**Asymptotic running-time analysis for selection sort**

The total running time for selection sort has three parts:

1. The running time for all the calls to indexOfMinimum.
2. The running time for all the calls to swap.
3. The running time for the rest of the loop in the selectionSort function.

Parts 2 and 3 are easy. We know that there are n*n*n calls to swap, and each call takes constant time. Using our [asymptotic notation](https://www.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/asymptotic-notation), the time for all calls to swap is \Theta(n)Θ(*n*). The rest of the loop in selectionSort is really just testing and incrementing the loop variable and calling indexOfMinimum and swap, and so that takes constant time for each of the n*n*n iterations, for another \Theta(n)Θ(*n*) time.

For part 1, the running time for all the calls to indexOfMinimum, we've already done the hard part. Each individual iteration of the loop in indexOfMinimum takes constant time. The number of iterations of this loop is n*n*n in the first call, then n-1*n*−1n, minus, 1, then n-2*n*−2n, minus, 2, and so on. We've seen that this sum, 1 + 2 + \cdots + (n-1) + n1+2+⋯+(*n*−1)+*n* is an arithmetic series, and it evaluates to (n+1)(n/2)(*n*+1)(*n*/2)left parenthesis, n, plus, 1, right parenthesis, left parenthesis, n, slash, 2, right parenthesis, or n^2/2 + n/2*n*2/2+*n*/2n, start superscript, 2, end superscript, slash, 2, plus, n, slash, 2. Therefore, the total time for all calls to indexOfMinimum is some constant times n^2/2 + n/2*n*2/2+*n*/2n, start superscript, 2, end superscript, slash, 2, plus, n, slash, 2. In terms of big-Θ notation, we don't care about that constant factor, nor do we care about the factor of 1/2 or the low-order term. The result is that the running time for all the calls to indexOfMinimum is \Theta(n^2)Θ(*n*2).

Adding up the running times for the three parts, we have \Theta(n^2)Θ(*n*2) for the calls to indexOfMinimum, \Theta(n)Θ(*n*) for the calls to swap, and \Theta(n)Θ(*n*) for the rest of the loop in selectionSort. The \Theta(n^2)Θ(*n*2) term is the most significant, and so we say that the running time of selection sort is \Theta(n^2)Θ(*n*2).

Notice also that no case is particularly good or particularly bad for selection sort. The loop in indexOfMinimum will always make n^2/2 + n/2*n*2/2+*n*/2n, start superscript, 2, end superscript, slash, 2, plus, n, slash, 2 iterations, regardless of the input. Therefore, we can say that selection sort runs in \Theta(n^2)Θ(*n*2)time in all cases.

Let's see how the \Theta(n^2)Θ(*n*2) running time affects the actual execution time. Let's say that selection sort takes approximately n^2/10^6*n*2/106n, start superscript, 2, end superscript, slash, 10, start superscript, 6, end superscript seconds to sort n*n*n values. Let's start with a fairly small value of n*n*n, let's say n = 100*n*=100n, equals, 100. Then the running time of selection sort is about 100^2/10^6 = 1/1001002/106=1/100100, start superscript, 2, end superscript, slash, 10, start superscript, 6, end superscript, equals, 1, slash, 100 seconds. That seems pretty fast. But what if n = 1000*n*=1000n, equals, 1000? Then selection sort takes about 1000^2/10^6 = 110002/106=11000, start superscript, 2, end superscript, slash, 10, start superscript, 6, end superscript, equals, 1 second. The array grew by a factor of 10, but the running time increased 100 times. What if n = 1{,}000{,}000*n*=1,000,000n, equals, 1, comma, 000, comma, 000? Then selection sort takes 1{,}000{,}000^2/10^6 = 1{,}000{,}0001,000,0002/106=1,000,0001, comma, 000, comma, 000, start superscript, 2, end superscript, slash, 10, start superscript, 6, end superscript, equals, 1, comma, 000, comma, 000 seconds, which is a little more than 11.5 *days*. Increasing the array size by a factor of 1000 increases the running time a million times!