[TAP:PCOQD] Vector vs Array

- Which of the following are correct?
  - A. Vectors can "grow" = ~~extend~~ = get bigger = increase capacity
  - 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

download .jar

CLASSPATH=.

.bashrc

Agenda

- Measuring Growth (Big-O)
- Recursion

Measuring Computational Cost

efficiency [time / space]

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

- compute # of seconds
  - difference in hardware ✓
  - input ✓
- Count # of operations
- express in terms of input size

(=> "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;
```

$\approx n$

$= O(n)$

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;
```

$\approx n^2$

$= O(n^2)$

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

$$|f(n)| \leq \underline{c} \cdot g(n) \text{ for all } n \geq \underline{n_0}$$

- g is "grows at least as fast as" f **for large n**
  - Up to a multiplicative constant c

input size

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 $O(n^2)$ ✓ $3n^2 \leq c \cdot n^2$
  - $f(n)$ is $O(n^3)$ ✓ $3n^2 \leq n^3$
  - $f(n)$ is $O(2^n)$ ✓
  - $f(n)$ is NOT $O(n)$ (!!) $3n^2 \overset{?}{\leq} c \cdot n$
  - "Best" upper bound is $O(n^2)$

$\Theta(n^2)$

Function Growth & Big-O

- Rule of thumb: 1 find the most *significant* or *dominant* term & 2 ignore multiplicative constant
  - $f(n) = \underbrace{\cancel{a_0} n^k}\ \cancel{+ a_1 n^{k-1} + a_2 n^{k-2} + \dots a_k}$ is roughly n^k

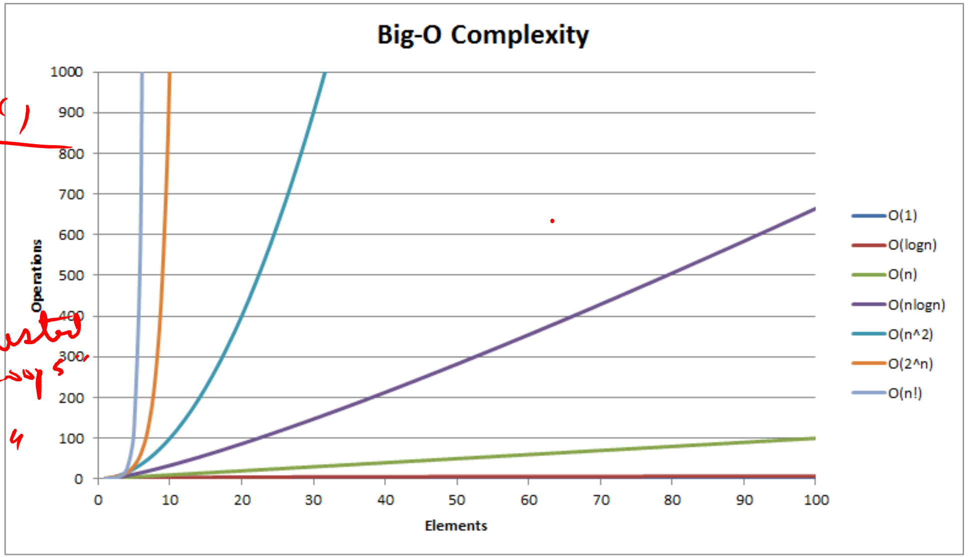
$= O(n^k)$

Asymptotic Analysis (Big-O Analysis)

- "How scalable is the algorithm?" as input size ↑
- Commonly split into the following *classes*:

"good"

  - O(1) : "constant" = "no loop through the input"
  - O(log n) : "logarithmic" or "log n"
  - O(n) : "linear" = "1 loop" e.g. indexOf()
  - 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"



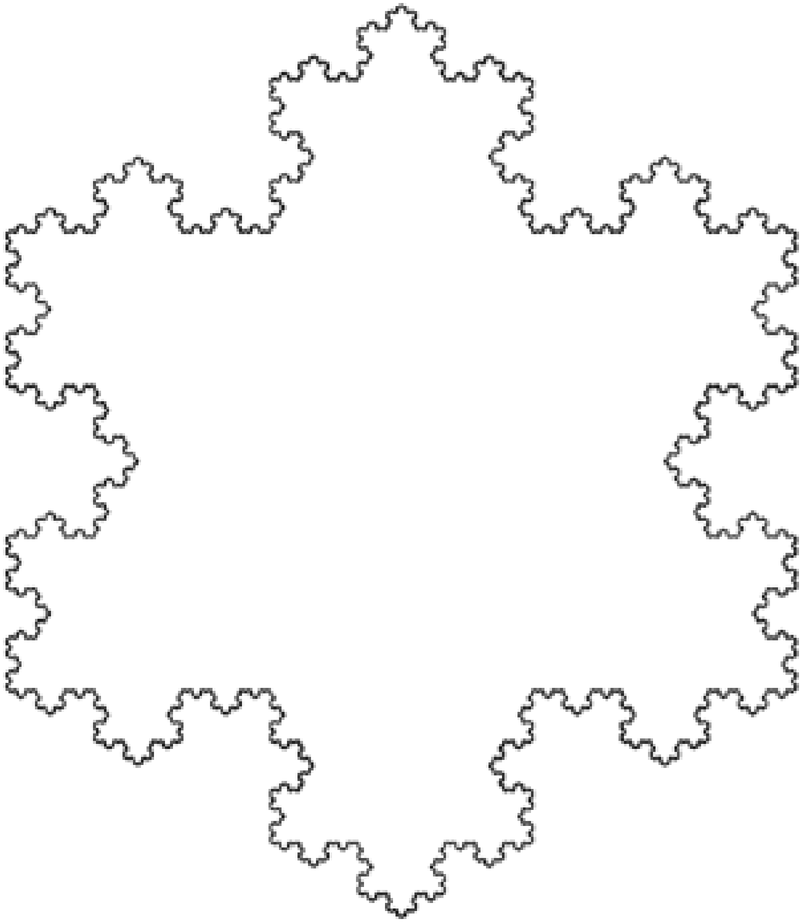
Agenda

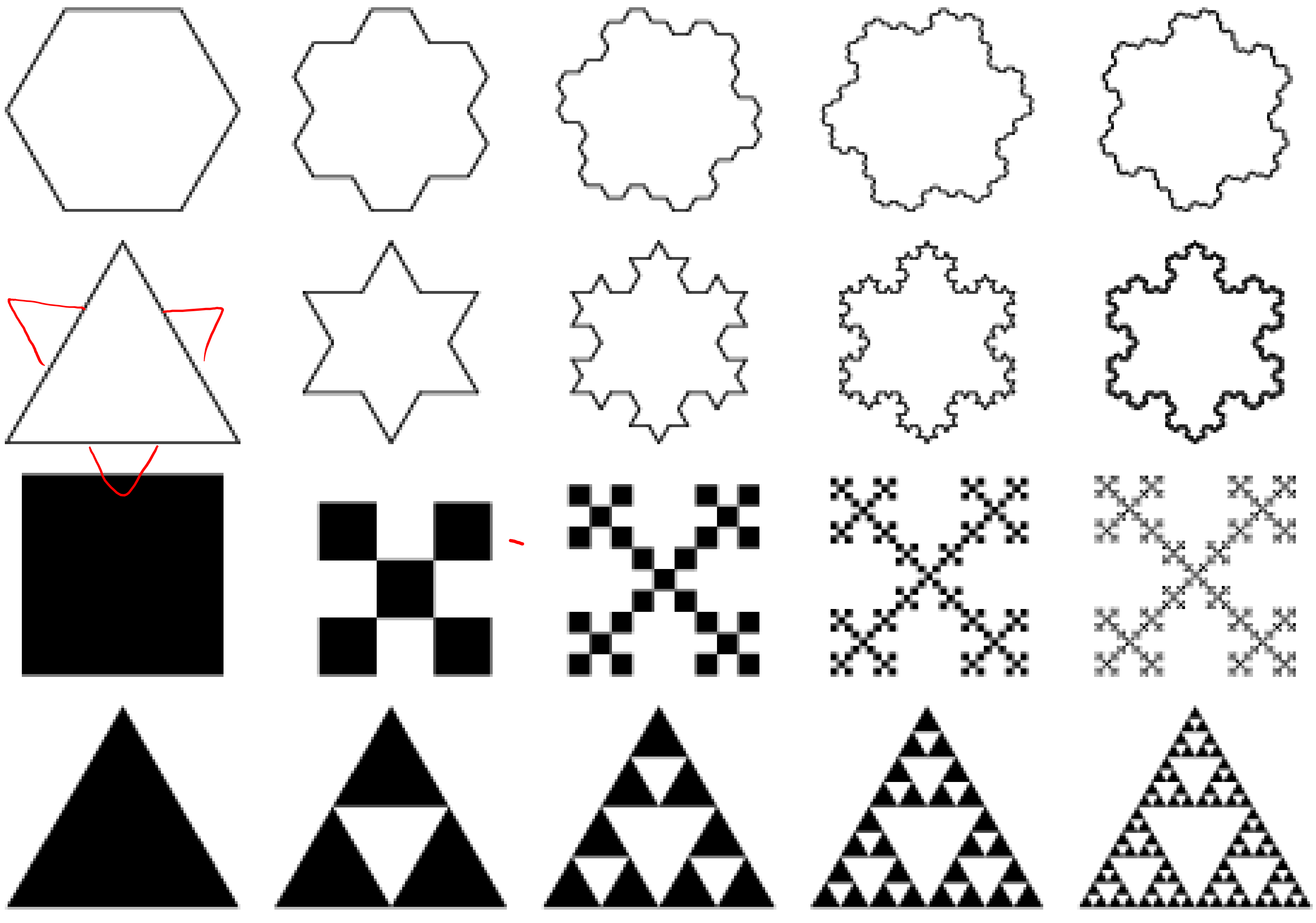
- Measuring Growth (Big-O)
- Recursion

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

fractal





Recursion

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

loops

Factorial

iteration

$n! = n \cdot n-1 \cdot n-2 \cdot \ldots \ldots 1$

recursive

$n! = n \cdot (n-1)!$

$0! = 1$