Decrease & Conquer

Design & Analysis of Algorithms

Department of Computer Science, Fergusson College,(Autonomous) Pune

Decrease-and-Conquer

1. Reduce problem instance to smaller instance of the same problem
2. Solve smaller instance
3. Extend solution of smaller instance to obtain solution to original instance

- Can be implemented either top-down or bottom-up
- Also referred to as *inductive* or *incremental* approach

3 Types of Decrease and Conquer

- Decrease by a constant (usually by 1)
- Decrease by a constant factor (usually by half)
- Variable-size decrease

Decrease by a constant :

- Insertion sort
- Topological sorting
- Algorithms for generating permutations, subsets

Decrease by a constant factor

- Binary search
- Exponentiation by squaring

Variable-size decrease

- Euclid’s algorithm
- Selection by partition

Problem: Compute a^n

- Decrease by one

$$a^n = \begin{cases} a^{n-1} \times a, if\ n > 0 \\ 1, if\ n = 0 \end{cases}$$

Problem: Compute a^n

- Decrease by constant factor
  (*by half*)

$$a^n = \begin{cases} \left(a^{\frac{n}{2}}\right)^2, if\ n\ is\ even\ and\ positive \\ (a^{(n-1)/2})^2 \times a, if\ n\ is\ odd \\ 1, if\ n = 0 \end{cases}$$

Insertion Sort

- To sort array A[0..*n*-1], sort A[0..*n*-2] recursively and then insert A[*n*-1] in its proper place among the sorted A[0..*n*-2]
- Usually implemented bottom up (non-recursively)

Analysis of Insertion Sort

- Space efficiency: in-place
- Stability: yes
- Best elementary sorting algorithm overall
- Binary insertion sort

Analysis of Insertion Sort

- Time efficiency: Worst case

$$C_{worst}(n) = n(n\text{-}1)/2 \in \Theta(n^2)$$

- Each element must be compared to all preceding elements

Analysis of Insertion Sort

- Time efficiency: Best case

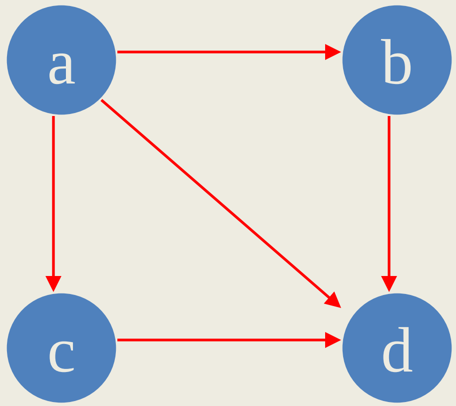
  $$C_{best}(n) = n - 1 \in \Theta(n)$$

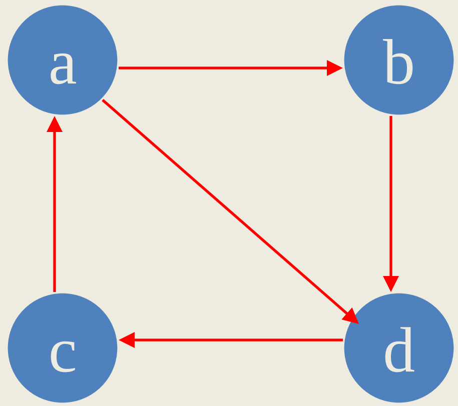
  (also fast on almost sorted arrays)
- Only one comparison is necessary for each element

Dags and Topological Sorting

A *dag*: a directed acyclic graph, i.e. a directed graph with no (directed) cycles

DAG





Not a DAG

Arise in modeling many problems that involve prerequisite constraints (construction projects, document version control)

Vertices of a dag can be linearly ordered so that for every edge its starting vertex is listed before its ending vertex (*topological sorting*). Being a dag is also a necessary condition for topological sorting be possible.

Topological Sorting Example

Order the following items in a food chain