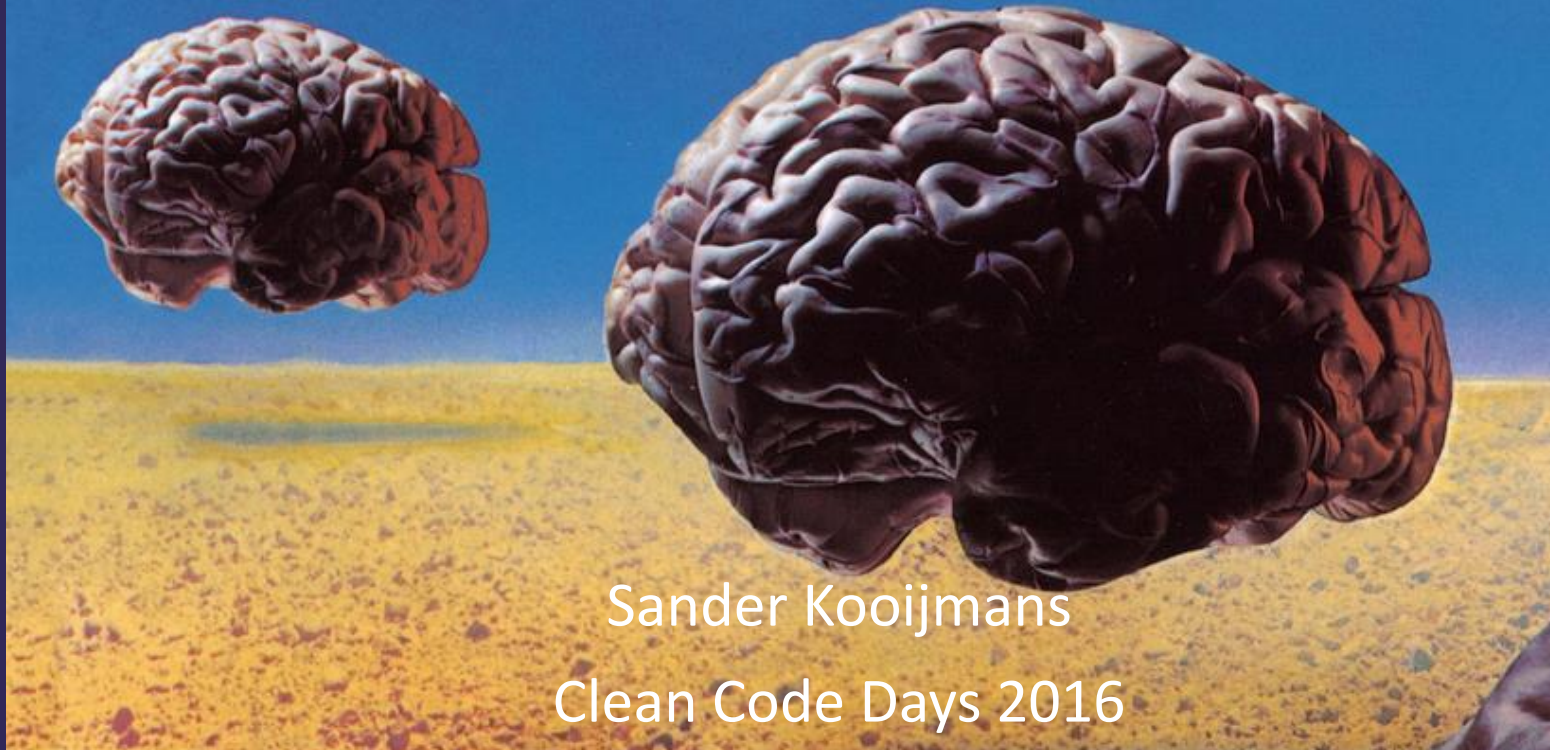
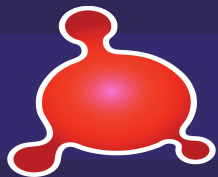


How invariants help to write loops

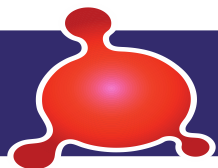


Sander Kooijmans
Clean Code Days 2016

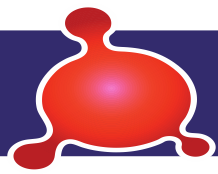


high tech ICT

A short story

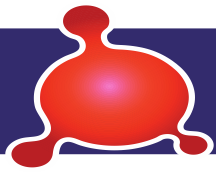


A short story



high tech ICT

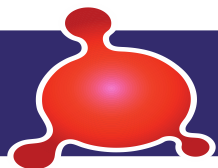
A short story



high tech ICT

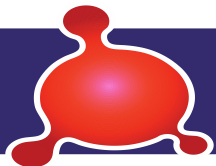
A short story

All the same we take our chances
Laughed at by time
Tricked by circumstances



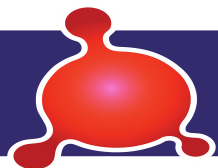
A short story

All the same we take our chances
Laughed at by time
Tricked by circumstances
Plus ça change
Plus c'est la même chose



Rush - Circumstances

All the same we take our chances
Laughed at by time
Tricked by circumstances
Plus ça change
Plus c'est la même chose
The more that things change
The more they stay the same

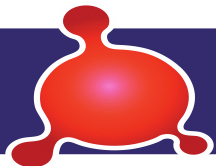


Who is Sander Kooijmans?



TU/e

That's another cook

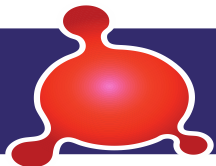


high tech ICT

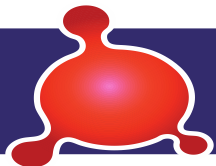
www.gogognome.nl

What can you expect?

- Introduction to invariants
- Proving a while-loop is correct using an invariant
- Starting with an invariant derive a while-loop
- A little bit of mathematics
- Try deriving while-loops yourself

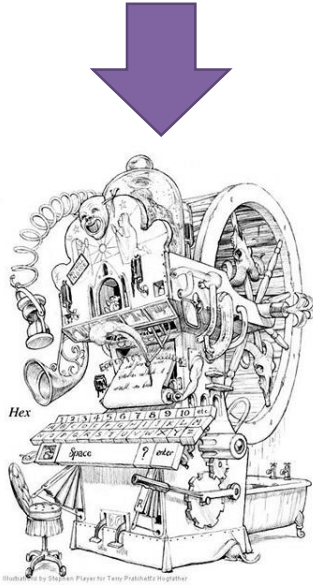


Why this presentation?



Predicates

All variables of your program



true or false

```
int a=4; int b=8;
```

After execution these predicates hold:

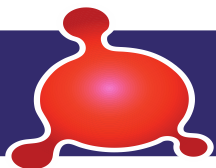
$$P(a) \equiv a == 4$$

$$Q(a, b) \equiv a < b$$

$$R(a, b) \equiv 2 * a == b$$

For brevity we write

$$R \equiv 2 * a == b$$



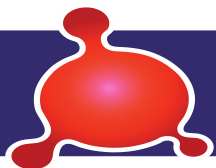
Hoare triple

The Hoare triple $\{Pre\} \ S \ \{Post\}$ means:

When Pre holds and S is executed
then if S terminates $Post$ holds

Hoare used these triples to specify how to prove

- Assignments
- If-statements
- While-loops

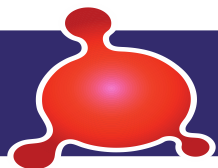


While-loop

$\{Inv\} \text{ while } (B) \text{ } S \text{ } \{Post\}$ follows from

- Inv holds before the while-loop is executed
- Inv holds after each iteration
- $Inv \ \&\& \ !B \Rightarrow Post$
- Provided that the loop terminates

Inv is called an **invariant**

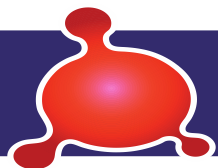


Termination of a while-loop



Termination is proved using a **bound function**

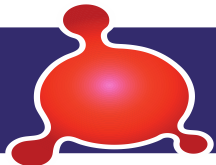
- A bound function decreases with at least one each iteration
- The bound function is bounded from below



Proving an algorithm to calculate 2^n

// pre: $n \geq 0$

// post: $p == 2^n$



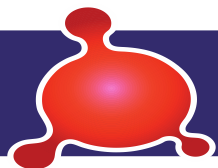
Proving an algorithm to calculate 2^n

```
// pre: n >= 0
int p=1; int i=0;

while (i != n) {
    p = 2*p; i = i+1;
}

// post: p == 2^n
```

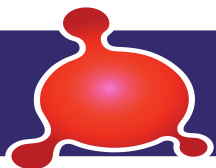
p	i	2 ⁱ
1	0	1
2	1	2
4	2	4
8	3	8



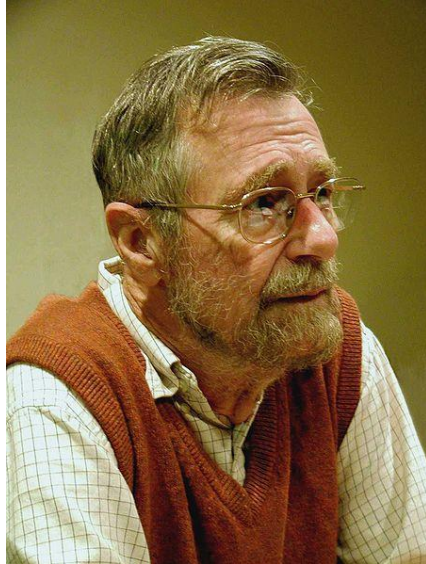
Proving an algorithm to calculate 2^n

```
// pre:  $n \geq 0$ 
int p=1; int i=0;
// invariant:  $p == 2^i$ 
// bound function:  $n-i \geq 0$ 
while (i != n) {
    p = 2*p; i = i+1;
}
// post:  $p == 2^n$ 
```

p	i	2^i
1	0	1
2	1	2
4	2	4
8	3	8



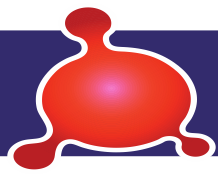
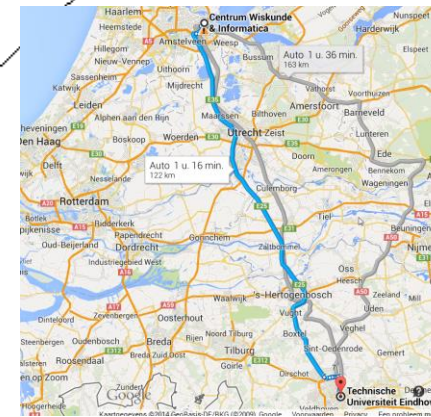
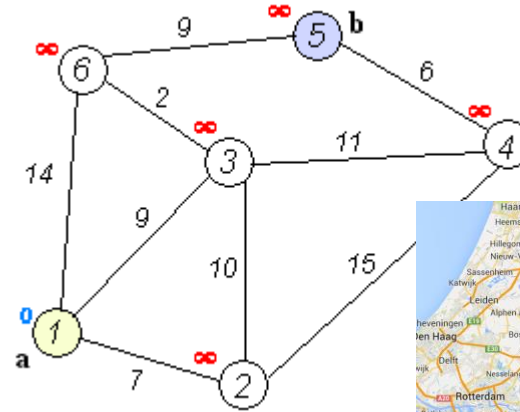
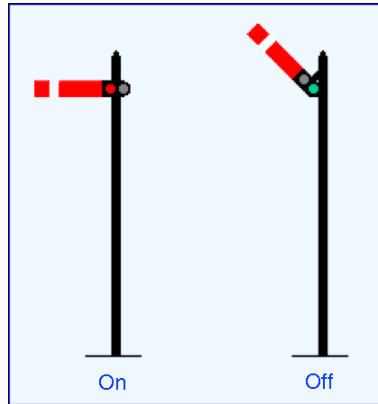
Edsger Wybe Dijkstra



TU/e

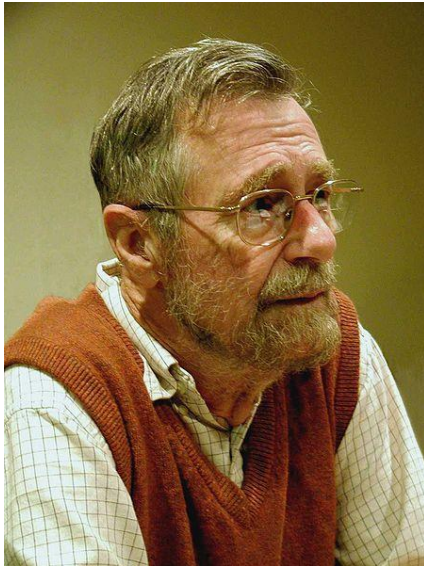



**GOTO
STATEMENT
CONSIDERED
HARMFUL**

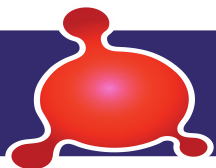


high tech ICT

Edsger Wybe Dijkstra



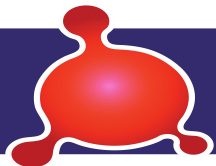
program derivation: to “develop proof and program hand in hand”



Deriving the algorithm to calculate 2^n

```
// pre:  $n \geq 0$ 
```

```
// post:  $p == 2^n$ 
```

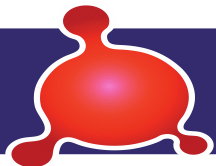


Deriving the algorithm to calculate 2^n

// pre: $n \geq 0$

// invariant: $p == 2^i$

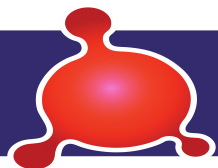
// post: $p == 2^n$



Deriving the algorithm to calculate 2^n

```
// pre:  $n \geq 0$   
int p=1; int i=0;  
// invariant:  $p = 2^i$ 
```

```
// post:  $p = 2^n$ 
```



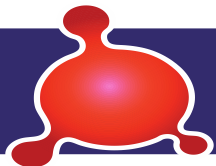
Deriving the algorithm to calculate 2^n

```
// pre:  $n \geq 0$   
int p=1; int i=0;  
// invariant:  $p == 2^i$ 
```

```
while (i != n) {
```

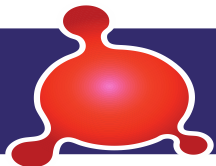
```
}
```

```
// post:  $p == 2^n$ 
```



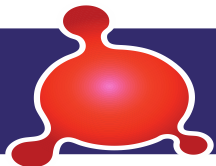
Deriving the algorithm to calculate 2^n

```
// pre:  $n \geq 0$   
int p=1; int i=0;  
// invariant:  $p == 2^i$   
  
while (i != n) {  
    p = 2*p; i = i+1;  
}  
// post:  $p == 2^n$ 
```



Deriving the algorithm to calculate 2^n

```
// pre:  $n \geq 0$ 
int p=1; int i=0;
// invariant:  $p == 2^i$ 
// bound function:  $n-i \geq 0$ 
while (i != n) {
    p = 2*p; i = i+1;
}
// post:  $p == 2^n$ 
```

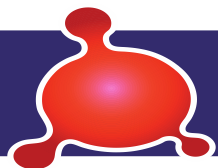


What if we started with another invariant?

// pre: $n \geq 0$

// **invariant:** $p \cdot 2^i == 2^n$

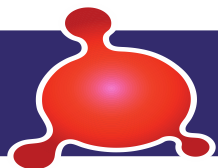
// post: $p == 2^n$



What if we started with another invariant?

```
// pre:  $n \geq 0$   
int p=1; int i=n;  
// invariant:  $p \cdot 2^i == 2^n$ 
```

```
// post:  $p == 2^n$ 
```



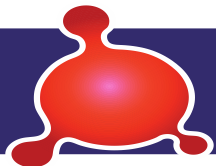
What if we started with another invariant?

```
// pre:  $n \geq 0$   
int p=1; int i=n;  
// invariant:  $p \cdot 2^i == 2^n$ 
```

```
while (i != 0) {
```

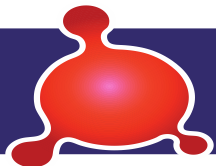
```
}
```

```
// post:  $p == 2^n$ 
```



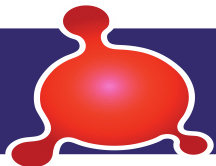
What if we started with another invariant?

```
// pre:  $n \geq 0$   
int p=1; int i=n;  
// invariant:  $p \cdot 2^i == 2^n$   
  
while (i != 0) {  
    p = 2*p; i = i-1;  
}  
// post:  $p == 2^n$ 
```



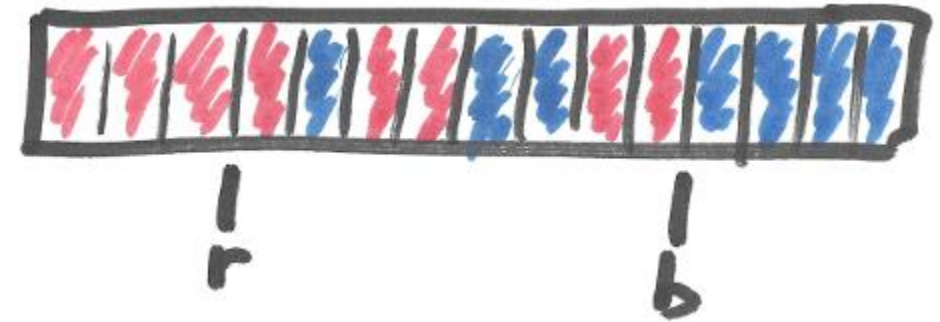
What if we started with another invariant?

```
// pre:  $n \geq 0$   
int p=1; int i=n;  
// invariant:  $p \cdot 2^i == 2^n$   
// bound function:  $i \geq 0$   
while (i != 0) {  
    p = 2*p; i = i-1;  
}  
// post:  $p == 2^n$ 
```

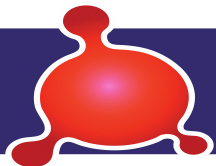


Sorting red and blue elements using swaps

// pre: colors contains red and blue elements in no particular order



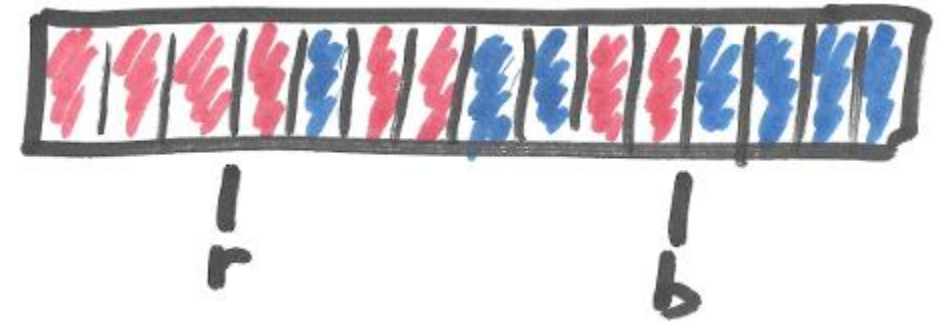
// post: colors is sorted in the order red, blue



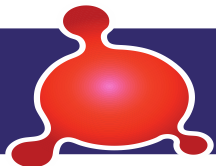
Sorting red and blue elements using swaps

// pre: colors contains red and blue elements in no particular order

// invariant: **colors[0..r)** are red and **colors[b..colors.length)** are blue

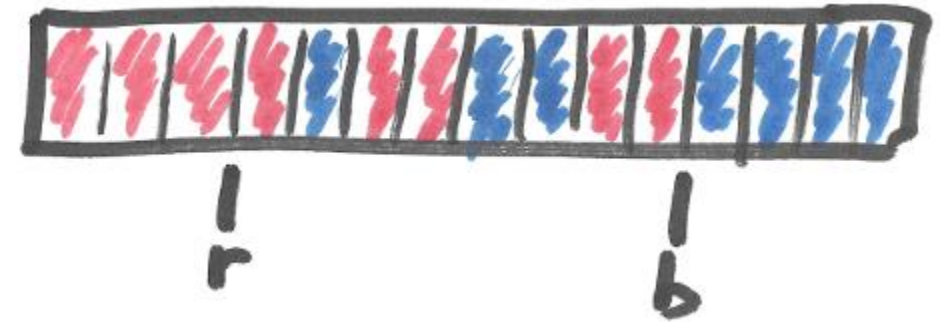


// post: colors is sorted in the order red, blue

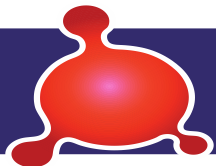


Sorting red and blue elements using swaps

```
// pre: colors contains red and blue elements in no particular order  
int r=0; int b=colors.length;  
// invariant: colors[0..r) are red and colors[b..colors.length) are blue
```



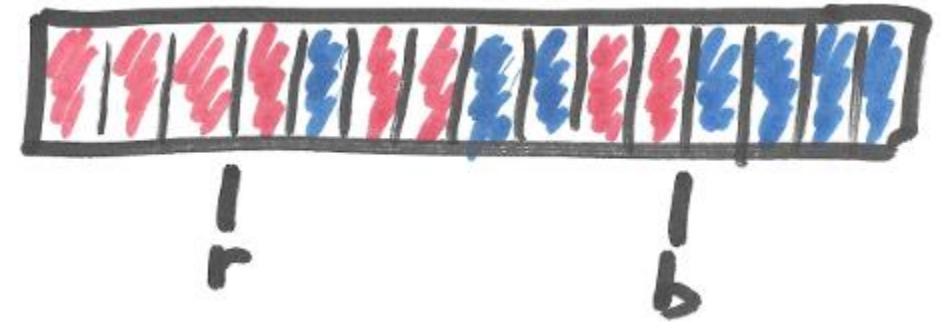
```
// post: colors is sorted in the order red, blue
```



Sorting red and blue elements using swaps

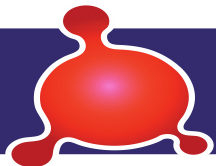
```
// pre: colors contains red and blue elements in no particular order  
int r=0; int b=colors.length;  
// invariant: colors[0..r) are red and colors[b..colors.length) are blue
```

```
while (r != b) {
```



```
}
```

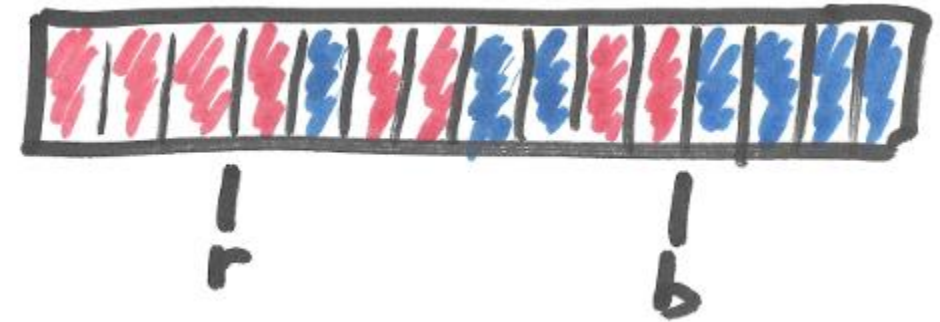
```
// post: colors is sorted in the order red, blue
```



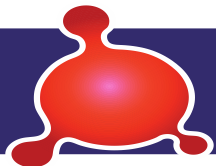
Sorting red and blue elements using swaps

```
// pre: colors contains red and blue elements in no particular order  
int r=0; int b=colors.length;  
// invariant: colors[0..r) are red and colors[b..colors.length) are blue
```

```
while (r != b) {  
    if (colors[r] == Color.RED) {  
        r++;  
    }
```



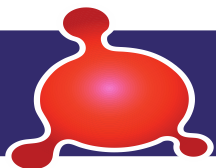
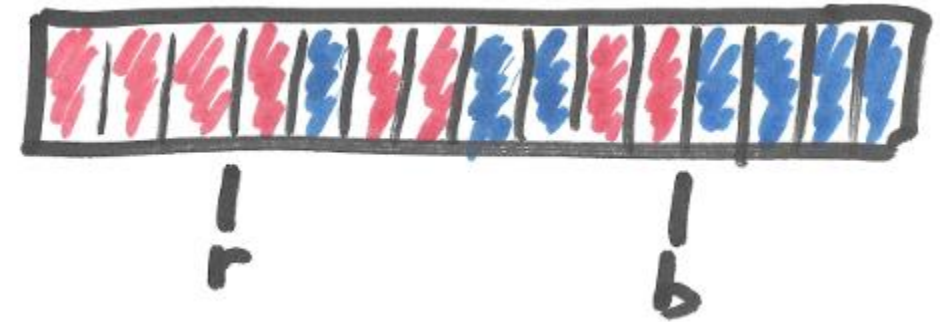
```
}  
// post: colors is sorted in the order red, blue
```



Sorting red and blue elements using swaps

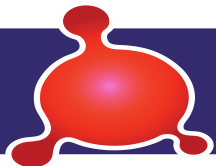
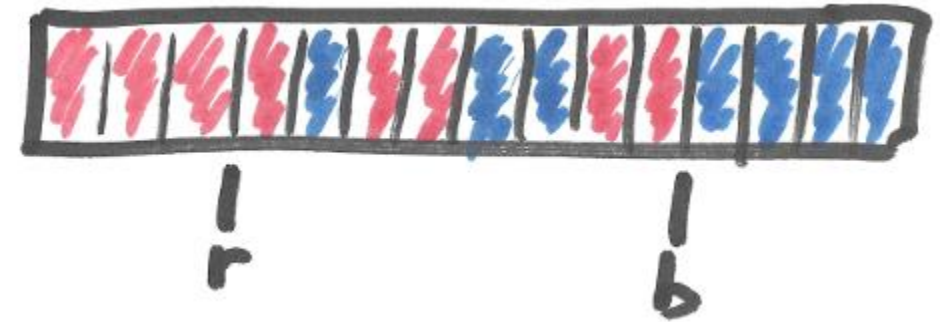
```
// pre: colors contains red and blue elements in no particular order
int r=0; int b=colors.length;
// invariant: colors[0..r) are red and colors[b..colors.length) are blue

while (r != b) {
    if (colors[r] == Color.RED) {
        r++;
    } else { // colors[r] == Color.BLUE
        b--;
        swap(colors, r, b);
    }
}
// post: colors is sorted in the order red, blue
```



Sorting red and blue elements using swaps

```
// pre: colors contains red and blue elements in no particular order
int r=0; int b=colors.length;
// invariant: colors[0..r) are red and colors[b..colors.length) are blue
// bound function: b-r >= 0
while (r != b) {
    if (colors[r] == Color.RED) {
        r++;
    } else { // colors[r] == Color.BLUE
        b--;
        swap(colors, r, b);
    }
}
// post: colors is sorted in the order red, blue
```



Try it yourself!

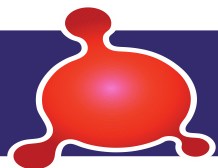
Exercises: tinyurl.com/jj9wkgx

www.hightechict.nl

sander.kooijmans@hightechict.nl



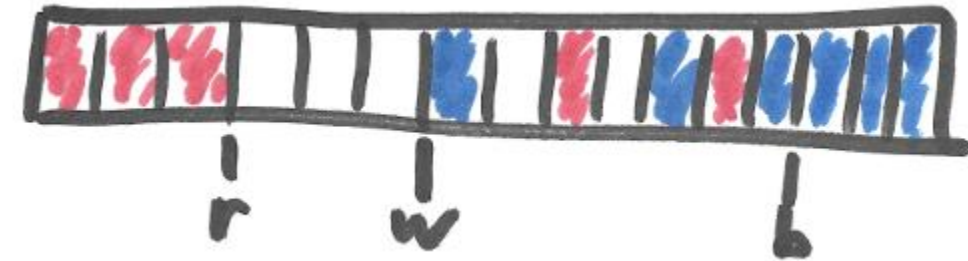
www.gogognome.nl



high tech ICT

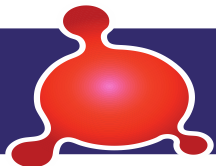
Sorting red, white and blue elements

// pre: colors contains red, white & blue elements in no particular order



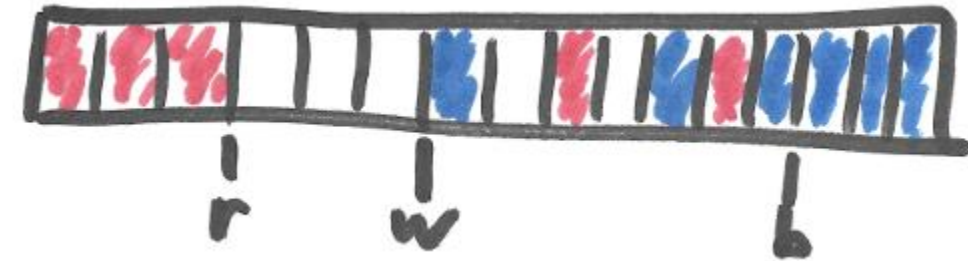
Invariant: colors[0..r) are red,
colors[r..w) are white,
colors[b..colors.length) are blue

// post: colors is sorted in the order red, white and blue



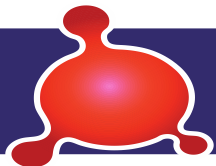
Sorting red, white and blue elements

```
// pre: colors contains red, white & blue elements in no particular order  
int r = 0; int w = 0; int b = colors.length;
```



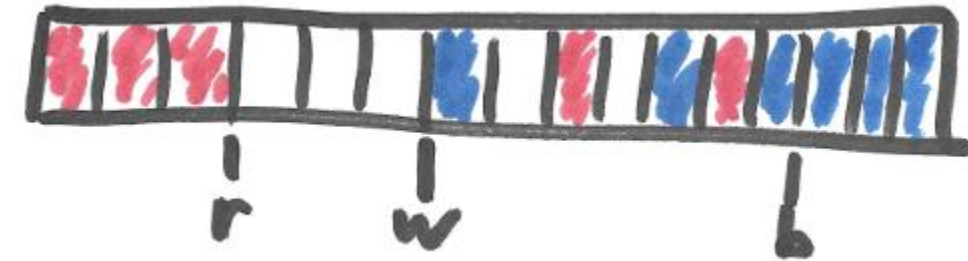
Invariant: colors[0..r) are red,
colors[r..w) are white,
colors[b..colors.length) are blue

```
// post: colors is sorted in the order red, white and blue
```



Sorting red, white and blue elements

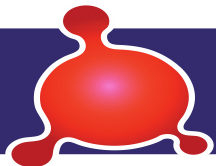
```
// pre: colors contains red, white & blue elements in no particular order  
int r = 0; int w = 0; int b = colors.length;  
while (w != b) {
```



Invariant: colors[0..r) are red,
colors[r..w) are white,
colors[b..colors.length) are blue

```
}
```

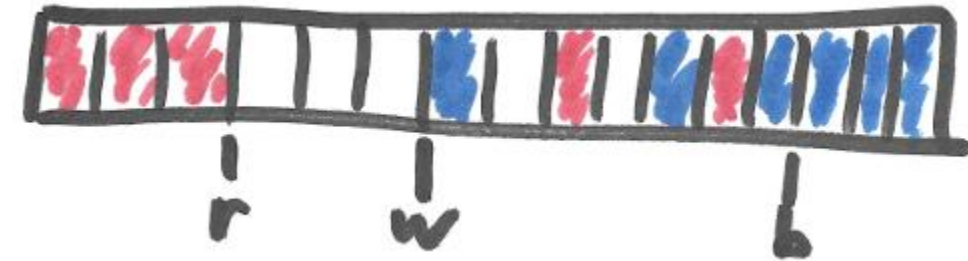
```
// post: colors is sorted in the order red, white and blue
```



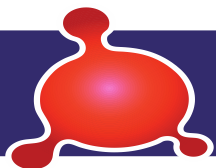
Sorting red, white and blue elements

```
// pre: colors contains red, white & blue elements in no particular order
int r = 0; int w = 0; int b = colors.length;
while (w != b) {
    if (colors[w] == Color.RED) {
        swap(colors, r, w);
        r++;
        w++;
    }
}

// post: colors is sorted in the order red, white and blue
```



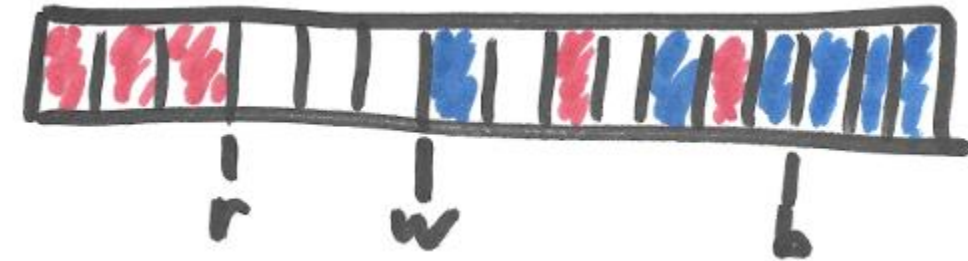
Invariant: colors[0..r) are red,
colors[r..w) are white,
colors[b..colors.length) are blue



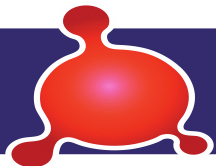
Sorting red, white and blue elements

```
// pre: colors contains red, white & blue elements in no particular order
int r = 0; int w = 0; int b = colors.length;
while (w != b) {
    if (colors[w] == Color.RED) {
        swap(colors, r, w);
        r++;
        w++;
    } else if (colors[w] == Color.WHITE) {
        w++;
    }
}

// post: colors is sorted in the order red, white and blue
```

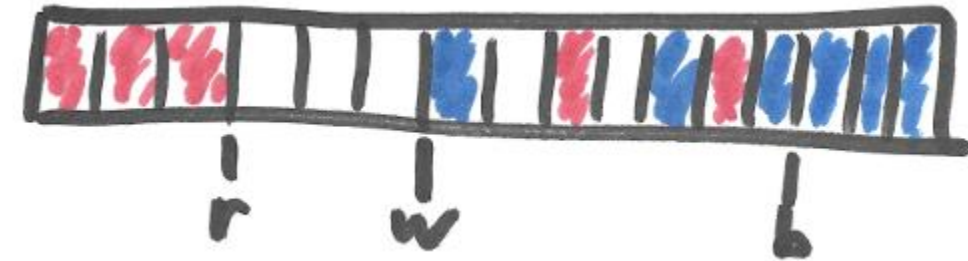


Invariant: colors[0..r) are red,
colors[r..w) are white,
colors[b..colors.length) are blue

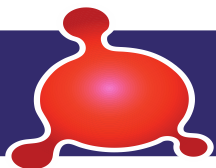


Sorting red, white and blue elements

```
// pre: colors contains red, white & blue elements in no particular order
int r = 0; int w = 0; int b = colors.length;
while (w != b) {
    if (colors[w] == Color.RED) {
        swap(colors, r, w);
        r++;
        w++;
    } else if (colors[w] == Color.WHITE) {
        w++;
    } else { // colors[w] == Color.BLUE;
        b--;
        swap(colors, w, b);
    }
}
// post: colors is sorted in the order red, white and blue
```

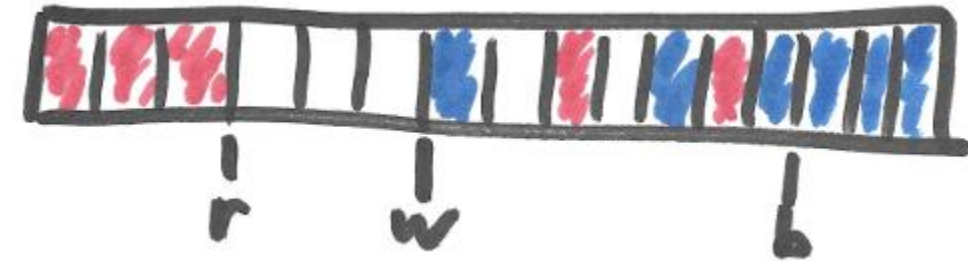


Invariant: colors[0..r) are red,
colors[r..w) are white,
colors[b..colors.length) are blue

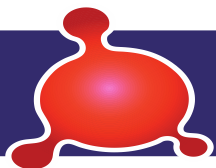


Sorting red, white and blue elements

```
// pre: colors contains red, white & blue elements in no particular order
int r = 0; int w = 0; int b = colors.length;
while (w != b) {
    if (colors[w] == Color.RED) {
        swap(colors, r, w);
        r++;
        w++;
    } else if (colors[w] == Color.WHITE) {
        w++;
    } else { // colors[w] == Color.BLUE;
        b--;
        swap(colors, w, b);
    }
}
// post: colors is sorted in the order red, white and blue
```



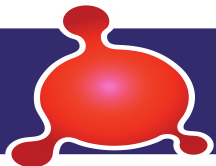
Invariant: colors[0..r) are red,
colors[r..w) are white,
colors[b..colors.length) are blue
Bound: $b - w \geq 0$



Binary search

```
// pre: array is sorted ascendingly
```

```
// post: b == "n is present in array"
```

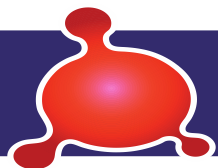


Binary search

// pre: array is sorted ascendingly

// invariant: if array contains n then it is in array[low..high)

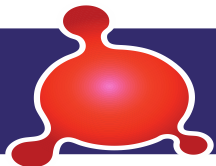
// post: b == “n is present in array”



Binary search

```
// pre: array is sorted ascendingly  
int low=0; int high = array.length;  
// invariant: if array contains n then it is in array[low..high)
```

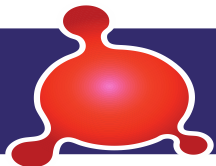
```
// post: b == “n is present in array”
```



Binary search

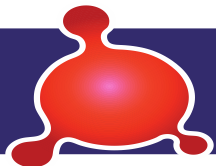
```
// pre: array is sorted ascendingly
int low=0; int high = array.length;
// invariant: if array contains n then it is in array[low..high)
while (low+1 < high) {

}
boolean b = array[low] == n
// post: b == “n is present in array”
```



Binary search

```
// pre: array is sorted ascendingly
int low=0; int high = array.length;
// invariant: if array contains n then it is in array[low..high)
while (low+1 < high) {
    int middle = (low+high)/2;
    if (array[middle] > n) {
        high = middle;
    } else {
        low = middle;
    }
}
boolean b = array[low] == n
// post: b == "n is present in array"
```



Try it yourself!

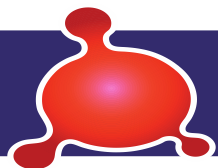
Exercises: tinyurl.com/jj9wkgx

www.hightechict.nl

sander.kooijmans@hightechict.nl



www.gogognome.nl



high tech ICT