Design & analysis of Algorithms (17-18/II)

- You want to solve a problem on some computational device.

  The process or the recipe that you devise, to solve the problem, is called an algorithm.

- This course teaches you how to:
  i) design an algorithm,
  ii) prove the correctness, &
  iii) prove the time (or space) complexity.

- We will study practical algorithms.

  But, we will not go into the detailed implementation.

  The complexity analysis will be done using asymptotic notation & may be unoptimized!

Paradigm

- Often problems / algorithms share certain approaches. They are broadly called paradigms.

- Some of our paradigms are:
  i) Recursion (or, divide & conquer)
  ii) Greedy (or, local to global)
  iii) Dynamic programming (iterative version of (i))
  iv) Amortization (average time per operation)
  v) Collection of problems & Reductions (or, complexity classes)
  vi) Approximation, Randomization

- We will define these as we go along. It requires creativity & practice to see which of these paradigms could solve a given problem.

Data Structure

- Fast algorithms rely on fast data accessibility. Often we have to organize data in a structure.

- Some of them are:
  i) Binary search tree (BST)
  ii) Red-black tree
  iii) Augmented BST
  iv) Heaps

- Basically, they have the structure of a tree (in contrast to an array!) & varying implementations of insert/delete/search.

- Note that eventually data is stored as sequence of bits in RAM or Disk.
  The data structure in this course will only be an abstract/logical construct.

Policies

- The grading will be based on the following components (roughly):

  - Assignment - 15%
  - Quiz - 15%
  - Midsem - 30%
  - Endsem - 40%

- A TA will be assigned to each ~20 students. You are encouraged to contact them for help & submit assignments.

- If you have not done a good algorithms course already, then you will need a lot of practice before you become confident in applying the tools.

  Your first attempt should be yours, without taking any help!

Divide & Conquer Paradigm

- Given a problem the approach is:
  i) Divide the problem instance into chunks,
  ii) Solve each chunk,
  iii) Combine the smaller solutions to solve the original instance.

- Examples?
  - Merge Sort, Quick Sort....
  - Multiply two polynomials or integers...
  - # inversions in a given array.
  - Median finding in linear time!

- Sometimes this paradigm is bad (?)
  Eg. determinant computation.