5.3.2 An Improved $O(N^2)$ Algorithm

When we remove a deeply nested loop from an algorithm, we generally lower the running time.

If we can remove a loop from the algorithm, we genereally can lower the running time. How do we remove a loop? Obviously, we cannot always do so. However, the previous algorithm has many unnecessary computations. The inefficiency that the improved algorithm corrects can be seen by noticing that since $\sum_{k=i}^{j} A_k = A_j + \sum_{k=i}^{j-1} A_k$ the computation in the inner `for` loop in Figure 5.4 is unduly expensive. Put another way, suppose we have just calculated the sum for the subsequence extending from i to $j - 1$. Then computing the sum for the subsequence extending from i to j should not take long because we need only one more addition. However, the cubic algorithm throws away this information. If we use this observation, we obtain the improved algorithm shown in Figure 5.5. We have two rather than three nested loops, and the running time is $O(N^2)$.

```
1     /**
2      * Quadratic maximum contiguous subsequence sum algorithm.
3      * seqStart and seqEnd represent the actual best sequence.
4      */
5     public static int maxSubsequenceSum( int [ ] a )
6     {
7         int maxSum = 0;
8
9         for( int i = 0; i < a.length; i++ )
10        {
11            int thisSum = 0;
12
13            for( int j = i; j < a.length; j++ )
14            {
15                thisSum += a[ j ];
16
17                if( thisSum > maxSum )
18                {
19                    maxSum = thisSum;
20                    seqStart = i;
21                    seqEnd   = j;
22                }
23            }
24        }
25
26        return maxSum;
27    }
```

Figure 5.5 Quadratic maximum contiguous subsequence sum algorithm