

# Induction as a Problem-Solving Technique

Induction, as a mathematical term, is a proof technique for theorems about numbers. For example, let  $T(n)$  be a theorem about integers:

$$T(n) : \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

Then a *proof by induction* would consist of the following steps:

1. Prove that  $T(1)$  is true.
2. Prove that if  $T(1), T(2), \dots, T(n-1)$ , and  $T(n)$  are true, then  $T(n+1)$  is also true.

The first part is easy - simply evaluate  $T(1)$ :

$$\sum_{i=1}^1 i = \frac{1(1+1)}{2} = 1$$

This is certainly true. Now, assuming that for any  $k \leq n$ , the theorem is true, we wish to prove:

$$T(n+1) : \sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$$

Another way of writing  $T(n+1)$  is:

$$T(n+1) : \sum_{i=1}^n i + (n+1)$$

This can be transformed into:

$$\begin{aligned} \frac{n(n+1)}{2} + (n+1) &= \frac{n^2}{2} + \frac{3n}{2} + \frac{2}{2} \\ &= \frac{(n+1)(n+2)}{2} \end{aligned}$$

Which is what we want to prove.

Induction also provides a means for expressing a computer algorithm as well. If  $x_i$  is the solution to a problem of size  $i$ , then solving for a solution of size  $n$  involves:

1. Find an algorithm **A** that solves the problem for  $i = 1$  (or some other trivial case).

2. Find an algorithm **B** which given the inputs  $x_1, x_2, \dots, x_i$  and solves the problem for  $i + 1$ .

Then, we put these two parts together to form a complete algorithm, using the following framework:

```
x[0] = A();    // written to accommodate C's zero-based arrays.
i = 0;
while(i < n)
{
    x[i + 1] = B(x[i]);
    i++;
}
```

### A Simple Example

Suppose we wish to find the largest value in an array using induction. According to the rules above, we first need an algorithm that solves the problem for  $i = 1$ , i.e., for a single element array. This is trivial:

```
{ // Algorithm A
    maxval = x[0];
    maxind = 0;
}
```

Next, we need an algorithm, which given that we have the largest value for the first  $i$  elements, we can find the largest value for the first  $i + 1$  elements. This is also easy – assume that the variables `maxval` and `maxind` contain the largest value and the index of the largest value for  $i$  elements:

```
{ // Algorithm B
    if(x[i + 1] > maxval)
    {
        maxval = x[i + 1];
        maxind = i + 1;
    }
}
```

Putting these two algorithms together provides a complete solution:

```
// Algorithm A
maxval = x[0];
maxind = 0;
i = 0;
while(i < n)
{ // Algorithm B
    if (x[i + 1] > maxval)
    {
        maxval = x[i + 1];
        maxind = i + 1;
    }
    i++
} // end while
```

## A More Complex Example

A polynomial is a mathematical function that has the general form:

$$\begin{aligned} p(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_nx^n \\ &= \sum_{i=0}^n a_i x^i \end{aligned}$$

The obvious way to evaluate the polynomial is to calculate each term independently and then add it to the sum. However, another way to formulate  $p(x)$  is called *Horner's method*; it is derived below:

$$\begin{aligned} p(x) &= \sum_{i=0}^n a_i x^i \\ &= a_0 + x \sum_{i=0}^{n-1} a_{i+1} x^i \\ &= a_0 + x \left( a_1 + x \sum_{i=0}^{n-2} a_{i+2} x^i \right) \\ &= a_0 + x \left( a_1 + x \left( a_2 + x \sum_{i=0}^{n-3} a_{i+3} x^i \right) \right) \\ &\quad \vdots \\ &= a_0 + x \left( a_1 + x \left( a_2 + x \left( a_3 + \cdots + x \left( a_{n-1} + a_n x \right) \cdots \right) \right) \right) \end{aligned}$$

This reformulation provides the basis for a more efficient, induction based algorithm by starting from the inner-most parenthesis and calculating the value of the next outer parentheses by using the previously calculated inner value:

$$\text{NextOuterValue} = a[n-i] + x * \text{LastInnerValue};$$

where  $n$  is the degree of the polynomial and  $i$  is a count of how many sets of inner parenthesis values have been calculated so far. Now, to create the induction algorithm, algorithm A is the inner-most value, and algorithm B is the equation given above.