Lecture 8: Induction and Sorting

CS 62

Fall 2018

Alexandra Papoutsaki & William Devanny

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

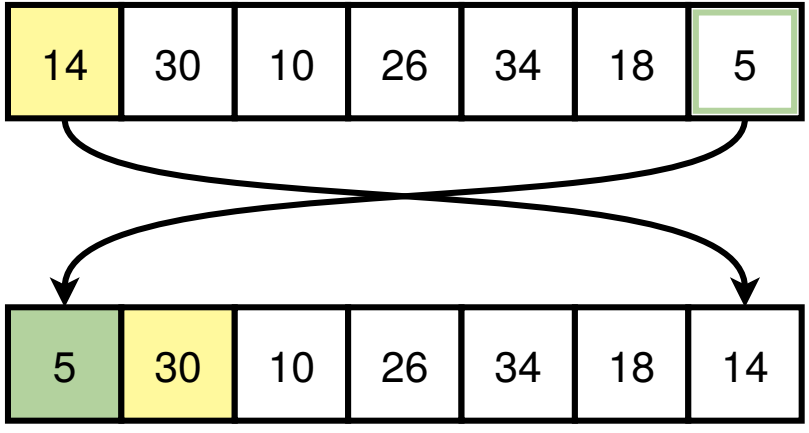
- Let $P(n)$ be some proposition
  - $P(n)$ should be an assertion

- 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 some $k \geq 0$
  - (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 $n \geq 4$

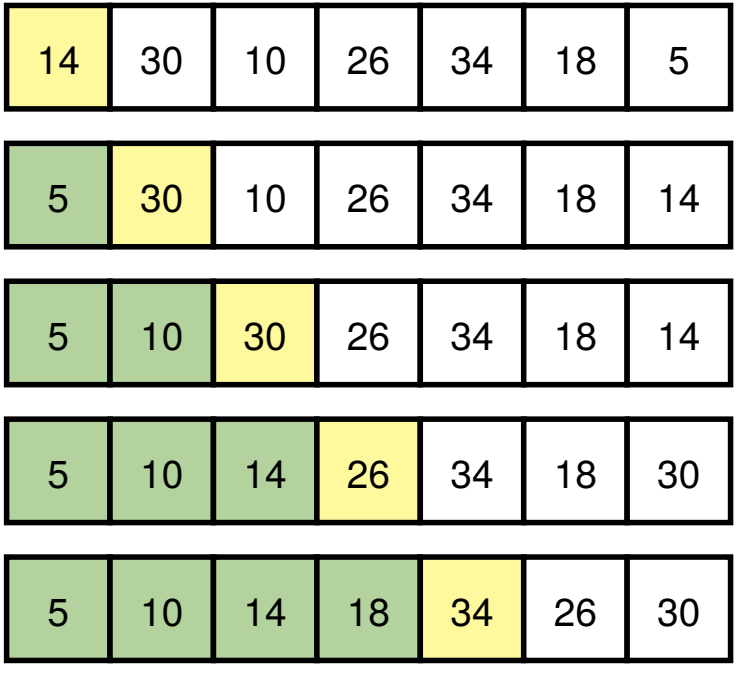
Selection Sort



Array can be seen as split in two parts: Unsorted (white) and sorted (green)

1. Select the first element of the unsorted list (yellow)
2. Find the smallest element in the remaining unsorted array (white)
3. Swap it with the first selected element and adjust the sorted array
4. Repeat with the rest of the array

Selection Sort



Selection Sort (helper)

```
/*
* @param array array of integers
* @param startIndex valid index into array
* @return index of smallest value in array[startIndex…length]
*/
protected static int indexofSmallest(int[] array, int startIndex) {
  int smallest = startIndex;
  for (int i = startIndex + 1; i < array.length; i++) {
    if (array[i] < array[largestIndex ]) {
      smallest = i;
    }
  }
  return smallest;
}
```

Selection Sort (helper)

```
/*
* Swaps array[n1] and array[n2]
* @param array array of integers
* @param n1 valid index into array
* @param n2 valid index into array
* @throws IllegalArgumentException if n1 or n2 are not valid indices
*/
protected static void swap(int[] array, int n1, int n2) {
  if (n1 >= array.length || n2>=array.length || n1<0 || n2>0) {
    throw new IllegalArgumentException("Invalid array indices");
    }
  }
  int tmp = array[n1];
  array[n1] = array[n2];
  array[n2] = tmp;
}
```

Recursive Selection Sort

```
/**
 * @param array array of integers
 * @param startIndex a valid index into array
 */
private static void selectionSortRecursive(int[] array, int startIndex) {
        if(startIndex > 0) {
                //recursively sort array[0,…startIndex-1];
                selectionSortRecursive(array, startIndex-1);
        }
        // find smallest element in array[startIndex…n]
        int smallest = indexOfSmallest(array, startIndex);
        // move smallest element to position startIndex
        swap(array, smallest, startIndex);
        }
}
```

Correctness of Selection Sort

For all $n \geq 0$ after running `selectionSortRecursive(array,n)`, `array[0…n]` has $n + 1$ elements sorted in non-descending order.

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

Base case: prove $P(0)$

`selectionSortRecursive(array,0)` skips the recursive call, finds the minimum element of `array` and places it at `array[0]`. So the one-element array `array[0…0]` contains the 1st smallest element, which is trivially in non-descending order.

Selection Sort – Induction

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

- Prove $P(k+1)$:
  - Call of `selectionSortRecursive(array,k+1)` recursively calls `selectionSortRecursive(array,k)`
  - By induction hypothesis, recursive call of `selectionSortRecursive(array,k)` leaves `array[0...k]` in non-descending order by containing the k smallest elements sorted. Selection sort then finds the minimum element in `array[k+1…n]`, which would have to be the $(k+1)$st smallest element overall, and swaps it with the element at `array[k+1]`. Therefore `array[0...k+1]` is in order, by containing the $k+1$ smallest elements of array.✔

Time Complexity Analysis

- Count number of operations, i.e. comparisons of elements from array
- `indexOfSmallest(array,0)` → n-1 comparisons.
- `indexOfSmallest(array,1)` → n-2 comparisons.
- …
- `indexOfSmallest(array,n-1)` → 1 comparisons.

- In total: $(n-1) + \cdots + 2 + 1 = n(n-1)/2$
- If array has length n then **`selectionSortRecursive(array,n)`** takes time $n(n-1)/2$, so $O(n^2)$

Iterative Selection Sort

```
/**
* Sorts integer array using iterative selection sort
* @param array array of integers to be sorted
*/
private static void selectionSortIterative(int[] array) {
      for(int i = 0; i < array.length; ++i) {
            int min = indexOfSmallest(array, i);
            swap(array, i, min);
      }
}
```

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
  - $fastPower(x, k+1) = x * fastPower(x, k) = x * x^k = x^{k+1}$