Lecture 6: Complexity

Fall 2016

Kim Bruce & Peter Mawhorter

Lab this Week

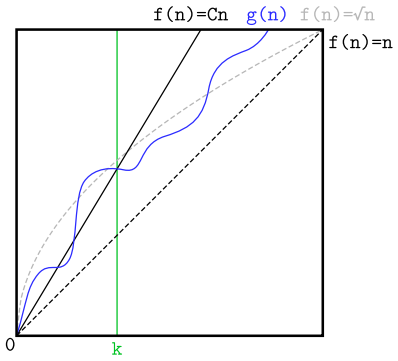
- Timing ArrayList operations
  - Encourage working in pairs
  - Stopwatch class: `start()`, `stop()`, `getTime()`, `reset()`
- Java has just-in-time compiler
  - Must "warm-up" before you get accurate timing.
  - What can mess up timing?
- Uses Vector from Bailey rather than ArrayList for control over growth policy.

Assignment this Week

- Weak AI / Natural Language Processing
  - Generate text by counting word pairs.
  - ArrayList of Associations of String (words) and Integer (count of that pair).

Order of Magnitude

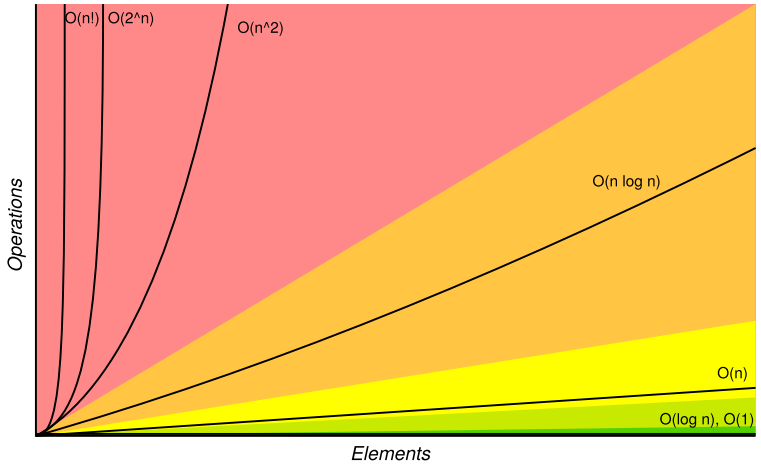
- Definition: A function $g(n)$ is in $O(f(n))$ if there exist two constants C and k such that $|g(n)| \leq C|f(n)|$ for all $n > k$.



$|g(n)| \leq C|f(n)|$ where $n > k$

Order of Magnitude

- Used to measure time and space complexity of algorithms and data structures.
- Examples:
  - $2n+1$ is $O(n)$
  - $n^3 + 10000n^2 + 10000$ is $O(n^3)$
  - $2^n + n^{17}$ is $O(2^n)$
- Most common:
  - $O(1)$ for any constant
  - $O(\log_2 n)$, $O(n)$, $o(n \log_2 n)$, …, $O(2^n)$, $O(n!)$



Common examples diverge quickly.

Comparing Orders of Magnitude

- If processing 5 elements takes 1 second, 50 will take:

O	Time (approx.)
$O(\log_2 n)$	< 2.5 seconds
$O(n)$	< 10 seconds
$O(n^2)$	< 100 seconds
$O(2^n)$	< 1 million years

What about 500 elements?

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)
- Otherwise:
  - What is the 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 = $1 + 2 + 3 + \ldots + (n\text{-}1) = n(n\text{-}1)/2$
    - This is in $O(n^2)$
  - Average cost is O(n)
- What if double size each time?
  - Adding n elements at end:
    - Total cost of copying = $1 + 2 + 4 + \ldots + n/2 = n\text{-}1 \rightarrow O(n)$
  - Average cost is O(1) but it's "lumpy"

ArrayList Ops

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

Sums

- $1 + 2 + \ldots + n$ comes up often in complexity
  - E.g., selection and insertion sorts
  - $1 + 2 + \ldots + n = n(n+1)/2$
  - Similarly, $1 + 2 + \ldots (n\text{-}1) = (n\text{-}1)n/2$
  - Proof by induction:

Proof by induction

- Induction is key to understanding recursion
  - It's like splitting a program into functions and writing one at a time.
- To prove P(i) for all $i \geq 0$
  1. Prove that P(0).
  2. Let $k \geq 0$ and prove that P(k+1) if P(k)

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
 */
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

```
/*
 * PRE: startIndex must be valid index for array
 * POST: Array is sorted from startIndex -- array.length.
 */
int 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 after startIndex
    selectionSort(array, startIndex + 1);
  }
}
```

Analysis

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