HW1

**3.** 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 × N^2. For what values of N would you choose each algorithm?

**Answer:**

1500 x N 🡪 30 x N^2; If we observe both runtimes, we see that the equal state is reached when the value of N is 50 (N=50), so when N is less than 50, we would use first algorithm, and when N is greater than 50, we would use second algorithm.

**5.** Suppose a program takes as inputs N letters and generates all possible unordered pairs of the letters. For example, with inputs ABCD, the program 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?

**Answer:**

To choose the first element of the pair 🡪 N possibilities

To choose the second element of the pair 🡪 we already took one, hence we are left with N-1 possibilities

So the algorithm’s runtime would be N x (N-1) = N^2 – N (neglect the lower one) 🡪 **O(n^2)**

**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?

**Answer:**

N x N x N cube has 6 sides, and the surface for each of the sides which are the unit squares would be N^2 (for whole cube 6N^2). So algorithm’s runtime is **O(n^2)**

**9.** Can you have an algorithm without a data structure? Can you have a data structure without an algorithm?

**Answer:**

Yes there can be an algorithm without a data structure, because algorithm is a set of instructions to execute a task. However, you cannot have a data structure without an algorithm, because just to build the structure you an 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

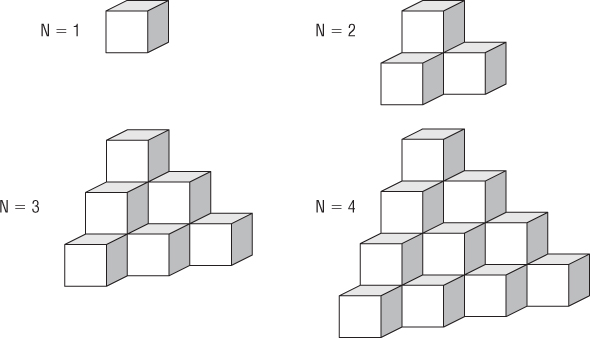
End Algorithm2

What are the runtimes for these two algorithms, where N is the number of boards in the fence? Which algorithm is better?

**Answer:**

Runtime for algorithm1 is O(n) because it paints one board in fence by time, and it goes from beginning to the end. In algorithm2, the fence is divided into two parts, but again it paints one board in fence by time in each portion (beginning-middle, middle-end), so the runtime for algorithm2 is O(n) as well.

**\*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?



1 🡪 1

2 🡪 4

3 🡪 10

4 🡪 20

**Answer:**

Looking the shapes I would say that the runtime is O(n^3), because the height, length, and depth, all three increase by the same number, as the input itself increases.