# CS207 Design and Analysis of Algorithms

Sajith Gopalan

Indian Institute of Technology Guwahati

*sajith@iitg.ac.in*

January 8, 2022

# Algorithm design techniques

# Algorithm design is an art

- Many are algorithms that generate a sense of wonder with a design that does not fit any paradigm
- Still very many problems can be solved using well known techniques
- Some well known tricks of the trade are
  - Divide and conquer
  - Dynamic programming
  - Greedy method
  - Backtracking
  - Branch and bound
  - Local Search

# Divide and Conquer

- Solve a problem recursively
  - Divide the problem into a number of smaller instances of the same problem
  - For each of these subproblems, if its size is sufficiently large, then Conquer it recursively, else Conquer it directly
  - Combine the solutions to the subproblems into the solution for the original problem

# Dynamic Programming

- Particularly useful for optimization problems. Solution space. Constraints. Many feasible solutions, each of which satisfies the constraints, and has a value. We wish to find one feasible solution that minimises (maximizes) value.
- Steps:
  - Formulate the structure of an optimal solution
  - Recursively define the value of an optimal solution
  - Compute the value of an optimal solution, typically in a bottom-up fashion
  - Construct an optimal solution from computed information
- Works particularly when the following properties hold:
  - Optimal Substructure: an optimal solution to the problem contains within it optimal solutions to subproblems
  - Overlapping subproblems: recursive algorithm for the problem solves the same subproblems repeatedly

# Greedy Method

- Works for some optimization problems, not all
- A greedy choice is one that looks best at the moment
- Steps:
  - Formulate a solution in which we make a *greedy* choice and are left with one subproblem.
  - Prove that there is always an optimal solution reachable thorugh the greedy choice; that is, the greedy choice is safe
  - Devise a way to combine an optimal solution to the subproblem with the greedy choice to obtain an optimal solution to the original problem

# Backtracking

- Useful for constraint satisfaction problems (CSPs)
- (A CSP has a set of variables, each with a domain of values, and a set of constaints on these variables specified through relations on them. We need to find values for all variables, so that all constraints are satisfied. E.g., Sudoku)
- Incrementally builds candidates to the solutions
- Abandons a candidate ("backtracks") as soon as it becomes clear that the candidate cannot be completed into a valid solution

# Branch and Bound

- Branch and bound (BB) is useful for various kinds of optimization problems
- Applicable when the set $S$ of candidate solutions can be partitioned using a rooted tree
- The root corresponds to $S$
- Subtrees rooted at the root's children represent a partition of $S$ into disjoint subsets
- Before exploring a branch, we check it against upper and lower estimated bounds on the optimal solution
- The branch is not explored if it cannot produce a solution better than the best one found so far
- If estimations are not possible, then the algorithm is an exhaustive search

# Local Search

- A heuristic method for solving computationally hard optimization problems
- Move from feasible solution to feasible solution by applying local changes, until a locally optimal solution is found or a time bound is exceeded
- Analogy: A ball rolling down a slope settles at the bottom of the first ditch, take it out on the other side until it starts rolling down again; repeat when it settles at the bottom of another ditch; continue until your time runs out; declare the lowest point found as *the lowest point*