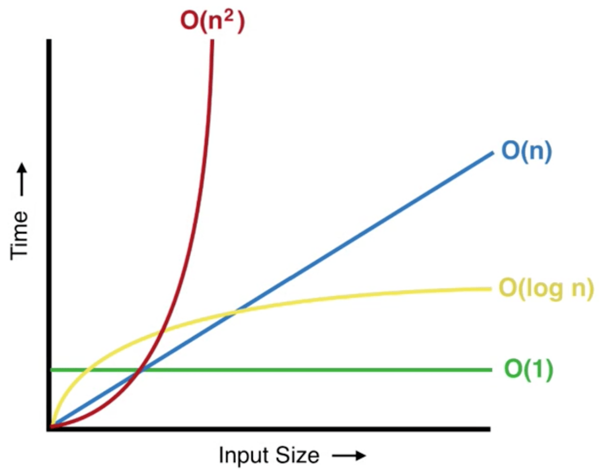
Session 5 - Some CS-ey things

UCAS Program 2020

Chloe Lau

August 2020

1 Big O Notation



The Big O Notation ($\mathcal{O}\sim$) is used in Computer Science to describe the performance or complexity of an algorithm. Big O specifically describes the worst-case scenario, and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

As it is hard to visualise easily what the complexity of an algorithm is, below are some common orders of growth along with descriptions and examples where possible.

2 Examples

2.1 $\mathcal{O}(1)$

$\mathcal{O}(1)$ describes an algorithm that will always execute in the same time (or space) regardless of the size of the input data set.

```
bool IsFirstElementNull(IList<string> elements)
{
    return elements[0] == null;
}
```

This is simple huh? Now let's get to other examples:

2.2 $\mathcal{O}(\mathrm{N})$

$\mathcal{O}(\mathrm{N})$ describes an algorithm whose performance will grow linearly and in direct proportion to the size of the input data set. The example below also demonstrates how Big O favours the worst-case performance scenario; a matching string could be found during any iteration of the `for` loop and the function would return early, but Big O notation will always assume the upper limit where the algorithm will perform the maximum number of iterations.

```
bool ContainsValue(IList<string> elements, string value)
{
    foreach (var element in elements)
    {
        if (element == value) return true;
    }

return false;
}
```

Still easy? Let's play with exponentials then!

2.3 $\mathcal{O}(N^2)$

$\mathcal{O}(N^2)$ represents an algorithm whose performance is directly proportional to the square of the size of the input data set. This is common with algorithms that involve nested iterations over the data set. Deeper nested iterations will result in $\mathcal{O}(N^3)$, $\mathcal{O}(N^4)$ etc.

```
bool ContainsDuplicates(IList<string> elements)
{
    for (var outer = 0; outer < elements.Count; outer++)
    {
        for (var inner = 0; inner < elements.Count; inner++)
        {
            // Don't compare with self
            if (outer == inner) continue;

            if (elements[outer] == elements[inner]) return true;
        }
    }

    return false;
}
```

Makes sense right? How about some Binary Powers:

2.4 $\mathcal{O}(2^N)$

$\mathcal{O}(2^N)$ denotes an algorithm whose growth doubles with each addition to the input data set. The growth curve of an $\mathcal{O}(2^N)$ function is exponential - starting off very shallow, then rising meteorically. An example of an O(2N) function is the recursive calculation of Fibonacci numbers:

```
int Fibonacci(int number)
{
    if (number <= 1) return number;

    return Fibonacci(number - 2) + Fibonacci(number - 1);
}
```

Wait for the finale:

2.5 Logarithms

Logarithms are slightly trickier to explain so I'll use a common example:

Binary search is a technique used to search sorted data sets. It works by selecting the middle element of the data set, essentially the median, and compares it against a target value. If the values match it will return success. If the target value is higher than the value of the probe element it will take the upper half of the data set and perform the same operation against it. Likewise, if the target value is lower than the value of the probe element it will perform the operation against the lower half. It will continue to halve the data set with each iteration until the value has been found or until it can no longer split the data set.

This type of algorithm is described as $\mathcal{O}(\log \mathrm{N})$. The iterative halving of data sets described in the binary search example produces a growth curve that peaks at the beginning and slowly flattens out as the size of the data sets increase e.g. an input data set containing 10 items takes one second to complete, a data set containing 100 items takes two seconds, and a data set containing 1000 items will take three seconds. Doubling the size of the input data set has little effect on its growth as after a single iteration of the algorithm the data set will be halved and therefore on a par with an input data set half the size. This makes algorithms like binary search extremely efficient when dealing with large data sets.