

In this course, we will be interested in the worst-case running time because

- 1) it is an upper bound on running time for any input
- 2) it often happens frequently
- 3) “average case” is often roughly as bad as the worst case

Problems with average case analysis: what is the “average” input?

Simplifying abstractions for analysis of running time

- we ignore actual cost of each statement, using constants c_i instead
- even constants c_i provide more detail than we want

we simplify to an^2+bn+c

- what we really want to compare algorithms is order of growth

which is the leading term of a formula (e.g. an^2), since the lower-order terms are relatively insignificant for large values of n

- finally, we ignore the leading term’s constant factors

since constant factors are less significant than the rate of growth

Thus, we say that Insertion-Sort has worst-case running time of $\Theta(n^2)$

Worst case for Insertion-Sort: array is in reverse sorted order

$t_j=j$ for every j

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

$$T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4 \left(\frac{n(n+1)}{2} - 1 \right) + c_5 \frac{n(n-1)}{2}$$

$$+ c_6 \frac{n(n-1)}{2} + c_7(n-1)$$

$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2} \right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7 \right) n$$

$$- (c_2 + c_3 + c_4 + c_7)$$

This running time can be expressed as an^2+bn+c for constants $a, b,$ and c that depend on the statement costs c_i

thus, $T(n)$ is a quadratic function of n

$T(n)$ may depend on which input of that size is given

Best case for Insertion-Sort: array is already sorted

$t_j=1$ for every j

$$\begin{aligned}T(n) &= c_1n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1) \\ &= (c_1 + c_2 + c_3 + c_4 + c_7) n - (c_2 + c_3 + c_4 + c_7)\end{aligned}$$

This running time can be expressed as $an+b$ for constants a and b that depend on the statement costs c_i

thus, $T(n)$ is a linear function of n

	cost	#times executed
Insertion-Sort(A)		
for j=2 to length[A]	c_1	n
key=A[j]	c_2	n-1
i=j-1	c_3	n-1
while i>0 and A[i]>key	c_4	$\sum_{j=2}^n (t_j)$
A[i+1]=A[i]	c_5	$\sum_{j=2}^n (t_j - 1)$
i=i-1	c_6	$\sum_{j=2}^n (t_j - 1)$
A[i+1]=key	c_7	n-1

t_j is the number of times while loop test is executed for that value of j

$T(n)$ = running time of the algorithm on an input of size n

$$\begin{aligned}
 &= c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^n (t_j) + c_5 \sum_{j=2}^n (t_j - 1) \\
 &+ c_6 \sum_{j=2}^n (t_j - 1) + c_7(n-1)
 \end{aligned}$$

Analyzing algorithms

we assume a random-access machine (RAM)

objective: show the important characteristics of the algorithm's resource requirements, with minimal tedious details

2 important resources: time and space

today: memory is plentiful; for most algorithms not a limitation
chip speed is fast, BUT for many problems, bad algorithms run very slowly, even with the fastest chips

input size depends on problem being studied
most often, it's the number of items on the input

running time of an algorithm: the number of primitive operations the algorithm performs on the given input

primitive operations: we assume they take a constant amount of time

Insertion-Sort(A)

for j=2 to length[A]

key=A[j]

/* Insert A[j] into sorted sequence A[1...j-1] */

i=j-1

while i>0 and A[i]>key

A[i+1]=A[i]

i=i-1

A[i+1]=key

5 (2) 4 6 1 3

2 5 (4) 6 1 3

2 4 5 6 (1) 3

1 2 4 5 6 (3)

sorts numbers in-place: the numbers are rearranged within array A
using a constant number of additional storage slots

Lecture 1: Introduction

Algorithm:

any well-defined computational procedure that takes some set of values as input and produces some set of values as output

Sorting problem

Input: a sequence of n numbers

Output: a reordering of the input numbers into a non-decreasing sequence

Example: when given an input 14, 6, 7, 43, 21, 14

a sorting algorithm should produce output 6, 7, 14, 14, 21, 43

Algorithm is correct if for every input it produces the right output

Pseudocode we use:

English and/or C-like code (without worrying about syntax)