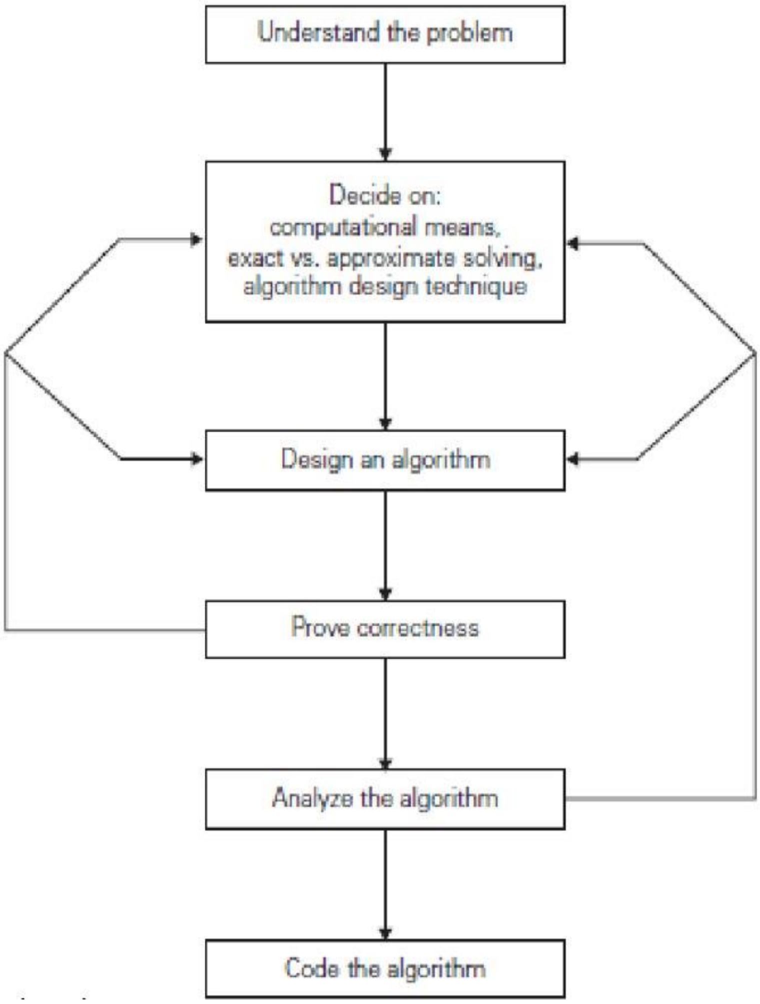
DAA Unit 1 Important

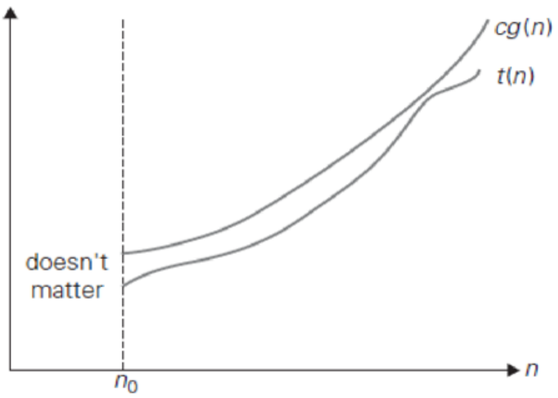
1. Algorithm design and analysis process



2. Big Oh, Big omega and Big theta formal definitions with example each

Big Oh Notation(O)

- A function t(n) is said to be O(g(n)), denoted $t(n) \in O(g(n))$, if t(n) is bounded above by some constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some non negative integer n_0 Such that $t(n) \le c.g(n)$ for all $n \ge n_0$



Example:

Let t(n)=100n+5. Express t(n) using Big-Oh(O)

$t(n) \le c.g(n)$ for all $n \ge n_0$

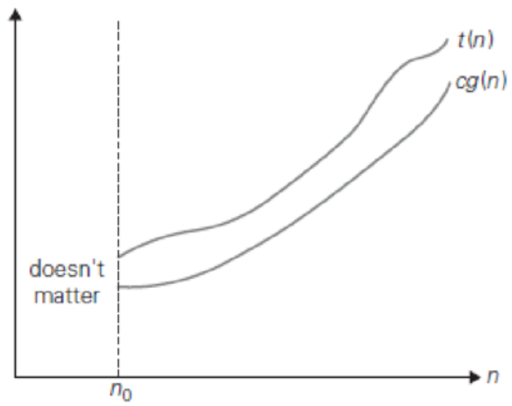
$100n+5 \le 101n$ for $n \ge 5$

$c=101$, $g(n)=n$, $n_0=5$

$\therefore t(n) \in O(g(n))$

Big Omega Notation(Ω)

- A function t(n) is said to be in Ω(g(n)), denoted t(n) ∈Ω(g(n)), if t(n) is bounded below by some constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some non negative integer n_0 Such that $t(n) \geq c.g(n)$ for all $n \geq n_0$



Let $t(n)=10n^3+5$. Express t(n) using Omega(Ω)

$t(n) \geq c.g(n)$ for all $n \geq n_0$

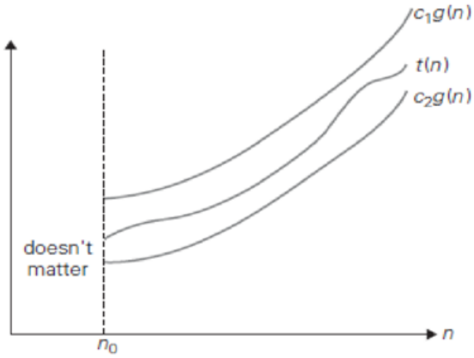
$10n^3+5 \leq 10n^3$ for $n \geq 0$

$c=10$, $g(n)=n^3$, $n_0 = 0$

$\therefore t(n) \in \Omega(g(n^3))$

Big Theta Notation(θ)

- A function t(n) is said to be in θ(g(n)), denoted t(n) ∈ θ(g(n)), if t(n) is bounded both above and below by some constant multiple of g(n) for all large n, i.e., if there exist some positive constant c_1 and c_2 and some non negative integer n_0 Such that $c_2 .g(n) \le t(n) \le c_1.g(n)$ for all $n \ge n_0$



For example, let us prove that $\frac{1}{2}n(n-1) \in \Theta(n^2)$. First, we prove the right inequality (the upper bound):

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \le \frac{1}{2}n^2 \quad \text{for all } n \ge 0.$$

Second, we prove the left inequality (the lower bound):

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \ge \frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n \text{ (for all } n \ge 2) = \frac{1}{4}n^2.$$

Hence, we can select $c_2 = \frac{1}{4}$, $c_1 = \frac{1}{2}$, and $n_0 = 2$.

3. General Plan for Analyzing the Time Efficiency of Nonrecursive Algorithms

- Decide on a parameter (or parameters) indicating an input's size.

- Identify the algorithm’s basic operation. (As a rule, it is located in the innermost loop.)
- Check whether the number of times the basic operation is executed depends only on the size of an input. If it also depends on some additional property, the worst-case, average-case, and, if necessary, best-case efficiencies have to be investigated separately.
- Set up a sum expressing the number of times the algorithm’s basic operation is executed
- Using standard formulas and rules of sum manipulation, either find a closed form formula for the count or, at the very least, establish its order of growth.

4. General Plan for Analyzing the Time Efficiency of Recursive Algorithms

- Decide on a parameter (or parameters) indicating an input’s size
- Identify the algorithm’s basic operation
- Check whether the number of times the basic operation is executed can vary on different inputs of the same size; if it can, the worst-case, average-case, and best-case efficiencies must be investigated separately.
- Set up a recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed

- Solve the recurrence or, at least, ascertain the order of growth of its solution