On the Science Behind Computing (Part I)

Samir Khuller *

1 Insertion Sort

Suppose we are given an array A of numbers (we assume for simplicity that these are integers) the objective is to put them in non-decreasing order. For example, if the input is $100, 3, 201, 17, 91$ then the output should be $3, 17, 91, 100, 201$. However, the sorting operation can be defined for any set of objects that have some order imposed on them – by this we really mean that a notion of "comparison" exists. For example, given a set of words such as `Ant, Aardvark, Tiger, Lion, Zebra` the sorted order would be `Aardvark, Ant, Lion, Tiger, Zebra` using the alphabetical ordering used in a dictionary for example (compare the first letter, then the second if the first letters match etc).

This is an easy process if we do not need to order a large number of objects. However, computers deal with a very large amount of data – how can we quickly order a million numbers?

This leads us to a simple algorithm for ordering data called Insertion Sort. (The main insight comes from mathematical induction.) Let us first consider the following simple insight – suppose we have a set of n elements that we need to sort. Imagine for a minute that we have already sorted the first $n-1$ elements; then inserting the last element $A[n]$ in the correct position can be accomplished as follows. First compare $A[n]$ with $A[n-1]$, if $A[n] \geq A[n-1]$ then there is nothing to do, since $A[1] < A[2] < A[n-1] (< A[n])$! Otherwise, we exchange $A[n]$ and $A[n-1]$ and compare $A[n-1]$ (old $A[n]$) with $A[n-2]$. If it is at least as large as $A[n-2]$, then there is nothing more to do, otherwise we exchange $A[n-2]$ with $A[n-1]$ and continue until we find the correct place for the element.

Formally, the way the algorithm works is as follows: we scan the array from left to right. At the end of each iteration our objective is to ensure that the elements $A[0] \ldots A[i]$ are in sorted order (we do this for each value of i going from 0 to $n-1$). If we assume that $A[0] \ldots A[i-1]$ are in sorted order, then our goal is to insert $A[i]$ into the correct place. To do this we take all elements $A[j]$ $(j < i)$ and if $A[j] > A[i]$, we move these elements to the right by one. This way we are able to find the first spot where we can insert $A[i]$. If $A[i-1] < A[i]$ then there is nothing to do and all the elements until $A[i]$ are in sorted order.

This sorting algorithm is actually not very fast. For some input instances (like an array where the elements are close to the sorted order, the algorithm is really quick). However, there are cases when this program takes time roughly proportional to the square of the number of elements! To see this consider an array where all the elements are in decreasing order. Each element i will cause the inner while loop to move almost $i-1$ elements over. Summing this over all values of i gives us the bound $\sum_{i=1}^{n}(i-1) = \frac{n(n-1)}{2}$.

We also include a Ruby program.

*Department of Computer Science, University of Maryland, College Park, MD 20742. E-mail : samir@cs.umd.edu.

```
Arr = Array[4,10,6,8,7,2,1,9,0]

def insertsort()

Arr.length.times{ |i|
  cur_elem = Arr[i]
  j = i - 1
 while(j >= 0 and Arr[j] > cur_elem)
  Arr[j+1] = Arr[j]
  j = j - 1
 end
 Arr[j+1] = cur_elem
}
 return Arr
end

A_sorted = insertsort()
A_sorted.each{ |k| puts k}
```