Analysis of Algorithms

1 Analysis of Algorithms

What are we measuring when we analyze an algorithm?

- How fast?
- Memory usage?
- What to measure?
  - Number of:
    - Instructions (add, multiply)
    - Loops
    - Comparisons
  - Input size

Example 1. Summing an array

```
int SumArray( int A[], int nA )
{
    int sum = 0;

    for( int i = 0 ; i < nA ; i++ )
    {
        sum += A[i];
    }

    return sum;
}
```

Example 2. Summing a Two-dimensional array

```
int Sum2DArray( int A[MAX_ROWS][], int nRows, int nCols )
{
    int sum = 0;

    for( int i = 0 ; i < nRows ; i++ )
    {
        for( int j = 0 ; j < nCols ; j++ )
            sum += A[i][j];
    }

    return sum;
}
```

1.1 Goals of Algorithm Analysis

- Characterize algorithm efficiency (time or space utilization) in terms of the input size
- Implementation independent (when possible)
- Ignore implementation dependent constants
- Ignore finite number of special cases

1.2 Big Oh ($\mathcal{O}(n)$)

Mathematical definition:

Function $f(n)$ is in $\mathcal{O}(g(n))$ ("f is Big-Oh of g") when there are constants c and n_0 such that for all $n \geq n_0$

$$f(n) \leq c\, g(n_0)$$

1.3 Useful Rules

- Additive constants don't matter (so, throw them out).
- Multiplicative constants don't matter (so, throw them out).
- Only dominant terms (in sums) matter (throw out the rest)
  Formally: $\mathcal{O}(f(n) + g(n)) = \mathcal{O}(max(f(n), g(n)))$.

  So, if $T(n)$ is in $\mathcal{O}(f(n)+g(n))$ then $T(n)$ is in $\mathcal{O}(max(f(n), g(n)))$, where $max(f, g)$ is f if there is some n_0 such that for all $n \geq n_0$, $f(n) > g(n)$ and g otherwise.

Example 3. 4N + 5

An algorithm takes 4N + 5

What is $\mathcal{O}(N)$?

Example 4. $7N + 2\log N + 2N^2$

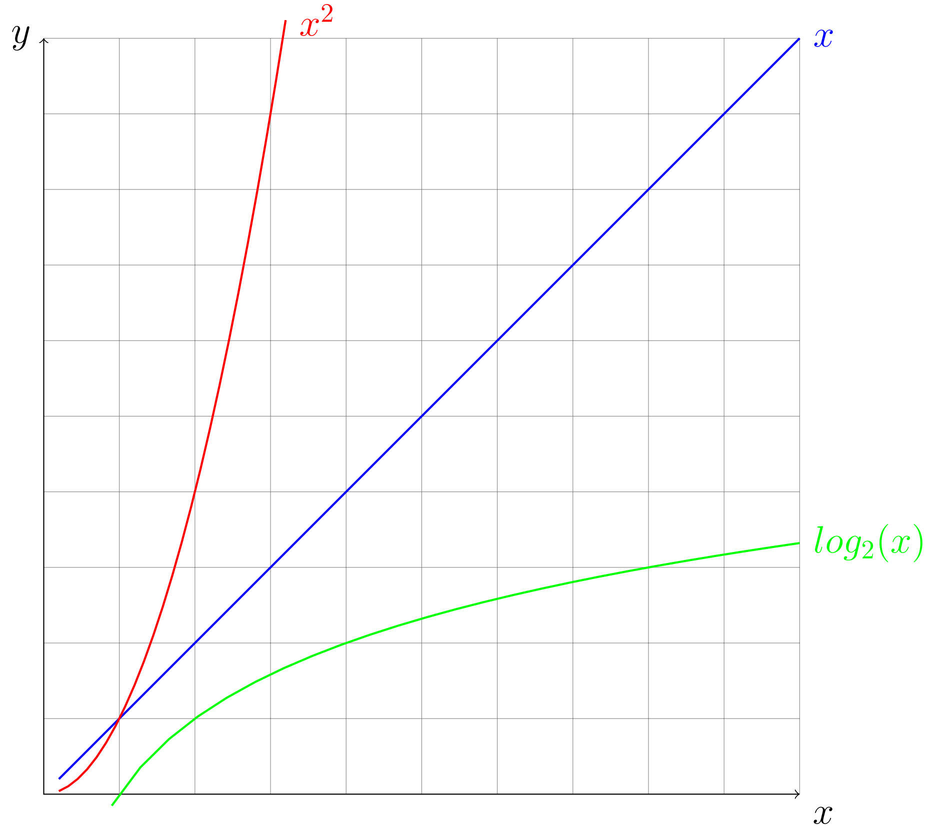
An algorithm takes $7N + 2\log N + 2N^2$

What is $\mathcal{O}(N)$?

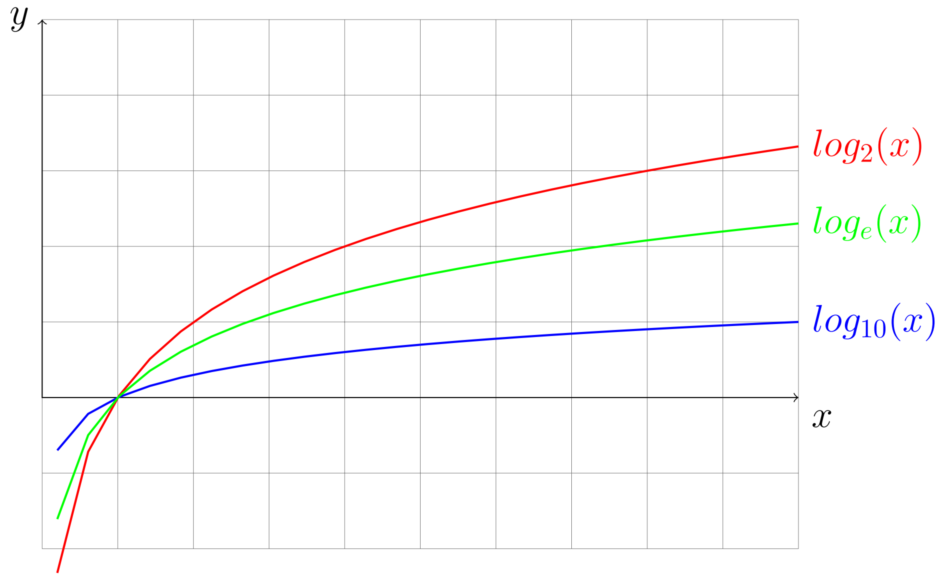
1.4 Common Growth Rate Comparison

$\mathcal{O}(1)$	constant
$\mathcal{O}(\log N)$	logarithmic
$\mathcal{O}(N)$	linear
$\mathcal{O}(N \log N)$	linear-logarithmic
$\mathcal{O}(N^2)$	quadratic
$\mathcal{O}(N^3)$	cubic
$\mathcal{O}(2^N)$	exponential

Comparison of Growth Rate Curves



Comparison of Logarithmic Curves



N	$log_2 N$	N	N^2	N^3	2^N
4	2	4	16	64	16
8	3	8	64	512	256
16	4	16	256	4096	10^4
32	5	32	1024	10^4	10^9
64	6	64	4096	10^5	10^{19}
128	7	128	10^4	10^6	10^{38}

Table 1: Comparison Of Common Growth Rates

Note: Magnitude growth for 2^N! How long is 10^{38} seconds, milliseconds, microseconds, nanoseconds?

How long is 10^{38} seconds?

Google search: `10^38 seconds in years`

```
(10^38) seconds = 3.16887646 10^30 years ?
365 days = 0.999337 years
```

1.5 Brief Summary

Most algorithms perform differently for different inputs. This leads to several types of analysis:

- Worst case
- Best case
- Average case: performance averaged over all inputs of same size.

We will see examples of these in the near future.