COMP 250
INTRODUCTION TO COMPUTER SCIENCE

Lecture 14 – Iterative Sorting Algorithms

Roman Sarrazin-Gendron, Winter 2020

Slides very much based on Michael Langer and Giulia Alberini

SORTING

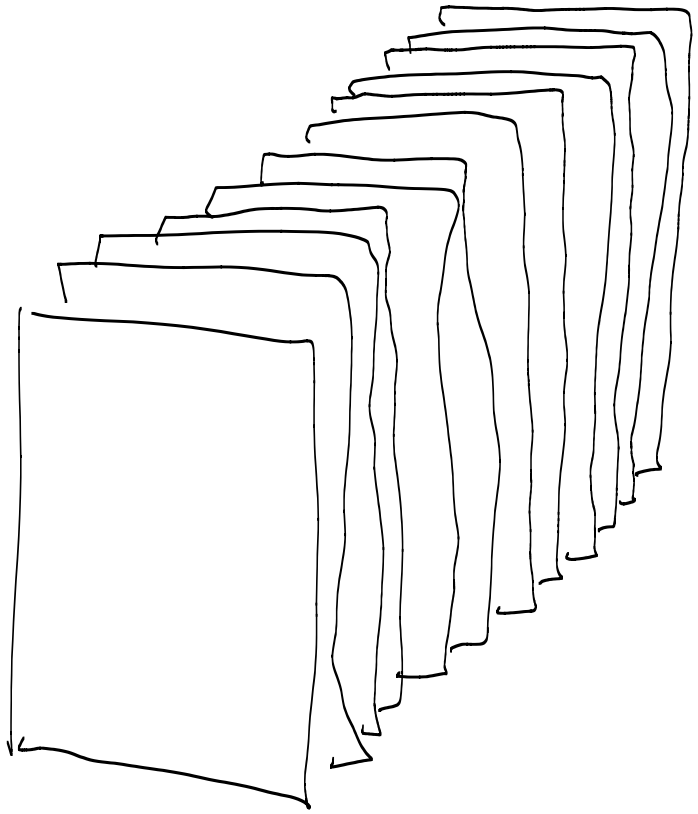
BEFORE

3
17
-5
-2
23
4

AFTER

-5
-2
3
4
17
23

EXAMPLE 1: SORTING EXAMS BY LAST NAME



EXAMPLE 2 : EMAIL PACKETS

- When you send a large file by email, it is broken down in small pieces (called packets).

- Each packet takes an independent network path to the destination
- Packets then must be put together in the correct order.

SORTING ALGORITHMS

- Bubble sort
- Selection sort
- Insertion sort

today $O(N^2)$

- Mergesort
- Heapsort
- Quicksort

later $O(N \log N)$

SOME EXTRA MATERIAL

- Many algorithms for sorting exist
- The best way to develop an intuition for how they work is to see them executed on examples.
- We will discuss some small examples today, but there are excellent videos demonstrating this.

- Some sorting algorithms : https://www.youtube.com/watch?v=ZZuD6iUe3Pc (credit to Viktor Bohush)

- Sorting is very often covered on tech job interviews.
- Example of such interview: https://www.youtube.com/watch?v=k4RRi_ntQc8

SORTING ALGORITHMS

- Today we are concerned with algorithms, not data structures.
- The following algorithms are independent of whether we use an array list or a linked list.
- We will look at how many iterations each method requires, i.e. how many times each element is visited in different cases.
- **Objective**: understand the differences between the three methods.

BUBBLE SORT

Editor's note: the name is a confusing analogy with bubbles rising in a liquid. It might not make sense to you and that's fine because it doesn't really make sense to me either.

Repeatedly loop (iterate) through the list.

For each iteration,

    if two neighboring elements are in the wrong order,

        then swap them.

REMINDER ABOUT SWAPPING

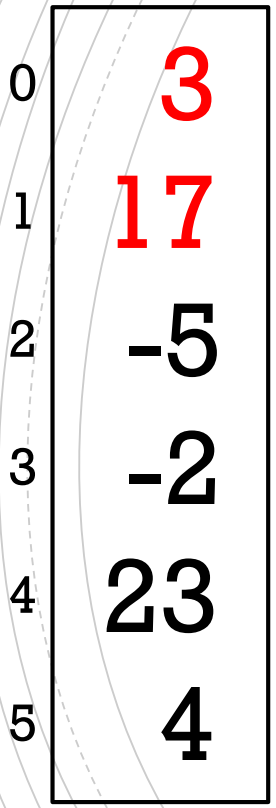
The following does not work:

```
x = y
y = x
```

Rather, you need to use a temporary variable:

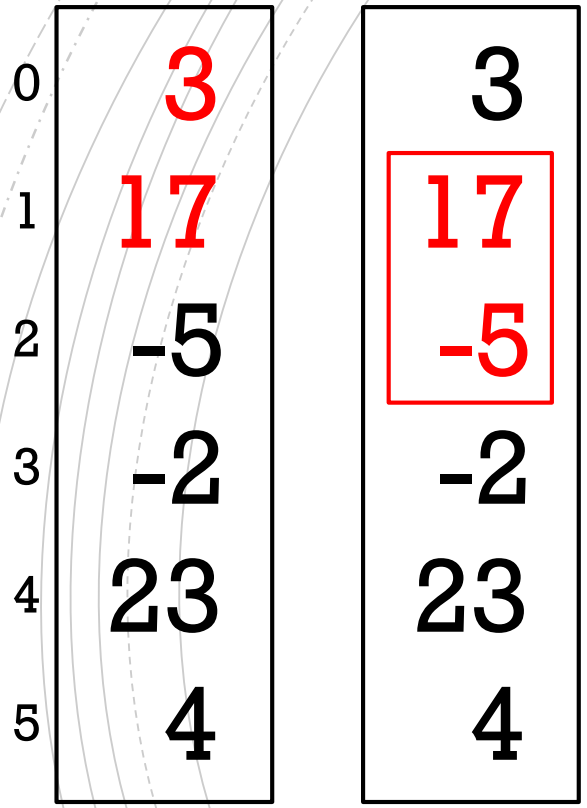
```
tmp = y
y = x
x = tmp
```

EXAMPLE: FIRST PASS




```
if  list[ 0 ] > list[ 1 ]
        swap( list[ 0 ], list[1 ] )
```

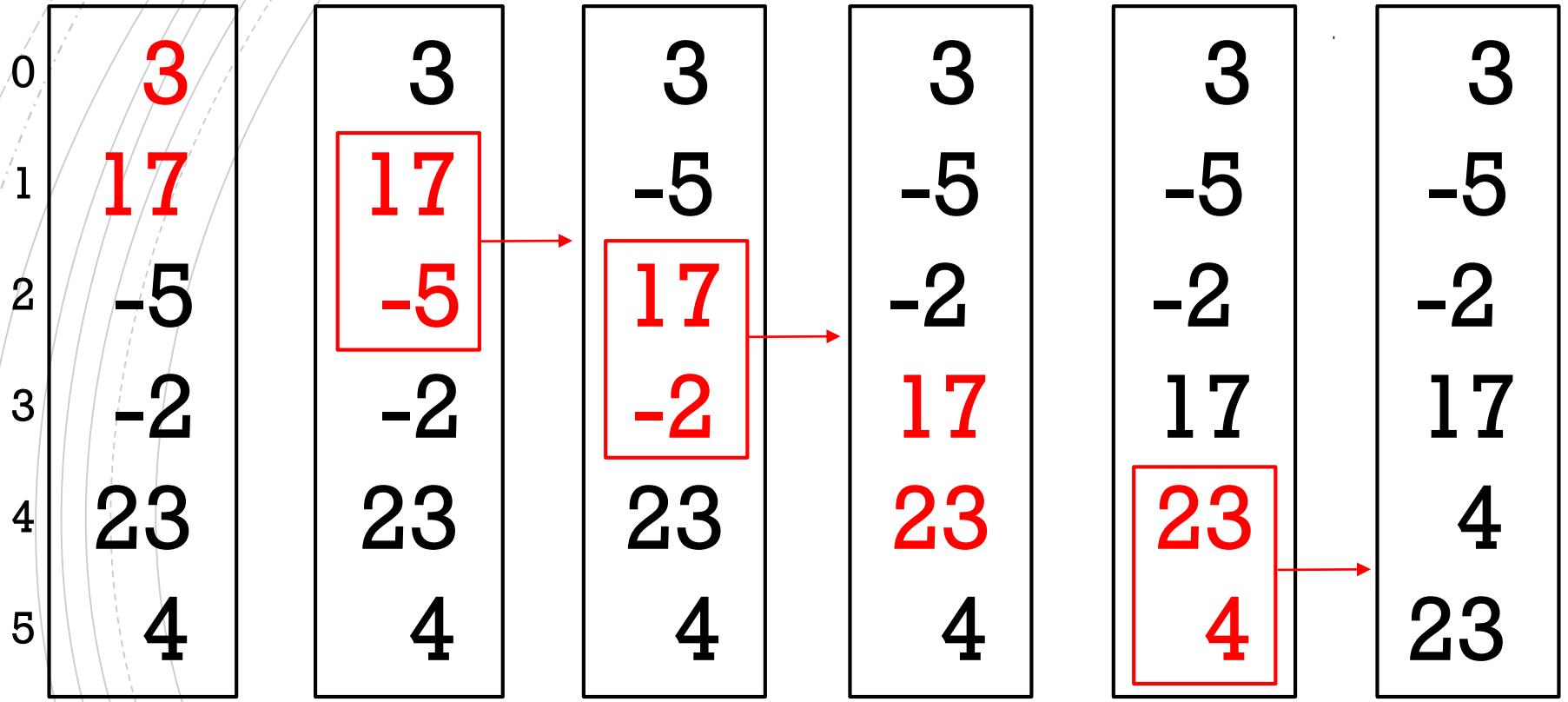
EXAMPLE: FIRST PASS



Indicates elements get swapped

```
if  list[ 1 ] > list[ 2 ]
        swap( list[ 1 ], list[  2 ] )
```

EXAMPLE: FIRST PASS



Indicates elements get swapped

WHAT CAN WE SAY AT END OF THE FIRST PASS?

Q: Where is the largest element ?

A:

Q: Where is the smallest element?

A:

WHAT CAN WE SAY AT END OF THE FIRST PASS?

Q: Where is the largest element ?

A: It must be at the end of the list (position N-1).

Q: Where is the smallest element ?

A: Anywhere (except position N-1).

BUBBLE SORT ALGORITHM

```
repeat {
    continue = false
    for i = 0 to N – 2                    //  N-1 is the last index
        if  list[ i ] > list[ i + 1 ]{
            swap( list[ i ], list[ i + 1 ] )
            continue = true
        }
} until continue == false
```

BUBBLE SORT ALGORITHM

```
ct = 0
repeat {
    continue = false
    for i = 0 to N – 2 – ct  {          //  N-1 is the last index
        if  list[ i ] > list[ i + 1 ]{
            swap( list[ i ], list[ i + 1 ] )
            continue = true
        }
        ct++    //  now list[ N - ct, … N-1] is sorted
    }
}  until continue == false
```

SELECTION SORT

- Partition the list into two parts:

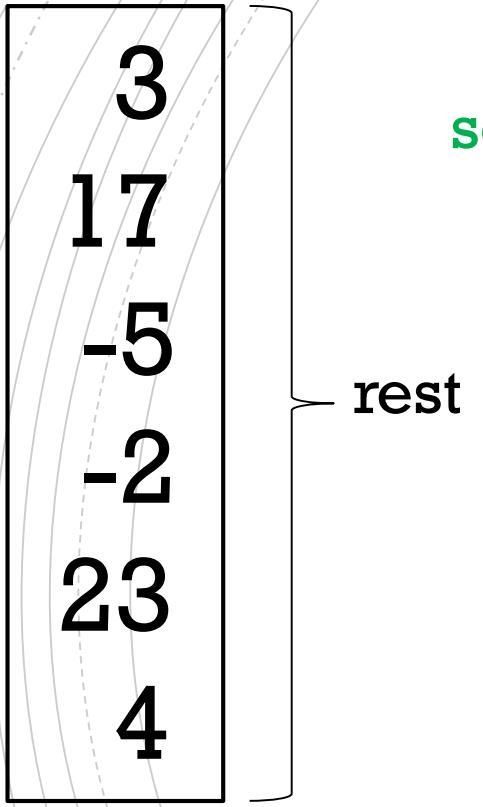
(1) a sorted part, initially empty, in which elements are sorted

(2) a "rest" part, initially of size N, in which elements can be in any order.

Repeat list.size times:

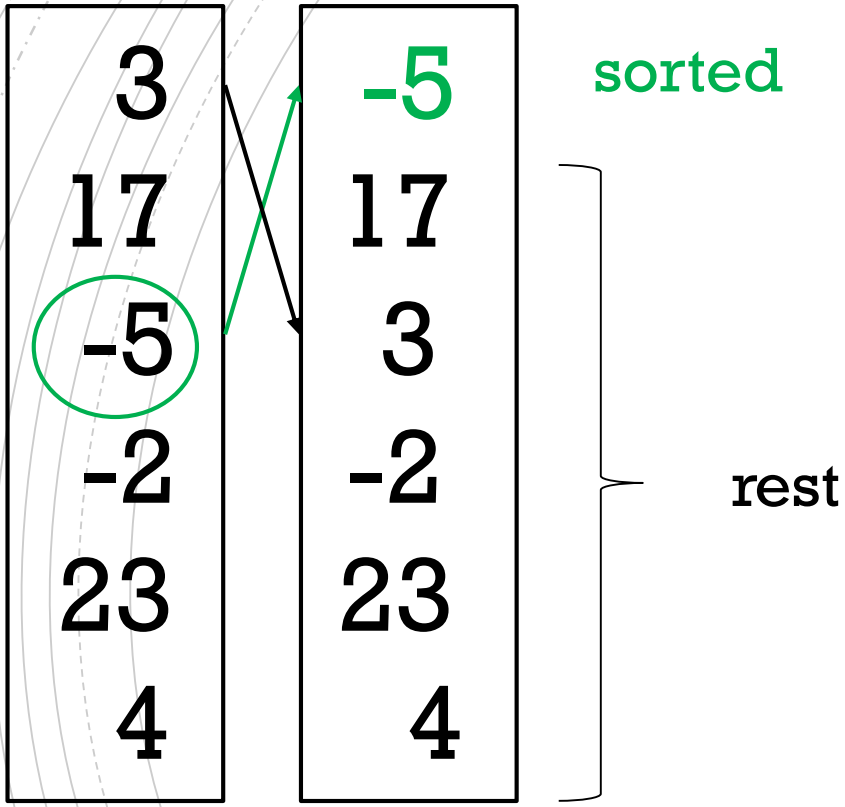
- find the smallest element in **"the rest"**
- swap it with the first element in **"the rest".**
  - this element is now in the **sorted** part.

EXAMPLE

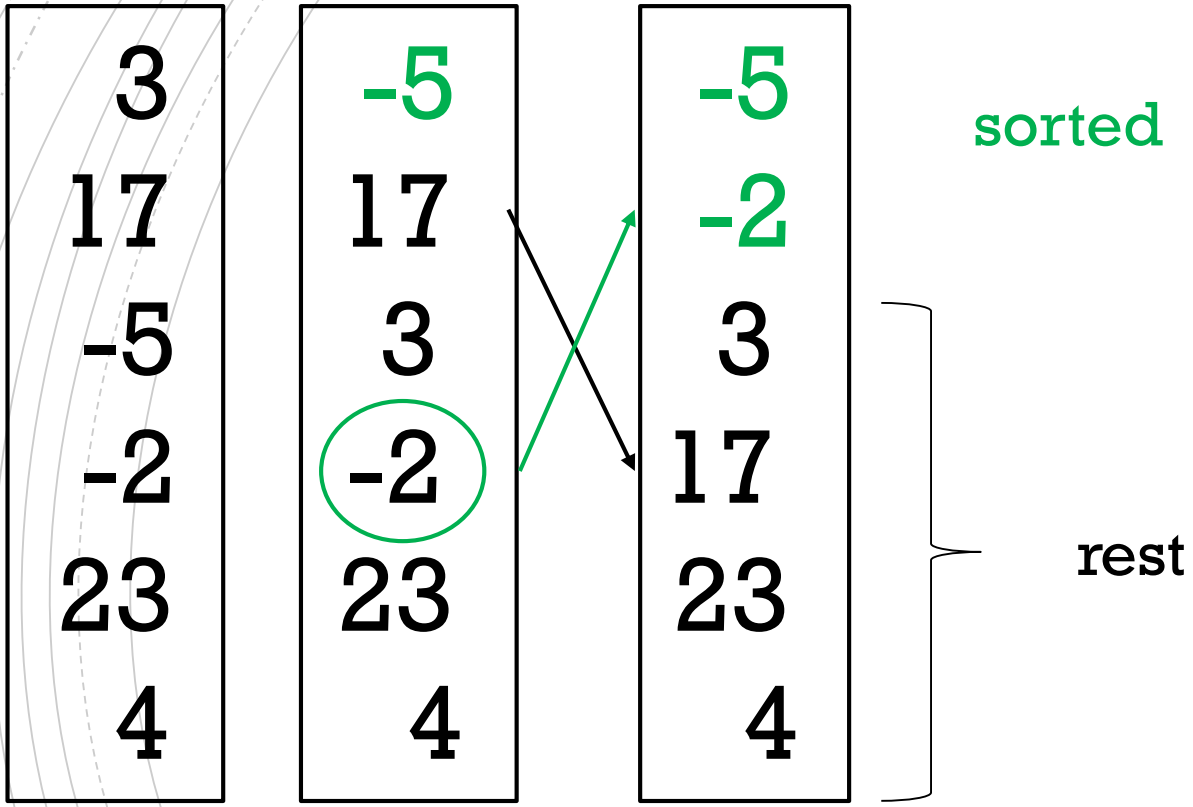


sorted part is empty

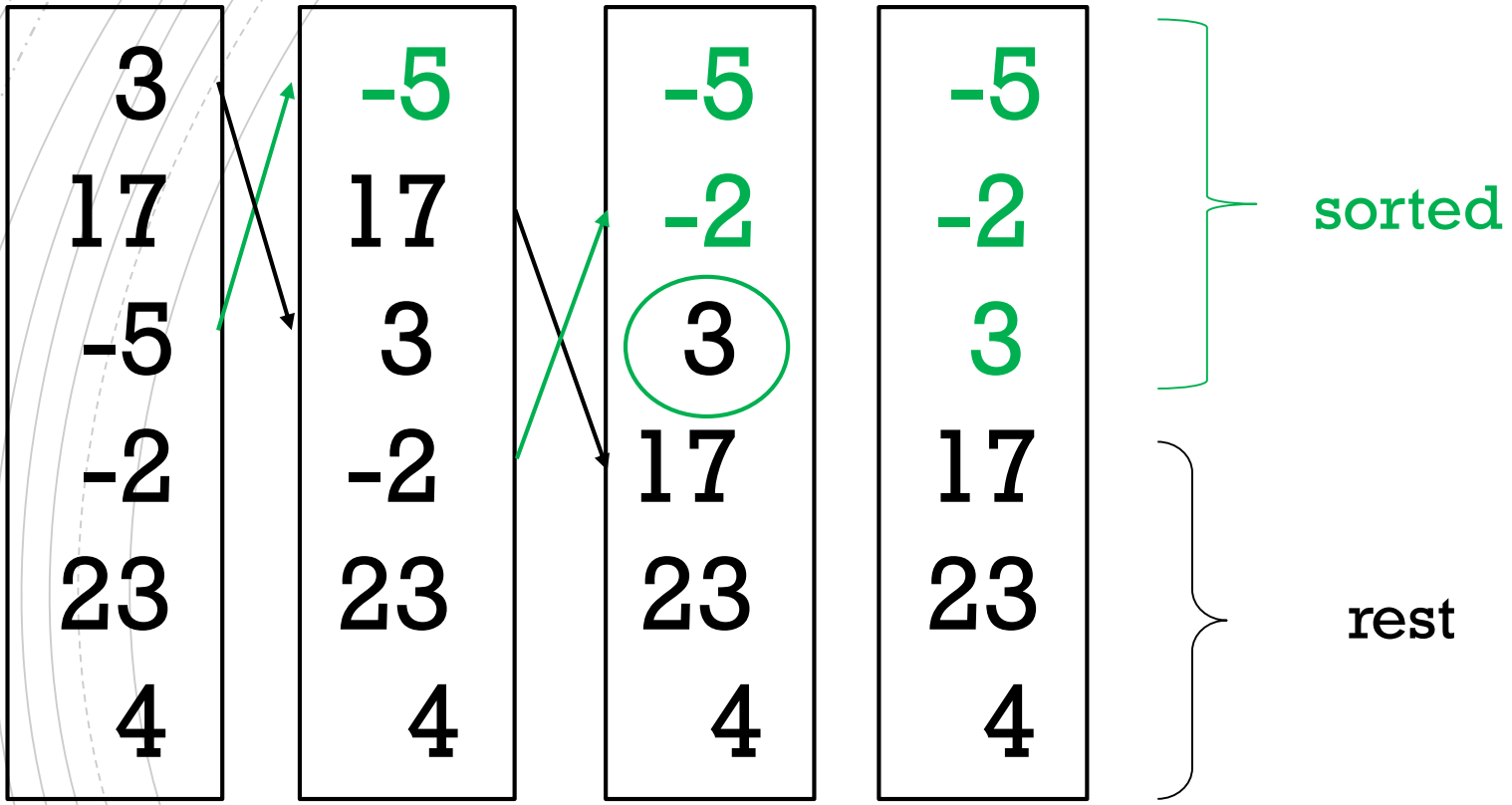
EXAMPLE



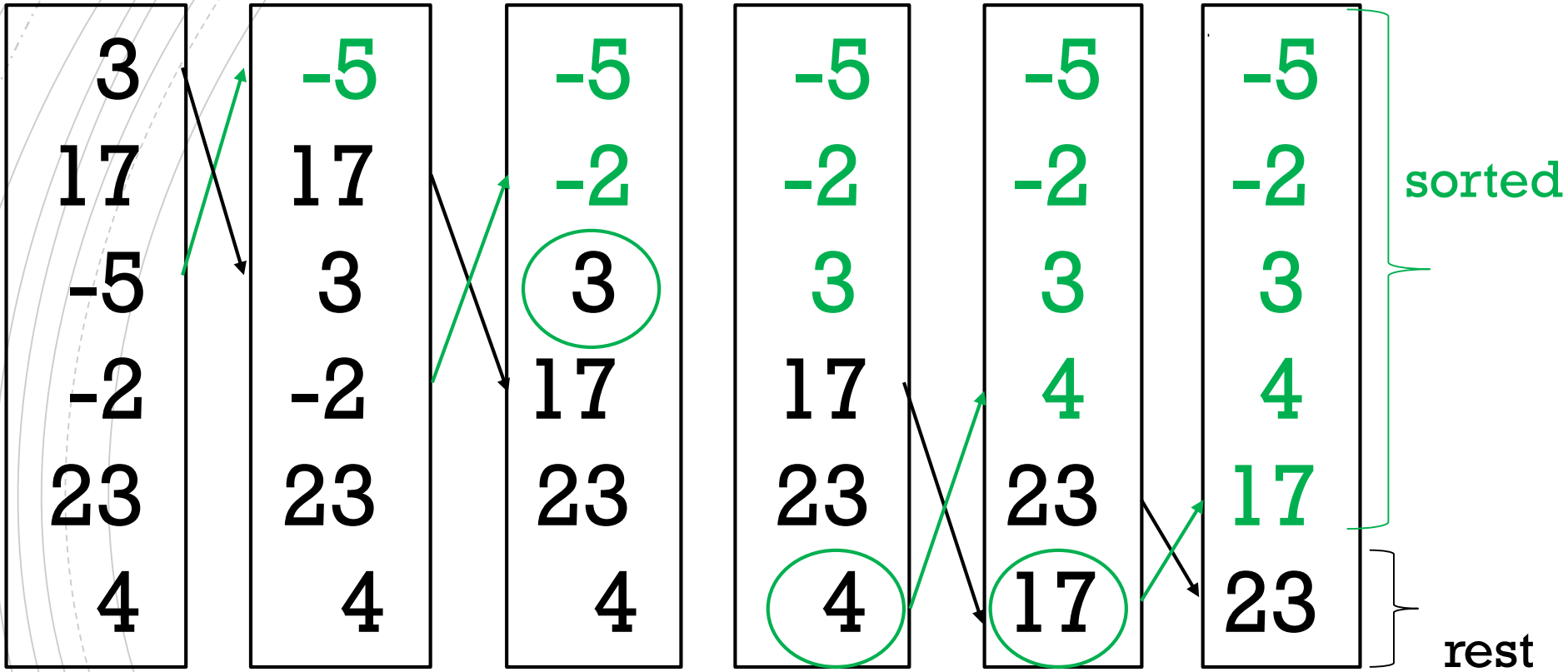
EXAMPLE



EXAMPLE



EXAMPLE


3
17
-5
-2
23
4
-5
17
3
-2
23
4
-5
-2
3
17
23
4
-5
-2
3
17
23
4
-5
-2
3
4
23
17
-5
-2
3
4
17
23
sorted
rest

SELECTION SORT ALGORITHM

```
for  i = 0  to N-2  {                    //  repeat  N times
    index = i                            //  Take the first element in the rest.
    minValue = list[ i ]                 //   It has the min value so far.
    for k = i+1  to  N-1  {
       if ( list[k] < minValue ){        //  For each other element in rest,
           index = k                     //   if it is smaller than the min value,
           minValue = list[k]            //    then remember its index.
       }                                 //    It is the new min value.
    }
    if ( index != i )                    //  Swap if necessary
        swap(  list[i], list[ index ] )
}
```

SELECTION SORT

```
for  i = 0  to N-2
        for k = i+1  to  N-1
              .......
```

Q: how many passes through the inner loop?

SELECTION SORT

```
for  i = 0  to N-2
        for k = i+1  to  N-1
                .......
```

Q: how many passes through inner loop?

A: N-1 + N-2 + N-3 + ... + 2 + 1

SELECTION SORT

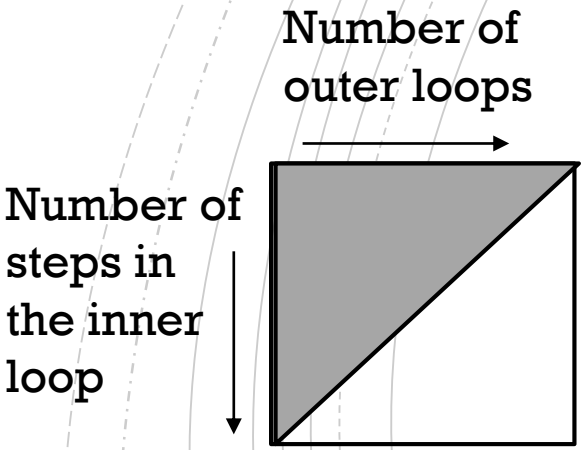
```
for  i = 0  to N-2
        for k = i+1  to  N-1
              .......
```

Q: how many passes through inner loop?

A: N-1 + N-2 + N-3 + ... + 2 + 1

= N (N-1) / 2

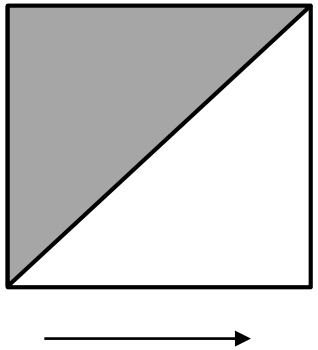
COMPARISON



Bubblesort

```
repeat
   for i = 0 to N – 2 – ct
until continue == false
```

We can terminate outer loop if there are no swaps during a pass.

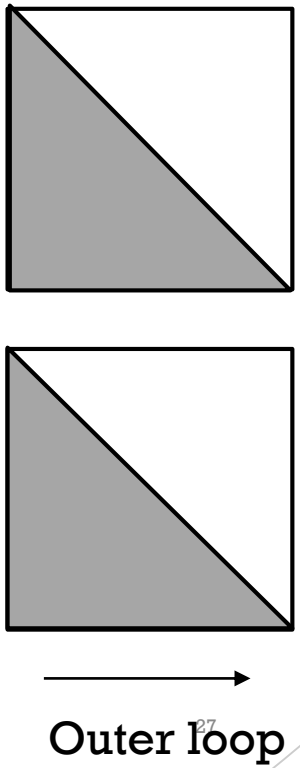


Best case

Worst case

Selection sort

```
for  i = 0  to N-2
    for k = i+1  to  N-1
```



INSERTION SORT

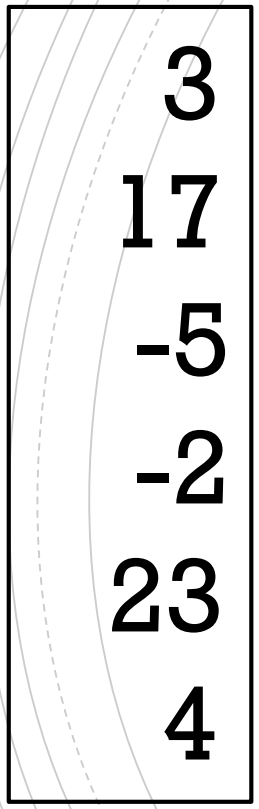
```
for k = 1 to N- 1 {

        Insert list element at index k into its correct
          position with respect to the elements
          at indices 0 to k – 1

}
```

INITIAL LIST

3
17
-5
-2
23
4

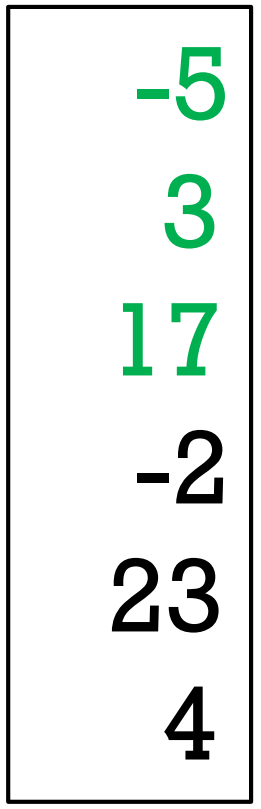
THREE STEPS LATER

Initial list

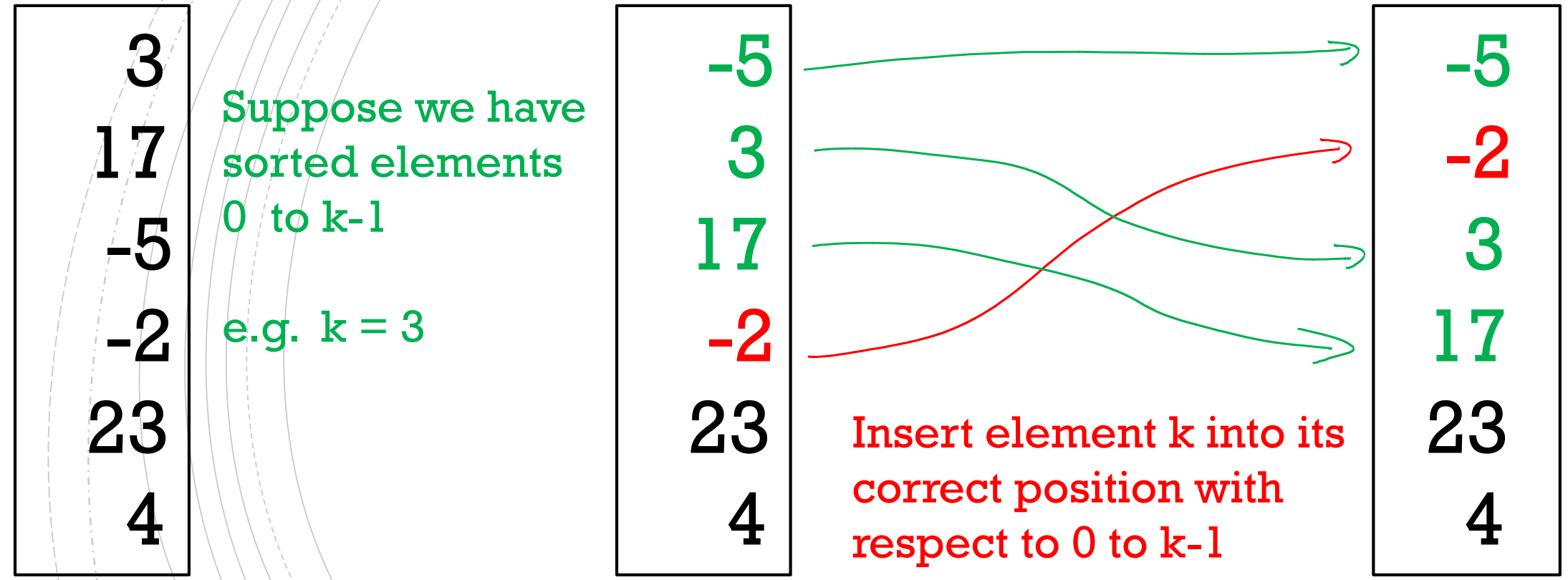


Suppose we have sorted elements 0 to k-1

e.g. k = 3



THE FOURTH STEP



MECHANISM OF INSERTION SORT

- Mechanism is similar to inserting (adding) an element to an array list:

- Shift all elements ahead by one position to make a hole, and then fill the hole.

INSERTION SORT ALGORITHM

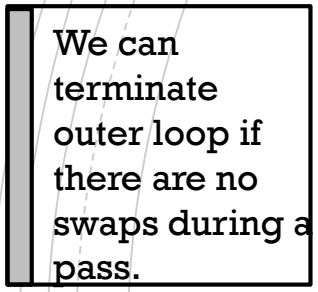
```
for k = 1 to N - 1   {                    //  index of element to move

     elementK =  list[k]
      i = k
      while (i > 0) and ( elementK  < list[ i - 1]){
          list[i] =  list[i - 1]        // copy to next
          i  = i -1
       }
       list[i] = elementK               //  paste elementK
}
```

COMPARISON OF 3 METHODS

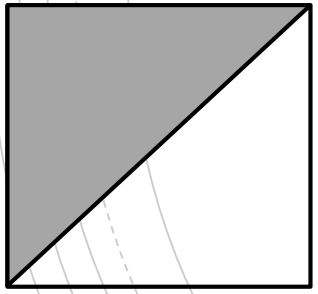
Bubblesort

```
repeat
    for i = 0 to N – 2 – ct
until continue == false
```

Best case

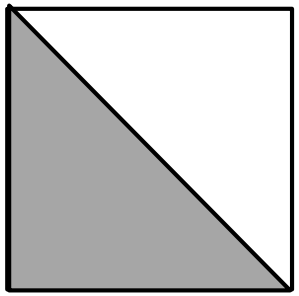


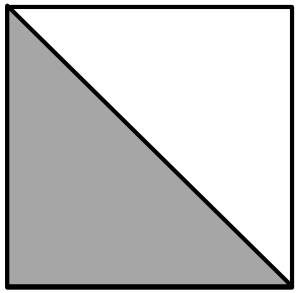
Worst case



Selection sort

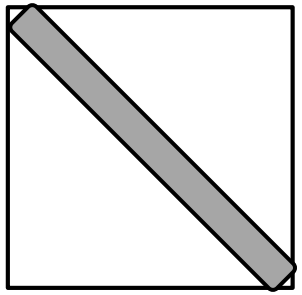
```
for  i = 0  to N-2
    for k = i+1  to  N-1
```

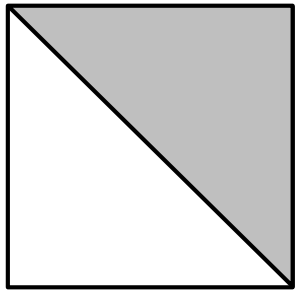




Insertion sort

```
for k = 1 to N - 1  {
    while ….
```





Performance depends highly on initial data. Also, it depends on implementation (array vs. linked list), e.g. what is cost of swap and 'shift'.

ASSIGNMENT2 IS OUT

- You can still submit assignment 1 until tomorrow
- Assignment 2 is out! Due in two weeks
- Quiz 2 is Friday