Informatik II

Übung 2

FS 2019

Program Today

1. Feedback of last exercise
2. Repetition Theory
   - Analysis of programs
   - Further sorting algorithms
3. Programming Task
   - Collections in Java

1. Feedback of last exercise

2. Repetition Theory

Analysis

How many calls to `f()`?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

Analysis

How many calls to `f()`?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

The code fragment implies $\Theta(n^2)$ calls to `f()`: the outer loop is executed $n/9$ times and the inner loop contains i calls to `f()`.

Analysis

How many calls to `f()`?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

Analysis

How many calls to `f()`?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

We can ignore the first inner loop because it contains only a constant number of calls to `f()`

Analysis

How many calls to `f()`?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

We can ignore the first inner loop because it contains only a constant number of calls to `f()`
The second inner loop contains $\lfloor \log_2(n) \rfloor + 1$ calls to `f()`. Summing up yields $\Theta(n \log(n))$ calls.

Analysis

How many calls to `f()` in `g(n)`, depending on $n > 0$?

```
void g(int n){
        if (n>1){
                g(n/2);
        }
        else{
                f();
        }
}
```

Analysis

How many calls to `f()` in `g(n)`, depending on $n > 0$?

```
void g(int n){
        if (n>1){
                g(n/2);
        }
        else{
                f();
        }
}
```

There is only a single call to f() at the end of the recursion.

Analysis

How many calls to `f()` in `g(n)`, depending on $n > 0$?

```
void g(int n){
        if (n>1){
                g(n-1);
        }
        f();
}
```

Analysis

How many calls to `f()` in `g(n)`, depending on $n > 0$?

```
void g(int n){
        if (n>1){
                g(n-1);
        }
        f();
}
```

Recurrence

$$T(n) = \begin{cases} T(n-1)+1 & n > 1 \\ 1 & n = 1 \end{cases}$$

Analysis

How many calls to `f()` in `g(n)`, depending on $n > 0$?

```
void g(int n){
        if (n>1){
                g(n−1);
        }
        f();
}
```

Recurrence

$$T(n) = \begin{cases} T(n-1)+1 & n > 1 \\ 1 & n = 1 \end{cases} \qquad \in \Theta(n)$$

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        else{
                f();
        }
}
```

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        else{
                f();
        }
}
```

Recurrence

$$T(n) = \begin{cases} 2T(n/2) & n > 1 \\ 1 & n = 1 \end{cases}$$

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        else{
                f();
        }
}
```

Recurrence

$$T(n) = \begin{cases} 2T(n/2) & n > 1 \\ 1 & n = 1 \end{cases} \qquad \in \Theta(n)$$

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        for (int i = 0; i<n; ++i){
                f();
        }
}
```

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        for (int i = 0; i<n; ++i){
                f();
        }
}
```

Recurrence

$$T(n) = \begin{cases} 2T(n/2) + n & n > 1 \\ 1 & n = 1 \end{cases}$$

Analysis

How many calls to `f()` in `g(n)`, depending on $n = 2^k$?

```
void g(int n){
        if (n>1){
                g(n/2);
                g(n/2);
        }
        for (int i = 0; i<n; ++i){
                f();
        }
}
```

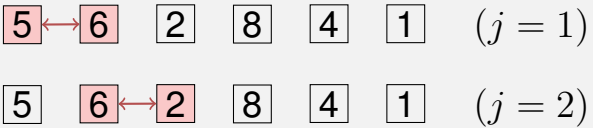
Recurrence

$$T(n) = \begin{cases} 2T(n/2) + n & n > 1 \\ 1 & n = 1 \end{cases} \qquad \in \Theta(n \log n)$$

Bubble Sort

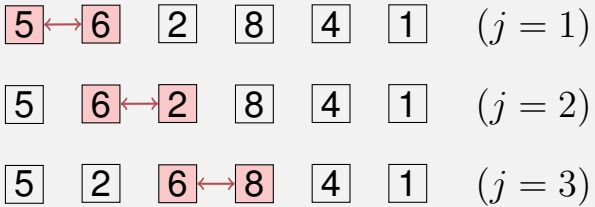
- Swap elements that are not in correct order

Bubble Sort



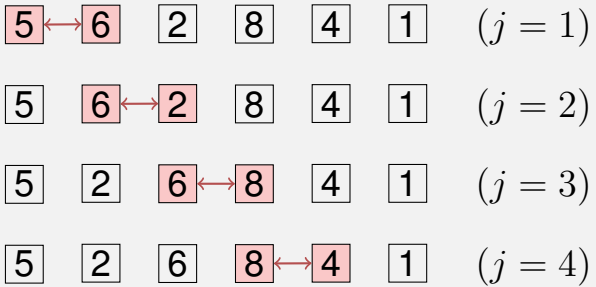
- Swap elements that are not in correct order

Bubble Sort



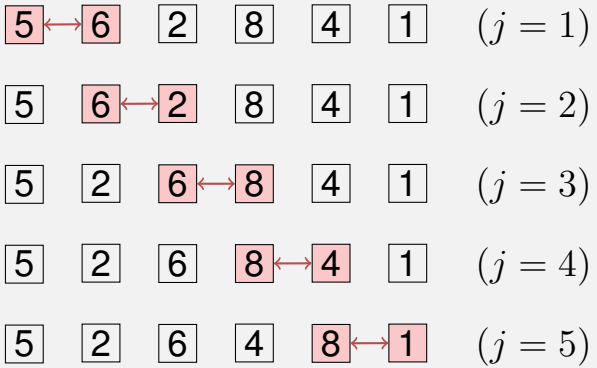
- Swap elements that are not in correct order

Bubble Sort



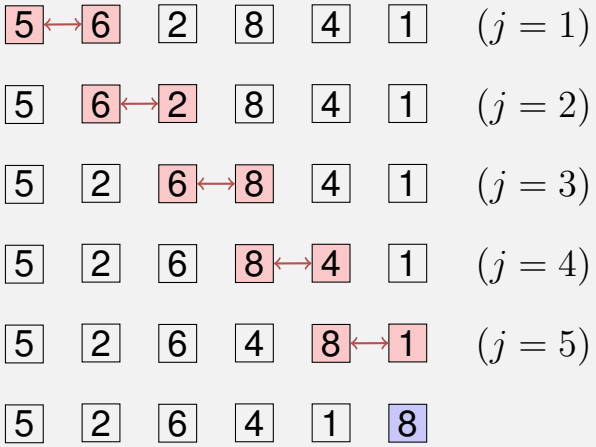
- Swap elements that are not in correct order

Bubble Sort



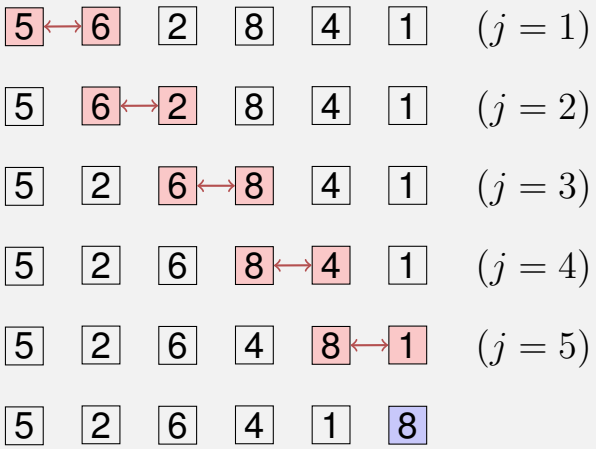
- Swap elements that are not in correct order

Bubble Sort



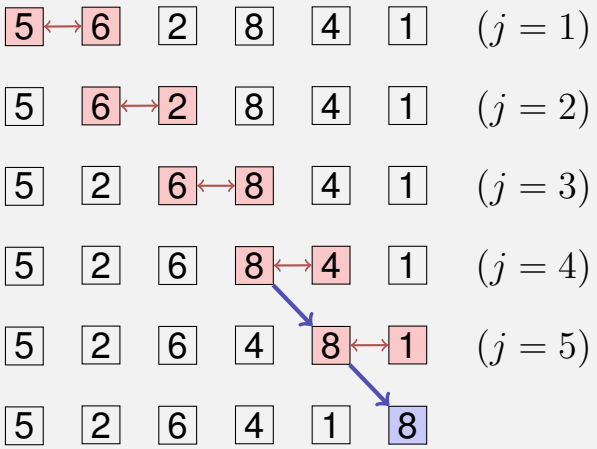
- Swap elements that are not in correct order

Bubble Sort



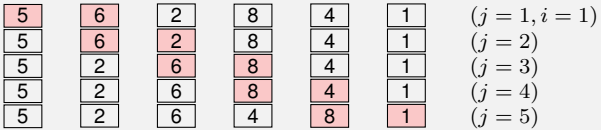
- Swap elements that are not in correct order
- Not sorted! ☹.

Bubble Sort



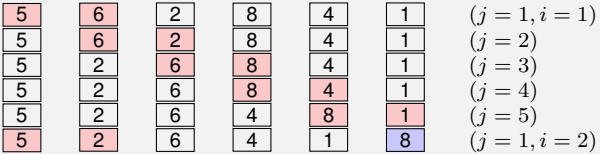
- Swap elements that are not in correct order
- Not sorted! ☹.
- But the greatest element moves to the right
  $\Rightarrow$ Iterate! ☺

Bubble Sort



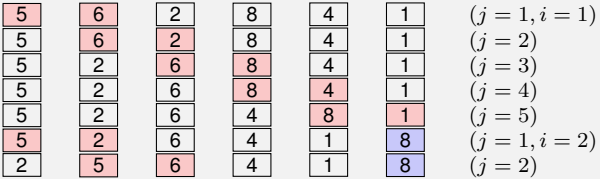
- Apply the procedure iteratively.

Bubble Sort



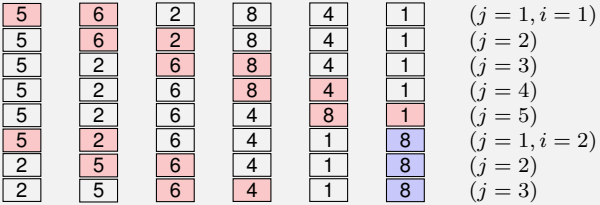
- Apply the procedure iteratively.
- For $A[1, \ldots, n]$,

Bubble Sort



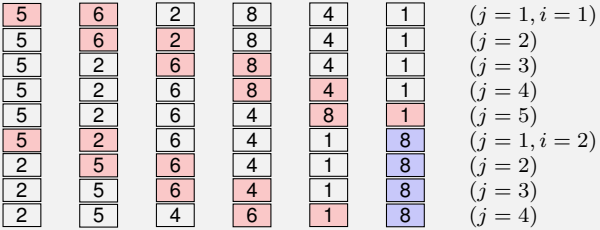
- Apply the procedure iteratively.
- For $A[1, \ldots, n]$, then $A[1, \ldots, n-1]$,

Bubble Sort



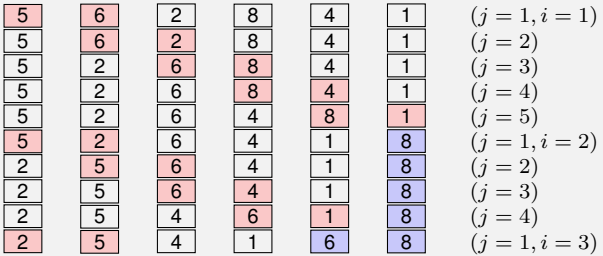
- Apply the procedure iteratively.
- For $A[1, \ldots, n]$, then $A[1, \ldots, n-1]$,

Bubble Sort



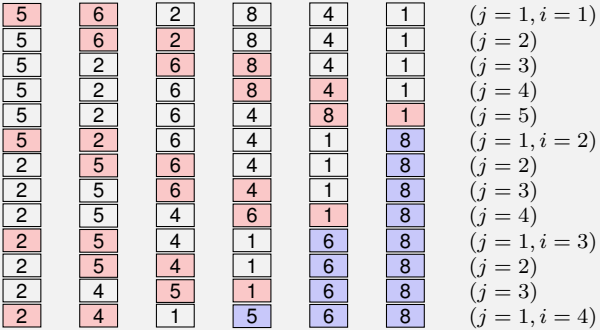
- Apply the procedure iteratively.
- For $A[1, \ldots, n]$, then $A[1, \ldots, n-1]$,

Bubble Sort



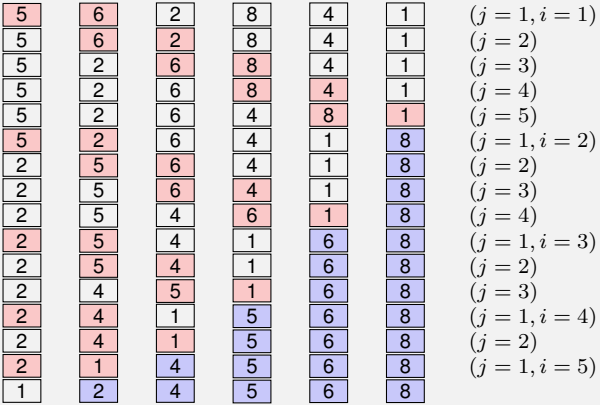
- Apply the procedure iteratively.
- For $A[1, \ldots, n]$, then $A[1, \ldots, n-1]$, then $A[1, \ldots, n-2]$,

Bubble Sort



- Apply the procedure iteratively.
- For $A[1, \ldots, n]$, then $A[1, \ldots, n-1]$, then $A[1, \ldots, n-2]$,

Bubble Sort



- Apply the procedure iteratively.
- For $A[1, \ldots, n]$, then $A[1, \ldots, n-1]$, then $A[1, \ldots, n-2]$, etc.

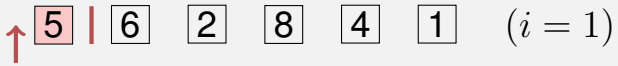
Algorithm: Bubblesort

Input: Array $A = (A[1], \ldots, A[n])$, $n \geq 0$.
Output: Sorted Array A

```
for i ← 1 to n − 1 do
    for j ← 1 to n − i do
        if A[j] > A[j + 1] then
            swap(A[j], A[j + 1]);
```

Insertion Sort

Insertion Sort

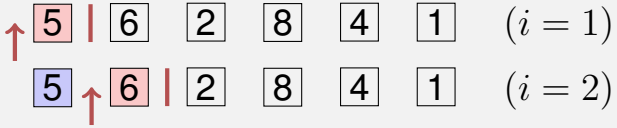


- Iterative procedure:
  $i = 1...n$

Insertion Sort

- Iterative procedure: $i = 1...n$
- Determine insertion position for element i.

Insertion Sort

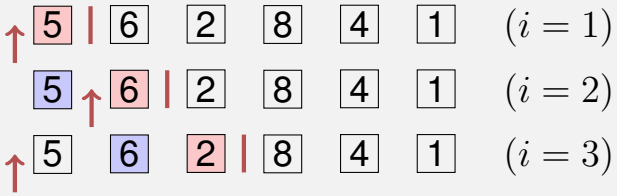


- Iterative procedure: $i = 1...n$
- Determine insertion position for element i.
- Insert element i

Insertion Sort

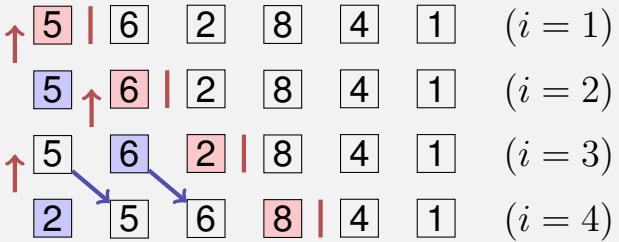
- Iterative procedure: $i = 1...n$
- Determine insertion position for element i.
- Insert element i

Insertion Sort



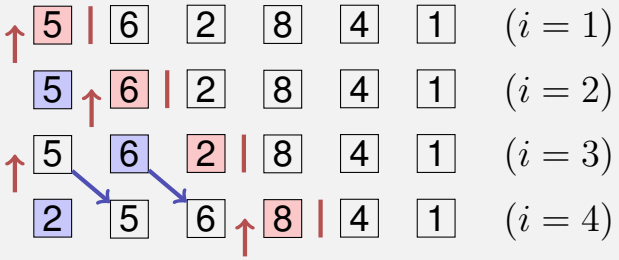
- Iterative procedure: $i = 1...n$
- Determine insertion position for element i.
- Insert element i

Insertion Sort



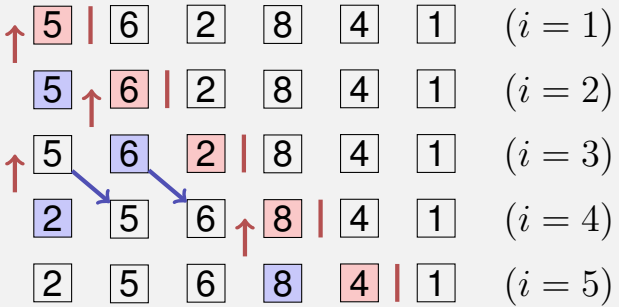
- Iterative procedure: $i = 1...n$
- Determine insertion position for element i.
- Insert element i array block movement potentially required

Insertion Sort



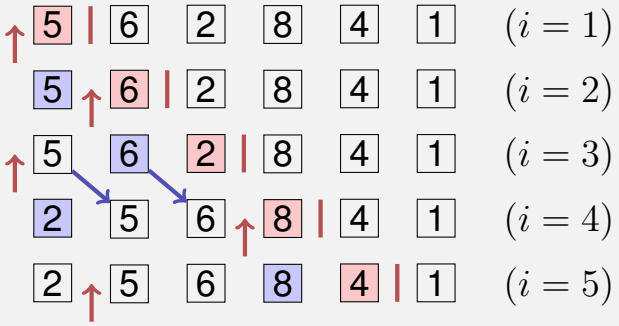
- Iterative procedure: $i = 1...n$
- Determine insertion position for element i.
- Insert element i array block movement potentially required

Insertion Sort



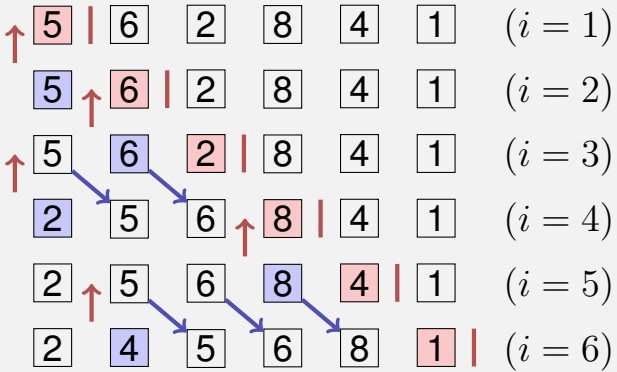
- Iterative procedure: $i = 1...n$
- Determine insertion position for element i.
- Insert element i array block movement potentially required

Insertion Sort



- Iterative procedure: $i = 1...n$
- Determine insertion position for element i.
- Insert element i array block movement potentially required

Insertion Sort

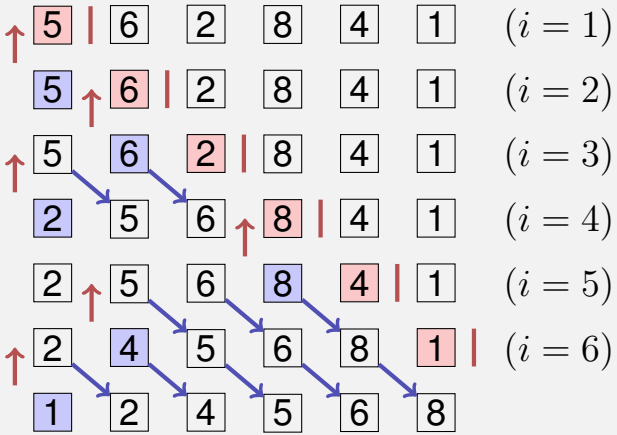


- Iterative procedure: $i = 1...n$
- Determine insertion position for element i.
- Insert element i array block movement potentially required

Insertion Sort

- Iterative procedure: $i = 1...n$
- Determine insertion position for element i.
- Insert element i array block movement potentially required

Insertion Sort



- Iterative procedure: $i = 1...n$
- Determine insertion position for element i.
- Insert element i array block movement potentially required

Insertion Sort

? What is the disadvantage of this algorithm compared to sorting by selection?

Insertion Sort

? What is the disadvantage of this algorithm compared to sorting by selection?

! Many element movements in the worst case.

? What is the advantage of this algorithm compared to selection sort?

Insertion Sort

? What is the disadvantage of this algorithm compared to sorting by selection?

! Many element movements in the worst case.

? What is the advantage of this algorithm compared to selection sort?

! The search domain (insertion interval) is already sorted. Consequently: binary search possible.

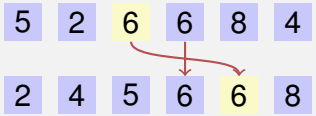
Algorithm: Insertion Sort

Input: Array $A = (A[1], \ldots, A[n])$, $n \geq 0$.
Output: Sorted Array A

```
for i ← 2 to n do
    x ← A[i]
    p ← BinarySearch(A[1...i − 1], x); // Smallest p ∈ [1, i] with A[p] ≥ x
    for j ← i − 1 downto p do
        A[j + 1] ← A[j]
    A[p] ← x
```

Stable and in-situ sorting algorithms

- Stabe sorting algorithms don't change the relative position of two elements. [1]

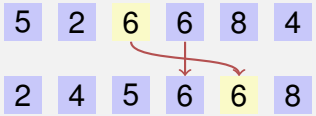


not stable

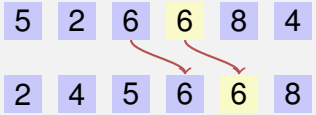
[1] Every sorting algorithm can be made stable by storing the array position together with the element.

Stable and in-situ sorting algorithms

- Stabe sorting algorithms don't change the relative position of two elements. [1]



not stable

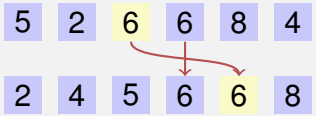


stable

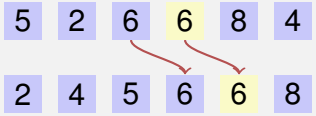
[1] Every sorting algorithm can be made stable by storing the array position together with the element.

Stable and in-situ sorting algorithms

- Stabe sorting algorithms don't change the relative position of two elements. [1]



not stable



stable

- In-situ algorithms require only a constant amount of additional memory. (Mergesort is not in-situ)

[1] Every sorting algorithm can be made stable by storing the array position together with the element.

Quiz: Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort				

Quiz: Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$		

Quiz: Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection				

Quiz: Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$		

Quiz: Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion				

Quiz: Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion	$\Theta(n \log n)$	$\Theta(n \log n)$		

Quiz: Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort				

Quiz: Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$		

Quiz: Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$		

Quiz: Sorting and Running Times

Algorithm	Comparisons		Swaps	
	average	worst	average	worst
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Selection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n)$
Insertion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$

3. Programming Task

Organizing Data

- Data Structures that we know
  - Arrays – Fixed-size sequences
  - Strings – Sequences of characters
  - Linked Lists (up to now: self-made for a fixed element type)
- General Collection Concept in Java
  - ArrayList on generic element types
  - LinkedList, HashMaps, Sets, Maps, ...

Generic List in Java: `java.util.List`

```
import java.util.ArrayList;
import java.util.List;
...

// Liste von Strings
List<String> list = new ArrayList<String>();

list.add("abc");
list.add("xyz");
list.add(1,"123"); // Fuege 123 an Position 1 ein
System.out.println(list.get(0)); // abc

for (String s: list) // For auf Iterator der Liste
       System.out.println(s); // abc 123 xyz
```

Lists of Integers

- Java generics (e.g. collections) can only operate on objects
- Fundamental types `int`, `float` (etc.) are no objects
- java offers wrapper classes for fundemental types, e.g. type `Integer`
- java provides *autoboxing* and automatically wraps a fundamental type into a wrapper class, where necessary.

Lists of Integers

```
import java.util.ArrayList;
import java.util.List;
...

// Lists of integers
List<Integer> list = new ArrayList<Integer>();

list.add(3);
list.add(4);
list.add(1,5); // Fuege 5 an Position 1 ein
System.out.println(list.get(0)); // 3

for (int i: list){ // For auf Iterator der Liste
        System.out.println(i); // 3 5 4
}
```

Questions / Suggestions?