1. Class (Growth) of Functions

Class or growth of functions is a set of functions that have similar growth behavior. The growth behavior of a function is determined by the highest order term in the function. The highest order term is the term with the highest exponent. For example, in the function $f(n) = 3n^2 + 2n + 1$, the highest order term is $3n^2$. The highest order term is also called the dominant term. The dominant term determines the growth behavior of the function. The dominant term is also called the asymptotic term.

Class	Example Function	Dominant Term	Example Algorithm
Constant	$f(n) = 5$	5	Adding two numbers
Logarithmic	$f(n) = \log_2 n$	$\log_2 n$	Binary search
Linear	$f(n) = 3n + 2$	$3n$	Finding the maximum element in an array
Quadratic	$f(n) = 3n^2 + 2n + 1$	$3n^2$	Insertion sort
Polynomial	$f(n) = 3n^3 + 2n^2 + 1$	$3n^3$	Matrix multiplication
Exponential	$f(n) = 3^n + 2n + 1$	3^n	Towers of Hanoi

Table 1: Examples of functions and their dominant terms

Ordered by increasing growth behavior, the classes of functions are:

$$O(1) < O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$

2. Asymptotic Notations (Big-O, Big-Ω, Big-Θ)

Asymptotic notations are used to describe the running time of an algorithm. They are used to describe the running time in terms of input size. Asymptotic notations are used to describe the growth behavior of a function. The asymptotic notations are:

- **Big-O notation**: Maximum number of steps taken by the algorithm to solve a problem of size n. $f(n) = O(g(n))$ means that $f(n)$ is bounded above by $g(n)$.
- **Big-Ω notation**: Minimum number of steps taken by the algorithm to solve a problem of size n. $f(n) = \Omega(g(n))$ means that $f(n)$ is bounded below by $g(n)$.
- **Big-Θ notation**: Average number of steps taken by the algorithm to solve a problem of size n. $f(n) = \Theta(g(n))$ means that $f(n)$ is bounded both above and below by $g(n)$.

3. The Big-O Notation

The Big-O notation is used to describe the worst-case running time of an algorithm.

$O(g(n)) = \{f(n) : 0 \leq f(n) \leq \text{cg}(n)\ \exists c > 0, n_0 > 0\ \forall n \geq n_0\}$

$O(g(n))$ is the set of all functions $f(n)$ where there exists a constant $c > 0$ and a constant $n_0 > 0$ such that $0 \leq f(n) \leq \text{cg}(n)$ for all $n \geq n_0$.

Examples:

Let $f(n) = 2n + 3$, then we can write:

$$2n + 3 \leq 5 \cdot n \rightarrow f(n) = O(n) : c = 5 \land n_0 = 1$$
$$2n + 3 \leq 5 \cdot n^2 \rightarrow f(n) = O(n^2) : c = 5 \land n_0 = 1$$
$$2n + 3 \leq 5 \cdot 2^n \rightarrow f(n) = O(2^n) : c = 5 \land n_0 = 1$$

But only $f(n) = O(n)$ is useful.

- $f(n) = n^2 = O(n^2)$ where $c = 1$ and $n_0 = 1$
- $f(n) = 2n^2 + 3n + 1 = O(n^2)$ where $c = 3$ and $n_0 = 1$
- $f(n) = 2^n = O(2^n)$ where $c = 1$ and $n_0 = 1$
- $f(n) = \frac{n^2}{\log n} = O(n^2)$ where $c = 1$ and $n_0 = 1$

Note: Since changing base of logarithm only changes the value by a constant factor, the base of the logarithm is usually not specified. For example, $O(\log_2 n)$ is written as $O(\log n)$.

4. The Big-Ω Notation

The Big-Ω notation is used to describe the best-case running time of an algorithm.

$\Omega(g(n)) = \{f(n) : 0 \leq \text{cg}(n) \leq f(n)\ \exists c > 0, n_0 > 0\ \forall n \geq n_0\}$

$\Omega(g(n))$ is the set of all functions $f(n)$ where there exists a constant $c > 0$ and a constant $n_0 > 0$ such that $0 \leq \text{cg}(n) \leq f(n)$ for all $n \geq n_0$.

Examples:

Let $f(n) = 2n + 3$, then we can write:

$$5 \cdot n \leq 2n + 3 \rightarrow f(n) = \Omega(n) : c = 5 \land n_0 = 1$$
$$5 \cdot n^2 \leq 2n + 3 \rightarrow f(n) = \Omega(n^2) : c = 5 \land n_0 = 1$$
$$5 \cdot 2^n \leq 2n + 3 \rightarrow f(n) = \Omega(2^n) : c = 5 \land n_0 = 1$$

But only $f(n) = \Omega(n)$ is useful.

- $f(n) = n^2 = \Omega(n^2)$ where $c = 1$ and $n_0 = 1$
- $f(n) = 2n^2 + 3n + 1 = \Omega(n^2)$ where $c = 2$ and $n_0 = 1$
- $f(n) = 2^n = \Omega(2^n)$ where $c = 1$ and $n_0 = 1$
- $f(n) = \frac{n^2}{\log n} = \Omega(n^2)$ where $c = 1$ and $n_0 = 1$

5. The Big-Θ Notation

The Big-Θ notation is used to describe the average-case running time of an algorithm.

$\Theta(g(n)) = \{f(n) : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\ \exists c_1 > 0, c_2 > 0, n_0 > 0\ \forall n \geq n_0\}$

$\Theta(g(n))$ is the set of all functions $f(n)$ where there exists a constant $c_1 > 0$, a constant $c_2 > 0$ and a constant $n_0 > 0$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

Examples:

Let $f(n) = 2n + 3$, then we can write:

$$5 \cdot n \leq 2n + 3 \leq 5 \cdot n \rightarrow f(n) = \Theta(n) : c_1 = 5, c_2 = 5 \wedge n_0 = 1$$
$$5 \cdot n^2 \leq 2n + 3 \leq 5 \cdot n^2 \rightarrow f(n) = \Theta(n^2) : c_1 = 5, c_2 = 5 \wedge n_0 = 1$$
$$5 \cdot 2^n \leq 2n + 3 \leq 5 \cdot 2^n \rightarrow f(n) = \Theta(2^n) : c_1 = 5, c_2 = 5 \wedge n_0 = 1$$

But only $f(n) = \Theta(n)$ is useful.

- $f(n) = n^2 = \Theta(n^2)$
- $f(n) = 2n^2 + 3n + 1 = \Theta(n^2)$
- $f(n) = 2^n = \Theta(2^n)$
- $f(n) = \frac{n^2}{\log n} = \Theta(n^2)$

6. Practice Problems

Q 6.1 For the function defined by $f(n) = 2n^2 + 3n + 1$, proove/disprove the following statements:

i. $f(n) = O(n^2)$
ii. $f(n) = O(n^3)$
iii. $n^2 = O(f(n))$
iv. $f(n) \neq O(n)$
v. $n^3 \neq O(f(n))$

Q 6.2 For the function defined by $f(n) = 2n^3 + 3n^2 + 1$ and $g(n) = 2n^2 + 3$, proove/disprove the following statements:

i. $f(n) = \Omega(g(n))$
ii. $g(n) \neq \Omega(f(n))$
iii. $n^3 = \Omega(g(n))$
iv. $f(n) \neq \Omega(n^4)$
v. $n^2 \neq \Omega(f(n))$

Q 6.3 For the function defined by $f(n) = 2n^3 + 3n^2 + 1$ and $g(n) = 2n^3 + 1$, show that:

i. $f(n) = \Theta(g(n))$
ii. $f(n) \neq \Theta(n^2)$
iii. $n^4 \neq \Theta(g(n))$