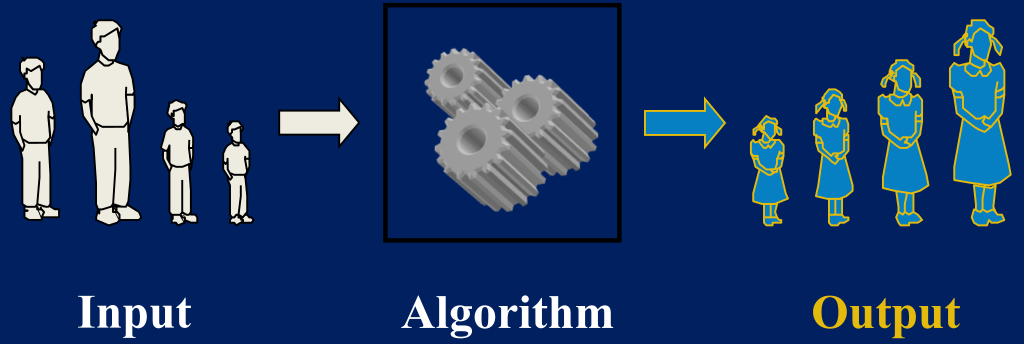
DESIGN & ANALYSIS OF ALGORITHM (BCSC0012)

Chapter 4: Sorting

Insertion Sort



Prof. Anand Singh Jalal

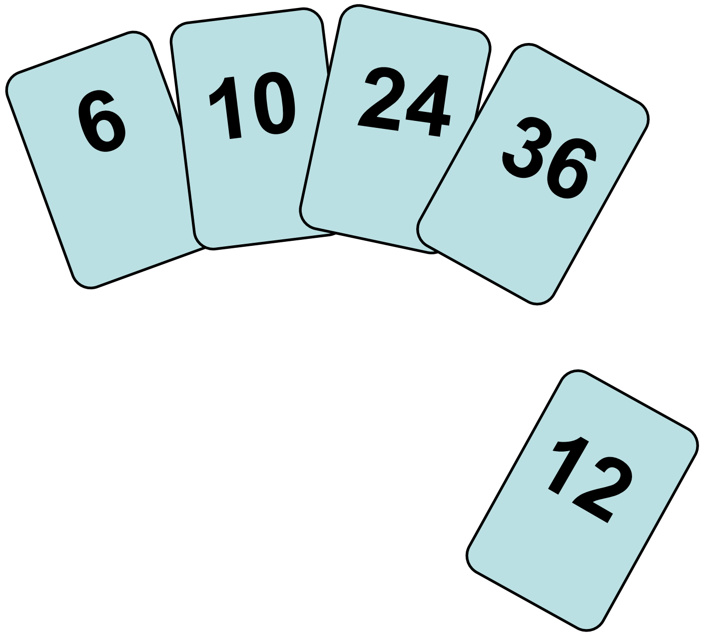
Department of Computer Engineering & Applications

Insertion Sort

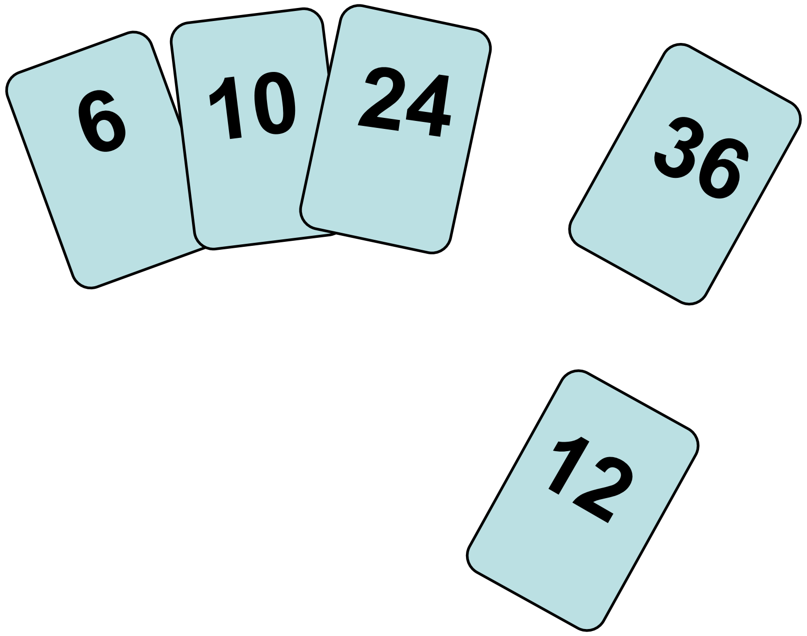
- **Idea:** like sorting a hand of playing cards
  - Start with an empty left hand and the cards facing down on the table.
  - Remove one card at a time from the table, and insert it into the correct position in the left hand
    - compare it with each of the cards already in the hand, from right to left
  - The cards held in the left hand are sorted
    - these cards were originally the top cards of the pile on the table

Insertion Sort …

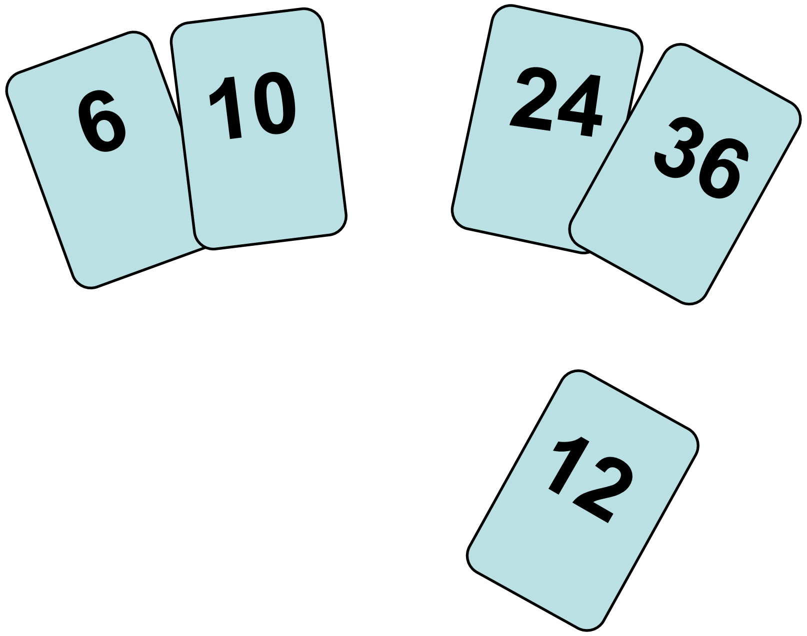
To insert 12, we need to make room for it by moving first 36 and then 24.



Insertion Sort …



Insertion Sort ...



GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

Insertion Sort ...

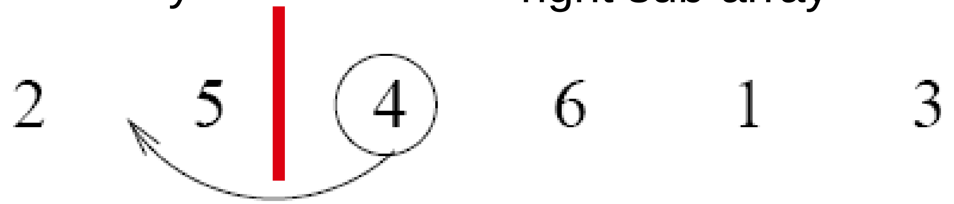
input array

5 2 4 6 1 3

at each iteration, the array is divided in two sub-arrays:

left sub-array

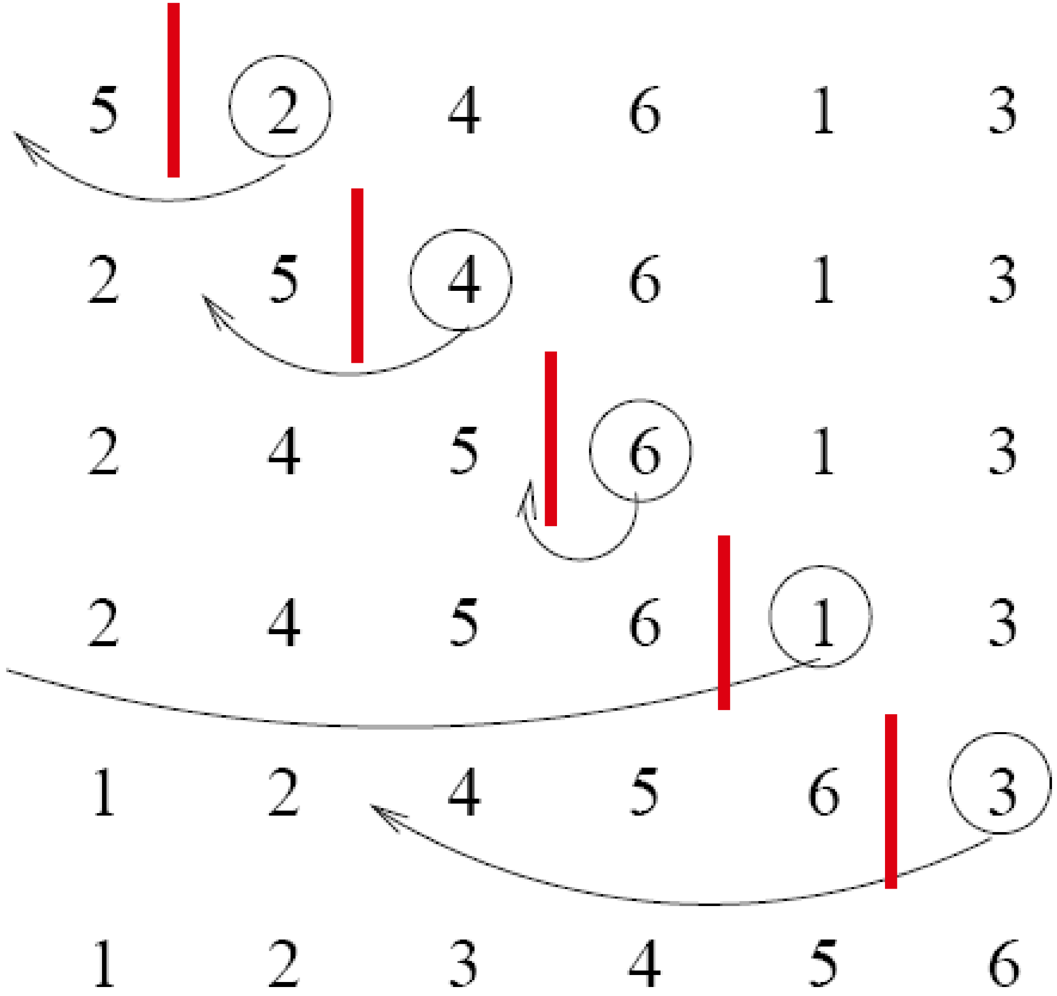
right sub-array

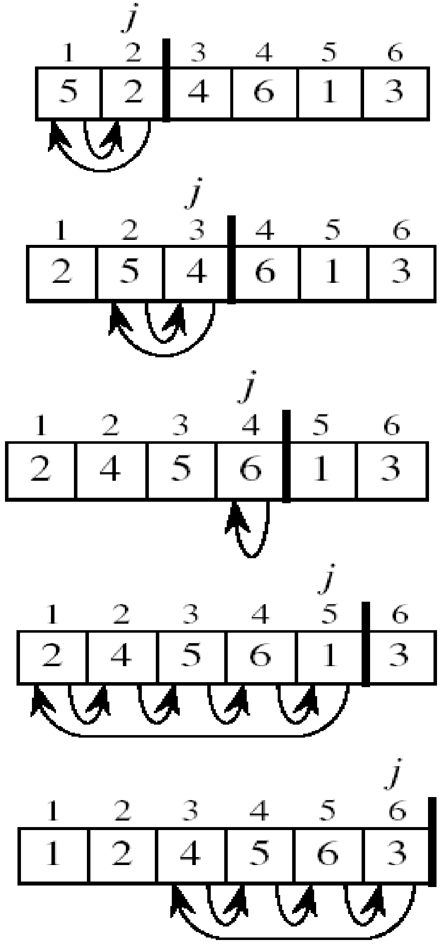


sorted

unsorted

Insertion Sort ...





Insertion Sort ...

Alg.: INSERTION-SORT(*A*)

1	2	3	4	5	6	7	8
a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8

key

for j ← 2 **to** n

  do key ← *A*[j]

    ▷Insert *A*[j] into the sorted sequence *A*[1 . . j -1]

    i ← j - 1

    while i > 0 and *A*[i] > key

      do *A*[i + 1] ← *A*[i]

        i ← i - 1

    A[i + 1] ← key

- Insertion sort – sorts the elements in place

Analysis of Insertion Sort

INSERTION-SORT(*A*)

	cost	times
for j ← 2 **to** n	c_1	n
do key ← A[j]	c_2	n-1
▷Insert A[j] into the sorted sequence A[1 . . j -1]	0	n-1
i ← j - 1	c_4	n-1
while i > 0 and A[i] > key	c_5	$\sum_{j=2}^{n} t_j$
do A[i + 1] ← A[i]	c_6	$\sum_{j=2}^{n} (t_j - 1)$
i ← i – 1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
A[i + 1] ← key	c_8	n-1

t_j: # of times the while statement is executed at iteration j

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}\left(t_j - 1\right) + c_7\sum_{j=2}^{n}\left(t_j - 1\right) + c_8(n-1)$$

Insertion Sort: Best Case Analysis

- The array is already sorted **“while** i > 0 and A[i] > key”
  - $A[i] \le key$ upon the first time the **while** loop test is run
    (when $i = j - 1$)
  - $t_j = 1$
- $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$

$= (c_1 + c_2 + c_4 + c_5 + c_8)n + (c_2 + c_4 + c_5 + c_8)$

$= an + b = \Theta(n)$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}\left(t_j - 1\right) + c_7\sum_{j=2}^{n}\left(t_j - 1\right) + c_8(n-1)$$

Insertion Sort: Worst Case Analysis

- The array is in reverse sorted order **“while** i > 0 and A[i] > key”
  - Always A[i] > key in **while** loop test
  - Have to compare key with all elements to the left of the j-th position $\Rightarrow$ compare with j-1 elements $\Rightarrow$ $t_j = j$

using $\sum_{j=1}^{n} j = \frac{n(n+1)}{2} \Rightarrow \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \Rightarrow \sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$ we have:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6 \frac{n(n-1)}{2} + c_7 \frac{n(n-1)}{2} + c_8(n-1)$$

$$= an^2 + bn + c$$ a quadratic function of n

- $T(n) = \Theta(n^2)$ order of growth in n^2

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Insertion Sort: Time Complexity …

- Advantages
  - Good running time for “almost sorted” arrays $\Theta(n)$
- Disadvantages
  - $\Theta(n^2)$ running time in worst and average case
  - $\approx n^2/2$ comparisons and exchanges

Time Complexity	
Best Case	n
Average Case	n^2
Worst Case	n^2

Insertion Sort: Summary

Time Complexity: O(n^2)
Auxiliary Space: O(1)
Boundary Cases: Insertion sort takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.
Sorting In Place: Yes
Stable: Yes
Online: Yes

Uses: Insertion sort is used when number of elements is small. It can also be useful when input array is almost sorted, only few elements are misplaced in complete big array.

Any Questions ?

Dr. Anand Singh Jalal
Professor
Email: asjalal@gla.ac.in