1. Thesection“Rule4”describedaContainsDuplicatesalgorithmthathas runtime O(N2). Consider the following improved version of that algorithm:

Boolean: ContainsDuplicates (Integer: array [])

// Loop over all of the array's items except the last one.

For i = 0 To <largest index> - 1

// Loop over the items after item i.

For j = i + 1 To <largest index>

// See if these two items are duplicates.

If (array[i] == array[j]) Then Return True

Next j

Next i

// If we get to this point, there are no duplicates.

Return False

End ContainsDuplicates

What is the runtime of this new version?   
  
*Solution :  
The outer loop executes O(N) times in the new version of the algorithm. When the outer loop’s counter is i, the inner loop executes O(N – i) times. If we add up the number of times the inner loop executes, the result is N + (N – 1) +  
 (N – 2) + ... + 1 = N × (N – 1) / 2 = (N2 – N) / 2. The runtime of is O(N2).*

2. Table 1-1 shows the relationship between problem size N and various runtime functions. Another way to study that relationship is to look at the largest problem size that a computer with a certain speed could execute within a given amount of time.

For example, suppose a computer can execute 1 million algorithm steps per second. Consider an algorithm that runs in O(N2) time. In 1 hour the computer could solve a problem where N = 60,000 (because 60,0002 = 3,600,000,000, which is the number of steps the computer can execute in 1 hour). Make a table showing the largest problem size N that this computer could execute for each of the functions listed in Table 1-1 in one second, minute, hour, day, week, and year.

*Solution:*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Time | LOG2(n) | SQRT(n) | n | n2 | 2n | N! |
| Second | infinity | 1×1012 | 1×106 | 1,000 | 20 | 10 |
| Minute | infinity | 4×1015 | 6×107 | 7,746 | 26 | 12 |
| Hour | infinity | 1×1019 | 4×109 | 60,000 | 32 | 13 |
| Day | infinity | 7×1021 | 9×1010 | 293,939 | 36 | 14 |
| Week | infinity | 4×1011 | 6×1011 | 777,689 | 39 | 15 |
| Year | infinity | 1×1027 | 3×1013 | 5,617,615 | 45 | 17 |

1. Sometimes the constants that you ignore in Big O notation are important. For example, suppose you have two algorithms that can do the same job. The first requires 1,500 × N steps, and the other requires 30 × N2 steps. For what values of N would you choose each algorithm?    
     
   *Solution:   
     
   From question we understand that is asking us “For what N is 1,500 × N > 30 × N2? ” Solving for N gives 50 < N, so the first algorithm is slower if N > 50. I would use the first algorithm if N ≤ 50 and the second if N > 50.*

5. Suppose a program takes as inputs N letters and generates all possible unordered pairs of the letters. For example, with inputs ABCD, the pro- gram generates the combinations AB, AC, AD, BC, BD, and CD. (Here unordered means that AB and BA count as the same pair.) What is the algorithm’s runtime?    
  
*Solution:  
  
For the first letter we have N choices. After we select the first letter then we have n-1 choices for the second letter which give us N(n-1) choices in total.  
This includes counts for each pairs (AB and BA), so the total number of unordered pairs is N (N – 1) / 2.The Algorithm’s run time is O(N2).*

6.Suppose an algorithm with N inputs generates values for each unit square on the surface of an N × N × N cube. What is the algorithm’s runtime?    
  
*Solution:  
Since the cube has side lengths N, each side has an area of N2 . But since A cube has six sides the total surface area of all sides is 6 × N2. The algorithm runtime is O(N2).*

9. Can you have an algorithm without a data structure? Can you have a data structure without an algorithm?   
  
  
*Solution:   
An algorithm is a recipe for performing a certain task. A data structure is a way of arranging data to make solving a particular problem easier. A data structure could be a way of arranging values in an array, a linked list that connects items in a certain pattern, a tree, a graph, a network, or something even more exotic. The algorithm can’t exist without the data structure, and there’s no point in building the data structure if you don’t plan to use it with the algorithm.*

10. Consider the following two algorithms for painting a fence:

Algorithm1()

For i = 0 To <number of boards in fence> - 1

<paint board number i>

Next i

End Algorithm1

Algorithm2(Integer: first\_board, Integer: last\_board)

If (first\_board == last\_board) Then

// There's only one board. Just paint it.

<paint board number first\_board>

Else

// There's more than one board. Divide the boards

// into two groups and recursively paint them.

Integer: middle\_board = (first\_board + last\_board) / 2

Algorithm2(first\_board, middle\_board)

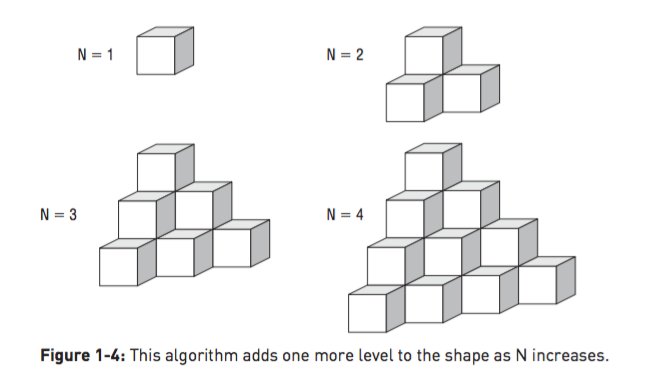
Algorithm2(middle\_board, last\_board)

End If

End Algorithm2

What are the runtimes for these two algorithms, where N is the number of boards in the fence? Which algorithm is better?   
  
Solution:  
 *The first algorithm simply paints the boards from one end to the other. It paints N boards and therefore has a run time of O(N).  The second algorithm divides the boards in a recursive way, but eventually it paints all N boards. Dividing the boards recursively requires O(log N) steps. Painting the boards requires O(N) steps. The total number of steps is N + log N, so the run time is O(N).*

8\*Suppose you have an algorithm that, for N inputs, generates a value for each small cube in the shapes shown in Figure 1-4. Assuming that the obvious hidden cubes are present so that the shapes in the figure are not hollow, what is the algorithm’s runtime?



From the way the shapes grow, we can probably guess that the number of cubes involves N3 in some manner. If we assume the number of cubes is A × N3 + B × N2 + C × N + D for some constants A, B, C, and D, we can plug in the values from and solve for A, B, C, and D. If we do that, we’ll find that the number of cubes is (N3 + 3 × N2 + 2 × N) ÷ 6, so the run time is O(N3).