CSCI 241

Lecture 3:
Loop Invariants
Insertion and Selection Sort
Roadmap

• Tools for reasoning about algorithms

• Sorting algorithms (A1)
  • Insertion
  • Merge
  • Quick
  • Radix

• Onward to more data structures
Today

• Quiz recap

• Tools for reasoning about algorithms

• Sorting algorithms (A1)
  • Insertion
  • Selection
  • Merge
  • Quick
  • Radix

• Onward to more data structures
public class LinkedList {
    public Node head;

    /** if n is in the list, remove all elements
     * *following* it and return true.
     * otherwise, return false leaving the list unchanged
     * precondition: n is not null */
    public boolean truncateAfter(Node n) {
        Node current = head;
        while (current != null) {
            if (current == n) {
                // INSERT CODE HERE
                return true;
            }
            current = current.next;
        }
        return false;
    }
}

public class Node {
    Node next;
    int value;
}
public class Factorial {
    /** return n factorial. precondition: n > 0 */

    public static int fact2(int n) {
        return n * fact2(n-1);
    }

    public static int fact3(int n) {
        if (n > 0) {
            return n * fact3(n-1);
        }
        return 1;
    }

    public static int fact4(int n) {
        if (n > 1) {
            return n * fact4(n);
        }
    }
}

public class Factorial {
    /** return n factorial. precondition: n > 0 */

    public static int fact2(int n) {
        return n * fact2(n-1);
    }

    public static int fact3(int n) {
        if (n > 0) {
            return n * fact3(n-1);
        }
        return 1;
    }

    public static int fact4(int n) {
        if (n > 1) {
            return n * fact4(n);
        }
    }
}
public class Factorial {
    /** return n factorial. precondition: n > 0 */

    public static int fact2(int n) {
        return n * fact2(n-1);
    }

    public static int fact3(int n) {
        if (n > 0) {
            return n * fact3(n-1);
        }
        return 1;
    }

    public static int fact4(int n) {
        if (n > 1) {
            return n * fact4(n);
        }
    }
}
Goals:

• Be able to analyze an algorithm’s correctness using a given loop invariant.

• Be able to describe in words, implement using a loop invariant, and analyze the runtime of:
  
  • Selection sort
  
  • Insertion sort
Tools for Reasoning about Algorithms
/** return the max value in A
   * precondition: A is nonempty
   * postcondition: max value of A is returned */
public int findMax(int[] A) {
    int max = A[0];
    // invariant: max is the largest value in A[0..i]
    for (int i = 1; i < A.length; i++) {
        if (A[i] > max) {
            max = A[i];
        }
    }
    return max;
}

The precondition is true before method execution.
The postcondition is true after method execution.

Interface, not implementation
Loop Invariant

/** return the max value in A
  * precondition: A is nonempty
  * postcondition: max value of A is returned */
public int findMax(int[] A) {
    int currentMax = A[0];
    // invariant: currentMax is the max of A[0..i]
    for (int i = 1; i < A.length; i++) {
        if (A[i] > max) {
            max = A[i];
        }
    }
    currentMax is the largest value in:
    return max;    A[0..1]
}
The loop invariant is true before, during, and after the loop.
Another Example

/** rearrange A so all negative values are to
 * the left of all non-negative values */
public void separateSign(int[] A);

Postcondition: A < 0 >= 0
/** rearrange A so all negative values are to
 * the left of all non-negative values */

public void separateSign(int[] A) {

Precondition:   

Invariant:  

Postcondition:  

Four concerns:

1. Initialization: Make the invariant true at the start.
2. Termination: Make the loop end when the postcondition is true.
3. Progress: Make progress towards the postcondition.
4. Maintenance: Make the invariant true after each iteration.
Insertion Sort

Insert $A[i]$ into the sorted sublist $A[0..i-1]$.

Selection Sort

Find the smallest element in $A[i..n]$ and place it at $A[i]$.

https://visualgo.net/bn/sorting
Insertion Sort

Insert A[i] into the sorted sublist A[0..i-1].

Invariant: A sorted

Selection Sort

Find the smallest element in A[i..n] and place it at A[i].

Invariant: A sorted, <= A[i..n]


insertionSort(A):
    i = 0;
    while i < A.length:
        // push A[i] to its sorted position
        // increment i

selectionSort(A):
    i = 0;
    while i < A.length:
        // find min of A[i..A.length]
        // swap it with A[i]
        // increment i
Developing InsertionSort

Precondition: A

Invariant: A \textit{sorted} i

Postcondition: A \textit{sorted}

Four tasks:

1. **Initialization**: Make the invariant true at the start.
2. **Termination**: Make the loop end when the postcondition is true.
3. **Progress**: Make progress towards the postcondition.
4. **Maintenance**: Make the invariant true after each iteration.
Developing SelectionSort

Precondition: $A$

Invariant: $A$ sorted, $\leq A[1..n]$

Postcondition: $A$ sorted

Four concerns:

1. Initialization: Make the invariant true at the start.
2. Termination: Make the loop end when the postcondition is true.
3. Progress: Make progress towards the postcondition.
4. Maintenance: Make the invariant true after each iteration.
insertionSort(A):
    i = 0;
    while i < A.length:
        j = i;
        while j > 0 and A[j] > A[j-1]:
            swap(A[j], A[j-1])
            j--
        i++

ABCD: What’s the best and worst-case asymptotic runtime complexity of insertionSort?

<table>
<thead>
<tr>
<th></th>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>B</td>
<td>O(n²)</td>
<td>O(n)</td>
</tr>
<tr>
<td>C</td>
<td>O(n)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>D</td>
<td>O(n²)</td>
<td>O(n²)</td>
</tr>
</tbody>
</table>

Why is this best-case runtime interesting?
insertionSort1(A):
i = 0;
while i < A.length:
    j = i;
    while j > 0 and A[j] < A[j-1]:
        swap(A[j], A[j-1])
        j--
i++

insertionSort2(A):
i = 0;
while i < A.length:
    j = i;
    tmp = A[i];
    while j > 0 and tmp < A[j-1]:
        j--
i++

ABCD: What’s the best and worst-case asymptotic runtime complexity of insertionSort2?

<table>
<thead>
<tr>
<th></th>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>B</td>
<td>O(n^2)</td>
<td>O(n)</td>
</tr>
<tr>
<td>C</td>
<td>O(n)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>D</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
</tr>
</tbody>
</table>
Practice problems

Write a precondition, postcondition, and loop invariant for the *inner* loop of InsertionSort.

Develop SelectionSort using the precondition, loop invariant, and postcondition by completing the four tasks:
- Initialization
- Termination
- Progress
- Maintenance