Lecture 6: Complexity

CS 62
Spring 2015
Kim Bruce & America Chambers

Lab This Week

- Timing ArrayList operations
  - Encourage working in pairs
  - Stopwatch class: start(), stop(), getTime(), reset()
- Java has Just-In-Time compiler
  - Must "warm-up" before you get accurate timing
  - What can mess up timing?
- Uses Vector from Bailey rather than ArrayList from Java libraries because can change way it increases in size.

Programming Assignment This Week

- Weak AI/Natural Language Processing:
  - Generate text by building frequency lists based on pairs of words. ArrayList of Associations of String (words) and Integer (count of that word).

Order of Magnitude

- Definition: We say that $g(n)$ is $O(f(n))$ if there exist two constants C and k such that $|g(n)| <= C\ |f(n)|$ for all $n > k$.
- Examples: $2n+1$, n^3-n^2+83, 2^n+n^2
- Used to measure time and space complexity of algorithms on data structures of size n.
- Most common are
  - $O(1)$ - for any constant
  - $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, …, $O(2^n)$

Use simplest version in O(...)

Complexity

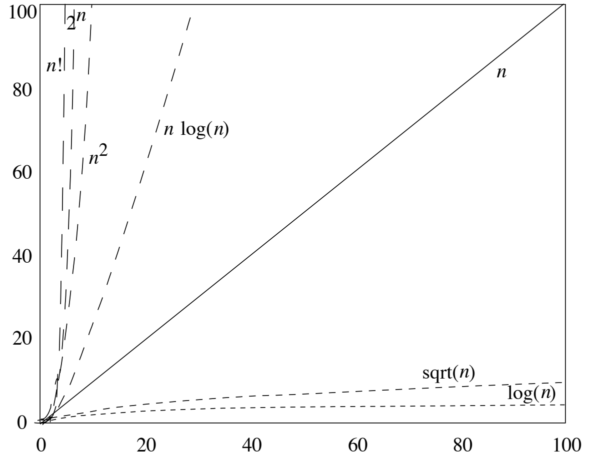

Figure 5.3 Long-range trends of common curves. Compare with Figure 5.2.

Comparing Orders of Magnitude

- Suppose have ops w/complexities given & problem of size n taking time t.
- How long if increase size of problem?

Problem Size:	*10 n*	*100n*	*1000n*
O(log n)	3+t	7 + t	10+ t
O(n)	10 t	100 t	1000 t
O(n log n)	> 10 t	> 100 t	> 1000 t
$O(n^2)$	100 t	10,000 t	1,000,000 t
$O(2^n)$	$\sim t^{10}$	$\sim t^{100}$	$\sim t^{1000}$

Adding to ArrayList

- Suppose n elements in ArrayList and add 1.
- If space:
  - Add to end is O(1)
  - Add to beginning is O(n)
- If not space,
  - What is cost of ensureCapacity?
  - O(n) because n elements in array

EnsureCapacity

- What if only increase in size by 1 each time?
  - Adding n elements one at a time to end
    - Total cost of copying over arrays: $1+2+3+\ldots+(n-1) = n(n-1)/2$
    - Total cost of $O(n^2)$
  - Average cost of each is O(n)
- What if double in size each time?
  - Suppose add $n = 2^m$ new elts to end
    - Total cost of copying over arrays: $1+2+4+\ldots+n/2 = n-1$, O(n)
    - Average cost of O(1), but “lumpy”

ArrayList Ops

- Worst case
  - O(1): size, isEmpty, get, set
  - O(n): remove, add
- Add to end, on average O(1)

Complexity

- 1 + 2 + ... + n comes up often in complexity
  - E.g. selection and insertion sorts
  - 1 + 2 + ... + n = n(n+1)/2
  - Similarly, 1 + 2 + ... (n-1) = n(n-1)/2
  - Proof by cleverness
    - or mathematical induction

Proof-by-induction

- Induction key to understanding recursion
  - and lots of other things
- To prove P(i) for all i ≥ 0
  - Base case: Prove P(0)
  - Induction case: Let k ≥ 0 and assume P(k) is true Use assumption to prove P(k+1).

Selection Sort (helper)

```
/*
 * Return index of smallest number in array between
 * startIndex and array.length.
 * PRE: startIndex must be valid index for array
 * POST: returns index of smallest value in range
        startIndex - array.length
 */
int indexOfSmallest(int[] array, int startIndex) {
  int smallIndex = startIndex;
  for (int i = startIndex + 1; i < array.length; i++) {
    if (array[i] < array[smallIndex]) {
      smallIndex = i;
    }
  }
  return smallIndex;
}
```

Selection Sort (helper)

```
/*
   * PRE: startIndex must be valid index for array
   * POST: Sorts array from startIndex to array.length.
   */
  void selectionSort(int[] array, int startIndex) {
    if (startIndex < array.length - 1) {
      // find smallest element in rest of array
      int smallest = indexOfSmallest(array, startIndex);

      // move smallest to index startIndex
      swap(array, smallest, startIndex);

      // sort everything in the array after startIndex
      selectionSort(array, startIndex + 1);
    }
  }
```

Analysis

- Count number of comparisons of elts of array
  - All comparisons are in “indexOfSmallest”
    - At most n-1 if startIndex ... array.length-1 has n elements.
  - Prove # of comparisons in selection sort of array of size n is 1 + 2 + ... + (n-1).
    - Base case: n = 0 or n = 1: No comparisons
    - Assume true for startIndex ... array.length has n-1 elements
    - Show for n elements.