Lecture #4
Sort (1)

Algorithm

JBNU Spring 2021

Jinhong Jung

In Previous Lecture

Recursive complexity

- Complexity of a recursive algorithm is also represented recursively as recurrence relation

How to analyze a recursive complexity function

- **Repeated substitution**
    - Repeatedly substitute the complexity function whose input size decreases toward a base case
- **Mathmetical induction**
    - Estimate the closed form of a recursive complexity, and then prove it by induction
- **Master theorem**
    - Can solve any function in form of $T(n) = aT(n/b) + f(n)$
- For some cases, changing variables makes an equation simple

In This Lecture

Sorting problem

Basic sorting algorithms

- Selection Sort
- Bubble Sort
- Insertion Sort

Outline

Sorting problem ☜

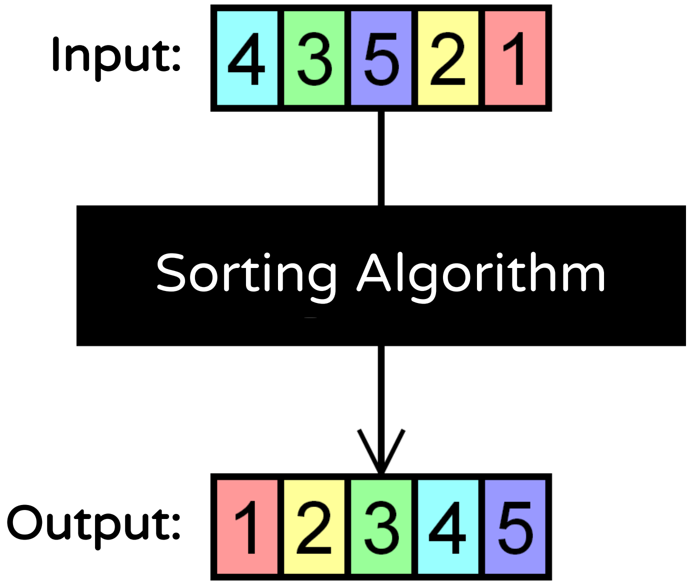
Basic sorting algorithms

- Selection Sort
- Bubble Sort
- Insertion Sort

Sorting Problem

What is sort in CS?

- To rearrange elements in an array in an order
  - If it contains numbers, an order is numerical order
  - If it contains characters, an order is alphabetical order
- Sorting algorithms aim to efficiently sort an arbitrary array in a certain order



Classical Sorting Algorithms

Comparison based sorting

- **Basic sorting algorithms**
  - Show $\Theta(n^2)$ time complexity for a worst case
  - Selection sort, Bubble sort, Insertion sort
- **Advanced sorting algorithms**
  - Show $\Theta(n \log n)$ time complexity for a worst case
  - Merge sort, Quick sort, Heap sort

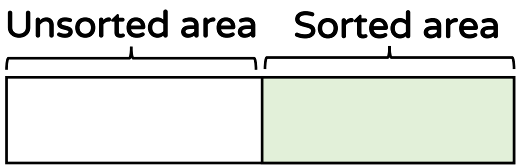
Non-comparison based sorting

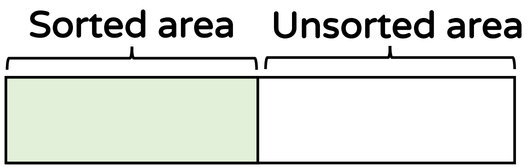
- **Special sorting algorithms**
  - Show $\Theta(n)$ time complexity for special cases (not general)
  - Radix sort, Counting sort

Before Going Further

Notes

- The index of an array A starts from 1
- An array A will be sorted in the ascending order
  - $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \cdots$
- $\mathrm{A}[1 \cdots i]$ indicates elements from index 1 to i
- Sorted area (colored by light green)
  - Consecutive sorted partial part in an array while it is sorted by a sorting algorithm
- Unsorted area (colored by white)
  - Remaining part except the sorted area





Outline

Sorting problem

Basic sorting algorithms

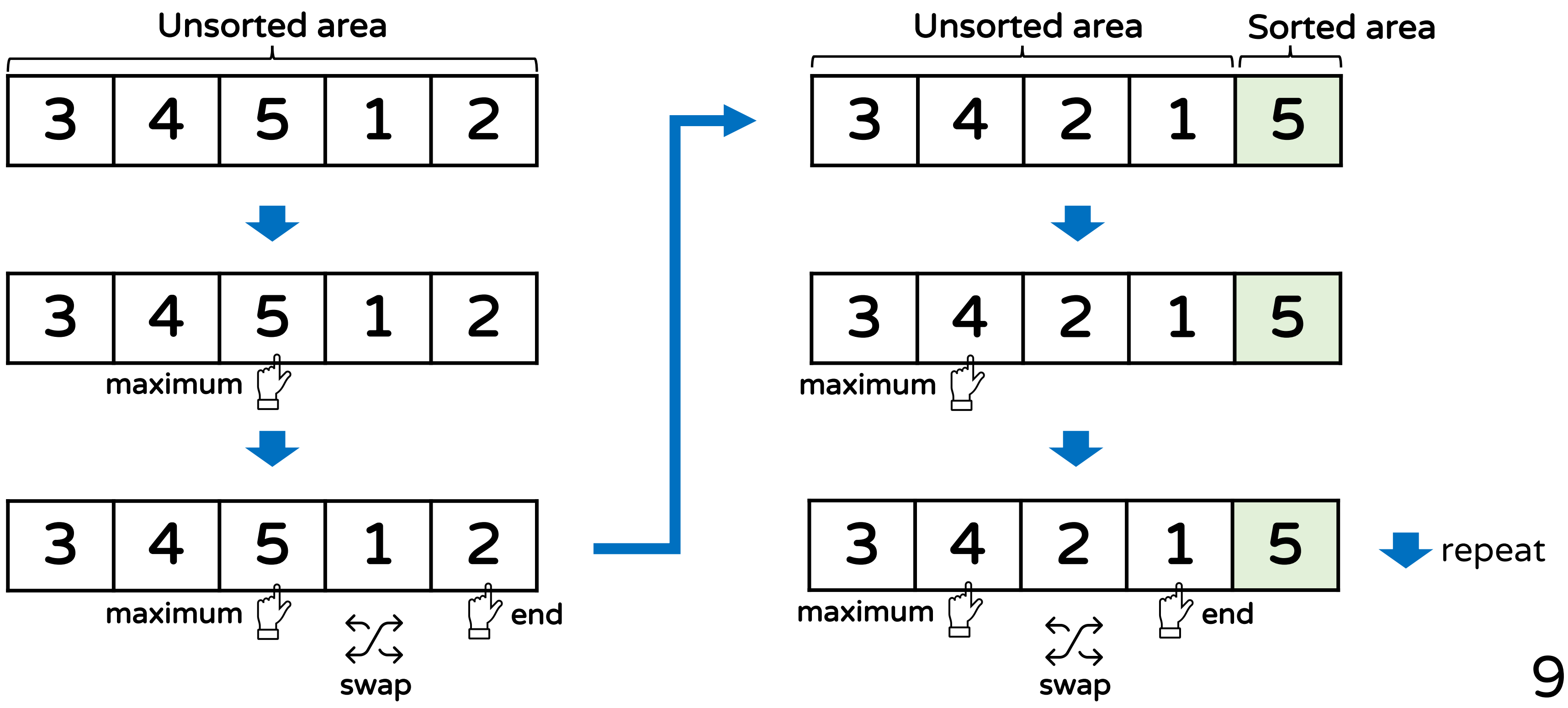
- Selection Sort
- Bubble Sort
- Insertion Sort

Selection Sort (1)

Idea of selection sort

Or move the minimum
to the front

- Move the maximum to the end of unsorted area
  - Find the maximum & swap it with the end of unsorted area
  - Repeat these until the unsorted area becomes empty



Selection Sort (2)

Pseudocode of selection sort

```
def selection_sort(A, $n$):
    for end ← $n$ downto 2:
        # at this step, unsorted area is [1, end]

        # selection stage
        $k$ ← find the maximum's index among A[1⋯end]
        swap A[$k$] and A[end]

        # at this step, unsorted area is [1, end-1]
```

- **Correctness**
  - Obviously true since for each step, the sorted area is expanded correctly, and finally its size becomes n at the end
- **T.M.I. :** $k \leftarrow$ find the maximum's index among $\mathrm{A}[1 \cdots \mathrm{end}]$

$$k = \operatorname*{argmax}_{i \in [1, \mathrm{end}]} \mathrm{A}[i]$$

Selection Sort (3)

Time complexity of selection sort

```
def selection_sort(A, $n$):
    for end ← $n$ downto 2:
        # selection stage
        $k$ ← find the maximum's index among A[1⋯end]
        swap A[$k$] and A[end]
```

selection stage

- Time complexity analysis
  - The for-loop repeats the selection process $n-1$ times
  - In the selection stage,
    - Main operations are comparisons for linearly searching the maximum's index
    - $i-1$ comparisons are required for i-th selection process

$$T(n) = (n-1) + (n-2) + \cdots + 2 + 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$$

Outline

Sorting problem

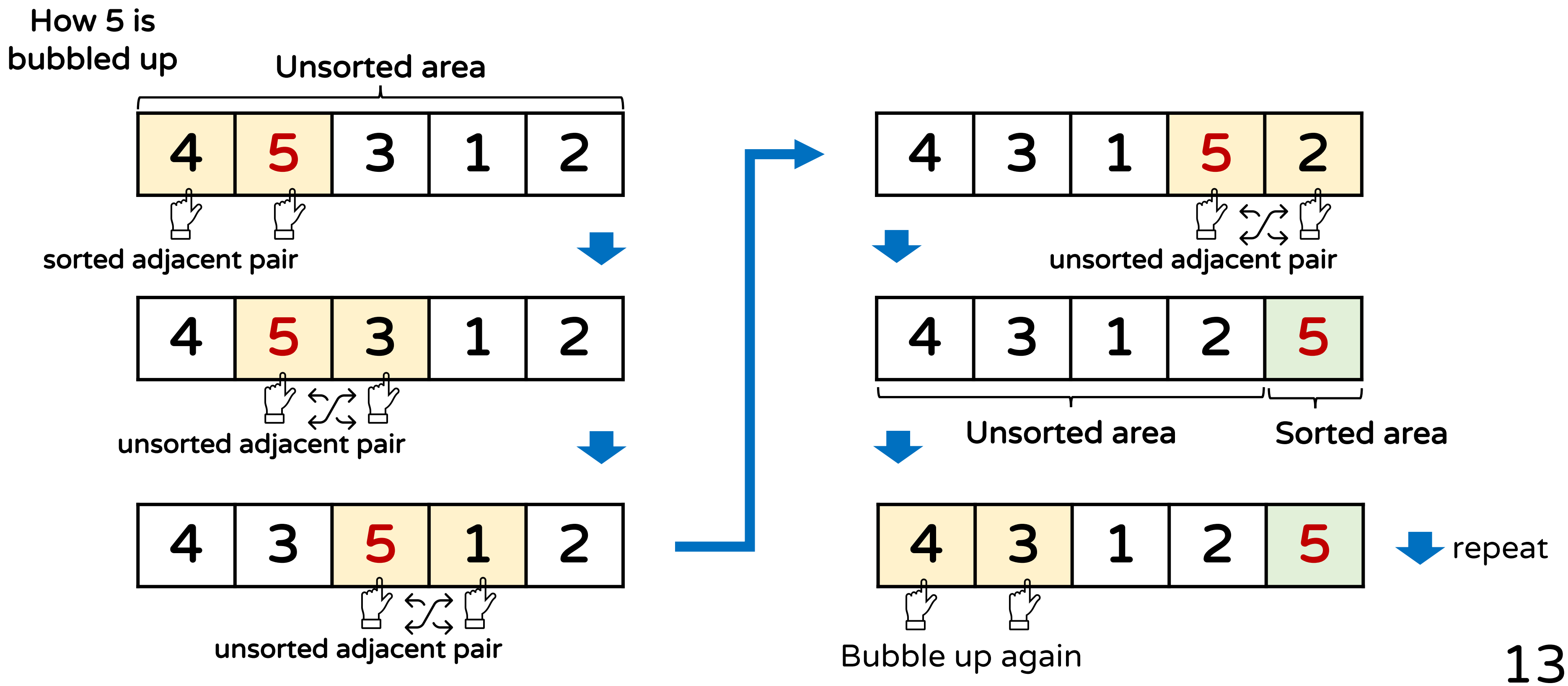
Basic sorting algorithms

- Selection Sort
- **Bubble Sort** ☜
- Insertion Sort

Bubble Sort (1)

Idea of bubble sort

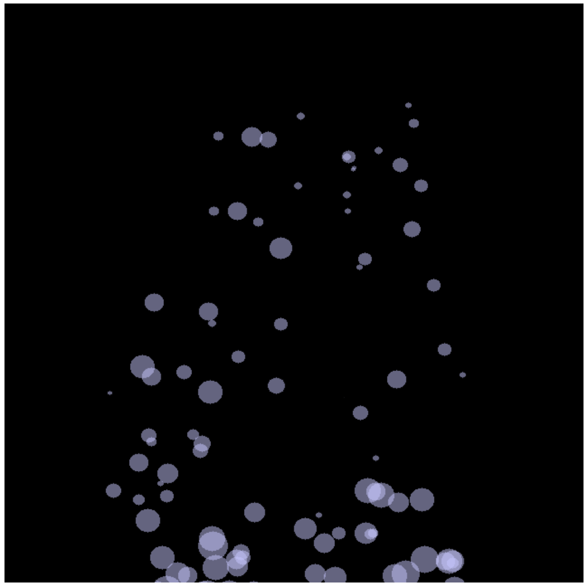
- Move the maximum to the end of unsorted area
  - Bubble it up by repeatedly doing swap unsorted adjacent pairs
  - Repeat these until the unsorted area becomes empty



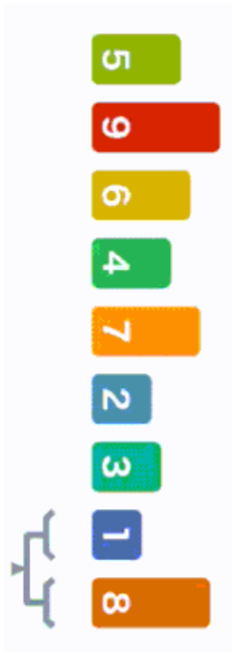
Bubble Sort (2)

Idea of bubble sort

- Move the maximum to the end of unsorted area
  - Bubble it up by repeatedly doing swap unsorted adjacent pairs
  - Repeat these until the unsorted area becomes empty



Bubble up



This sort bubbles the **minimum** up each stage

Bubble Sort (3)

Pseudocode of bubble sort

```
def bubble_sort(A, $n$):
    for end ← $n$ downto 2:
        # at this step, unsorted area is [1, end]

        # bubble up stage (push the maximum to the end)
        for i ← 1 to end – 1:
            if A[i] > A[i+1]:
                swap A[i] and A[i+1]

        # at this step, unsorted area is [1, end-1]
```

- Correctness
    - Like selection sort, the sorted area is expanded correctly, and finally its size becomes n at the end

Bubble Sort (4)

Time complexity of bubble sort

```
def bubble_sort(A, $n$):
    for end ← $n$ downto 2:
        # bubble up stage
        for i ← 1 to end - 1:
            if A[i] > A[i+1]:
                swap A[i] and A[i+1]
```

$\left.\right\}$ Bubble up stage

- Time complexity analysis
  - The outer for-loop repeats the bubble up stage $n-1$ times
  - In the bubble up stage
    - Main operations are comparison and swap having the same # of operations
    - $i-1$ comparisons are required for the i-th bubble up stage

$$T(n) = (n-1) + (n-2) + \cdots + 2 + 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$$

Outline

Sorting problem

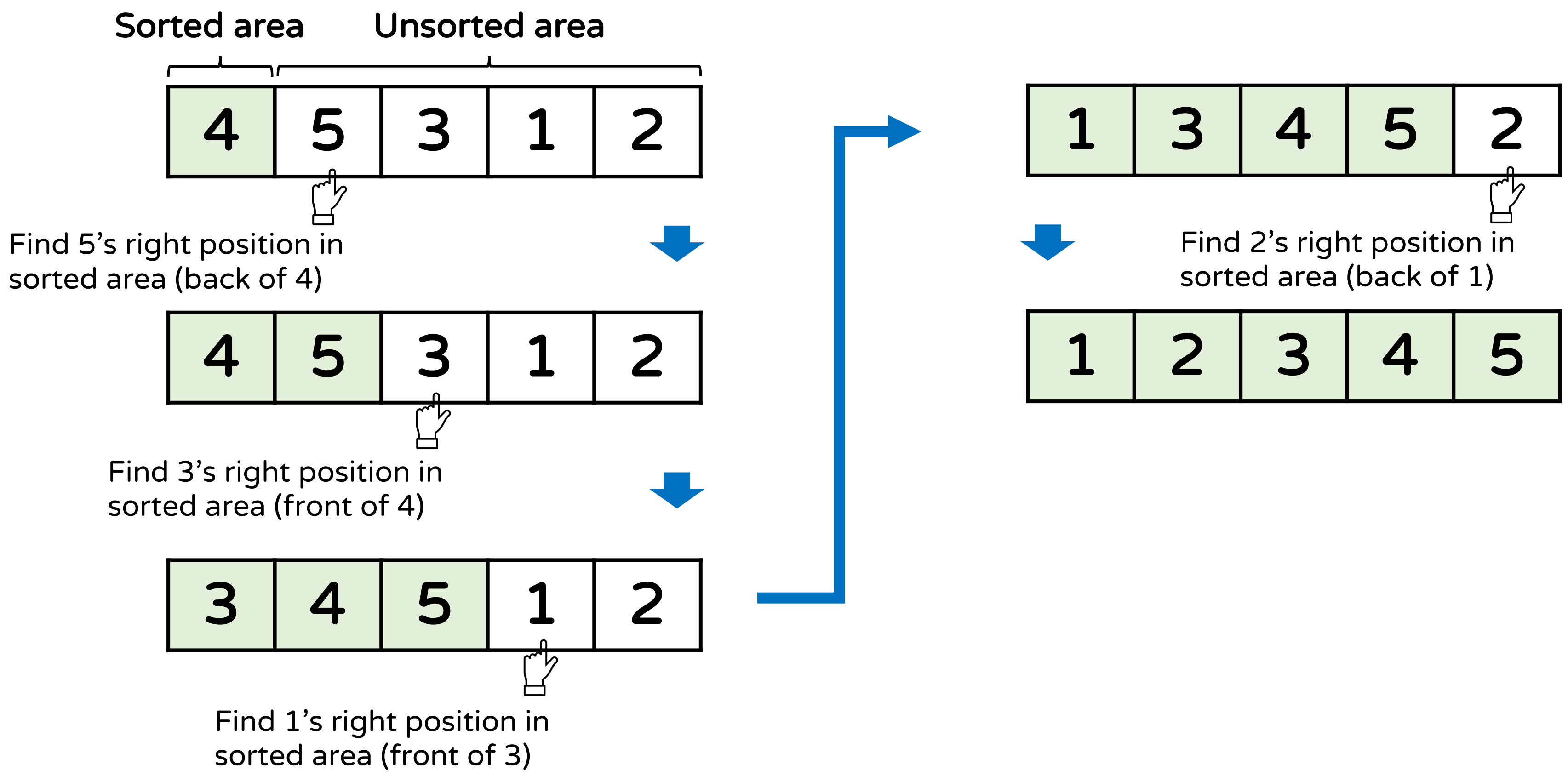
Basic sorting algorithms

- Selection Sort
- Bubble Sort
- Insertion Sort ☜

Insertion Sort (1)

Idea of insert sort

- Pick the front of unsorted area
- Insert it into its right position in sorted area


Sorted area
Unsorted area
4 5 3 1 2
Find 5's right position in sorted area (back of 4)
4 5 3 1 2
Find 3's right position in sorted area (front of 4)
3 4 5 1 2
Find 1's right position in sorted area (front of 3)
1 3 4 5 2
Find 2's right position in sorted area (back of 1)
1 2 3 4 5

Insertion Sort (2)

Pseudocode of insertion sort

```
def insertion_sort(A, n):
    for i ← 2 to n:
        # at this step, sorted area is [1, i-1]

        # insertion stage
        insert A[i] into its right position in A[1⋯i]

        # at this step, sorted area is [1, i]
```

- **Correctness**
  - For base case ($i = 1$), $\mathrm{A}[i]$ is already sorted
  - Assume it correctly works for $i = k$ (i.e., $\mathrm{A}[1 \cdots k]$ is sorted)
  - Then, the insertion stage will correctly insert $\mathrm{A}[k + 1]$ into the sorted area, meaning $\mathrm{A}[1 \cdots k + 1]$ is also sorted
  - Thus, for $i = n$, $\mathrm{A}[1 \cdots n]$ will be sorted correctly (by induction)

Insertion Sort (3)

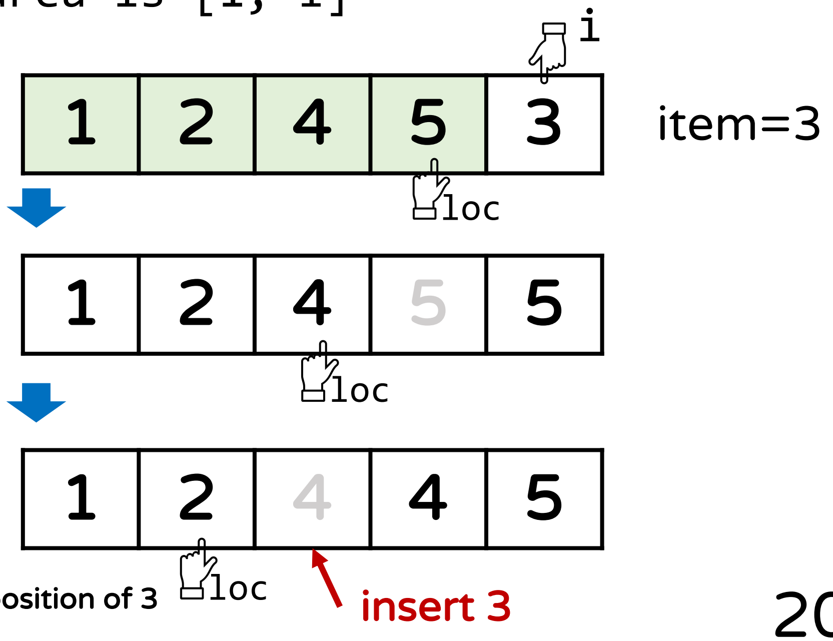
Pseudocode of insertion sort

```
def insertion_sort(A, n):
    for i ← 2 to n:
        # at this step, sorted area is [1, i-1]

        # insertion stage
        insert A[i] into its right position in A[1⋯i]

        # at this step, sorted area is [1, i]
```

```
# i-th insertion stage
loc ← i - 1
item ← A[i]
while loc >= 1 and item < A[loc]:
    A[loc+1] ← A[loc]
    loc ← loc - 1
A[loc+1] ← item
```



Back of 2 is the right position of 3

Insertion Sort (4)

Time complexity of insertion sort

```
def insertion_sort(A, $n$):
    for i ← 2 to n:
        # insertion stage
        insert A[i] into its right position in A[1⋯i]
```

- **Time complexity analysis**
  - The outer for-loop repeats the insertion stage $n-1$ times
  - The i-th insertion stage performs at most i operations (comparisons)

$$T(n) = 1 + 2 + \cdots + (n-2) + (n-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$$

What You Need To Know

Sorting problem

- To efficiently rearrange elements in an array in an order

Basic sorting algorithms

- **Selection Sort**
  - Move the maximum to the end of unsorted area
    - Find the maximum directly by linearly searching
- **Bubble Sort**
  - Move the maximum to the end of unsorted area
    - Move the maximum by bubbling it up
- **Insertion Sort**
  - Pick the front of unsorted area
  - Insert it into its right position in sorted area

In Next Lecture

Discussion on basic sorting algorithms

- Selection Sort
- Bubble Sort
- Insertion Sort

Advanced sorting algorithms

- Merge Sort
- Quick Sort

Thank You