Big O Notation (Optional)

By now we've solved quite a few algorithms, but we may be wondering roughly how long it will take our function to run. One common way to represent this is called **Big O Time complexity**. Big O is not just a [giant robot anime](https://myanimelist.net/anime/567), it is a plot of the upper bound of how long a algorithm should take to run versus how big the problem is. These values are typically referred to in powers of **"N"** (size) with constants and lower power terms left out.

Imagine if we are searching an array to find its smallest value.

*function* smallest( arr ) {

*let* min = arr[0];

    for( *let* element of arr ) {

        if( element < min ) {

            min = element;

        }

    }

    return min;

}

smallest( [2, 5, 6, 12, 14, 28, 37, 41, 44, 45] );copy

In the above algorithm we will have to look through all **10** items to find the smallest value. As this is the same as the length of the array, it can be said that the algorithm took **"N"** number of operations, so we would say this algorithm runs in **O( N )**time.

If we were told that the array will always be sorted in ascending order, this algorithm could be rewritten to the following.

*function* smallest( sortedArr ) {

    return sortedArray[0];

}

smallest( [2, 5, 6, 12, 14, 28, 37, 41, 44, 45] );copy

This only takes **one** operation, as we don't need to loop through the array and can just return the first value. This algorithm would run in **O( 1 )**time. You can see this in the chart below.

| Big O | Description | Example |
| --- | --- | --- |
| O( 1 ) | Algorithms that can be completed in a constant number of operations. | Find the smallest value in a sorted array |
| O( N ) | Algorithms that may have to run **"N"** number of times | Find the smallest value in an unsorted array |
| O( N^2 ) | Algorithms that may have to go through **"N"** things **"N"** times | Bubble Sort |
| O( log(N) ) | Algorithms that can reduce the remaining pool of values by half or more for each value read through | Binary Search |
| O( N\*log(N) ) | Algorithms that can employ a divide and conquer approach | Quick Sort |

Some common Big O notations plotted

The steeper the line on the chart above, the more work is needed to solve that type of algorithm. If for some reason we needed to write an algorithm like the following...

for(*let* i=0; i<results.length; i++){

    for(*let* value of results) {

        for(*let* j=results.length-1; j>0; j--) {

            console.log(i, value, j);

        }

    }

}copy

we would say that this runs in **O( N^3 )** time. This would be a fairly steep curve and take a lot of time to run. If the length of results is **1000**,that console.log would get run **1,000,000,000** times.

**Note:** We might naïvely think that we can just count how many loops are nested within each other and that will be equal to the power of **"N"** for our Big O notation (which is true in many cases). This is only true if the loops could all run **"N"** times.