Lecture 11: More Sorting and Induction

CS 62
Spring 2019
William Devanny & Alexandra Papoutsaki
Saturday Mentor Hours

• Saturday 10-noon
  • New mentor: Gabe Alzate
/**
 * MergeSort  Sorts data >= low and < high
 * @param list data to be sorted
 * @param low start of the data to be sorted
 * @param high end of the data to be sorted (exclusive)
 */

private void mergeSort(int[] data, int low, int high){
    if( high-low > 1 ){
        int mid = low + (high-low)/2;
        mergeSort(data, low, mid);
        mergeSort(data, mid, high);
        merge(data, low, mid, high);
    }
}
Complexity

- Claim: *mergeSort* is $O(n \log n)$
  - where $\log$ is base 2
- Merge of two lists of combined size $n$ takes $\leq n - 1$ comparisons.
  - Think of merging [1,3,5,7] and [2,4,6,8]
- If $l$ levels:
  - $n/2^l = 1$
  - $n = 2^l$
  - $l = \log n$
- $\log n$ levels
- each taking $O(n)$ operations
- $O(n \log n)$ in total
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 $1 + 2 + \ldots + n = \frac{n(n + 1)}{2}$ 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 $n \geq 4$
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 \( \text{selectionSortRecursive}(\text{array}, n) \), \( \text{array}[0...n] \) has \( n + 1 \) elements sorted in non-descending order.

\( P(n) \): After running \( \text{selectionSortRecursive}(\text{array}, n) \), \( \text{array}[0...n] \) contains the \( n + 1 \) smallest elements sorted in non-descending order.

Base case: prove \( P(0) \)

\( \text{selectionSortRecursive}(\text{array}, 0) \) skips the recursive call, finds the minimum element of \( \text{array} \) and places it at \( \text{array}[0] \). So the one-element array \( \text{array}[0...0] \) 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 contain the $k+1$ smallest elements 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 $k + 1$ smallest elements sorted. Selection sort then finds the minimum element in `array[k+1...n]`, which would have to be the $(k + 2)\text{nd}$ smallest element overall, and swaps it with the element at `array[k+1]`. Therefore `array[0...k+1]` contains the $k + 2$ smallest elements of `array` in order.
Strong induction

• Sometimes need to assume more than just the previous case, so instead
  • Prove $P(0)$
  • Assumption holds for $P(j)$ for every $j = 0, \ldots, k$ in order to prove $P(k + 1)$. 
/**
 * MergeSort   Sorts data >= low and < high
 * @param list data to be sorted
 * @param low start of the data to be sorted
 * @param high end of the data to be sorted (exclusive)
 */

private void mergeSort(int[] data, int low, int high){
    if( high-low > 1 ){
        int mid = low + (high-low)/2;
        mergeSort(data, low, mid);
        mergeSort(data, mid, high);
        merge(data, low, mid, high);
    }
}
Correctness

• $P(n)$: If $\text{high} - \text{low} = n$ then $\text{mergeSort}(\text{data}, \text{low}, \text{high})$ will result in $\text{data}[\text{low} .. \text{high}]$ being correctly sorted
• For simplicity, assume $\text{merge}$ is correct
• Assume $P(k)$ for all $k < n$, show $P(n)$
• If $n = 0$ or $1$ then (correctly) do nothing
• Assume $n > 1$
  • Call $\text{mergeSort}(\text{data}, \text{low}, \text{mid})$ and $\text{mergeSort}(\text{data}, \text{mid} + 1, \text{high})$ where $\text{mid} = \text{low} + (\text{high} - \text{low})/2$.
  • Hence $\text{mid} - \text{low} < n, \text{high} - (\text{mid} + 1) < n$
  • By induction $\text{data}[\text{low}..\text{mid}]$ and $\text{data}[\text{mid} + 1 .. \text{high}]$ now sorted.
  • call $\text{merge}(\text{data}, \text{low}, \text{mid}, \text{high})$ and, by assumption on $\text{merge}$, $\text{data}[\text{low} .. \text{high}]$ now sorted! Thus $P(n)$ true.
Complexity

- \( P(m) \): if data has \( 2^m \) elements then \textit{mergesort} makes \(< m \times 2^m \) total comparisons.
- Assume \( P(k) \) for all \( k < m \). Prove \( P(m) \)
- \( P(0) \) is clear. Show \( P(m) \)
- Sort first half, second half, and then merge
- Each half has size \( \frac{2^m}{2} = 2^{m-1} < 2^m \), so by induction, each takes \(< (m - 1) \times 2^{m-1} \) comparisons
- Therefore total number of comparisons in \textit{mergesort} \\
  \(< (m - 1) \times 2^{m-1} + (m - 1) \times 2^{m-1} + (2^m - 1) \) \\
  = (m - 1) \times 2^m + (2^m - 1) = m \times 2^m - 1 < m \times 2^m \\
- Thus \( P(m) \) is true
- If \( n = 2^m \) then \textit{mergeSort} takes \( n \log n \) comparisons (\( m = \log n \)).