Sorting

Manoj Gupta

August 21, 2016

In algorithm design, after designing your algorithm, you have to show the following:

1. Show that your algorithm is correct.
2. Find its running time.

Till now, we have vaguely argued that our algorithm is correct. In this lecture, we will see how to formally show that your algorithm is correct. Though there is no single method to prove that your algorithm is correct, it is advisable to learn all the methods that are previously employed. In this context we will first take a look at a very simple problem of finding the minimum. (See Figure 1)

```
1 min ← A[1];
2 foreach i ← 2 to n do
3 |  if A[i] < min then
4 |  |  min ← A[i];
5 |  end
6 end
```

Figure 1: Finding the minimum

We all believe that this piece of code will find the minimum. But how do we prove such that there is no doubt in the mind of the reader. To this end, we will try to find some property of this procedure which is always true (**in every iteration of the for loop**). Such a property is called an *Invariant* as the property does not vary as the algorithm proceeds. In this case, this invariant is called the *loop invariant* as the property will hold for each iteration of this for loop. It is not hard to come up with a loop invariant:

Lemma 0.1. *After the* i*th iteration of the for loop,* min *contains the minimum element if* $A[1 \dots i]$.

Proof. We will prove by induction. The base case is $i = 1$, that is before the start of the for loop. In this case, we have set $min \leftarrow A[1]$. So trivially, min is the minimum element of the array $A[1 \dots 1] = A[1]$.

Induction hypothesis: We will assume that after iteration $i - 1$, min contains the minimum element of the array $A[1 \dots i - 1]$.

Induction Step: We will now prove that min contains the minimum element of the array $A[1 \dots i]$ after the ith iteration. Using induction hypothesis, we know that the min contains the minimum element of the array $A[1 \dots i - 1]$ after iteration $i - 1$. At iteration i, there are two possible cases:

1. $min \leq A[i]$

   This means that min is $\leq$ all the elements $A[1 \dots i]$. Since we don't change the minimum if this condition is true, min contains the minimum element of array $a[1 \dots i]$ after the ith iteration.

2. $min > A[i]$

   This means that $A[i]$ is less than all the elements in $A[1 \ldots i-1]$. Since we change the minimum if this condition is satisfied, min contains the minimum element of $A[1 \ldots i]$ after iteration i.

Thus, we see that under both the conditions, the statement of the lemma holds. □

Even though the above example was simple, it shows some basic proof techniques:

1. Finding the loop invariant of your algorithm.

   In fact, such a technique is not just limited to this particular example. Finding invariant is like finding some basic feature of your algorithm. In my opinion, one needs to find such an invariant in any algorithm.

2. Using Induction to prove it.

   Having found a basic feature of your algorithm. we can now use various proof techniques to prove it. Induction is one of such many techniques. Though there may be many ways to prove an invariant, in this section we will see how induction has been put to use for a very large basic class of algorithms called **Sorting**, which is defined as follows:

Question 1. *Given an array of integers $A[1 \ldots n]$ of size n, design an algorithm which sorts this array in ascending order.*

A very simple way to sort integers is by handling elements one by one. So at the starts you have an array of size 1. Note that array of size 1 is always sorted. Now, we add the second element of the array which may lead to an unsorted array of size 2. To fix this issue, we check is $A[2] < A[1]$ and if it is then we swap $A[1]$ and $A[2]$. This process can be naturally extended to the iteration i when the ith element is added to the array. So after ith iteration, let us assume that the array $A[1 \ldots i-i]$ is sorted. The addition of $A[i]$ may lead to an unsorted array. To fix this issue, we again resort to comparing and swapping. However, this time we have to find an index $1 \leq j \leq i$ such that all the elements below index j are less than $A[i]$ and all the elements above index j are greater than $A[i]$. It is as if we are **inserting** $A[i]$ in its right position in the sorted array $A[1 \ldots i-1]$. Such a sorting algorithm is called insertions sort which is coded in Figure 2.

```
1  foreach i ← 2 to n do
2  |   foreach j ← i to 2 do
3  |   |   if A[j] < A[j − 1] then
4  |   |   |   swap(A[j], A[j − 1]);
5  |   |   end
6  |   |   else
7  |   |   |   break;
8  |   |   end
9  |   end
10 end
```

Figure 2: Insertion Sort

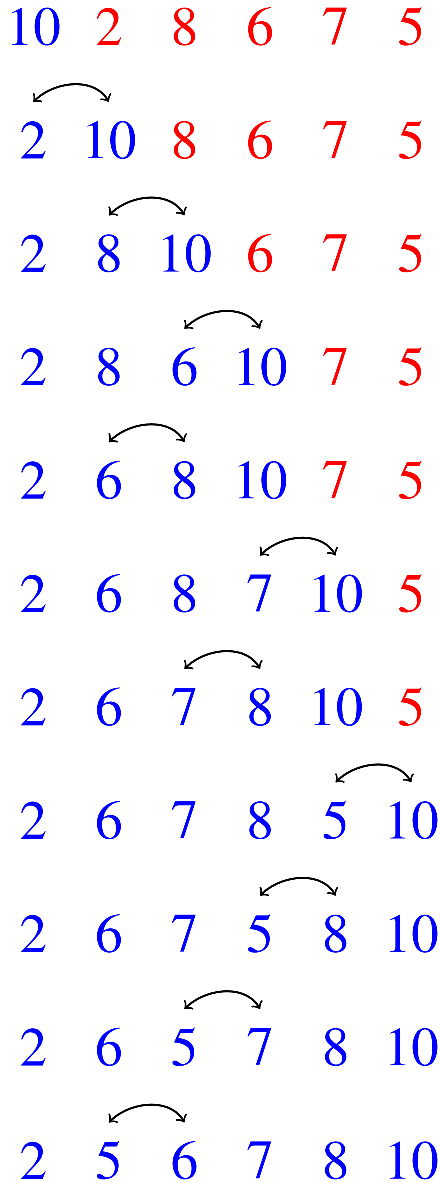

Figure 3: A run of Insertion Sort

The loop invariant for insertion sort is as follows

Lemma 0.2. *After the ith iteration of Insertion sort, the array $A[1 \dots i]$ is sorted.*

The reader is encouraged to prove the above lemma (again using induction). We now try to find the running time of Insertion sort. A careful reader might have realized that Insertion sort take $O(n^2)$ time in the worst case. In fact, for the following array $A = \{n, n-1, n-2, \dots, 3, 2, 1\}$, insertion sort take $O(n^2)$ time. Similarly, for array $A = \{1, 2, 3, \dots, n-2, n-1, n\}$, insertion Sort takes $O(n)$ time. How does then insertion sort perform in general. For this we find the average case running time(AVG) of our algorithm which is defined as follows: assume that S denote the set of all the inputs to insertion sort.

$$\text{AVG} = \frac{\sum_{S' \in S} \text{Time taken by Insertion Sort on S'}}{|S|}$$

To complete the calculation, one needs to find the set of all the possible inputs to our algorithm. Consider the following two different sequence {10 3 20 5} and {2345 100 6789 200}. One can check that the execution of Insertion on both these sequences are same. In fact, the values of each element in the sequence does not influence the running time of the algorithm, but the ordering of these elements matter. Thus both these sequence are isomorphic to {3 1 4 2}. Hence, we will assume that the input sequence contains numbers from 1 to n. So, the total number of different sequences on n numbers is $n!$. Thus, $|S| = n!$.

Consider any sequence $S' \in S$. Let $T_2(S')$ be the time taken by Insertion sort in the second iteration of the for loop ($i = 2$). Similarly, let $T_i(S')$ be the time taken by Insertion sort in the ith iteration of the for loop (where $2 \leq i \leq n$). Thus, $\textsc{Avg} = \frac{\sum_{S' \in S} \sum_{i=2}^{n} T_i(S')}{n!}$. And, $\textsc{Avg} = \frac{\sum_{i=2}^{n} \sum_{S' \in S} T_i(S')}{n!}$. Let $C_i = \frac{\sum_{S' \in S} T_i(S')}{n!}$. Thus, $\textsc{Avg} = \sum_{i=2}^{n} C_i$.

Note that C_i is the average time taken by the insertion sort in the ith iteration of the algorithm. At the start of the ith iteration, by Lemma 0.2, $A[1 \dots i-1]$ is sorted. Thus, after element $A[i]$ is inserted, $A[i]$ can be placed at i positions namely $\{1, 2, 3, \dots i\}$. For any $1 \leq j \leq i$, if the final place of $A[i]$ is j, the running time at the ith iteration is $O(j)$. Let U_j $(1 \leq j \leq i)$ be the set of all sequences in S such that Insertion sort takes $O(j)$ in its ith iteration.

We now prove the following lemma:

Lemma 0.3. $|U_j| = |U_k|$ *for* $1 \leq j \neq k \leq i$.

Proof. We claim that $|U_j| = \binom{n}{i}(i-1)!(n-i)!$. We first choose any i elements from n elements. Out of these i elements, we first find an element a_j which is the jth smallest among these i elements. a_j will be the ith element in sequences of U_j. The rest of the $(i-1)$ elements can be populated at any positions from 1 to $i-1$. The number of ways to do so is $(i-1)!$. There are $(n-i)$ elements left which can be placed at index $[i+1 \dots n]$ in $(n-i)!$ ways. Thus $|U_j| = \binom{n}{i}(i-1)!(n-i)! = n!/i$ and $|U_j| = |U_k|$ for $1 \leq j \neq k \leq i$. $\square$

From lemma 0.3, $|U_j| = n!/i$. Thus $\sum_{j=1}^{i} |U_j| = n!$. C_i now can be calculated as follows:

$C_i = \frac{\sum_{j=1}^{i} |U_j| \times j}{n!}$, where j is the time required by insertion sort to process sequences in $|U_j|$ at iteration i. Using lemma 0.3, $C_i = |U_1| \sum_{j=1}^{i} j$, since $|U_1|$ is equal to each other U_j. Thus, $C_i = \frac{n!}{i} \frac{i(i+1)}{2n!} = \frac{i+1}{2}$. Thus, $\textsc{Avg} = \sum_{i=2}^{n} C_i = \sum_{i=2}^{n} \frac{i+1}{2} = O(n^2)$.

Thus, we conclude that the average running time of Insertion sort is $O(n^2)$.