- $g$ is “grows at least as fast as” $f$ for large $n$
  - Up to a multiplicative constant $c$
Determining “Best” Upper Bounds

• We typically want the smallest upper bound when we estimate running time

• Example: Let \( f(n) = 3n^2 \)
  - \( f(n) \) is \( O(n^2) \)
  - \( f(n) \) is \( O(n^3) \)
  - \( f(n) \) is \( O(2^n) \)
  - \( f(n) \) is NOT \( O(n) \) (!!)
  - “Best” upper bound is \( O(n^2) \)

\[ \Theta(n^2) \]
Function Growth & Big-O

- Rule of thumb: find the most *significant* or *dominant* term & ignore multiplicative constant

\[ f(n) = \sum_{i=0}^{k} a_in^{k-i} \]

\[ O(n^k) \]

\[ a_0n^k + a_1n^{k-1} + a_2n^{k-2} + \ldots + a_k \text{ is roughly } n^k \]
Asymptotic Analysis (Big-O Analysis)

• “How scalable is the algorithm?” as input size

• Commonly split into the following classes:
  • $O(1)$: “constant” = no loop through the input
  • $O(\log n)$: “logarithmic” or “log n”
  • $O(n)$: “linear” = 1 loop
  • $O(n \log n)$: “n log n”
  • $O(n^c)$: “polynomial”
    • $O(n^2)$: “quadratic” = 2 nested loops
    • $O(n^3)$: “cubic” = 3 nested loops
  • $O(c^n)$: “exponential”
Agenda

- Measuring Growth (Big-O)
- Recursion
Recursion

• General problem solving strategy
  • Break problem into smaller pieces
  • Sub-problems may look a lot like original - may in fact by smaller versions of same problem
• Examples
Recursion

• Many algorithms are recursive
  • Can be easier to understand (and prove correctness & state efficiency of) than iterative versions

loops
Factorial

Inductive
\[ n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1 \]

Recursive
\[ n! = n \cdot (n-1)! \]
\[ 0! = 1 \]