



DESIGN AND ANALYSIS OF ALGORITHMS

Surabhi Narayan

Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

DECREASE AND CONQUER

Surabhi Narayan

Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

Decrease and Conquer



- **Decrease** or reduce problem instance to smaller instance of the same problem and extend solution.
 - **Conquer** the problem by solving smaller instance of the problem.
 - **Extend** solution of smaller instance to obtain solution to original problem .
 - **Exploit** the relationship between a solution to a given instance of a problem and a solution to its smaller instance.
-
- Can be implemented either top-down or bottom-up
 - Also referred to as *inductive* or *incremental* approach

DESIGN AND ANALYSIS OF ALGORITHMS

Decrease and Conquer



3 Types of Decrease and Conquer

Decrease by a constant (usually by 1):

- insertion sort
- graph traversal algorithms (DFS and BFS)
- 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

This usually results in a recursive algorithm.

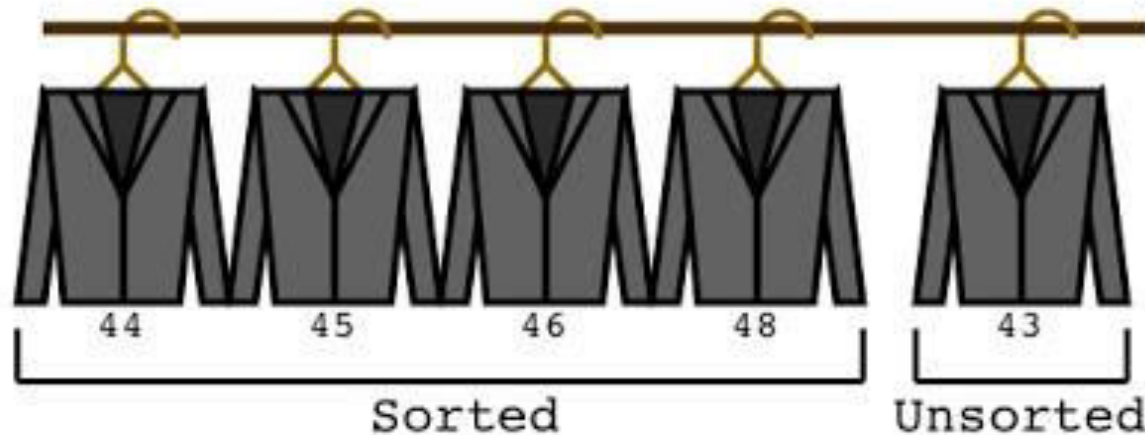
Insertion Sort

Imagine a card game

Cards in your hand are sorted.

The dealer hands you exactly one new card.

How would you rearrange your cards



Insertion Sort

- Insertion sort is based on the idea that one element from the input elements is consumed in each iteration to find its correct position i.e, the position to which it belongs in a sorted array.
- grows the sorted array at each iteration
- compares the current element with the largest value in the sorted array.
- If the current element is greater, then it leaves the element in its place and moves on to the next element else it finds its correct position in the sorted array and moves it to that position.
- This is done by shifting all the elements, which are larger than the current element, in the sorted array to one position ahead

Insertion Sort

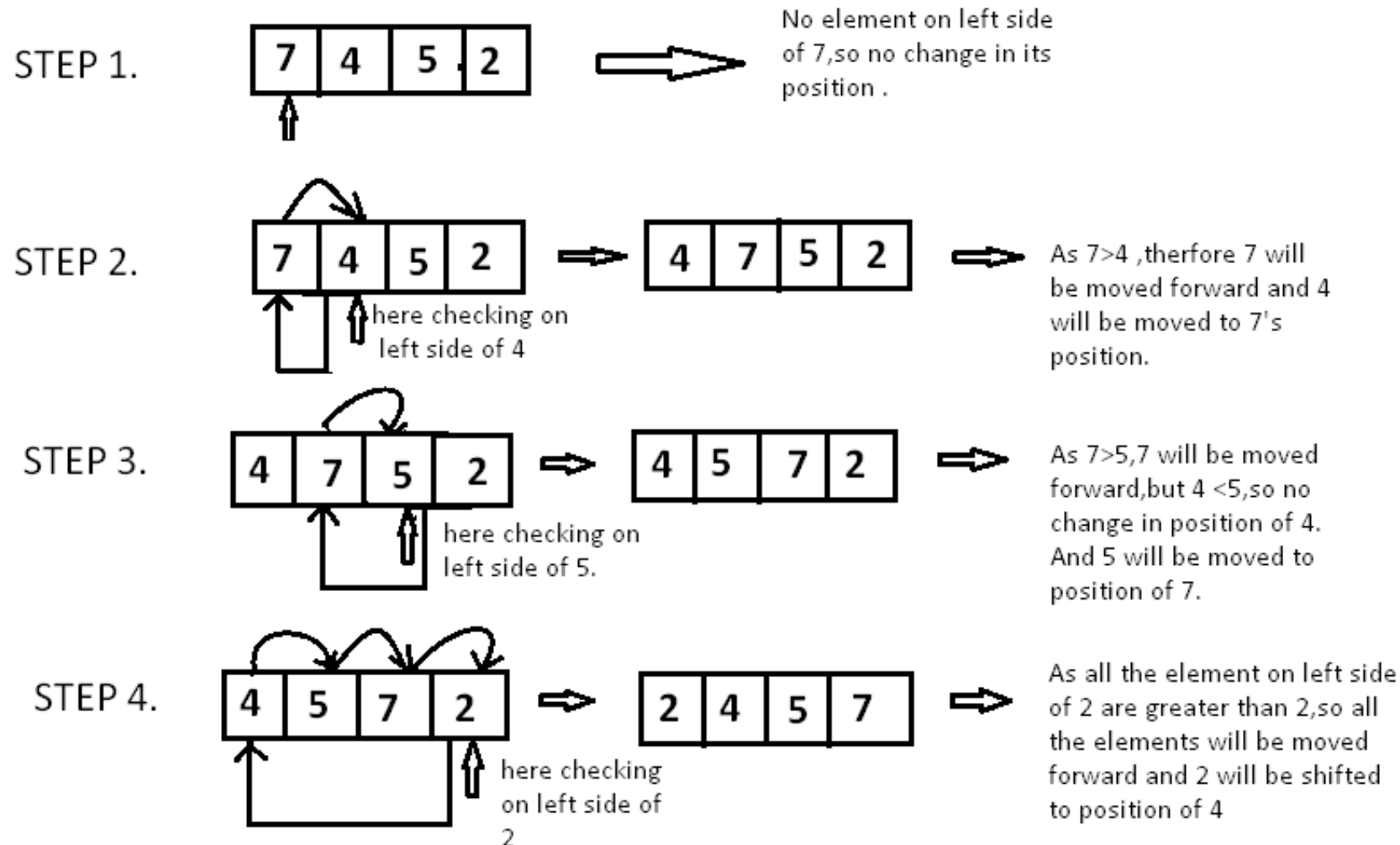
To sort array $A[0..n-1]$, sort $A[0..n-2]$ recursively and then insert $A[n-1]$ in its proper place among the sorted $A[0..n-2]$

Usually implemented bottom up (nonrecursively)

Example: Sort 6, 4, 1, 8, 5

```
6 | 4 1 8 5
  4 6 | 1 8 5
  1 4 6 | 8 5
  1 4 6 8 | 5
  1 4 5 6 8
```

Insertion Sort



DESIGN AND ANALYSIS OF ALGORITHMS

Decrease and Conquer

Insertion Sort

54	26	93	17	77	31	44	55	20	Assume 54 is a sorted list of 1 item
26	54	93	17	77	31	44	55	20	inserted 26
26	54	93	17	77	31	44	55	20	inserted 93
17	26	54	93	77	31	44	55	20	inserted 17
17	26	54	77	93	31	44	55	20	inserted 77
17	26	31	54	77	93	44	55	20	inserted 31
17	26	31	44	54	77	93	55	20	inserted 44
17	26	31	44	54	55	77	93	20	inserted 55
17	20	26	31	44	54	55	77	93	inserted 20

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**

$A[j + 1] \leftarrow A[j]$

$j \leftarrow j - 1$

$A[j + 1] \leftarrow v$

Time efficiency

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

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

$$C_{best}(n) = n - 1 \in \Theta(n) \text{ (also fast on almost sorted arrays)}$$

Space efficiency: in-place

Stability: yes

Best elementary sorting algorithm overall

Binary insertion sort



THANK YOU

Surabhi Narayan

Department of Computer Science & Engineering

surabhinarayan@pes.edu