

CSC 301: Analysis of Algorithms

Lecture 2: Loop Invariants and Insertion Sort

William Rebelsky
[rebelskyw]

Grinnell College

Wednesday 24th January, 2024

Reminders

- Keep track of the Syllabus: Readings should be done before class not after
- Homework 0 has been released on gradescope, it is due next Wednesday
- We have a course website: rebelskyw.cs.grinnell.edu
- Tokens: Let me know if you have any events you would like me to add
- Quiz 0: one of the questions is: what days do you know you are going to miss for an approved reason (e.g., sporting events, religious observances). As a reminder, the college asks you to let your faculty know by the end of the second week of classes
- Attendance: I'm keeping track of attendance, please sign in on the sign in sheet

Table of Contents

1 Analyzing Algorithms

- Loop Invariants
- Time Analysis

2 Practice

- Insertion Sort
- Rotated Dutch Flag

Loop Invariants

- **Initialization:** It is true prior to the first iteration of the loop
- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration
- **Termination:** The loop terminates, and when it terminates, the invariant and the termination gives us a useful property that helps show the algorithm is correct

Induction

- Prove a base case (e.g. when $n=1$)
- Inductive step (e.g. show it is true for $n=k+1$ given that it is true for $n=k$)

RAM Model

- Each instruction or data access takes a constant amount of time
- Add, subtract, multiply, divide, remainder, floor, ceiling, load, store, copy, etc.
- Run time is considered in terms of the number of the constant time calls above based on the size of the input
- May need to use combinatorics, statistics and probability, and algebra
- We only care about the leading term (e.g. $an^2 + bn + c \approx an^2$ for large n)
- $\Theta(n)$ means roughly proportional to n when n is large

Table of Contents

1 Analyzing Algorithms

- Loop Invariants
- Time Analysis

2 Practice

- Insertion Sort
- Rotated Dutch Flag

Insertion Sort

Algorithm 1: Insertion-Sort(A, n)

Input: sequence of n numbers $\{a_1, \dots, a_n\}$ **Output:** permutation $a'_1, \dots, a'_n$ such that $a'_1 \leq \dots \leq a'_n$

```
1 for  $i=2$  to  $n$  do
2    $\text{key} = A[i]$ 
3   // Insert  $A[i]$  into the sorted subarray  $A[1:i-1]$ 
4    $j = i-1$ 
5   while  $j > 0$  and  $A[j] > \text{key}$  do
6      $A[j+1] = A[j]$ 
7      $j = j-1$ 
8    $A[j+1] = \text{key}$ 
9 return ( $A$ )
```

Insertion Sort

- What are the indices of the array?
- How would you use a loop invariant to show correctness of the pseudocode?

BLANK

This frame intentionally left blank

Insertion Sort: Correctness

- **Initialization:** Before the first loop (when $i=2$), we have the subarray $A[1:i-1]=A[1]$ which is sorted by definition.
- **Maintenance:** (Informally) The for loop works by moving values in A one position to the right until it finds the proper position for $A[i]$. The subarray $A[1:i]$ then consists of the original elements in $A[1:i]$ now in sorted order. Incrementing i thus preserves the loop invariant.
- **Termination:** Examine the termination of the loop. When i equals $n+1$ the for loop terminates. Substituting $n+1$ into the loop invariant gives us that $A[1:n]$ consists of the original elements of $A[1:n]$ in sorted order.

Insertion Sort: Run Time

Algorithm 2: Insertion-Sort(A, n)

Input: sequence of n numbers $\{a_1, \dots, a_n\}$

Cost

times

Output: permutation $a'_1, \dots, a'_n$ such that $a'_1 \leq \dots \leq a'_n$

1	for $i=2$ to n do	c_1	n
2	$\text{key} = A[i]$	c_2	$n-1$
3	// Insert $A[i]$ into the sorted subarray $A[1:i-1]$	0	$n-1$
4	$j = i-1$	c_4	$n-1$
5	while $j > 0$ <i>and</i> $A[j] > \text{key}$ do	c_5	$\sum_{i=2}^n t_i$
6	$A[j+1] = A[j]$	c_6	$\sum_{i=2}^n (t_i - 1)$
7	$j = j-1$	c_7	$\sum_{i=2}^n (t_i - 1)$
8	$A[j+1] = \text{key}$	c_8	$n-1$
9	return (A)		

Insertion Sort: RunTime (formally)

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^n t_i + c_6 \sum_{i=2}^n (t_i - 1) + c_7 \sum_{i=2}^n (t_i - 1) + c_8(n-1)$$

- Best case scenario: already sorted
- $T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) = an + b$.
This is linear time $\Theta(n)$
- Worst case scenario: reverse order
- The summations are $\sum_{i=2}^n i = \frac{(n)(n-1)}{2} - 1$ and $\sum_{i=2}^n (i-1) = \frac{(n)(n-1)}{2} = an^2 + bn + c$. This is quadratic time $\Theta(n^2)$

Insertion Sort: RunTime (informally)

- Outer for loop: n times
- Inner while loop: Worst case n times, best case 1 time (per for loop)
- Worst case runtime: Θn^2

BLANK

Rotated Dutch Flag Problem

- **Input:** Array of size n with each entry being a color red, white, or blue
 - e.g. $A = \text{rbwbwbwbwrrrbwwbwb}$
- **Output:** Array of size n : $[[\text{red}], [\text{white}], [\text{blue}]]$
 - e.g. $A = \text{rrrrwwwwwwbBBBBBB}$
- **Restrictions:**
 - Space: Sorting must be done in place
 - Time: Linear time

BLANK

DutchFlag Hint

- consider numbers i, j, k such that every location less than i is red, between i and j is white, between j and k is unsorted, and larger than k is blue.

Dutch Flag Invariant

- **Initialization:** $i=j=0$, $k=n$. Clearly meets the invariant above
- while $j < k$:
 - if $A[j]=w$, $j++$
 - if $A[j]=r$, swap $A[i]$ and $A[j]$, $i++$, $j++$
 - if $A[j]=b$, swap $A[j]$ and $A[k-1]$, $k--$
- **Maintenance:** In each step, the invariant remains true as we move the color to the correct index and update i,j,k as needed
- **Termination:** In each step, either j increases or k decreases
- Thus by the loop invariant we see that this is correct, the runtime is linear and in place