Analysis of insertion sort

- The time taken by an algo depends on input

  eg sorting 500 items takes longer than sorting 5 numbers.

- A sorting algorithm may take different amounts of time on two inputs of the same size

  eg insertion sort can take diff time to sort two inputs of same size depending on how nearly sorted they already are.

- In general, the time taken by an algo grows with the size of input, therefore, the running time of an algorithm is described as a function of input size.

- Input size
  - it depends on the problem being studied.
  - usually, the number of items in the input

    eg the array size n for sorting problem.

    But it could be something else.

  - it can be described by more than one number

    eg graph algo ⇒ (n, e) → no. of edges

Running time

- The running time of an algo. on a particular input is the number of primitive operations / steps executed.
- For simplicity, we will define the concept of step in such a way that is as <u>machine - independent</u> as possible.

We will use following interpretation :—

" A constant amount of time is required to execute each line of our pseudocode.

- One line may take a different amount of time than another, but each execution of line i takes the amount of time c_i.

(this is in accordance with our RAM model)

⇒ this is assuming that the line consists of only primitive operations.

- Rewriting insertion-sort procedure with the time cost of each statement and the number of times each statement is executed

Insertion-sort (A)	time cost	no. of times
1. for $j \leftarrow 2$ to length[A]	c_1	
2. key $\leftarrow$ A[j]	c_2	
3. ▷ insert A[j] into the sorted sequence A[1 .. j-1]	c_3	
4. $i \leftarrow j-1$	c_4	
5. while $i > 0$ and $A[i] >$ key	c_5	
6. $A[i+1] \leftarrow A[i]$	c_6	
7. $i \leftarrow i-1$	c_7	
8. $A[i+1] \leftarrow$ key	c_8	

it should be noted that :

- comments are not executable statements, and so they take no time. Here $c_3 = 0$
- when a looping construct (for, while) exits then the test (in the loop header) is executed one time more than the loop body
- Here, we assume that for each $j = 2, 3 \dots n$; the inner "while loop test" is executed t_j times.

The running time of an algo. is the sum of running times of each statement executed.

⇒ if a statement takes c_i time to execute and is executed n times will contribute $c_i n$ to the total running time

$$\Rightarrow \text{Running time} = \sum_{\text{all statements}} (\text{time cost of statement}) \cdot (\text{number of time statement is executed})$$

Here, for insertion sort, the running time $T(n)$ is calculated as follows:

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

- we know that even for inputs of a given size, an algorithm's running time may depend on which input of that size is provided!

Here, for insertion sort, the best case occurs if array is already sorted.

⇒ For each $j = 2, 3 \dots n$ we found that $A[i] \leq key$ (in line 5 when i has its initial value of $j-1$)

Thus $t_j = 1$

and $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$

(there is no term of c_6 & c_7)

$\Rightarrow T(n) = an + b$ (linear fn. of n)

- If array is in reverse sorted order, then the worst case occurs.

⇒ we have to compare each element $A[j]$ with each element in the sorted subarray $A[1..j-1]$

therefore, $t_j = j$ for $j = 2, 3 \ldots n$

so $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left\{\frac{n(n+1)}{2} - 1\right\}$

$+ c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$

$$\boxed{T(n) = an^2 + bn + c}$$ (quadratic fn. of n)

<u>Note</u>: Usually, we will focus on calculating "worst-case running time" which is the longest time for any input of size n.

Because of following observations:

(1) The worst-case running time of an algo is an upper bound on the running time for any input (of given size)

⇒ it gives us guarantee that the algo will never take any longer.

⇒ no need to make some educated guess about the running time.

(2) For some algos, the worst-case occurs fairly often.

eg searching a particular item in the database, the search algo's worst-case will occur when the item is not present in the database.

(3) The average-case is <u>often</u> roughly as bad as the worst case

eg for insertion-sort the average-case running time is also quadratic fn. of n (just like the worst-case running time)

☆☆

Order of growth

- In order to simplify our procedure for algorithm analysis, we have used different level of abstractions such as
  - we ignored actual cost of each statement (by using the constants c_i to represent these costs)
  - Further, we observed that even these constants give us more detail than we really need, so again we abstracted these constants. (by using some constants a, b, c)
- Furthermore, we can make one more simplifying abstraction, by considering the rate of growth (order of growth) of the running time

⇒ therfore, we consider only the leading term of the running time formula (eg an^2) because lower-order terms are relatively insignificant for large n

⇒ we also ignore the leading term's constant, since constant factors are less significant than

Order of Growth

- The order of growth of the running time of an algorithm provides a simple characteristic of the algorithm's efficiency.
- It enables us to compare the relative performance of alternative algorithms.
- Although we can determine the exact running time of an algorithm, but the extra precision is not usually worth the effort of computing it. (eg. as we did for insertion sort)

For large enough inputs, the multiplicative constants and lower-order terms of an exact running time are dominated by the effects of the input size itself.

- when we look at input sizes large enough to make only order of growth of the running time relevant, we are studying the asymptotic efficiency of algorithms. (it means, we are concerned with how the running time of an algorithm increases with the size of input in the limit, … t increases without bound.

- Usually, one algorithm is considered to be more efficient than another if its worst-case running time has lower order of growth.
- There exists several standard ways for simplifying the asymptotic analysis of algorithms..