Algorithms

- Consider an array a_1, a_2, ..., a_n of numbers, and suppose we want to know if a particular number b appears in the list. The following algorithm returns the *first* position i for which $a_i = b$ (if such an i exists), and returns 0 if no such i exists

```
1: LinearSearch(a_1, a_2, ..., a_n : array of reals, b:real)
2:   for i := 1 to n
3:     if a_i = b then
4:       return i
5:     end if
6:   end for
7:   return 0   // did not find b in the array!
```

- Running time analysis: Line 1 runs once, Lines 2-6 run at most n times each, Line 7 runs at most once. Hence, the running time is $O(1) + O(n) + O(1)$ which is $O(n)$

Worst case analysis vs. Average case analysis

- Notice that the analysis on the previous page is a "worst-case" analysis. The worst case occurs when *b* is not in the array, and the loop executes exactly *n* times, and the final line (Line 7) also gets executed.

- Alternatively, we could do an average case analysis that assumes something about how likely it is for *b* to be in the array, and where it is likely to be if it is in the array.

- We will almost always do a worst-case analysis, but if we knew that *b* was definitely in the array and was equally likely to be at any location *i*, we could compute the *average* number of times the loop executes as:

$$\frac{1+2+\ldots+n}{n} = \frac{1}{n}\sum_{k=1}^{n} k = \frac{n(n+1)}{2n} = \frac{n+1}{2} = O(n)$$

- Hence, the average running time is also $O(n)$