Imperative Programming

(week 9)

Recap week 8

- Compare elements of type `El`:

```
bool operator< (const El& a, const El& b)
{     compare value a with b         }
```

- Example with `Date` struct:

```
bool operator< (const Date& a, const Date& b)
{ if (a.year == b.year)
    if (a.month == b.month)
         return a.day   < b.day ;
    else return a.month < b.month ;
  else   return a.year  < b.year ;
}
```

```
struct Date
{   int day ;
    int month ;
    int year ;
} ;
```

Recap week 8

- Output elements of type `El`:

```
ostream& operator<< (ostream& out, const El& el)
{   output el to out   }
```

- Vectors:
  - `vector<El> data ;`
  - `data.push_back (el)`
  - `data.pop_back ()`
  - `data.size ()`
  - `int size (vector<El>& data) { ... }`

we always add this function for convenience

Imperative Programming

(week 9)

Topics:

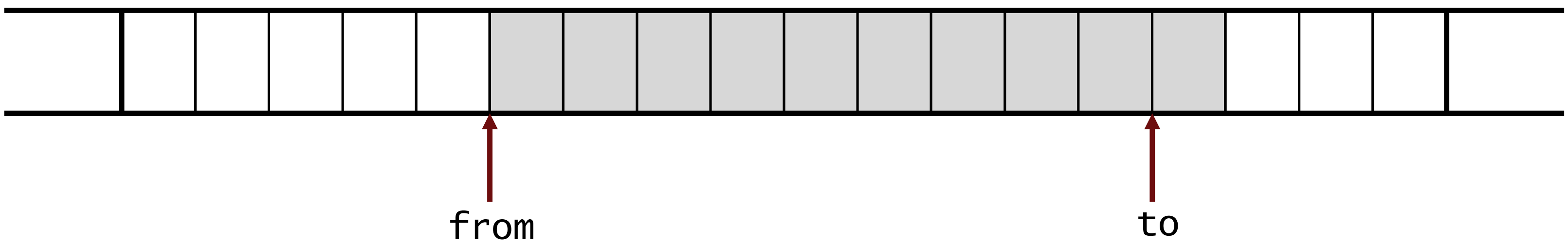
- Slices (more general array functions)
- In-situ sorting algorithms
- Note: we talk about arrays, but this is also valid for vectors (do it yourself)

More general array functions

- Abstract from manipulating entire array

Abstract from manipulating entire array

- Search and shift operations
- Not an *entire* array, but a *slice* of array

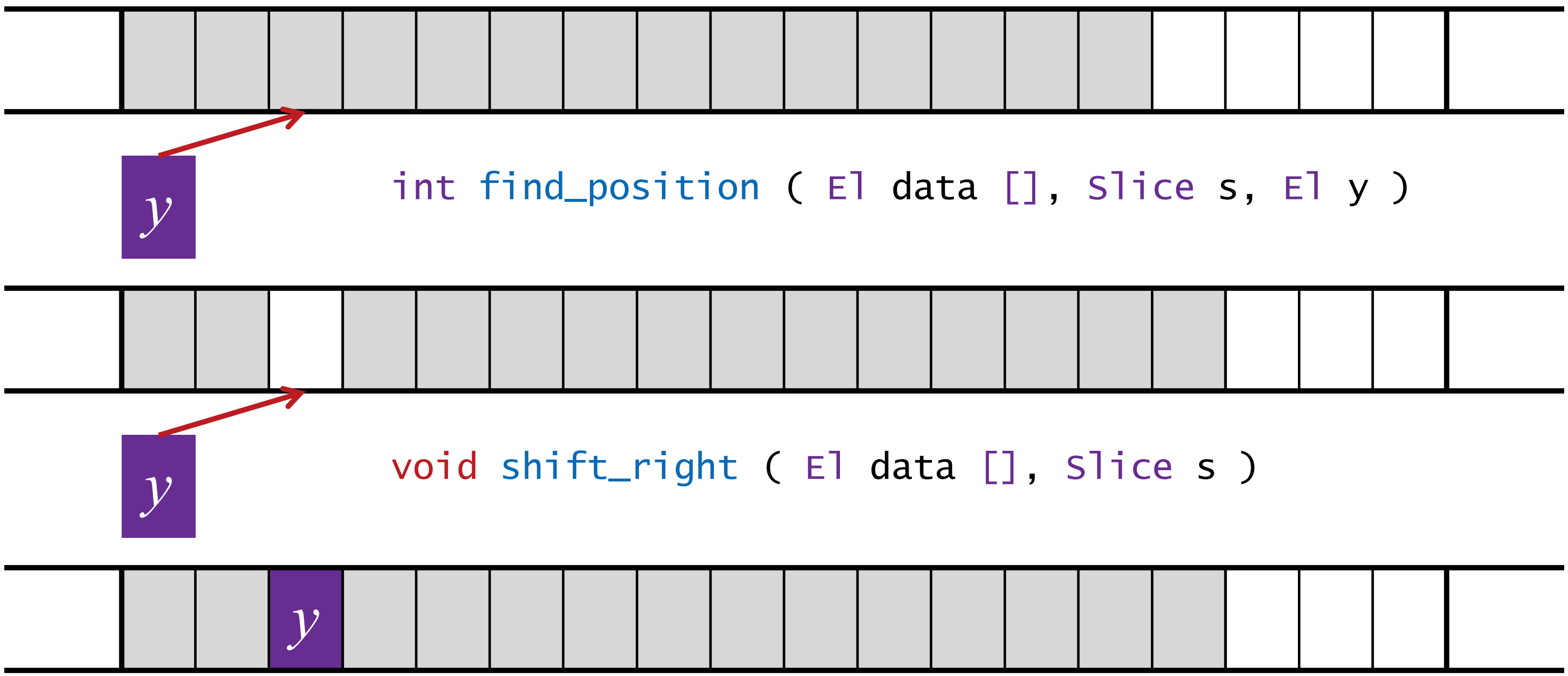



```
struct Slice
{   int from ;     // 0    ≤ from
    int to ;       // from ≤ to
} ;
Slice mkSlice (int from, int to)
{   assert (0 <= from && from <= to);
    Slice s = { from, to } ;
    return s ;
}
bool valid_slice (Slice s)
{   return 0 <= s.from && s.from <= s.to ; }
```

Example 1: test if array-slice is sorted

```
bool is_sorted (El data [], Slice s)
{// Pre-condition:
    assert (valid_slice (s)) ;
// Post-condition:
// result is true if data[s.from]   ≤ data[s.from+1]
//                   data[s.from+1] ≤ data[s.from+2]
//                              ...
//                   data[s.to-1]   ≤ data[s.to]
   bool sorted = true ;
   for (int i = s.from; i < s.to && sorted; i++)
       if (data[i] > data[i+1])
             sorted = false ;
   return sorted ;
}
```

Example 2: insert in sorted array

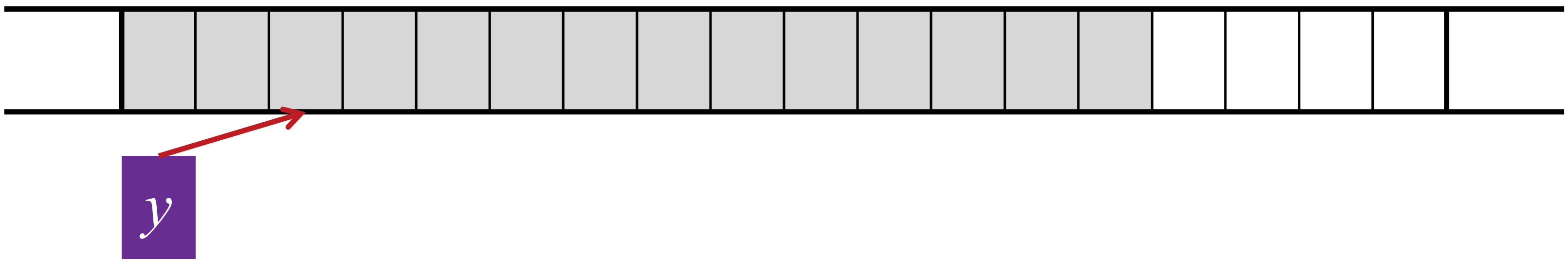


Problem solved?

```
void insert ( El data [], int& length, El y )
{ // pre-condition:
    assert (length > 0 && is_sorted (data, mkSlice (0, length-1))) ;
// post-condition: y is inserted in data, and data is still sorted

   const int POS = find_position (data, mkSlice (0, length-1), y) ;
   if (POS < length) shift_right (data, mkSlice (POS, length-1) ) ;
   data[POS] = y ;
   length++;
}
```

Example 2: insert in sorted array

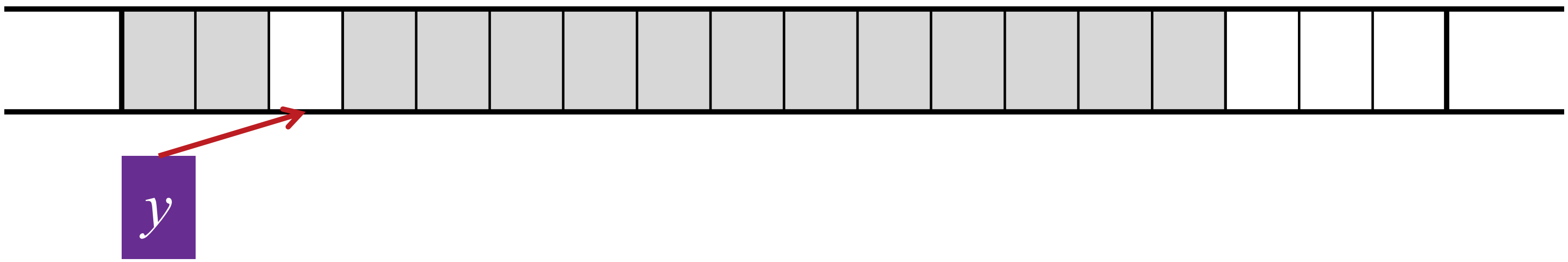



```
int find_position ( El data [], Slice s, El y )

{// Pre-condition:
    assert (valid_slice (s) && is_sorted (data, s)) ;
//  Post-condition: s.from ≤ result ≤ s.to+1
   for ( int i = s.from ; i <= s.to ; i++ )
      if ( y <= data[i] )
         return i ;
   return s.to+1 ;
}
```

elem is bigger than all elements in slice,
so must be added right after slice

Example 2: insert in sorted array

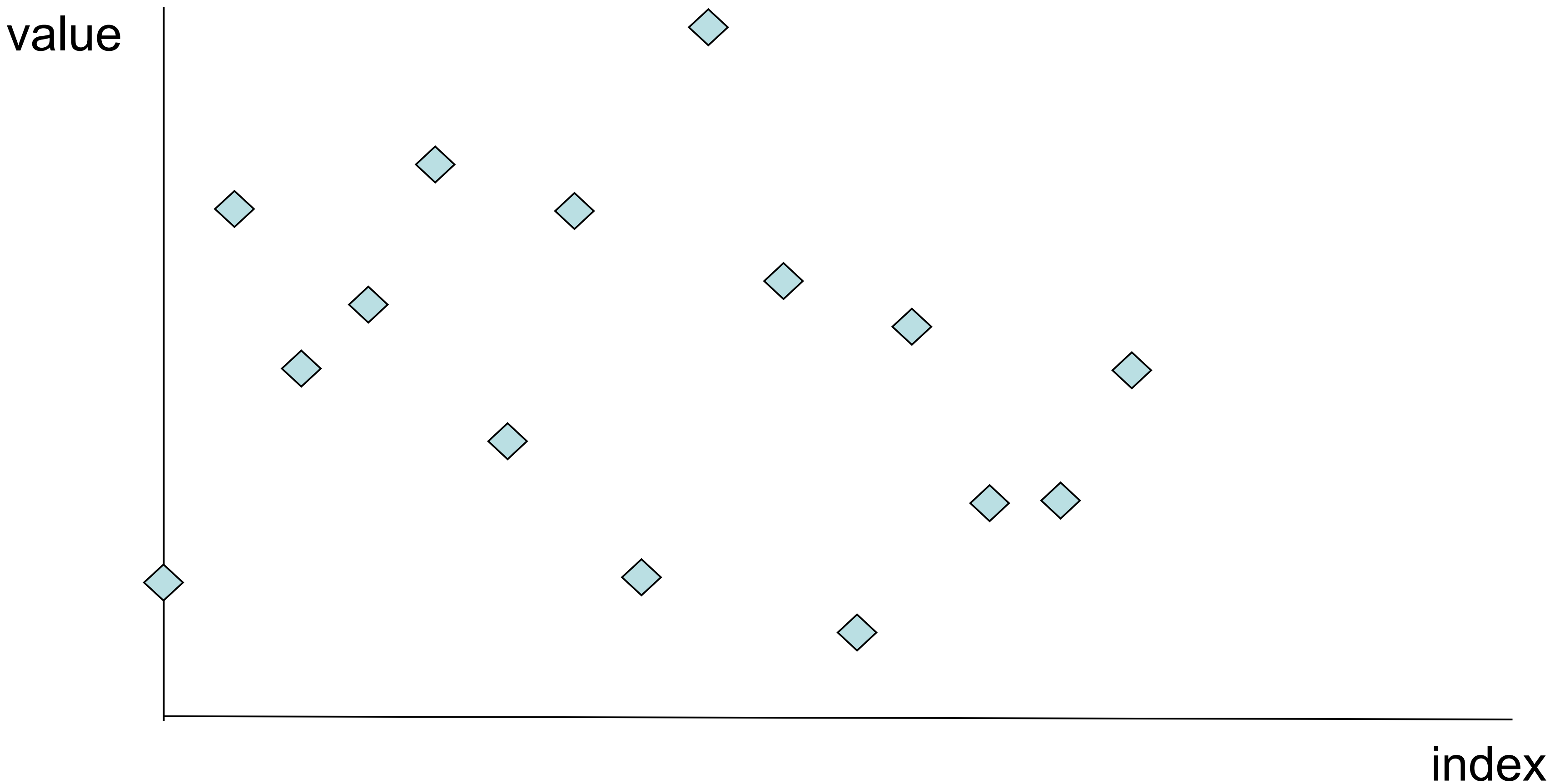



```
void shift_right ( El data [], Slice s )
{ // Pre-condition:
    assert (valid_slice (s)) ;
//  Post-condition: data[s.from+1] = old data[s.from]
//                  data[s.from+2] = old data[s.from+1]
//                       ...
//                  data[s.to+1]   = old data[s.to]
    for ( int i = s.to+1 ; i > s.from ; i-- )
        data[i] = data[i-1] ;
}
```

Problem solved.

Sorting

Unsorted array

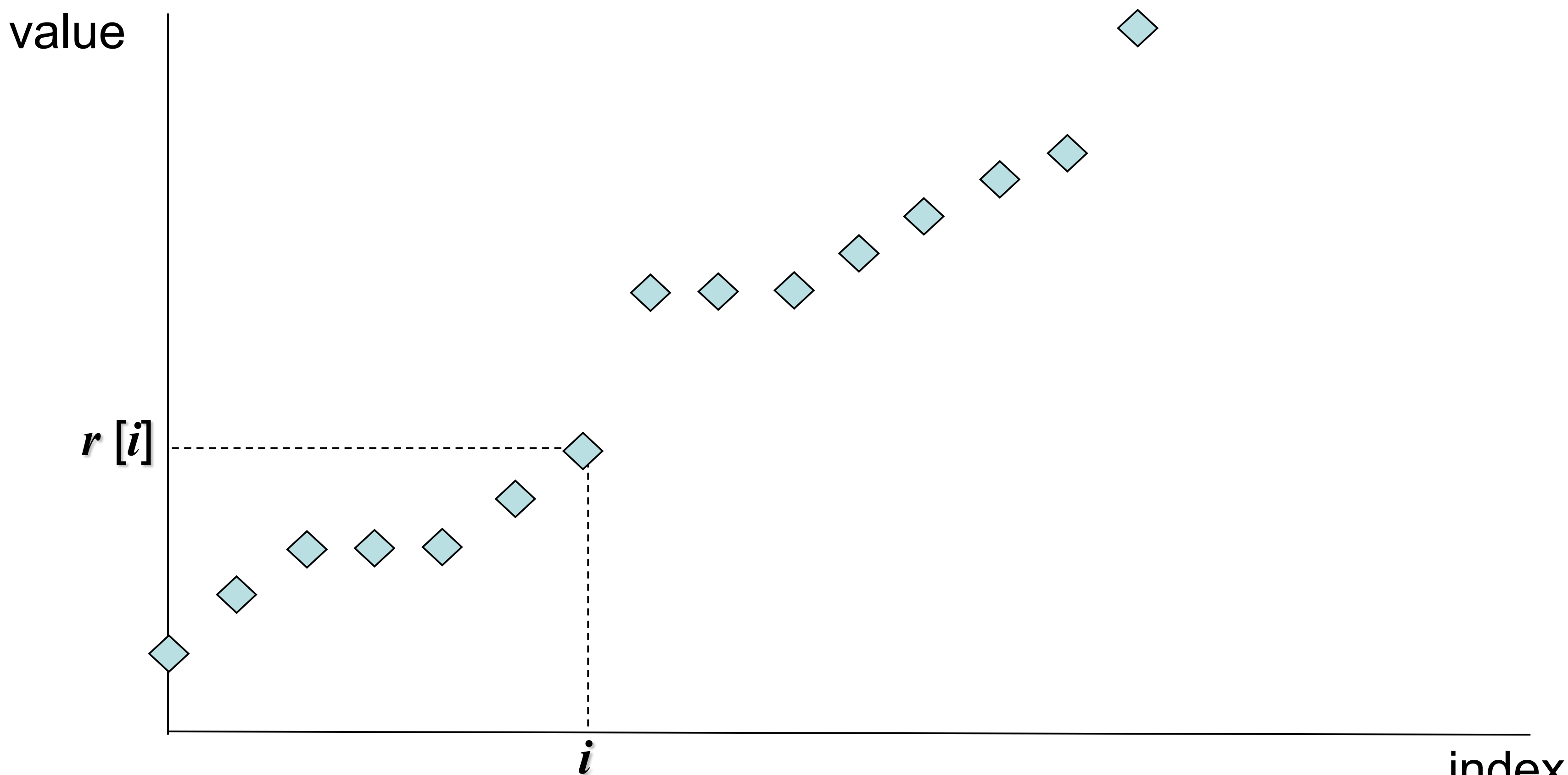


Sorted array

- An array r is sorted if

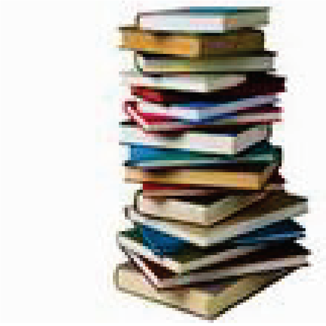
  for each $i < j$: $r[i] \le r[j]$

why not $r[i] < r[j]$?



Sorting

- Why do we study sorting?
  - sorting data occurs a lot
  - as entrance to discuss in more detail:
    - algorithms
    - complexity



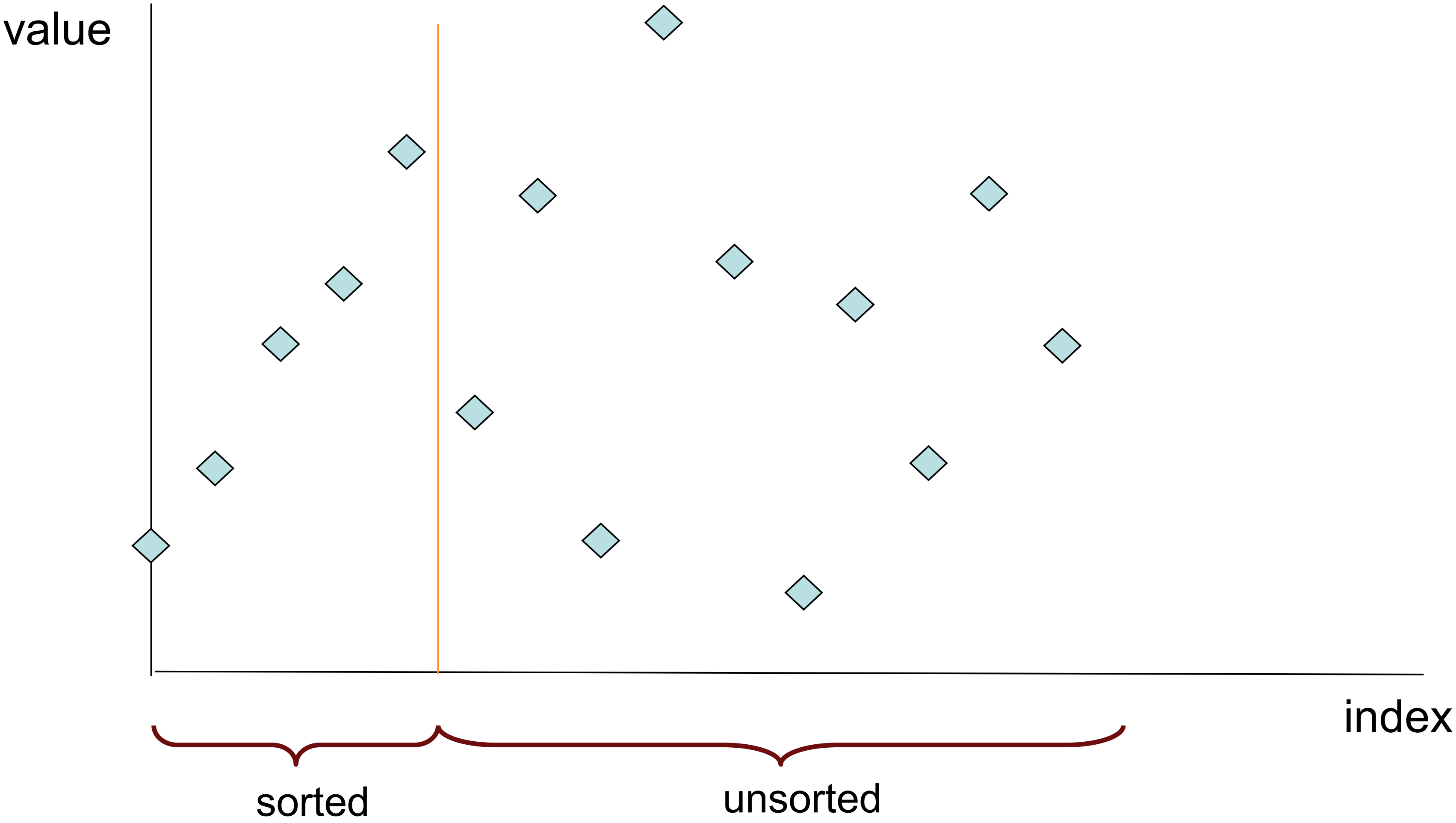
- Can be solved in many different ways
- **In situ**: "in place", that is, without extra array. A constant number of extra variables is OK.

```
void swap (El data [], int i, int j)
{// Pre-condition:
    assert ( i >= 0 && j >= 0 ) ;
//  Post-condition: data[i] = old data[j] and
//                  data[j] = old data[i]
     const El HELP = data [i] ;
     data [i] = data [j] ;
     data [j] = HELP ;
}
```

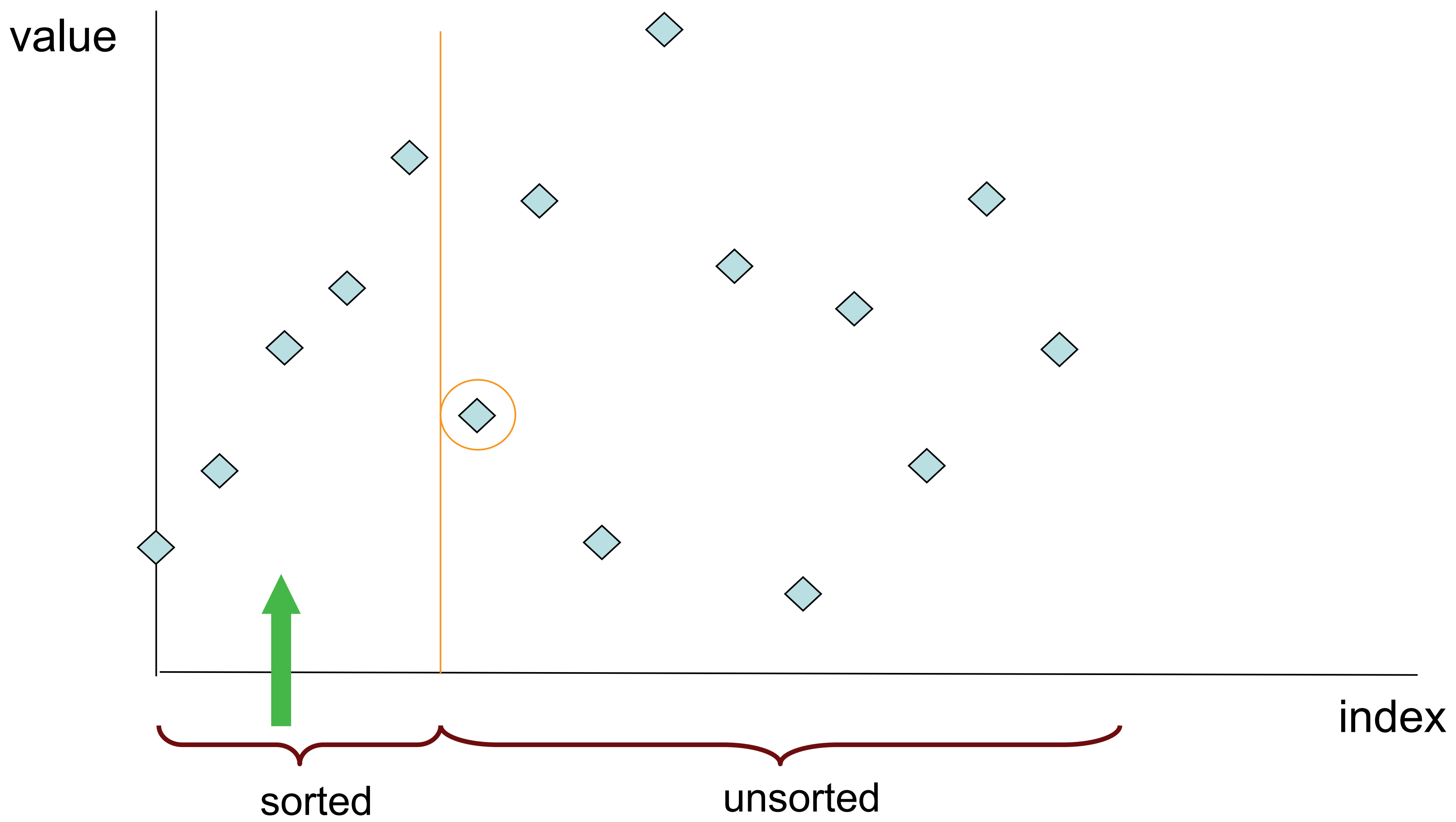
In situ sorting

- The key issue is: which elements to swap?
- Algorithms:
  1. insertion sort
  2. selection sort
  3. bubble sort
  4. …

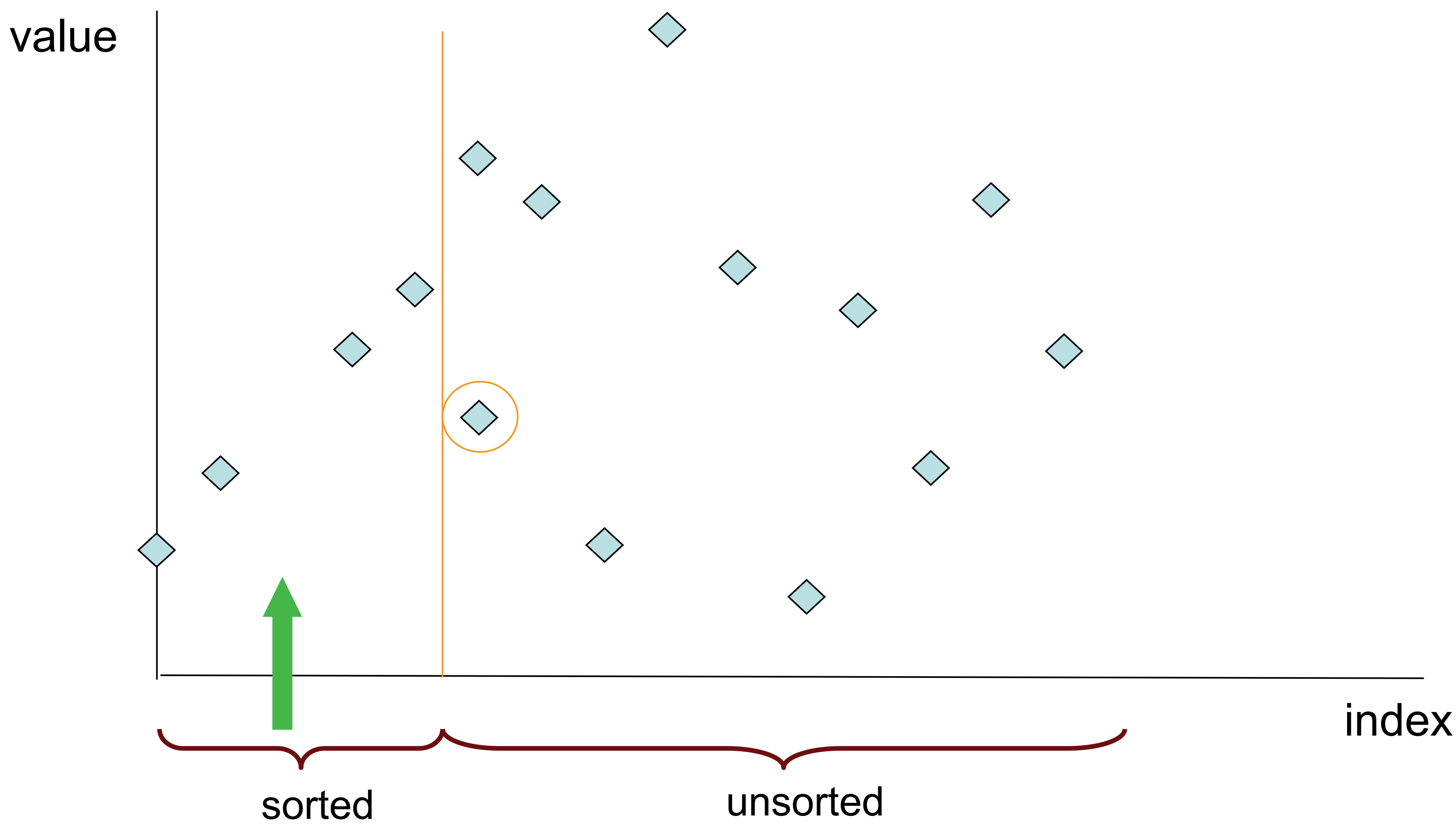
Insertion Sort



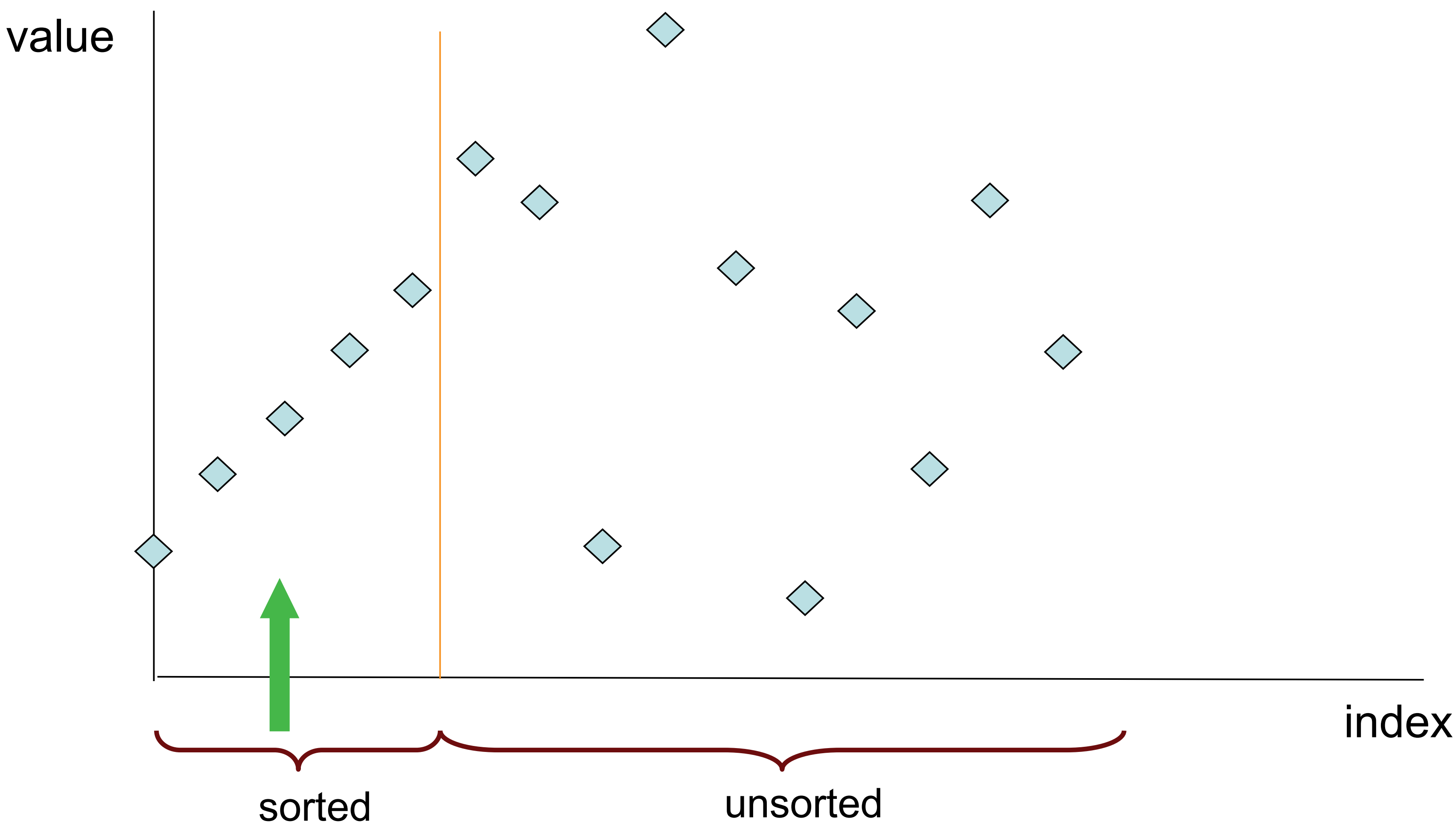
Insertion Sort



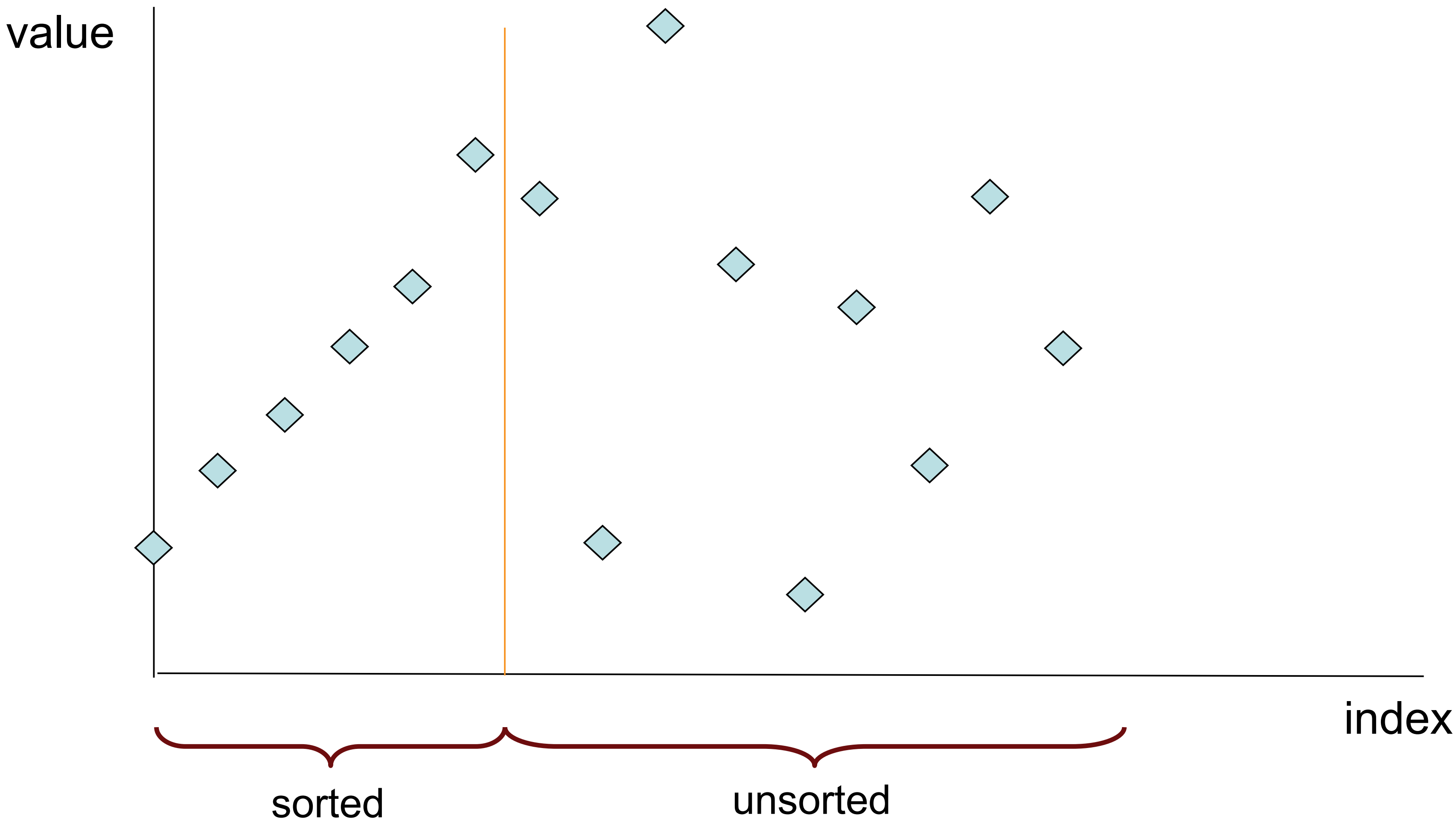
Insertion Sort



Insertion Sort



Insertion Sort



Insertion Sort

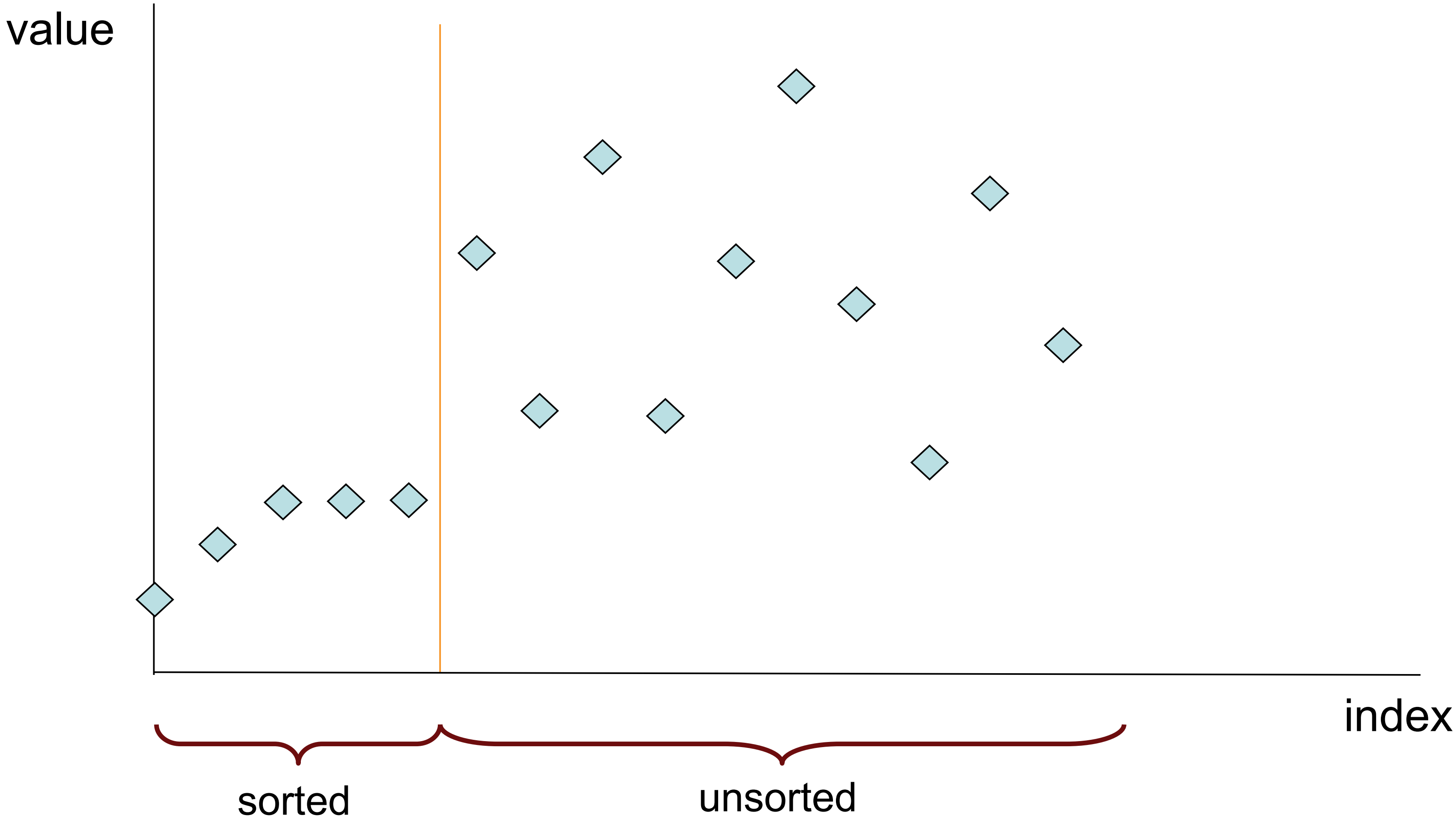
- Split array in two slices:
  - front slice is sorted (initially first element of array)
  - back slice is unsorted (initially rest of array)
- insert the first element from unsorted slice in sorted slice
  - already solved in slide #8 – slide #10

```
void insert ( El data [], int& length, El y )
{
  const int POS = find_position (data, mkSlice (0,length-1), y);
  if (POS < length)
     shift_right ( data, mkSlice (POS, length-1) ) ;
  data[POS] = y ;
  length++;
}
```

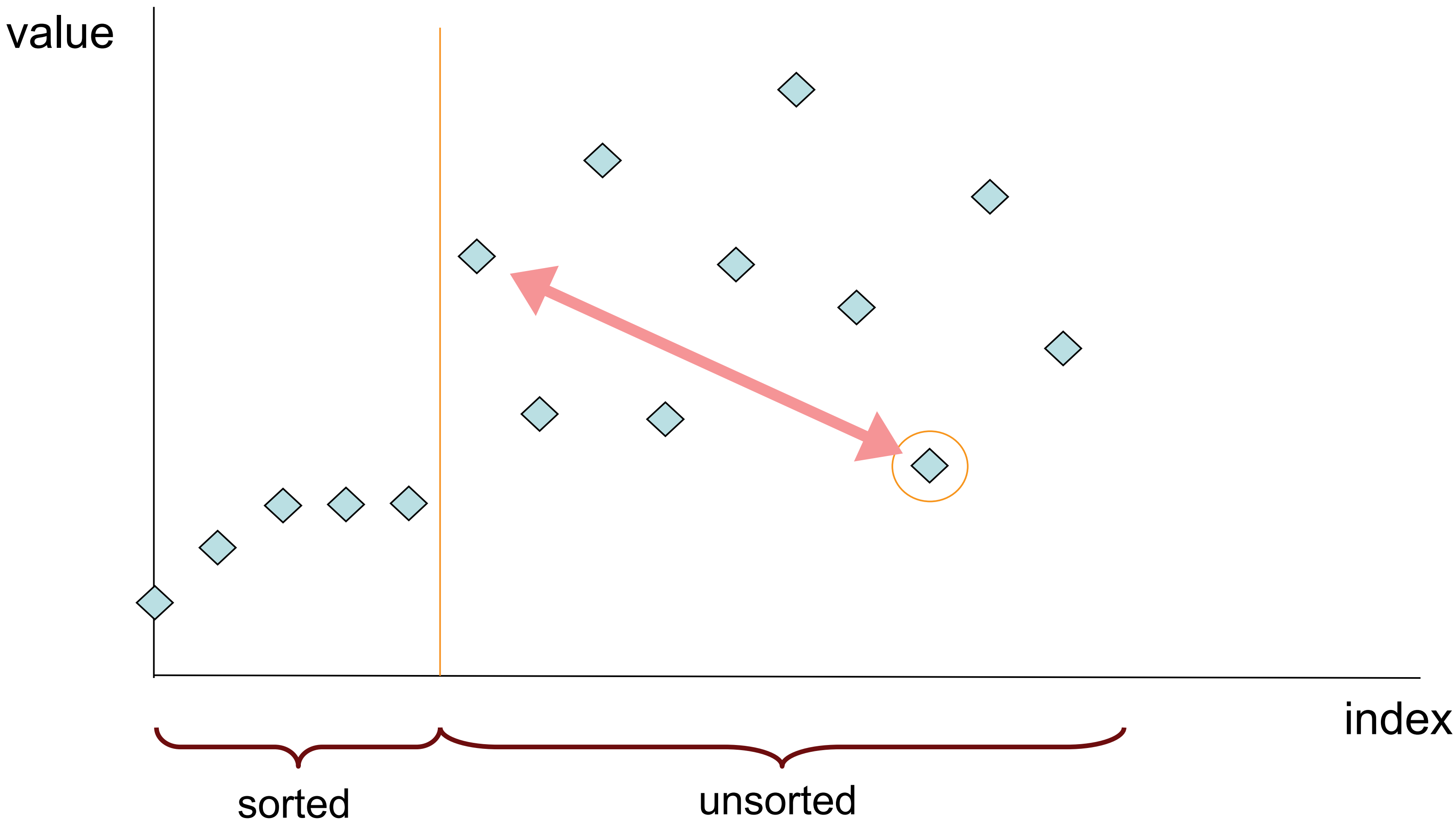
- Problem solved?

```
void insertion_sort ( El data [], int length )
{ int sorted = 1 ;
  while ( sorted < length )
    insert ( data, sorted, data[sorted] ) ;
}
```

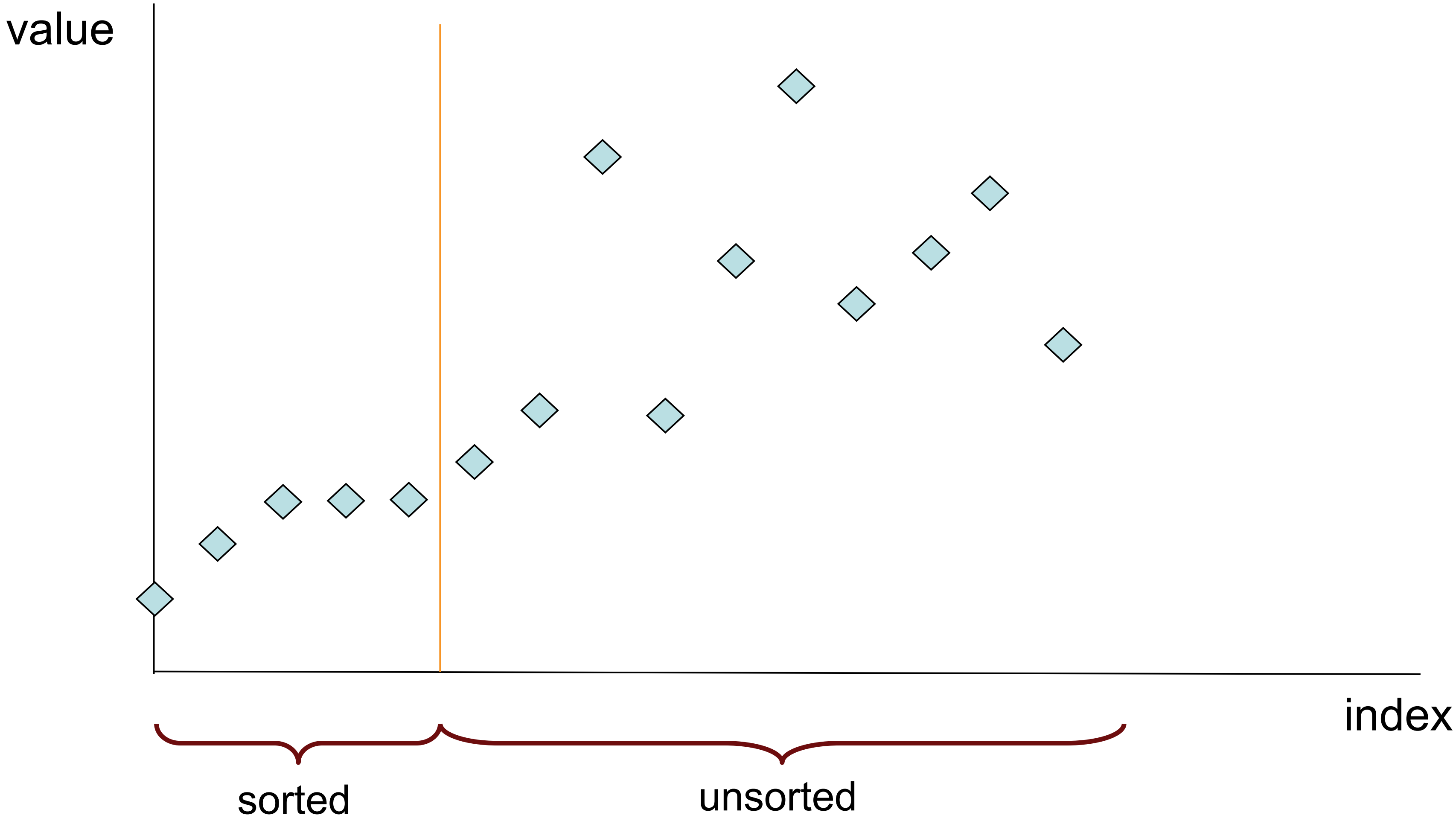
Selection Sort



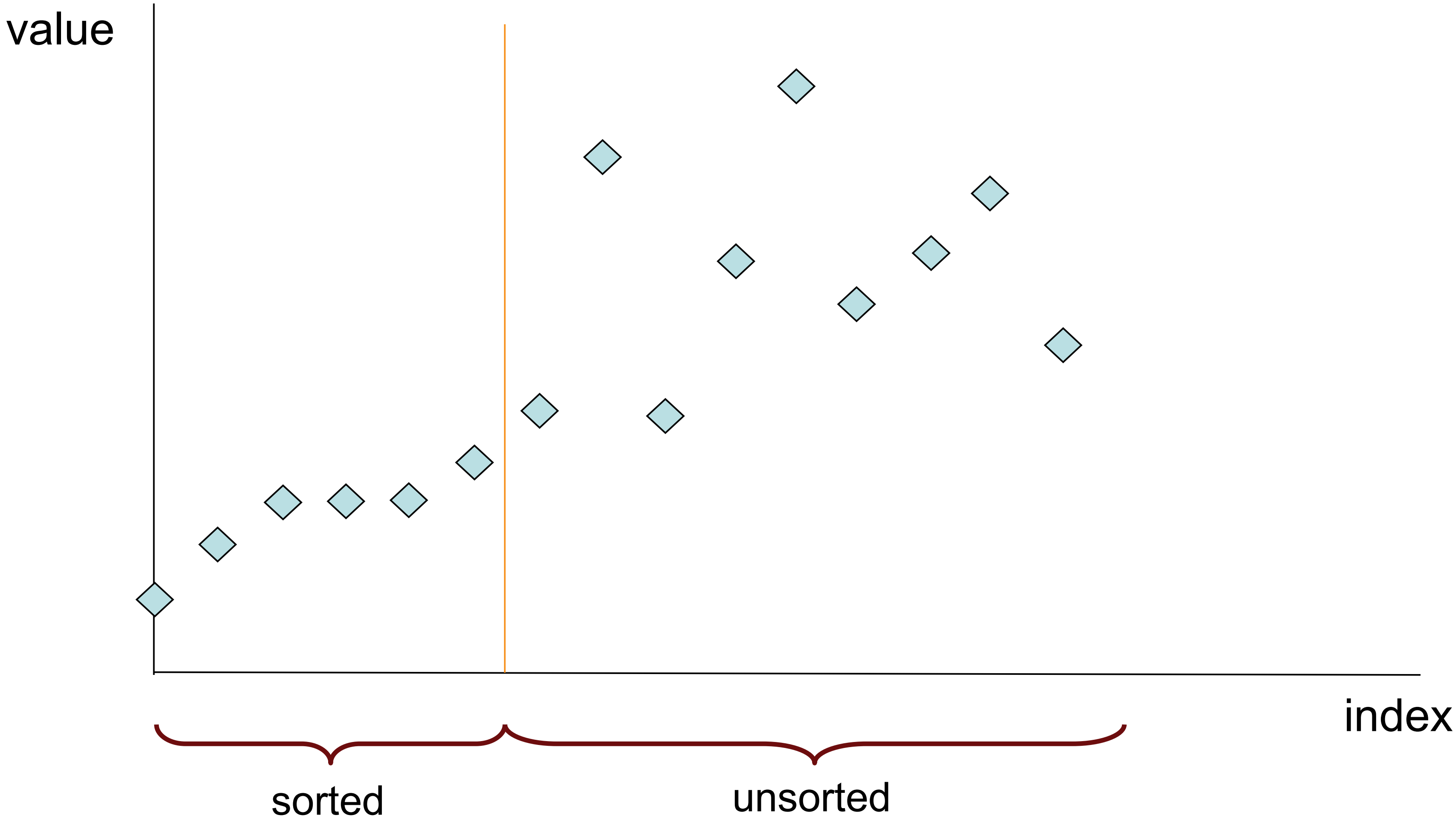
Selection Sort



Selection Sort



Selection Sort



Selection Sort

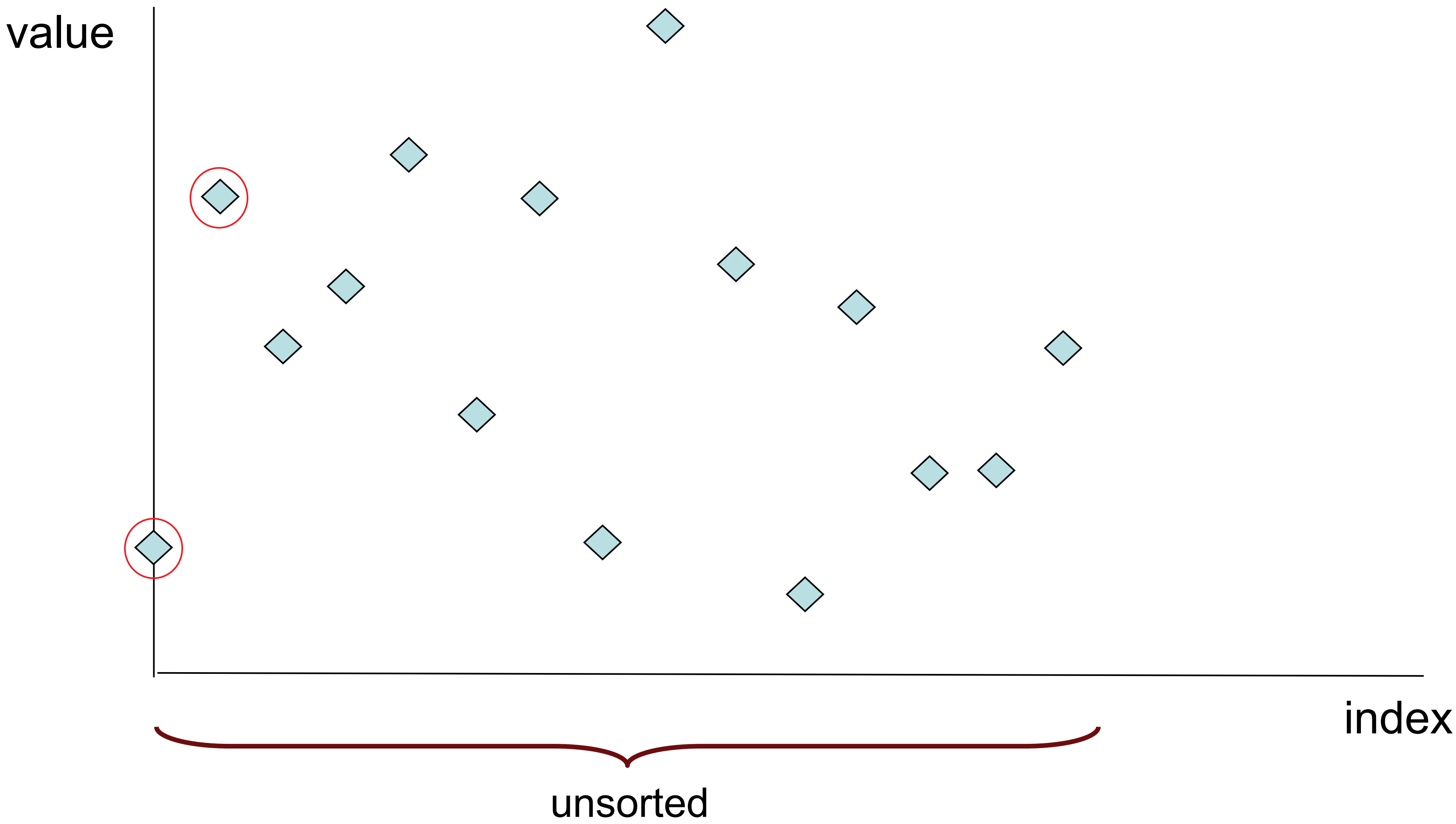
- Split array in two slices:
  – front slice is sorted (initially empty)
  – back slice is unsorted (initially whole array)
- select smallest element from unsorted slice:
  – `int smallest_value_at ( El data [], Slice s )` to do...
- swap it with first of unsorted slice
  – `void swap (El data [], int i, int j)` ✓ slide 7
- Problem solved?

```
void selection_sort ( El data [], int length )
{
  for ( int unsorted = 0 ; unsorted < length ; unsorted++ )
  {
     const int K = smallest_value_at
                          (data, mkSlice (unsorted, length-1));
     swap ( data, unsorted, K ) ;
  }
}
```

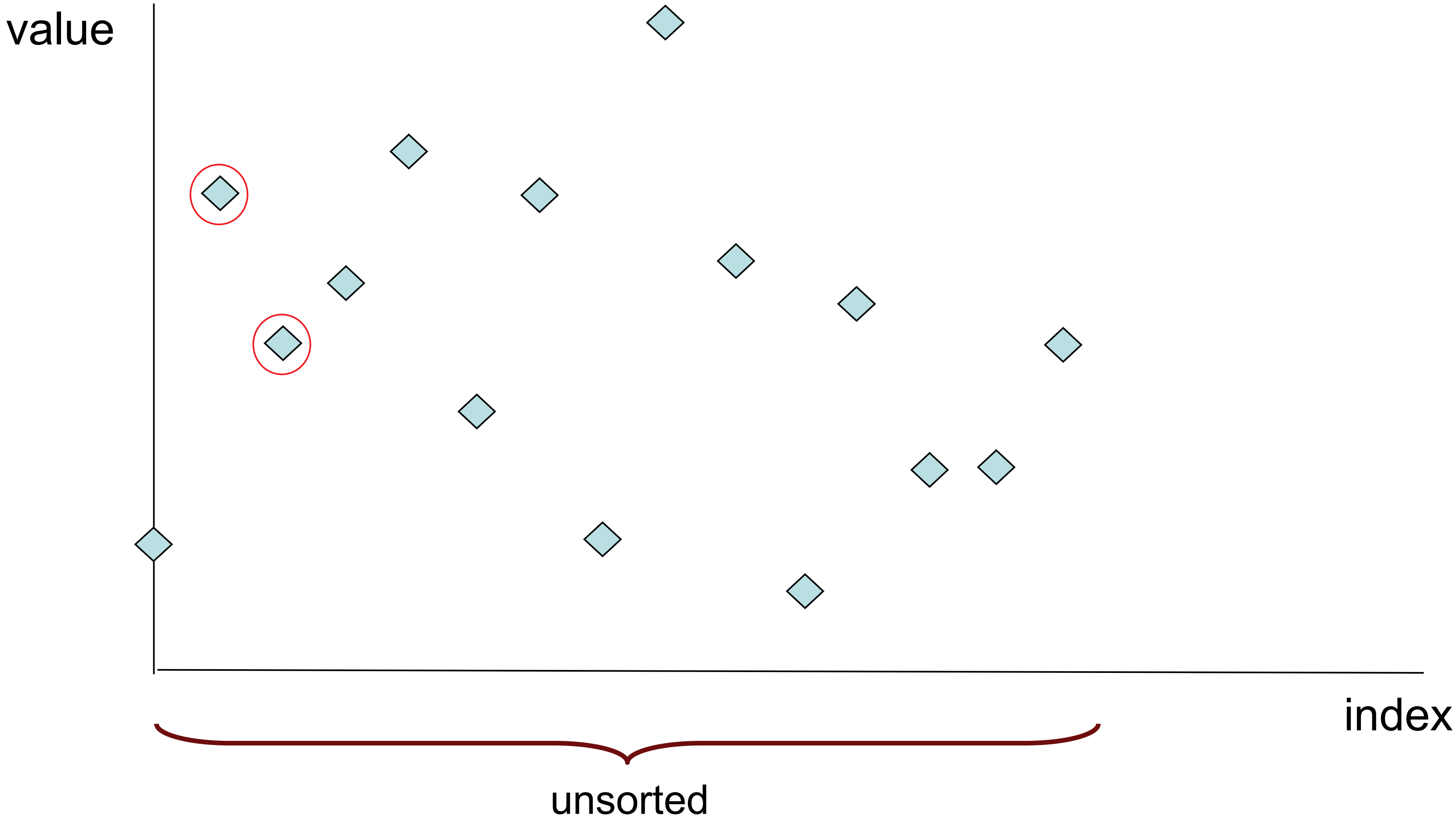
Searching position smallest value

```
int smallest_value_at ( El data [], Slice s )
{// Pre-condition:
    assert (valid_slice (s)) ;
// Post-condition: s.from ≤ result ≤ s.to and
// data[result] is the minimum value of
// data[s.from] .. data[s.to]
   int smallest_at = s.from ;
   for ( int index = s.from+1 ; index <= s.to ; index++ )
       if ( data[index] < data[smallest_at] )
             smallest_at = index ;
   return smallest_at ;
}
```

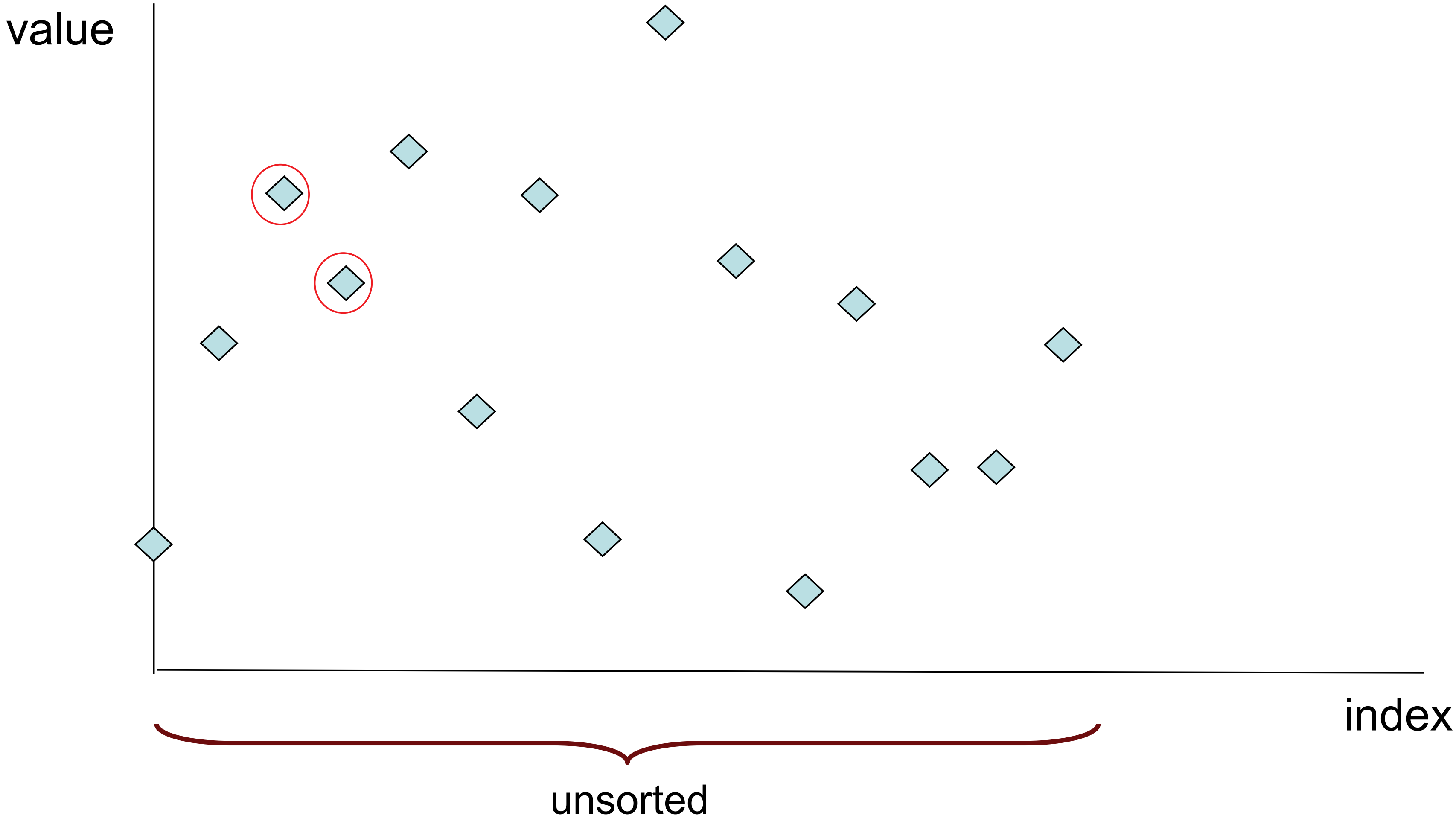
Bubble Sort



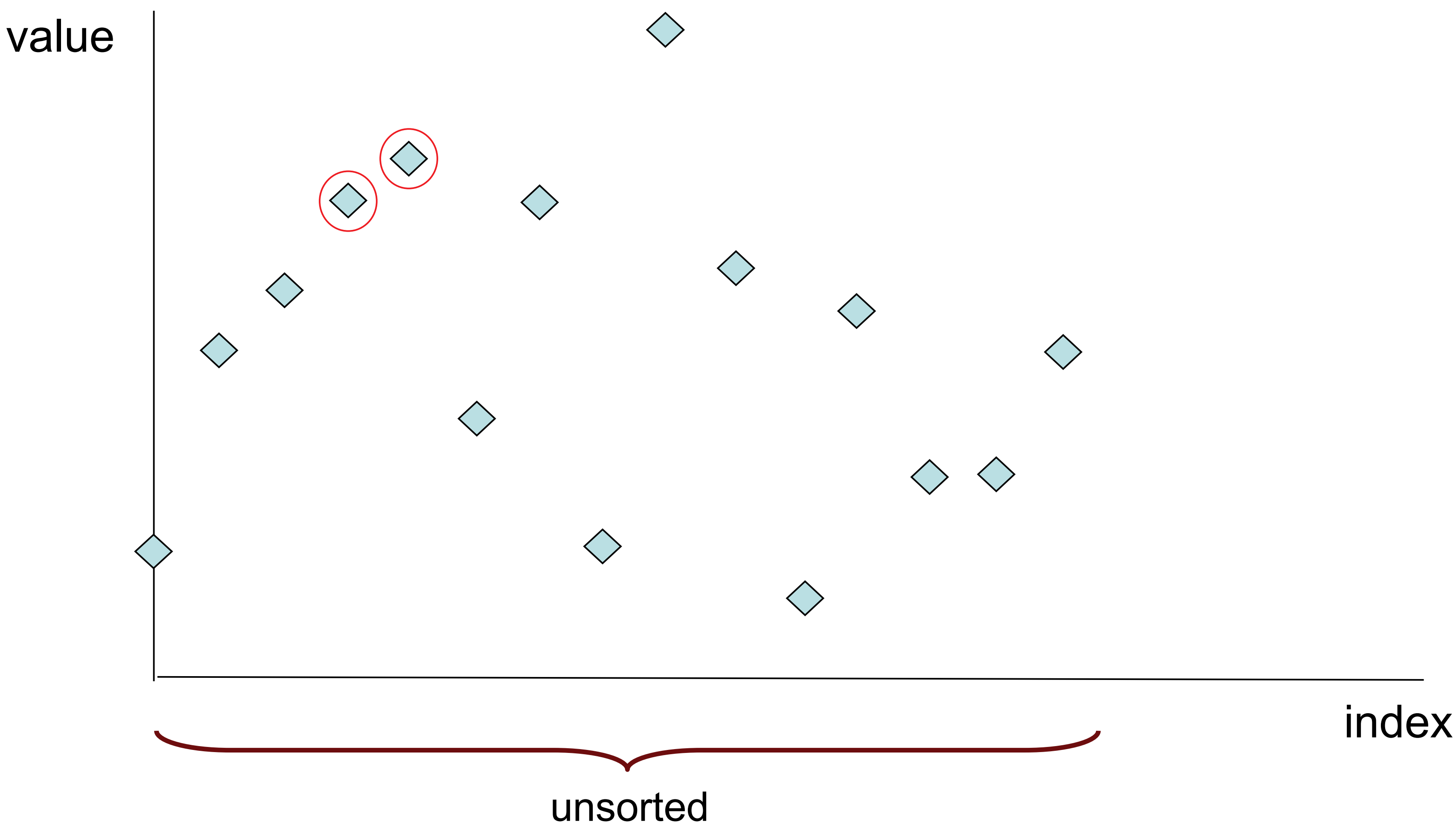
Bubble Sort



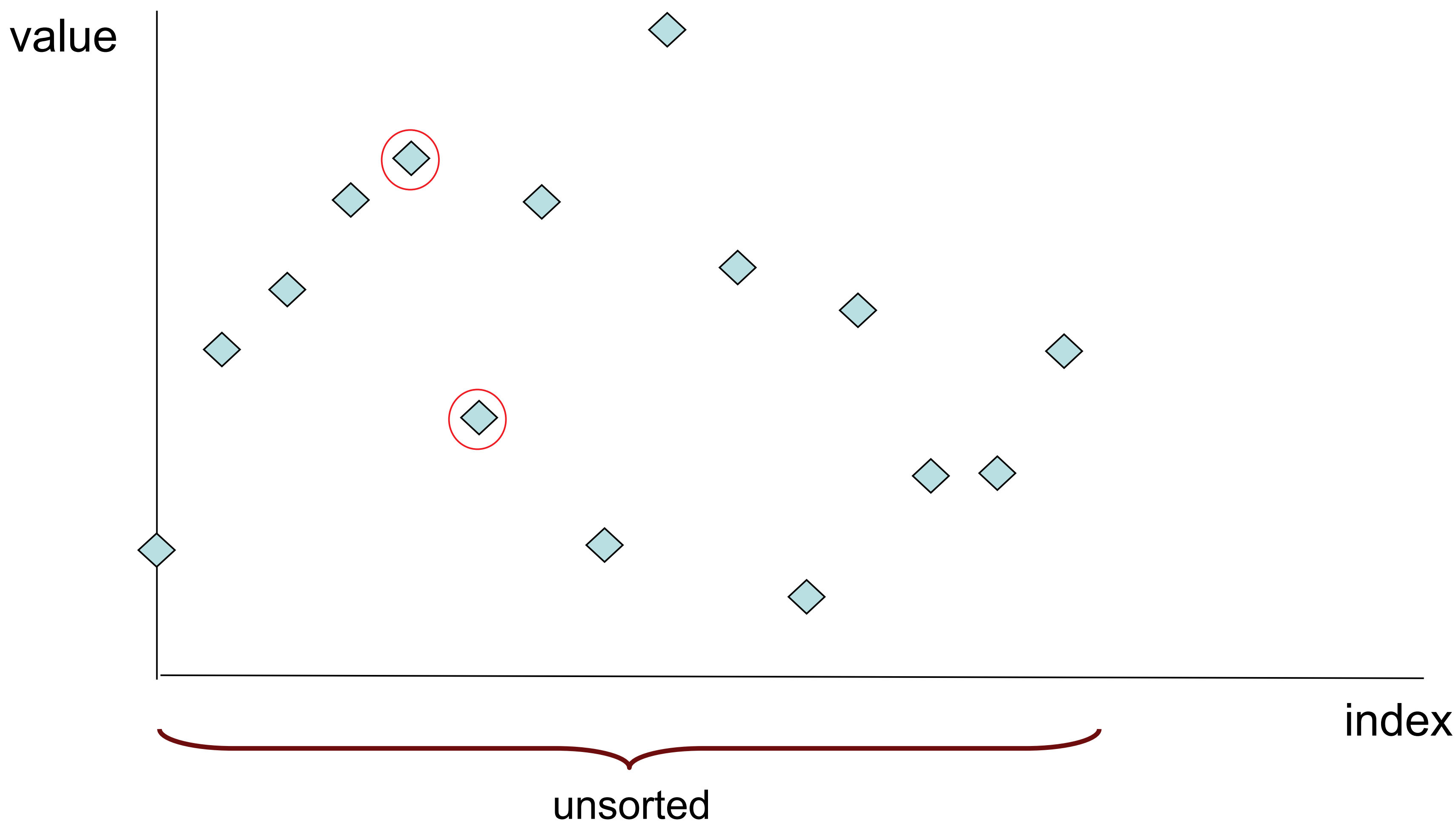
Bubble Sort



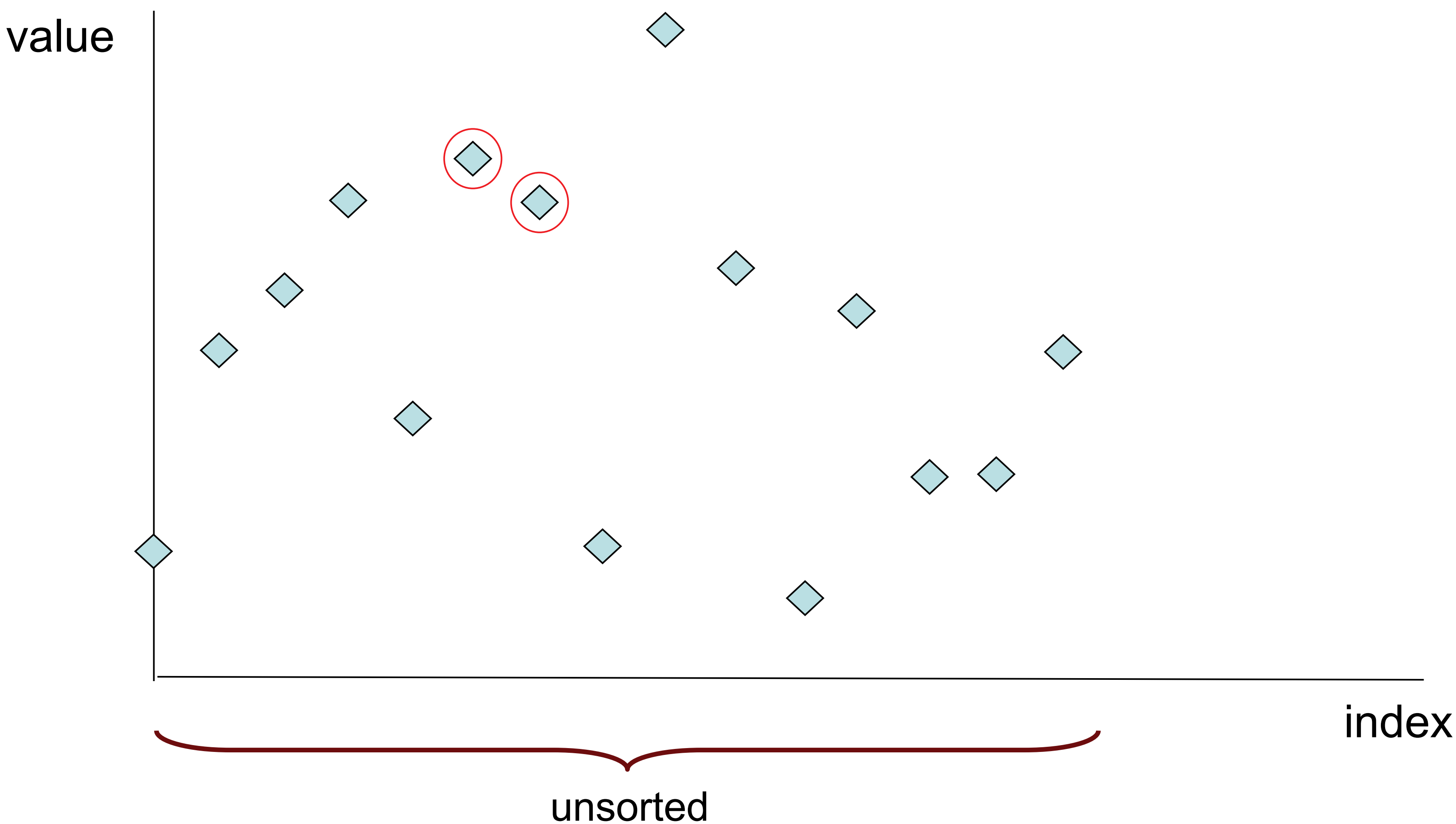
Bubble Sort



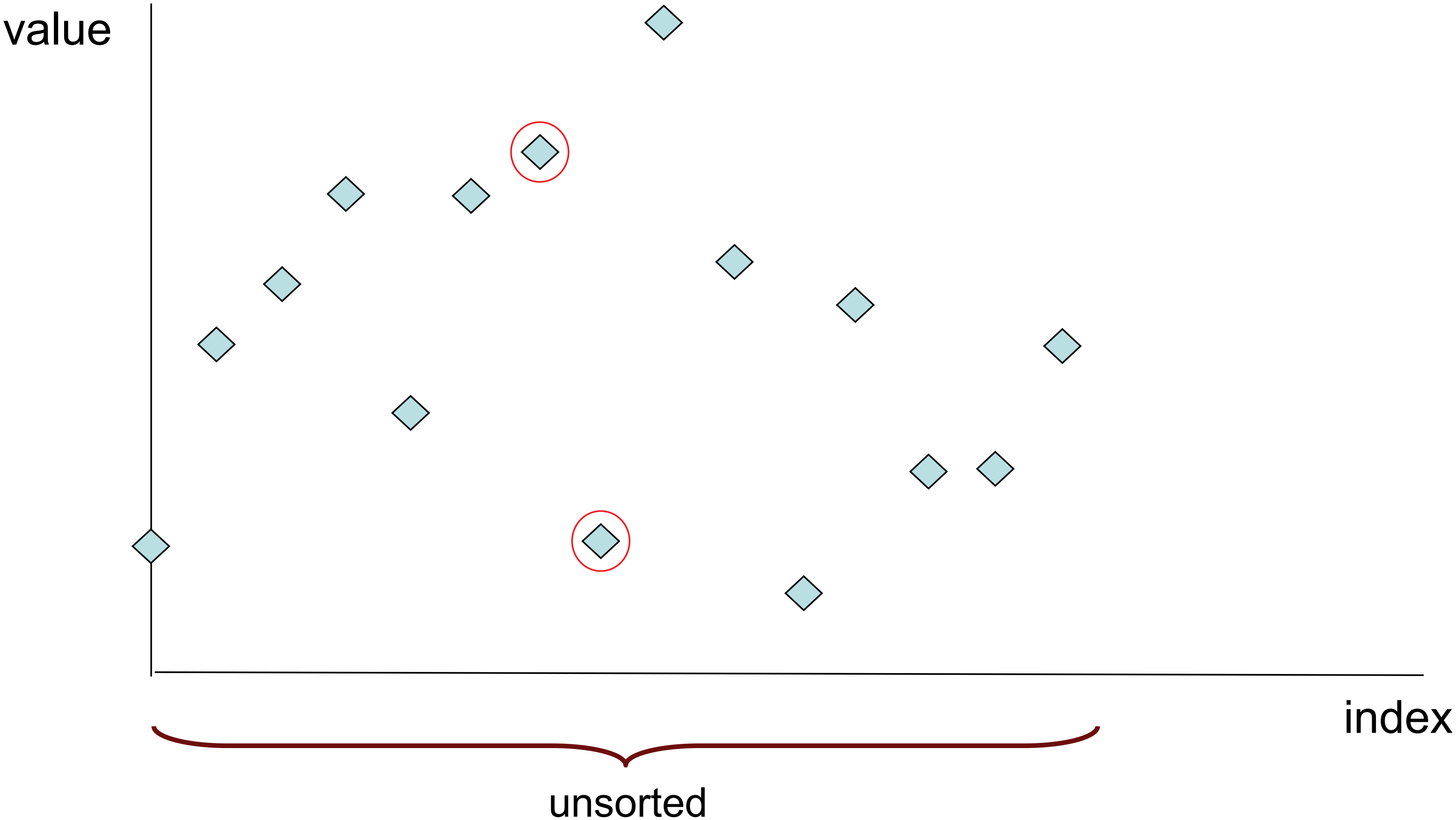
Bubble Sort



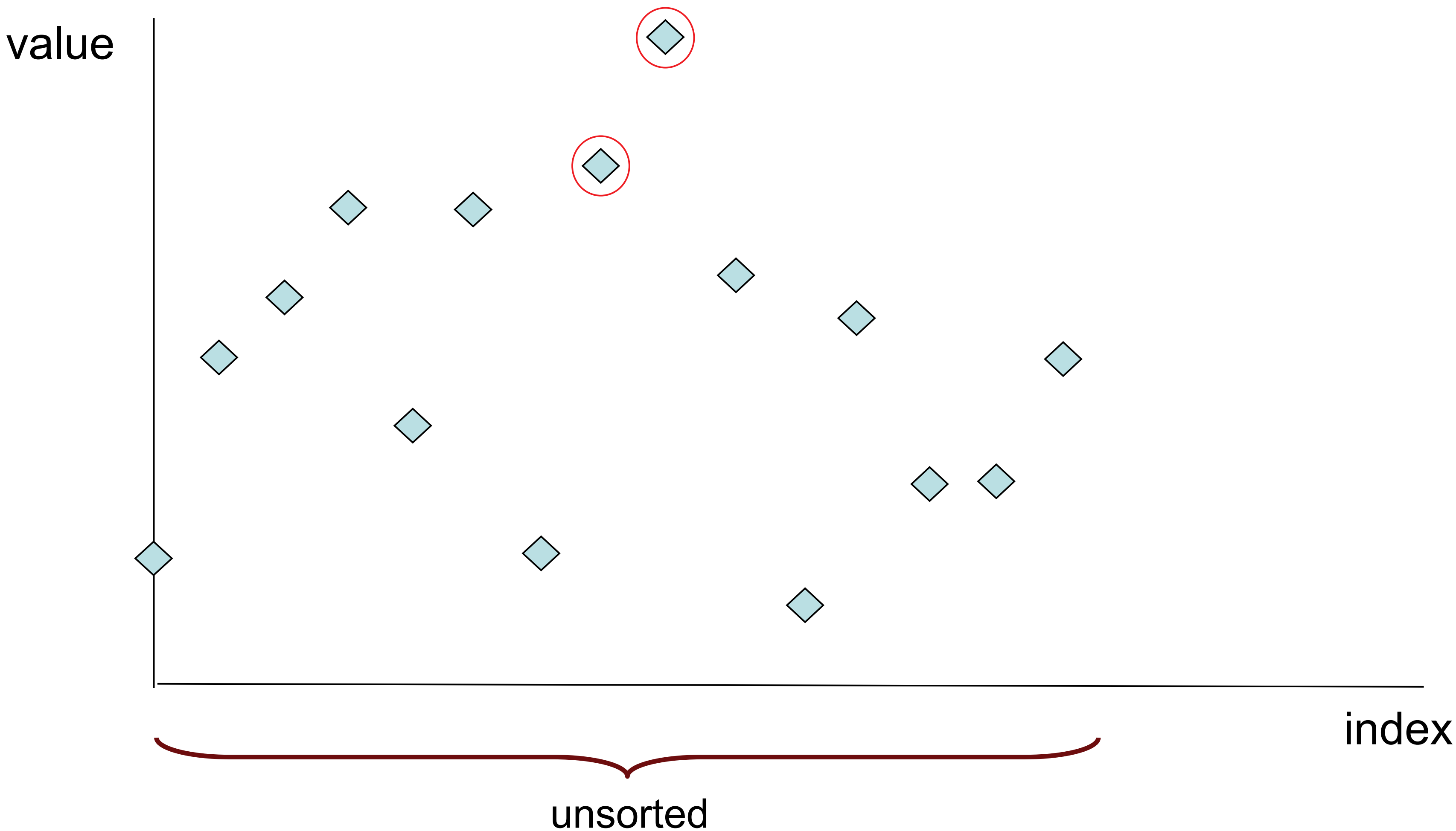
Bubble Sort



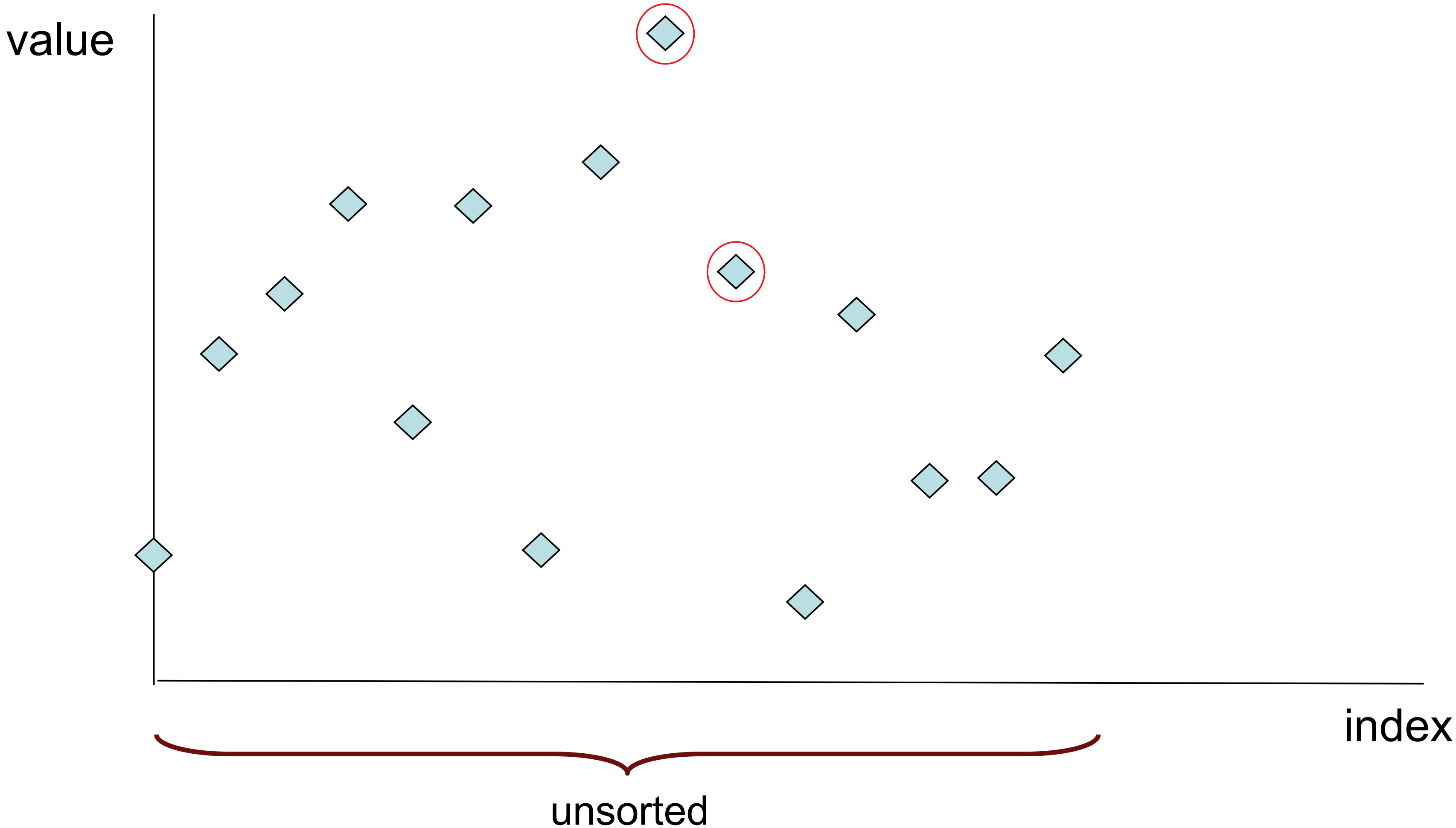
Bubble Sort



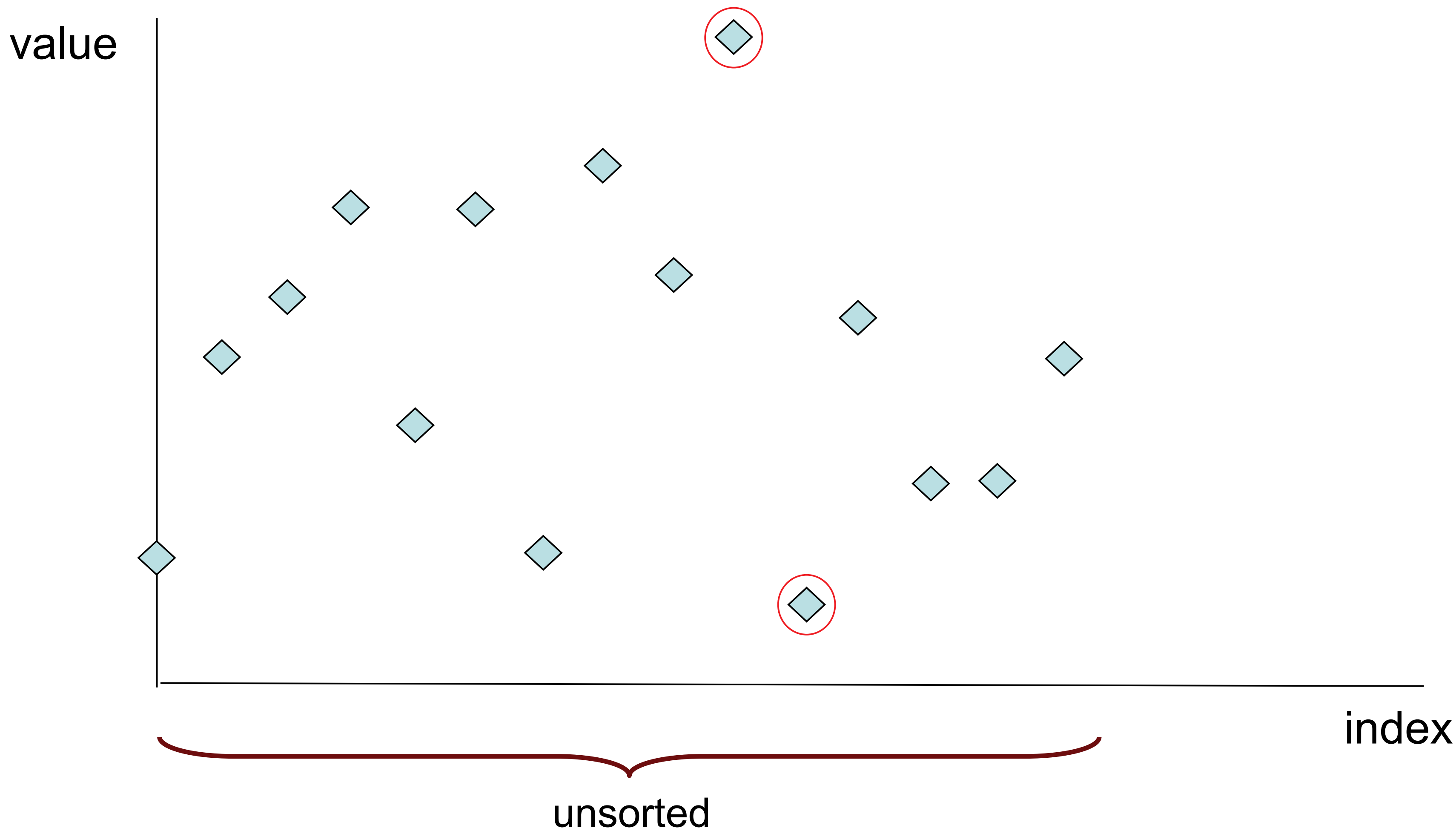
Bubble Sort



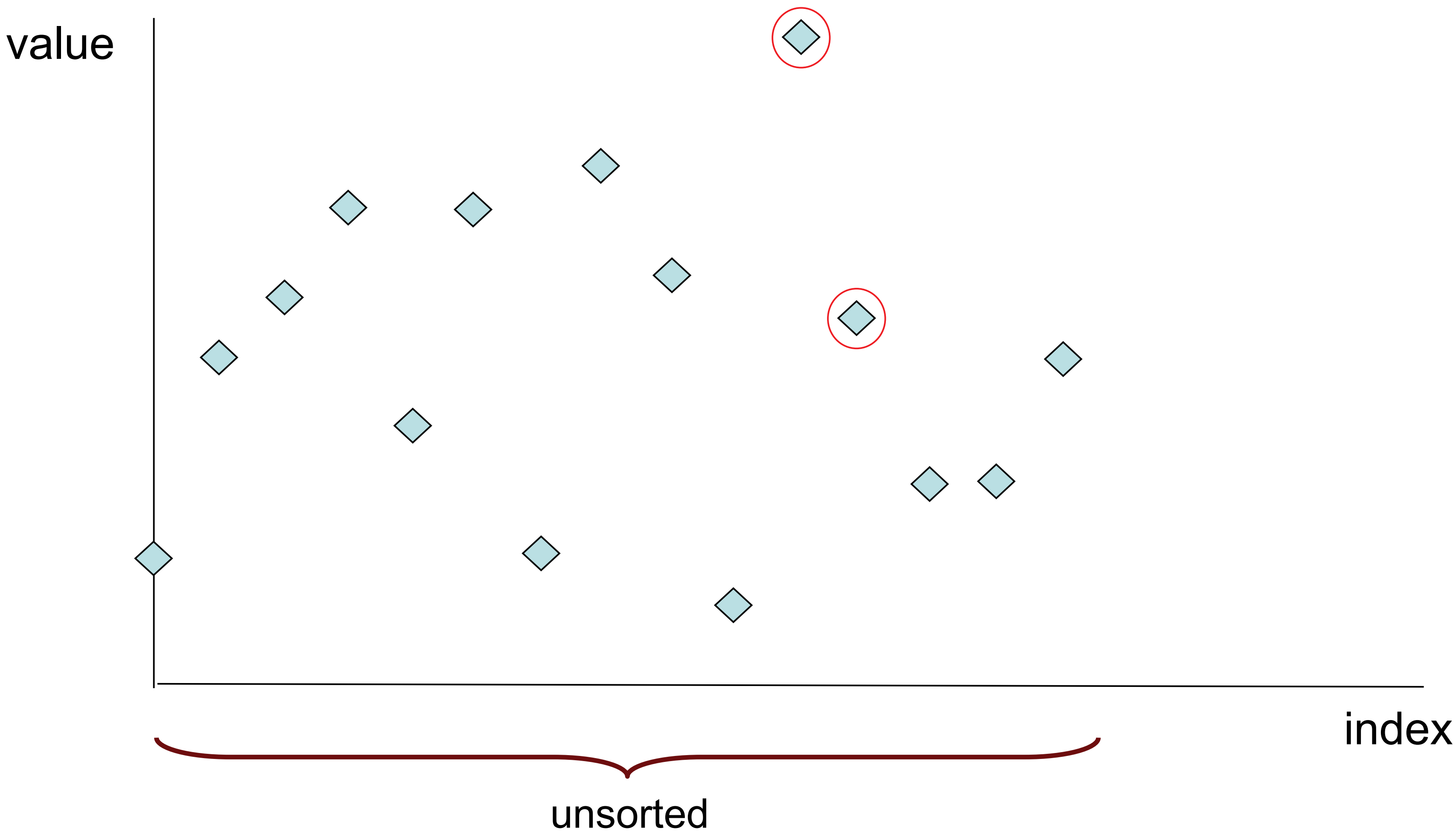
Bubble Sort



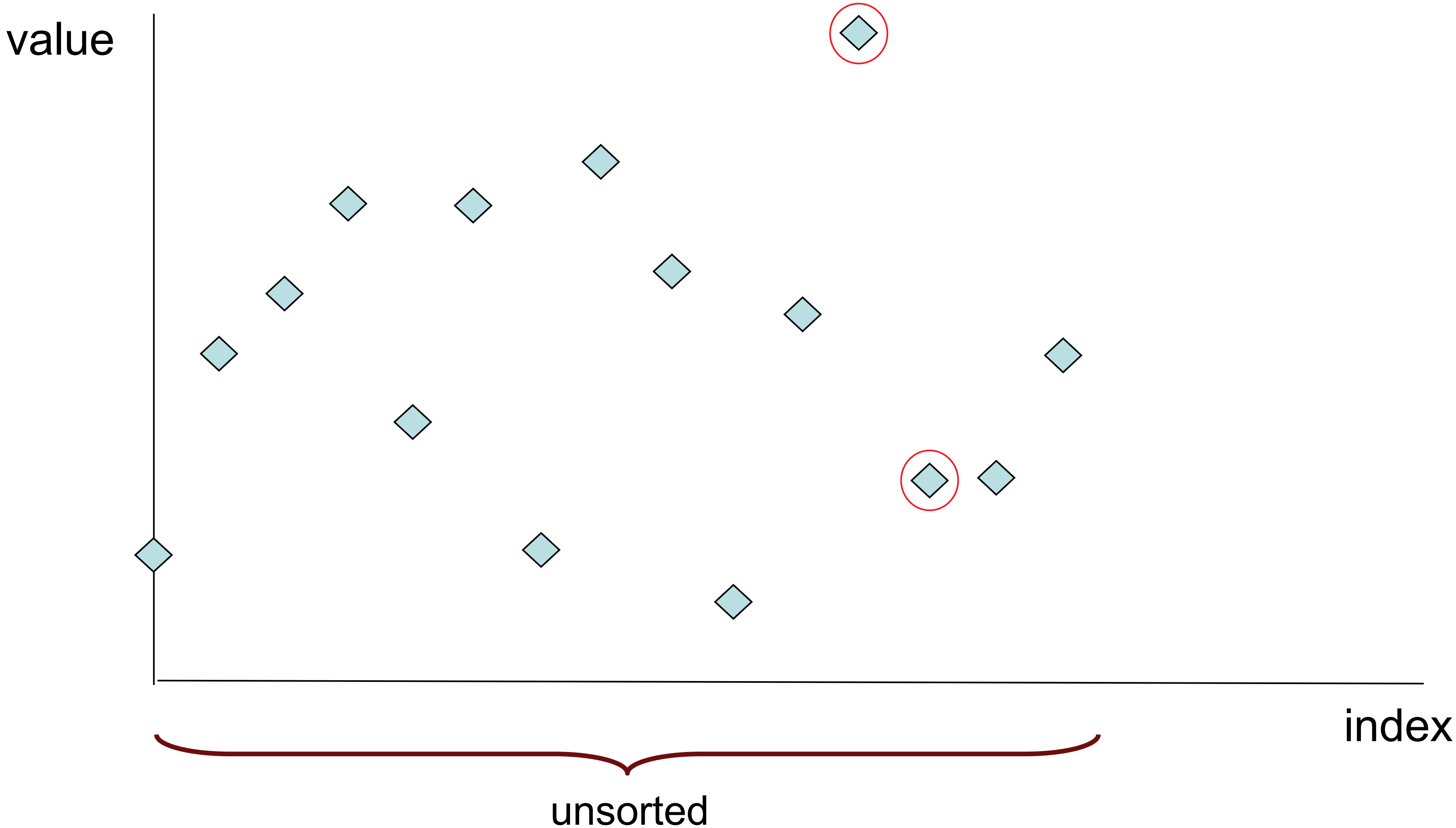
Bubble Sort



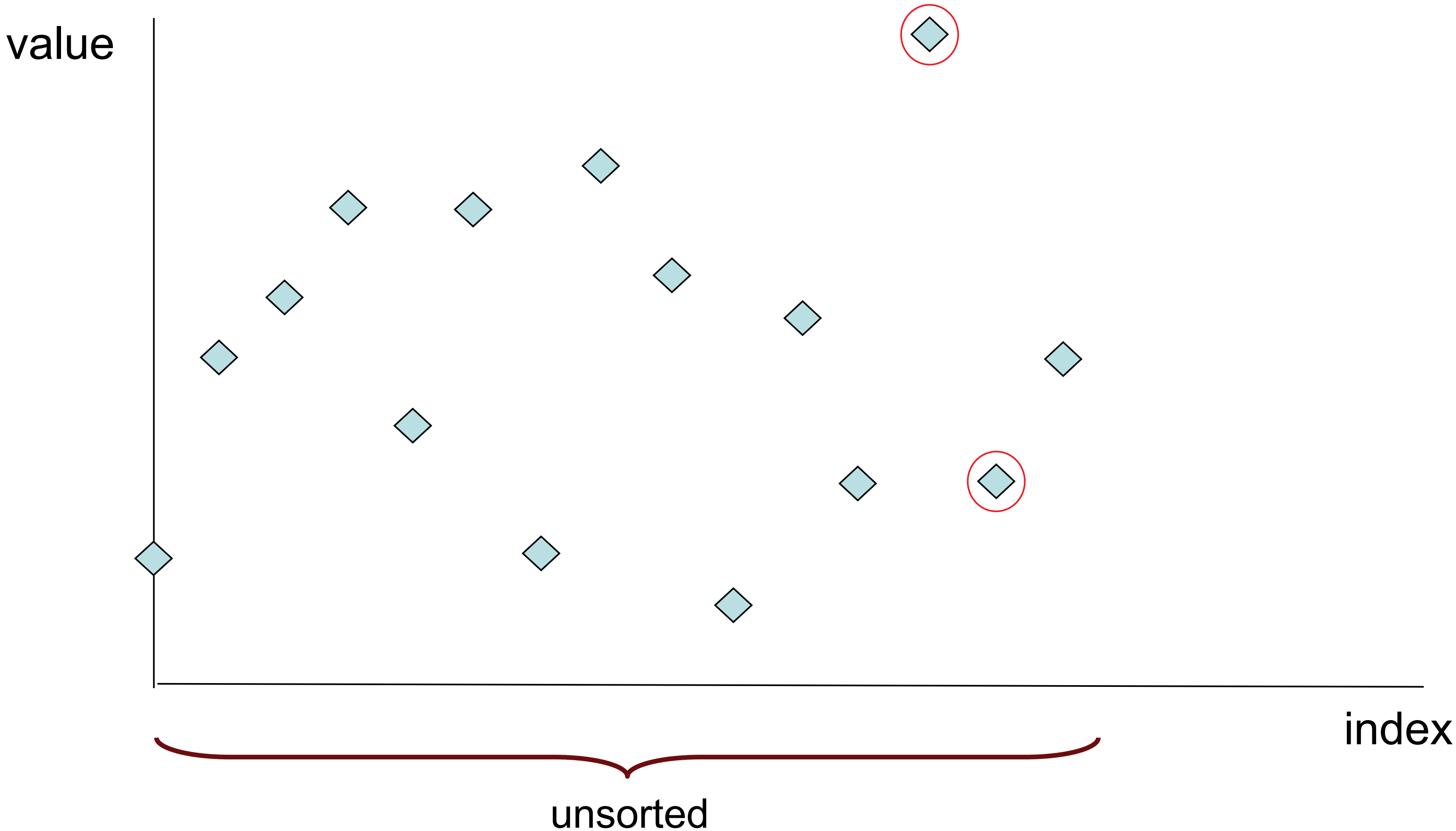
Bubble Sort



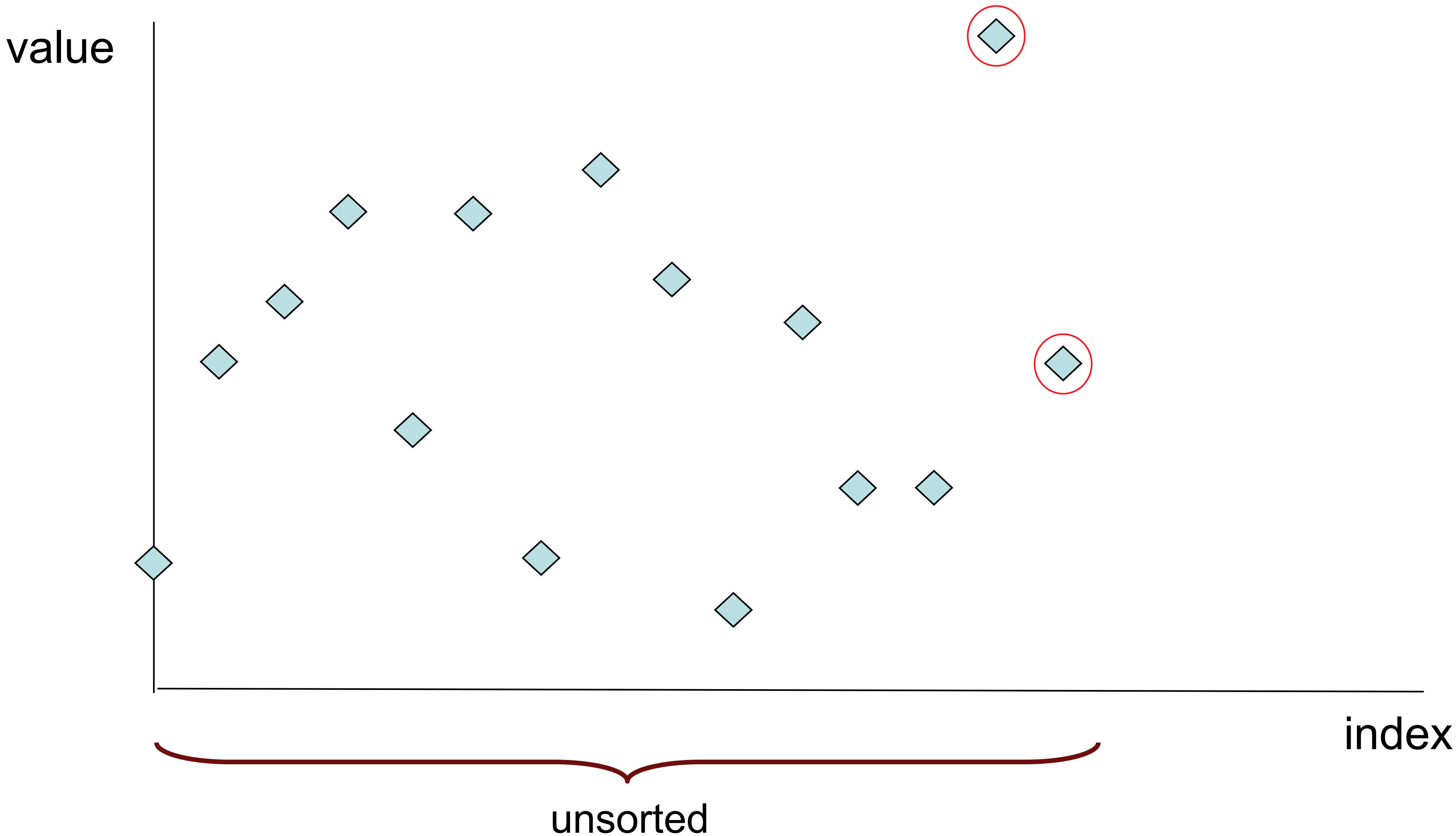
Bubble Sort



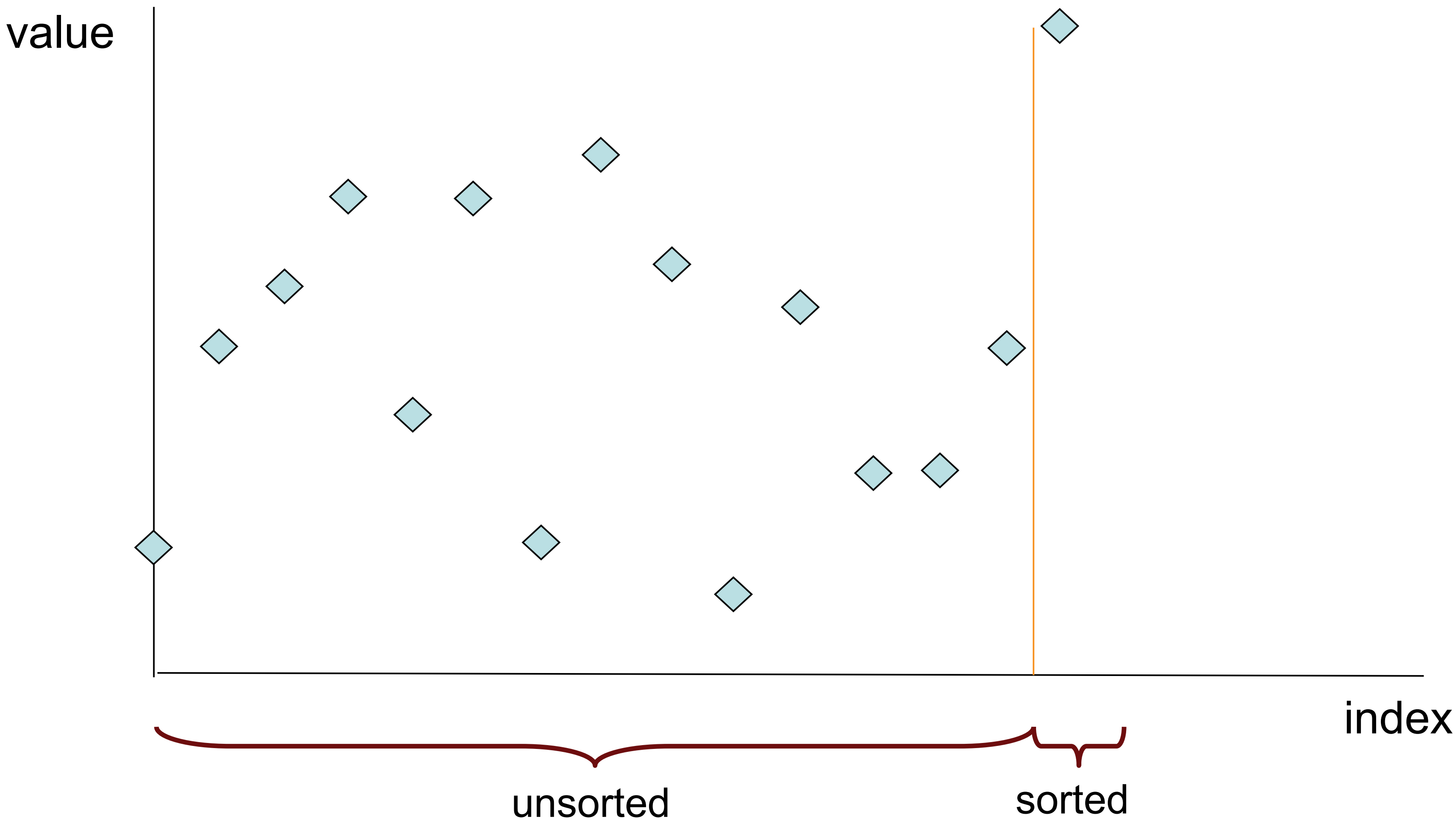
Bubble Sort



Bubble Sort



Bubble Sort



Bubble Sort

- Split array in two slices:
  - Front slice is unsorted (initially whole array)
  - Back slice is sorted (initially empty)
- Single sweep through unsorted slice:
  - Swap values of adjacent pairs that are unsorted ✓ slide #7
- Observation:
  - When nothing was swapped, the array is sorted
  - `bool bubble ( El data [], Slice unsorted )` to do...
- Problem solved?

```
void bubble_sort ( El data [], int length )
{
   while ( !bubble ( data, mkSlice (0, length-1) ) )
      length-- ;
}
```

Bubble

```
bool bubble ( El data [], Slice s )
{// Pre-condition:
    assert (valid_slice (s)) ;
//  Post-condition:
//  maximum of data[s.from]..data[s.to] is at data[s.to]
//  if result is true then sorted (data, s)
    bool no_swapping_needed = true ;
    for ( int i = s.from ; i < s.to ; i++ )
       if ( data[i] > data[i+1] )
       {
          swap ( data, i, i+1 ) ;
          no_swapping_needed = false ;
       }
    return no_swapping_needed ;
}
```

bubble sort live

What have we done?

- Array algorithms: insertion, slice
- Sorting algorithms: insertion sort, selection sort, bubble sort