CS3205 : Design and Analysis of Algorithms

- Setting:

  Strategies and

  Analysis (Time/ Space/ Correctness)

Strategies

- Divide and Conquer (Sort)
- Greedy (MST)
- Dynamic Prog (Fibonacci)
- Backtracking (nqueens)
- Randomized
- Approximation

Analysis

- Time Complexity
- Space Complexity (Polynomial v/s exp)
- Correctness of algo (Greedy)
- How approximate solution is?

CS3205 : Design and Analysis of Algorithms

Course Prerequisites: Basic courses on programming, data structures, Discrete structures, theory of computing.

Course Objectives:

Students will gain understanding of asymptotic notations and will be able to apply suitable mathematical techniques to find asymptotic time and space complexities of algorithms.

Students will develop ability to formulate computational problems in abstract and mathematically precise manner.

Student will gain understanding of different algorithm design paradigms such as divide and conquer, dynamic programming, greedy, backtracking and will apply suitable paradigm for designing algorithms for computational problems

Students will develop understanding of notions of NP-hardness and NP-completeness and their relationship with the intractability of decision problems.

Students will design randomized, approximation algorithms for some computational problems.

CS3205 : Design and Analysis of Algorithms

Credits: 5 **Teaching Scheme Theory: 3** Hours/Week

Tut: 1 Hours/Week

Lab: 2 Hours/Week

Course Relevance: This is a foundational course for Computer science and Engineering. This course develops algorithmic thinking capability of students. Designing algorithms using suitable paradigm and analysing the algorithms for computational problems has a high relevance in all domains where computer science plays a crucial role (equally in Industry as well as research). This course is also an essential pre-requisite for advanced domain specific algorithmic courses such as Algorithmic Graph Theory, Algorithmic Number Theory, Computational Geometry, Motion planning and Robotics, etc, to give a few examples.

Once the student gains expertise in Algorithm design and in general gains ability of Algorithmic thinking, it facilitates in systematic study of any other domain (in computer science or otherwise) which demands logical thinking.

This course is also relevant for students who want to pursue research career in theory of computing, computational complexity theory, advanced algorithmic research.

CS3205 : Design and Analysis of Algorithms

SECTION-1

Basic introduction and time and space complexity analysis:

Asymptotic notations (Big Oh, small oh, Big Omega, Theta notations). Best case, average case, and worst-case time and space complexity of algorithms. Overview of searching, sorting algorithms. Adversary lower bounds (for the comparison-based sorting algorithms, for finding second minima). Using Recurrence relations and Mathematical Induction to get asymptotic bounds on time complexity. Master's theorem and applications. Proving correctness of algorithms.

Divide and Conquer: General strategy**,** Binary search and applications, Analyzing Quick sort, Merge sort, Counting Inversions, Finding a majority element, Order statistics (randomized and deterministic algorithms), Josephus problem using recurrence, Efficient algorithms for Integer arithmetic (Euclid's algorithm, Karatsuba's algorithm for integer multiplication, fast exponentiation).

Dynamic Programming: General strategy, simple dynamic programming based algorithms to compute Fibonacci numbers, binomial coefficients, Matrix Chain multiplication, Optimal binary search tree (OBST) construction, Coin change problem, 0-1 Knapsack, Traveling Salesperson Problem, All pair shortest path algorithm, Longest increasing subsequence problem, Largest independent set for trees.

CS3205 : Design and Analysis of Algorithms

SECTION-1I

Greedy and Backtracking strategy:

Greedy: General strategy, Analysis and correctness proof of minimum spanning tree and shortest path algorithms, fractional knapsack problem, Huffman coding, conflict free scheduling.

Backtracking: General strategy, n-queen problem, backtracking strategy for some NP-complete problems (e.g. graph coloring, subset sum problem, SUDOKU)

Introduction to complexity classes and NP-completeness:

Complexity classes P, NP, coNP, and their interrelation, Notion of polynomial time many one reductions reduction, Notion of NP-hardness and NP-completeness, Cook-Levin theorem and implication to P versus NP question, NP-hardness of halting problem. NP-Complete problems (some selected examples from - Satisfiability problem, Circuit-SAT, 3-CNF SAT, vertex cover problem, independent set problem, clique problem, Hamiltonian-circuit problem, subset sum problem, Integer Linear Programming.), reducing NP problems to Integer Linear Programming.

CS3205 : Design and Analysis of Algorithms

SECTION II (Continued)

Introduction to Randomized and Approximation algorithms:

Introduction to randomness in computation, Las-Vegas and Monte-Carlo algorithms, Abundance of witnesses/solutions and application of randomization, solving SAT for formulas with "many" satisfying assignments, randomized quick sort, Las-Vegas and Monte-Carlo algorithms for majority search, Karger's Min-cut algorithm, coupon collector problem, randomized data structures (randomized BST, skip lists)

Introduction to Approximation algorithms for NP-optimization problems, Approximation algorithm for Vertex Cover, metric Traveling-Sales-Person Problem (metric-TSP), Hardness of approximation for TSP.

Time Complexity

- Posterior (Depends on factors like speed of machine, network issues, parallel tasks)
- Apriori (Only the algorithm)
- What is our interest?

Time Complexity

- Apriori Analysis

  Measure of Instruction count
- Two rules for time complexity:

  Should have SINGLE 'n' a size of input on which it depends

  In a polynomial $n^2 + 5n$, the higher polynomial dominates.

Notations of TIME Complexity (Big Oh (O), Omega (Ω) , Theta(ϴ))

- Big (Oh) Notation (O) Definition :

  f(n) is of the Order of g(n)

  f(n)=O(g(n)) iff f(n)≤ c*g(n) for all $n \geq n_0$

If $f(n)=n^2 + 5n$ g(n)= ? (f(n) : instr count)

If f(n)=10000000, g(n)=? O(1) (g(n) :TC)

Big (Oh) is UPPER BOUND

Concept of TIGHT UPPER BOUND.

Notations of TIME Complexity (Big Oh (O), Omega (Ω) , Theta(ϴ))

Omega Notation (Ω) Definition :

f(n) is of the Order of g(n)

f(n)=O(g(n)) iff f(n) ≥ c*g(n) for all n≥n_0

If f(n)=n^2 + 5n   g(n)= ?

If f(n)=10000000, g(n)=?

OMEGA is LOWER BOUND

Concept of Tight LOWER Bound.

Notations of TIME Complexity (Big Oh (O), Omega (Ω) , Theta(ϴ))

Theta Notation (ϴ) Definition :

f(n) is of the Order of g(n)

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

$c_1*g(n) \leq f(n) \leq c_2*g(n)$ for all $n \geq n_0$

If $f(n)=n^2 + 5n$ g(n)= ?

If f(n)=10000000, g(n)=? O(1)

Theta is EXACT BOUND

Introductory Problem: Stable Matching

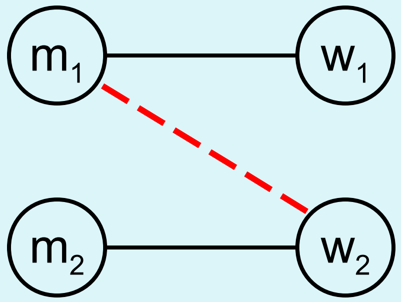
- Setting:
  - Assign TAs to Instructors
  - Avoid having TAs and Instructors wanting changes
    - E.g., Prof A. would rather have student X than her current TA, and student X would rather work for Prof A. than his current instructor.

STABLE MARRIAGE PROBLEM

SET A	SET B
A (2,4,1,3)	1 (C,B,A,D)
B (4,2,1,3)	2 (D,A,C,B)
C (4,3,2,1)	3 (C,D,B,A)
D(1,3,4,2)	4 (B,A,D,C)

Formal notions

- Perfect matching
- Ranked preference lists
- Stability



Intuitive Idea for an Algorithm

- m proposes to w
  - If w is unmatched, w accepts
  - If w is matched to m_2
    - If w prefers m to m_2, w accepts
    - If w prefers m_2 to m, w rejects
- Unmatched m proposes to highest w on its preference list that m has not already proposed to