CS1083 Week 4 : Polymorphism

Sorting, Merging

David Bremner

2018-01-22

Reducing disorder: Bubble Sort

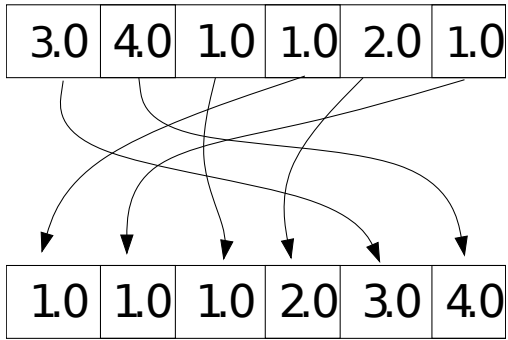
Increasing Order: Selection Sort

Merging

Sorting

Given An array A.

Return Array B, such that each element of B is an element of A (and vice-versa), and $B[i] \leq B[j]$ when $i < j$.



Reducing Disorder: Bubble Sort

Rough Idea

1. Initially $B = A$.
2. At each step, we reduce the number of *bad pairs*:
   - pairs (i, j) such that $i \prec j$ and $B[i] \succ B[j]$
3. When no bad pairs remain, we are done.

$$\begin{array}{ccccccc} 5 & [6 & 1] & 3 & 2 & 4 & 7 \\ 5 & [1 & 6] & 3 & 2 & 4 & 7 \end{array}$$

Reducing Disorder: Bubble Sort

Rough Idea

1. Initially $B = A$.
2. At each step, we reduce the number of *bad pairs*:
   - pairs (i, j) such that $i < j$ and $B[i] > B[j]$
3. When no bad pairs remain, we are done.

$$\begin{array}{ccccccc} 5 & [6 & 1] & 3 & 2 & 4 & 7 \\ 5 & [1 & 6] & 3 & 2 & 4 & 7 \end{array}$$

Reducing Disorder: Bubble Sort

Rough Idea

1. Initially $B = A$.
2. At each step, we reduce the number of *bad pairs*:
   - pairs (i, j) such that $i < j$ and $B[i] > B[j]$
3. When no bad pairs remain, we are done.

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

Reducing Disorder: Bubble Sort

Rough Idea

1. Initially $B = A$.
2. At each step, we reduce the number of *bad pairs*:
   - pairs (i, j) such that $i < j$ and $B[i] > B[j]$
3. When no bad pairs remain, we are done.

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

Reducing Disorder: Bubble Sort

Rough Idea

1. Initially $B = A$.
2. At each step, we reduce the number of *bad pairs*:
   - pairs (i, j) such that $i < j$ and $B[i] > B[j]$
3. When no bad pairs remain, we are done.

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

Checking all pairs is boring

- If $B[i] \leq B[i+1]$ for $0 \leq i < B.length$, then there are no bad pairs.

  Math Note: *This is because $\leq$ is transitive. And needs a proof by induction. See CS1303.*

- So we only need to check pairs $(i, i+1)$.
- Basic operation will be to adjacent elements.
- But notice this may introduce a new bad pair, so we start at the again.

Checking all pairs is boring

- If $B[i] \leq B[i+1]$ for $0 \leq i < B.length$, then there are no bad pairs.

  Math Note: *This is because $\leq$ is transitive. And needs a proof by induction. See CS1303.*

- So we only need to check pairs $(i, i+1)$.
- Basic operation will be to adjacent elements.
- But notice this may introduce a new bad pair, so we start at the again.

Checking all pairs is boring

- If $B[i] \leq B[i+1]$ for $0 \leq i < B.length$, then there are no bad pairs.

  Math Note: *This is because $\leq$ is transitive. And needs a proof by induction. See CS1303.*

- So we only need to check pairs $(i, i+1)$.
- Basic operation will be to adjacent elements.
- But notice this may introduce a new bad pair, so we start at the again.

Checking all pairs is boring

- If $B[i] \leq B[i+1]$ for $0 \leq i < B.length$, then there are no bad pairs.

  Math Note: *This is because $\leq$ is transitive. And needs a proof by induction. See CS1303.*

- So we only need to check pairs $(i, i+1)$.
- Basic operation will be to adjacent elements.
- But notice this may introduce a new bad pair, so we start at the again.

Checking all pairs is boring

- If $B[i] \leq B[i+1]$ for $0 \leq i < B.length$, then there are no bad pairs.

  Math Note: *This is because $\leq$ is transitive. And needs a proof by induction. See CS1303.*

- So we only need to check pairs $(i, i+1)$.
- Basic operation will be to swap adjacent elements.
- But notice this may introduce a new bad pair, so we start at the again.

Checking all pairs is boring

- If $B[i] \leq B[i+1]$ for $0 \leq i < B.length$, then there are no bad pairs.

  Math Note: *This is because $\leq$ is transitive. And needs a proof by induction. See CS1303.*

- So we only need to check pairs $(i, i+1)$.
- Basic operation will be to swap adjacent elements.
- But notice this may introduce a new bad pair, so we start at the again.

Checking all pairs is boring

- If $B[i] \leq B[i+1]$ for $0 \leq i < B.length$, then there are no bad pairs.

  Math Note: *This is because $\leq$ is transitive. And needs a proof by induction. See CS1303.*

- So we only need to check pairs $(i, i+1)$.
- Basic operation will be to swap adjacent elements.
- But notice this may introduce a new bad pair, so we start at the beginning of the array again.

Bubbling in Action

pass 1

5	[6	1]	3	**2**	**4**	**7**
5	1	[	3]	2	4	7
5	1	3	[6	]	4	7
5	1		2	[6	4]	7
5	1	3	2	4	[6	7]

Bubbling in Action

pass 1

5	[6	1]	3	**2**	**4**	**7**
5	1	[6	3]	2	4	7
5	1	3	[6	]	4	7
5	1		2	[6	4]	7
5	1	3	2	4	[6	7]

pass 2

[5	1]	3	**2**	4	6	7
1	[5	3]	2	4	6	7
1	3	[5	2]	4	6	7
1	3	2	[	4]	6	7
1	3	2	4	[5	6]	7

pass 3

1	[3	2]	4	5	6	7
1	2	[3	4]	5	6	7

pass 4

1	2	3	4	5	6	7

Bubbling in Action

pass 1

5	[6	1]	3	**2**	**4**	**7**
5	1	[6	3]	2	4	7
5	1	3	[6	2]	4	7
5	1		2	[6	4]	7
5	1	3	2	4	[6	7]

pass 2

[5	1]	3	**2**	**4**	**6**	**7**
1	[5	3]	2	4	6	7
1	3	[5	2]	4	6	7
1	3	2	[	4]	6	7
1	3	2	4	[5	6]	7

pass 3

1	[3		]	4	5	6	7
1	2	[3	4]	5	6	7

pass 4

1 2 3 4 5 6 7

Bubbling in Action

pass 1

5	[6	1]	3	**2**	**4**	**7**
5	1	[6	3]	2	4	7
5	1	3	[6	2]	4	7
5	1	3	2	[6	4]	7
5	1	3	2	4	[6	7]

pass 2

[5	1]	3	**2**	**4**	**6**	**7**
1	[5	3]	2	4	6	7
1	3	[5	2]	4	6	7
1	3	2	[	4]	6	7
1	3	2	4	[5	6]	7

pass 3

1	[3	]	4	5	6	7
1	2	[3	4]	5	6	7

pass 4

1	2	3	4	5	6	7

Bubbling in Action

pass 1

$$
\begin{array}{ccccccc}
5 & [6 & 1] & 3 & \mathbf{2} & \mathbf{4} & \mathbf{7} \\
5 & 1 & [6 & 3] & 2 & 4 & 7 \\
5 & 1 & 3 & [6 & 2] & 4 & 7 \\
5 & 1 & 3 & 2 & [6 & 4] & 7 \\
5 & 1 & 3 & 2 & 4 & [6 & 7]
\end{array}
$$

pass 2

$$
\begin{array}{ccccccc}
[5 & 1] & 3 & \mathbf{2} & \mathbf{4} & \mathbf{6} & \mathbf{7} \\
1 & [5 & 3] & 2 & 4 & 6 & 7 \\
1 & 3 & [5 & 2] & 4 & 6 & 7 \\
1 & 3 & 2 & [& 4] & 6 & 7 \\
1 & 3 & 2 & 4 & [5 & 6] & 7
\end{array}
$$

pass 3

$$
\begin{array}{ccccccc}
1 & [3 &] & 4 & \mathbf{5} & \mathbf{6} & \mathbf{7} \\
1 & 2 & [3 & 4] & 5 & 6 & 7
\end{array}
$$

pass 4

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

Bubbling in Action

pass 1

$$
\begin{array}{ccccccc}
5 & [6 & 1] & 3 & \mathbf{2} & \mathbf{4} & \mathbf{7} \\
5 & 1 & [6 & 3] & 2 & 4 & 7 \\
5 & 1 & 3 & [6 & 2] & 4 & 7 \\
5 & 1 & 3 & 2 & [6 & 4] & 7 \\
5 & 1 & 3 & 2 & 4 & [6 & 7]
\end{array}
$$

pass 2

$$
\begin{array}{ccccccc}
[5 & 1] & 3 & \mathbf{2} & \mathbf{4} & \mathbf{6} & \mathbf{7} \\
1 & [5 & 3] & 2 & 4 & 6 & 7 \\
1 & 3 & [5 & 2] & 4 & 6 & 7 \\
1 & 3 & 2 & [5 & 4] & 6 & 7 \\
1 & 3 & 2 & 4 & [5 & 6] & 7
\end{array}
$$

pass 3

$$
\begin{array}{ccccccc}
1 & [3 &] & 4 & \mathbf{5} & \mathbf{6} & \mathbf{7} \\
1 & 2 & [3 & 4] & 5 & 6 & 7
\end{array}
$$

pass 4

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

Bubbling in Action

pass 1

5	[6	1]	3	**2**	**4**	**7**
5	1	[6	3]	2	4	7
5	1	3	[6	2]	4	7
5	1	3	2	[6	4]	7
5	1	3	2	4	[6	7]

pass 2

[5	1]	3	**2**	**4**	**6**	**7**
1	[5	3]	2	4	6	7
1	3	[5	2]	4	6	7
1	3	2	[5	4]	6	7
1	3	2	4	[5	6]	7

pass 3

1	[3	]	**4**	**5**	**6**	**7**
1	2	[3	4]	5	6	7

pass 4

1	2	3	4	5	6	7

Bubbling in Action

pass 1

$5\ [6\ 1]\ 3\ \mathbf{2}\ \mathbf{4}\ \mathbf{7}$
$5\ 1\ [6\ 3]\ 2\ 4\ 7$
$5\ 1\ 3\ [6\ 2]\ 4\ 7$
$5\ 1\ 3\ 2\ [6\ 4]\ 7$
$5\ 1\ 3\ 2\ 4\ [6\ 7]$

pass 2

$[5\ 1]\ 3\ \mathbf{2}\ \mathbf{4}\ \mathbf{6}\ \mathbf{7}$
$1\ [5\ 3]\ 2\ 4\ 6\ 7$
$1\ 3\ [5\ 2]\ 4\ 6\ 7$
$1\ 3\ 2\ [5\ 4]\ 6\ 7$
$1\ 3\ 2\ 4\ [5\ 6]\ 7$

pass 3

$1\ [3\ \mathbf{2}]\ \mathbf{4}\ \mathbf{5}\ \mathbf{6}\ \mathbf{7}$
$1\ 2\ [3\ 4]\ 5\ 6\ 7$

pass 4

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

Bubbling in Action

pass 1

$$
\begin{array}{ccccccc}
5 & [6 & 1] & 3 & \mathbf{2} & \mathbf{4} & \mathbf{7} \\
5 & 1 & [6 & 3] & 2 & 4 & 7 \\
5 & 1 & 3 & [6 & 2] & 4 & 7 \\
5 & 1 & 3 & 2 & [6 & 4] & 7 \\
5 & 1 & 3 & 2 & 4 & [6 & 7]
\end{array}
$$

pass 2

$$
\begin{array}{ccccccc}
[5 & 1] & 3 & \mathbf{2} & \mathbf{4} & \mathbf{6} & \mathbf{7} \\
1 & [5 & 3] & 2 & 4 & 6 & 7 \\
1 & 3 & [5 & 2] & 4 & 6 & 7 \\
1 & 3 & 2 & [5 & 4] & 6 & 7 \\
1 & 3 & 2 & 4 & [5 & 6] & 7
\end{array}
$$

pass 3

$$
\begin{array}{ccccccc}
1 & [3 & \mathbf{2}] & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} \\
1 & 2 & [3 & 4] & 5 & 6 & 7
\end{array}
$$

pass 4

$$
\mathbf{1}\ \mathbf{2}\ \mathbf{3}\ \mathbf{4}\ \mathbf{5}\ \mathbf{6}\ \mathbf{7}
$$

Are we making progress or just bubbles?

There are two ways of looking at bubble-sort's progress.

1. After each pass, one element is bubbled into its correct place.
2. After each pass, the sorted sequence at the *end* of the array gets one longer.

$$
\begin{array}{ccccccc}
5 & [6 & 1] & 3 & 2 & 4 & 7 \\
5 & 1 & [& 3] & 2 & 4 & 7 \\
5 & 1 & 3 & [6 & 2] & 4 & 7 \\
5 & 1 & 3 & 2 & [6 & 4] & 7 \\
5 & 1 & 3 & 2 & 4 & [6 & 7]
\end{array}
$$

Are we making progress or just bubbles?

There are two ways of looking at bubble-sort's progress.

1. After each pass, one element is bubbled into its correct place.
2. After each pass, the sorted sequence at the *end* of the array gets one longer.

$$
\begin{array}{ccccccc}
5 & [6 & 1] & 3 & \mathbf{2} & 4 & \mathbf{7} \\
5 & 1 & [6 & 3] & \mathbf{2} & 4 & \mathbf{7} \\
5 & 1 & 3 & [6 & \mathbf{2}] & 4 & \mathbf{7} \\
5 & 1 & 3 & \mathbf{2} & [6 & 4] & \mathbf{7} \\
5 & 1 & 3 & \mathbf{2} & 4 & [6 & \mathbf{7}]
\end{array}
$$

Are we making progress or just bubbles?

There are two ways of looking at bubble-sort's progress.

1. After each pass, one element is bubbled into its correct place.
2. After each pass, the sorted sequence at the *end* of the array gets one longer.

$$
\begin{array}{ccccccc}
5 & [6 & 1] & 3 & \mathbf{2} & 4 & \mathbf{7} \\
5 & 1 & [6 & 3] & \mathbf{2} & 4 & \mathbf{7} \\
5 & 1 & 3 & [6 & \mathbf{2}] & 4 & \mathbf{7} \\
5 & 1 & 3 & \mathbf{2} & [6 & 4] & \mathbf{7} \\
5 & 1 & 3 & \mathbf{2} & 4 & [6 & \mathbf{7}]
\end{array}
$$

Are we making progress or just bubbles?

There are two ways of looking at bubble-sort's progress.

1. After each pass, one element is bubbled into its correct place.
2. After each pass, the sorted sequence at the *end* of the array gets one longer.

$$
\begin{array}{ccccccc}
5 & [6 & 1] & 3 & \mathbf{2} & 4 & \mathbf{7} \\
5 & 1 & [6 & 3] & \mathbf{2} & 4 & \mathbf{7} \\
5 & 1 & 3 & [6 & \mathbf{2}] & 4 & \mathbf{7} \\
5 & 1 & 3 & \mathbf{2} & [6 & 4] & \mathbf{7} \\
5 & 1 & 3 & \mathbf{2} & 4 & [\mathbf{6} & \mathbf{7}]
\end{array}
$$

Swapping elements

```
public void swap(T[] a,
                 int i, int j){
    T temp = a[i];


}
```

BubbleSort

Swapping elements

```
public void swap(T[] a,
                 int i, int j){
    T temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}
```

Implementing Bubble Sort

BubbleSort

```
public void sort(T[] a){
    boolean finished=false;
    while(!finished){
        finished=true; // assume last pass









    }
}
```

Implementing Bubble Sort

BubbleSort

```
public void sort(T[] a){
    boolean finished=false;
    while(!finished){
        finished=true; // assume last pass
        for (int i=0; i<a.length-1; i++){




        }
    }
}
```

Implementing Bubble Sort

BubbleSort

```
public void sort(T[] a){
    boolean finished=false;
    while(!finished){
        finished=true; // assume last pass
        for (int i=0; i<a.length-1; i++){
            if (a[i].compareTo(a[i+1]) > 0){
                swap(a,i,i+1);
                finished=false;
            }
        }
    }
}
```

Analyzing Bubblesort

- What is the minimum number of passes?
- When does this make a difference?
- How many “steps” (comparisons) in each pass?

Using Bubble Sort for Strings

```
String[] names = {"Harry", "Tom",
                  "Moe", "Curly"};

BubbleSort<String> stringSorter=
    new BubbleSort<String>();

stringSorter.sort(names);

for (String name : names) {
    System.out.println(name);
}
```

BubbleSort

Using Bubble Sort for Integers

```
BubbleSort<Integer> integerSorter=
    new BubbleSort<Integer>();

Integer [] numbers = { 8, 4, 5, 2, 9, 1,
                       -3, 0, 1, 2, 3 };

integerSorter.sort(numbers);

for (Integer number : numbers) {
    System.out.println(number);
}
```

BubbleSort

Reducing disorder: Bubble Sort

Increasing Order: Selection Sort

Merging

Selection Sort

Idea

- Divide the array into two parts
- Left part is sorted
- Each pass extends the sorted part by one.

Improvement

Partition everything in the left part is smaller than everything in the unsorted part.

Selection each pass consists of *swapping* the smallest element in the unsorted part to the end of the sorted part.

Selection Sort in Action

$\|5$	7	9	1	4	8	6	2	10	3
1	$\|7$	9		4	8	6	2	10	3
1	2	$\|9$	5	4	8	6		10	3
1	2	3	$\|5$	4	8	6	7	10	
1	2	3		$\|5$	8	6	7	10	9
1	2	3	4		$\|8$	6	7	10	9
1	2	3	4	5	6	$\|$	7	10	9
1	2	3	4	5	6	7	$\|8$	10	9
1	2	3	4	5	6	7	8	$\|10$	9
1	2	3	4	5	6	7	8	9	10

- How many passes?
- How many comparisons in each pass? Minimum? Maximum?

Selection Sort in Action

$\|5$	7	9	1	4	8	6	2	10	3
1	$\|7$	9	5	4	8	6	2	10	3
1	2	$\|9$	5	4	8	6		10	3
1	2	3	$\|5$	4	8	6	7	10	
1	2	3		$\|5$	8	6	7	10	9
1	2	3	4		$\|8$	6	7	10	9
1	2	3	4	5	6	$\|$	7	10	9
1	2	3	4	5	6	7	$\|8$	10	9
1	2	3	4	5	6	7	8	$\|10$	9
1	2	3	4	5	6	7	8	9	10

- How many passes?
- How many comparisons in each pass? Minimum? Maximum?

Selection Sort in Action

$\|5$	7	9	1	4	8	6	2	10	3
1	$\|7$	9	5	4	8	6	2	10	3
1	2	$\|9$	5	4	8	6	7	10	3
1	2	3	$\|5$	4	8	6	7	10	
1	2	3		$\|5$	8	6	7	10	9
1	2	3	4		$\|8$	6	7	10	9
1	2	3	4	5	6	$\|$	7	10	9
1	2	3	4	5	6	7	$\|8$	10	9
1	2	3	4	5	6	7	8	$\|10$	9
1	2	3	4	5	6	7	8	9	10

- How many passes?
- How many comparisons in each pass? Minimum? Maximum?

Selection Sort in Action

$\|5$	7	9	1	4	8	6	2	10	3
1	$\|7$	9	5	4	8	6	2	10	3
1	2	$\|9$	5	4	8	6	7	10	3
1	2	3	$\|5$	4	8	6	7	10	9
1	2	3		$\|5$	8	6	7	10	9
1	2	3	4		$\|8$	6	7	10	9
1	2	3	4	5	6	$\|$	7	10	9
1	2	3	4	5	6	7	$\|8$	10	9
1	2	3	4	5	6	7	8	$\|10$	9
1	2	3	4	5	6	7	8	9	10

- How many passes?
- How many comparisons in each pass? Minimum? Maximum?

Selection Sort in Action

‖5	7	9	1	4	8	6	2	10	3
1	‖7	9	5	4	8	6	2	10	3
1	2	‖9	5	4	8	6	7	10	3
1	2	3	‖5	4	8	6	7	10	9
1	2	3	4	‖5	8	6	7	10	9
1	2	3	4		‖8	6	7	10	9
1	2	3	4	5	6	‖	7	10	9
1	2	3	4	5	6	7	‖8	10	9
1	2	3	4	5	6	7	8	‖10	9
1	2	3	4	5	6	7	8	9	10

- How many passes?
- How many comparisons in each pass? Minimum? Maximum?

Selection Sort in Action

$\|5$	7	9	1	4	8	6	2	10	3
1	$\|7$	9	5	4	8	6	2	10	3
1	2	$\|9$	5	4	8	6	7	10	3
1	2	3	$\|5$	4	8	6	7	10	9
1	2	3	4	$\|5$	8	6	7	10	9
1	2	3	4	5	$\|8$	6	7	10	9
1	2	3	4	5	6	$\|$	7	10	9
1	2	3	4	5	6	7	$\|8$	10	9
1	2	3	4	5	6	7	8	$\|10$	9
1	2	3	4	5	6	7	8	9	10

- How many passes?
- How many comparisons in each pass? Minimum? Maximum?

Selection Sort in Action

$\|5$	7	9	1	4	8	6	2	10	3
1	$\|7$	9	5	4	8	6	2	10	3
1	2	$\|9$	5	4	8	6	7	10	3
1	2	3	$\|5$	4	8	6	7	10	9
1	2	3	4	$\|5$	8	6	7	10	9
1	2	3	4	5	$\|8$	6	7	10	9
1	2	3	4	5	6	$\|8$	7	10	9
1	2	3	4	5	6	7	$\|8$	10	9
1	2	3	4	5	6	7	8	$\|10$	9
1	2	3	4	5	6	7	8	9	10

- How many passes?
- How many comparisons in each pass? Minimum? Maximum?

Finding minimum position

SelectionSort

```
public int minimumPosition(T [] a,
                           int from)  {
    int minPos = from;




    return minPos;
}
```

How many steps to find smallest element after position `from`?

Finding minimum position

SelectionSort

```
public int minimumPosition(T [] a,
                           int from)  {
    int minPos = from;
    for (int i = from + 1; i < a.length; i++){


    }
    return minPos;
}
```

How many steps to find smallest element after position `from`?

Finding minimum position

SelectionSort

```
public int minimumPosition(T [] a,
                           int from)  {
    int minPos = from;
    for (int i = from + 1; i < a.length; i++){
        if (a[i].compareTo(a[minPos]) <0)
            minPos = i;
    }
    return minPos;
}
```

How many steps to find smallest element after position `from`?

Implimenting Selection Sort

SelectionSort

```
public void sort(T [] a)
{
    for (int n = 0; n < a.length - 1; n++) {




    }
}
```

Implimenting Selection Sort

SelectionSort

```
public void sort(T [] a)
{
    for (int n = 0; n < a.length - 1; n++) {
        int minPos = minimumPosition(a, n);
        if (minPos != n){
            // swap



        }
    }
}
```

Implimenting Selection Sort

SelectionSort

```
public void sort(T [] a)
{
    for (int n = 0; n < a.length - 1; n++) {
        int minPos = minimumPosition(a, n);
        if (minPos != n){
            // swap
            T temp=a[n];
            a[n]=a[minPos];
            a[minPos]=temp;
        }
    }
}
```

Implimenting Selection Sort

SelectionSort

```
public void sort(T [] a)
{
    for (int n = 0; n < a.length - 1; n++) {
        int minPos = minimumPosition(a, n);
        if (minPos != n){
            // swap
            T temp=a[n];
            a[n]=a[minPos];
            a[minPos]=temp;
        }
    }
}
```

Implimenting Selection Sort

SelectionSort

```
public void sort(T [] a)
{
    for (int n = 0; n < a.length - 1; n++) {
        int minPos = minimumPosition(a, n);
        if (minPos != n){
            // swap
            T temp=a[n];
            a[n]=a[minPos];
            a[minPos]=temp;
        }
    }
}
```

Using Selection Sort for Strings

```
String[] names = {"Harry", "Tom",
                  "Moe", "Curly"};

SelectionSort<String> stringSorter=
    new SelectionSort<String>();

stringSorter.sort(names);

for (String name : names) {
    System.out.println(name);
}
```

SelectionSort

Using Selection Sort for Integers

```
SelectionSort<Integer> integerSorter=
    new SelectionSort<Integer>();

Integer [] numbers = { 8, 4, 5, 2, 9, 1,
                       -3, 0, 1, 2, 3 };

integerSorter.sort(numbers);

for (Integer number : numbers) {
    System.out.println(number);
}
```

SelectionSort

Sorting Custom Comparables

```
SelectionSort<IntPair> pairSorter=
    new SelectionSort<IntPair>();
IntPair [] pairs = { new IntPair(8, 4), new IntPair(5, 2),
                     new IntPair(9, 1), new IntPair(-3, 0),
                     new IntPair(1, 2) };
pairSorter.sort(pairs);
for (IntPair pair : pairs) {
    System.out.println(pair);
}
```

SelectionSort

Reducing disorder: Bubble Sort

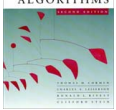
Increasing Order: Selection Sort

Merging

Merging two sorted arrays

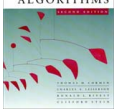
```
public void merge(T [] a, T[] b, T[] c) {
    int i=0, j=0;
    while (i + j < a.length + b.length) {
        T next;
        if (i >= a.length)
            next=b[j++];
        else if (j >= b.length)
            next=a[i++];
        else
            if (a[i].compareTo(b[j]) <= 0)
                next=a[i++];
            else
                next=b[j++];
        c[i+j-1]=next;
    }
}
```

ArrayMerger



Merging two sorted arrays

20	12
13	11
7	9
2	1



Merging two sorted arrays

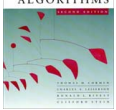
20 12

13 11

7 9

2 1

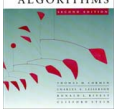
1



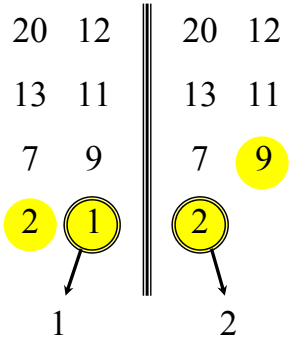
Merging two sorted arrays

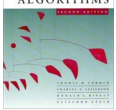
20	12		20	12
13	11		13	11
7	9		7	9
2	1		2	

1

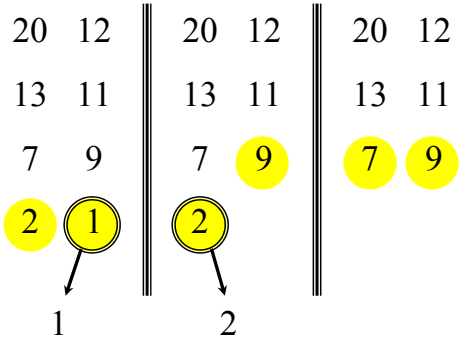


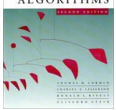
Merging two sorted arrays



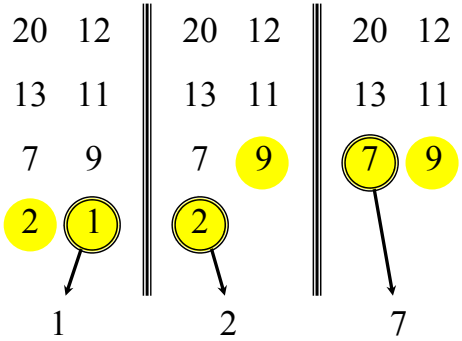


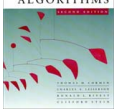
Merging two sorted arrays



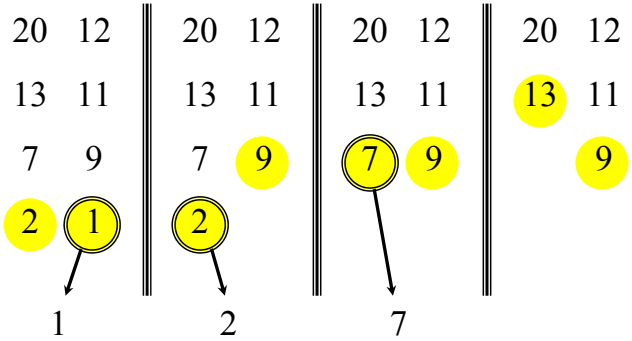


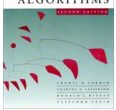
Merging two sorted arrays



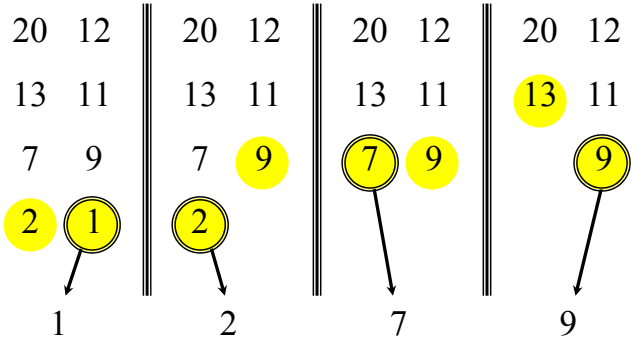


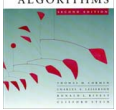
Merging two sorted arrays



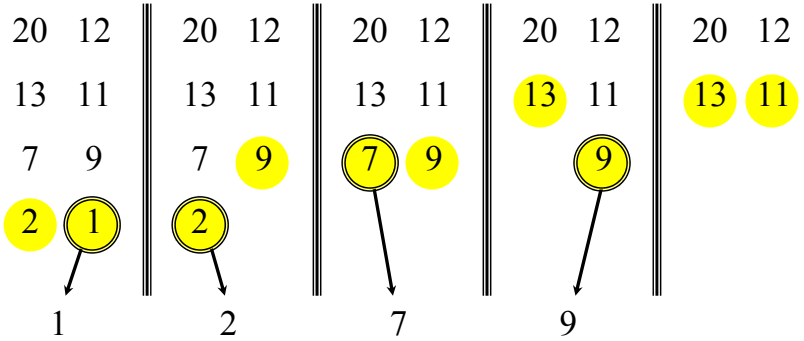


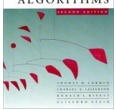
Merging two sorted arrays



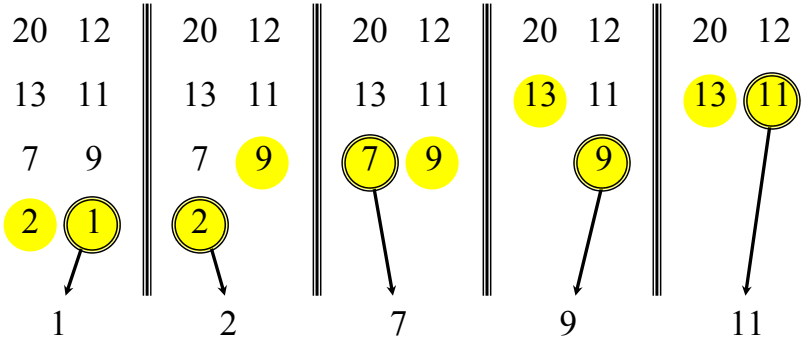


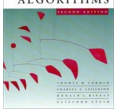
Merging two sorted arrays



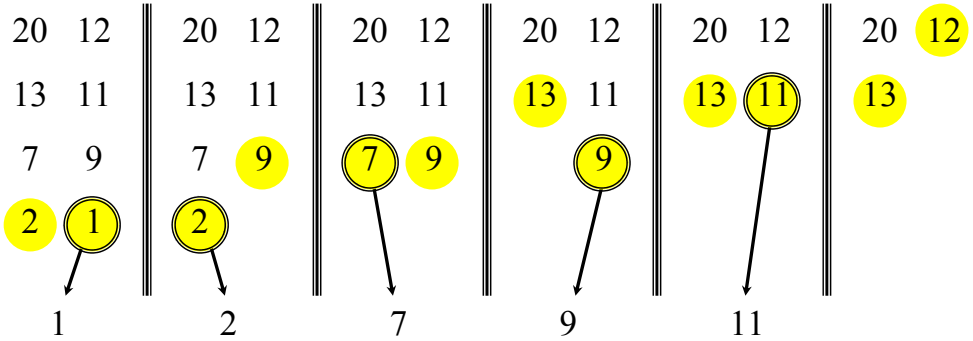


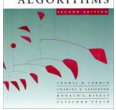
Merging two sorted arrays



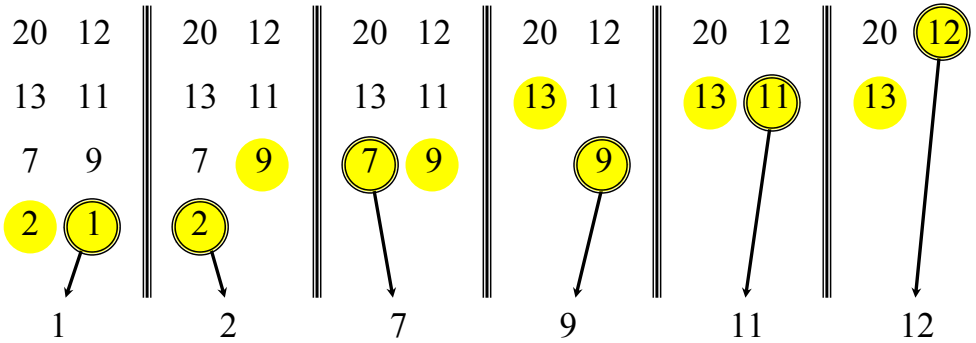


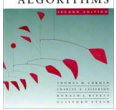
Merging two sorted arrays



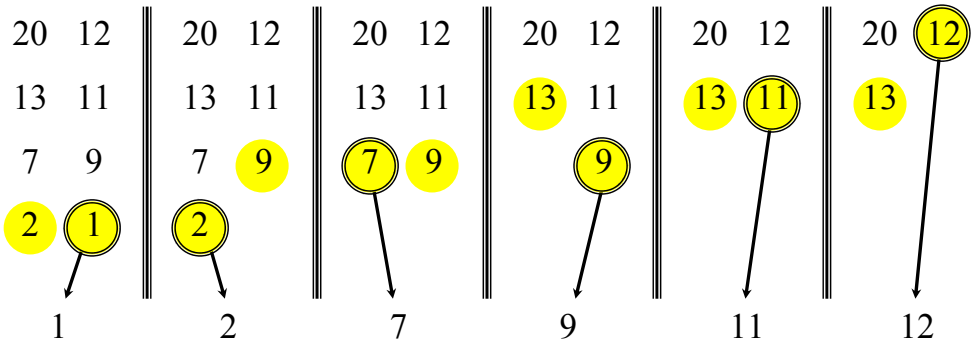


Merging two sorted arrays





Merging two sorted arrays



Time = $\Theta(n)$ to merge a total of n elements (linear time).

ArrayMerger Class

```
public class ArrayMerger<T extends Comparable<T>> {

    public void merge(T [] a, T[] b, T[] c) {

        int i=0, j=0;

        while (i + j < a.length + b.length) {
        ⋮
        }
    }

    ⋮
}
```

ArrayMerger

Using Array Merger

```
String[] names1 = {"Harry", "Tom", "Moe", "Curly"};
String[] names2 = {"Bob", "Alice", "Mallory"};

SelectionSort<String> sorter=new SelectionSort<String>();
sorter.sort(names1);
sorter.sort(names2);

String[] names3 = new String[names1.length+names2.length];
ArrayMerger<String> merger = new ArrayMerger<String>();

merger.merge(names1,names2, names3);

for (String name : names3) {
    System.out.println(name);
}
```

ArrayMerger