Getting Started

Manuel Serna-Aguilera

Insertion Sort

Sorting algorithms solve the sorting problem, given a sequence of numbers $< a_1, a_2, ..., a_n >$, the result is a permutation $< a_1^{'}, a_2^{'}, ..., a_n^{'} >$ of the input sequence such that $a_1^{'} \leq a_2^{'} \leq ... \leq a_n^{'}$. The values are also referred to as keys in the book, which can make things a bit confusing.

Insertion sort is efficient for sorting a small number of values. Insertion sort works by–quite simply–inserting values into the sorted subsequence, that is, when you insert a new value, call it a_i, into the subsequence $< a_1^{'}, a_2^{'}, ..., a_k^{'} >$ for $k \leq n$, you compare a_i with $a_k^{'}$. If a_i is smaller, now compare it with $a_{k-1}^{'}$, and so on and so forth until a_i is not smaller, in which case every value bigger than it will be shifted one place to the right to make place for the newly inserted value. Now the list is sorted again, then add another value if there are any remaining.

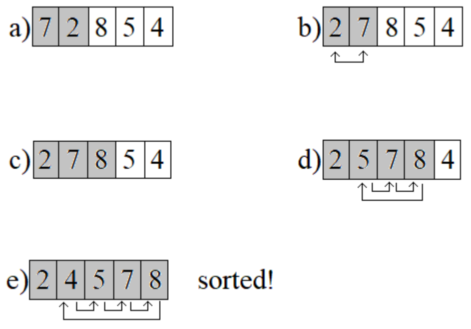

Figure 1: The process of insertion sort. **(a)** Start with the first two values–7 and 2. **(b)** 2 and 7 are properly sorted. **(c)** Consider the third value, 8 is already greater than 7, so do not swap. **(d)** Consider the value 5, insert it after 2, which means iteratively swapping values towards the left to make room. **(e)** Place value 4 in between 2 and 5, adjust like in (d).

Here is some pseudo code from the book (without comments).

```
$A$        // some array A with n values

INSERTION-SORT($A$)
1  for $j = 2$ to $A.length$
2      $key = A[j]$
3      $i = j$ - 1
4      while $i > 0$ and $A[i] > key$
5          $A[i+1] = A[i]$
6          $i = i + 1$
7      $A[i+1] = key$
```

Analysis of Insertion Sort

Writing pseudo code is neat and all, but it would be nice if we could understand its running time, below is how the book approaches tackling running time, and it displays it neatly. Simply put, write a cost for each line, and the number of times that line may run.

Line	Cost	Times
1	c_1	n
2	c_2	$n-1$
3	c_3	$n-1$
4	c_4	$\sum_{j=2}^{n} t_j$
5	c_5	$\sum_{j=2}^{n}(t_j-1)$
6	c_6	$\sum_{j=2}^{n}(t_j-1)$
7	c_7	$n-1$

On an input of n values, sum the products of the cost and times columns, this gives

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

For insertion sort, the best case occurs when the input is already sorted, giving

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + 0 + 0 + c_7(n-1) \\ &= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) \end{aligned}$$

This can be expressed in the form $an+b$, and will give a best case running time of $\Theta(n)$.

The worst-case occurs when the input is in reverse-order, this gives

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

This can be expressed as an^2+bn+c, and will give the form of $\Theta(n^2)$, the worst case.