Algorithm Execution
Insertion Sort

Ira A. Fulton Schools of Engineering
Arizona State University

Algorithm Execution

- We will study the execution of an algorithm on a given instance and predict the execution process.

- We will use Insertion Sort as an example.

Sorting

- Sorting is the process of arranging a sequence of objects into order (either increasing or decreasing)
- There are many sorting algorithms
- Different sorting algorithms may have different time complexities.
- The problem also has a complexity.
- There is a difference between problem complexity and algorithm complexity.

Insertion Sort

```
Insertion-Sort(A, n)
for i := 2 to n do
    key := A[i]
    // insert A[i] into sorted A[1:i-1]
    j := i -1
    while j > 0 and A[j] > key do
        A[j+1] := A[j]
        j := j - 1
    A[j+1] := key
```

Trace the algorithm using inputs: 3 5 2 8 3

Walk through example

A	3	5	2	8	3		
	1	2	3	4	5	6	7

n=5

Walk through example

A	3	5	2	8	3		
	1	2	3	4	5	6	7

n=5

i=2

key=5

Walk through example

A	3	5	2	8	3		
	1	2	3	4	5	6	7

n=5

i=2

key=5

j=1

Walk through example

A	3	5	2	8	3		
	1	2	3	4	5	6	7

n=5

i=2

key=5

j=1

j > 0?

Yes

Walk through example

A	3	5	2	8	3		
	1	2	3	4	5	6	7

n=5

i=2

key=5

j=1

j > 0?	Yes
A[j]>key?	No

Walk through example

A	3	5	2	8	3		
	1	2	3	4	5	6	7

n=5

i=2

key=5

j=1

j > 0?	Yes
A[j]>key?	No

A[j+1] := key

Walk through example

A	3	5	2	8	3		
	1	2	3	4	5	6	7

n=5

i=2

key=5

j=1

j > 0?

Yes

A[j]>key?

No

A[j+1] := key

Array A is overwritten.

Walk through example

A	**3**	**5**	2	8	3		
	1	2	3	4	5	6	7

n=5

i=2

key=5

Walk through example

A	3	5	2	8	3		
	1	2	3	4	5	6	7

n=5

i=3

key=2

Walk through example

A	3	5	2	8	3		
	1	2	3	4	5	6	7

n=5

i=3

key=2

j=2

Walk through example

A	3	5	2	8	3		
	1	2	3	4	5	6	7

n=5

i=3

key=2

j=2

j > 0?

Yes

Walk through example

A	3	5	2	8	3		
	1	2	3	4	5	6	7

n=5

i=3

key=2

j=2

j > 0?	Yes
A[j]>key?	Yes

Walk through example

A							
	3	5	5	8	3		
	1	2	3	4	5	6	7

n=5

i=3

key=2

j=2

j > 0?

Yes

A[j]>key?

Yes

A[j+1] := A[j]

Array A is overwritten.

Walk through example

A	3	5	5	8	3		
	1	2	3	4	5	6	7

n=5

i=3

key=2

j=1

Walk through example

A	3	5	5	8	3		
	1	2	3	4	5	6	7

n=5

i=3

key=2

j=1

j > 0?

Yes

Walk through example

A	3	5	5	8	3		
	1	2	3	4	5	6	7

n=5

i=3

key=2

j=1

j > 0?	Yes
A[j]>key?	Yes

Walk through example

A	3	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=3

key=2

j=1

j > 0?

Yes

A[j]>key?

Yes

A[j+1] := A[j]

Array A is overwritten.

Walk through example

A	3	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=3

key=2

j=0

Walk through example

A	3	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=3

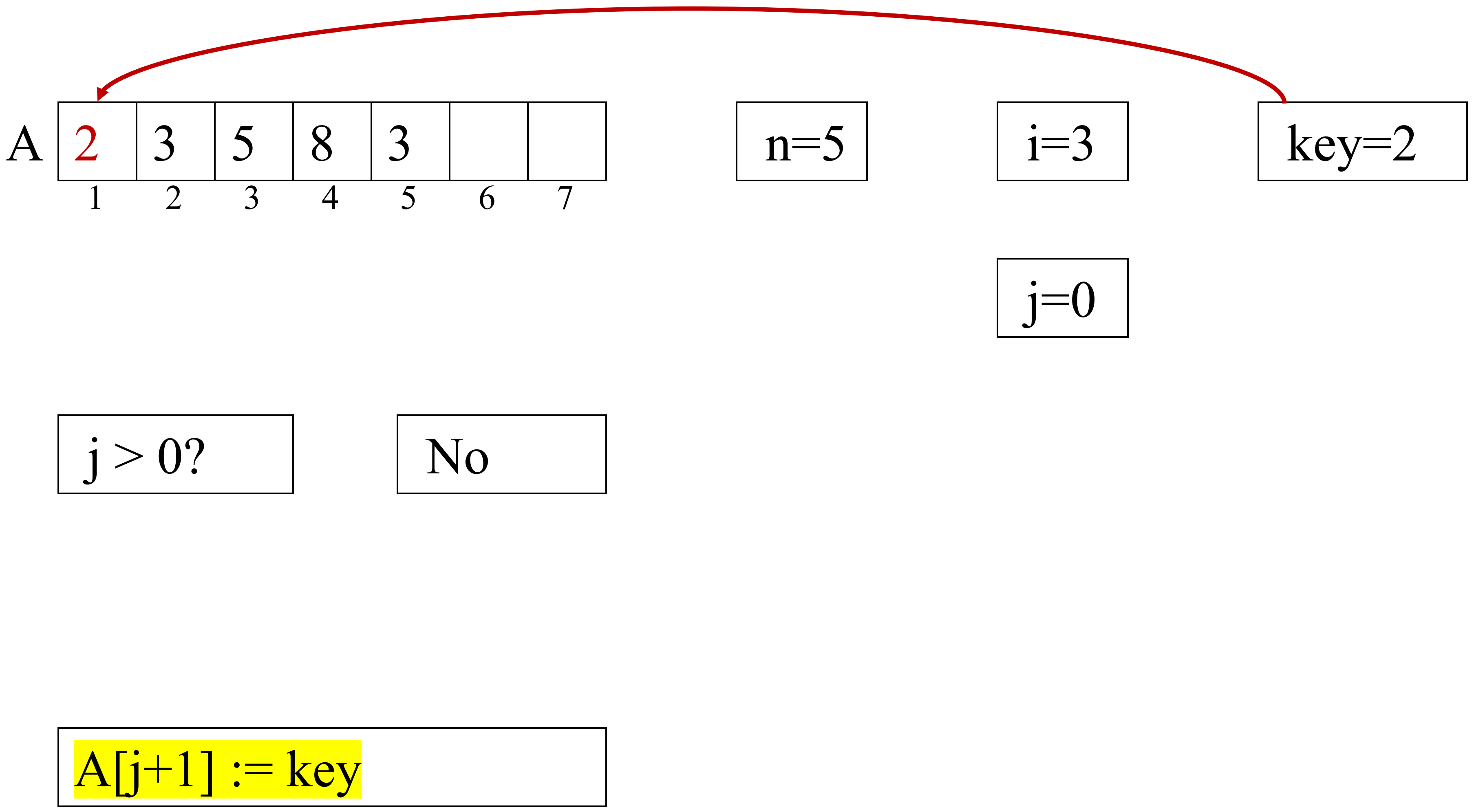
key=2

j=0

j > 0?

No

Walk through example



Array A is overwritten.

Walk through example

A	**2**	**3**	**5**	8	3		
	1	2	3	4	5	6	7

n=5

i=3

key=2

Walk through example

A	2	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=4

key=8

Walk through example

A	2	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=4

key=8

j=3

Walk through example

A	2	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=4

key=8

j=3

j > 0?

Yes

Walk through example

A	2	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=4

key=8

j=3

j > 0?

Yes

A[j]>key?

No

Walk through example

A	2	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=4

key=8

j=3

j > 0? — Yes

A[j]>key? — No

A[j+1] := key

Array A is overwritten.

Walk through example

A	**2**	**3**	**5**	**8**	3		
	1	2	3	4	5	6	7

n=5

i=4

key=8

Walk through example

A	2	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=5

key=3

Walk through example

A	2	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=4

Walk through example

A	2	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=4

j > 0?

Yes

Walk through example

A	2	3	5	8	3		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=4

j > 0?	Yes
A[j]>key?	Yes

Walk through example

A	2	3	5	8	8		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=4

j > 0? — Yes

A[j]>key? — Yes

A[j+1] := A[j]

Array A is overwritten.

Walk through example

A	2	3	5	8	8		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=3

Walk through example

A	2	3	5	8	8		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=3

j > 0?

Yes

Walk through example

A	2	3	5	8	8		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=3

j > 0?	Yes
A[j]>key?	Yes

Walk through example

A	2	3	5	5	8		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=3

j > 0? | Yes

A[j]>key? | Yes

A[j+1] := A[j]

Array A is overwritten.

Walk through example

A	2	3	5	5	8		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=2

Walk through example

A	2	3	5	5	8		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=2

j > 0?

Yes

Walk through example

A	2	3	5	5	8		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=2

j > 0?	Yes
A[j]>key?	No

Walk through example

A	2	3	3	5	8		
	1	2	3	4	5	6	7

n=5

i=5

key=3

j=2

j > 0?	Yes
A[j]>key?	No

A[j+1] := key

Array A is overwritten.

Walk through example

A	**2**	**3**	**3**	**5**	**8**		
	1	2	3	4	5	6	7

n=5

i=5

key=3

Walk through example

A	**2**	**3**	**3**	**5**	**8**		
	1	2	3	4	5	6	7

n=5

i=6

Invariance of Insertion Sort

- **Before the for-loop with i=k, A[1:k-1] are in sorted order**
- **A[i] is copied to the variable key**
- **The for-loop with i=k inserts key into A[1..k-1].**
- **Each element in A[1..k-1] that is larger than key is copied to its right neighbor, from right to left.**
- **key is inserted into the correct position.**
- **After the for-loop with i=k, A[1:k] are in sorted order**

Running Time

- Number of time steps required for the algorithm to terminate
- In terms of the **size of the input** to the algorithm, which is related to the amount of work to be done. E.g., sorting 1000 integers takes longer than sorting 3 integers.
- running time = function(input size)

Running Time

Insertion-Sort(A[1..n])	times
for i := 2 to n **do**	n
key := A[i]	n-1
j := i -1	n-1
while j > 0 and A[j] > key **do**	$\sum_{i=2}^{n}(t_i+1)$
A[j+1] := A[j]	$\sum_{i=2}^{n} t_i$
j := j- 1	$\sum_{i=2}^{n} t_i$
A[j+1] := key	n-1

where t_i is the number of elements in A[1..i-1] that is larger than A[i]

Running time of Insertion Sort

Let $T(n, A)$ denote the number of operations required for insertion sort to sort array $A[1:n]$. From the above analysis, we have

$$T(n, A) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n}(t_i + 1) + (c_6 + c_7)\sum_{i=2}^{n} t_i + c_8(n-1) \quad (1)$$

$$= (c_5 + c_6 + c_7)\sum_{i=2}^{n} t_i + (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8). \quad (2)$$

Since $0 \leq t_i \leq i - 1, i = 2, 3, \ldots, n$, we have

$$T(n, A) \geq (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8), \quad (3)$$

$$T(n, A) \leq (c_5 + c_6 + c_7)\sum_{i=2}^{n}(i-1) + (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \quad (4)$$

$$= (c_5 + c_6 + c_7)\sum_{i=1}^{n} i + (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \quad (5)$$

$$= (c_5 + c_6 + c_7)\frac{(n-1)n}{2} + (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8). \quad (6)$$

From (3), we get $T(n, A) \in \Omega(n)$. From (4)-(6), we get $T(n, A) \in O(n^2)$.

Running time of Insertion Sort

The running time of Insertion Sort for sorting an n-element array A is $\Theta(n + \sum_{i=2}^{n} t_i)$, where t_i is the number of elements in $A[1 : n - 1]$ that are larger than $A[i]$, $i = 2, 3, \ldots, n$. $\sum_{i=2}^{n} t_i$ is also known as the **inversion number** of array A.

Without knowing the content of array A, but know the number of elements n, the **Best-case** running time of Insertion Sort is $O(n)$. The **worst-case** running time of Insertion Sort is $O(n^2)$. When array is reversely sorted, the running time of insertion sort if $\Theta(n^2)$. We can prove that the **Average-case** running time of Insertion Sort is $O(n^2)$.

Inversion Number of A Sequence

- Let A[1..n] be a sequence of n numbers.
- For 1 <= i< j <= n, we say <i, j> is an inversion if A[i]>A[j]. Note that <i, j> is the inversion, not <A[i], A[j]>.
- The inversion number of A is the total number of inversions for the sequence.
- The inversion number of 3 5 2 8 3 is 4
- The inversion number of 2 3 3 5 8 is 0

In-class example

- The inversion number of 5 4 3 2 1
- <1, 2>, <1, 3>, <1, 4>, <1, 5>
- <2, 3>, <2, 4>, <2, 5>

- 4 + 3 + 2 + 1 = 10

- (n-1)n/2

Inversion Number of A Sequence

- How many times are we copying elements to the right while applying insertion sort on the input sequence 3 5 2 8 3?
- How many times are we copying elements to the right while applying insertion sort on the input sequence 5 4 3 2 1?
- The running time of insertion sort is Θ(n + Inversion(A)), where Inversion(A) is the inversion number of A.

Summary

- An algorithm is a well-defined step by step computational procedure.
- We studied Insertion Sort, and analyzed its best case time complexity and worst-case time complexity.
- Inversion number and its relationship with the time complexity of insertion sort.