Algorithm analysis & design

Sorting Algorithms

Presented By:

T.A. **Asmaa Hamad El-saied**

E-mail: **eng.asmaa134@gmail.com**

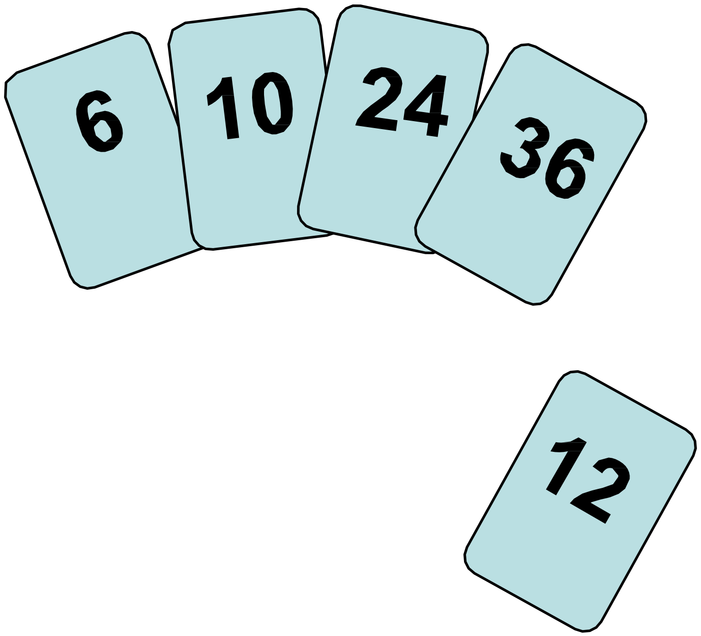
Agenda

- **Insertion Sort**
  - Algorithm
  - Analysis of Insertion Sort
- **Bubble Sort**
  - Algorithm
  - Analysis of Bubble Sort
- **Selection Sort**
  - Algorithm
  - Analysis of Selection Sort

Insertion Sort

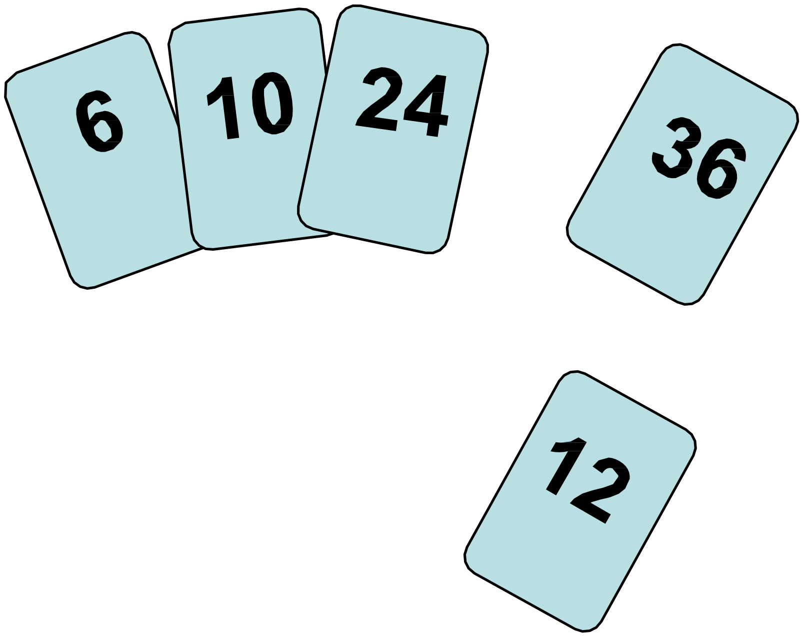
- Idea: like sorting a hand of playing cards
  - Start with an empty left hand and the cards facing down on the table.
  - Remove one card at a time from the table, and insert it into the correct position in the left hand
    - compare it with each of the cards already in the hand, from right to left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

Insertion Sort

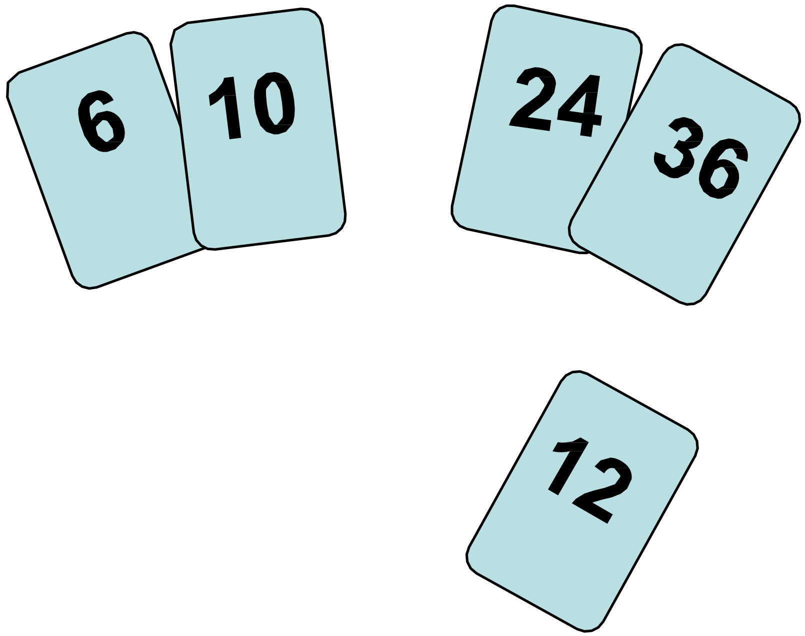


To insert 12, we need to make room for it by moving first 36 and then 24.

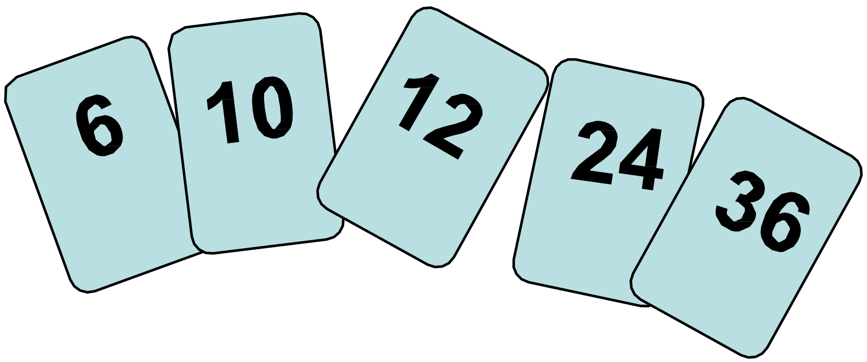
Insertion Sort



Insertion Sort



Insertion Sort



Insertion Sort: Example

input array

5 2 4 6 1 3

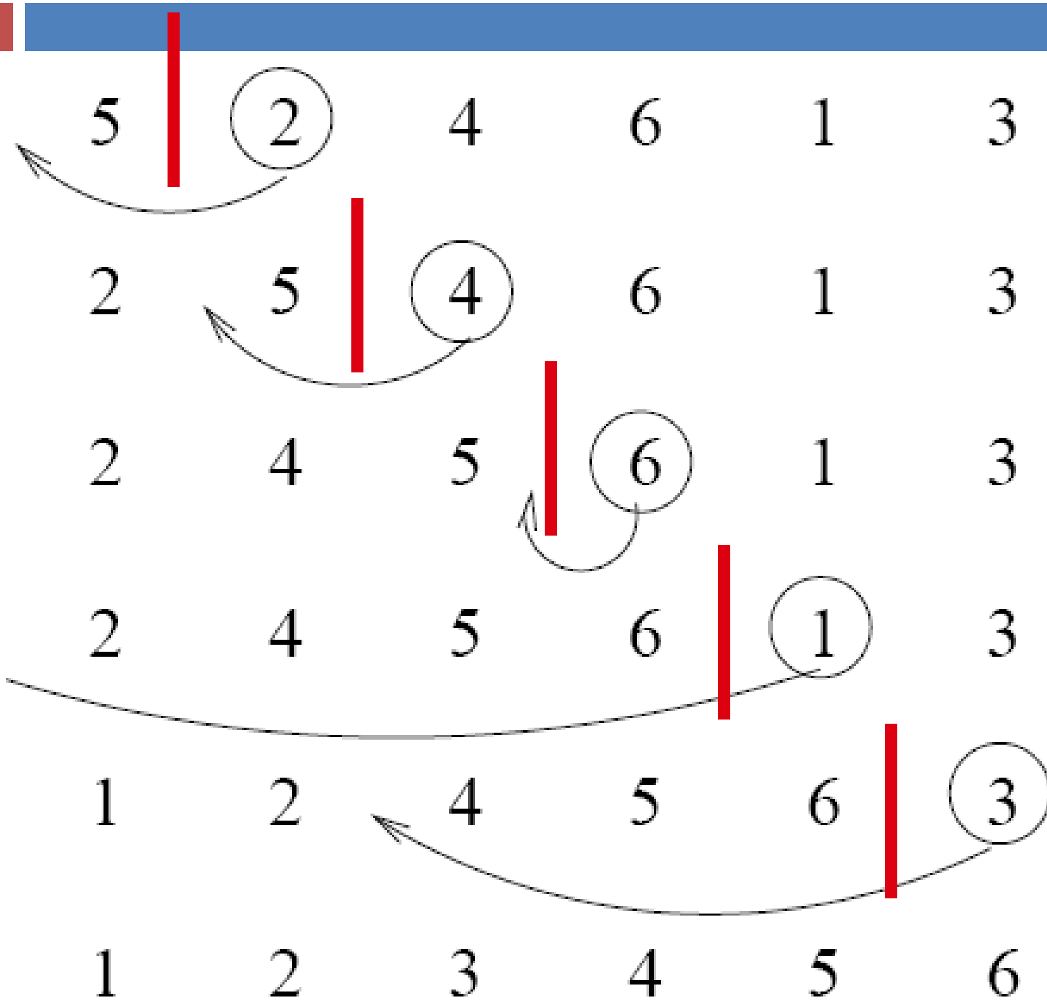
at each iteration, the array is divided in two sub-arrays:

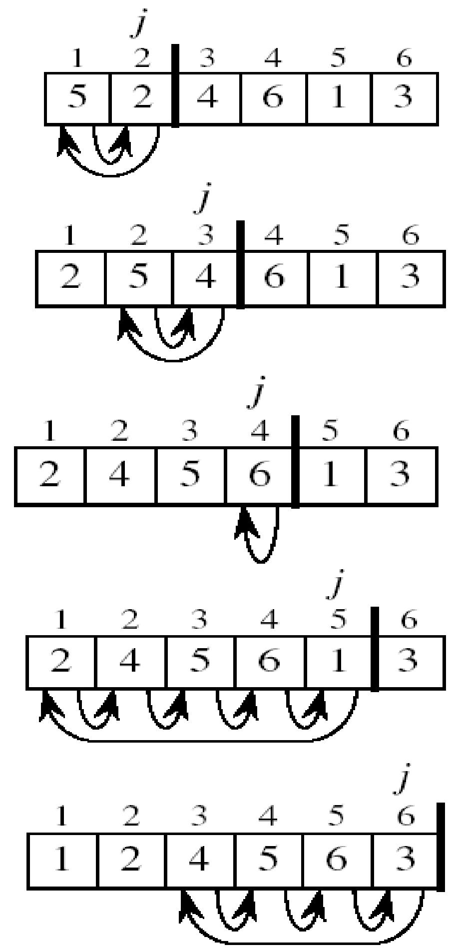
left sub-array | right sub-array

5 | 2 4 6 1 3

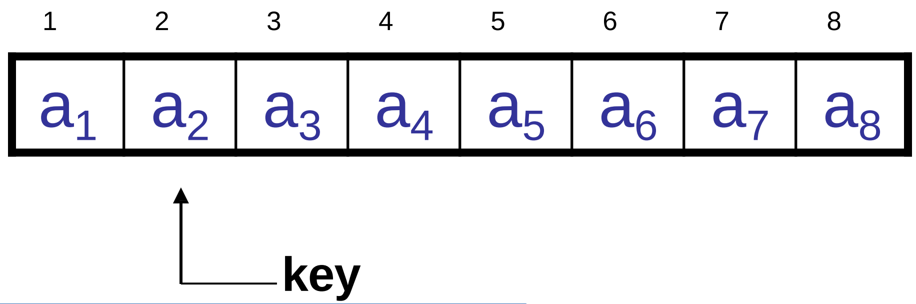
sorted | unsorted

Insertion Sort : Example





Insertion Sort: pseudo-code




```
Algorithm Insertion-sort(A,n)
    for j ← 2 to n do
        key ← A[ j ]
        i ← j - 1
        while i > =1 and A[i] > key do
            A[i + 1] ← A[i]
            i ← i – 1
        A[i + 1] ← key
```

Analysis of Insertion Sort: Running Time

Algorithm Insertion-sort*(A,n)*

	cost	times
for j ← 2 **to** n **do**	c_1	n
key ← A[j]	c_2	$n-1$
i ← j - 1	c_3	$n-1$
while i >= 1 and A[i] > key **do**	c_4	$\sum_{j=2}^{n} t_j$
A[i + 1] ← A[i]	c_5	$\sum_{j=2}^{n} (t_j - 1)$
i ← i – 1	c_6	$\sum_{j=2}^{n} (t_j - 1)$
A[i + 1] ← key	c_7	$n-1$

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

Analysis of Insertion Sort: Running Time

- The running time of the algorithm is the sum of running times for each statement executed; a statement that takes c_i steps to execute and executes n times will contribute $c_i n$ to the total running time.
- To compute *T(n)*, the running time of **INSERTION-SORT** on an input of n values, we sum the products of the *cost* and *times* columns, obtaining

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

Best Case Analysis

- The array is already sorted "**while** i > 0 and A[i] > key"
  - A[i] ≤ key upon the first time the **while** loop test is run (when $i = j - 1$)
  - $t_j = 1$
- $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$

$= (c_1 + c_2 + c_3 + c_4 + c_7)n + (c_2 + c_3 + c_4 + c_5)$

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

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 ⇒ compare with j-1 elements ⇒ $t_j = j$
  - Using $\sum_{j=1}^{N} j = \frac{(N-1)N}{2}$ $\Rightarrow$ $\sum_{j=2}^{N} j = \frac{(N-1)N}{2} - 1$ $\Rightarrow$ $\sum_{j=1}^{N} (j-1) = \frac{(N-1)N}{2}$ we have

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{(n-1)n}{2} - 1\right) + c_5\left(\frac{(n-1)n}{2}\right) + c_6\left(\frac{(n-1)n}{2}\right) + c_7(n-1)$$

$$= an^2 + bn + c$$

a quadratic function of n

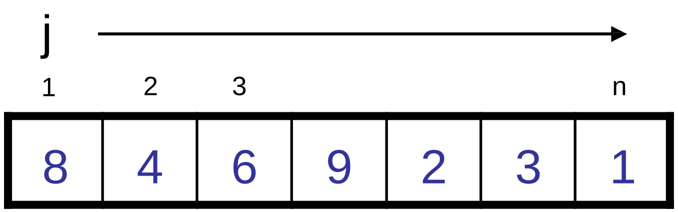
- $T(n) = \Theta(n^2)$

order of growth in n^2

Bubble sort

Bubble Sort

- Idea:
  - Repeatedly pass through the array
  - Swaps adjacent elements that are out of order



- Easier to implement, but slower than Insertion sort

Example

i= 1

8	4	6	9	2	3	1

J=1

4	8	6	9	2	3	1

J = 2

4	6	8	9	2	3	1

J = 3

4	6	8	2	9	3	1

J = 4

4	6	8	2	9	3	1

J = 5

4	6	8	2	3	9	1

J = 6

4	6	8	2	3	1	9

i=2

4	6	8	2	3	1	9

J=1

4	6	8	2	3	1	9

J=2

j

4	6	8	2	3	1	9

J=3

4	6	2	8	3	1	9

J=4

4	6	2	3	8	1	9

J=5

4	6	2	3	1	8	9

Example

i= 3

4	6	2	3	1	8	9

J=1

4	6	2	3	1	8	9

J = 2

4	2	6	3	1	8	9

J = 3

4	2	3	6	1	8	9

J = 4

4	2	3	1	6	8	9

i = 4

4	2	3	1	6	8	9

J = 1

2	4	3	1	6	8	9

J =2

2	3	4	1	6	8	9

J=3

2	3	1	4	6	8	9

i = 5

2	3	1	4	6	8	9

J=1

2	3	1	4	6	8	9

J=2

2	1	3	4	6	8	9

i= 6

2	1	3	4	6	8	9

J=1

1	2	3	4	6	8	9

J = 2

Bubble Sort : pseudo-code


```
Algorithm Bubble-sort(A, n)
    for i ← 1 to n-1 do
        for j ← 1 to n-i do
            if A[j] > A[j +1] then
                exchange A[j] ↔ A[j-1]
```

8	4	6	9	2	3	1

j = 1

Bubble-Sort Running Time

Algorithm Bubble-sort(A, n)

```
for i ← 1 to n-1 do
    for j ← 1 to n-i do
        if A[j] > A[j +1] then
            exchange A[j] ↔ A[j-1]
```

cost	*Times*
c_1	n
c_2	$\sum_{i=1}^{n}(n-i+1)$
c_3	$\sum_{i=1}^{n}(n-i)$
c_4	$\sum_{i=1}^{n}(n-i)$

$$T(n) = c_1(n+1) + c_2 \sum_{i=1}^{n}(n-i+1) + c_3 \sum_{i=1}^{n}(n-i) + c_4 \sum_{i=1}^{n}(n-i)$$

$$= \Theta(n) + (c_2 + c_3 + c_4) \sum_{i=1}^{n}(n-i)$$

where $\sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i = n^2 - \frac{n(n+1)}{2} = \frac{n^2}{2} - \frac{n}{2}$

Thus, $T(n) = \Theta(n^2)$

Best Case Analysis

- The array is already sorted

$$T(n) = c_1(n+1) + c_2 \sum_{i=1}^{n}(n-i+1) + c_3 \sum_{i=1}^{n}(n-i)$$

$$= \Theta(n) + (c_2 + c_3) \sum_{i=1}^{n}(n-i)$$

$$where \quad \sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i = n^2 - \frac{n(n+1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

Thus, $T(n) = \Theta(n^2)$

Worst Case Analysis

- The array is in reverse sorted order

$$T(n) = c_1(n+1) + c_2 \sum_{i=1}^{n}(n-i+1) + c_3 \sum_{i=1}^{n}(n-i) + c_4 \sum_{i=1}^{n}(n-i)$$

$$= \Theta(n) + (c_2 + c_3 + c_4) \sum_{i=1}^{n}(n-i)$$

$$where \quad \sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i = n^2 - \frac{n(n+1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

Thus, $T(n) = \Theta(n^2)$

Analysis of Bubble-Sort

- In any cases, (worse case, best case or average case) to sort the list in ascending order the number of comparisons between elements is the same.
- Best case : $O(n^2)$
- Average case: $O(n^2)$
- Worst case: $O(n^2)$
- How to optimize bubble sort in case sorted array ?

Selection sort

Selection Sort

- Idea:
  - Find the smallest element in the array
  - Exchange it with the element in the first position
  - Find the second smallest element and exchange it with the element in the second position
  - Continue until the array is sorted

Example

8	4	6	9	2	3	1

1	4	6	9	2	3	8

1	2	6	9	4	3	8

1	2	3	9	4	6	8

1	2	3	4	9	6	8

1	2	3	4	6	9	8

1	2	3	4	6	8	9

1	2	3	4	6	8	9

Selection Sort

8	4	6	9	2	3	1

Algorithm Selection-sort*(A,n)*

```
for j ← 1 to n - 1 do
    smallest ← j
        for i ← j + 1 to n do
            if A[i] < A[smallest]
                then smallest ← i
        exchange A[j] ↔ A[smallest]
```

Analysis of Selection Sort: running time

	cost	times
Algorithm Selection-sort*(A,n)*		
for j ← 1 to n - 1 do	c_1	n
smallest ← j	c_2	n-1
for i ← j + 1 to n do	c_3	$\sum_{j=1}^{n-1}(n-j+1)$
if *A*[i] < *A*[smallest]	c_4	$\sum_{j=1}^{n-1}(n-j)$
then smallest ← i	c_5	$\sum_{j=1}^{n-1}(n-j)$
exchange *A*[j] ↔ *A*[smallest]	c_6	n-1

$$T(n) = c_1\, n + c_2\,(n-1) + c_3 \sum_{j=1}^{n-1}(n-j+1) + c_4 \sum_{j=1}^{n-1}(n-j) + c_5 \sum_{j=1}^{n-1}(n-j) + c_6\,(n-1) = \Theta(n^2)$$

Best Case Analysis

$$T(n) = c_1 n + c_2 (n-1) + c_3 \sum_{j=1}^{n-1}(n-j+1) + c_4 \sum_{j=1}^{n-1}(n-j) + c_6 (n-1) = \Theta(n^2)$$

Worst Case Analysis

$$T(n) = c_1 n + c_2 (n-1) + c_3 \sum_{j=1}^{n-1}(n-j+1) + c_4 \sum_{j=1}^{n-1}(n-j) + c_5 \sum_{j=1}^{n-1}(n-j) + c_6 (n-1) =$$

where $\sum_{j=1}^{n-1}(n-j) = \sum_{j=1}^{n-1} n - \sum_{j=1}^{n-1} j = n^2 - \frac{n(n+1)}{2} = \frac{n^2}{2} - \frac{n}{2}$

Thus, $T(n) = \Theta(n^2)$

Summary

- **Bubble sort and Insertion sort –**

  Average and worst case time complexity: n^2

  Best case time complexity: n when array is already sorted.

  Worst case: when the array is reverse sorted.

- **Selection sort –**

  Best, average and worst case time complexity: n^2 which is independent of distribution of data.

Thanks