CS 375 – Analysis of Algorithms

Professor Eric Aaron

Lecture – M W 1:00pm

Lecture Meeting Location: It’s complicated…

Business, Mar. 24

• Reading: Ch. 2.1

• Small Assignment(s) assigned today
  – One is due by the next class meeting; the other is due by the class meeting on Mar. 31—see assignment sheets for specifics

• See Asymptotic Complexity slides for other Mar. 24 course business
Insertion Sort

- An iterative insertion sort! Is it efficient?
  - Also, is it correct? Does this meet the specifications for the sorting problem?
  - (More on that later in the course, when we talk about loop invariants!)

```
InsertionSort(A)
1. for j = 2 to length[A]
2. key = A[j]
3. i = j - 1
4. while i > 0 and A[i] > key
6. i = i - 1
7. A[i+1] = key
```

Input: A sequence L of n numbers <a_1, ..., a_n>
Output: A permutation (reordering) <b_1, ..., b_n> of the input sequence (perhaps leaving them unchanged) such that b_1 ≤ b_2 ≤ ... ≤ b_n

Please read CLRS section 2.1

In the algo:
- j is the element to be inserted in order
- i ranges over elements of the previously sorted sub-array

Insertion Sort: An Example

- Example of insertion sort working on an array (taken from CLRS, section 2.1)

How would we argue correctness, for the pseudocode in this example?
Algorithm Correctness Analysis: Loop Invariants

• To prove algorithms correct, can use loop invariants to analyze loops
  – Loop invariants are properties of loops that are true each time through the loop
    • Initialization: Property is true before the first iteration
    • Maintenance: If a property is true before an iteration, it is true (after that iteration) before the next iteration
    • Termination: When the loop terminates, the property (or the violation of the property) is useful in showing algorithm correctness

See CLRS 2.1

Insertion Sort and Loop Invariants

• From CLRS (which starts counting at 1), Insertion sort:

```
INSERTION-SORT (A)
1 for j = 2 to A.length
2   key = A[j]
3 // Insert A[j] into the sorted sequence A[1..j-1].
4   i = j - 1
5   while i > 0 and A[i] > key
6     A[i+1] = A[i]
7     i = i - 1
8   A[i+1] = key
```

• Outer loop invariant: sub-array A[1..j-1] consists of elements originally in A[1..j-1], but in sorted order
  – How can we explain the correctness of the invariant?
  – How does it help to show the correctness of the overall algorithm?
Insertion Sort and Loop Invariants

- From CLRS (which starts counting at 1), Insertion sort:

```
INSERTION-SORT (A)
1  for j = 2 to A.length
2     key = A[j]
3     // Insert A[j] into the sorted sequence A[1 .. j − 1].
4     i = j − 1
5     while i > 0 and A[i] > key
6         A[i + 1] = A[i]
7         i = i − 1
8         A[i + 1] = key
```

- Use invariant to show correctness:
  - Show invariant is true at loop initialization (How?)
  - Show maintenance step: invariant is true at next iteration, assuming that it’s true at the start of this iteration (How?)
  - Use termination condition (What is it?) to show correctness of the overall algorithm (How?)

**Outer loop invariant:** sub-array A[1..j-1] consists of elements originally in A[1 .. j-1], but in sorted order

**Sorting Problem**

Input: Sequence of numbers \(<a_1, ..., a_n>\)

Output: Permutation (reordering) \(<b_1, ..., b_n>\) of the input sequence (perhaps leaving them unchanged) such that \(b_1 \leq b_2 \leq ... \leq b_n\)

Exercise: What might an invariant be for the inner loop?
Another example: Bubble Sort

- (Yes, bubble sort is the actual name of this sorting algorithm)
- In pseudocode:

```
BubbleSort(A)
    1. for i = 1 to A.length - 1
    2.   for j = A.length downto i + 1
```

- How do we argue correctness (i.e., that it sorts A in non-decreasing order)?
  - What might the loop invariant be for the outer loop?

Another example: Bubble Sort

- Loop invariant for outer loop:

Subarray $A[1..i-1]$ consists of the $i-1$ smallest values of $A$, in sorted order, and $A[i..n]$ consists of the remaining values of $A$ (no constraint on order)

```
BubbleSort(A)
    1. for i = 1 to A.length - 1
    2.   for j = A.length downto i+1
```

- Is the invariant true at initialization ($i = 1$)?
- Is the invariant maintained by each iteration of the loop—i.e., is it true at the end of an iteration if we assume it’s true at the beginning?
- What does the invariant show is true at termination of the loop? How does that help us show the algorithm overall is correct?
### Loop Invariants

<table>
<thead>
<tr>
<th>Design Paradigm</th>
<th>Analysis</th>
<th>Correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterative</td>
<td>Counting</td>
<td>Loop invariants</td>
</tr>
<tr>
<td></td>
<td>(Exact count of operations / space used)</td>
<td></td>
</tr>
<tr>
<td>Recursive</td>
<td>Solving <em>recurrences</em></td>
<td>Induction</td>
</tr>
</tbody>
</table>

*recurrences*