Insertion Sort

Shusen Wang

Problem of Sorting

Input:

8	1	4	9	0	6	5	2	7	3

- **Input:** An (unordered) array containing n elements.
- **Goal:** Sort the n elements in ascending order.

Problem of Sorting

Input:

8	1	4	9	0	6	5	2	7	3

- **Input:** An (unordered) array containing n elements.
- **Goal:** Sort the n elements in ascending order.

Output:

0	1	2	3	4	5	6	7	8	9

Insertion Sort

Initial State

8	1	4	9	0	6	5	2	7	3

Iteration 1

8	**1**	4	9	0	6	5	2	7	3

Iteration 1

8	1	4	9	0	6	5	2	7	3

Iteration 1

8		4	9	0	6	5	2	7	3

Key: 1

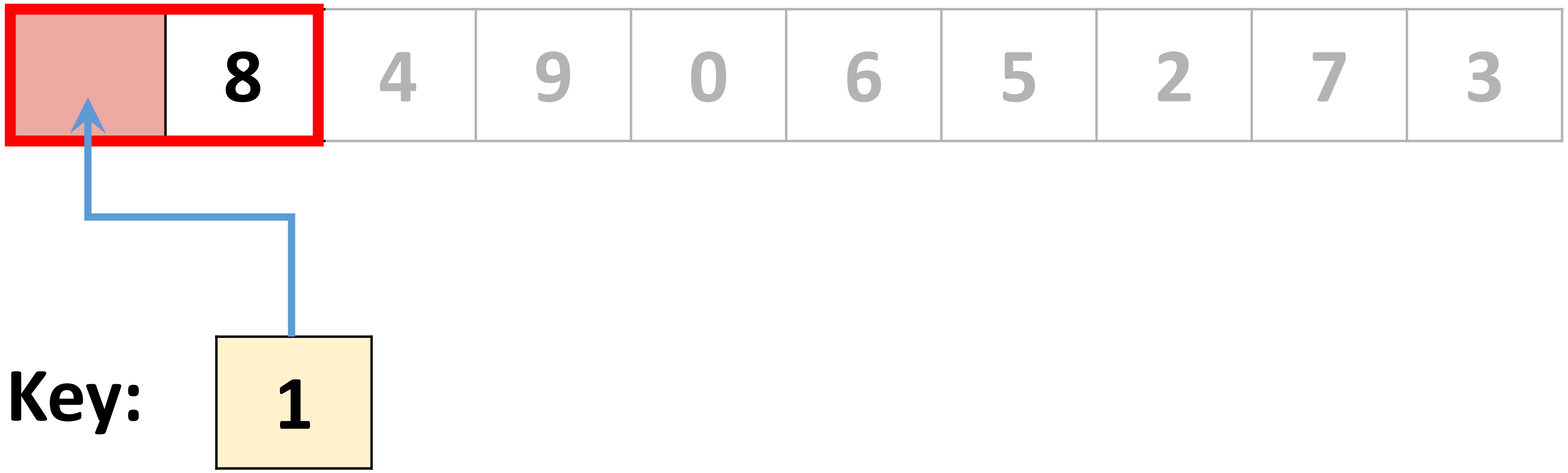
Iteration 1

8		4	9	0	6	5	2	7	3

$>$ key

Key: 1

Iteration 1



Iteration 1

1	8	4	9	0	6	5	2	7	3

Key:

After Iteration 1

1	8	4	9	0	6	5	2	7	3

Key:

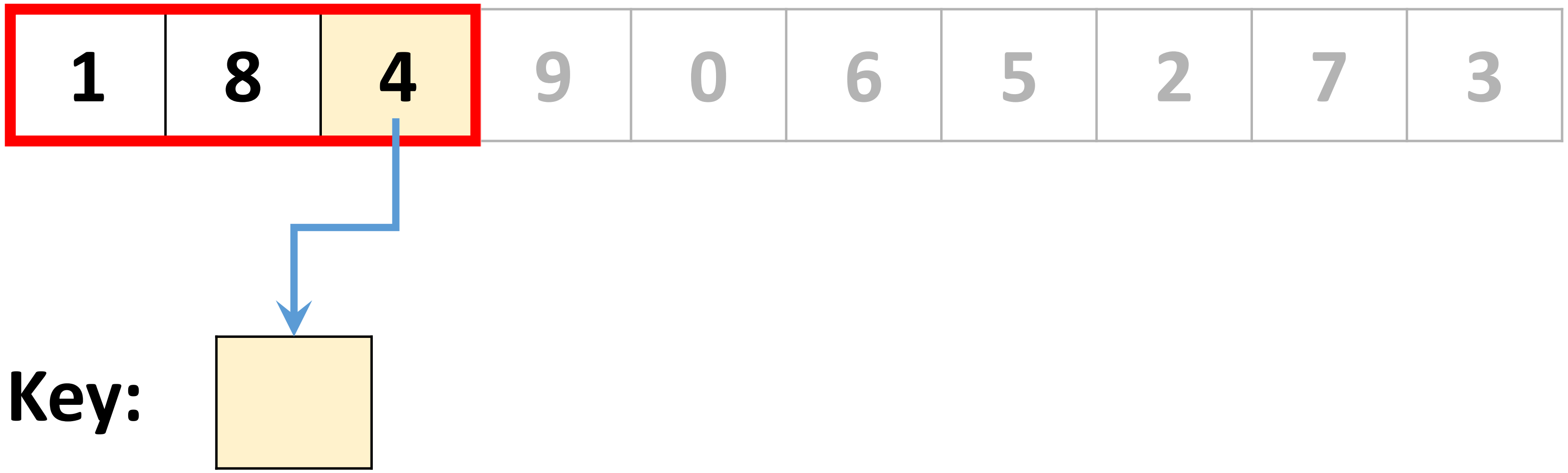
After Iteration 1, the first 2 elements are in ascending order.

Iteration 2

1	8	4	9	0	6	5	2	7	3

Key:

Iteration 2



Iteration 2

1	8		9	0	6	5	2	7	3

Key: 4

Iteration 2

1	8		9	0	6	5	2	7	3
	> key								

Key: 4

Iteration 2

1		8	9	0	6	5	2	7	3

Key: 4

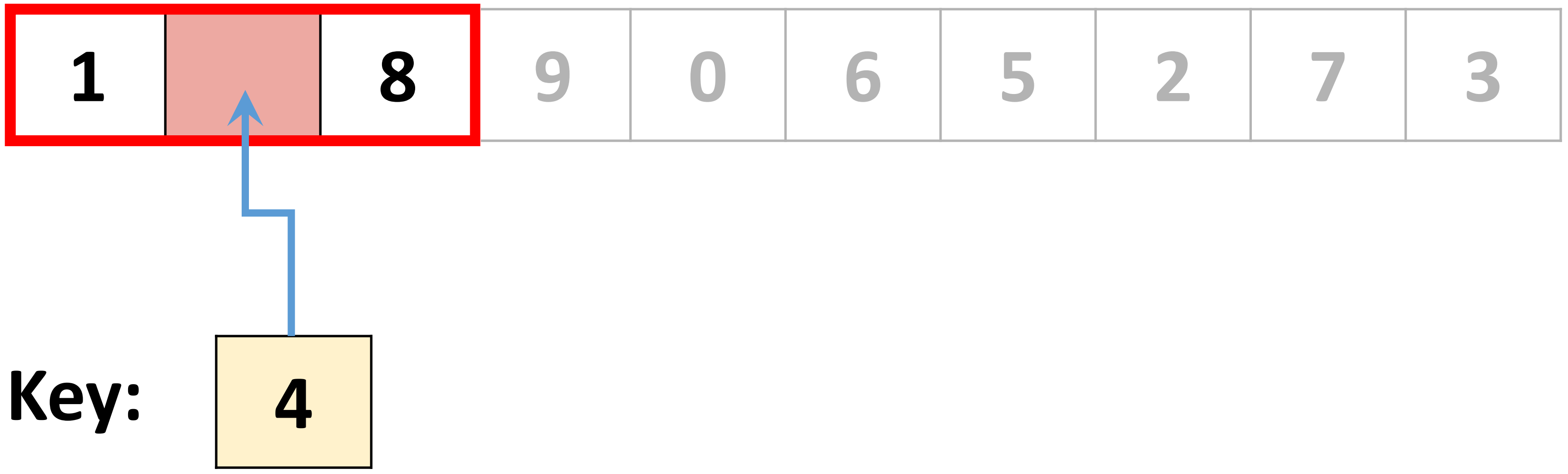
Iteration 2

1		8	9	0	6	5	2	7	3

< key

Key: 4

Iteration 2



Iteration 2

1	4	8	9	0	6	5	2	7	3

Key:

After Iteration 2

1	4	8	9	0	6	5	2	7	3

Key:

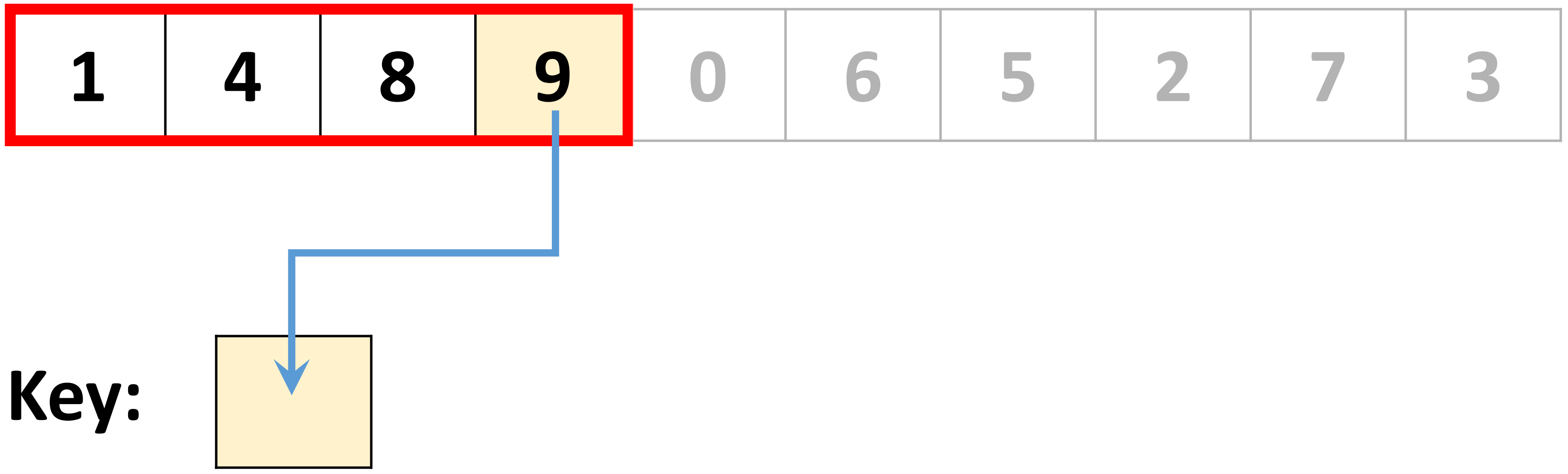
After Iteration 2, the first 3 elements are in ascending order.

Iteration 3

1	4	8	9	0	6	5	2	7	3

Key:

Iteration 3



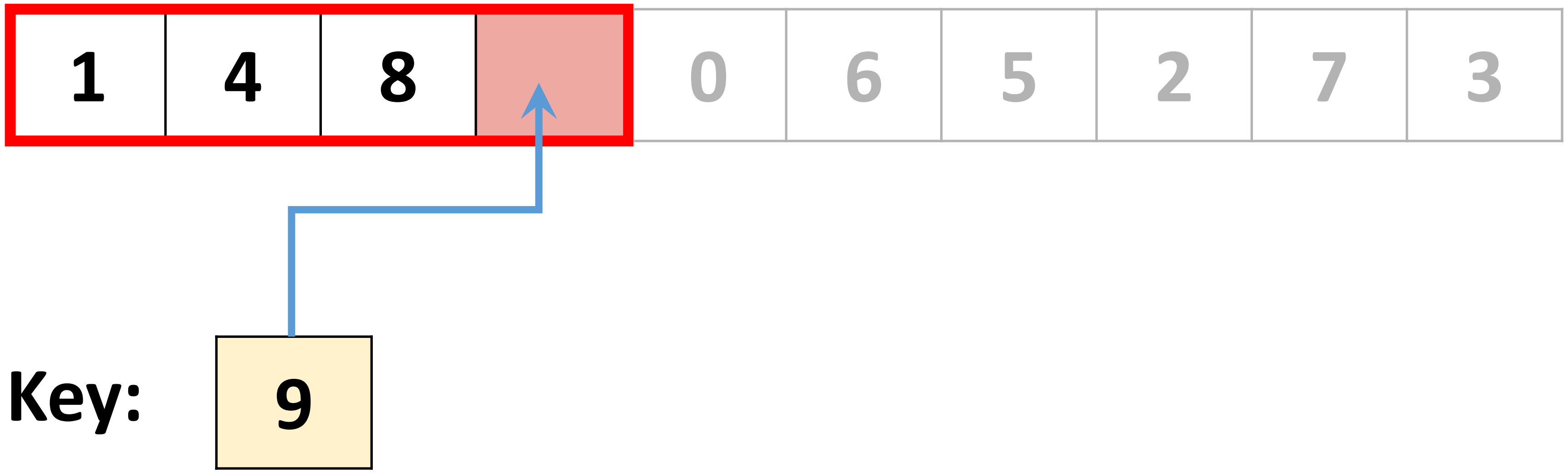
Iteration 3

1	4	8		0	6	5	2	7	3

Key: 9

Iteration 3

Iteration 3



Iteration 3

1	4	8	9	0	6	5	2	7	3

Key:

After Iteration 3

1	4	8	9	0	6	5	2	7	3

Key:

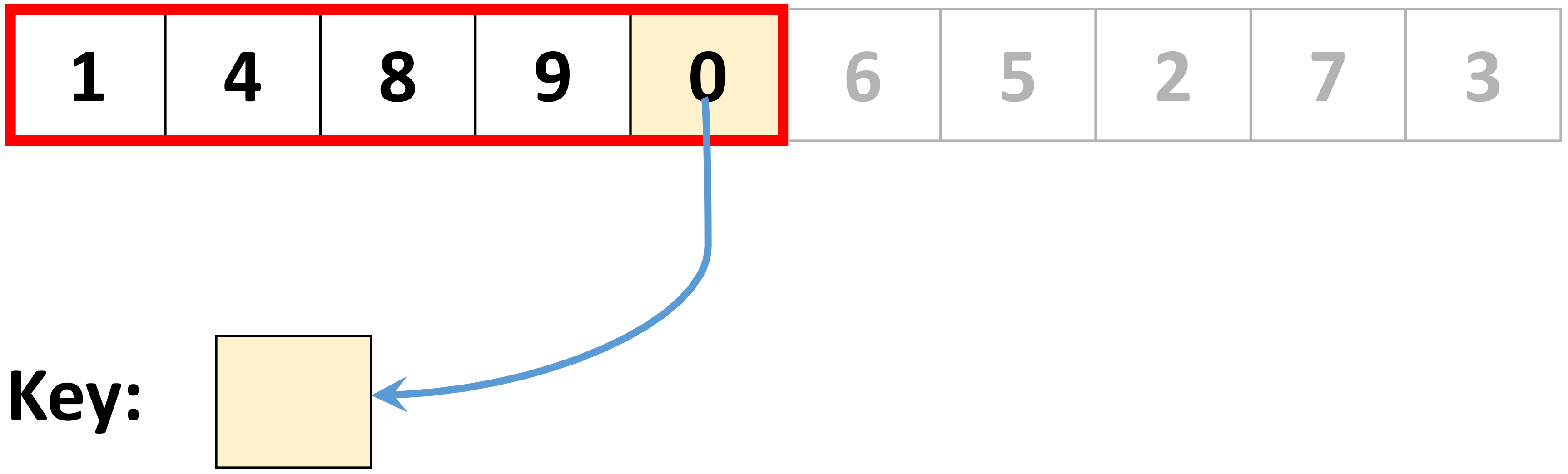
After Iteration 3, the first 4 elements are in ascending order.

Iteration 4

1	4	8	9	0	6	5	2	7	3

Key:

Iteration 4

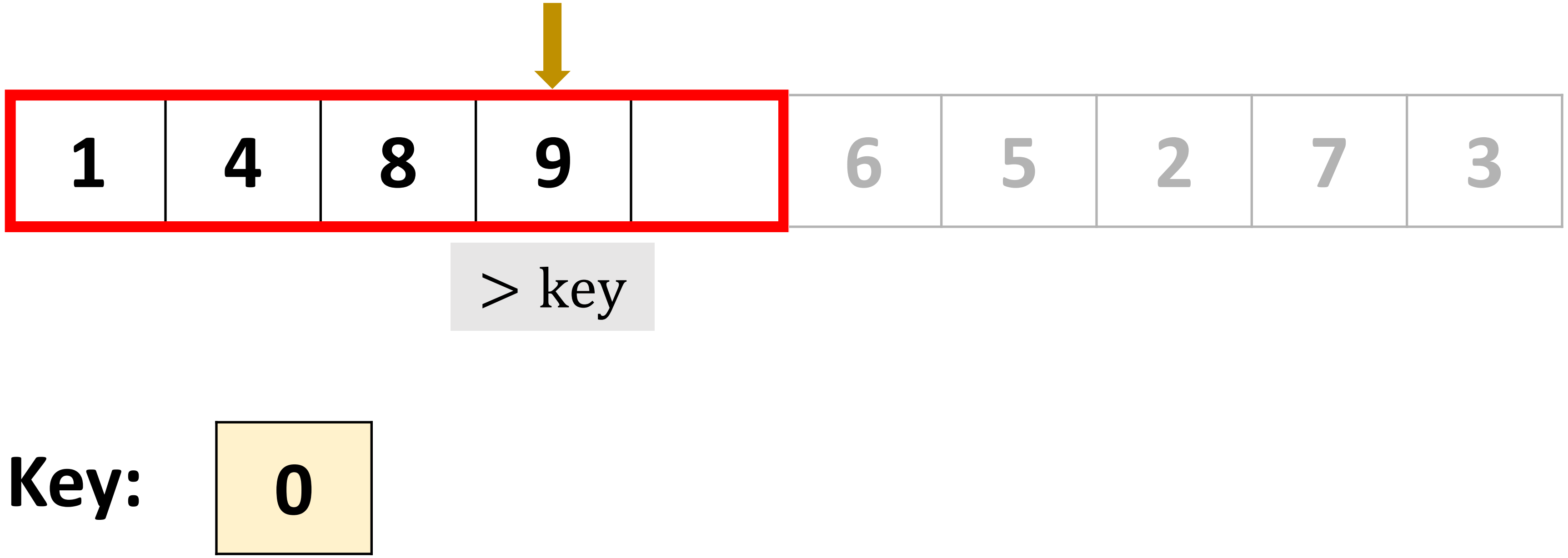


Iteration 4

1	4	8	9		6	5	2	7	3

Key: 0

Iteration 4

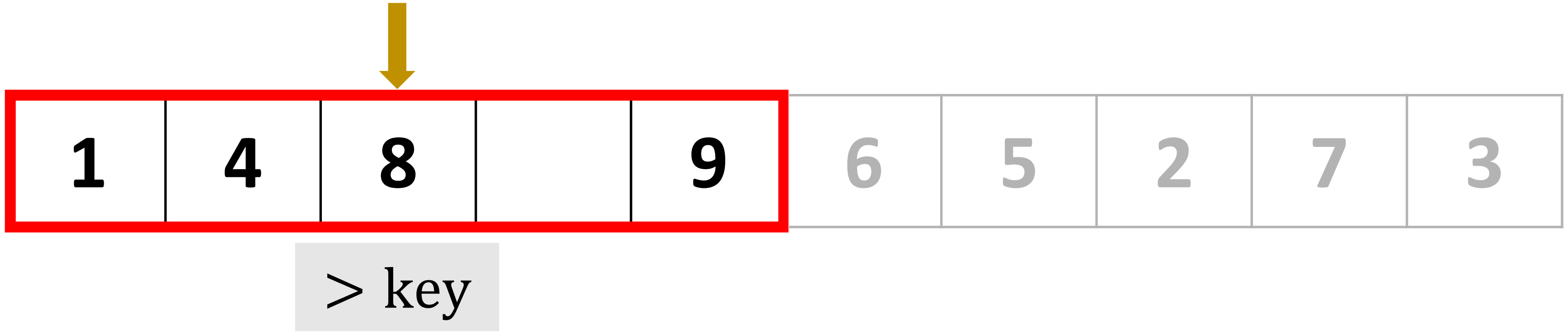


Iteration 4

1	4	8	9		6	5	2	7	3

Key: 0

Iteration 4



Key: 0

Iteration 4

1	4	8		9	6	5	2	7	3

Key: 0

Iteration 4

Iteration 4

1	4		8	9	6	5	2	7	3

Key: 0

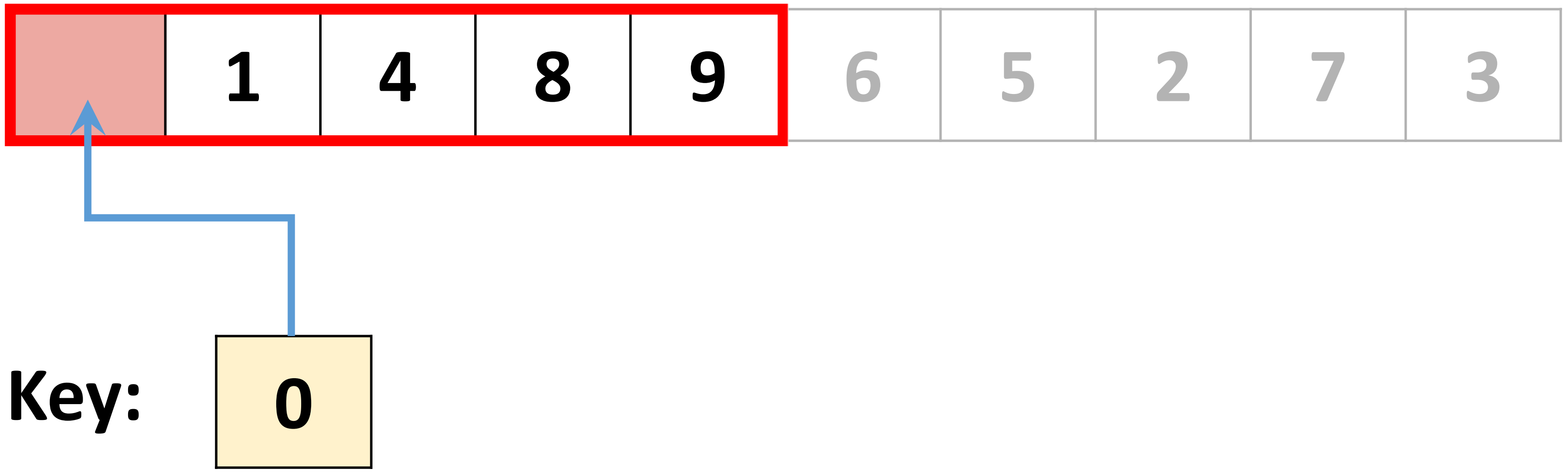
Iteration 4

Iteration 4

1		4	8	9	6	5	2	7	3

Key: 0

Iteration 4



Iteration 4

0	1	4	8	9	6	5	2	7	3

Key:

After Iteration 4

0	1	4	8	9	6	5	2	7	3

Key:

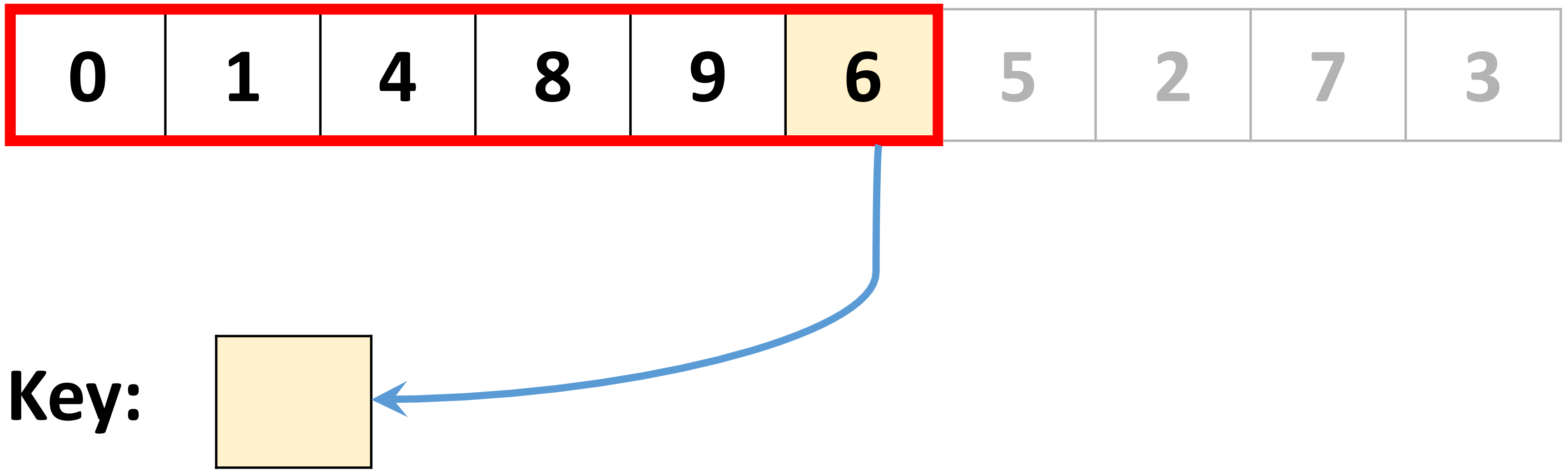
After Iteration 4, the first 5 elements are in ascending order.

Iteration 5

0	1	4	8	9	6	5	2	7	3

Key:

Iteration 5

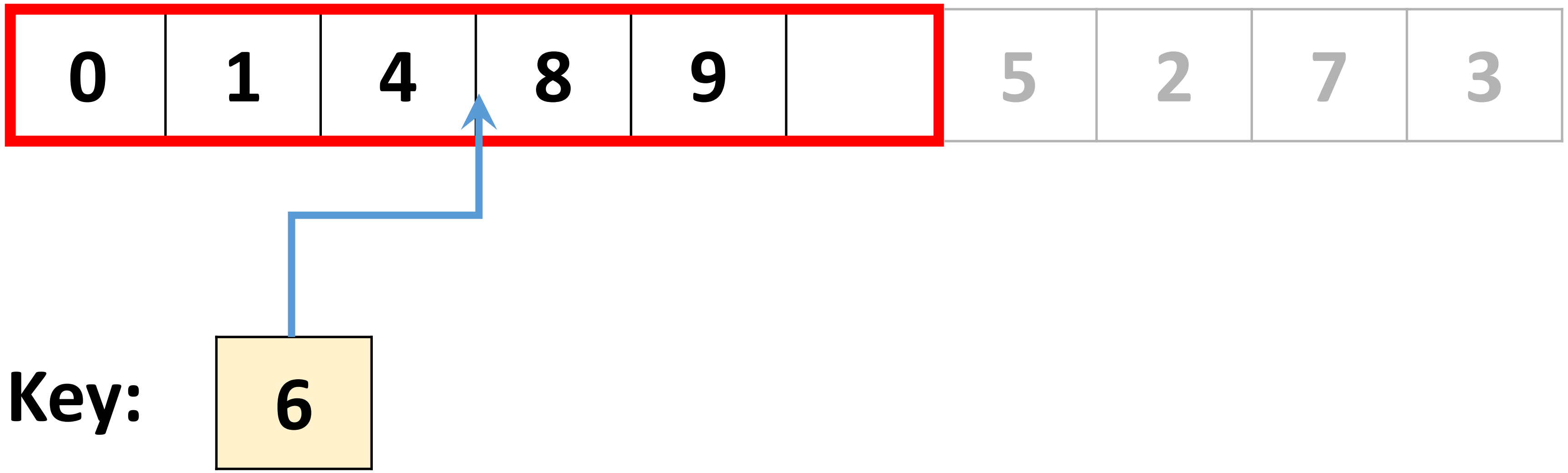


Iteration 5

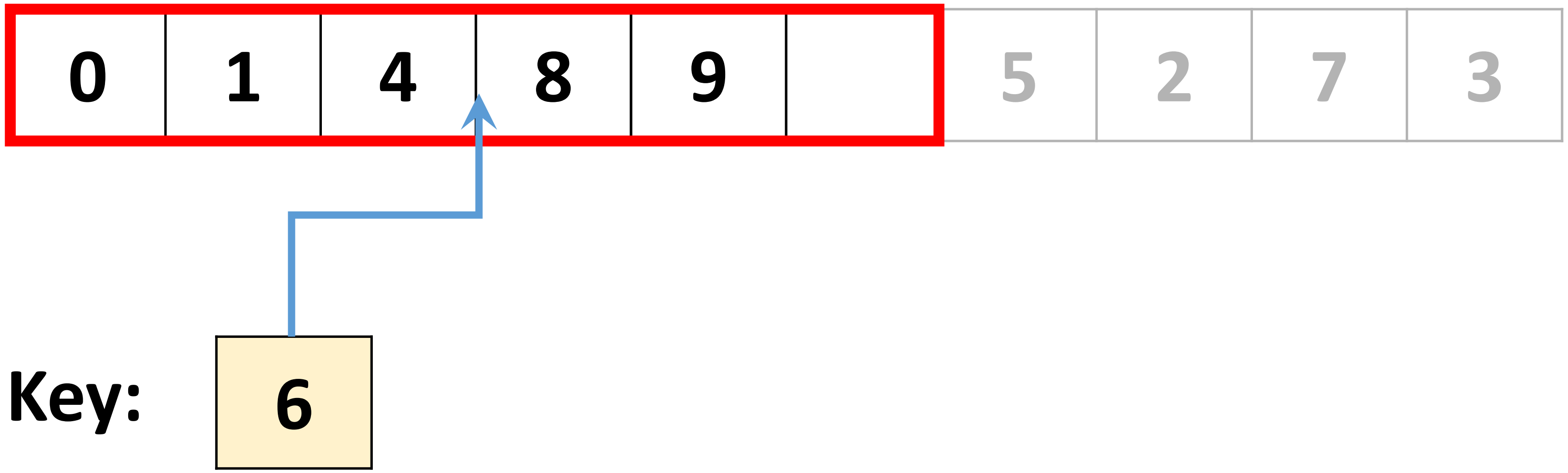
0	1	4	8	9		5	2	7	3

Key: 6

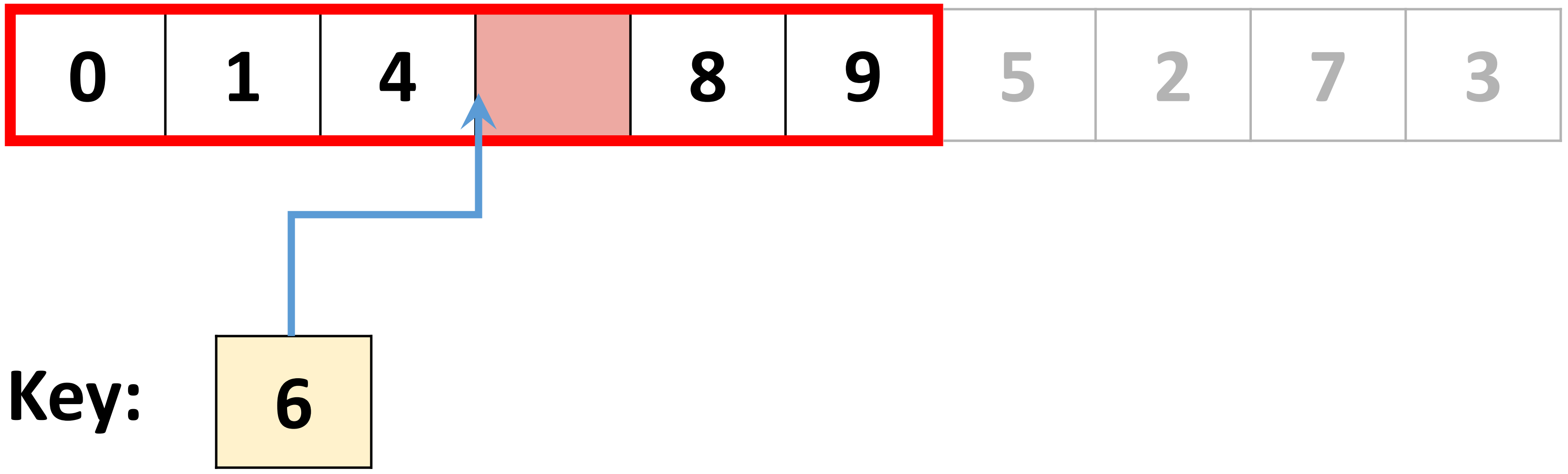
Iteration 5



Iteration 5



Iteration 5



Iteration 5

0	1	4	6	8	9	5	2	7	3

Key:

After Iteration 5

0	1	4	6	8	9	5	2	7	3

Key:

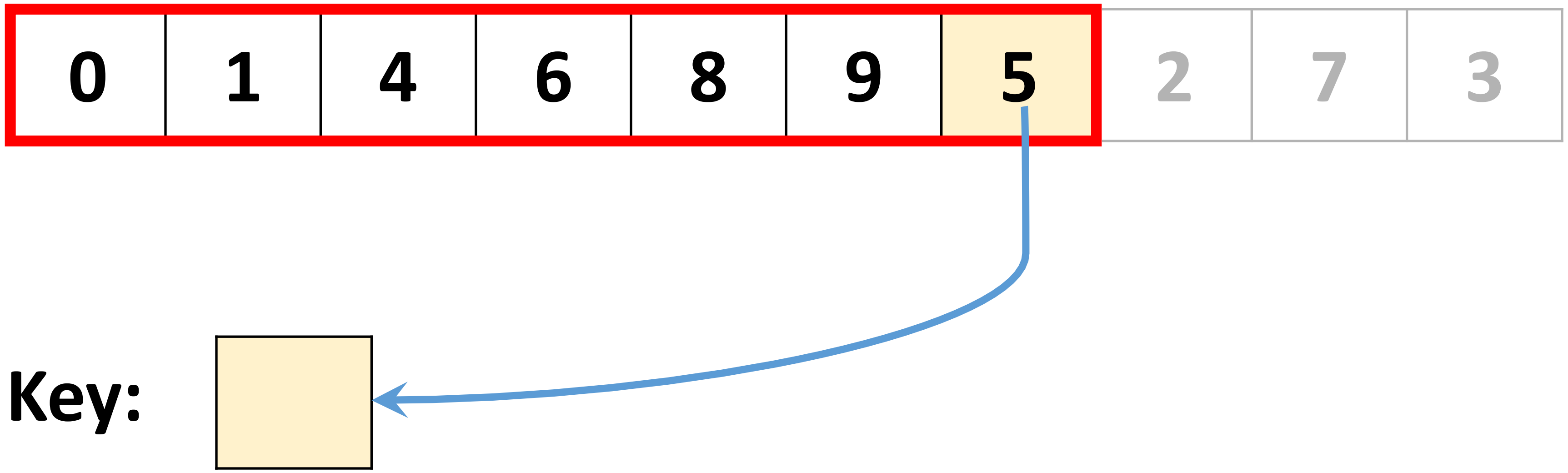
After Iteration 5, the first 6 elements are in ascending order.

Iteration 6

0	1	4	6	8	9	5	2	7	3

Key:

Iteration 6

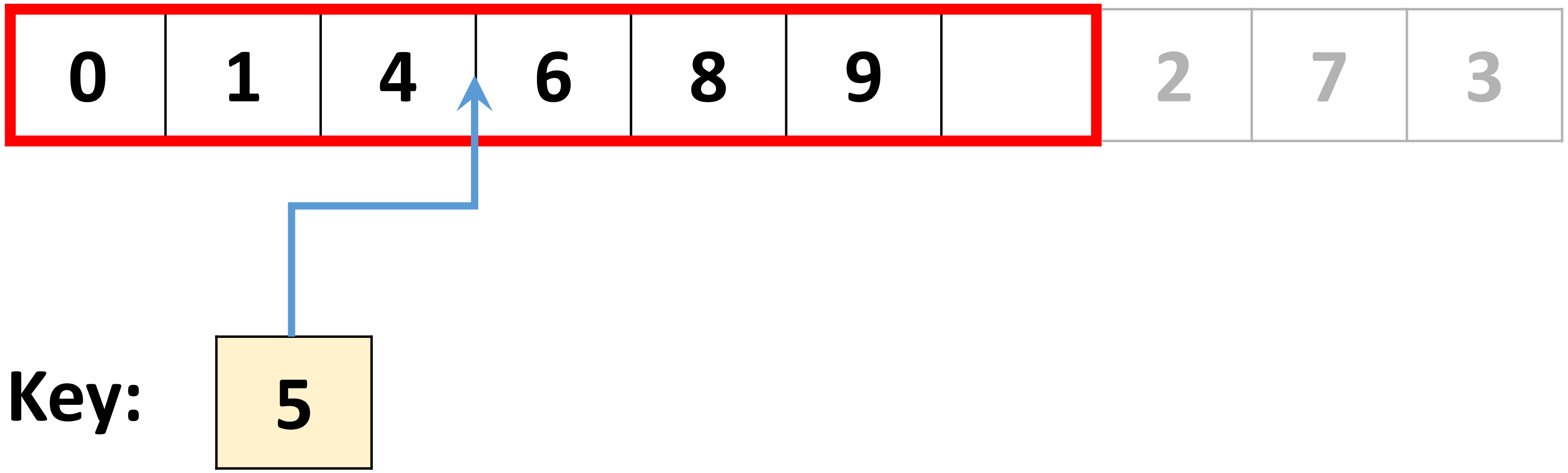


Iteration 6

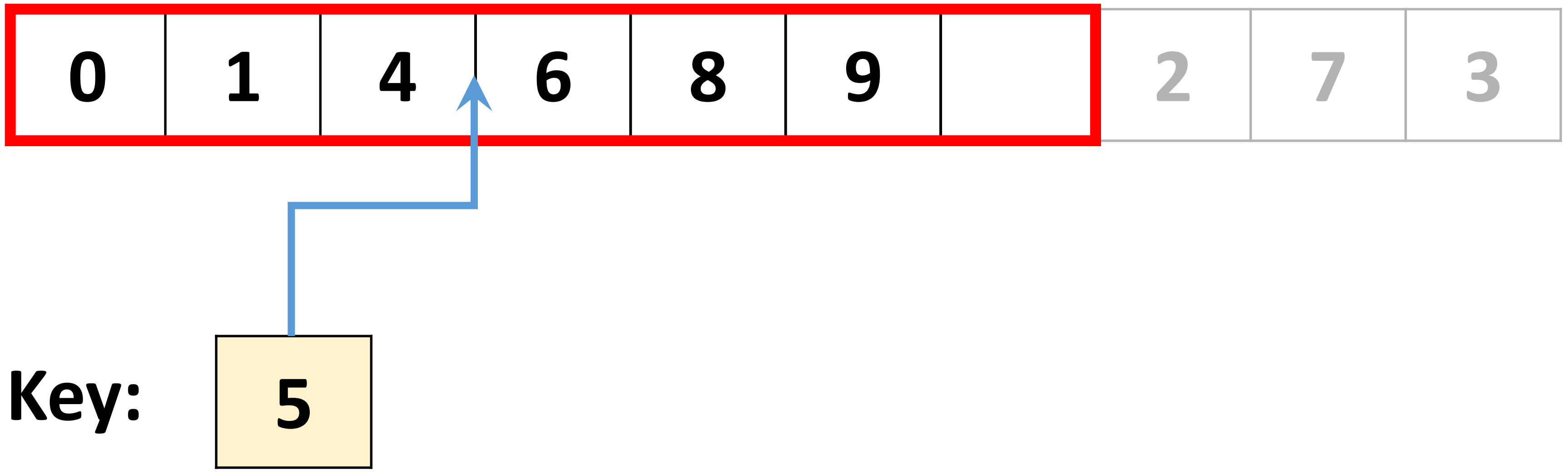
0	1	4	6	8	9		2	7	3

Key: 5

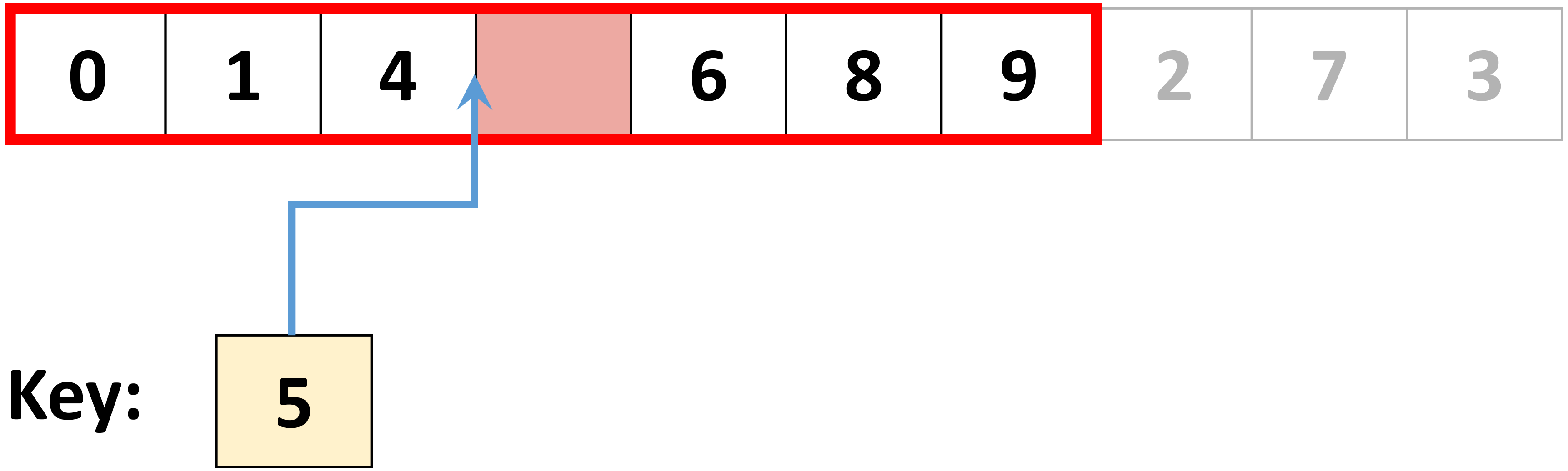
Iteration 6



Iteration 6



Iteration 6



Iteration 6

0	1	4	5	6	8	9	2	7	3

Key:

After Iteration 6

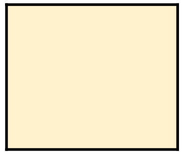
0	1	4	5	6	8	9	2	7	3

Key:

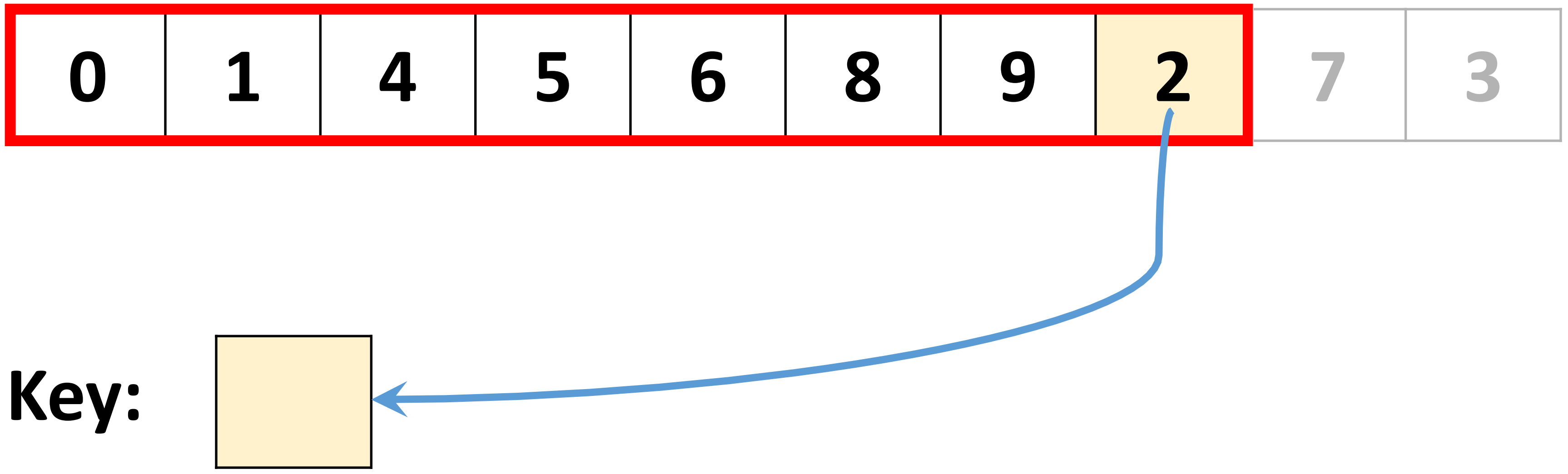
After Iteration 6, the first 7 elements are in ascending order.

Iteration 7

0	1	4	5	6	8	9	2	7	3

Key: 

Iteration 7

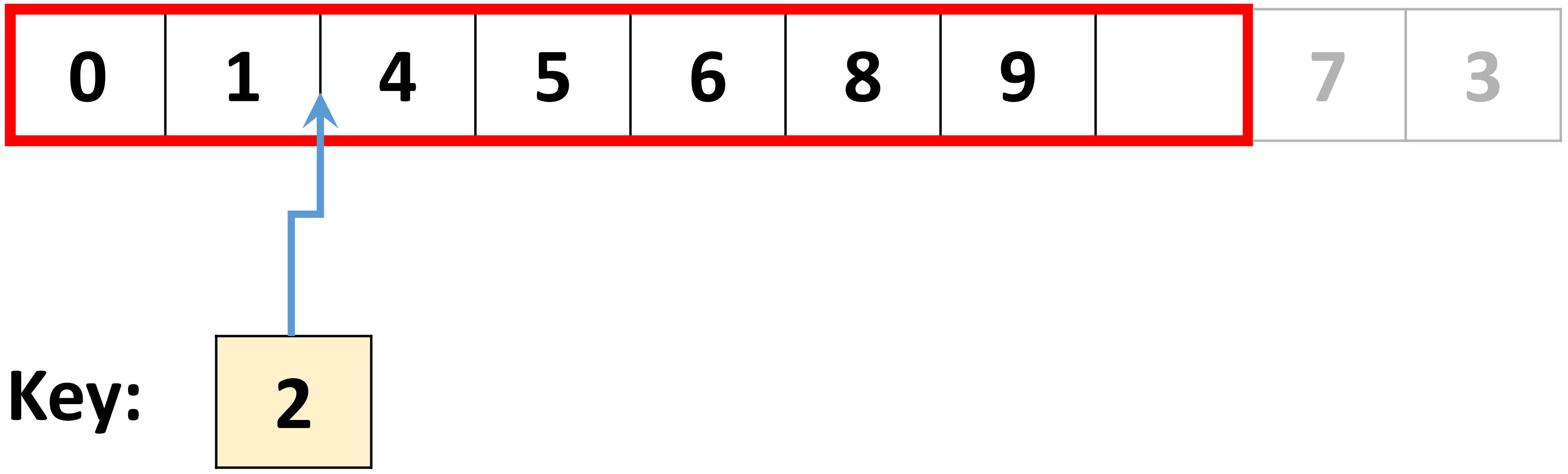


Iteration 7

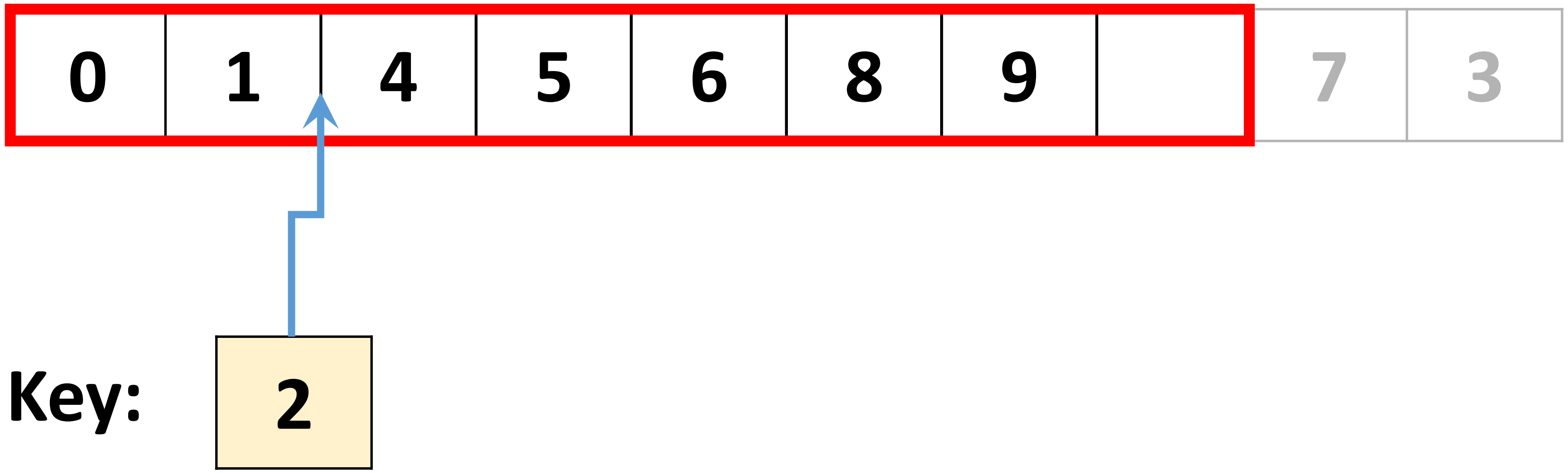
0	1	4	5	6	8	9		7	3

Key: 2

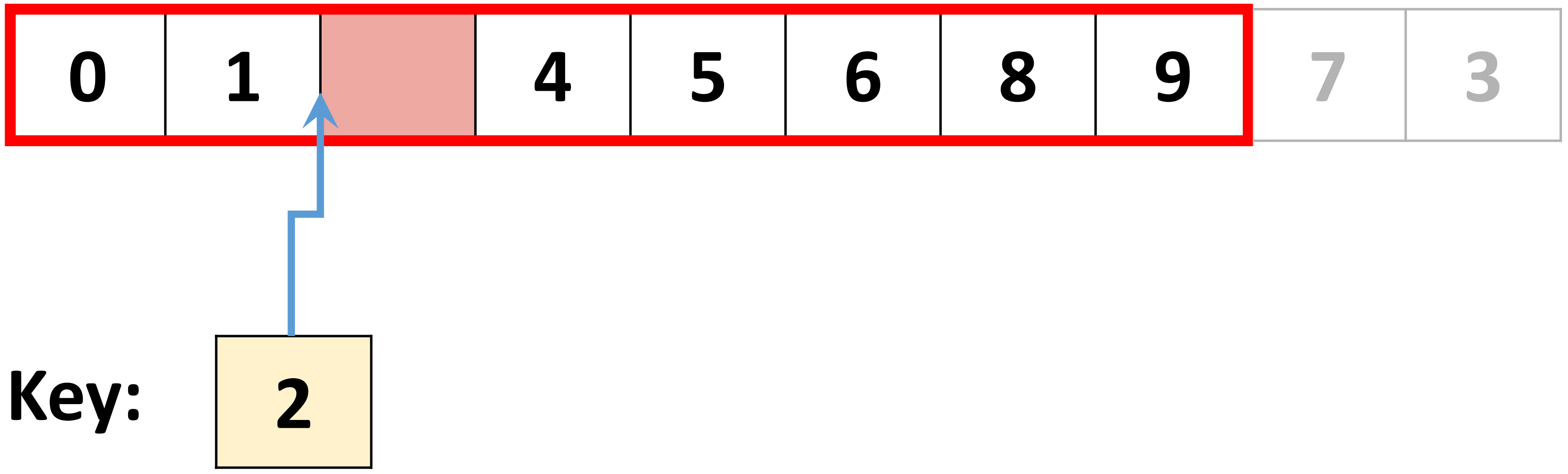
Iteration 7



Iteration 7



Iteration 7



Iteration 7

0	1	2	4	5	6	8	9	7	3

Key:

After Iteration 7

0	1	2	4	5	6	8	9	7	3

Key:

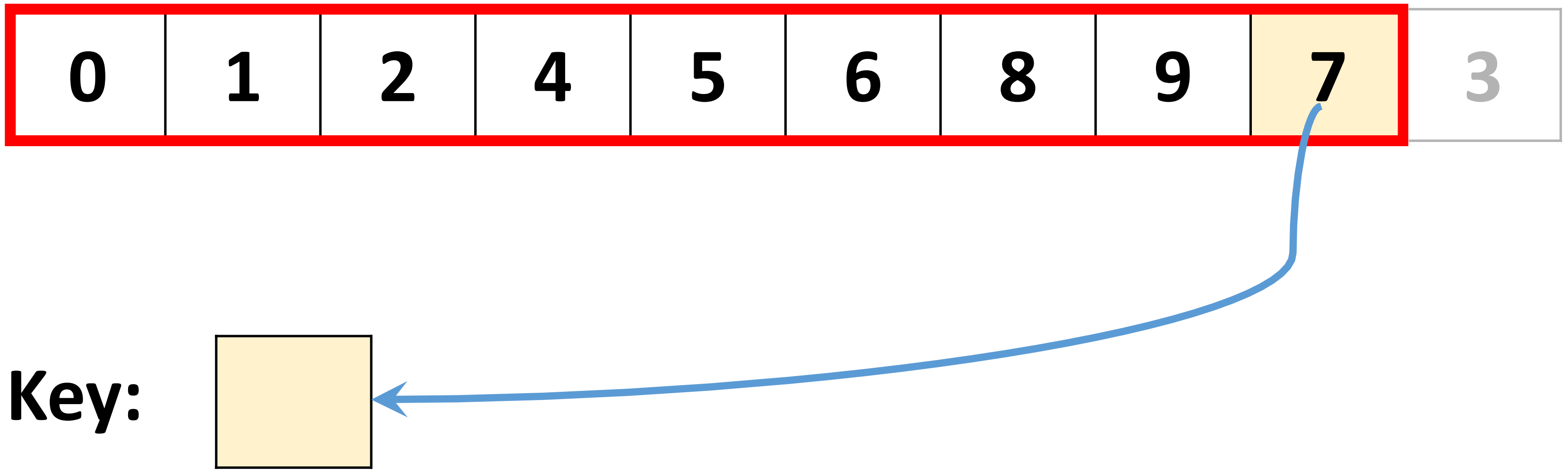
After Iteration 7, the first 8 elements are in ascending order.

Iteration 8

0	1	2	4	5	6	8	9	7	3

Key:

Iteration 8

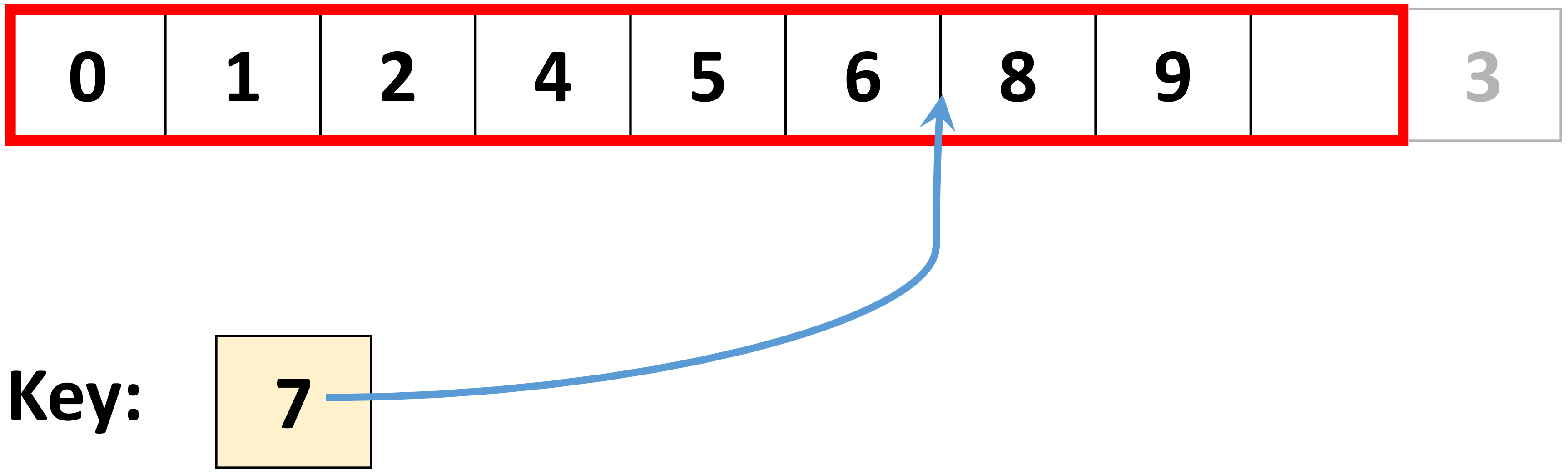


Iteration 8

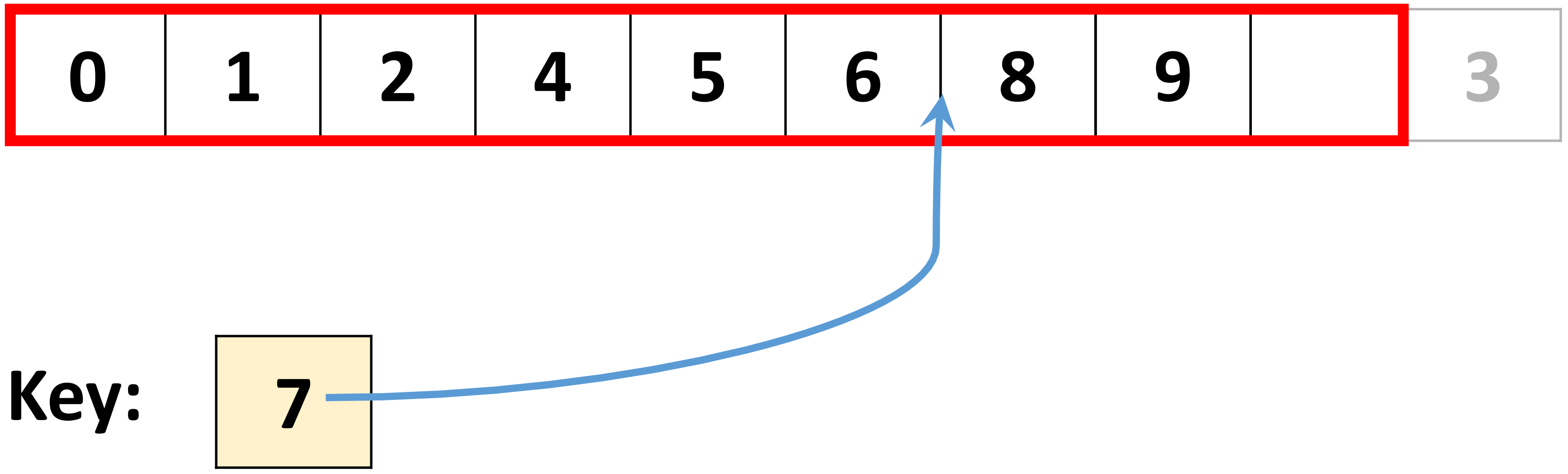
0	1	2	4	5	6	8	9		3

Key: 7

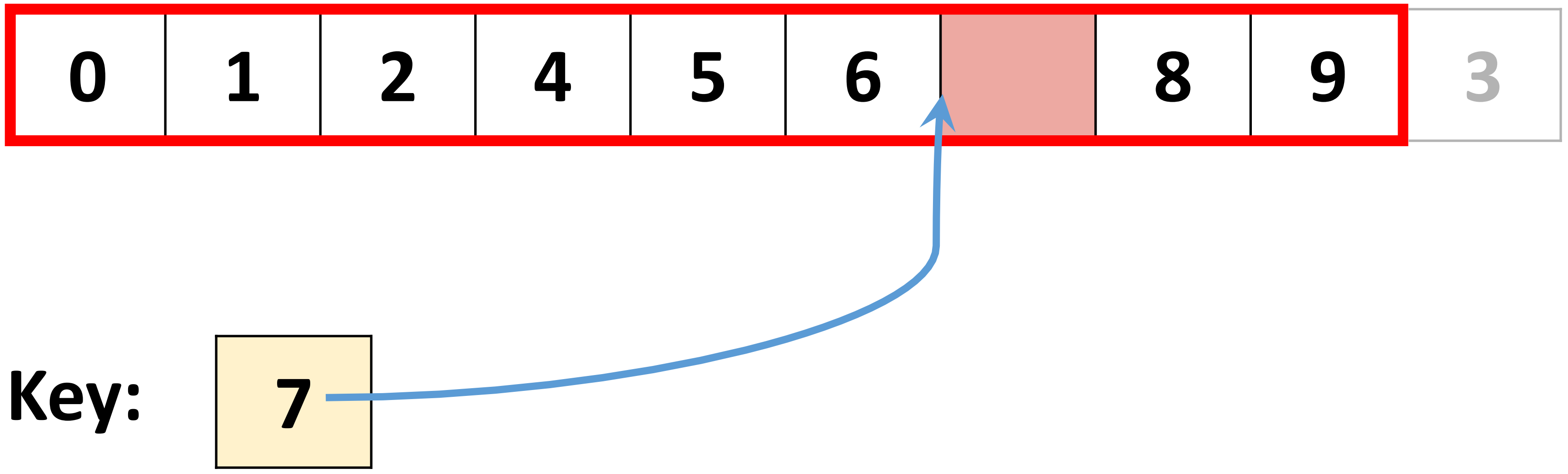
Iteration 8



Iteration 8



Iteration 8



Iteration 8

0	1	2	4	5	6	7	8	9	3

Key:

After Iteration 8

0	1	2	4	5	6	7	8	9	3

Key:

After Iteration 8, the first 9 elements are in ascending order.

Iteration 9

0	1	2	4	5	6	7	8	9	3

Key:

Iteration 9

0	1	2	4	5	6	7	8	9	3

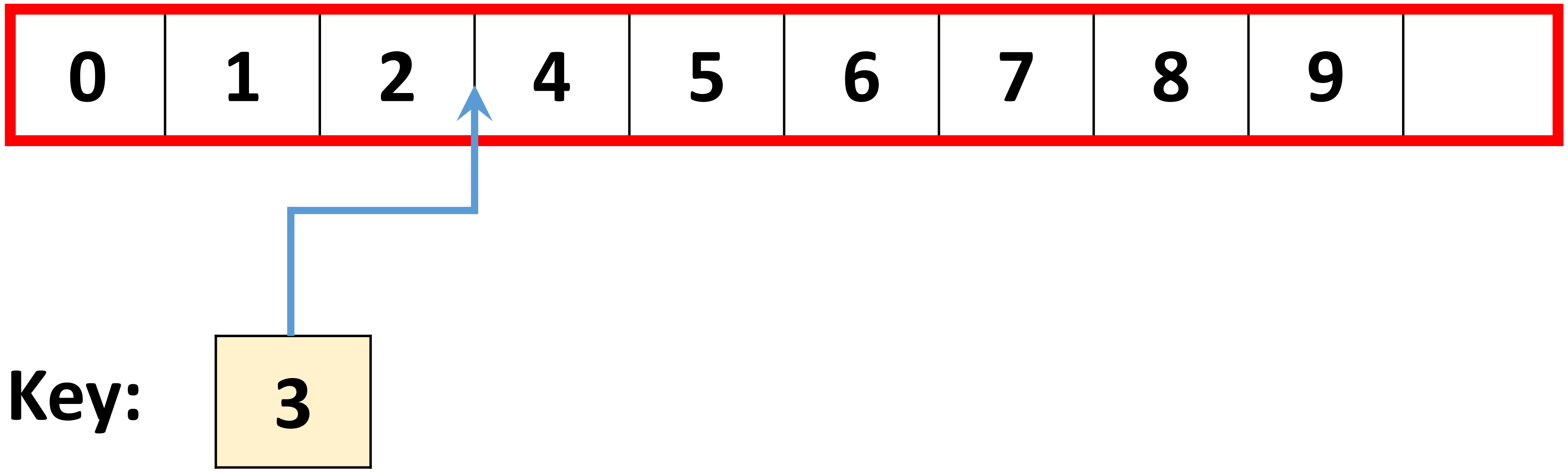
Key:

Iteration 9

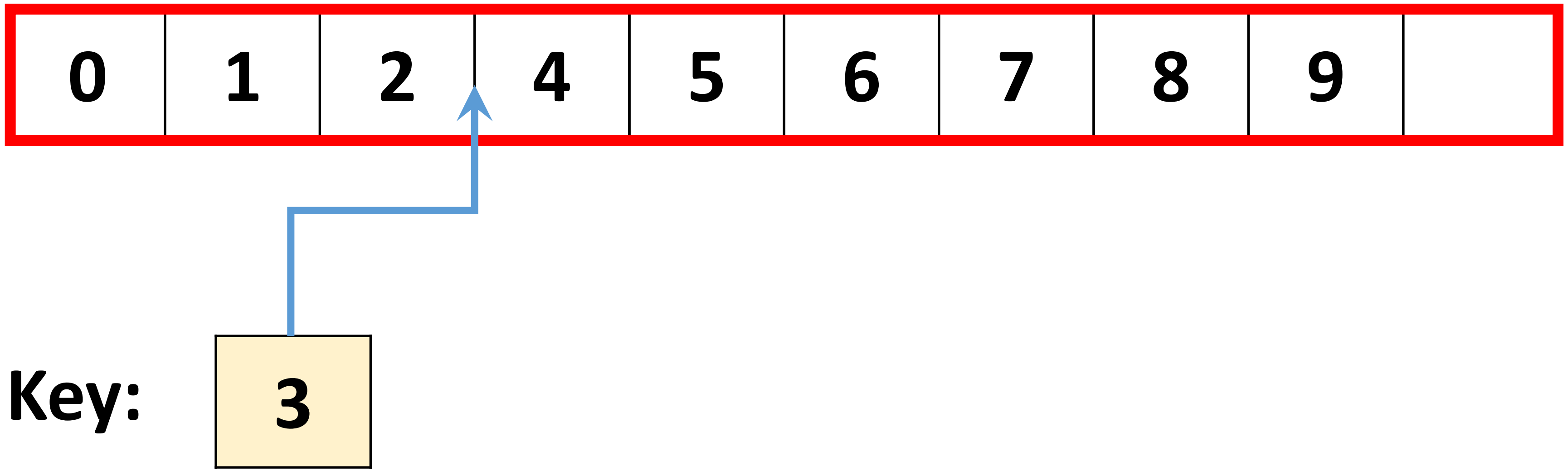
0	1	2	4	5	6	7	8	9	

Key: 3

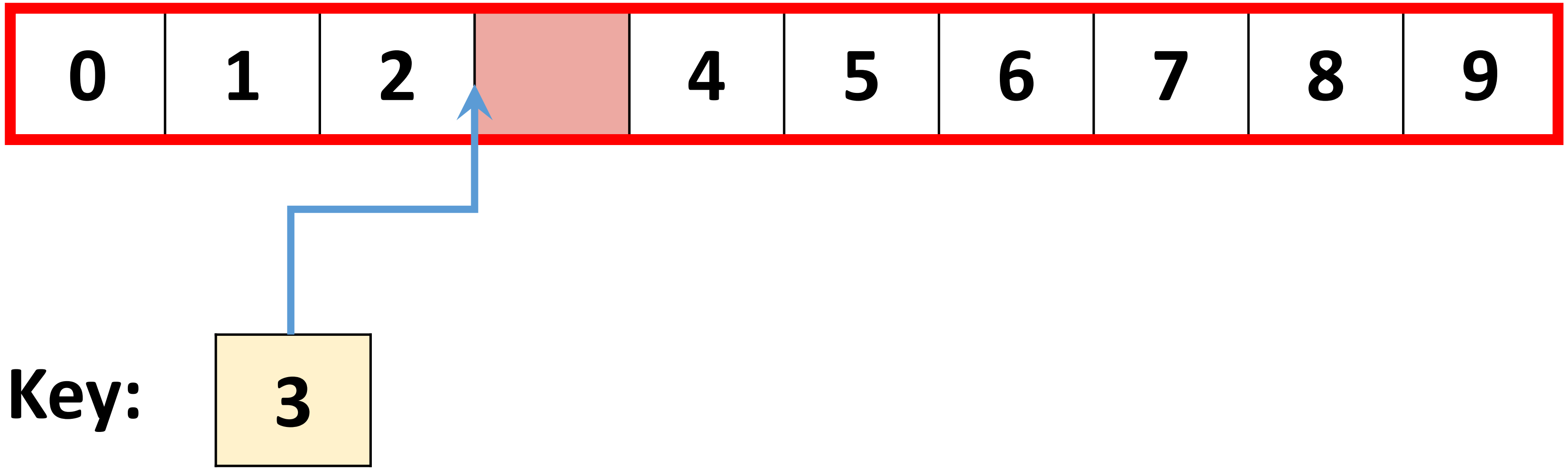
Iteration 9



Iteration 9



Iteration 9



Iteration 9

0	1	2	3	4	5	6	7	8	9

Key:

After Iteration 9

0	1	2	3	4	5	6	7	8	9

After Iteration 9, all the elements are in ascending order.

Implementation

```
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

```
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

```
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
```

1	8	4	9	0	6	5	2	7	3

```
        arr[j + 1] = key;
    }
}
```

```
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];

        arr[j + 1] = key;
    }
}
```

1	8	4	9	0	6	5	2	7	3

```
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
```

1	8	4	9	0	6	5	2	7	3

```
        arr[j + 1] = key;
    }
}
```

```
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
```

4

1	8		9	0	6	5	2	7	3

```
        arr[j + 1] = key;
    }
}
```

```
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

```
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Time Complexity

Worst-Case Time Complexity

5	6	7	8	9	4	3	2	1	0

In the i-th iteration, work on the first $i + 1$ elements.

- In the worst case, the i-th iteration performs i comparisons.

Worst-Case Time Complexity

5	6	7	8	9	4	3	2	1	0

In the i-th iteration, work on the first $i + 1$ elements.

- In the worst case, the i-th iteration performs i comparisons.
- Totally $n - 1$ iterations are needed.

Worst-Case Time Complexity

5	6	7	8	9	4	3	2	1	0

In the i-th iteration, work on the first $i + 1$ elements.

- In the worst case, the i-th iteration performs i comparisons.
- Totally $n - 1$ iterations are needed.
- Worst-case time complexity:

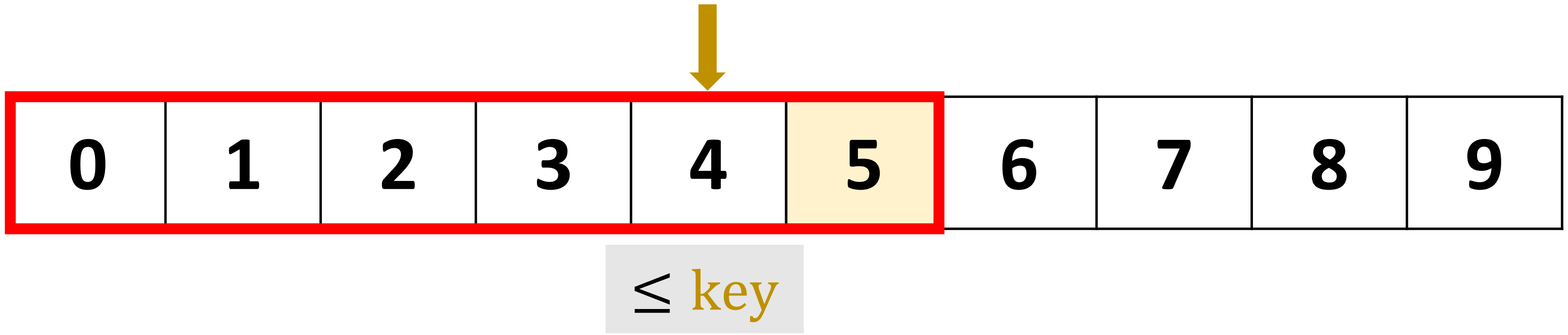
$$\sum_{i=1}^{n-1} i = O(n^2).$$

Best-Case Time Complexity

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

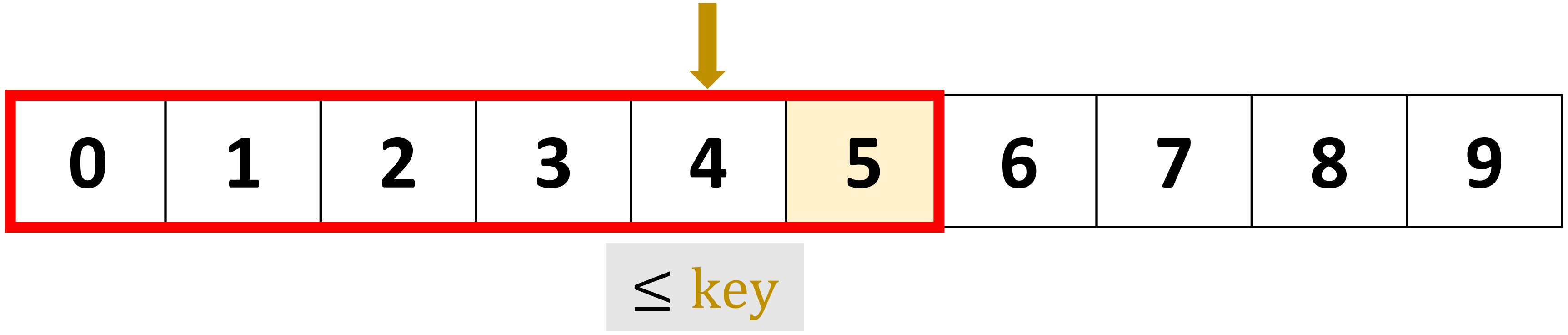
- In the best case, the input array is in ascending order.

Best-Case Time Complexity



- In the best case, the input array is in ascending order.
- The i-th iteration:
  - $\mathrm{arr}[i-1] \leq \mathrm{key}$.

Best-Case Time Complexity



- In the best case, the input array is in ascending order.
- The i-th iteration:
  - $\mathrm{arr}[i-1] \le \mathrm{key}$.
  - Break after only one comparison.
  - Thus one iteration has $O(1)$ time complexity.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9

- In the best case, the input array is in ascending order.
- The i-th iteration has $O(1)$ time complexity.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9

- In the best case, the input array is in ascending order.
- The i-th iteration has $O(1)$ time complexity.
- Totally $n-1$ iterations are needed.
- Best-case time complexity: $O(n)$.

Average-Case Time Complexity

- Worst-case time complexity: $O(n^2)$.
- Best-case time complexity: $O(n)$.
- What about the average case?

- In the average case, the input array has random order.
- Average-case time complexity: $O(n^2)$.

Thank You!