CSCI 136
Data Structures &
Advanced Programming

Lecture 6

Fall 2019

Instructors: Bill & Sam

Last Time

- Finished Java overview/review
- Introduction to Vectors
  - Example: Word Frequencies
  - Vector instance variable and method declarations
  - First details of implementation

Today's Outline

From Previous Lecture Slides

- Vector Implementation
- Miscellany: Wrappers
- Lab 2 Design and Strategies

Today's Slides

- Generic Data Types
- Condition Checking
  - Pre- and post-conditions, Assertions

Implementing Vectors

- A Vector holds an array of Objects
- Key difference is that the number of elements can grow and shrink dynamically
- How are they implemented in Java?
  - What instance variables do we need?
  - What methods? (start simple)
- Let's explore the implementation….

Class Vector : Instance Variables

```
public class Vector {
private Object[] elementData;     // Underlying array
protected int elementCount;       // Number of elts in Vector
protected final static int defaultCapacity;
protected int capacityIncrement;  // How much to grow by
protected Object initialValue;    // A default elt value
}
```

- Why Object[]?
  - Don't know the actual type of data
- Why elementCount?
  - size won't usually equal capacity
- Why capacityIncrement?
  - We'll "grow" the array as needed

Core Vector Methods

```
public class Vector {
   public Vector()  // Make a small Vector

   // Make Vector of given capacity
   public Vector(int initCap)

   // Add elt to (high) end of Vector
   public void add(Object elt)

   // Add elt at position I
   public void add(int i, Object elt)

   // Remove (and return) elt
   public Object remove(Object elt)

   // Remove (and return) elt at pos I
    public Object remove(int i)  //
```

Core Vector Methods

```
    public int capacity()   // Return capacity
    public int size()       // Return current size
    public boolean isEmpty()// Is size == 0?

    // Is elt in Vector?
    public boolean contains(Object elt)

    // Return elt at position I
    public Object get(int i)

    // Change value at position I
    public Object set(int i, Object elt)

    // Return earliest position of elt
    public int indexOf(Object elt)
}
```

Class Vector : Basic Methods

- Much work done by few methods:
  - indexOf(Object elt, int i)
    - Find first occurrance of elt at/after pos. I
    - Used by indexOf(Object elt)
    - remove methods use indexOf(Object elt)
  - firstElement(), lastElement() use get(int i)
- Method names/functions in spirit of Java classes
  - indexOf has same behavior as for Strings
- Methods are straightforward except when array is full
- How do we add to a full Vector?
  - We make a new, larger array and copy values to it

Extending the Array

- How should we extend the array?
- Possible extension methods:
  - Grow by fixed amount when capacity is reached
  - Double array when capacity is reached
- How could we compare the two techniques?
  - Run speed tests?
    - Hardware/system dependent
  - Count operations!
  - We'll do this soon

ensureCapacity

- How to implement `ensureCapacity(int minCapacity)`?

```
// post: the capacity of this vector is at least minCapacity
public void ensureCapacity(int minCapacity) {
   if (elementData.length < minCapacity) {
      int newLength = elementData.length; // initial guess
      if (capacityIncrement == 0) {
      // increment of 0 suggests doubling (default)
         if (newLength == 0) newLength = 1;
             while (newLength < minCapacity) {
               newLength *= 2;
             }
       } else {
      // increment != 0 suggests incremental increase
         while (newLength < minCapacity) {
             newLength += capacityIncrement;
         }
      }
```

```
  // assertion: newLength > elementData.length.
     Object newElementData[] = new Object[newLength];
     int i;

  // copy old data to array
     for (i = 0; i < elementCount; i++) {
        newElementData[i] = elementData[i];
     }

     elementData = newElementData;
        // garbage collector will pick up old elementData
  }
  // assertion: capacity is at least minCapacity
}
```

Notes About Vectors

- Primitive Types and Vectors

```
Vector v = new Vector();
v.add(5);
```

  - This (technically) shouldn't work! Can't use primitive data types with vectors…they aren't Objects!
  - Java is now smart about some data types, and converts them automatically for us -- called *autoboxing*
- We used to have to "box" and "unbox" primitive data types:

```
Integer num = new Integer(5);
v.add(num);
…
Integer result = (Integer)v.get(0);
int res = result.intValue();
```

- Similar wrapper classes (Double, Boolean, Character) exist for all primitives
  - Each has a valueOf() method to return primitive

Vector Summary & Notes

Vectors: "extensible arrays" that automatically manage adding elements, removing elements, etc.

1. Must cast Objects to correct type when removing from Vector
2. Use wrapper classes (with capital letters) for primitive data types (use "Integers" not "ints")
3. Define equals() method for Objects being stored for contains(), indexOf(), etc. to work correctly

A Vector-Based Dictionary (read on your own)

```
protected Vector defs;
public Dictionary() {
   defs = new Vector();
}

public void addWord(String word, String def) {
   defs.add(new Association(word, def));
}

// post: returns the definition of word, or "" if not found.
public String lookup(String word) {
   for (int i = 0; i < defs.size(); i++) {
       Association a = (Association)defs.get(i);
       if (a.getKey().equals(word)) {
           return (String)a.getValue();
       }
   }
   return "";
}
```

Dictionary.java

```
public static void main(String args[]) {
   Dictionary dict = new Dictionary();
   dict.addWord("perception", "Awareness of an object of
        thought");
   dict.addWord("person", "An individual capable of moral
        agency");
   dict.addWord("pessimism", "Belief that things generally
        happen for the worst");
   dict.addWord("philosophy", "Literally, love of
        wisdom.");
   dict.addWord("premise", "A statement whose truth is used to
        infer that of others");
}
```

Randomizing a Vector (discuss with a friend)

- How would we shuffle the elements of a Vector?
- `shuffle(Vector v)`
  - Many ways to implement.
  - An efficient way
    - Randomly move elements to “tail” of vector
    - Do this by swapping random element with last element
- swap is a little tricky
  - Three step process, not two!

Lab 2 Preview

- Three classes:
  - FrequencyList.java
  - Table.java
  - WordGen.java
- (eventually) Two Vectors of Associations
- toString() in Table and FrequencyList for debugging
- What are the key stages of execution?
  - Test code thoroughly before moving on to next stage
- Use WordFreq as example

Lab 2: Core Tasks

- FreqencyList
  - A Vector of Associations of String and Int
  - Add a letter
    - Is it a new letter or not?
    - Use indexOf from Vector class
- Pick a random letter based on frequencies
  - Let total = sum of frequencies in FL
  - generate random int r in range [0…total]
  - Find smallest k s.t. r <= sum of first k frequencies

Lab 2: Core Tasks

- Table
  - A Vector of Associations of String and FrequencyList
  - Add a letter to a k-gram
    - Is it a new k-gram or not?
  - Pick a random letter given a k-gram
    - Find the k-gram then ask its FrequencyList to pick
- WordGen
- Convert input into (very long) String
  - Use a StringBuffer---see handout

Using Generic (Parameterized) Types

- What limitations are associated with casting Objects as they are added and removed from Associations?
  - Errors cannot be detected by compiler
  - Must rely on runtime errors

- Instead of casting Objects, Java supports using generic or parameterized data types (Read Ch 4)
- Instead of:

```
Association a = new Association(“Bill”,(Integer) 97);
Integer grade = (Integer) a.getValue();  //Cast to String
```

- Use:

```
Association<String, Integer> a =
    new Association<String, Integer>(”Bill”, (Integer) 97);
    Integer grade = a.getValue();  //no cast!
```

Generic Association<K,V> Class

```
class Association<K,V> {
   protected K theKey;
   protected V theValue;

   //pre: key != null
   public Association (K key, V value) {
        Assert.pre (key != null, “Null key”);
        theKey = key;
        theValue = value;
   }

   public K getKey() {return theKey;}
   public V getValue() {return theValue;}
   public V setValue(V value) {
        V old = theValue;
        theValue = value;
        return old;
   }
}
```

Using Generic Data Types

- Instead of casting Objects, Java supports using generic or parameterized data types (Read Ch 4)
  - Instead of:

```
Vector v = new Vector();   //Vector of Objects
String word = (String)v.get(index);   //Cast to String
```

  - Use:

```
Vector<String> v = new Vector<String>(); //Vector of Strings
String word = v.get(index);   //no cast!
```

  - Or:

```
Vector<Association<String, Integer>> v =
  new Vector<Association<String, Integer>>();
int count = v.get(index).getValue(); //no cast!
```

  - See GenWordFreq.java…

(Look at WordFreq.java with gen)

Class Vector<E>

```
public class Vector<E> {
private Object[] elementData;     // Underlying array
protected int elementCount;       // Number of elts in Vector
protected final static int defaultCapacity;
protected int capacityIncrement;  // How much to grow by
protected E initialValue;         // A default elt value
}
```

- Why (still!) Object[]?
  - Java restriction: Can't use a type variable for an array declaration, only a concrete type

Basic Vector<E> Methods

```
public class Vector<E> {
public Vector()              // Make a small Vector
public Vector(int initCap) // Make Vector of given capacity
public void add(E elt)       // Add elt to (high) end of Vector
public void add(int i, E elt)      // Add elt at position i
public E remove(E elt)       // Remove (and return) elt
public E remove(int i)       // Remove (and return) elt at pos i
public int capacity()        // Return capacity
public int size()            // Return current size
public boolean isEmpty()     // Is size == 0?
public boolean contains(E elt) // Is elt in Vector?
public E get(int i)          // Return elt at position i
public E set(int i, E elt) // Change value at position i
public int indexOf(E elt)    // Return earliest position of elt
}
```

Pre and Post Conditions

- Recall `charAt(int index)` in Java String class
- What are the pre-conditions for charAt?
  - 0 <= index < length()
- What are the post-conditions?
  - Method returns char at position index in string
- We put pre and post conditions in comments above most methods

```
/* pre: 0 ≤ index < length
 * post: returns char at position index
 */
public char charAt(int index) { … }
```

Pre and Post Conditions

- Pre and post conditions “form a contract”
- Post-condition is guaranteed if method is called when pre-condition is true
- Examples:
  - `s.charAt(s.length() - 1)`: index < length, so valid
  - `s.charAt(s.length() + 1)`: index > length, not valid
- These conditions document requirements that user of method should satisfy
- But, as comments, they are not enforced

Other Examples

- Other places pre and post conditions are useful

```
// Pre: other is of type Card
// Post: Returns true if suits and ranks match
public boolean equals(Object other) {
   Card oc = (Card) other;
   return this.getRank() == oc.getRank() &&
          this.getSuit() == oc.getSuit();
}
```

Assert Class

- Pre- and post-condition comments are useful as a programmer, but it would be *really* helpful to know as soon as a pre-condition is violated (and return an error)

- The Assert class (in structure5 package) allows us to programmatically check for pre- and post-conditions

Assert Class

The Assert class contains the methods

```
public static void pre(boolean test, String message);
public static void post(boolean test, String message);
public static void condition(boolean test, String message);
public static void fail(String message);
```

If the boolean test is NOT satisfied, an exception is raised, the message is printed and the program halts

Assert Examples

The Vector class uses Assert in a many places

```
// Pre: initialCapacity >= 0
public Vector(int initialCapacity) {
      Assert.pre(initialCapacity >= 0,"Capacity
            must not be negative");

// Pre: 0 <= index && index < size()
public E elementAt(int index) {
      Assert.pre(0 <= index && index < size(),"index
            is within bounds");
```

General Rules about Assert

1. State pre/post conditions in comments
2. Check conditions in code using "Assert"
3. Use Fail in unexpected cases (such as the default block of a switch statement)

- Any questions?
- You can use Assertions in Lab 2

The Java assert keyword

- An alternative to Duane's Assert class
- Added in Java 1.4
- Two variants
  - assert boolean_expression
    - Throws an AssertionError if the expression is false
  - assert boolean_expression : other_expression
    - In addition, prints value of other_expression\

Measuring Computational Cost

Consider these two code fragments…

```
for (int i=0; i < arr.length; i++)
   if (arr[i] == x) return "Found it!";
```

…and…

```
for (int i=0; i < arr.length; i++)
   for (int j=0; j < arr.length; j++)
     if( i !=j && arr[i] == arr[j]) return "Match!";
```

How long does it take to execute each block?

Measuring Computational Cost

- How can we measure the amount of work needed by a computation?
  - Absolute clock time
    - Problems?
      - Different machines have different clocks
      - Too much other stuff happening (network, OS, etc)
      - Not consistent. Need lots of tests to predict future behavior

Measuring Computational Cost

- Counting computations
  - Count *all* computational steps?
  - Count how many "expensive" operations were performed?
  - Count number of times "x" happens?
    - For a specific event or action "x"
    - i.e., How many times a certain variable changes
- Question: How accurate do we need to be?
  - 64 vs 65? 100 vs 105? Does it really matter??

An Example

```
// Pre: array length n > 0
public static int findPosOfMax(int[] arr) {
       int maxPos = 0 // A wild guess
       for(int i = 1; i < arr.length; i++)
              if (arr[maxPos] < arr[i]) maxPos = i;
       return maxPos;
}
```

- Can we count steps exactly?
  - ”if” makes it hard
- Idea: Overcount: assume “if” block always runs
- Overcounting gives *upper bound* on run time
- Can also undercount for lower bound
- Overcount: 4(n-1) + 4; undercount: 3(n-1) + 4

Measuring Computational Cost

- Rather than keeping exact counts, we want to know the *order of magnitude* of occurrences
  - 60 vs 600 vs 6000, *not* 65 vs 68
  - n, *not* 4(n-1) + 4
- We want to make comparisons without looking at details and without running tests
- Avoid using specific numbers or values
- Look for overall trends

Measuring Computational Cost

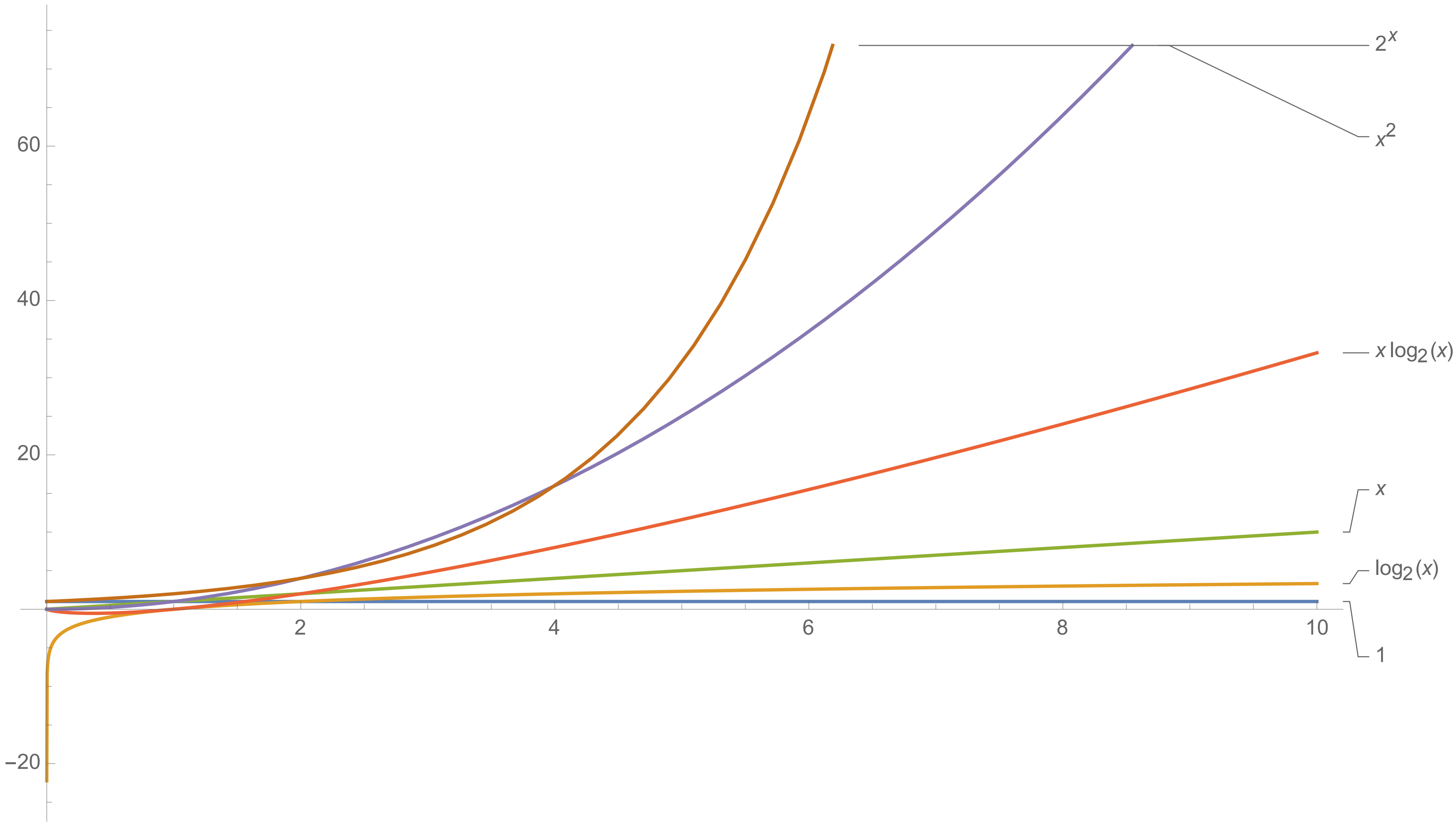
- How does algorithm scale with problem size?
  - E.g.: If I double the size of the problem instance, how much longer will it take to solve:
    - Find maximum: $n - 1 \rightarrow (2n) - 1$ (≈ twice as long)
    - Bubble sort: $n(n-1)/2 \rightarrow 2n(2n - 1)/2$ (≈ 4 times as long)
    - Subset sum: $2^{n-1} \rightarrow 2^{2n-1}$ (2^n times as long!!!)
    - Etc.
- We will also measure amount of space used by an algorithm using the same ideas….

Function Growth

Consider the following functions, for $x \geq 1$

- $f(x) = 1$
- $g(x) = \log_2(x)$ // Reminder: if x=2^n, $\log_2(x) = n$
- $h(x) = x$
- $m(x) = x \log_2(x)$
- $n(x) = x^2$
- $p(x) = x^3$
- $r(x) = 2^x$

Function Growth



Function Growth & Big-O

- Rule of thumb: ignore multiplicative constants
- Examples:
  - Treat n and n/2 as same order of magnitude
  - $n^2/1000$, $2n^2$, and $1000n^2$ are "pretty much" just n^2
  - $a_0n^k + a_1n^{k-1} + a_2n^{k-2} + \cdots a_k$ is roughly n^k
- The key is to find the most *significant* or *dominant* term
- Ex: $\lim_{x\to\infty} (3x^4 - 10x^3 - 1)/x^4 = 3$ (Why?)
  - So $3x^4 - 10x^3 - 1$ grows "like" x^4

Asymptotic Bounds (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

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

- $c \cdot g$ is "at least as big as" f **for large n**
  - Up to a multaplicative constant c!
- Example:
  - $f(n) = n^2/2$ is $O(n^2)$
  - $f(n) = 1000n^3$ is $O(n^3)$
  - $f(n) = n/2$ is $O(n)$

Determining "Best" Upper Bounds

- We typically want the *most conservative* upper bound when we estimate running time
  - And among those, the *simplest*
- Example: Let $f(n) = 3n^2$
  - $f(n)$ is $O(n^2)$
  - $f(n)$ is $O(n^3)$
  - $f(n)$ is $O(2^n)$ (see next slide)
  - $f(n)$ is NOT $O(n)$ (!!)
- "Best" upper bound is $O(n^2)$
- We care about **c** and $\mathbf{n_0}$ in practice, but focus on size of **g** when designing algorithms and data structures

What's n_0? Messy Functions

- Example: Let $f(n) = 3n^2 - 4n + 1$. f(n) is $O(n^2)$
  - Well, $3n^2 - 4n + 1 \le 3n^2 + 1 \le 4n^2$, for $n \ge 1$
  - So, for $c = 4$ and $n_0 = 1$, we satisfy Big-O definition
- Example: Let $f(n) = n^k$, for any fixed $k \ge 1$. f(n) is $O(2^n)$
  - Harder to show: Is $n^k \le c\,2^n$ for some $c > 0$ and large enough n?
  - It is if and only if $\log_2(n^k) \le \log_2(2^n)$, that is, iff $k \log_2(n) \le n$.
  - That is iff $k \le n/\log_2(n)$. But $n/\log_2(n) \to \infty$ as $n \to \infty$
  - This implies that for some n_0 on $n/\log_2(n) \ge k$ if $n \ge n_0$
  - Thus $n \ge k \log_2(n)$ for $n \ge n_0$ and so $2^n \ge n^k$

Input-dependent Running Times

- Algorithms may have different running times for different input values
- Best case (typically not useful)
  - Sort already sorted array in O(n)
  - Find item in first place that we look O(1)
- Worst case (generally useful, sometimes misleading)
  - Don't find item in list O(n)
  - Reverse order sort $O(n^2)$
- Average case (useful, but often hard to compute)
  - Linear search O(n)
  - QuickSort random array O(n log n) ← We'll sort soon

Vector Operations : Worst-Case

For n = Vector size (*not* capacity!):

- O(1): size(), capacity(), isEmpty(), get(i), set(i), firstElement(), lastElement()
- O(n): indexOf(), contains(), remove(elt), remove(i)
- What about add methods?
  - If Vector doesn't need to grow
    - add(elt) is O(1) but add(elt, i) is O(n)
  - Otherwise, depends on ensureCapacity() time
    - Time to compute newLength : $O(\log_2(n))$
    - Time to copy array: O(n)
    - $O(\log_2(n)) + O(n)$ is O(n)

Vector: Add Method Complexity

Suppose we grow the Vector's array by a fixed abount d. How long does it take to add n items to an empty Vector?

- The array will be copied each time its capacity needs to exceed a multiple of d
  - At sized 0, d, 2d, …, n/d
- Copying an array of size kd takes ckd steps for some constant c, giving a total of

$$\sum_{k=1}^{n/d} c \cdot k \cdot d = c \cdot d \sum_{k=1}^{n/d} k = c \cdot d \cdot \frac{\left(n/d\right)\left(n/d + 1\right)}{2} = O(n^2)$$

Vector: Add Method Complexity

Suppose we want to grow the Vector's array by doubling. How long does it take to add *n* items to an empty Vector?

- The array will be copied each time it's capacity needs to exceed a power of 2.
  - At sizes 0, 1, 2, 4, 8, …, $2^{\log_2 n}$
- Copying an array of size 2^k takes $c2^k$ steps for some constant c, giving a total of:

$$\sum_{k=1}^{\log_2 n} c \cdot 2^k = c \sum_{k=1}^{\log_2 n} 2^k = c \cdot (2^{1+\log_2 n} - 1) = O(n)$$

Common Complexities

For n = measure of problem size:

- $O(1)$: constant time and space
- $O(\log n)$: divide and conquer algorithms, binary search
- $O(n)$: linear dependence, simple list lookup
- $O(n \log n)$: divide and conquer sorting algorithms
- $O(n^2)$: matrix addition, selection sort
- $O(n^3)$: matrix multiplication
- $O(n^k)$: cell phone switching algorithms
- $O(2^n)$: subset sum, graph 3-coloring, satisfiability, ...
- $O(n!)$: traveling salesman problem (in fact $O(n^2 2^n)$)