Insertion Sort

Algorithm 1: Insertion Sort

```
Data: An array A[0 ... n-1] of size n
Result: Sorted array A in non decreasing order i.e
        A[0] <= A[1] <= ... <= A[n − 1]
begin
    i ← 1;
    while i < n do
        tmp ← A[i];
        j ← i − 1;
        /* Insert A[i] into sorted sub-array A[0..i-1]      */
        while j >= 0 and A[j] > tmp do
            A[j + 1] ← A[j];
            j ← j − 1;
        end
        A[j + 1] ← tmp;
        i ← i + 1;
    end
end
```

1 Proof of Correctness

1.1 Invariant

At the beginning of each outer loop, the sub array $A[0 \,..\, i-1]$ is sorted and consists of the same elements that were originally in $A[0 \,..\, i-1]$.

1.2 Initialization

Initially, $i = 1$ and hence the sub array $A[0 \,..\, i-1]$ consists of only one element i.e $A[0]$. As a single element is trivially sorted, the invariant holds.

1.3 Maintenance

The inner while loop shifts elements $A[i-1]$, $A[i-2]$ and so on by one position to the right until the right position for $A[i]$ is found. The element $A[i]$ is then inserted into it's correct position. The sub array $A[0 \,..\, i = j+1]$ contains the original elements of $A[0 \,..\, i = j+1]$ but in ascending order. Incrementing i to $i+1$ for the next iteration, then preserves the invariant.

1.4 Termination

The procedure terminates when $i >= n$. As i is always incremented by 1, when the outer while loop terminates i will always be equal to n. According to the invariant, when $i = n$, the sub array $A[0 \,..\, n-1]$ must contain the same original elements of the array but in ascending order. In other words, the array A is now sorted. ■