A verbose definition

A non negative function $T(n)$ is said to be O(f(n)) provided there exists a constant $c > 0$ and an integer $n_0 > 0$ such that $T(n) \leq c.f(n)$ for all integer $n > n_0$.

- Consider $\frac{1}{3}n^2 - 5n$. The dominating term is n^2.
- Therefore it should be of $\mathrm{O}(n^2)$.
- Given a positive constant c, a positive integer n_0 to be found such that
$$\frac{1}{3}n^2 - 5n \leq cn^2.$$
- Dividing the inquality throughout by n^2,
$$\frac{1}{3} - \frac{5}{n} \leq c.$$
- Therfore, if $n_0 \geq 1$, choosing $c \geq \frac{1}{3}$, the inequality will never be violated.
- Hence, the expression is indeed of $\mathrm{O}(n^2)$.

Myths and facts

- Big-Oh is not a function.
- It is wrong to say $T_1(n) = T_2(n)$ even if both are $\mathrm{O}(f(n))$. For example, if $T_1(n) = n^2$ and $T_2(n) = n^3$, then both are of $\mathrm{O}(n^3)$ whereas clearly, $T_1(n) \neq T_2(n)$.
- The problem is we implicitly assume here that if $g(n) = O(f(n))$ then $f(n) = O^{-1}(g(n))$.

More examples

- The sum of first n squares$= \frac{1}{6}n(2n+1)(n+1)$.
- Rewrite it as
$$\frac{1}{3}n(n+\frac{1}{2})(n+1) = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n.$$
- Applying following simplifications
$$\begin{aligned} |\tfrac{1}{3}n^3 + \tfrac{1}{2}n^2 + \tfrac{1}{6}n| &\le |\tfrac{1}{3}n^3| + |\tfrac{1}{2}n^2| + |\tfrac{1}{6}n| \\ &\le |\tfrac{1}{3}n^3| + |\tfrac{1}{2}n^3| + |\tfrac{1}{6}n^3| \\ &\le |n^3| \end{aligned}$$
- Therefore, the sum of first n square is of $\mathrm{O}(n^3)$

More examples

```
class bubbleSort{
   int [] a;
   // Constructor
   public bubbleSort(int n) {
      a = new int[10];
      for(int i=0;i<n;i++)
         a[i]=(int)(Math.random() * 100);
   }
   // Main sorting procedure
   public void sorting(){
      for (int i=a.length; i>1; i--)
         for (int j=0; j<i-1; j++)
            if (a[j] > a[j+1])
            // Exchange a[j] and a[j+1]
               swap(j, j+1);
   }
}
```

Analysis

- The function of sorting is restricted to `sorting`.
- It consists of a pair of `for` loops.
- The outer one is executed exactly n times.
- The inner `for` loop is executed i times, where i varies from n to 1 with the execution of the exterior `for`.

Analysis contd.

- The total number of times `if` statement gets executed $= \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- The worst case scenario requiring exactly $\frac{n(n+1)}{2}$ swaps is reproducible taking a reverse sorted sequence as input.
- Let `if` to require one time unit for execution.
- The time for bubble sort is $\frac{n(n+1)}{2}$.
- Therefore, bubble sort is of $O(n^2)$ time complexity.

Big-Oh is not symmetric

- $T(n) = O(f(n)) \not\Rightarrow f(n) = O(T(n))$.
- Consider $T(n) = \frac{2}{3}n^2 - \frac{5}{3}n$.
- Clearly, $T(n) = O(n^2)$.
- But $\mathrm{O}(\frac{2}{3}n^2 - \frac{5}{3}n)$ can not be an upper bound for n^2 as we are concerned with positive valued functions.

Big-Oh is transitive

- $T(n) = O(g(n))$, $g(n) = O(f(n)) \Rightarrow T(n) = O(f(n))$
- Suppose $T(n) = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$. So, $T(n) = O(\frac{1}{3}n^3)$, which can be proved as follow.

$$\frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n \leq c.\frac{1}{3}n^3 \Rightarrow \frac{1}{3} + \frac{1}{2n} + \frac{1}{6n^2} \leq \frac{c}{3}$$

- Inequality is valid for case $c \geq 3$ and integers $n \geq 1$.
- We can prove $\frac{1}{3}n^3 = O(n^3)$ in similar manner.

Big-Oh is reflexive

- $f(n) = O(f(n))$
- The proofs for upper bound could be some times very tricky.
- Set of rules for manipulation are:

$$\begin{aligned}
n^k &= O(n^{k'}), \text{ where } k \leq k' \\
c.O(f(n)) &= O(f(n)) \\
O(f(n)) + O(g(n)) &= O(|f(n)| + |g(n)|) \\
O(O(f(n))) &= O(f(n)) \\
O(f(n))O(g(n)) &= O(f(n)g(n)) \\
O(f(n)g(n)) &= f(n)O(g(n))
\end{aligned}$$

Worst/best/average cases

- *Worst case* is the longest running time for any input of size n.
  - O-notation represents upper bound $\Rightarrow$ an upper bound for worst case.
  - Can be reproducible.
  - Search for element not present.
- *Best case* is the input data set that results in best possible performance. You can not do better. This is lower bound. Also reproducible. Insertion sort on a sorted sample.
- *Average case* is the average performance. No clue as to what can be an average input. So assume all inputs equally likely.

Growth Functions

- $T(n) = O(1)$: Constant growth function.
- $T(n) = O(\lg n)$: Logarithmic growth function. The base of logarithm is not important. Binary search is an example.
- $T(n) = O(n)$: Called linear growth function.
- $T(n) = O(n \lg n)$: This is called $n \lg n$ growth function. For most practical purpose these algorithms behave like linear algorithm, Heap sort and merge sort are examples.
- $T(n) = O(n^k)$: This is called polynomial growth function.
- $T(n) = O(2^n)$: This is called exponential growth function.