Insertion Sort

By: Dr. Surjeet Kumar

Dept. of Computer Application

Source: “Introduction to Algorithms” PHI 3rd Edition

by Thomas H. Cormen & Others.

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

The running time of the algorithm is the sum of running times for each state ment executed; a statement that takes c_i steps to execute and executes n times wi contribute $c_i n$ to the total running time.[6] To compute $T(n)$, the running time c INSERTION-SORT on an input of n values, we sum the products of the *cost* an *times* columns, obtaining

$$
\begin{aligned}
T(n) \;=\;& c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\
& + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)\,.
\end{aligned}
$$

ven for inputs of a given size, an algorithm's running time may depend on ch input of that size is given. For example, in INSERTION-SORT, the best e occurs if the array is already sorted. For each $j = 2, 3, \ldots, n$, we then find $A[i] \leq key$ in line 5 when i has its initial value of $j - 1$. Thus $t_j = 1$ for $= 2, 3, \ldots, n$, and the best-case running time is

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

can express this running time as $an + b$ for *constants* a and b that depend on statement costs c_i; it is thus a ***linear function*** of n.

f the array is in reverse sorted order—that is, in decreasing order—the worst e results. We must compare each element $A[j]$ with each element in the entire ed subarray $A[1 \,..\, j-1]$, and so $t_j = j$ for $j = 2, 3, \ldots, n$. Noting that

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

$$j - 1) = \frac{n(n-1)}{2}$$

ind that in the worst case, the running time of INSERTION-SORT is

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

an express this worst-case running time as $an^2 + bn + c$ for constants a, b, that again depend on the statement costs c_i; it is thus a ***quadratic function***

pically, as in insertion sort, the running time of an algorithm is fixed for a input.

-case and average-case analysis

analysis of insertion sort, we looked at both the best case, in which the input vas already sorted, and the worst case, in which the input array was reverse

worst-case running time of an algorithm gives us an upper bound on the ning time for any input. Knowing it provides a guarantee that the algorithm never take any longer. We need not make some educated guess about the ning time and hope that it never gets much worse.

some algorithms, the worst case occurs fairly often. For example, in search- a database for a particular piece of information, the searching algorithm's st case will often occur when the information is not present in the database. ome applications, searches for absent information may be frequent.

- The "average case" is often roughly as bad as the worst case. Suppose that w randomly choose n numbers and apply insertion sort. How long does it take determine where in subarray $A[1\,..\,j-1]$ to insert element $A[j]$? On averag half the elements in $A[1\,..\,j-1]$ are less than $A[j]$, and half the elements a greater. On average, therefore, we check half of the subarray $A[1\,..\,j-1]$, ar so t_j is about $j/2$. The resulting average-case running time turns out to be quadratic function of the input size, just like the worst-case running time.

In some particular cases, we shall be interested in the ***average-case*** running tin of an algorithm; we shall see the technique of ***probabilistic analysis*** applied various algorithms.

The scope of average-case analysis limited, because it may not be apparent what constitutes an "average" input f a particular problem. Often, we shall assume that all inputs of a given size a equally likely. In practice, this assumption may be violated, but we can sometim use a ***randomized algorithm***, which makes random choices, to allow a probabilist analysis and yield an ***expected*** running time.