Tune Your App Using Algorithms and Data Structures

In this chapter, you will learn about:

- How bad algorithms and data structure can affect your app’s performance.
- Theoretical issues of measuring your algorithms.
- Practical measurement of your app’s performance.
- Main data structures and algorithms:
  - iPhone data structures: NSSet, NSArray, and NSDictionary.
  - Other important data structures and their implementations.
  - Other algorithms and problem-solving approaches:
    - Recursion
    - SAX vs. DOM in XML parsing

You may hear that in mobile phone development you don’t need to worry about algorithms and data structure due to the computing power of the server side. However, as mentioned with regards to caching issues in the previous chapter, especially when your phone is offline, you should store your data locally and compute it within the phone environment. Here is the issue: your phone environment is not as powerful as your server environment. In other words, you don’t have the power of cloud computing or a data center.
First Example

This first example will show you how a bad algorithm can affect your program when it runs in the strict mobile phone environment.

My sample code is simple:

- In the first benchmark, the example contains two arrays, each of which contains 1,000 elements. The first and second arrays are the same in terms of the number of elements. I loop through two arrays to check how many common elements are contained between the two arrays.

- In the second benchmark, the example contains two sets, each of which contains 1,000 elements. I use a special method inside the NSSet API to get the set of common elements between the two original sets.

Then, I benchmark each of them based on a simulator, a new device (iPhone 4 with iOS4), and an old device (iPhone 3G with iOS3). The results are shown in Table 5–1.

Table 5–1. Benchmark Test Results Between Different Data Structures and Algorithms

<table>
<thead>
<tr>
<th>Environment</th>
<th>First Benchmark with Arrays</th>
<th>Second Benchmark with Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulator</td>
<td>0.099</td>
<td>0.001</td>
</tr>
<tr>
<td>New Device</td>
<td>0.9</td>
<td>0.0084</td>
</tr>
<tr>
<td>Old Device</td>
<td>5.44</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 5–1 shows that the second benchmark is 100 times faster than the first benchmark. Considering that 1,000 items is not practical when you deal with real applications, 5.44 seconds may not be much for new devices. However, remember that there are many old devices with less power out there, and their owners may not update these devices any time soon. In that light, 5.44 seconds is actually a significant delay when running in those old devices.

What follows is a look at the initial source code and few explanations before I move on to explain some concepts in-depth.

The purpose of Listings 5–1 and 5–2 is to count how many objects inside the first array/set also belong to the second array/set. Listing 5–1 solves the problem by using an array and loop while Listing 5–2 solves the problem by using a set.

Listing 5–1. First Benchmark Using an Array and Loop

```objective-c
// [self defaultData] returns an array with 1000 thousands different NSObjects
NSArray *myFirstArray = [NSArray arrayWithArray:[self defaultData]];
NSArray *mySecondArray = [NSArray arrayWithArray:[self defaulData]];
NSDate *date1 = [NSDate date];
```