Lecture 8: Induction and Sorting

CS 62

Fall 2017

Kim Bruce & Alexandra Papoutsaki

Reading

- Chapter 5.2 covers recursion/induction
- Chapter 5.3 has some design guidelines
- Chapter 6 covers sorting

Induction

- Mathematical technique for proving:
  - Mathematical statements over natural numbers
  - Complexity (big-o) of algorithm
  - The correctness of algorithms

- Intimately related to recursion
  - Inductive proofs reference themselves

Induction steps

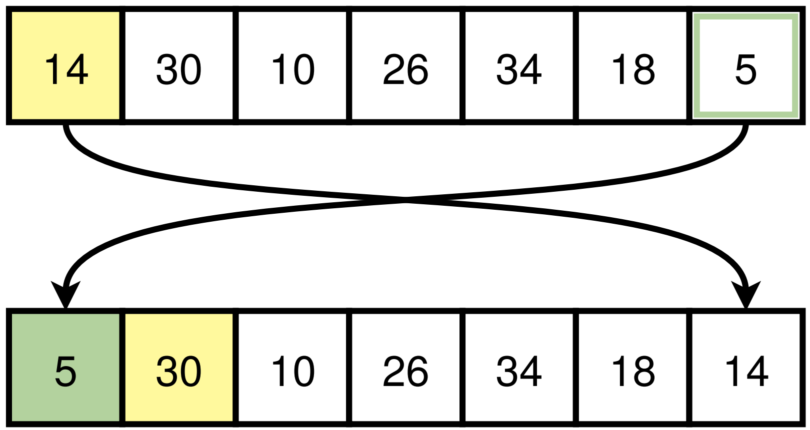
- Let $P(n)$ be some proposition

- To prove $P(n)$ is true for all $n \geq 0$
  - (Step 1) Base case: Prove $P(0)$
  - (Step 2) Assume $P(k)$ is true for $k \geq 0$
  - (Step 3) Use this assumption to prove $P(k + 1)$

Practice Examples

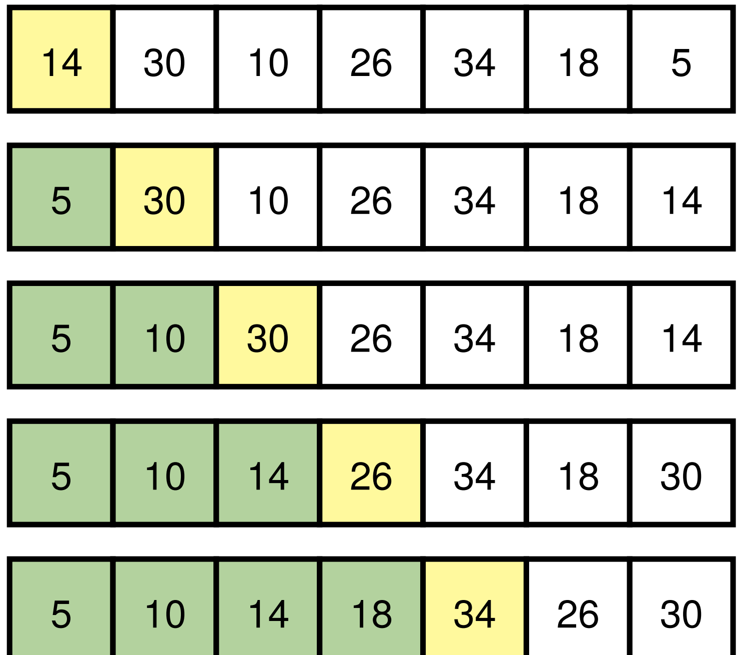
- Prove $1 + 2 + \ldots + n = [n(n+1)]/2$ for all $n \geq 1$
- Prove $2^0 + 2^1 + \ldots + 2^n = 2^{n+1} - 1$ for all $n \geq 0$
- Prove $2^n < n!$ for all $n \geq 4$

Selection Sort



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

Selection Sort



Selection Sort (helper)

```
/*
* @param array array of integers
* @param endIndex valid index into array
* @return index of largest value in array[0...endIndex]
*/
private int indexofLargest[] array, int endIndex) {
  int largestIndex = 0;
  for (int i = largestIndex + 1; i < endIndex; i++) {
    if (array[i] > array[largestIndex ]) {
      largestIndex = i;
    }
  }
  return largestIndex ;
}
```

Selection Sort

```
/**
 * @param array array of integers
 * @param endIndex a valid index into array
 */
private static void selectionSortRecursive(int[] array, int endIndex ) {
        if(endIndex > 0) {

                // find largest element in rest of array
                int largest= indexOfLargest(array, endIndex);

                // move smallest element to position endIndex
                swap(array, largest, endIndex);

                // recurse on everything to the left of startIndex
                selectionSortRecursive(array, endIndex-1);
        }
}
```

Correctness of Selection Sort

For all $n \geq 0$ where array.length $> n$, after running selectionSort(array,n), array[0…n] is sorted in non-descending order.

$P(n)$: After running selectionSort(array,n), array[0...n] is sorted in non-descending order.

Base case: prove $P(0)$

selectionSort(array,0) does nothing, but array[0…0] has only one element and hence is in order.

Selection Sort – Induction

- Suppose $P(k)$ is true. i.e. if we call selectionSort(array,k), then array[0..k] will be in (non-descending) order

- Prove $P(k+1)$:
  - Call of selectionSort(array,k+1) starts by finding index of largest element in array[0…k+1] and swaps with element in array[k+1].

  - By induction assumption, recursive call of selectionSort(array,k) leaves array[0...k] in order, and array[k+1] is larger, so array[0...k+1] is in order. ✔

Analysis

- Count number of comparisons of elts from array
  - All comparisons are in "indexOfLargest(array,n)"
    - At most n comparisons.
  - Prove # of comparisons in selectionSort(array,n) is 1 + 2 + … + n.
    - Base case: n = 0: No comparisons
    - Assume true for selectionSort(array,k-1): 1 + 2 + … + (k-1)
    - Show for k elements:
      - indexOfLargest(array,k) takes k comparisons,
      - swap takes none.
      - By induction selectionSort(array,k-1) takes 1 + 2 +…+(k-1).
      - Therefore total: 1 + 2 + … + (k-1) + k

Complexity of Selection Sort

- If array has length n then **`selectionSort(array,0)`** takes time $n(n-1)/2$, so $O(n^2)$

- Iterative version of selection sort is in text.

Strong induction

- Sometimes need to assume more than just the previous case, so instead
  - Prove $P(0)$
  - Assumption holds for $P(j)$ for every j $= 0, .., k$ in order to prove $P(k+1)$.

FastPower

- $fastPower(x, n)$ algorithm to calculate x^n:
  - if $n == 0$ then return 1
  - if n is even, return $fastPower(x^2, n/2)$
  - if n is odd, return $x * fastPower(x, n-1)$

FastPower - Proof by induction on n

- Base case: $n = 0$
  - $x^0 = 1$ and $fastPower(x, 0) = 1$
- Assume $fastPower(x, j)$ is x^j for all j $\leq k$.
- Show $fastPower(x, k + 1)$ is x^{k+1}
- Case: $k + 1$ is even
  - $fastPower(x, k + 1) = fastPower(x^2, (k + 1)/2) = (x^2)^{(k+1)/2} = x^{k+1}$
- Case: $k + 1$ is odd
  - $fastPower(x, k + 1) = x * fastPower(x, k) = x * x^k = x^{k+1}$