Figure 11.1 The median example (continued).

Sooner or later this scheme will find the median, but there are a couple of major problems here:

1. It's tedious; we have to do *N* comparisons of two numbers, *N* times, so it takes about $\boxed{\quad\quad}$ comparisons to find the median. We say this is an $\boxed{\quad\quad\quad}$ algorithm, while by contrast finding the average is $\boxed{\quad\quad\quad}$.
2. We need to be able to go over and over the list of numbers, so we should store the list first. We *could* use a Vector, but an array is more natural. Why?

So how do we use an array? An array **A** of integers is an object that contains a lot of separate integers. Unlike other objects, though, we don't need to use an explicit method call to retrieve the elements. Instead, there's a special notation:

> **A[0]** is the array element at index **0**
> (i.e., in our case, the first integer in **A**).
> **A[1]** is element **1**, the second integer.
> . . .
> **A[N-1]** is the **N**-th integer.

As in a Vector, the indices can be expressions (of integer type):

> **A[i]** is the **i**-th element.
> (That's right, it's **not** the (**i+1**)-st element!)
> **A[i+1]** is the (**i+1**)-st element.

So to count the number of elements that are less than 2,

```
int count = 0;
for (int i = 0; i < N; i++)
    if (A[i] > 2) count++;
System.out.println("# of elements < 2: " + count);
```

But before doing this, of course, we must read the contents of **A** from the input stream:

```
for (int i = 0; i < N; i++) {
    String line = keyb.readLine();
    A[i] = Integer.parseInt(line);
}
```

All of this looks pretty straightforward …except how did we get the value of **N** before we started to read the list of integers?