Lecture 7: Induction & Sorting
Fall 2016
Kim Bruce & Peter Mawhorter

Qualitative Skills Center
- [www.pomona.edu/qsc](http://www.pomona.edu/qsc)
- Can set up 1-on-1 tutoring or just drop in for help.
- Dedicated tutoring available for CS062.
- You can see the schedule and make appointments online.

Reading
- JS § 5.2 covers recursion/induction
- JS § 5.3 has some design guidelines
- JS Ch. 6 covers sorting

Quiz on Friday!
- ArrayLists and “big-O” notation.
Lab Today

- 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?
- Vector constructor:
  ```java
  public Vector(int initialCapacity, int capacityIncrement)
  ```

Induction

- A mathematical technique for proving:
  - mathematical statements over natural numbers
  - the correctness of algorithms
- Intimately related to recursion
  - Inductive proofs reference themselves

Induction

- Let P(n) be some proposition.
- To prove that P(n) is true for all \( n \geq 0 \):
  1. Base case: prove that P(n) for \( n = 0 \)
  2. Assume P(n) is true for some \( n = k, k \geq 0 \)
  3. Using (2), prove P(n) for \( n = k + 1 \)
**Induction**

- P(n) ⇒ “The n\textsuperscript{th} domino will fall over.”
- To prove that all of the dominoes will fall over:
  1. Base case: the first domino will fall over (we will push it)
  2. Assume that the k\textsuperscript{th} domino is falling over.
  3. Therefore the k+1\textsuperscript{st} domino will fall over (the k\textsuperscript{th} will hit it)

Conclusion: for every $n \geq 0$, the n\textsuperscript{th} domino will fall.

**Selection Sort**

1. Take the smallest element
2. Swap it with the first element
3. Repeat with the rest of the array
Selection Sort

```c
/*
 * XXX startIndex must be valid index for array
 * MUST: Array is sorted from startIndex == arrayLength. 
 */
int selectionSort(int[] array, int startIndex) {
    // find smallest element in rest of array
    int smallest = findMin(array, startIndex);
    // move smallest to index startIndex
    swap(array, smallest, startIndex);
    // sort everything after startIndex
    selectionSort(array, startIndex + 1);
}
```

Correctness

Can we prove that our algorithm works?
(Use induction)
What must be true after each step?

Selection Sort (helper)

```c
/*
 * Return index of smallest number in array between
 * startIndex and arrayLength.
 * XXX startIndex must be valid index for array
 * MUST: returns index of smallest value in range
 */
int indexOfMin(array[] array, int startIndex) {
    int smallestIndex = startIndex;
    for (int i = startIndex; i < array.length; i++) {
        if (array[i] < array[smallestIndex]) {
            smallestIndex = i;
        }
    }
    return smallestIndex;
}
```

Complexity

Can we prove that our algorithm works quickly?
How many operations does each indexOfSmallest take?
Strong Induction

- Instead of just assuming \( P(k) \) and proving \( P(k+1) \)...
- Assume \( P(k) \) for all \( 0 \leq k < n \) to prove \( P(n) \)

Fast Exponentiation

- \( \text{fastPower}(x, n) \) calculates \( x^n \):
  - if \( n = 0 \), return 1
  - if \( n \) is even, return \( \text{fastPower}(x^2, n/2) \)
  - if \( n \) is odd, return \( x \cdot \text{fastPower}(x, n-1) \)

- Proof by induction...
  - Base case: \( n = 0 \)
  - Assumption: assume \( \text{fastPower}(x, k) \) is \( x^k \) for all \( k < n \).
  - Inductive case: show \( \text{fastPower}(x, n) \) is \( x^n \)