Review: Top Down Design

Algorithm Design

Strategy: Top-Down Design

Technique: Divide-and-Conquer

Algorithm Design – Top Down Design

- Top-Down Design (Top Down Decomposition)
  1. Divide the problem into sub problems.
  2. Find solutions for sub problems
  3. Combine the sub solutions.
- Note:

  Steps 1 and 3 are not independent!

End of Note.

Top Down Design – Atomic Sub-Problems

- Top-Down Design (Top Down Decomposition)
  1. Divide the problem into sub problems.
  2. Find solutions for sub problems
  3. Combine the sub solutions.
- How do we find solutions for sub problems?
  - Apply top-down design recursively
    - Q: When do we stop dividing?
    - A: When we reach “atomic” problems.
      - Atomic problems have known solutions
        - Exercise: Provide examples!

Top Down Design – Number of Sub-Problems

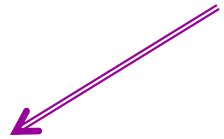
- Question:
  - When a problem is divided into sub-problems how does one decide on the number of sub-problems (in general)?
- Answer:
  - In general, working with fewer sub-problems is easier to manage
    - i.e. results in lesser overhead – at design time as well as at execution time – as the division is recursive!
- Exception:
  - For instance, if the underlying machine model includes a large number or processors:
    - then the number of sub-problems should (eventually, i.e. after some number of divisions) be large
      - so as to utilize the number of processors.

Design Technique: Divide-and-Conquer

- **Divide-and-Conquer** is a special case of Top-Down Design:
  - The structure of one or more sub-problems is same as that of the (original) problem:

Divide-and-Conquer - Induction

- Divide-and-Conquer is a special case of Top-Down Design:
  - The structure of one or more sub-problems is the same as that of the (original) problem:
    - i.e. the structure of the problem is inductive and
    - therefore the structure of the solution may be inductive / recursive



Induction referred to in here may not be numeric: it is usually ***structural induction***

  - Of course for the recursion to terminate (or the induction to have a base case)
    - the sub-problem(s) of the same structure as the original problem must have size(s) that is/are less than the size of the original problem.