Lecture 7: Induction & Sorting

Fall 2016

Kim Bruce & Peter Mawhorter

Qualitative Skills Center

- 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:

```
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

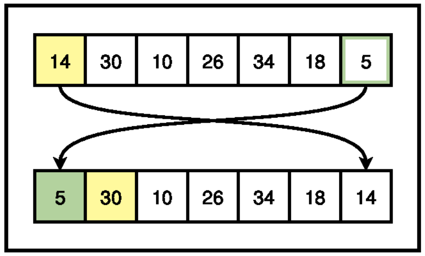
- P(n) ⇒ "The n^{th} domino will fall over."
- To prove that all of the dominoes will fall over:
  1. Base case: the first domino will fall over *(we will push it)*
  2. Assume that the k^{th} domino is falling over.
  3. Therefore the $k+1^{st}$ domino will fall over *(the k^{th} will hit it)*

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

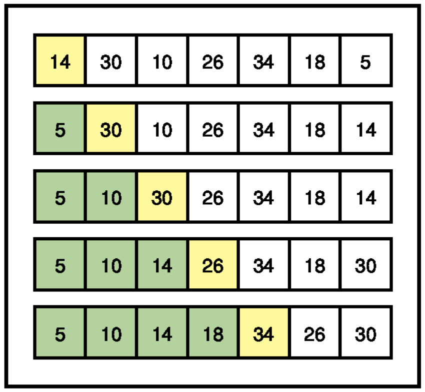
Induction

- Can be used to verify sum identities:
  - $1 + 2 + \ldots + n \Rightarrow n(n+1)/2$
  - $1 + 2 + 4 + \ldots + n/2 + n \Rightarrow 2n$

Selection Sort



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



Selection sort progress.

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);
  }
}
```

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;
}
```

Correctness

Can we prove that our algorithm works?

(use induction)

What must be true after each step?

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

- `fastPower`(x, n) calculates x^n:
  - if $n == 0$, return 1
  - if n is even, return `fastPower`(x*x, n/2)
  - if n is odd, return x*`fastPower`(x, n-1)
- Proof by induction…
  - Base case: $n == 0$
  - Assumption: assume `fastPower`(x, k) is x^k for all k < n.
  - Inductive case: show `fastPower`(x, n) is x^n