MIT Art Design and Technology University

MIT School of Computing, Pune

18BTCS601- Design and Analysis of Algorithms

Class - T.Y. (SEM-II), Core

Unit – I Fundamentals of Algorithms

Prof. Abhishek Das

AY 2022-2023 SEM-II

Unit I - Syllabus

MIT-ADT UNIVERSITY
PUNE, INDIA
A leap towards World Class Education

Unit – Fundamentals of Algorithms 09 hours

- Problem solving principles: Classification of problem, problem solving strategies: Brute force Approach
- Classification of time complexities (linear, logarithmic etc.), asymptotic notations, Lower bound and upper bound: Best case, worst case, average case analysis
- Amortized analysis
- Recurrences: Formulation and solving recurrence equations using Master Theorem

Unit 1 - Outline

- **Problem solving principles: Classification of problem, problem solving strategies: Brute force Approach**
- Classification of time complexities (linear, logarithmic etc.), asymptotic notations, Lower bound and upper bound: Best case, worst case, average case analysis
- Amortized analysis
- Recurrences: Formulation and solving recurrence equations using Master Theorem

INTRODUCTION TO ALGORITHMS

- **Def:** An algorithm is a procedure that takes any of the possible input instances and produces desired output.
- In other words, an algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.
- An algorithm is a finite set of instructions that accomplishes a particular task.

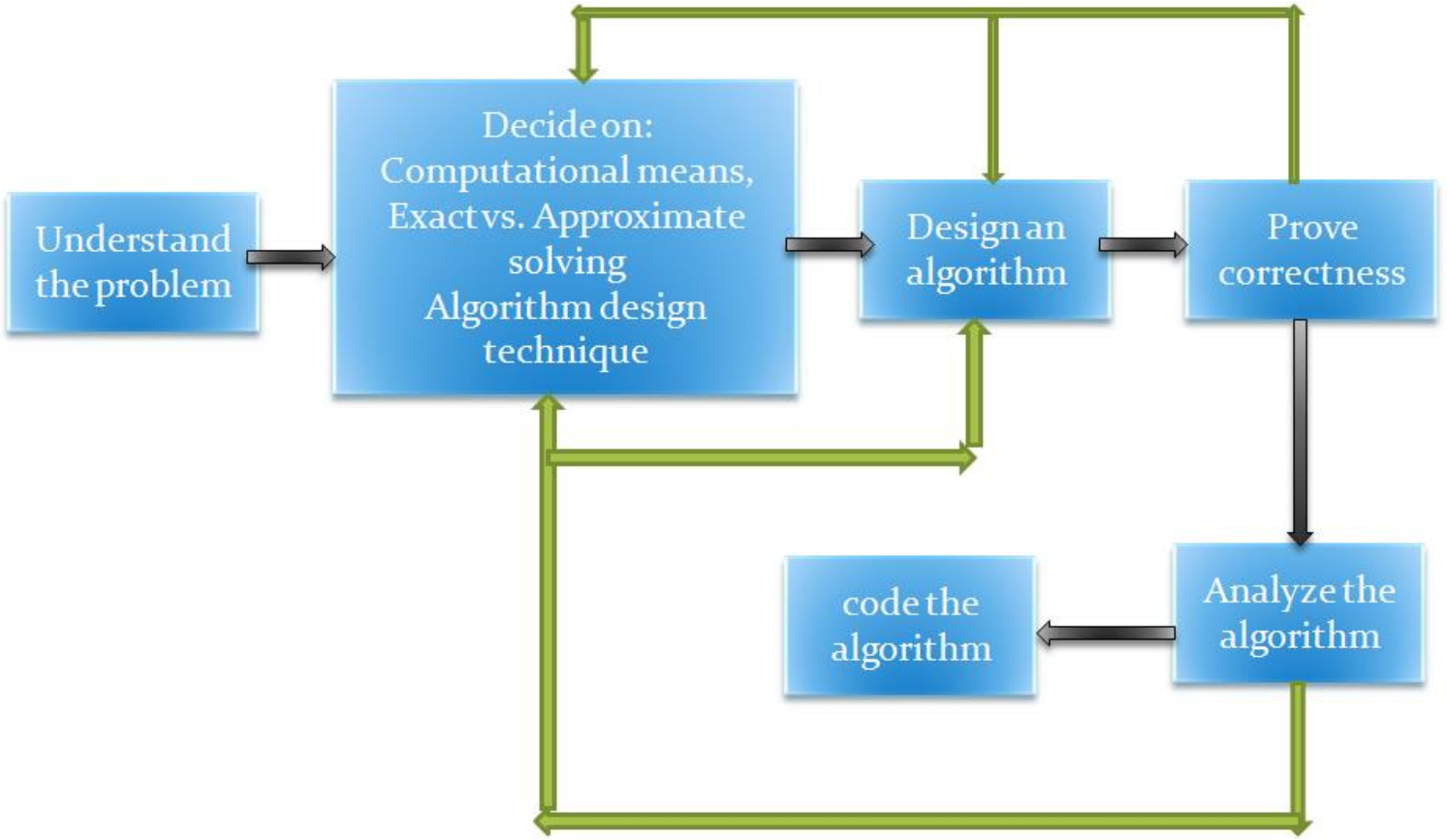
PROPERTIES OF ALGORITHMS

- Finiteness
- Correctness
- Robustness

CHARACTERISTICS OF AN ALGORITHM

- **Input:** Must take an input, i.e., zero or more quantities are externally supplied
- **Output:** Must give some output, i.e., at least one quantity is produced.
- **Definiteness:** Each instruction is clear and unambiguous
- **Finiteness:** Finite number of steps
- **Effectiveness:** Simple instructions, i.e., instruction is basic enough to be carried out

FUNDAMENTALS OF ALGORITHMIC PROBLEM SOLVING



FUNDAMENTALS OF ALGORITHMIC PROBLEM SOLVING

Steps in designing and analysing an algorithm

1. **Understanding the Problem**
2. **Ascertaining the Capabilities of the Computational Device**
3. **Choosing between Exact and Approximate Problem Solving**
4. **Deciding on appropriate Data Structures**
5. **Algorithm Design Techniques**
6. **Methods of Specifying an Algorithm**
7. **Proving an Algorithm's Correctness**
   Once an algorithm has been specified, you have to prove its ***correctness.***
8. **Analyzing an Algorithm**
   Time efficiency
   Space efficiency
9. **Coding an algorithm**

FUNDAMENTALS OF ANALYSIS OF ALGORITHMS

- **The Analysis Framework**
  - **Time efficiency**, also called time complexity, indicates how fast an algorithm in question runs.
  - **Space efficiency**, also called space complexity, refers to the amount of memory units required by the algorithm in addition to the space needed for its input and output.
- **Measuring an Input's Size**

"The choice of an appropriate size metric can be influenced by operations of the algorithm in question."

- An input to an algorithm specifies an instance of the problem to be solved

FUNDAMENTALS OF ANALYSIS OF ALGORITHMS

- **Units for Measuring Running Time**
  - ➢ We can use standard unit of time measurement—a second, or millisecond, and so on
  - ➢ Drawbacks:
    - dependence on the speed of a particular computer,
    - dependence on the quality of a program implementing the algorithm and
    - the compiler used in generating the machine code
- We need to identify most important operation of the algorithm, called the ***basic operation***
- The operation contributing ***the most to the total running time***, and compute the number of times the basic operation is executed

INTRODUCTION TO BRUTE FORCE APPROACH

- **Def:** This algorithm simply tries all possibilities until a satisfactory solution is found.
- Such an algorithm can be:
  - Optimizing: Find the best solution.
  - Satisficing: Stop as soon as a solution is found that is good enough.

BRUTE FORCE APPROACH

A Padlock

Unit 1 - Outline

- Problem solving principles: Classification of problem, problem solving strategies: Brute force Approach

- **Classification of time complexities (linear, logarithmic etc.), asymptotic notations, Lower bound and upper bound: Best case, worst case, average case analysis**

- Amortized analysis

- Recurrences: Formulation and solving recurrence equations using Master Theorem

TYPES OF TIME COMPLEXITY

- O(1)
- O(log n)
- O(n)
- O(n2)
- O(n3)
- O(2n)
- O(3n)
- ……
- $O(n^n)$

ORDERS OF GROWTH

Table: Values (some approximate) of several functions important for analysis of algorithms

TABLE 2.1 Values (some approximate) of several functions important for analysis of algorithms

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Algorithms that require an exponential number of operations are practical for solving only problems of very small sizes.

ANALYSIS OF ALGORITHMS

- We established that it is reasonable to measure an algorithm's efficiency as a function of a parameter indicating the size of the algorithm's input.
- But there are many algorithms for which running time depends not only on an input size but also on the specifics of a particular input.
- **Worst-Case, Best-Case, and Average-Case Efficiencies**

ANALYSIS OF ALGORITHMS

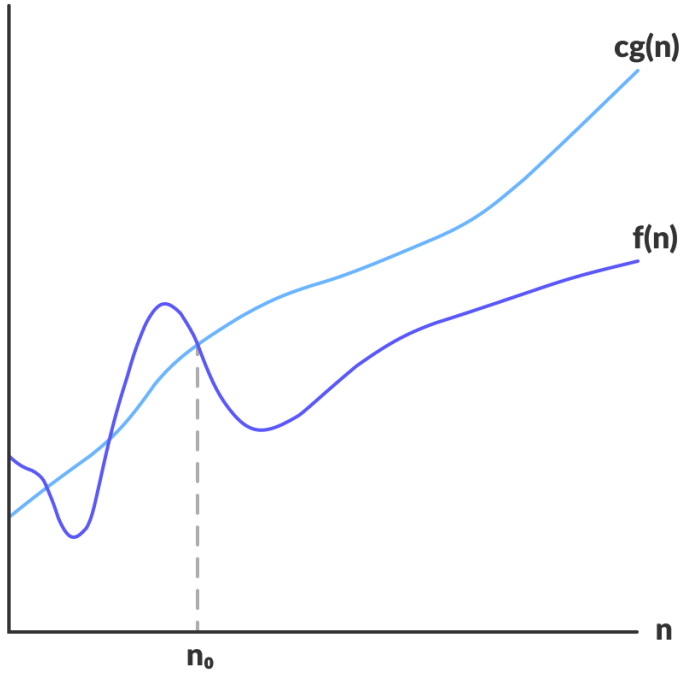
- The ***worst-case efficiency*** of an algorithm is its efficiency for the worst-case input of size *n*, which is an input (or inputs) of size *n* for which the algorithm runs the longest among all possible inputs of that size.
- The ***best-case efficiency*** of an algorithm is its efficiency for the best-case input of size *n*, which is an input (or inputs) of size *n* for which the algorithm runs the fastest among all possible inputs of that size.
- The ***average-case efficiency*** cannot be obtained by taking the average of the worst-case and the best-case efficiencies.

ASYMPTOTIC NOTATIONS

- The efficiency analysis framework concentrates on the order of growth of an **algorithm's basic operation** count as the principal indicator of the algorithm's efficiency.
- Time Complexity is expressed with the help of some notations, known as *Asymptotic notations*. These include:
  - Big-Oh Notation
  - Big Omega Notation
  - Theta Notation

ASYMPTOTIC NOTATIONS: *Big-oh (O) Notation*

f(n) = O(g(n)) if 0 <= f(n) <= a*g(n); ∀ n >= b where *a* and *b* are constants



f(n) = O(g(n))

ASYMPTOTIC NOTATIONS: *Big-oh (O) Notation*

f(n) = O(g(n)) if 0 <= f(n) <= a*g(n); ∀ n >= b where *a* and *b* are constants

Ex: Prove that log n! = O(n log n)

ASYMPTOTIC NOTATIONS: *Big-oh (O) Notation*

f(n) = O(g(n)) if 0 <= f(n) <= a*g(n); ∀ n >= b where *a* and *b* are constants

Ex: Prove that log n! = O(n log n)

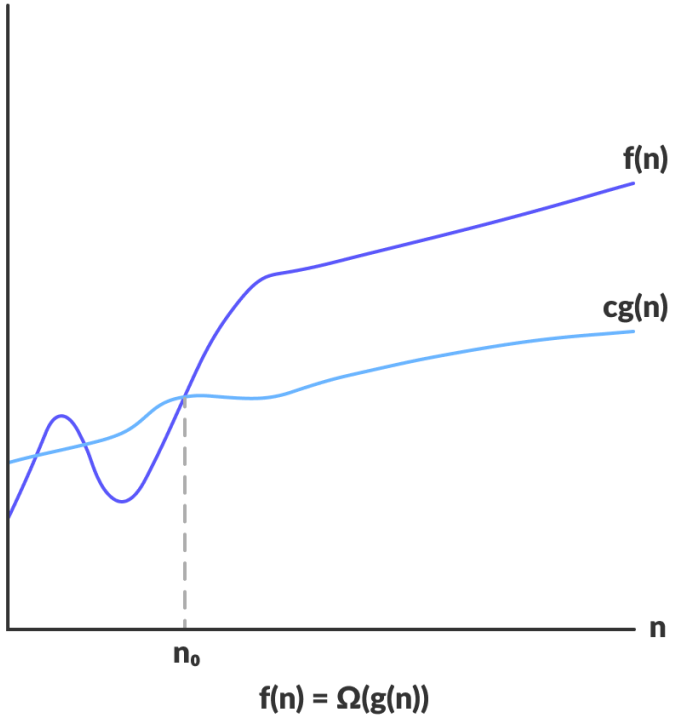
Solution: $n! = 1 \cdot 2 \cdot 3 \cdots n \leq n \cdot n \cdot n \cdots n = n^n$

$n! \leq n^n \Rightarrow \log n! \leq n \log n$

$\Rightarrow \log n! = O(n \log n)$

ASYMPTOTIC NOTATIONS: *Big-Omega (Ω) Notation*

f(n) = Ω(g(n)) if 0 <= a*g(n) <= f(n); ∀ n >= b where *a* and *b* are positive constants



f(n) = Ω(g(n))

ASYMPTOTIC NOTATIONS: *Big-Omega (Ω) Notation*

f(n) = Ω(g(n)) if 0 <= a*g(n) <= f(n); ∀ n >= b where *a* and *b* are positive constants

Ex: Prove that $7n^2 = \Omega(n)$

ASYMPTOTIC NOTATIONS: *Big-Omega (Ω) Notation*

f(n) = Ω(g(n)) if 0 <= a*g(n) <= f(n); ∀ n >= b where *a* and *b* are positive constants

Ex: Prove that $7n^2 = \Omega(n)$

Solution: $\lim_{n\to\infty} \frac{7n^2+n}{n} = \lim_{n\to\infty}(7n+1) = \infty \neq 0$

Hence, $7n^2 + n = \Omega(n)$

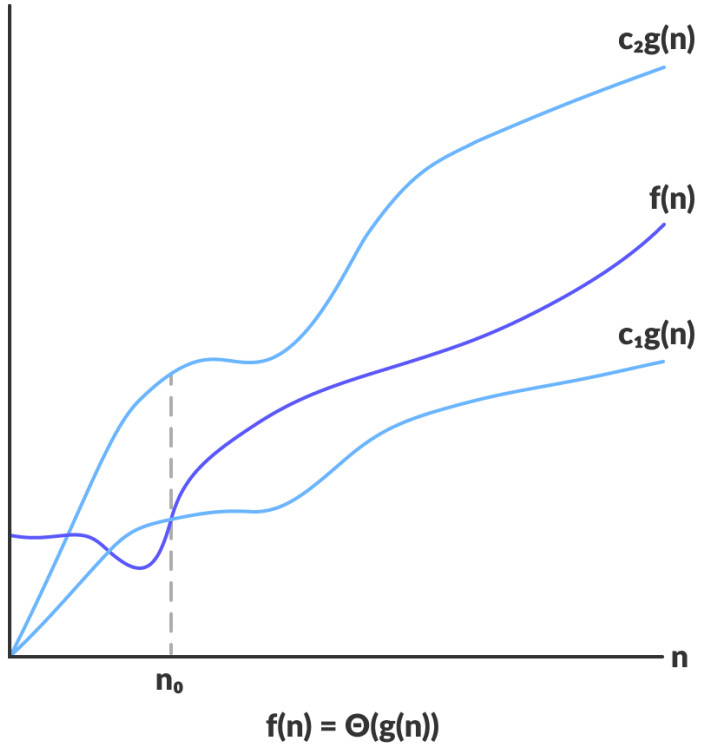
To prove also that $7n^2 + n = \Omega(n^2)$ we can write

$\lim_{n\to\infty} \frac{7n^2+n}{n^2} = 7 + \lim_{n\to\infty} \frac{1}{n} = 7 \neq 0$. Hence, $7n^2 + n = \Omega(n^2)$.

ASYMPTOTIC NOTATIONS: *Theta (Θ) Notation*

f(n) = Θ(g(n)) if

- f(n) = O(g(n))
- f(n) = Ω(g(n))



f(n) = Θ(g(n))

ASYMPTOTIC NOTATIONS: *Theta (Θ) Notation*

$f(n) = \Theta(g(n))$ if

- $f(n) = O(g(n))$
- $f(n) = \Omega(g(n))$

Ex: Prove that $t(n) = 10n^4 – 50n^3 + 200n^2 - 1000n = \Theta(n^4)$

ASYMPTOTIC NOTATIONS: *Theta (Θ) Notation*

$f(n) = \Theta(g(n))$ if

- $f(n) = O(g(n))$
- $f(n) = \Omega(g(n))$

Ex: Prove that $t(n) = 10n^4 - 50n^3 + 200n^2 - 1000n = \Theta(n^4)$

Solution: First we prove $t(n) = O(n^4)$ then prove $t(n) = \Omega(n^4)$

Now, $t(n) \le 10n^4 + 200n^2 \le n^4\left(10 + \frac{200}{n^2}\right) \le n^4\left(10 + \frac{200}{100n^2}\right)$

$\forall n \ge 100 \Rightarrow t(n) \le 10.24\, n^4 \Rightarrow t(n) = O(n^4)$

Again, $t(n) \ge 10n^4 - 50n^3 - 1000n > n^4\left[10 - \frac{50}{n} - \frac{1000}{n^3}\right]$

$\Rightarrow t(n) \ge n^4\left[10 - \frac{50}{100} - \frac{1000}{(100)^3}\right] \quad \forall n \ge 100$

$\Rightarrow t(n) > 9.5\, n^4 \Rightarrow t(n) = \Omega(n^4)$

Hence, $t(n) = O(n^4)$ and $\Omega(n^4)$. So, $t(n) = \Theta(n^4)$.

Unit 1 - Outline

- Problem solving principles: Classification of problem, problem solving strategies: Brute force Approach
- Classification of time complexities (linear, logarithmic etc.), asymptotic notations, Lower bound and upper bound: Best case, worst case, average case analysis
- **Amortized analysis**
- Recurrences: Formulation and solving recurrence equations using Master Theorem

AMORTIZED ANALYSIS OF ALGORITHMS

- *Amortized Analysis* is used for algorithms where an occasional operation is very slow, but most of the other operations are faster.
- In Amortized Analysis, we analyze a sequence of operations and guarantee a worst case average time which is lower than the worst case time of a particular expensive operation.
- For example, data structures whose operations are analyzed using Amortized Analysis are Hash Tables, Disjoint Sets and Splay Trees.

Unit 1 - Outline

- Problem solving principles: Classification of problem, problem solving strategies: Brute force Approach
- Classification of time complexities (linear, logarithmic etc.), asymptotic notations, Lower bound and upper bound: Best case, worst case, average case analysis
- Amortized analysis
- **Recurrences: Formulation and solving recurrence equations using Master Theorem**

MATHEMATICAL ANALYSIS OF RECURSIVE ALGORITHMS

- **Example 1:** Compute the factorial function *F(n) =n!* for an arbitrary non-negative integer *n*. since,

$$n! = 1 \cdot \ldots \cdot (n-1) \cdot n = (n-1)! \cdot n \quad \text{for } n \geq 1$$

- And *0!=1* by definition, we can compute F(n)=F(n-1).n with the following recursive algorithm

ALGORITHM $F(n)$

//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

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

TIME COMPLEXITY OF RECURSIVE ALGORITHMS

- Recurrence relation can be solved by different methods like:
  - Method of Substitution
  - Master's Theorem
  - Recursion Tree
  - Change of Variable

SUBSTITUTION ALGORITHMS

- The substitution method is a condensed way of proving an asymptotic bound on a recurrence by induction.
- Instead of trying to find an exact closed-form solution, we can only try to find a closed-form bound on the recurrence.
- It is an algebraic method to solve simultaneous linear equations.
- As the word says, in this method, the value of one variable from one equation is substituted in the other equation.

MASTER'S THEOREM

- The master's method is a formula for solving recurrence relations of the form:

$T(n) = aT(n/b) + f(n)$, where:

- n = size of input
- a = number of subproblems in the recursion
- n/b = size of each subproblem.
- All subproblems are assumed to have the same size.

MASTER'S THEOREM

Example 1:

$T(n) = 4T(n/2) + n$

MASTER'S THEOREM

Example 2:

$T(n) = 4T(n/2) + n^2$

MASTER'S THEOREM

Example 3:

$T(n) = 4T(n/2) + n^3$

MASTER'S THEOREM

Example 4:

$T(n) = 2T(n/2) + n$

MASTER'S THEOREM

Example 5:

$T(n) = 2T(n/2) + n^2$

MASTER'S THEOREM

Example 6:

$T(n) = 2T(n/2) + n^3$

MASTER'S THEOREM

Example 7:

$T(n) = T(n/2) + O(1)$

MASTER'S THEOREM

Example 8:

$T(n) = 4T(n/2) + n$

MASTER'S THEOREM

Example 9:

$T(n) = 4T(n/2) + n^2$

MASTER'S THEOREM

Example 10:

$T(n) = 4T(n/2) + n^3$

MASTER'S THEOREM

Example 11:

$T(n) = T(9n/10) + n$

MASTER'S THEOREM

Example 12:

$T(n) = T(9n/10) + n^2$

MASTER'S THEOREM

Example 13:

$T(n) = T(9n/10) + n^3$

MASTER'S THEOREM

Example 14:

$T(n) = 8T(n/4) + n$

MASTER'S THEOREM

Example 15:

$T(n) = 8T(n/4) + n^2$

MASTER'S THEOREM

Example 16:

$T(n) = 8T(n/4) + n^3$

MASTER'S THEOREM

Example 17:

$T(n) = 3T(2n/6) + n$

MASTER'S THEOREM

Example 18:

$T(n) = 3T(2n/6) + n^2$

MASTER'S THEOREM

Example 19:

$T(n) = 3T(2n/6) + n^3$

MASTER'S THEOREM

Example 20:

$T(n) = 16T(n/8) + n$

MASTER'S THEOREM

Example 21:

$T(n) = 16T(n/8) + n^2$

MASTER'S THEOREM

Example 22:

$T(n) = 16T(n/8) + n^3$

MASTER'S THEOREM

Example 23:

$T(n) = 16T(n/8) + n^4$