CSCI 470 Fundamentals of Algorithms: Loop Invariant, Insertion Sort, Growth of Functions

Vijay Chaudhary

August 23, 2023

Department of Electrical Engineering and Computer Science
Howard University

Overview

1. Revision
2. RAM Model of Computation
3. Insertion Sort: Runtime Analysis
4. Asymptotic Notation

Revision

In case you missed it the last time

Please fill out this form: https://tinyurl.com/csci-470-form. I don't have access to Canvas yet. I will use your emails to communicate with the class.

Quiz

No Quiz Today! There will at least one next week.

Revisiting one previous classwork problem

What is the smallest value of n, say n_0, such that an algorithm (Algorithm 1) whose running time is $100n^2$ runs faster than an algorithm (Algorithm 2) whose running time is 2^n on the same machine?

- The purpose of the classwork was to show some functions grow faster than others only after hitting a threshold input value.
- Instead of solving it algebraically (I made that mistake), we can simply enumerate the values of $100n^2$ and 2^n over a range of positive integers.

Revisiting one previous classwork contd ...

A	B	C	D
values of n	100n^2	2^n	100n^2 < 2^n
0	0	1	TRUE
1	100	2	FALSE
2	400	4	FALSE
3	900	8	FALSE
4	1600	16	FALSE
5	2500	32	FALSE
6	3600	64	FALSE
7	4900	128	FALSE
8	6400	256	FALSE
9	8100	512	FALSE
10	10000	1024	FALSE
11	12100	2048	FALSE
12	14400	4096	FALSE
13	16900	8192	FALSE
14	19600	16384	FALSE
15	22500	32768	TRUE
16	25600	65536	TRUE
17	28900	131072	TRUE
18	32400	262144	TRUE
19	36100	524288	TRUE
20	40000	1048576	TRUE

Revisiting Loop Invariant

SUM(A, n)

```
i = 0
sum = 0
while i < n
    sum = sum + A[i]
    i = i + 1
return sum
```

- What will be the loop invariant here?
- We may have to think in terms of the iterating value, i.

Revisiting Loop Invariant

Loop invariant: $\sum_{k=0}^{i-1} A[k]$, the sum before i-th iteration.

Base case/Initialization: $i = 0$, $\sum_{k=0}^{i-1} A[k] = 0$ because there is nothing to add. The sum before first iteration will also be 0.

Inductive Step: Let's assume that the algorithm gives us $sum_{j-1} = \sum_{k=0}^{j-1} A[k]$ before the j-th iteration where $j < n$.
Then, before $(j+1)$-th iteration,
$sum_j = sum_{j-1} + A[j] = \sum_{k=0}^{j-1} A[k] + A[j] = \sum_{k=0}^{j} A[k]$

Termination: The loop terminates when $i = n$. $sum_n = \sum_{k=0}^{n-1} A[k]$, which is the required answer for a 0-indexed array.

RAM Model of Computation

In this course, for most of the time we will use ***random-access machine (RAM)*** model of computation. Under this model of computation, we assume that the following instructions take constant time (borrowed from page 24 of CLRS):

- Arithmetic: add, subtract, multiply, divide, remainder, floor, ceiling
- Data movement: load, store, copy
- Flow control: conditional and unconditional branch, subroutine call, return

Insertion Sort: Runtime Analysis

Insertion Sort: Runtime Analysis

INSERTION-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
$i = j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i = i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] = key$	c_8	$n-1$

$$T(n) = c_1(n) + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Insertion Sort: Best case analysis

In the best case, the inner while loop doesn't run. **Why though?**

$$
\begin{aligned}
T(n) &= c_1(n) + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} j + c_8(n-1) \\
&= c_1(n) + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8).
\end{aligned}
$$

Insertion Sort: Worst Runtime Analysis

In the worst case, the inner while loop run for j times at each iteration, which makes $t_j = j$. Then,

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \text{ and } \sum_{j=2}^{n} j - 1 = \frac{n(n-1)}{2}$$

$$\begin{aligned}
T(n) = c_1(n) + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + \\
c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
= (\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2})n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\
- (c_2 + c_4 + c_5 + c_8)
\end{aligned}$$

Insertion Sort: Average Runtime Analysis

An average case runtime analysis is based on the probability distribution of the inputs. If we assume that half of the items in subarray $A[1, ..., j-1]$ are less than the key, $A[j]$, and the rest of $A[1, ..., j-1]$ are greater. The inner while loop will run $t_j = j/2$ to drop the key at the proper insertion index. **I will leave this as an exercise to show, the average case runtime will still be quadratic function of the input size.**

A Quick Question

How can we modify almost any algorithm to have a good best-case running time?

Asymptotic Notation

Growth of Functions

We just found that the running time of insertion sort is a quadratic function of its input array size, $T(n) = an^2 + bn + c$.
Will there be a big difference between $T_1(n) = 50n^2 + 100n + 8$ and $T_2(n) = n^2 + n + 1$, when n becomes very large?

Asymptotic Notation

If $T_1(n) = 50n^2 + 100n + 8$ and $T_2(n) = n^2 + n + 1$ become closer to each other for a very large value of n, won't it be wiser to categorize these two function into the same class?

Something like $T_1(n) = T_2(n) = \Theta(n^2)$ which essentially means they both belong to the same set of functions.

Θ-notation

$\Theta(g(n)) = \{f(n)$: there exists positive constants $c_1, c_2,$ and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$.

Example: If $f(n) = 2n^2 - 3n$, then $f(n) = \Theta(n^2)$. To prove this, we need to find two constants c_1 and c_2 such that $c_1 n^2 \leq f(n) \leq c_2 n^2$.
$c1 = 1, c_2 = 3, n_0 = 4$

O-notation

$O(g(n)) = \{f(n) :$ there exists positive constants $c,$ and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$.

Example: If $f(n) = 2n^2 - 3n$, then $f(n) = O(n^2)$. $c = 3, n_0 = 4$

Ω-notation

$\Omega(g(n)) = \{f(n) :$ there exists positive constants $c,$ and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$.

Example: If $f(n) = 2n^2 - 3n$, then $f(n) = O(n^2)$. $c = 1, n_0 = 4$

Summary of asymptotic notations

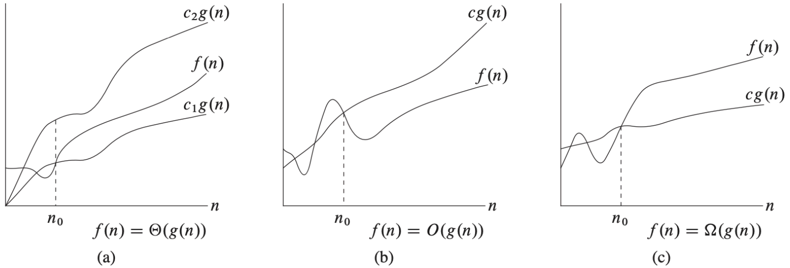

Figure 3.1 Graphic examples of the Θ, O, and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. **(a)** Θ-notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1, and c_2 such that at and to the right of n_0, the value of $f(n)$ always lies between $c_1 g(n)$ and $c_2 g(n)$ inclusive. **(b)** O-notation gives an upper bound for a function to within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that at and to the right of n_0, the value of $f(n)$ always lies on or below $cg(n)$. **(c)** Ω-notation gives a lower bound for a function to within a constant factor. We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that at and to the right of n_0, the value of $f(n)$ always lies on or above $cg(n)$.

Classwork

- Prove that $f(n) = n^2 + 4n + 5$ is $O(n^2)$.
- If $f(n) = O(n)$, then $f(n) = O(n^3)$. Explain.
- If a function is $O(n)$, can it also be $\Omega(n^2)$?