UNIT 2:
Decrease and Conquer

Decrease-and-conquer technique
Insertion sort

Decrease and Conquer

Is based on exploiting the relationship between a solution to a given instance of a problem and a solution to its smaller instance.

Sertorius, in order to teach his soldiers that **perseverance and wit** are better than brute force, had two horses brought before them, and set two men to pull out their tails.

One of the men was very strong, who tugged and tugged, but all to no purpose; the other was sharp, and witty, who plucked one hair at a time, amidst roars of laughter, and soon left the tail quite bare.

(Dictionary of Phrase and Fable, 1898)

Puzzle time!!

Fake coin problem:

You are given 8 coins which look exactly alike but one is fake. Assume the fake coin is slightly lighter than the real coins. You are asked to determine the fake coin using a balance.

In how many steps can you guarantee to find the fake coin?

What are the steps?

Solution to fake coin problem:

1. Put 4 coins on each side of the balance. Discard the coins on the side that is heavier because we know the fake coin is on the lighter side.
2. From the 4 put 2 coins on each side of the balance. Discard the coins on the side that is heavier. We now know that the fake coin is one of two.
3. Put one coin on each side. The coin that is lighter is the fake coin.

Total: 3 steps/comparisons

Problem continues…

Fake coin problem:

What is the difference in the strategy if we have 9 coins?

Will it take more steps to do 9 coins?

Solution to fake coin problem with 9 coins:

1. Put 4 coins on each side of the balance. Keep one coin aside. If the balance is level then the fake coin is the one to the side. **STOP**
2. If the balance is not level, discard the coins on the side that is heavier because we know the fake coin is on the lighter side. From the 4 put 2 coins on each side of the balance. Discard the coins on the side that is heavier. We now know that the fake coin is one of two.
3. Put one coin on each side. The coin that is lighter is the fake coin.

Total: 1 step or 3 steps

Problem again continues…

Fake coin problem:

Now there are 12 coins.

In how many steps can you guarantee to find the fake coin?

What are the steps?

Solution to fake coin problem with 12 coins:

1. Put 6 coins on each side of the balance. Discard the coins on the side that is heavier because we know the fake coin is on the lighter side.
2. From the 6 put 3 coins on each side of the balance. Discard the coins on the side that is heavier. We now know that the fake coin is 1 of 3.
3. Out of 3, keep 1 coin aside, and put 1 coin on each side of the balance. If the balance is level then the fake coin is the one to the side. Else the coin that is lighter is the fake coin.

Total: 1 step or 3 steps

Decrease and Conquer: General Plan

1. **Decrease**/reduce problem instance to smaller instance of the same problem
2. **Conquer**/solve smaller instance
3. **Extend** solution of smaller instance to obtain solution to original instance

(Can be implemented either top down (recursively) or bottom up (without a recursion))

Decrease and Conquer: Three variations

1. decrease by a constant
2. decrease by a constant factor
3. variable size decrease

Decrease and Conquer: Decrease by a constant

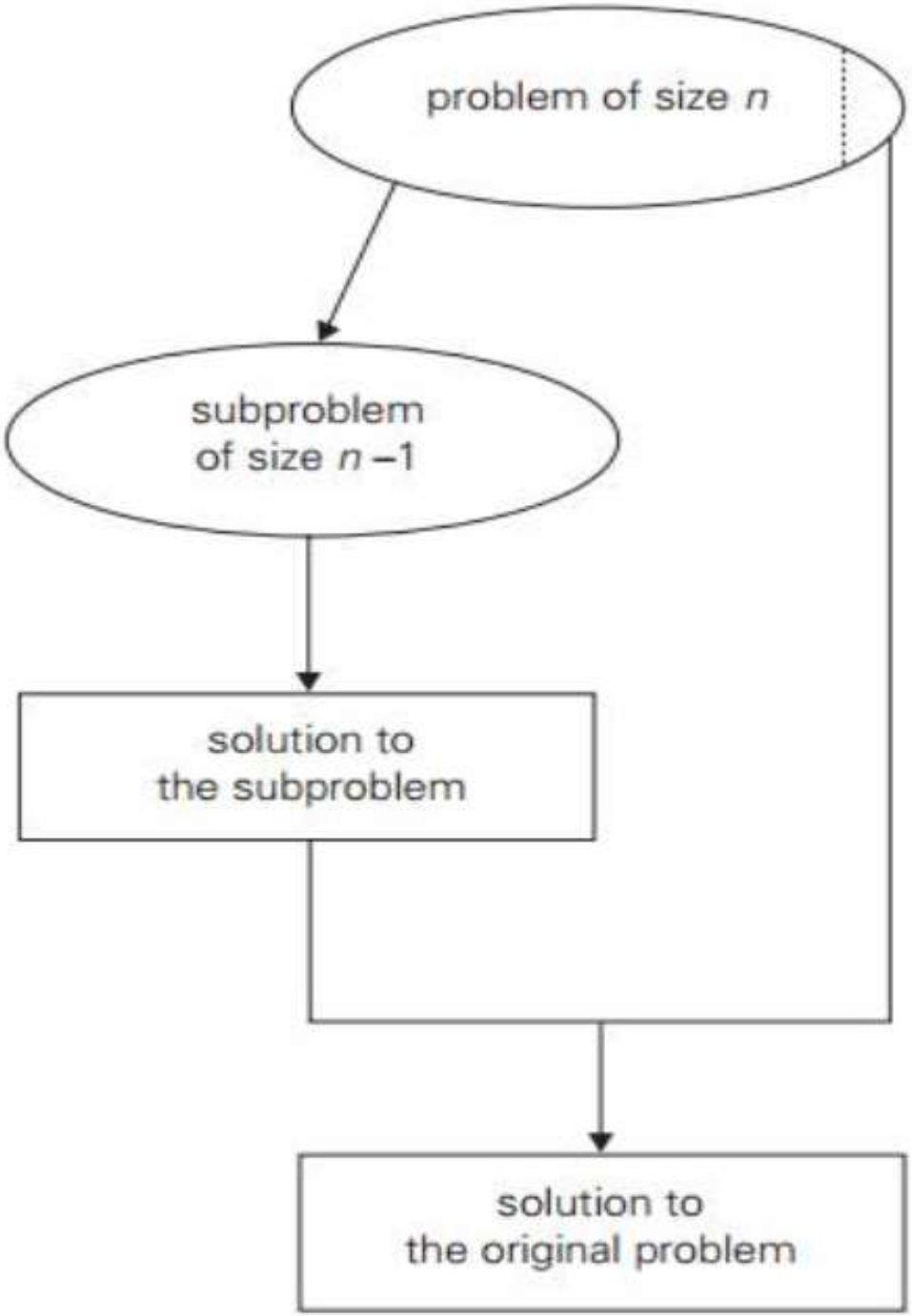
- problem size is reduced by the same constant on each iteration of the algorithm. Typically, this constant is equal to one.

computing a^n for positive integer exponents.

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

Examples:

- insertion sort
- graph traversal algorithms (DFS and BFS)
- topological sorting
- algorithms for generating permutations, subsets



Decrease-and-conquer technique (Decrease by a constant)

A problem's instance of size n is decreased by one

Decrease and Conquer: Decrease by a constant factor

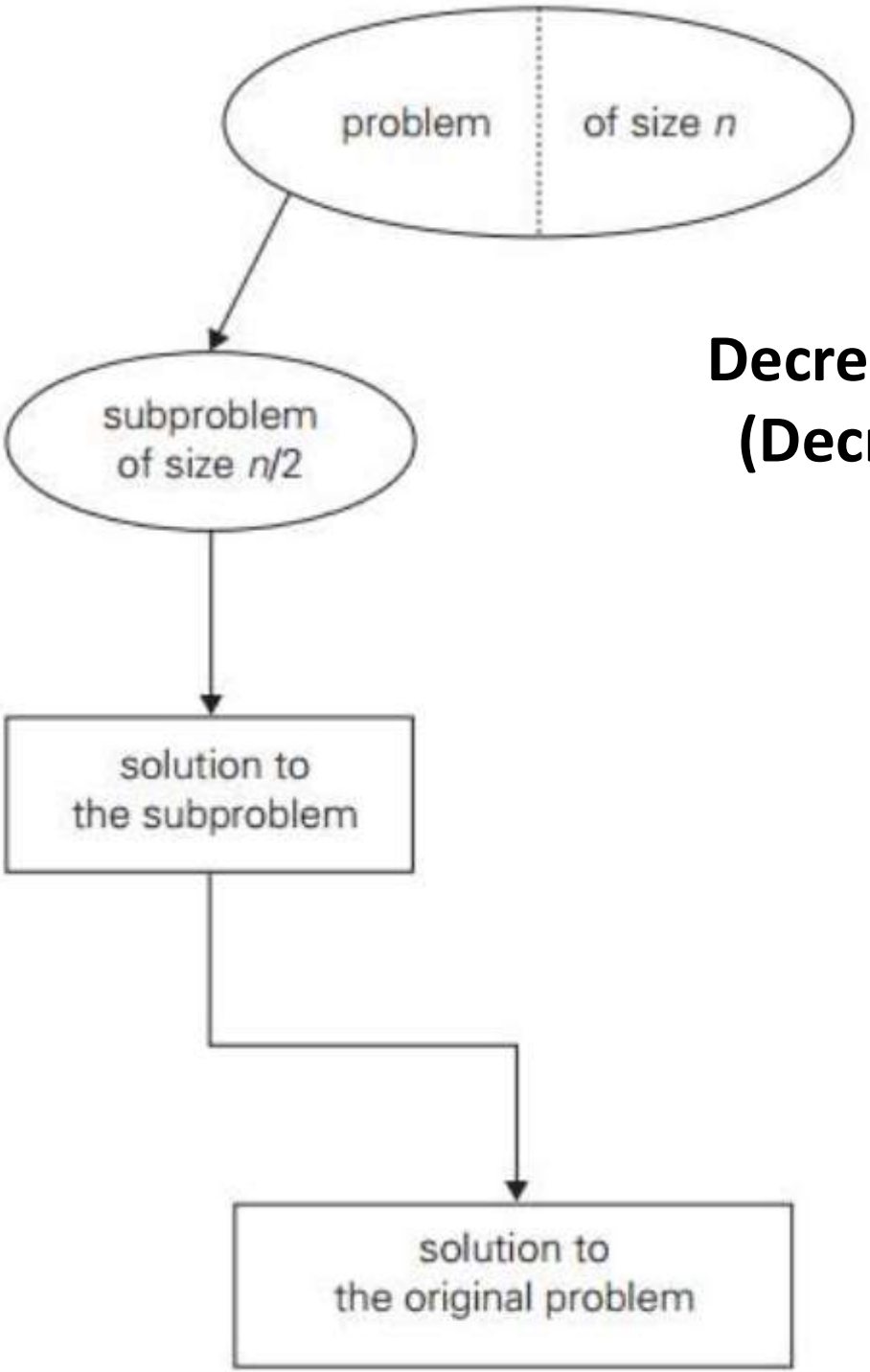
- problem size is reduced by the same constant factor on each iteration of the algorithm. Typically, this constant factor is equal to two.

computing a^n for positive integer exponents.

$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even and positive,} \\ (a^{(n-1)/2})^2 \cdot a & \text{if } n \text{ is odd,} \\ 1 & \text{if } n = 0. \end{cases}$$

Examples:

- binary search and bisection method
- exponentiation by squaring



Decrease-and-conquer technique
(Decrease by a constant factor)

A problem's instance of size n is decreased by half

Decrease and Conquer: Variable size decrease

- size reduction pattern varies from one iteration of an algorithm to another.

Euclid's algorithm for computing the GCD.

$$\gcd(m, n) = \gcd(n, m \bmod n).$$

Examples:

- Euclid’s algorithm
- selection by partition

Let's check our understanding…

What's the difference?

Consider the problem of exponentiation:

Compute x^n using

1. Brute Force
2. Divide and conquer
3. Decrease by one
4. Decrease by constant factor

Brute force

$a * a * \ldots * a$

n-1 multiplications

Divide and conquer

$$a^n = \begin{cases} a^{\lfloor n/2 \rfloor} \cdot a^{\lceil n/2 \rceil} & \text{if } n > 1 \\ a & \text{if } n = 1. \end{cases}$$

T(n) = 2T(n/2) + 1

n-1 multiplications

Decrease by One

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

T(n) = T(n-1) + 1

n-1 multiplications

Decrease by Constant factor

$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even and positive,} \\ (a^{(n-1)/2})^2 \cdot a & \text{if } n \text{ is odd,} \\ 1 & \text{if } n = 0. \end{cases}$$

T(n) = T(n/a) + a-1

$= (a-1)\log_a n$

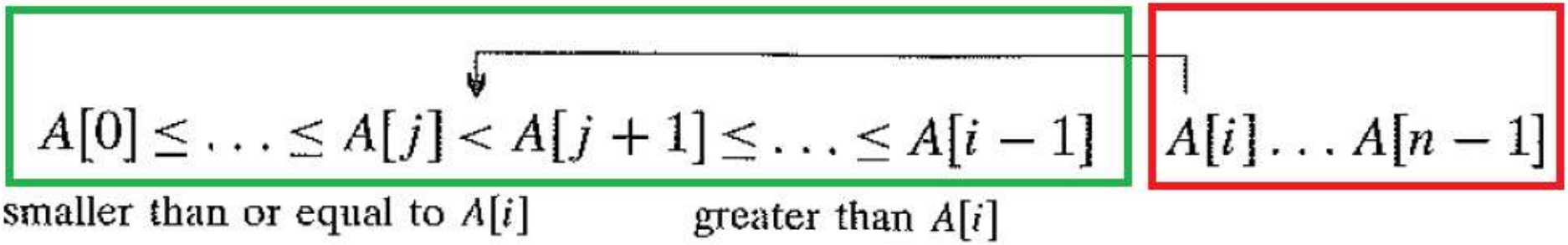
$= \log_2 n$ (when a=2)

Insertion sort

- Simple, general-purpose, comparison-based sorting algorithm
- stable and in-place sorting algorithm
- uses decrease and conquer (decrease-by-one) strategy

Working of Insertion sort

- iterates, consuming one input element each repetition, and grows a sorted output list.
- At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there.
- It repeats until no input elements remain.



Insertion sort

ALGORITHM *InsertionSort*$(A[0..n-1])$

//Sorts a given array by insertion sort
//Input: An array $A[0..n-1]$ of n orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 1$ **to** $n-1$ **do**
    $v \leftarrow A[i]$
    $j \leftarrow i-1$
    while $j \geq 0$ **and** $A[j] > v$ **do**
        $A[j+1] \leftarrow A[j]$
        $j \leftarrow j-1$
    $A[j+1] \leftarrow v$

Insertion sort : Visualization

https://visualgo.net/bn/sorting

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Insertion sort algorithm analysis

1. input's size: **n** – number of elements to be sorted.
2. basic operation: **comparison**

$$A[j] > v$$

3. **depends on the nature of the input, hence worst, average, and best cases exists.**
4. Let C(n) = number of times the basic operation is executed.

Insertion sort : Best case

- **when the array is already sorted**
- comparison A[j] > v is executed only once on every iteration of the outer loop.

Say ascending order: A[i-1] ≤ A[i] for every i=1,...,n-1

$$C_{best}(n) = \sum_{i=1}^{n-1} 1$$

$$= n - 1$$

$$\in \Theta(n).$$

Insertion sort : Worst case

- **when the array is sorted in reverse order**
- comparison A[j] > v is executed largest number of times, i.e., for every j = i-1, ... , 0.

Say strictly decreasing values,

A[0]>A[1] (for i= 1), A[1]>A[2] (for i= 2), ... , A[n- 2]>A[n -1] (for i=n-1)

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1$$

$$= \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \Theta(n^2)$$

Insertion sort : Average case

- **randomly ordered array**
- based on investigating the number of element pairs that are out of order

$$C_{avg}(n) \approx \frac{n^2}{4} \in \Theta(n^2)$$

Insertion sort : Applications

- sorting short lists
- sorting “almost sorted” lists
- sorting small sub-lists in Quicksort

Drawback:

For unsorted/reverse-sorted array, it's slow for large n value.

Insertion sort version and variants

- Straight insertion sort
- Binary insertion sort
- Improved insertion sort to reduce unnecessary swaps
- Insertion sort using skip list
- Shell sort
- Library sort