Analyzing Algorithms Part III

In this lesson, we will work out a generalization for the number of instructions executed for an array of length n

Generalizing Instruction Count

Let's try to generalize the running time for an array of size ***n***.

Code Statement	Cost
1. for (int i = 0; i < input.length; i++) {	executed **(2 x n)** + **2** times.
2. int key = input[i];	executed **n** times
3. int j = i - 1;	executed **n** times
4. while (j >= 0 && input[j] > key) {	We already calculated the cost of each iteration of the inner loop. We sum up the costs across all the iterations. Note that the inner loop will be executed ***n*** times and each execution will result in ***(iterations x 7)***+***2*** instructions being executed. And the number of iterations will successively increase from 0, 1, 2 all the way up to(n-1). See the dry run in the previous lesson to convince yourself of the validity of this result.
5. Inner loop statments	[(0 x 7) + 2] + [(1 x 7) + 2] + [(2 x 7) + 2] + ... [((n-1) x 7) + 2] = 2n + 7 * [0 + 1 + 2 + ... (n-1)]

	= 2n + 7 [$^{n(n-1)}/_{2}$]
11. } //inner loop ends	
12. }	

If we use the above formulas and apply an array of size 5 to them, then the cost should match with what we calculated earlier.

$$Total\ Cost = Outer\ loop\ instructions + Inner\ loop\ instructions$$

$$= [2*(n+1)+2n] + [2n + 7[\frac{n*(n-1)}{2}]]$$

$$= [2*(5+1)+10] + [10 + 7[\frac{5*(5-1)}{2}]]$$

$$= [12+10] + [10 + 7[\frac{5*4}{2}]]$$

$$= 22 + [10 + 7*10]$$

$$= 22 + 10 + 70$$

$$= 102\ instructions$$

Summation of Series

We glossed over how we converted the sum of the series 0 + 1 + 2 ...(n-1) to $^{n(n-1)}/_{2}$? However, even though we promised to steer clear of complicated mathematics as much as possible, this is one of the cardinal summations that you must know. Without a formal proof, remember that the sum of the first *k* natural numbers can be represented as:

$$1 + 2 + 3 + 4 \cdots k$$

$$= \frac{k*(k+1)}{2}$$

If you add the first 5 natural numbers the sum is:

$$
\begin{aligned}
&= \frac{k * (k+1)}{2} \\
&= \frac{5 * (5+1)}{2} \\
&= \frac{30}{2} \\
&= 15
\end{aligned}
$$

However, our summation sums up to n-1 and includes a zero. We can drop the zero because adding zero to anything yields the same value. We know:

$$1 + 2 + 3 + 4 \cdots (k-2) + (k-1) + k = \frac{k * (k+1)}{2}$$

We subtract k on both sides of the equation to get:

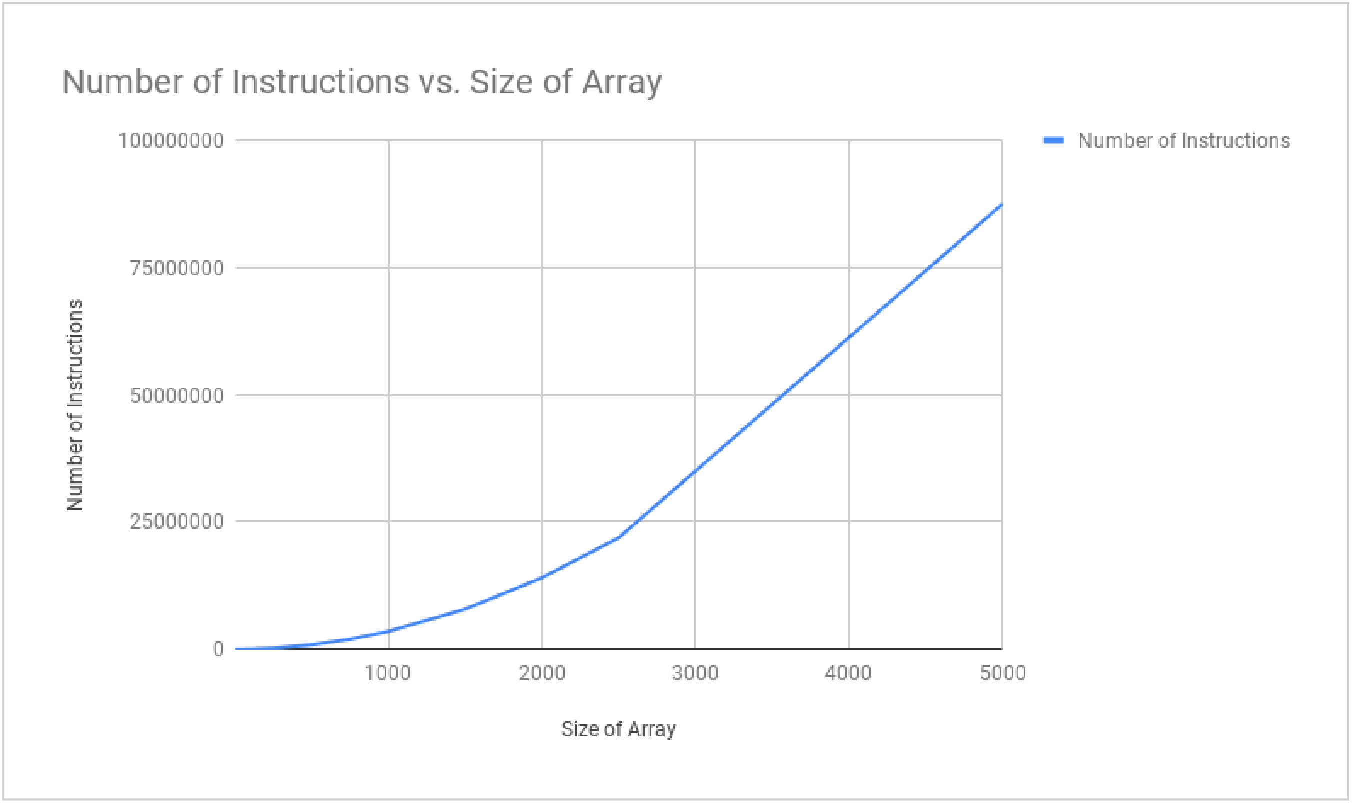
$$
\begin{aligned}
1 + 2 + 3 + 4 \cdots (k-2) + (k-1) &= \frac{k * (k+1)}{2} \\
1 + 2 + 3 + 4 \cdots (k-2) + (k-1) &= \frac{k^2 + k - 2k}{2} \\
1 + 2 + 3 + 4 \cdots (k-2) + (k-1) &= \frac{k^2 - k}{2} \\
1 + 2 + 3 + 4 \cdots (k-2) + (k-1) &= \frac{k * (k-1)}{2}
\end{aligned}
$$

Coming across this summation is very common in algorithmic analysis and, without getting too technical, if you can identify this series, then you know how to apply a formula to sum it up.

Running Time as *n* Increases

As the size of the array increases, so does the number of instructions executed. Below is a table that shows the instructions executed for the worst case as the size of the array increases.

Size of Array	Instructions Executed
5	102
10	377
25	2252
75	19877
100	35252
250	219377
500	876252
1000	3502502



Size of Array vs Number of Instructions Executed

The intention of doing the dry run and accounting for each line of code executed is to instill in the reader that the instruction count varies with the input size. Calculating down to this detail is often not necessary and with experience one can correctly recognize the complexity of various code snippets. As you'll learn in the next chapters, we reason about algorithm complexity in a ***ballpark*** sense, and calculating exact instruction counts is almost always unnecessary.