Bubble Sort

(What are Algorithms and How to Analyze Algorithms)

Hengfeng Wei

Institute of Computer Software
Nanjing University

December 17, 2016

Bubble Sort

Bubble Sort

Sorting

The sorting problem:

Input: A sequence A of n integers $a_1\ a_2\ \cdots\ a_n$.

Output: A permutation $a'_1\ a'_2\ \ldots\ a'_n$ of A *s.t.* $a'_1 \leq a'_2 \leq \cdots \leq a'_n$ (non-decreasing order).

Sorting

The sorting problem:

Input: A sequence A of n integers $a_1\ a_2\ \cdots\ a_n$.

Output: A permutation $a'_1\ a'_2\ \ldots\ a'_n$ of A *s.t.* $a'_1 \leq a'_2 \leq \cdots \leq a'_n$ (non-decreasing order).

$$3\ 1\ 4\ 2 \Longrightarrow 1\ 2\ 3\ 4$$

Sorting Algorithms

An algorithm is a sequence of computational steps that transform the input into the output.

Sorting Algorithms

An algorithm is a sequence of computational steps that transform the input into the output.

Definiteness: precisely defined steps

Sorting Algorithms

An algorithm is a sequence of computational steps that transform the input into the output.

Definiteness: precisely defined steps

Finiteness: termination

Sorting Algorithms

An algorithm is a sequence of computational steps that transform the input into the output.

Definiteness: precisely defined steps

Finiteness: termination

Effectiveness: operations are sufficiently basic

Sorting Algorithms

An algorithm is a sequence of computational steps that transform the input into the output.

Definiteness: precisely defined steps

Finiteness: termination

Effectiveness: operations are sufficiently basic

- RAM (Random-Access Machine) model

Sorting Algorithms

An algorithm is a sequence of computational steps that transform the input into the output.

Definiteness: precisely defined steps

Finiteness: termination

Effectiveness: operations are sufficiently basic

- RAM (Random-Access Machine) model
- unrealistic: `sort` instruction

Sorting Algorithms

An algorithm is a sequence of computational steps that transform the input into the output.

Definiteness: precisely defined steps

Finiteness: termination

Effectiveness: operations are sufficiently basic

- RAM (Random-Access Machine) model
- unrealistic: `sort` instruction
- realistic: arithmetic, data movement, and control

Sorting Algorithms

An algorithm is a sequence of computational steps that transform the input into the output.

Definiteness: precisely defined steps

Finiteness: termination

Effectiveness: operations are sufficiently basic

- RAM (Random-Access Machine) model
- unrealistic: `sort` instruction
- realistic: arithmetic, data movement, and control
- CAS for sort: compare and swap if out-of-order

Inversions

$$A = a_1, a_2, \ldots, a_n.$$

If $i < j$ and $a_i > a_j$, then (a_i, a_j) is an **inversion**.

Inversions

$$A = a_1, a_2, \ldots, a_n.$$

If $i < j$ and $a_i > a_j$, then (a_i, a_j) is an **inversion**.

$$A = 3, 1, 4, 2.$$

Inversions: $(3, 1), (3, 2), (4, 2)$

Inversions

$$A = a_1, a_2, \ldots, a_n.$$

If $i < j$ and $a_i > a_j$, then (a_i, a_j) is an **inversion**.

$$A = 3, 1, 4, 2.$$

Inversions: $(3, 1), (3, 2), (4, 2)$

Adjacent inversion: $j = i + 1$

Inversions

$$A = a_1, a_2, \ldots, a_n.$$

If $i < j$ and $a_i > a_j$, then (a_i, a_j) is an **inversion**.

$$A = 3, 1, 4, 2.$$

Inversions: $(3, 1), (3, 2), (4, 2)$

Adjacent inversion: $j = i + 1$

Inversions

$$
\begin{aligned}
A \text{ is sorted} &\Longrightarrow A \text{ has no inversions} \\
&\Longrightarrow A \text{ has no adjacent inversions.}
\end{aligned}
$$

Inversions

$$
\begin{aligned}
A \text{ is sorted} &\implies A \text{ has no inversions} \\
&\implies A \text{ has no adjacent inversions.}
\end{aligned}
$$

$$
\begin{aligned}
A \text{ has no adjacent inversions} &\implies \forall i \in [1, n-1] : a_i \leq a_{i+1} \\
&\implies A \text{ is sorted.}
\end{aligned}
$$

Inversions

$$A \text{ is sorted} \implies A \text{ has no inversions}$$
$$\implies A \text{ has no adjacent inversions.}$$

$$A \text{ has no adjacent inversions} \implies \forall i \in [1, n-1] : a_i \leq a_{i+1}$$
$$\implies A \text{ is sorted.}$$

$$\boxed{A \text{ is sorted} \iff A \text{ has no adjacent inversions.}}$$

Bubble Sort

Bubble Sort

Basic idea: to eliminate all adjacent inversions

```
1: procedure BubbleSortOverview(A : a_1 a_2 ··· a_n)
2:     repeat
3:         Pick any i
4:         if a_i > a_{i+1} then                    ▷ CAS
5:             Swap(a_i, a_{i+1})
6:     until no adjacent inversions
```

Bubble Sort

Basic idea: to eliminate all adjacent inversions

```
1: procedure BubbleSortOverview(A : a_1 a_2 ··· a_n)
2:     repeat
3:         Pick any i
4:         if a_i > a_{i+1} then                    ▷ CAS
5:             Swap(a_i, a_{i+1})
6:     until no adjacent inversions
```

Definiteness!

Bubble Sort

Basic idea: to eliminate all adjacent inversions

```
1: procedure BubbleSortOverview(A : a_1 a_2 ··· a_n)
2:     repeat
3:         Pick any i
4:         if a_i > a_{i+1} then                    ▷ CAS
5:             Swap(a_i, a_{i+1})
6:     until no adjacent inversions
```

Definiteness!

Finiteness!

Bubble Sort

1: **procedure** BUBBLESORT($A : a_1\ a_2\ \cdots\ a_n$)
2: **repeat**
3: [ ]
4: **for** $i \leftarrow 1 : n-1$ **do** ▷ Pick i
5: **if** $a_i > a_{i+1}$ **then**
6: SWAP(a_i, a_{i+1})
7: [ ]
8: **until** [ ]

Bubble Sort

1: **procedure** BUBBLESORT($A : a_1\ a_2\ \cdots\ a_n$)
2: **repeat**
3: swapped $\leftarrow$ false
4: **for** $i \leftarrow 1 : n-1$ **do** ▷ Pick i
5: **if** $a_i > a_{i+1}$ **then**
6: SWAP(a_i, a_{i+1})
7:
8: **until**

Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         [swapped ← false]
4:         for i ← 1 : n − 1 do                          ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 [swapped ← true]
8:     until [          ]
```

Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                          ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false
```

Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                          ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false
```

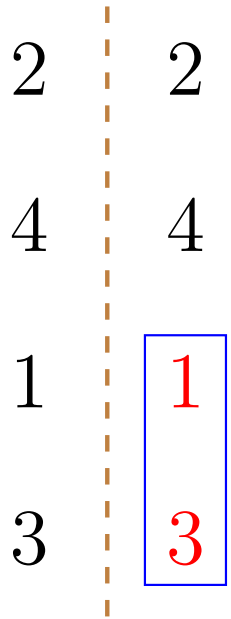
Definiteness!

Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```

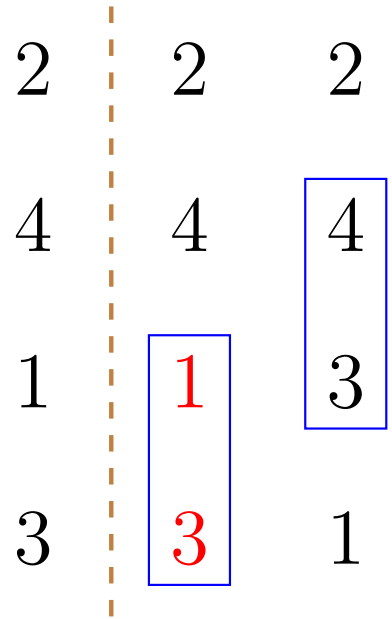
Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```



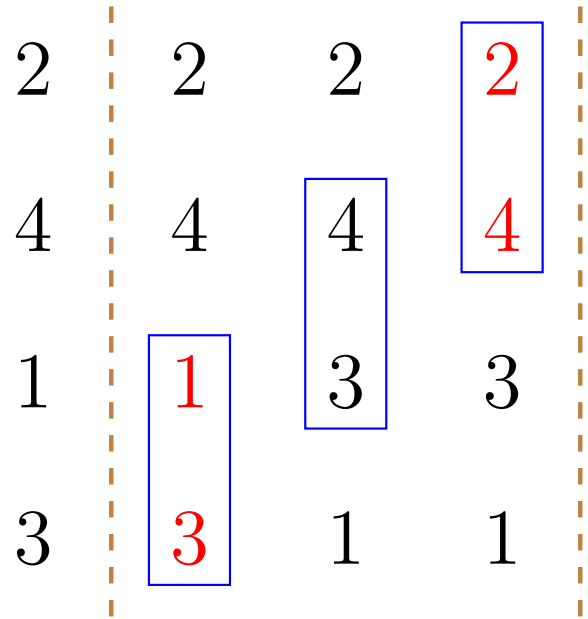
Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```



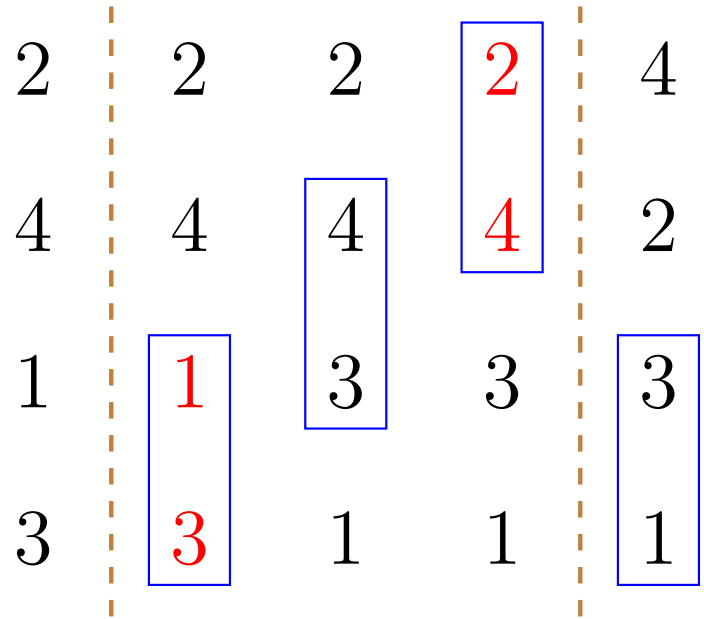
Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```



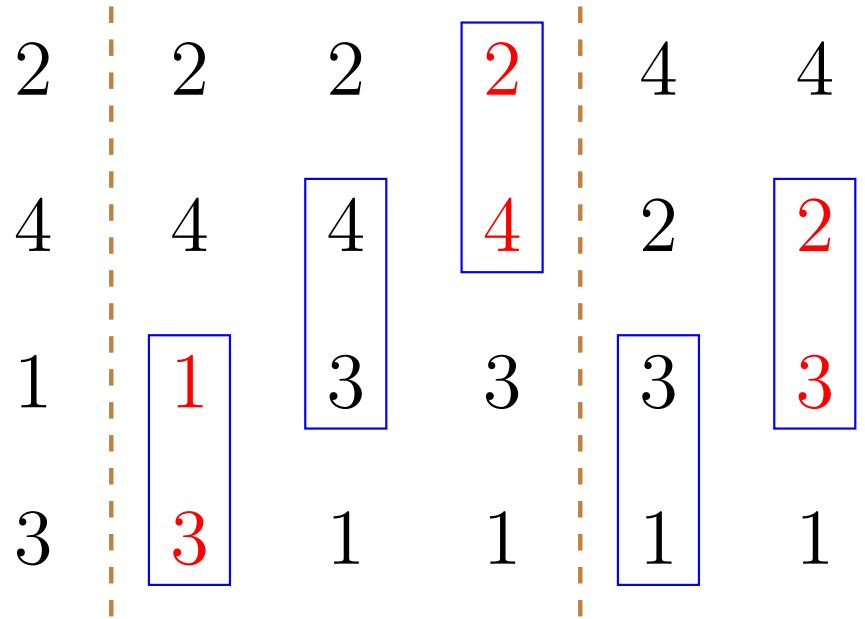
Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```



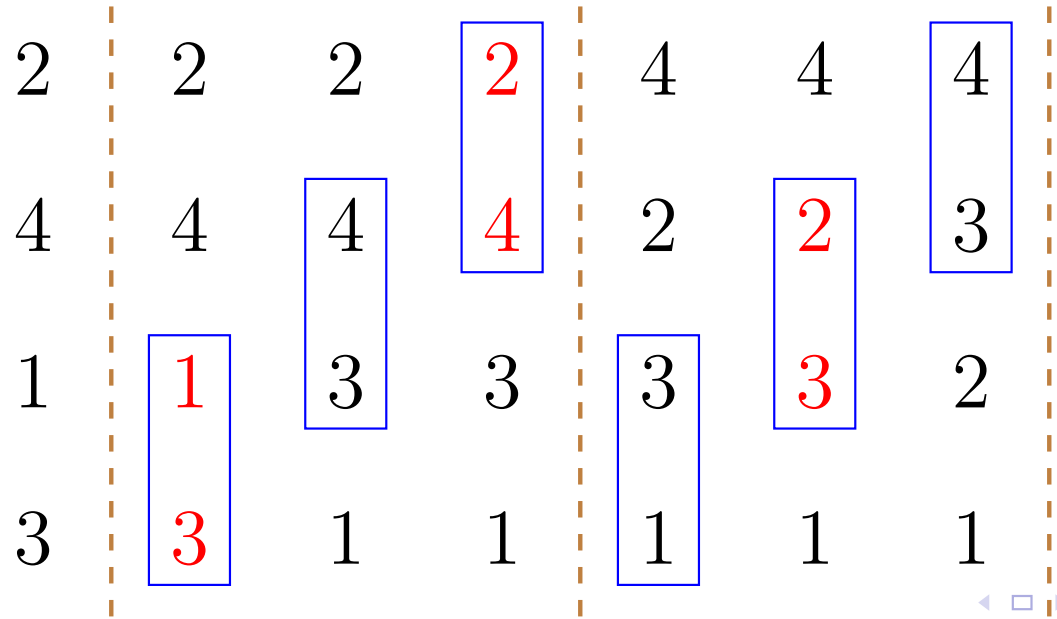
Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```



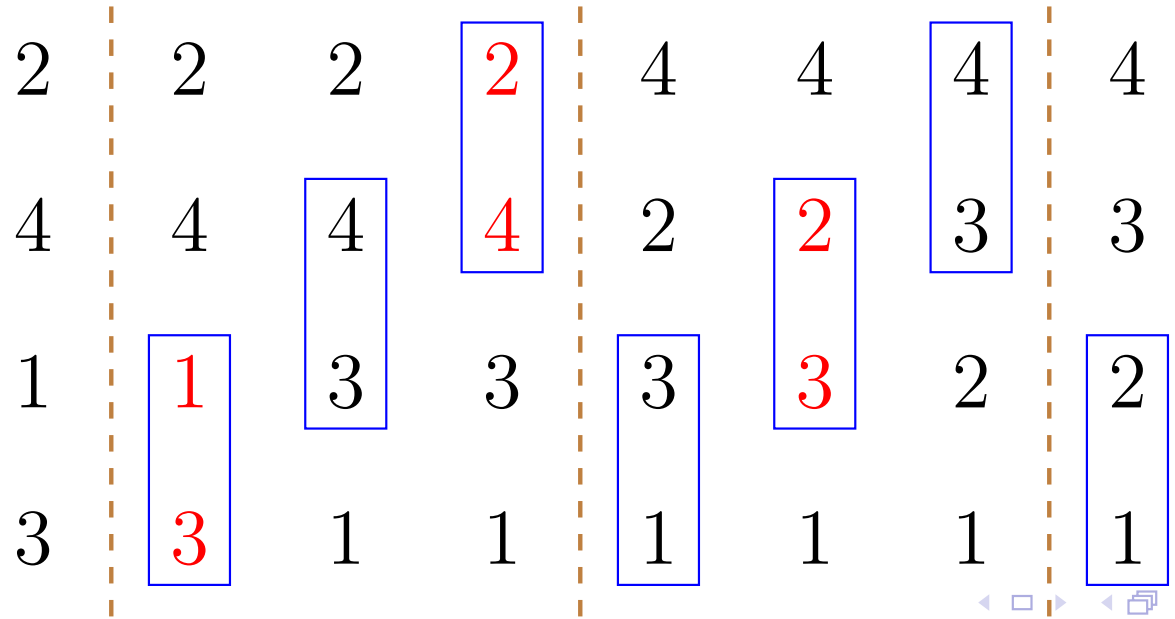
Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```



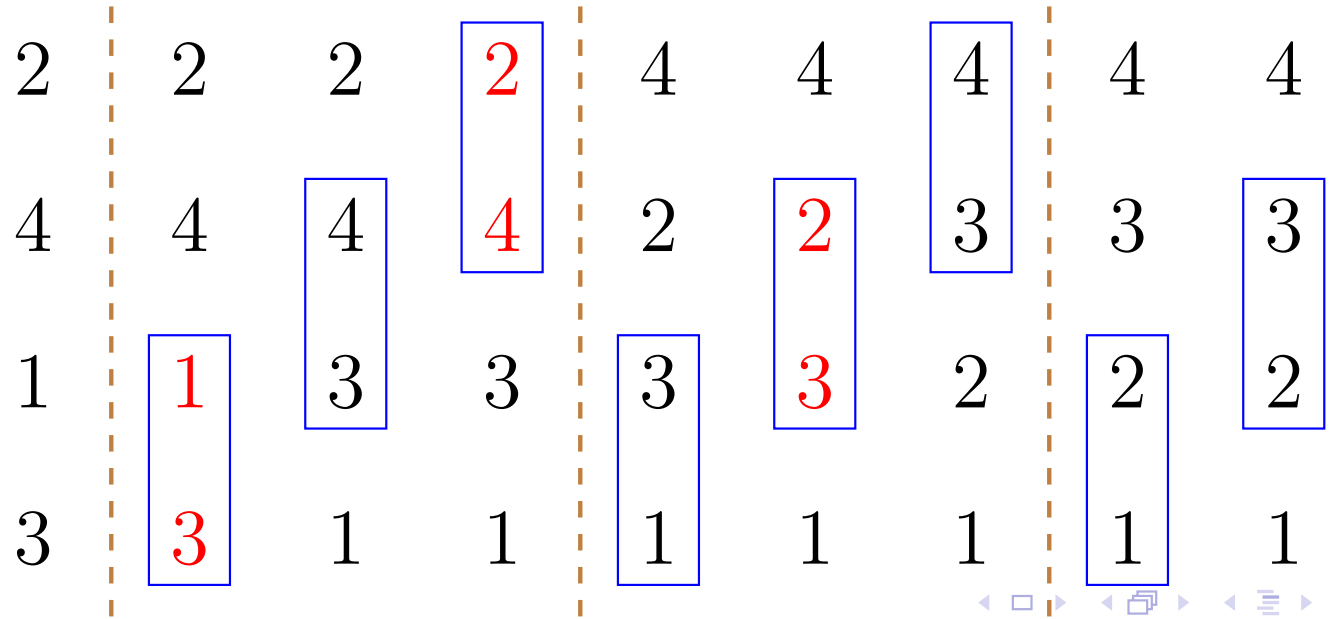
Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```



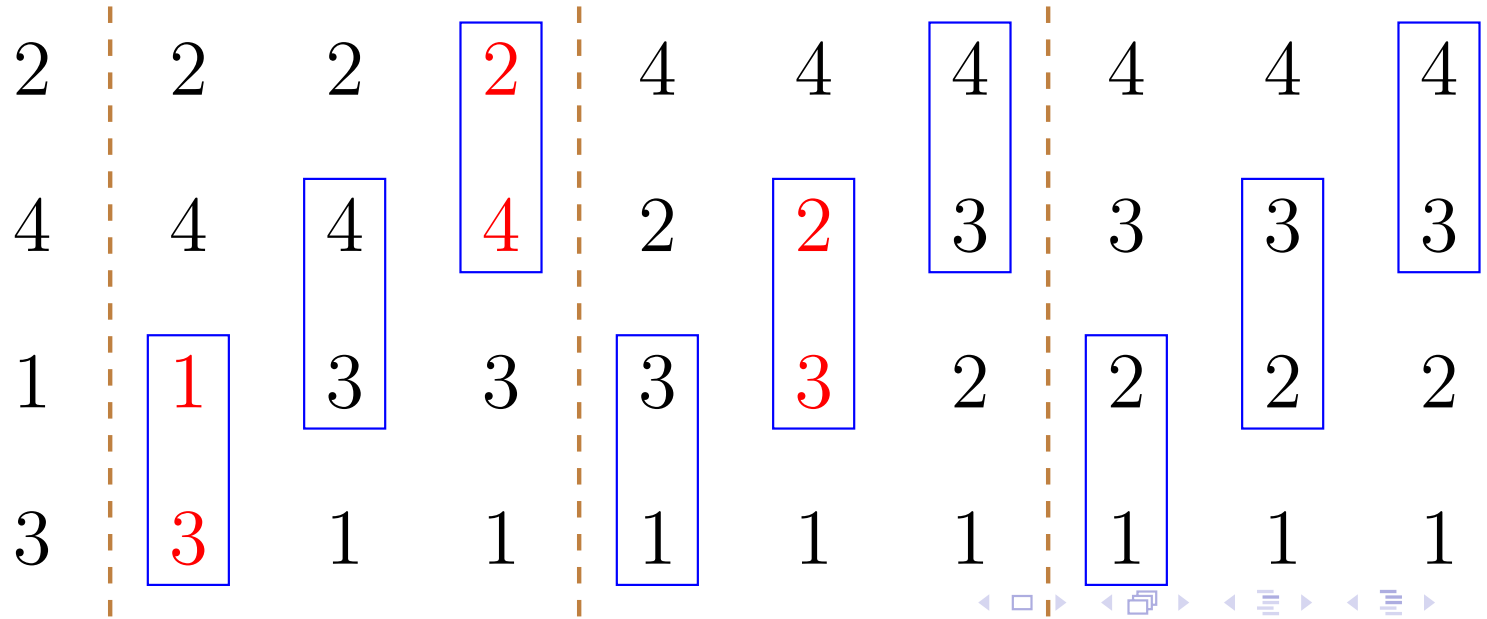
Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```



Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```



Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```

Finiteness!

Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```

Finiteness!

The inner "**for**" loops:

1) $\exists$ loop : no swaps $\Longrightarrow$ swapped = false $\Longrightarrow$ terminates

Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```

Finiteness!

The inner "**for**" loops:

1) $\exists$ loop : no swaps $\Longrightarrow$ swapped = false $\Longrightarrow$ terminates

2) $\forall$ loop : has swaps

Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         for i ← 1 : n − 1 do                ▷ Pick i
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 swapped ← true
8:     until swapped = false                   ▷ No swaps
```

Finiteness!

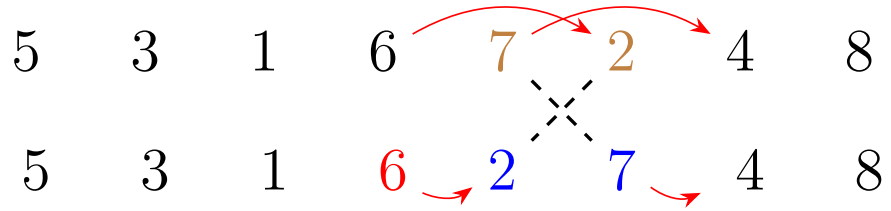
The inner "**for**" loops:

1) $\exists$ loop : no swaps $\Longrightarrow$ swapped = false $\Longrightarrow$ terminates

2) $\forall$ loop : has swaps **Impossible!**

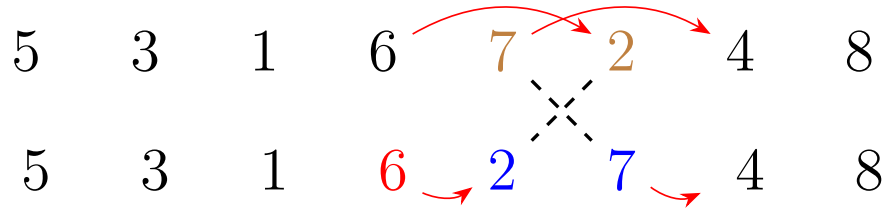
Bubble Sort

Effects of $\textsc{Swap}(a_i, a_{i+1})$ on adjacent inversions:



Bubble Sort

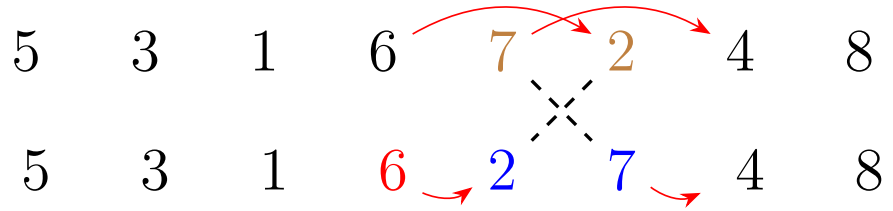
Effects of $\textsc{Swap}(a_i, a_{i+1})$ on adjacent inversions:



$-1 : (7, 2)$

Bubble Sort

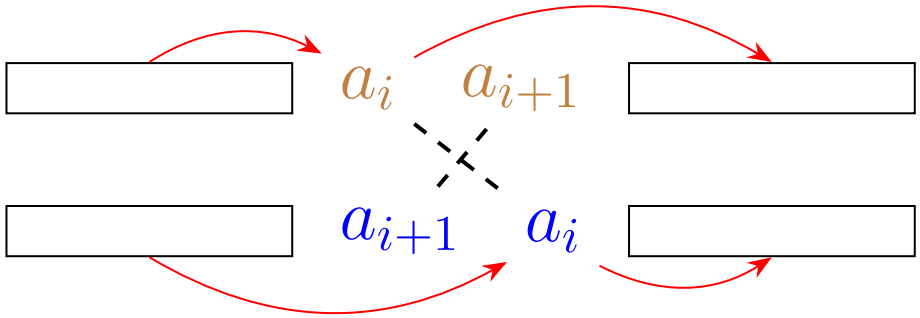
Effects of $\textsc{Swap}(a_i, a_{i+1})$ on adjacent inversions:



$-1 : (7, 2)$ $+2 : (6, 2), (7, 4)$

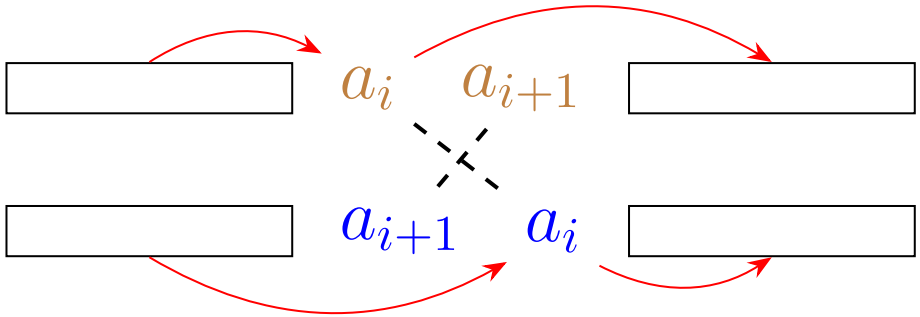
Bubble Sort

Effects of $\textsc{Swap}(a_i, a_{i+1})$ on #inversions:



Bubble Sort

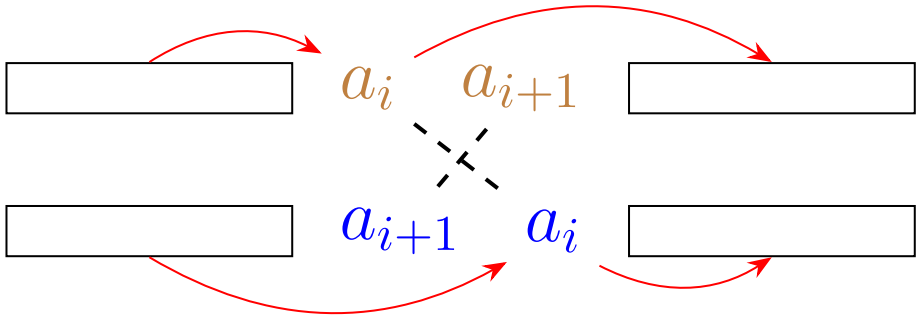
Effects of SWAP(a_i, a_{i+1}) on #inversions:



$-1 : (a_i, a_{i+1})$

Bubble Sort

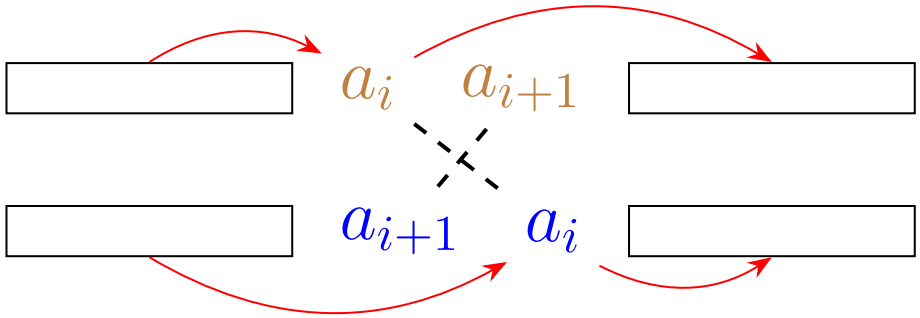
Effects of SWAP(a_i, a_{i+1}) on #inversions:



$-1 : (a_i, a_{i+1})$ $+0$

Bubble Sort

Effects of $\textsc{Swap}(a_i, a_{i+1})$ on #inversions:



$-1 : (a_i, a_{i+1})$ $+0$

Total #inversions is finite.

Optimizing Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     n ← len(A)
3:     repeat
4:         swapped ← false
5:         for i ← 1 : n − 1 do
6:             if a_i > a_{i+1} then
7:                 Swap(a_i, a_{i+1})
8:                 swapped ← true
9:         n ← n − 1                          ▷ One maximal bubbles up
10:    until swapped = false
```

Optimizing Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         swapped ← false
4:         lsp ← 0                          ▷ lsp: the last swap position
5:         for i ← 1 : n − 1 do
6:             if a_i > a_{i+1} then
7:                 Swap(a_i, a_{i+1})
8:                 swapped ← true
9:                 lsp ← i                  ▷ Update lsp
10:        n ← lsp                          ▷ Elements after lsp are sorted
11:    until swapped = false
```

Optimizing Bubble Sort

```
1:  procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:      repeat
3:          swapped ← false
4:          lsp ← 0                          ▷ lsp: the last swap position
5:          for i ← 1 : n − 1 do
6:              if a_i > a_{i+1} then
7:                  Swap(a_i, a_{i+1})
8:                  swapped ← true
9:                  lsp ← i                  ▷ Update lsp
10:         n ← lsp                          ▷ Elements after lsp are sorted
11:     until swapped = false
```

Optimizing Bubble Sort

```
1: procedure BubbleSort(A : a_1 a_2 ··· a_n)
2:     repeat
3:         lsp ← 0                          ▷ lsp: the last swap position
4:         for i ← 1 : n − 1 do
5:             if a_i > a_{i+1} then
6:                 Swap(a_i, a_{i+1})
7:                 lsp ← i                  ▷ Update lsp
8:         n ← lsp                          ▷ Elements after lsp are sorted
9:     until n = 0
```

Bubble Sort

Time Complexity of Bubble Sort

- Finiteness is NOT enough $\Longrightarrow$ Quantitative finiteness

Time Complexity of Bubble Sort

- Finiteness is NOT enough $\Longrightarrow$ Quantitative finiteness
- Time on real computers varies $\Longrightarrow$ #Ops on RAM model:

Time Complexity of Bubble Sort

- Finiteness is NOT enough $\Longrightarrow$ Quantitative finiteness
- Time on real computers varies $\Longrightarrow$ #Ops on RAM model:

$|P|$: #Passes (the "**for**" loops)

$|C|$: #Comparisons (**if** $a_{i-1} > a_i$)

$|S|$: #Swaps ($\textsc{Swap}(a_{i-1}, a_i)$)

Time Complexity of Bubble Sort

- Finiteness is NOT enough $\Longrightarrow$ Quantitative finiteness
- Time on real computers varies $\Longrightarrow$ #Ops on RAM model:

  $|P|$: #Passes (the "**for**" loops)

  $|C|$: #Comparisons (**if** $a_{i-1} > a_i$)

  $|S|$: #Swaps ($\textsc{Swap}(a_{i-1}, a_i)$)
- Different inputs $\Longrightarrow$ different execution time:

Time Complexity of Bubble Sort

- Finiteness is NOT enough $\Longrightarrow$ Quantitative finiteness
- Time on real computers varies $\Longrightarrow$ #Ops on RAM model:

  $|P|$: #Passes (the "**for**" loops)

  $|C|$: #Comparisons (**if** $a_{i-1} > a_i$)

  $|S|$: #Swaps (Swap(a_{i-1}, a_i))

- Different inputs $\Longrightarrow$ different execution time:
  - Best-case, worst-case, and average-case analysis

Best-case and Worst-case Analysis

	Best-case:	Worst-case:
$\lvert P\rvert$	$=($	$);$
$\lvert C\rvert$	$=($	$);$
$\lvert S\rvert$	$=($	$).$

Best-case and Worst-case Analysis

Best-case: 1 2 3 4 5 6 7 8

	Best-case: ascendingly sorted	Worst-case:
$\lvert P\rvert$	$= ($	$);$
$\lvert C\rvert$	$= ($	$);$
$\lvert S\rvert$	$= ($	$).$

Best-case and Worst-case Analysis

Best-case: 1 2 3 4 5 6 7 8

	Best-case: ascendingly sorted	Worst-case:
$\|P\|$	$= (\min: 1,$	$);$
$\|C\|$	$= (\min: n-1,$	$);$
$\|S\|$	$= (\min: 0,$	$).$

Best-case and Worst-case Analysis

Best-case: 1 2 3 4 5 6 7 8

	Best-case: ascendingly sorted	Worst-case: descendingly sorted
$\lvert P\rvert$	$= (\min: 1,$	$);$
$\lvert C\rvert$	$= (\min: n-1,$	$);$
$\lvert S\rvert$	$= (\min: 0,$	$).$

Worst-case: 8 7 6 5 4 3 2 1

Best-case and Worst-case Analysis

Best-case: 1 2 3 4 5 6 7 8

	Best-case: ascendingly sorted	Worst-case: descendingly sorted
$\|P\|$	$= (\min: 1,$	$\max: n);$
$\|C\|$	$= (\min: n-1,$	$\max: \frac{n^2-n}{2});$
$\|S\|$	$= (\min: 0,$	$\max: \frac{n^2-n}{2}).$

Worst-case: 8 7 6 5 4 3 2 1

$|S|$: #Swaps (Average Analysis)

Assumptions on inputs:

1. The input is a random permutation
2. All numbers are distinct

$|S|$: #Swaps (Average Analysis)

Assumptions on inputs:

1. The input is a random permutation
2. All numbers are distinct

$$\boxed{\textsc{Swap}(a_i, a_{i+1}) \implies -1 \text{ inversion}}$$

$|S|$: #Swaps (Average Analysis)

Assumptions on inputs:

1. The input is a random permutation
2. All numbers are distinct

$$\boxed{\textsc{Swap}(a_i, a_{i+1}) \implies -1 \text{ inversion}}$$

Question: What is the expected #inversions?

$|S|$: #Swaps (Average Analysis)

I_{ij} : indicator of inversion (a_i, a_j)

$|S|$: #Swaps (Average Analysis)

I_{ij} : indicator of inversion (a_i, a_j)

$$X = \sum_{j} \sum_{i<j} I_{ij}$$

$|S|$: #Swaps (Average Analysis)

$$I_{ij} : \text{indicator of inversion } (a_i, a_j)$$

$$X = \sum_{j} \sum_{i<j} I_{ij}$$

$$E(X) = E(\sum_{j} \sum_{i<j} I_{ij}) = \sum_{j} \sum_{i<j} E(I_{ij})$$

$|S|$: #Swaps (Average Analysis)

$$I_{ij} : \text{indicator of inversion } (a_i, a_j)$$

$$X = \sum_{j} \sum_{i<j} I_{ij}$$

$$E(X) = E(\sum_{j} \sum_{i<j} I_{ij}) = \sum_{j} \sum_{i<j} E(I_{ij})$$

$$E(I_{ij}) = P\{I_{ij}\} = \frac{1}{2}$$

$|S|$: #Swaps (Average Analysis)

$$I_{ij} : \text{indicator of inversion } (a_i, a_j)$$

$$X = \sum_j \sum_{i<j} I_{ij}$$

$$E(X) = E(\sum_j \sum_{i<j} I_{ij}) = \sum_j \sum_{i<j} E(I_{ij})$$

$$E(I_{ij}) = P\{I_{ij}\} = \frac{1}{2}$$

$$E(X) = \sum_j \sum_{i<j} \frac{1}{2} = \binom{n}{2} \cdot \frac{1}{2} = \frac{n(n-1)}{4}$$