

# Intro to Data Structures & Algorithms

Wagner Truppel  
Lecturer, Dept. of Computer Science & Engineering  
UC Riverside

[wagner@cs.ucr.edu](mailto:wagner@cs.ucr.edu)  
<http://www.cs.ucr.edu/~wagner>

<http://www.cs.ucr.edu/cs14>

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 1

---

---

---

---

---

---

---

---

# Today's Topics

- Sorting (I)
  - ◆ The best sorting algorithm
  - ◆ The worst sorting algorithm
- ◆ The rest...
  - ★ BubbleSort
  - ★ SelectionSort
  - ★ InsertionSort
  - ★ MergeSort

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 2

---

---

---

---

---

---

---

---

# Formal concepts

- Ascending (non-decreasing)
  - ◆ for all  $i$  in  $[0, \text{length}-2]$ :
    - ★  $a[i] \leq a[i+1]$
  - ◆ 2, 7, 9, 9, 14, 27, 31, 31, 56
- Descending (non-increasing)
  - ◆ for all  $i$  in  $[0, \text{length}-2]$ :
    - ★  $a[i] \geq a[i+1]$
  - ◆ 56, 31, 31, 27, 14, 9, 9, 7, 2

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 3

---

---

---

---

---

---

---

---

## Sorting (I)

- Sorting requires additional information
- Comparison/equality operator (==) alone isn't enough
- That is, sorting requires support for an additional operation: an **order** operator (<, >, <=, or >=)
- So sorting applies only to those ADTs which also satisfy the interface of the ADT **Ordered**
- ADT **Ordered** defines one or more operators which lets us order elements

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 4

---

---

---

---

---

---

---

---

## The *best* sorting alg

- The human sorting
  - ◆ 12, 4, 7, 3, 9
  - ◆ We "magically" sort this array by moving each element to its correct place in a single step:
  - ◆ 3, 4, 7, 9, 12
- Time complexity: roughly constant time
- Problems with this approach:
  - ◆ We can do it by eye; the computer can't do it at all
  - ◆ It seems we have some global view of the array; the computer only sees things locally
  - ◆ Even we cannot do it for large arrays

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 5

---

---

---

---

---

---

---

---

## The *worst* sorting alg

- Suppose we have a function `bool isAscending(Array<T> & a)` which returns true if the array is in **ascending** order
- First, how much time does that function take to execute ?
- It should take **O(n)** time
- Here's an implementation
 

```
bool isAscending(Array<T> & a)
{
    for (int i = 0; i < a.length() - 1; i++)
    {
        if (a[i] > a[i + 1])
            return false;
    }
    return true;
}
```

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 6

---

---

---

---

---

---

---

---

### The worst sorting alg

- Another question: how many different possibilities (permutations) are there for an array of  $n$  elements ?
- Example:  $n = 3, a = \{ 3, 2, 7 \}$
- $\{ 2, 3, 7 \}, \{ 2, 7, 3 \}$
- $\{ 3, 2, 7 \}, \{ 3, 7, 2 \}$
- $\{ 7, 2, 3 \}, \{ 7, 3, 2 \}$
- For  $n = 3$  we have 6 permutations
- In general, there are  $n!$  permutations  
[  $n$  factorial =  $n! = n \times (n-1) \times \dots \times 2 \times 1$  ]

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 7

---

---

---

---

---

---

---

---

### The worst sorting alg

- So here's a sorting algorithm:  
while there are permutations  
  produce the next permutation as an array  
  if isAscending(permutation)  
    Break
- How much time does it take ?
- There are  $n!$  permutations, each taking  $O(n)$  time, so the total time complexity is  $O(n \times n!)$  - **exponential** !
- Worse than  $O(n!)$ , which is **really** bad !
- The computer **could** do it, but would take a loooooooooooooong time

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 8

---

---

---

---

---

---

---

---

### Sorting extremes

- So we have two extremes
  - The best sorting algorithm
    - Roughly  $O(1)$ , which is **awesome**
    - But the computer cannot do it
  - The worst sorting algorithm
    - Worse than  $O(n!)$ , which is **horrible**
    - The computer could do it, but it would take too long
- All other sorting algorithms lie in between these extremes

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 9

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- Suppose we want to sort (*ascending*) the array {5,4,1,2,7}
- BubbleSort** selects two adjacent elements at a time and swaps them if they're out of sequence
- This is done for every sequential pair (**inner loop**)
- And then repeated from the start again (**outer loop**)

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 10

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- Outer loop: 0 Inner loop: 0
- Select the pair {5,4}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 11

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- Outer loop: 0 Inner loop: 0
- Select the pair {5,4}
- Its elements are out of sequence, so swap them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 12

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- Outer loop: 0 Inner loop: 1
- Select the pair {5,1}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 13

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- Outer loop: 0 Inner loop: 1
- Select the pair {5,1}
- Its elements are out of sequence, so swap them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 14

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- Outer loop: 0 Inner loop: 2
- Select the pair {5,2}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 15

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- Outer loop: 0 Inner loop: 2
- Select the pair {5,2}
- Its elements are out of sequence, so swap them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 16

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- Outer loop: 0 Inner loop: 3
- Select the pair {5,7}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 17

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- Outer loop: 0 Inner loop: 3
- Select the pair {5,7}
- Its elements are not out of sequence, so don't swap them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 18

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- The element 7 is now in its final position and won't be ever moved again
- Outer loop: 1 Inner loop: 0
- Start over by selecting the pair {4,1}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 19

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- The element 7 is now in its final position and won't be ever moved again
- Outer loop: 1 Inner loop: 0
- Start over by selecting the pair {4,1}
- Its elements are out of sequence, so swap them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 20

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

- The element 7 is now in its final position and won't be ever moved again
- Outer loop: 1 Inner loop: 1
- Select the pair {4,2}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 21

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

7	7	7	7	7	7				
2	2	2	5	5	5				
1	1	5	2	2	2	4	2		
4	5	1	1	1	4	2			
5	4	4	4	4	1	1			

- The element 7 is now in its final position and won't be ever moved again
- Outer loop: 1 Inner loop: 1
- Select the pair {4,2}
- **Its elements are out of sequence, so swap them**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 22

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

7	7	7	7	7	7				
2	2	2	5	5	5				
1	1	5	2	2	2	4			
4	5	1	1	1	4	2			
5	4	4	4	4	1	1			

- The element 7 is now in its final position and won't be ever moved again
- Outer loop: 1 Inner loop: 2
- **Select the pair {4,5}**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 23

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

7	7	7	7	7	7				
2	2	2	5	5	5	5	5		
1	1	5	2	2	2	4	4		
4	5	1	1	1	4	2	2		
5	4	4	4	4	1	1			

- The element 7 is now in its final position and won't be ever moved again
- Outer loop: 1 Inner loop: 2
- Select the pair {4,5}
- **Its elements are not out of sequence, so don't swap them**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 24

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

7	7	7	7	7	7	7			
2	2	2	5	5	5	5			
1	1	5	2	2	2	4	4		
4	5	1	1	1	4	2	2		
5	4	4	4	4	1	1	1		

- The elements 7 and 5 are now in their final positions and won't be ever again
- Outer loop: 2 Inner loop: 0
- Start over by selecting the pair {1,2}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 25

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

7	7	7	7	7	7	7			
2	2	2	5	5	5	5			
1	1	5	2	2	2	4	4		
4	5	1	1	1	4	2	2		
5	4	4	4	4	1	1	1		

- The elements 7 and 5 are now in their final positions and won't be ever again
- Outer loop: 2 Inner loop: 0
- Start over by selecting the pair {1,2}
- Its elements are not out of sequence, so don't swap them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 26

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

7	7	7	7	7	7	7			
2	2	2	5	5	5	5			
1	1	5	2	2	2	4	4		
4	5	1	1	1	4	2	2		
5	4	4	4	4	1	1	1		

- The elements 7 and 5 are now in their final positions and won't be ever again
- Outer loop: 2 Inner loop: 1
- Select the pair {2,4}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 27

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

7	7	7	7	7	7	7	7		
2	2	2	5	5	5	5	5		
1	1	5	2	2	2	4	4	4	
4	5	1	1	1	4	2	2	2	
5	4	4	4	4	1	1	1	1	

- The elements 7 and 5 are now in their final positions and won't be ever again
- Outer loop: 2 Inner loop: 1
- Select the pair {2,4}
- Its elements are not out of sequence, so don't swap them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 28

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

7	7	7	7	7	7	7	7		
2	2	2	5	5	5	5	5		
1	1	5	2	2	2	4	4	4	
4	5	1	1	1	4	2	2	2	
5	4	4	4	4	1	1	1	1	

- The elements 7, 5, and 4 are now in their final positions and won't be ever again
- Outer loop: 3 Inner loop: 0
- Start over by selecting the pair {1,2}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 29

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

7	7	7	7	7	7	7	7		
2	2	2	5	5	5	5	5		
1	1	5	2	2	2	4	4	4	
4	5	1	1	1	4	2	2	2	
5	4	4	4	4	1	1	1	1	

- The elements 7, 5, and 4 are now in their final positions and won't be ever again
- Outer loop: 3 Inner loop: 0
- Select the pair {1,2}
- Its elements are not out of sequence, so don't swap them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 30

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

7	7	7	7	7	7	7	7	7	7
2	2	2	5	5	5	5	5	5	5
1	1	5	2	2	2	4	4	4	4
4	5	1	1	1	4	2	2	2	2
5	4	4	4	4	1	1	1	1	1

- The elements 7, 5, 4, and 2 are now in their final positions and won't be ever again
- Outer loop: 4 Inner loop: 0
- There are no more pairs to select, so the algorithm ends

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 31

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

7	7	7	7	7	7	7	7	7	7
2	2	2	5	5	5	5	5	5	5
1	1	5	2	2	2	4	4	4	4
4	5	1	1	1	4	2	2	2	2
5	4	4	4	4	1	1	1	1	1

- The elements 7, 5, 4, and 2 are now in their final positions and won't be ever again
- Outer loop: 4 Inner loop: 0
- There are no more pairs to select, so the algorithm ends
- Note that the algorithm does **not** end when the array is "obviously sorted"

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 32

---

---

---

---

---

---

---

---

### BubbleSort ('Naïve')

```
int n = a.length();
for (int i = 0; i < n - 1; i++) // outer loop
    for (int j = 0; j < n - 1 - i; j++) // inner loop
        if (a[j+1] < a[j])
            Swap(a[j+1], a[j]);
```

- It's **very** slow...  $O(n^2)$  in the worst case
- With an optimization, it's possible to get  $O(n)$  in the **best** case [ array already sorted ]
- The optimization is: stop when no swaps have occurred for an entire pass (an entire iteration of the *outer* variable i)

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 33

---

---

---

---

---

---

---

---

### BubbleSort ('Smart')

- Swapped is a boolean flag to indicate whether there has been a swap in the current iteration of the outer loop
- The *smart* version of **BubbleSort** starts the same way as the *naïve* version, so I'll skip the steps from slides 10 to 20 - this is slide 21
- The element 7 is now in its final position and won't be ever moved again
- Outer loop: 1 Swapped: **Yes** Inner loop: 1
- Select the pair {4,2}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 34

---

---

---

---

---

---

---

---

---

---

---

---

### BubbleSort ('Smart')

- Swapped is a boolean flag to indicate whether there has been a swap in the current iteration of the outer loop
- The element 7 is now in its final position and won't be ever moved again
- Outer loop: 1 Swapped: **Yes** Inner loop: 1
- Select the pair {4,2}
- Its elements are out of sequence, so swap them**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 35

---

---

---

---

---

---

---

---

---

---

---

---

### BubbleSort ('Smart')

- Swapped is a boolean flag to indicate whether there has been a swap in the current iteration of the outer loop
- The element 7 is now in its final position and won't be ever moved again
- Outer loop: 1 Swapped: **Yes** Inner loop: 2
- Select the pair {4,5}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 36

---

---

---

---

---

---

---

---

---

---

---

---

### BubbleSort ('Smart')

- Swapped is a boolean flag to indicate whether there has been a swap in the current iteration of the outer loop
- The element 7 is now in its final position and won't be ever moved again
- Outer loop: 1 Swapped: **Yes** Inner loop: 2
- Select the pair {4,5}
- Its elements are not out of sequence, so don't swap them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 37

---

---

---

---

---

---

---

---

### BubbleSort ('Smart')

- Swapped is a boolean flag to indicate whether there has been a swap in the current iteration of the outer loop
- The elements 7 and 5 are now in their final positions and won't be ever again
- Outer loop: 2 Swapped: **No** Inner loop: 0
- Start over by selecting the pair {1,2}
- Starting over means to start a new iteration of the outer loop (on the i variable)
- If we detect no swaps through an entire iteration of that variable, we can stop

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 38

---

---

---

---

---

---

---

---

### BubbleSort ('Smart')

- Swapped is a boolean flag to indicate whether there has been a swap in the current iteration of the outer loop
- The elements 7 and 5 are now in their final positions and won't be ever again
- Outer loop: 2 Swapped: **No** Inner loop: 0
- Start over by selecting the pair {1,2}
- Its elements are not out of sequence, so don't swap them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 39

---

---

---

---

---

---

---

---

### BubbleSort ('Smart')

- Swapped is a boolean flag to indicate whether there has been a swap in the current iteration of the outer loop
- The elements 7 and 5 are now in their final positions and won't be ever again
- Outer loop: 2 Swapped: No Inner loop: 1
- Select the pair {2,4}

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 40

---

---

---

---

---

---

---

---

### BubbleSort ('Smart')

- Swapped is a boolean flag to indicate whether there has been a swap in the current iteration of the outer loop
- The elements 7 and 5 are now in their final positions and won't be ever again
- Outer loop: 2 Swapped: No Inner loop: 1
- Select the pair {2,4}
- Its elements are not out of sequence, so don't swap them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 41

---

---

---

---

---

---

---

---

### BubbleSort ('Smart')

- Swapped is a boolean flag to indicate whether there has been a swap in the current iteration of the outer loop
- The elements 7, 5, and 4 are now in their final positions and won't be ever again
- Outer loop: 2 Swapped: No Inner loop: 1
- The naïve version would continue by starting over and selecting the pair {1,2} but...

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 42

---

---

---

---

---

---

---

---

## BubbleSort ('Smart')

- **Swapped** is a boolean flag to indicate whether there has been a swap in the current iteration of the *outer* loop
- The elements **7, 5, and 4** are now in their final positions and won't be ever again
- Outer loop: **2** Swapped: **No** Inner loop: **1**
- ... the smart version realizes that an entire iteration of the *outer* loop went by without any swaps, so it stops
- The *smart* version is **still  $O(n^2)$  in the worst case**
- But it's  **$O(n)$  in the best case**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 43

---

---

---

---

---

---

---

---

## SelectionSort

- Similar to **BubbleSort** but instead of bubbling the largest element towards the end of the current sub-array, we *find* that largest element and just move it to its final position
- It's also **slow...  $O(n^2)$  in the worst case**, but typically much better than **BubbleSort**
- Two variations:
  - ◆ Select the **minimum**
  - ◆ Select the **maximum**
- No conceptual differences between them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 44

---

---

---

---

---

---

---

---

## SelectionSort (Min)

- Suppose again that we want to sort (**ascending**) the array {5,4,1,2,7}
- The **Min** version of **SelectionSort** finds the **minimum** element of the current sub-array and swaps it with the **first** element of the current sub-array
- Then it decreases the size of the current sub-array by 1 and starts again

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 45

---

---

---

---

---

---

---

---

### SelectionSort (Min)

- The current sub-array is the entire initial array, so its size is **5**.
- The algorithm searches for the **min** throughout the **entire current sub-array**, which takes  $O(n)$  time.
- The alg finds that the **minimum** element is **1**.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 46

---

---

---

---

---

---

---

---

### SelectionSort (Min)

- The current sub-array is the entire initial array, so its size is **5**.
- The algorithm searches for the **min** throughout the **entire current sub-array**.
- The alg finds that the **minimum** element is **1**.
- The alg swaps **1** with **5**, since **5** is the **first** element of the current sub-array.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 47

---

---

---

---

---

---

---

---

### SelectionSort (Min)

- The current sub-array has now size **4**.
- The algorithm searches for the **min** throughout the **entire current sub-array**.
- The alg finds that the **minimum** element is **2**.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 48

---

---

---

---

---

---

---

---

### SelectionSort (Min)

- The current sub-array has now size 4.
- The algorithm searches for the **min** throughout the **entire current sub-array**.
- The alg finds that the **minimum** element is **2**.
- *The alg swaps 2 with 4, since 4 is the **first** element of the current sub-array.*

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 49

---

---

---

---

---

---

---

---

### SelectionSort (Min)

- The current sub-array has now size 3.
- *The algorithm searches for the **min** throughout the **entire current sub-array**.*
- *The alg finds that the **minimum** element is 4.*

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 50

---

---

---

---

---

---

---

---

### SelectionSort (Min)

- The current sub-array has now size 3.
- The algorithm searches for the **min** throughout the **entire current sub-array**.
- The alg finds that the **minimum** element is 4.
- *The alg swaps 4 with 5, since 5 is the **first** element of the current sub-array.*

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 51

---

---

---

---

---

---

---

---

### SelectionSort (Min)

- The current sub-array has now size 2.
- The algorithm searches for the **min** throughout the **entire current sub-array**.
- The alg finds that the **minimum** element is 5.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 52

---

---

---

---

---

---

---

---

### SelectionSort (Min)

- The current sub-array has now size 2.
- The algorithm searches for the **min** throughout the **entire current sub-array**.
- The alg finds that the **minimum** element is 5.
- The alg swaps 5 with 5, since 5 is the **first** element of the current sub-array.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 53

---

---

---

---

---

---

---

---

### SelectionSort (Min)

- The current sub-array has now size 1.
- The algorithm searches for the **min** throughout the **entire current sub-array**.
- The alg finds that the **minimum** element is 7.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 54

---

---

---

---

---

---

---

---

### SelectionSort (Min)

7	7	7	7	7					
2	2	4	5	5					
1	5	5	4	4					
4	4	2	2	2					
5	1	1	1	1					

- The current sub-array has now size 1.
- The algorithm searches for the **min** throughout the **entire current sub-array**.
- The alg finds that the **minimum** element is 7.
- *The alg swaps 7 with 7, since 7 is the **first** element of the current sub-array.*
- *The size of the current sub-array would now be decreased to 0, so the algorithm stops.*

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 55

---

---

---

---

---

---

---

---

### SelectionSort (Max)

```

int csas = a.length(); // current sub-array size
while (csas > 0 )
{
    int idx_of_max = 0;
    for (int i = 0; i < csas; i++)
    {
        if (a[i] > a[idx_of_max])
            idx_of_max = i;
    }
    swap(a[idx_of_max], a[csas-1]);
    csas--;
}
    
```

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 56

---

---

---

---

---

---

---

---

### InsertionSort

- **InsertionSort** divides the array in **two** sections: the **sorted** region and the **unsorted** region
- At every step, the algorithm selects the **first** element of the **unsorted** region and **inserts** it in the **correct position** inside the **sorted** region
- Thus, in every step, the **sorted** region **grows** in size by 1 while the **unsorted** region **shrinks** in size by 1
- It's also **slow...  $O(n^2)$  in the worst case.**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 57

---

---

---

---

---

---

---

---

### InsertionSort

Initially, the **unsorted** region is the entire array, **{5,4,1,2,7}**, while the **sorted** region is **empty**.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 58

---

---

---

---

---

---

---

---

### InsertionSort

Initially, the **unsorted** region is the entire array, **{5,4,1,2,7}**, while the **sorted** region is **empty**.

- Select the **first** element of the **unsorted** region, that is, **5**.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 59

---

---

---

---

---

---

---

---

### InsertionSort

Initially, the **unsorted** region is the entire array, **{5,4,1,2,7}**, while the **sorted** region is **empty**.

- Select the **first** element of the **unsorted** region, that is, **5**.
- Insert it at the **proper position** inside the **sorted** region.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 60

---

---

---

---

---

---

---

---

### InsertionSort

- The **unsorted** region is now  $\{4,1,2,7\}$ , while the **sorted** region is  $\{5\}$ .
- Select the **first** element of the **unsorted** region, that is, **4**.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 61

---

---

---

---

---

---

---

---

### InsertionSort

- The **unsorted** region is now  $\{4,1,2,7\}$ , while the **sorted** region is  $\{5\}$ .
- Select the **first** element of the **unsorted** region, that is, **4**.
- *Insert it at the proper position inside the sorted region.*

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 62

---

---

---

---

---

---

---

---

### InsertionSort

- The **unsorted** region is now  $\{1,2,7\}$ , while the **sorted** region is  $\{4,5\}$ .
- Select the **first** element of the **unsorted** region, that is, **1**.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 63

---

---

---

---

---

---

---

---

### InsertionSort

- The **unsorted** region is now  $\{1,2,7\}$ , while the **sorted** region is  $\{4,5\}$ .
- Select the **first** element of the **unsorted** region, that is, 1.
- *Insert it at the proper position inside the sorted region.*

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 64

---

---

---

---

---

---

---

---

### InsertionSort

- The **unsorted** region is now  $\{2,7\}$ , while the **sorted** region is  $\{1,4,5\}$ .
- Select the **first** element of the **unsorted** region, that is, 2.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 65

---

---

---

---

---

---

---

---

### InsertionSort

- The **unsorted** region is now  $\{2,7\}$ , while the **sorted** region is  $\{1,4,5\}$ .
- Select the **first** element of the **unsorted** region, that is, 2.
- *Insert it at the proper position inside the sorted region.*

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 66

---

---

---

---

---

---

---

---

### InsertionSort

- The **unsorted** region is now **{7}**, while the **sorted** region is **{1,2,4,5}**.
- Select the **first** element of the **unsorted** region, that is, **7**.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 67

---

---

---

---

---

---

---

---

### InsertionSort

- The **unsorted** region is now **{7}**, while the **sorted** region is **{1,2,4,5}**.
- Select the **first** element of the **unsorted** region, that is, **7**.
- **Insert it at the proper position inside the sorted region.**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 68

---

---

---

---

---

---

---

---

### InsertionSort

- The **unsorted** region is now **empty**, while the **sorted** region is **{1,2,4,5,7}**.
- Since the **unsorted** region is empty, the algorithm stops.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 69

---

---

---

---

---

---

---

---

## InsertionSort

```

// n = size of the array
// uns_1st= 1st index of unsorted region
// ins_idx = index of where to insert
// uns_next = next item (of some type T)
//           in the unsorted region
for (int uns_1st = 1; uns_1st < n; uns_1st++)
{
    T uns_next = a[uns_1st];
    int ins_idx = uns_1st;
    for (; ins_idx > 0 && a[ins_idx - 1] > uns_next;
        ins_idx--);
    { a[ins_idx] = a[ins_idx - 1]; }
    a[ins_idx] = uns_next;
}
    
```

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 70

---

---

---

---

---

---

---

---

## MergeSort

- **MergeSort** is a *recursive divide-and-conquer* algorithm
- The algorithm
  - ◆ divides the array in half (the *divide* step)
  - ◆ sorts each half separately (the *recursion* step)
  - ◆ then merges the two sorted parts together (the *conquer* step)
- It's *always*  $O[n \log(n)]$ . Fast!
- It requires additional storage. Not so good.
- Merging the two sorted halves is the complicated part of the algorithm.
- I'll let you read this in the textbook.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 71

---

---

---

---

---

---

---

---

## Sorting (I) summary

	Worst case	Average case
<b>BubbleSort</b>	$O[n^2]$	$O[n^2]$
<b>SelectionSort</b>	$O[n^2]$	$O[n^2]$
<b>InsertionSort</b>	$O[n^2]$	$O[n^2]$
<b>MergeSort</b>	$O[n \log(n)]$	$O[n \log(n)]$
<b>HeapSort*</b>	$O[n \log(n)]$	$O[n \log(n)]$
<b>QuickSort**</b>	$O[n^2]$	$O[n \log(n)]$
<b>RadixSort***</b>	$O[n]$	$O[n]$

\* we'll look at the last three algorithms later in the course.  
 \*\* **QuickSort**'s worst-case is very rare and the algorithm is much faster in practice than the others.  
 \*\*\* **RadixSort** is not like the other sorting algorithms, so it's a bit unfair to compare it to them.

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 5 72

---

---

---

---

---

---

---

---

### Sorting (I) summary

- Why is sorting so important?
  - ◆ It's essential for any kind of fast search on large databases
  - ◆ It's often the first step in more complicated algorithms, for example, in *computational geometry* and *scene rendering*
- For java animations of all of these sorting algorithms, point your web browser to <http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/sorting.html>

©2003 W L Truppel      CS 14: Intro. Data Structures & Algs. • Lecture 5      73

---

---

---

---

---

---

---

---