CSD101: Introduction to computing and programming (ICP)

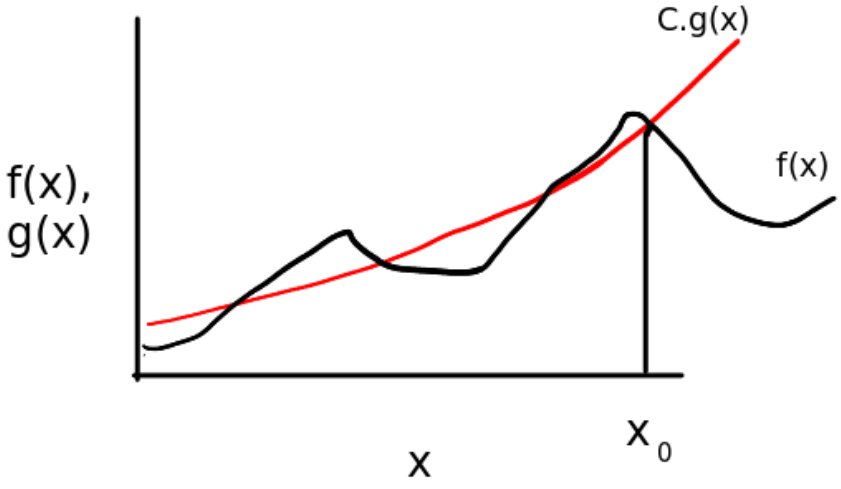
# Time complexity of algorithms

- We have seen different ways to sort, for example: bubble sort, insertion sort, selection sort, quick sort, merge sort.
- Which is the fastest? Or more generally, how do we estimate the time complexity of algorithms?
- Some observations:
  - The exact time taken clearly depends on the length of the input sequence. More generally length of the input.
  - It also depends on the state of the initial unsorted array.
- To be able to compare the time complexity of algorithms we estimate the time complexity in the **worst case**.
- Also, we measure complexity in terms of the **order** of the fastest growing term and ignore the slower growing and constant terms/factors.
- This way of measuring time complexity is called **big-O** complexity. We will symbolize it by $\mathcal{O}()$. It is called the Landau notation in mathematics.

# Definition of $\mathcal{O}()$ complexity

**Definition 5**

*Let $f(x)$, $g(x)$ be functions defined on some subset of the real line. Then function $f(x) = \mathcal{O}((g(x))$ if and only if there exist constants $C$ and $x_0$ such that for all $x > x_0$, $|f(x)| \leq C|g(x)|$.*
*That is beyond $x_0$, $Cg(x)$ always dominates $f(x)$. In CS $f(x), g(x) > 0$.*
`Note that '=' symbol has a different meaning here.`




This is for large values of $x$. That is $x \to \infty$. Called asymptotic complexity.

# Commonly used $\mathcal{O}()$ functions

| **Function** | **Name** |
|---|---|
| $\mathcal{O}(1)$ | Constant |
| $\mathcal{O}(\log(n))$ | Logarithmic |
| $\mathcal{O}((log(n))^c)$ | Poly-logarithmic |
| $\mathcal{O}(n)$ | Linear |
| $\mathcal{O}(n^2)$ | Quadratic |
| $\mathcal{O}(n^c)$ | Polynomial |
| $\mathcal{O}(c^n)$ | Exponential |

c is a constant. The above are in increasing order.

# Actual values for function growth

| $n$ | $log_{10}(n)$ | $n^2$ | $n^4$ | $2^n$ |
|---|---|---|---|---|
| 10 | 1 | $10^2$ | $10^4$ | $\sim 10^3$ |
| $10^2$ | 2 | $10^4$ | $10^8$ | $\sim 10^{30}$ |
| $10^4$ | 4 | $10^8$ | $10^{16}$ | $\sim 10^{3000}$ |

Already at $n = 100$, $2^n$ is intractable. So, an algorithm with exponential time complexity is intractable for even relatively small values of $n$. Higher order polynomials also become intractable for moderate values of $n$.

# Examples

- Expression evaluation will typically take constant time (if it does not involve a function call).
- Finding the maximum/minimum element in a sorted sequence of size $n$ can be done in constant or $\mathcal{O}(1)$ time. This is a better example for a constant time algorithm.
- Searching whether an element exists in a sorted sequence of size $n$ can be done in $\mathcal{O}(log_2(n))$ time.
- Finding the maximum or minimum element in a sequence of size $n$ can be done in $\mathcal{O}(\mathrm{n})$ time.
- Bubblesort, selection sort, quicksort take $\mathcal{O}(n^2)$ time.
- Calculating the matrix product $C = A \times B$ where A, B, C are $n \times n$ matrices using the standard formula takes $\mathcal{O}(n^3)$ time.
- Evaluating the nth fibonacci number using the recursive definition will take $\mathcal{O}(2^n)$ time.