CS 395 – Analysis of Algorithms

Chapter 4 – Decrease-and-Conquer

Read 4.1

- **Decrease-and-Conquer techniques**
  - **General idea**
  - **Three types of Decrease-n-Conquer algorithms**
- **Insertion Sort**
- **Shellsort (reading assignment)**

Decrease-and-Conquer

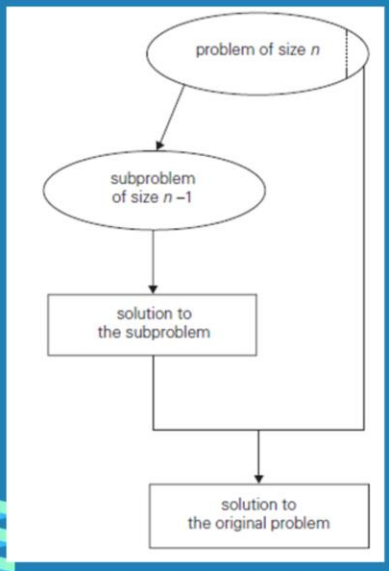
- **Based on exploiting relationship between a solution to the given instance of a problem and that to a smaller instance**
  - **Reduce problem instance to smaller instance of the same problem**
  - **Solve smaller instance**
  - **Extend solution of smaller instance to obtain solution to original instance**
- **Can be implemented either**
  - **bottom-up** (usually implemented iteratively) or
  - **top-down** (usually implemented using recursion)
- **Also referred to as *inductive* or *incremental* approach**

2

3 Types of Decrease and Conquer

- *Decrease by a constant* (usually by 1):
  - insertion sort
  - topological sorting
  - algorithms for generating permutations, subsets
- *Decrease by a constant factor* (usually by half)
  - binary search and bisection method
  - exponentiation by squaring
  - multiplication à la russe
- *Variable-size decrease*
  - Euclid's algorithm
  - selection by partition
  - Nim-like games

Decrease-(by one)-and-Conquer



For example, computing a^n

$a^n = a^{n-1}\ a$

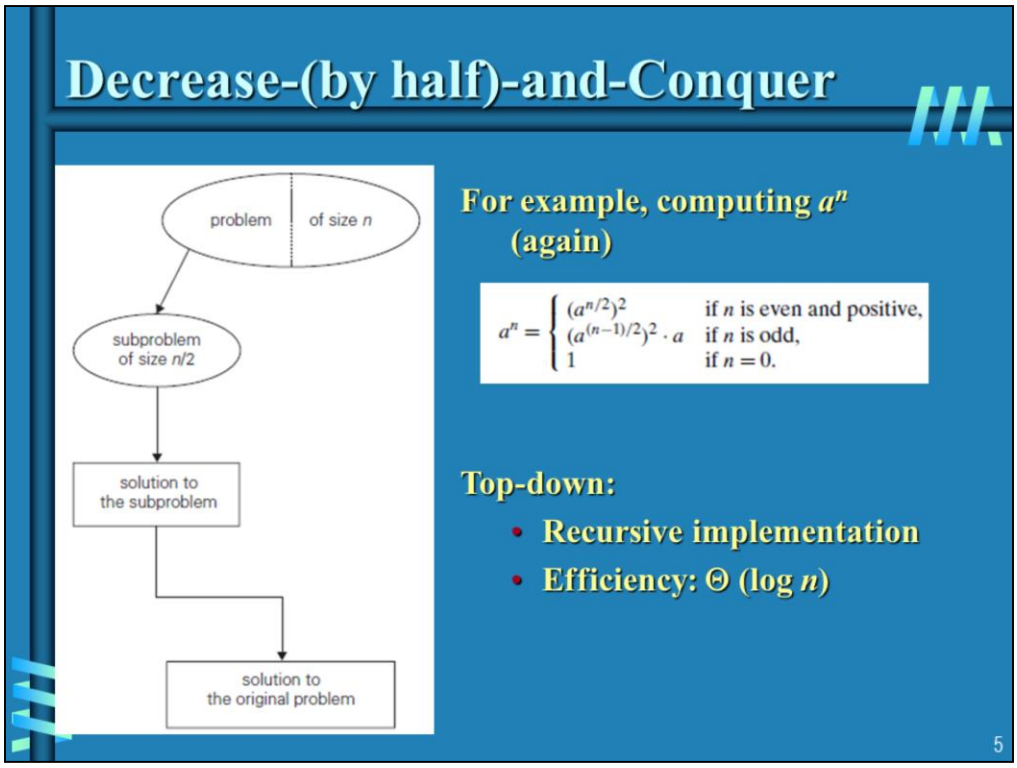
Top-down:

$$f(n) = \begin{cases} f(n-1) \cdot a & \text{if } n > 0, \\ 1 & \text{if } n = 0, \end{cases}$$

Bottom-up:

multiply 1 by a n times.

- **Similar to brute-force**
- **… but coming from a different thought process**



If the instance of size n is to compute a^n, the instance of half its size is to compute $a^{n/2}$, with the obviou

$a^n = (a^{n/2})^2$.

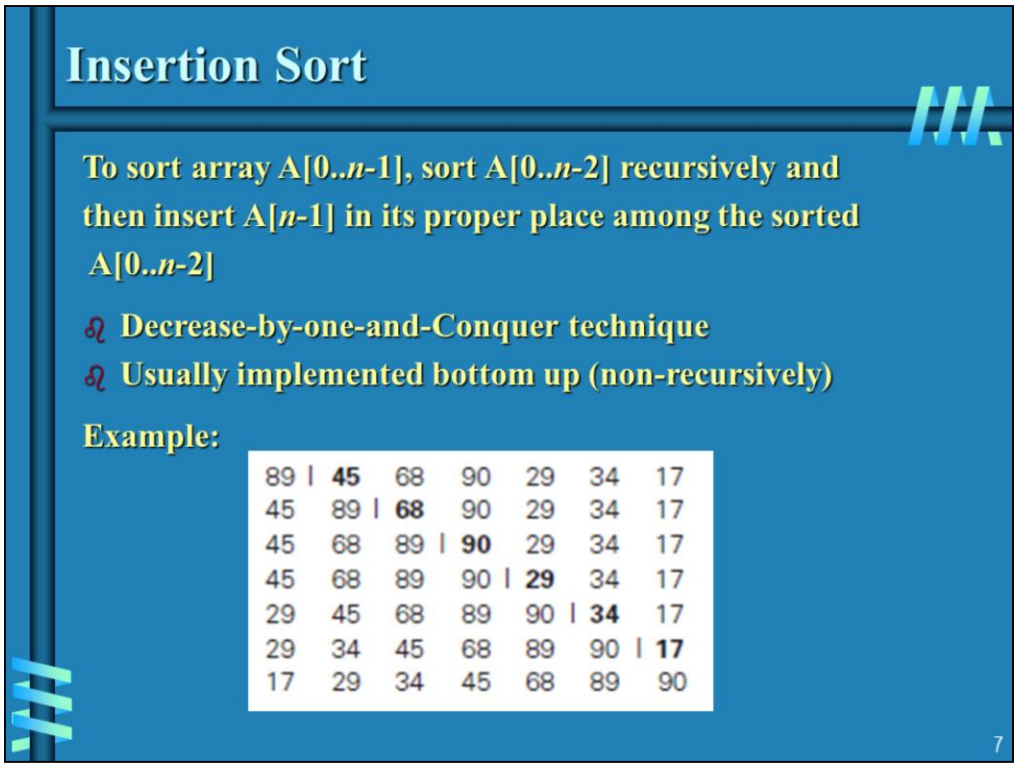
But this does not work for odd n.

If n is odd, we have to compute a^{n-1} by using the rule for even-valued exponents and then multiply the

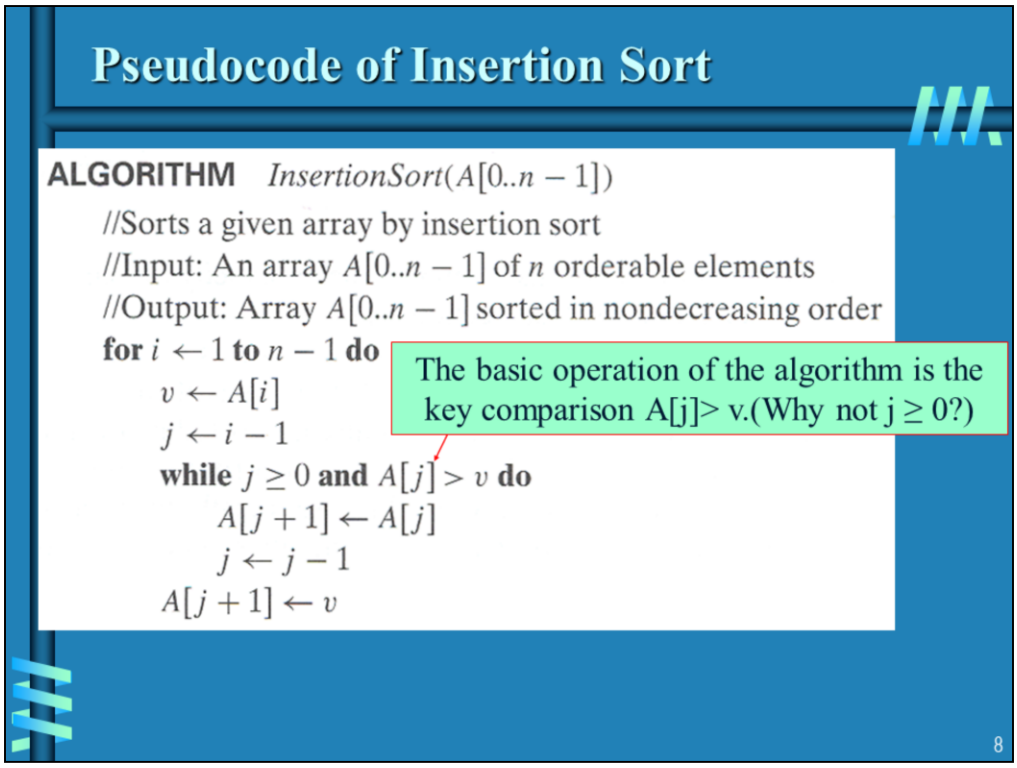
Decrease-(by variable-size)-and-Conquer

- Size-reduction pattern varies from one iteration to another

- Example: Euclid's algorithm for GCD
  - gcd(m, n) = gcd(n, m mod n)
  - Second argument decreasing at variable rate



If A[n − 1] elements are sorted, all we need to do is scan the sorted subarray from right to left until the

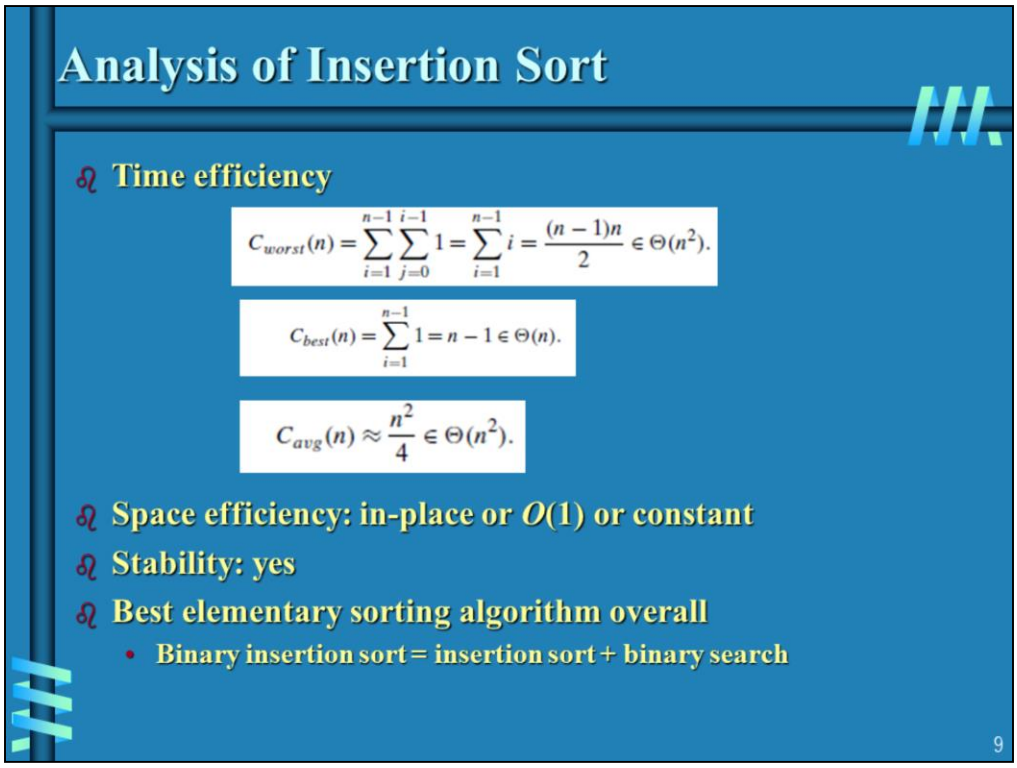


Why not j ≥ 0? Because it is almost certainly faster than the former in an actual computer implementat

Moreover, it is not relevant to the algorithm it is just avoiding an out of bounds error.

a better implementation with a **sentinel** —see Problem 8 in this section's exercises—eliminates it alto

- In general, a sentinel value makes it possible to detect the end of the data when no out-of-band data
- In general the value should be selected in such a way that it is guaranteed to be distinct from all leg
- In this case, the sentinel should stop the smallest element from moving beyond the first position in t



The worst-case input is an array of strictly decreasing values.

Thus, in the worst case, insertion sort makes exactly the same number of comparisons as selection sort

The best case is when the array is already sorted.

- Quick sort is worst on an already sorted array.

A rigorous analysis of the algorithm's average-case efficiency is based on investigating the number of