Bubble Sort

(A Taste of Algorithms: Definition, Design, and Analysis)

Hengfeng Wei

Institute of Computer Software
Nanjing University

December 18, 2016

Bubble Sort

Bubble Sort

Sorting

The sorting problem:

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

Output: A sorted (non-decreasing order).

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

Algorithms

Algorithms

Correctness!

Algorithms

Input → **Algorithm** (sequence of operations) → Output

Correctness!

Definiteness: precisely defined operations

Algorithms

Correctness!

Definiteness: precisely defined operations

Finiteness: termination

Algorithms

Input $\longrightarrow$ **Algorithm** (sequence of operations) $\longrightarrow$ Output

Correctness!

Definiteness: precisely defined operations

Finiteness: termination

Effectiveness: a reasonable model; basic operations

Algorithms

Input → **Algorithm** (sequence of operations) → Output

Correctness!

Definiteness: precisely defined operations

Finiteness: termination

Effectiveness: a reasonable model; basic operations

- for sorting: compare, swap

Inversions

$$A = a_1 \quad a_2 \quad \dots \quad a_n.$$

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

$$3 \qquad 1 \qquad 4 \qquad 2$$

Inversions

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

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

$$3 \qquad 1 \qquad 4 \qquad 2$$

Adjacent inversion: $j = i + 1$

Inversions

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

Inversions

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

Inversions

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

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

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

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

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

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

Bubble Sort

Bubble Sort

Basic idea: to eliminate all adjacent inversions

Bubble Sort

Basic idea: to eliminate all adjacent inversions

```
repeat
    pick any i
    if a_i > a_{i+1} then
        Swap(a_i, a_{i+1})
until no adjacent inversions
```

Bubble Sort

Basic idea: to eliminate all adjacent inversions

---

repeat
    pick any i ▷ **Definiteness!**
    if $a_i > a_{i+1}$ **then**
        $\textsc{Swap}(a_i, a_{i+1})$
until no adjacent inversions ▷ **Definiteness!**

Bubble Sort

Basic idea: to eliminate all adjacent inversions

---

repeat
  pick any i ▷ **Definiteness!**
  if $a_i > a_{i+1}$ **then**
    SWAP(a_i, a_{i+1})
until no adjacent inversions ▷ **Finiteness! Definiteness!**

Bubble Sort: Definiteness

```
procedure BubbleSort(A : a_1 a_2 ··· a_n)
    repeat
        
        for            do                                    ▷ Pick i
            if a_i > a_{i+1} then
                Swap(a_i, a_{i+1})
                
    until no adjacent inversions
```

Bubble Sort: Definiteness

```
procedure BubbleSort(A : a_1 a_2 ··· a_n)
    repeat
        [          ]
        for [i ← 1 : n − 1] do                      ▷ Pick i
            if a_i > a_{i+1} then
                Swap(a_i, a_{i+1})
                [          ]
    until [no adjacent inversions]
```

Bubble Sort: Definiteness

```
procedure BubbleSort(A : a_1 a_2 ... a_n)
    repeat
        [swapped]
        for [i ← 1 : n − 1] do                    ▷ Pick i
            if a_i > a_{i+1} then
                Swap(a_i, a_{i+1})
                [        ]
    until [no adjacent inversions]
```

Bubble Sort: Definiteness

```
procedure BubbleSort(A : a_1 a_2 ··· a_n)
    repeat
        [swapped ← false]
        for [i ← 1 : n − 1] do                    ▷ Pick i
            if a_i > a_{i+1} then
                Swap(a_i, a_{i+1})
                [          ]
    until [no adjacent inversions]
```

Bubble Sort: Definiteness

procedure BUBBLESORT($A : a_1\ a_2\ \cdots\ a_n$)
    repeat
        $\boxed{\text{swapped} \leftarrow \text{false}}$
        for $\boxed{i \leftarrow 1 : n-1}$ **do** ▷ Pick i
            if $a_i > a_{i+1}$ **then**
                SWAP(a_i, a_{i+1})
                $\boxed{\text{swapped} \leftarrow \text{true}}$
    until $\boxed{\text{no adjacent inversions}}$

Bubble Sort: Definiteness

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

Bubble Sort: Example

2

4

1

3

Bubble Sort: Example

2	2
4	4
1	1
3	3

Bubble Sort: Example

2	2	2
4	4	4
1	1	3
3	3	1

Bubble Sort: Example

2	2	2	2
4	4	4	4
1	1	3	3
3	3	1	1

Bubble Sort: Example

2	2	2	2	4
4	4	4	4	2
1	1	3	3	3
3	3	1	1	1

Bubble Sort: Example

2	2	2	2	4	4
4	4	4	4	2	2
1	1	3	3	3	3
3	3	1	1	1	1

Bubble Sort: Example

2	2	2	2	4	4	4
4	4	4	4	2	2	3
1	1	3	3	3	3	2
3	3	1	1	1	1	1

Bubble Sort: Example

2	2	2	2	4	4	4	4
4	4	4	4	2	2	3	3
1	1	3	3	3	3	2	2
3	3	1	1	1	1	1	1

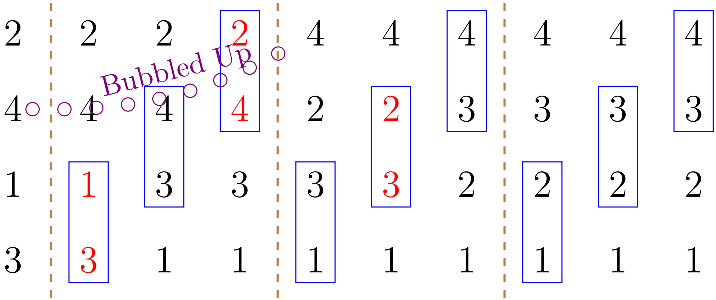
Bubble Sort: Example

2	2	2	2	4	4	4	4	4
4	4	4	4	2	2	3	3	3
1	1	3	3	3	3	2	2	2
3	3	1	1	1	1	1	1	1

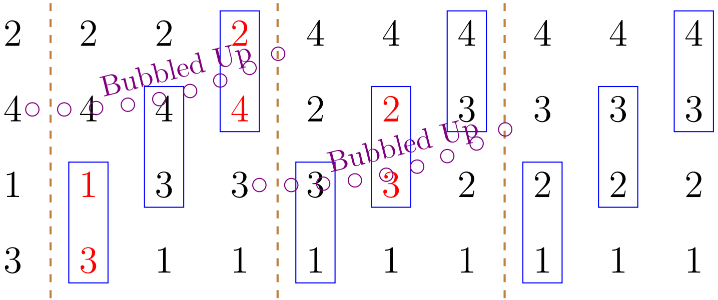
Bubble Sort: Example

2	2	2	2	4	4	4	4	4	4
4	4	4	4	2	2	3	3	3	3
1	1	3	3	3	3	2	2	2	2
3	3	1	1	1	1	1	1	1	1

Bubble Sort: Example



Bubble Sort: Example



Bubble Sort: Finiteness

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

The inner "**for**" loops:

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

Bubble Sort: Finiteness

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

The inner "**for**" loops:

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

2) $\forall$ loop : has swaps

Bubble Sort: Finiteness

procedure BUBBLESORT($A : a_1\ a_2\ \cdots\ a_n$)
    repeat
        swapped $\leftarrow$ false
        for $i \leftarrow 1 : n-1$ **do** ▷ Pick i
            if $a_i > a_{i+1}$ **then**
                SWAP(a_i, a_{i+1})
                swapped $\leftarrow$ true
    until swapped = false ▷ No swaps

The inner "**for**" loops:

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

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

Bubble Sort: Finiteness

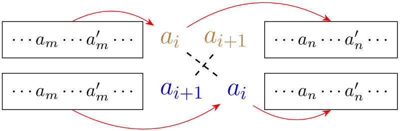
$\boxed{\text{Total \#inversions is finite.}}$

[1] Not on #adjacent inversions!

Bubble Sort: Finiteness

$\boxed{\text{Total \#inversions is finite.}}$

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

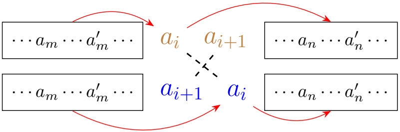


[1] Not on #adjacent inversions!

Bubble Sort: Finiteness

$$\boxed{\text{Total \#inversions is finite.}}$$

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



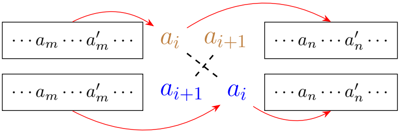
$$-1 : (a_i, a_{i+1})$$

[1] Not on #adjacent inversions!

Bubble Sort: Finiteness

$\boxed{\text{Total \#inversions is finite.}}$

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



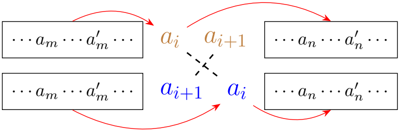
$-1 : (a_i, a_{i+1})$ $\qquad$ $+0$

[1]Not on #adjacent inversions!

Bubble Sort: Finiteness

$$\boxed{\text{Total \#inversions is finite.}}$$

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



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

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

[1] Not on #adjacent inversions!

Bubble Sort: Correctness

Finiteness

Bubble Sort: Correctness

Finiteness $\implies \exists$ loop : no swaps

Bubble Sort: Correctness

$$\textbf{Finiteness} \implies \exists\ \text{loop} : \text{no swaps}$$
$$\implies A \text{ has no adjacent inversions any more}$$

Bubble Sort: Correctness

$$
\begin{aligned}
\textbf{Finiteness} \quad & \Longrightarrow \exists \text{ loop} : \text{ no swaps} \\
& \Longrightarrow A \text{ has no adjacent inversions any more} \\
& \Longrightarrow A \text{ is already sorted.}
\end{aligned}
$$

Optimizing Bubble Sort (I)[2]

Idea: After each "**for**" loop, one more element is settled.

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

[2] See Appendix for "Optimizing Bubble Sort (II)".

Bubble Sort

Time Complexity of Bubble Sort

- Finiteness is NOT enough $\implies$ Quantitative finiteness

Time Complexity of Bubble Sort

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

Time Complexity of Bubble Sort

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

$$|C| : \text{\#Comparisons} \qquad (\textbf{if } a_i > a_{i+1})$$

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

Time Complexity of Bubble Sort

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

$$|C| : \#\text{Comparisons} \qquad (\textbf{if } a_i > a_{i+1})$$

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

- Different inputs $\Longrightarrow$ $|C|$ and $|S|$ vary:

Time Complexity of Bubble Sort

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

$$|C| : \#\text{Comparisons} \qquad (\textbf{if } a_i > a_{i+1})$$

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

- Different inputs $\implies$ $|C|$ and $|S|$ vary:
  - Best-case, worst-case, and average-case analysis

Best-case and Worst-case Analysis

Best-case: Worst-case:

$$
\begin{aligned}
|C| &= (\qquad\qquad\qquad\qquad); \\
|S| &= (\qquad\qquad\qquad\qquad).
\end{aligned}
$$

Best-case and Worst-case Analysis

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

Best-case: ascendingly sorted	Worst-case:

$$|C| = (\qquad);$$

$$|S| = (\qquad).$$

Best-case and Worst-case Analysis

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

$$\begin{aligned} |C| &= \big(\underbrace{\min : n-1}_{\text{Best-case: ascendingly sorted}}, \quad \underbrace{\phantom{\max : \frac{n(n-1)}{2}}}_{\text{Worst-case:}}\big); \\ |S| &= \big(\min : 0, \quad \phantom{\max : \frac{n(n-1)}{2}}\big). \end{aligned}$$

Best-case and Worst-case Analysis

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

$$\overset{\text{Best-case:}}{\overline{\text{ascendingly sorted}}} \qquad \overset{\text{Worst-case:}}{\overline{\text{descendingly sorted}}}$$

$$|C| = (\min : n-1, \qquad);$$

$$|S| = (\min : 0, \qquad).$$

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

$$\begin{aligned} & & \underline{\text{Best-case:}} & & \underline{\text{Worst-case:}} \\ & & \text{ascendingly sorted} & & \text{descendingly sorted} \\ |C| & = (& \min : n-1, & & \max : \tfrac{n^2-n}{2}); \\ |S| & = (& \min : 0, & & \max : \tfrac{n^2-n}{2}). \end{aligned}$$

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

$$\#\text{inversions} = (n-1) + (n-2) + \cdots + 1 = \frac{n^2-n}{2}.$$

$|S|$: #Swaps (Average Analysis)[3]

Assumptions on inputs:

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

[3] The calculation of $|C|$ in average is much more involved.

$|S|$: #Swaps (Average Analysis)[3]

Assumptions on inputs:

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

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

[3] The calculation of $|C|$ in average is much more involved.

$|S|$: #Swaps (Average Analysis)[3]

Assumptions on inputs:

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

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

$$|S| = \mathbb{E}(\#\text{inversions})$$

[3] The calculation of $|C|$ in average is much more involved.

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

$$I_{ij} = \begin{cases} 1 & (a_i, a_j) \text{ is an inversion} \\ 0 & \text{o.w.} \end{cases}$$

$$X = \sum_j \sum_{i<j} I_{ij} \qquad \text{(\#inversions)}$$

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

$$I_{ij} = \begin{cases} 1 & (a_i, a_j) \text{ is an inversion} \\ 0 & \text{o.w.} \end{cases}$$

$$X = \sum_j \sum_{i<j} I_{ij} \qquad (\#\text{inversions})$$

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

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

$$I_{ij} = \begin{cases} 1 & (a_i, a_j) \text{ is an inversion} \\ 0 & \text{o.w.} \end{cases}$$

$$X = \sum_j \sum_{i<j} I_{ij} \qquad (\#\text{inversions})$$

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

$$\mathbb{E}(I_{ij}) = \mathbb{P}\{I_{ij} = 1\} = \frac{1}{2} \qquad (a_i > a_j \lor a_i < a_j)$$

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

$$I_{ij} = \begin{cases} 1 & (a_i, a_j) \text{ is an inversion} \\ 0 & \text{o.w.} \end{cases}$$

$$X = \sum_j \sum_{i<j} I_{ij} \qquad (\#\text{inversions})$$

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

$$\mathbb{E}(I_{ij}) = \mathbb{P}\{I_{ij} = 1\} = \frac{1}{2} \qquad (a_i > a_j \lor a_i < a_j)$$

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

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

$$I_{ij} = \begin{cases} 1 & (a_i, a_j) \text{ is an inversion} \\ 0 & \text{o.w.} \end{cases}$$

$$X = \sum_{j} \sum_{i<j} I_{ij} \qquad (\#\text{inversions})$$

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

$$\mathbb{E}(I_{ij}) = \mathbb{P}\{I_{ij} = 1\} = \frac{1}{2} \qquad (a_i > a_j \vee a_i < a_j)$$

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

Faster Algorithms

> *It took a good deal of work to analyze the bubble sort; and although [...], the results are disappointing since they tell us that the bubble sort isn't really very good at all.*
>
> — *Donald E. Knuth*

Faster Algorithms

> *It took a good deal of work to analyze the bubble sort; and although [...], the results are disappointing since they tell us that the bubble sort isn't really very good at all.*
>
> — *Donald E. Knuth*

$$\text{faster: } O(n^2) \to O(n \lg n)?$$

Faster Algorithms

It took a good deal of work to analyze the bubble sort; and although [...], the results are disappointing since they tell us that the bubble sort isn't really very good at all.

— Donald E. Knuth

$$\text{faster: } O(n^2) \to O(n \lg n)?$$

$$\ldots\text{and faster: } O(n \lg n) \to O(n)?$$

Faster Algorithms

It took a good deal of work to analyze the bubble sort; and although [. . .], the results are disappointing since they tell us that the bubble sort isn't really very good at all.

— *Donald E. Knuth*

$$\text{faster: } O(n^2) \to O(n \lg n)?$$

$$\ldots \text{and faster: } O(n \lg n) \to O(n)?$$

hengxin0912@gmail.com

4. Appendix

Bubble Sort: Correctness

```
procedure BubbleSort(A : a_1 a_2 ··· a_n)
    repeat
        swapped ← false
        for i ← 1 : n − 1 do              ▷ Loop invariant?
            if a_i > a_{i+1} then         ▷ CAS
                Swap(a_i, a_{i+1})
                swapped ← true
    until swapped = false
```

Loop invariant:

Before the k-th ($k \ge 1$) "**for**" loop, $a_{n-(k-1)} \cdots a_n$
(1) consists of the largest $(k-1)$ elements
(2) in sorted order.

Correctness: Initialization + Maintenance + Termination

Bubble Sort: Finiteness

Idea: well-founded relation over $\mathbf{N}$

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

5 3 1 6 7 2 4 8

5 3 1 6 2 7 4 8

Bubble Sort: Finiteness

Idea: well-founded relation over $\mathbf{N}$

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

$$\begin{array}{cccccccc} 5 & 3 & 1 & 6 & 7 & 2 & 4 & 8 \\ 5 & 3 & 1 & 6 & 2 & 7 & 4 & 8 \end{array}$$

$-1 : (7, 2)$

Bubble Sort: Finiteness

Idea: well-founded relation over $\mathbf{N}$

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

$$5 \quad 3 \quad 1 \quad 6 \quad 7 \quad 2 \quad 4 \quad 8$$

$$5 \quad 3 \quad 1 \quad 6 \quad 2 \quad 7 \quad 4 \quad 8$$

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

Optimizing Bubble Sort (II)

Idea: After each "**for**" loop, all elements after "lsp" are settled.

---

procedure BubbleSort($A : a_1\ a_2\ \cdots\ a_n$)
  repeat
    swapped $\leftarrow$ false
    lsp $\leftarrow$ 0 ▷ lsp: the last swap position
    for $i \leftarrow 1 : n-1$ **do**
      if $a_i > a_{i+1}$ **then**
        Swap(a_i, a_{i+1})
        swapped $\leftarrow$ true
        lsp $\leftarrow$ i ▷ Update lsp
    $n \leftarrow$ lsp ▷ Elements after lsp are sorted
  until swapped = false

Optimizing Bubble Sort (II)

Idea: After each "**for**" loop, all elements after "lsp" are settled.

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

Optimizing Bubble Sort (II)

Idea: After each "**for**" loop, all elements after "lsp" are settled.

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