THEORY OF COMPUTATION

Subash Manandhar

Computational Complexity

- Computational complexity theory is a branch of theory of computation in computer science that focuses on classifying computational problems according to their inherent difficulty and relating those classes to each other.
- It involves classifying problems according to their inherent tractability and intractability that is whether they are easy or hard to solve.
- It deals with resources required during computation to solve a given problem.
  - Time complexity (how many steps it takes to solve a problem)
  - Space complexity (how much memory it takes)

Computational Complexity

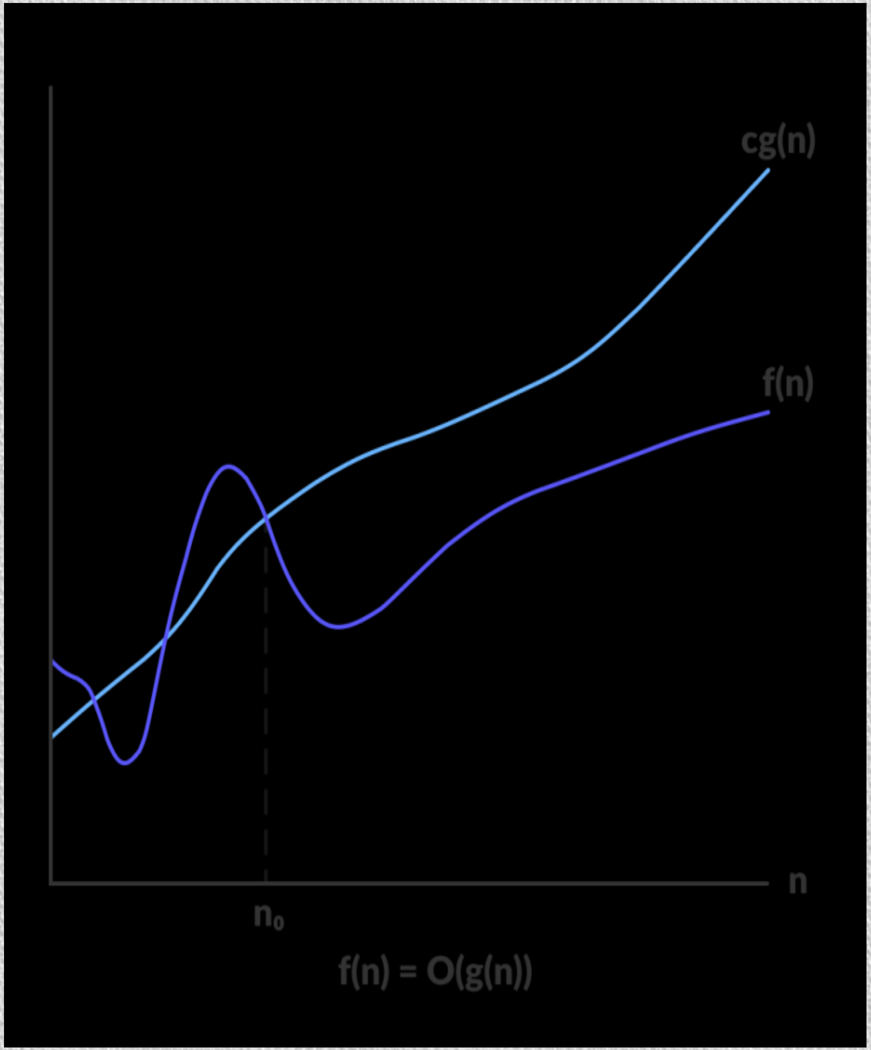
- The complexity of computational problems can be discussed by choosing a specific abstract machine as a model of computation and considering how much resource machine of that type require for the solution of that problem.
- Complexity Measure is a means of measuring the resource used during a computation.
- In case of Turing Machines, during any computation, various resources will be used, such as space and time.
- When a Turing machine answers a specific instance of a decision problem we can measure time as number of moves and the space as number of tape squares, required by the computation.
- The most obvious measure of the size of any instance is the length of input string.
- The worst case is considered as the maximum time or space that might be required by any string of that length.

Computational Complexity

- **Asymptotic Notation:**
  - Complexity analysis of an algorithm is very hard if we try to analyze exact.
  - we know that the complexity (worst, best, or average) of an algorithm is the mathematical function of the size of the input.
  - So if we analyze the algorithm in terms of bound (upper and lower) then it would be easier.
  - For this purpose we need the concept of asymptotic notations.
  - The study of change in performance of the algorithm with the change in the order of the input size is defined as asymptotic analysis.
  - Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.
    - Big Oh (O) notation
    - Big Omega (Ω) notation
    - Big Theta (θ) notation

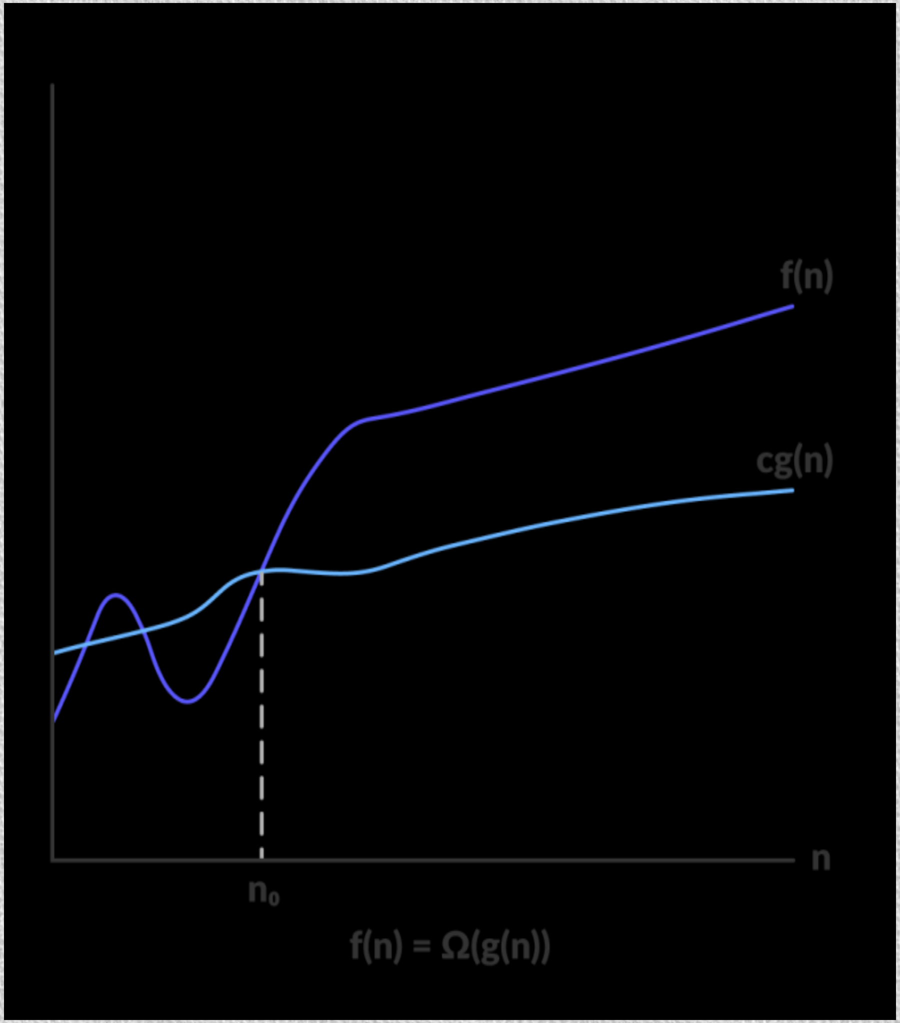
Computational Complexity

- **Asymptotic Notation:**
  - **Big Oh (O) notation**
    - Big-O notation represents the upper bound of the running time of an algorithm.
    - Thus, it gives the worst-case complexity of an algorithm.
    - A function f(x)=O(g(x)) (read as f(x) is big oh of g(x)) iff there exists two positive constants c and x_0 such that for all x >= x_0, 0 <= f(x) <= c*g(x)



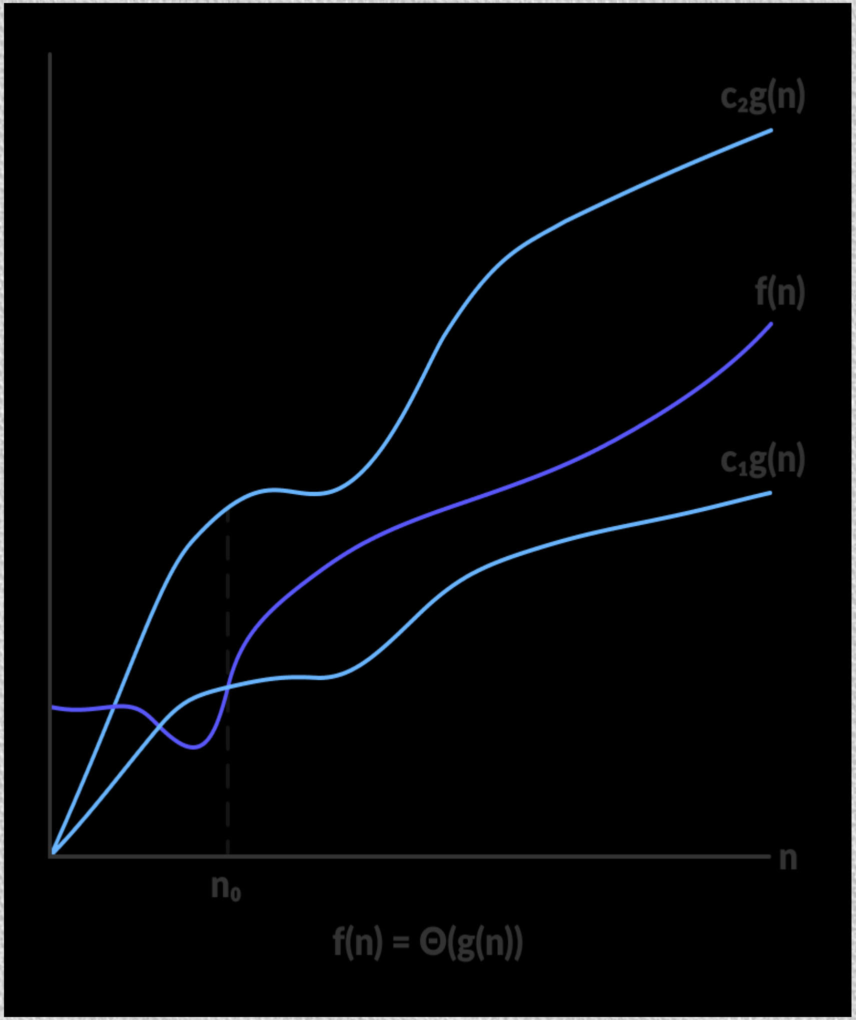
Computational Complexity

- **Asymptotic Notation:**
  - **Big Omega (Ω) notation**
    - Big Omega (Ω) notation represents the lower bound of the running time of an algorithm.
    - Thus, it provides the best case complexity of an algorithm.
    - A function f(x) =Ω (g(x)) (read as g(x) is big omega of g(x)) iff there exists two positive constants c and x_0 such that for all $x \geq x_0$, $0 \leq c*g(x) \leq f(x)$.



Computational Complexity

- **Asymptotic Notation:**
  - Big Theta (θ) notation
    - Big Theta (θ) notation represents the upper and the lower bound of the running time of an algorithm.
    - Thus, it provides the average case complexity of an algorithm.
    - A function f(x) = (g(x)) (read as f(x) is big theta of g(x)) iff there exists three positive constants c1, c2 and x0 such that for all x >= x0, 0 <= c1*g(x) <= f(x) <= c2*g(x)



Computational Complexity

- The computer can solve: some problems in limited time e.g. sorting, some problems requires unmanageable amount of time e.g. Hamiltonian cycles, and some problems cannot be solved e.g. Halting Problem.
- The problems that can be solved using polynomial time algorithms are called **tractable problems**.
- The problems that cannot be solved in polynomial time but requires super-polynomial time algorithm are called **intractable or hard problems.**
- There are many problems for which no algorithm with running time better than exponential time is known some of them are, traveling salesman problem, Hamiltonian cycles, and circuit satisfiability, etc.

Computational Complexity

- **Complexity Classes:**
  - In computational complexity theory, a **complexity class** is a set of problems of related resource-based complexity.
  - A typical complexity class has a definition of the form:
    - "The set of problems that can be solved by an abstract machine M using O(f(*n*)) of resource R, where *n* is the size of the input."
  - For example, the **class NP** is the set of decision problems that can be solved by a non-deterministic Turing machine in polynomial time, while the **class P** is the set of decision problems that can be solved by a deterministic Turing machine in polynomial space.
  - The set of problems that can be solved using polynomial time algorithm is regarded as **class P**.
  - The problems that are verifiable in polynomial time constitute the **class NP**.
  - The class of **NP complete** problems consists of those problems that are NP as well as they are *as hard as* any problem in NP.

Computational Complexity

- **Complexity Classes:**
  - The main concern of studying NP completeness is to understand how hard the problem is.
  - So if we can find some problem as NP complete then we try to solve the problem using methods like approximation, rather than searching for the faster algorithm for solving the problem exactly.
  - **Class P:**
    - The class P is the set of problems that can be solved by deterministic TM in polynomial time.
    - A language L is in class P if there is some polynomial time complexity T(n) such that L=L(M), for some Deterministic Turing Machine M of time complexity T(n).
  - **Class NP:**
    - The class NP is the set of problems that can be solved by a non-deterministic TM in polynomial time.
    - Formally, we can say a language L is in the class NP if there is a non-deterministic TM, M, and a polynomial time complexity T(n), such that L= L(M), and when M is given an input of length n, there are no sequences of more than T(n) moves of M.

Computational Complexity

- **NP-Complete:**
  - In computational complexity theory, the complexity class **NP-complete** (abbreviated **NP-C** or **NPC**), is a class of problems having two properties:
    - It is in the set of NP (nondeterministic polynomial time) problems: Any given solution to the problem can be *verified* quickly (in polynomial time).
    - It is also in the set of NP-hard problems: Any NP problem can be converted into this one by a transformation of the inputs in polynomial time.
  - Formally: Let L be a language in NP, we say L is NP-Complete if the following statements are true about L;
    - L is in class NP
    - For every language L1 in NP, there is a polynomial time reduction of L1 to L.
  - Once we have some NP-Complete problem, we can prove a new problem to be NP-Complete by reducing some known NP-Complete problem to it using polynomial time reduction.

Computational Complexity

- **Classes of problems:**
  - **Computational Problems:**
    - Any problem which in principle can be modeled to be solved by a computer is called computational problem.
  - **Decision Problems:**
    - are computational problems for which the intended output is either yes or no.
  - **Optimization Problems:**
    - Is the problem of finding the best solution from all feasible solutions.