COMP 250

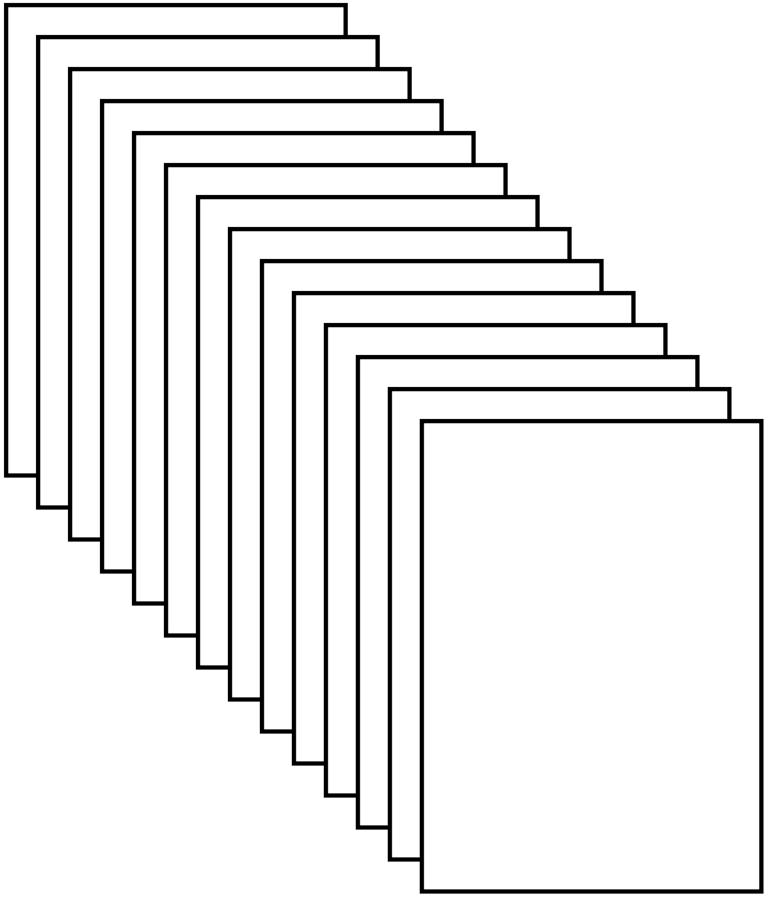
Lecture 12

Algorithms for Sorting a List:

bubble sort
selection sort
insertion sort

Feb. 2, 2022

Example 1: sorting exams by last name



Example 2: Email packets

When you send a large file by email, it gets broken down into small pieces called “packets” and each packet takes an independent network path to the destination.

Then the packets need to be put together again in their correct order.

https://computer.howstuffworks.com/question525.htm

Some sorting algorithms are faster than others.
See visualization:

https://www.youtube.com/watch?v=ZZuD6iUe3Pc

Barak Obama knows about sorting…

https://www.youtube.com/watch?v=k4RRi_ntQc8

Sorting Algorithms

- Bubble sort
- Selection sort
- Insertion sort

today $O(N^2)$

- Mergesort
- Heapsort
- Quicksort

later $O(N \log N)$

Sorting Algorithms

Today we are concerned with algorithms, not data structures.

Today's algorithms can be implemented easily using an array list or a (doubly) linked list.

Notation for today...

	BEFORE		AFTER
0	3		-5
1	17		-2
2	-5	sort into increasing order	3
3	-2		4
4	23		17
5	4		23

Bubble Sort

Given a list of size N, arrange the elements in *increasing* order.

Pass through the list N times.

For each pass,

if two neighboring elements are in the wrong order,

then swap them.

The name invokes the (vague) metaphor of bubbles rising in a liquid.

Bubble Sort Algorithm

```
for i = 0  to  N − 1 {                    //  i-th pass
    for k = 0  to  N − 2   {
        if ( list[k] > list[k+1] ) {      //  wrong order
            list.swap(k,  k+1)
        }
    }
}
```

Example: first pass

0	3
1	17
2	-5
3	23
4	-2
5	4

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

Example: first pass

0	3	3
1	17	17
2	-5	-5
3	23	23
4	-2	-2
5	4	4

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

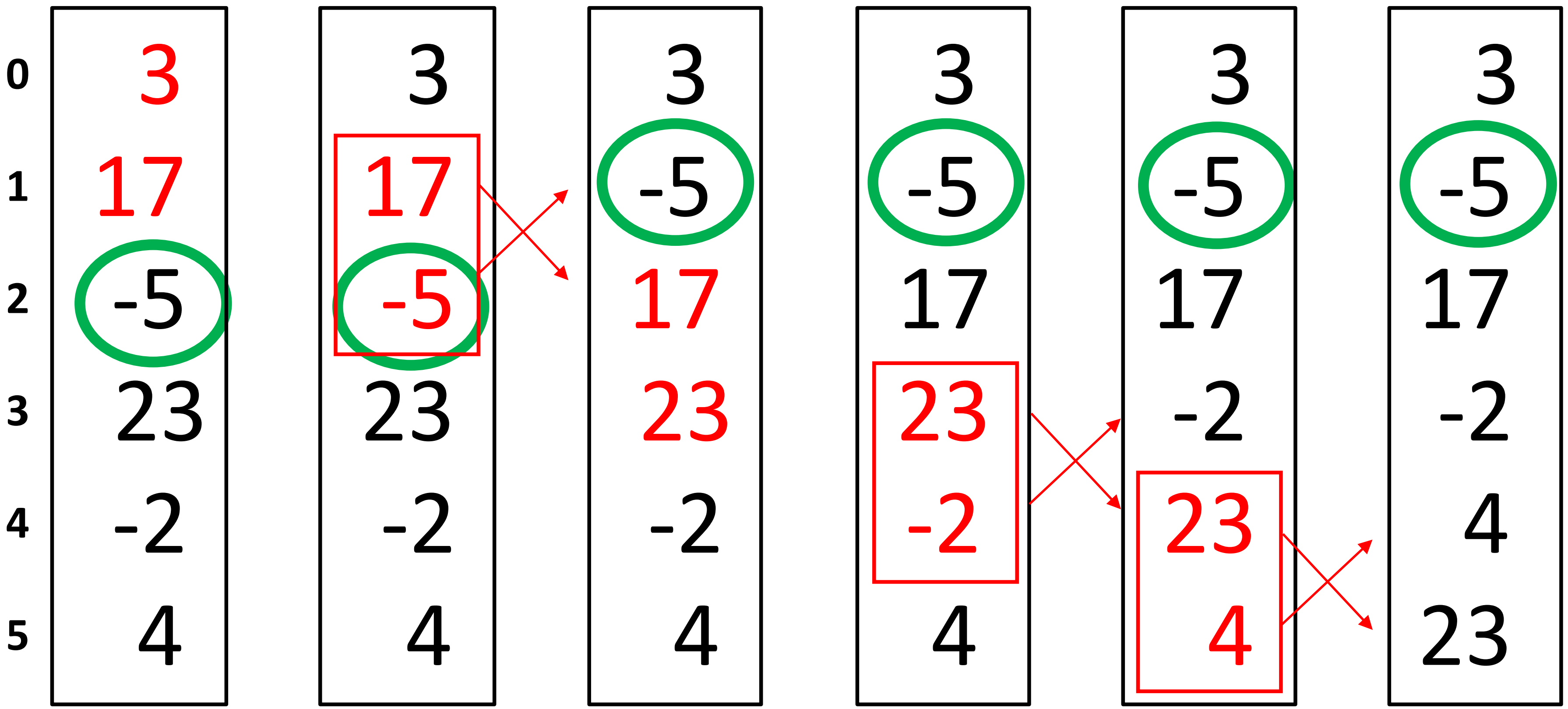
Indicates elements need to be swapped

Example: first pass

0	3	3	3	3	3	3
1	17	17	-5	-5	-5	-5
2	-5	-5	17	17	17	17
3	23	23	23	23	-2	-2
4	-2	-2	-2	-2	23	4
5	4	4	4	4	4	23

swap swap swap

Smallest element moves up*



*assuming it wasn't already at the front of the list

Largest element moves down *

0	3	3	3	3	3	3
1	17	17	-5	-5	-5	-5
2	-5	-5	17	17	17	-2
3	23	23	23	23	-2	17
4	-2	-2	-2	-2	23	4
5	4	4	4	4	4	23

*assuming it wasn't already at the end of the list

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: Could be anywhere except position N-1.

Bubble Sort Algorithm

```
for i  = 0 to N − 1 {
    for k = 0 to  N − 2 − i  {
        if ( list[k] > list[k+1] ) {
            list.swap( k,  k+1 )
        }
    }
}
```

Before pass i, the largest i elements must already be in their correct position at the end of the list.

Thus, the inner loop can get shorter each time.

Bubble Sort Algorithm

```
for i = 0 to N − 2 {
    for k = 0 to N − 2 − i {
        if ( list[k] > list[k+1] ) {
            list.swap( k, k+1 )
        }
    }
}
```

The outer loop only needs to run $N - 1$ times.

(If the largest $N - 1$ elements are in their correct position, then the smallest element must also be in it correct position.)

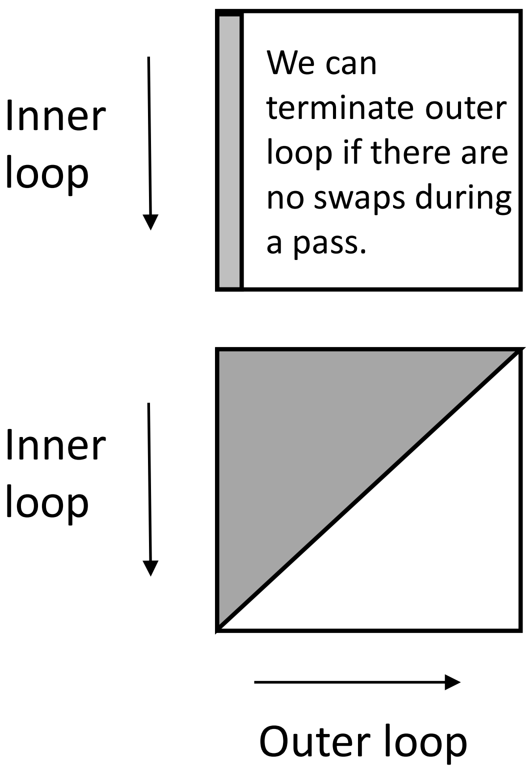
Bubble Sort Algorithm

// You don't always need to make $N - 1$ passes in outer loop.

```
for i  = 0 to N − 2  {
    swapped = false
    for k = 0 to N − 2  −  i  {
        if ( list[k] > list[k+1] ) {
             list.swap(  k, k+1 )
             swapped = true
        }
    }
   if   !(swapped)
        break    // return
}
```

Time Complexity ?

Bubblesort



Best case

Worst case

Gray regions in the square indicate the indicies examined through each pass through the inner loop.

COMP 250

Lecture 12

Algorithms for Sorting a List:

bubble sort
selection sort
insertion sort

Feb. 2, 2022

Selection Sort

Partition the list into two parts:

- the first part contains the smallest elements and is sorted
- the second part contains "the rest" of the elements (not necessarily sorted)

The sorted part is initially empty.

Repeat $N - 1$ times {

- find the smallest element in "the rest"
- swap that element with the first element in "the rest",
- this expands the first part of the list by 1

Example

sorted part is empty

0	3
1	17
2	-5
3	-2
4	23
5	4

rest

Example

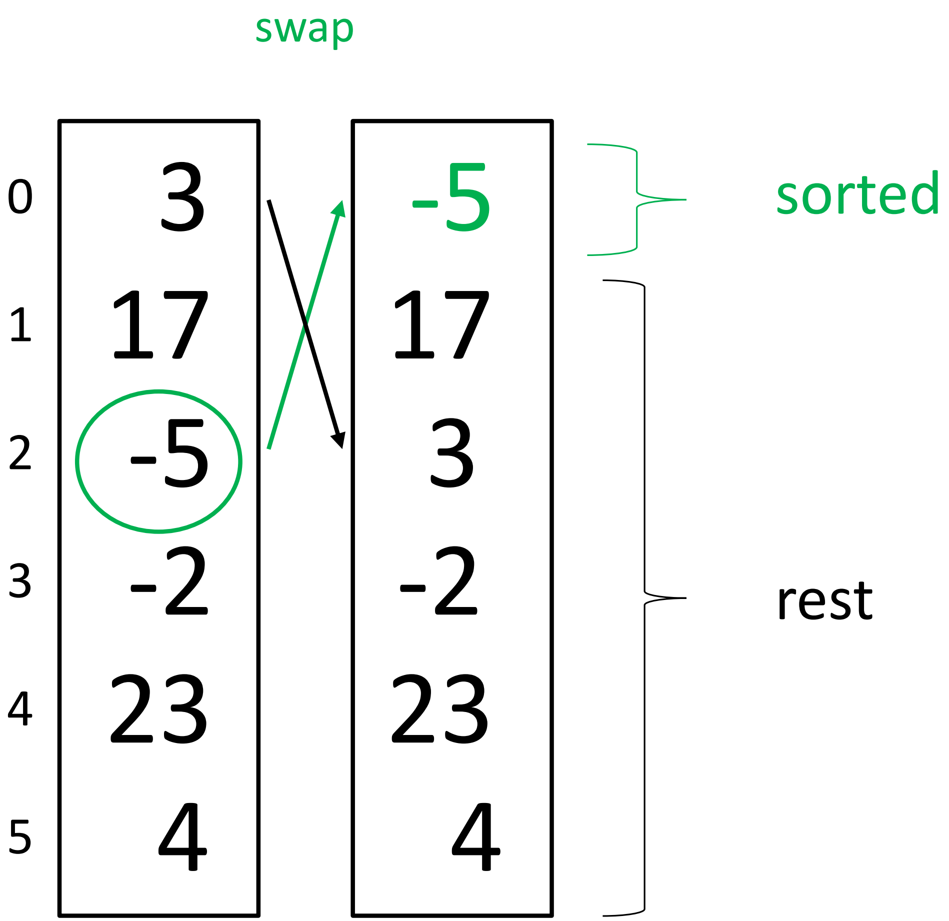
sorted part is empty

0	3
1	17
2	-5
3	-2
4	23
5	4

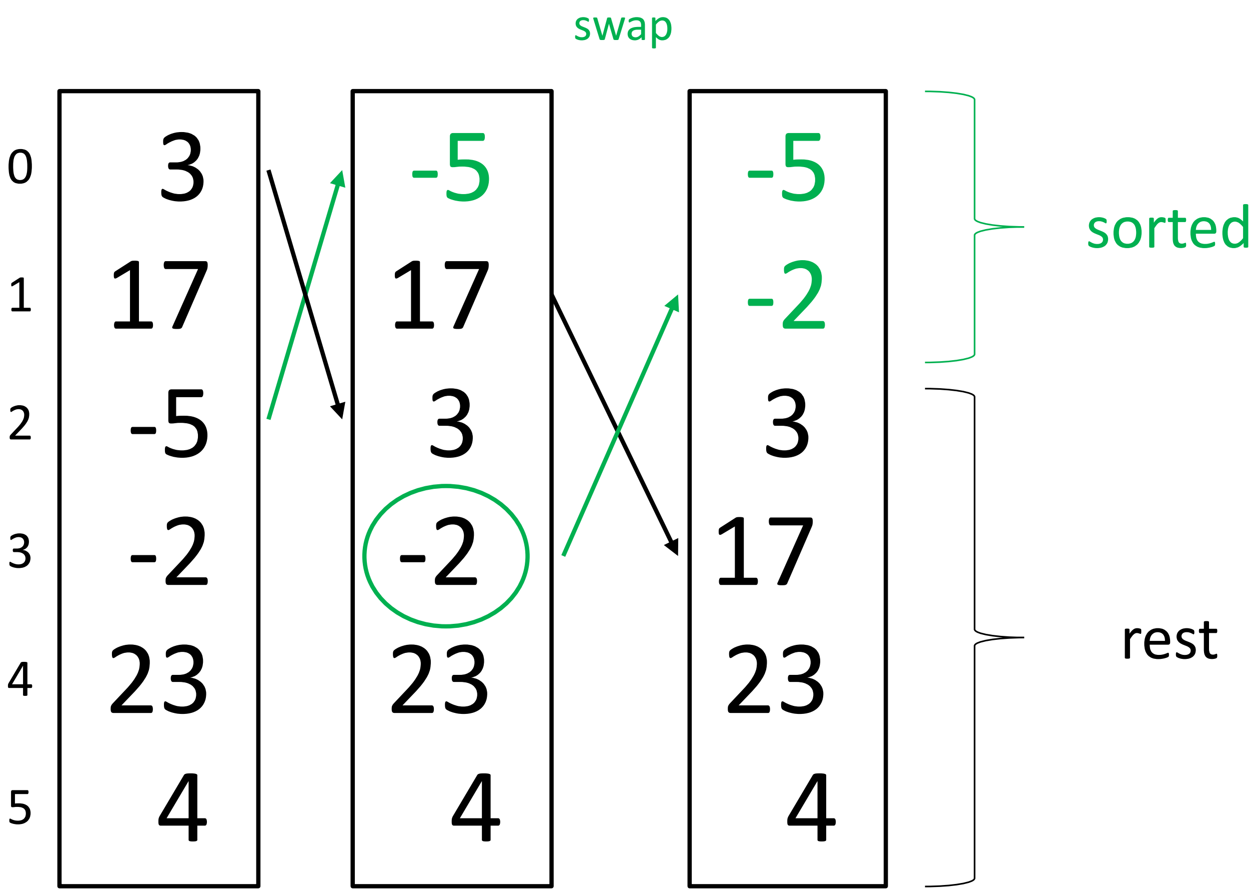
minimum element in “rest”

rest

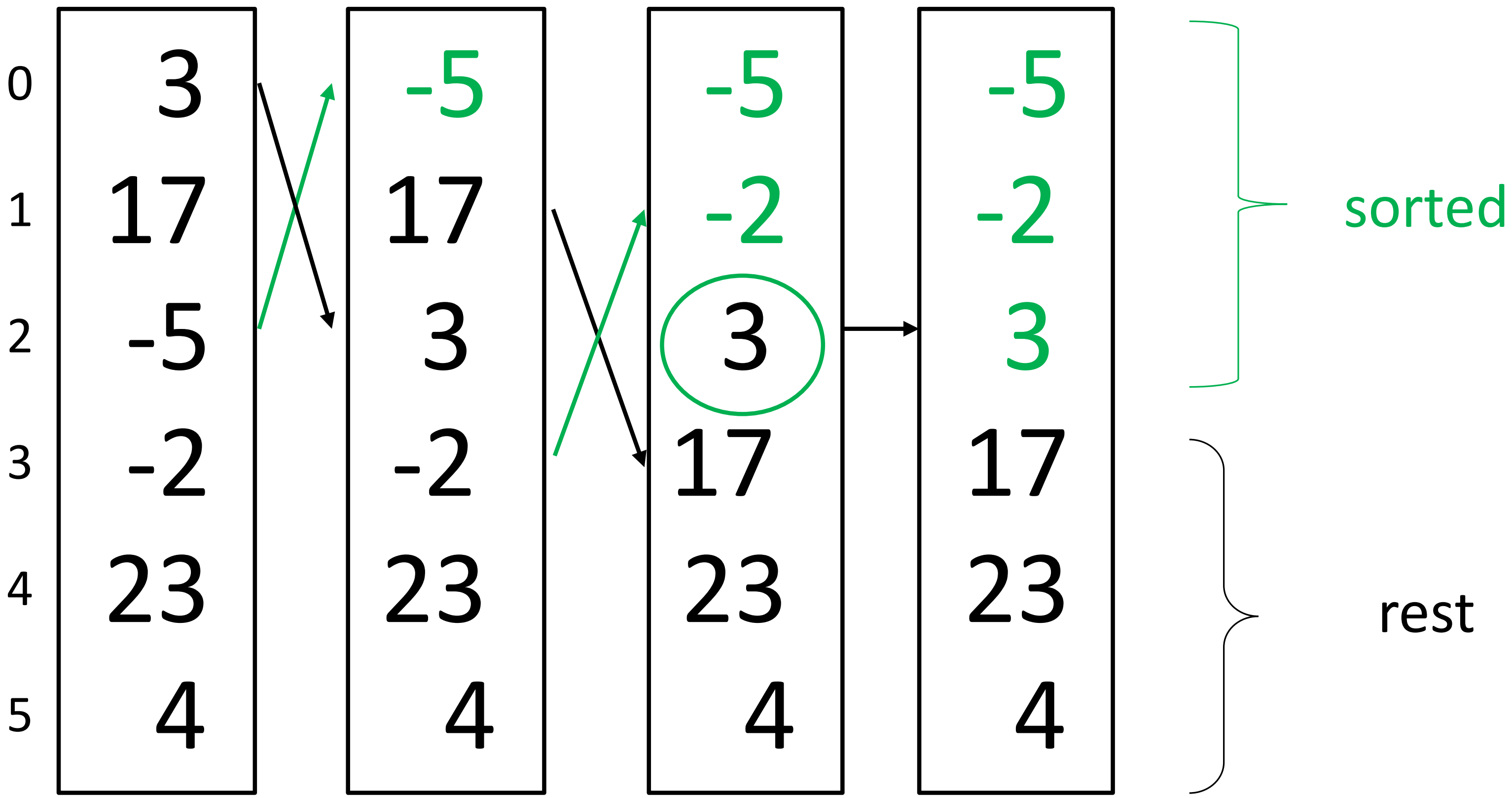
Example



Example

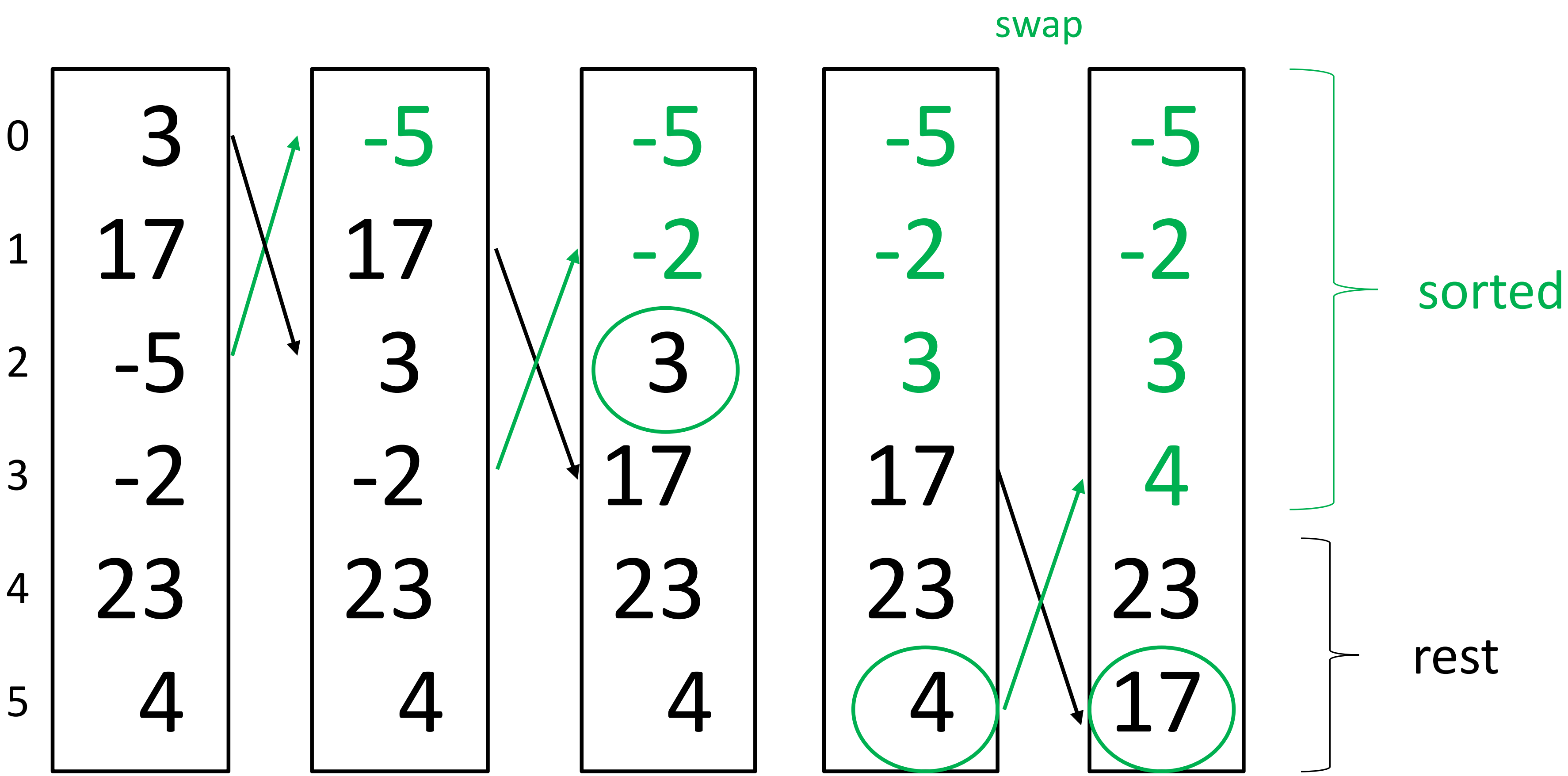


Example

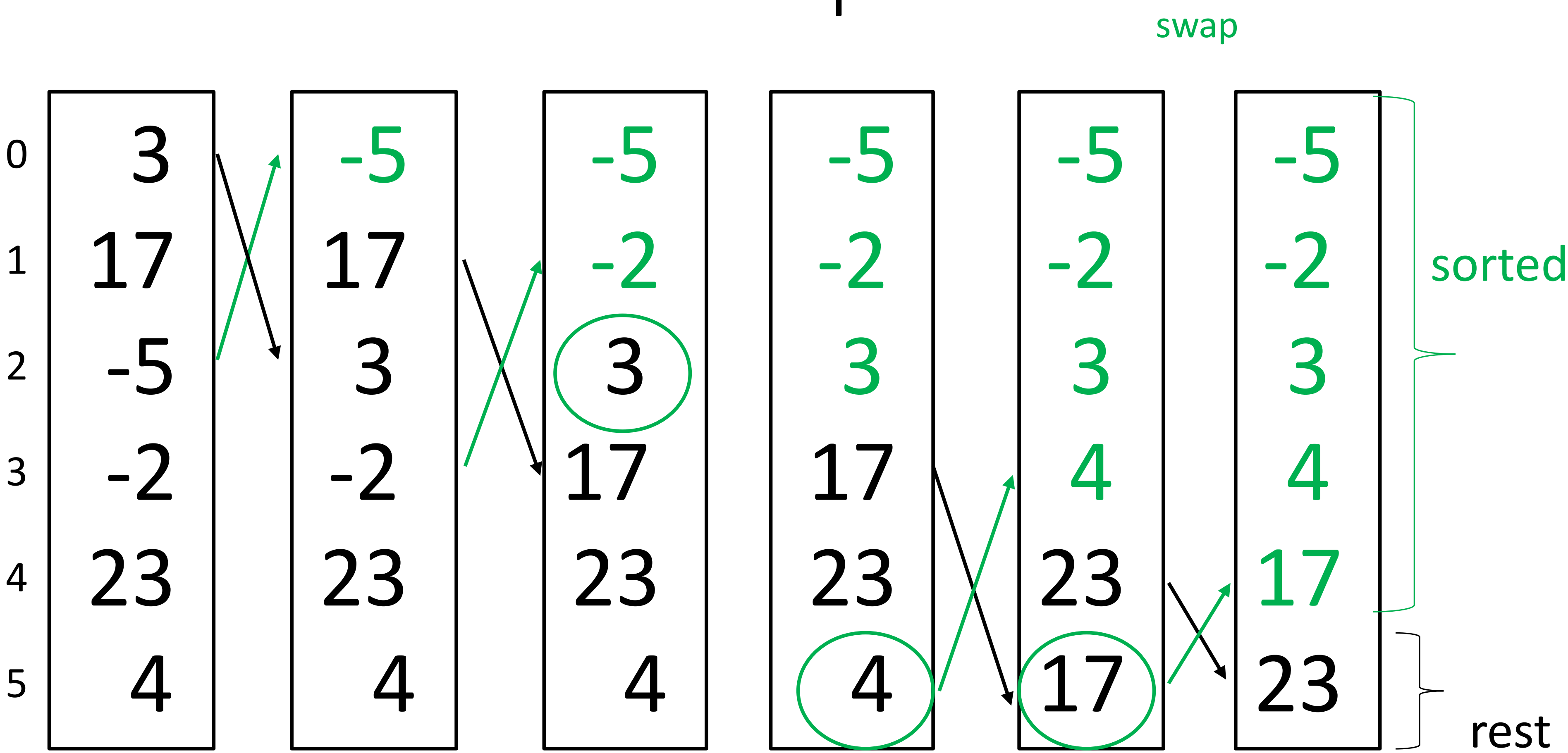


3 is the minimum
element already

Example


swap
0
1
2
3
4
5
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
sorted
rest

Example



swap
0
1
2
3
4
5
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

for $i = 0$ to $N - 2$ {	repeat $N - 1$ times
index = i minValue = list[i]	Take the first element in the rest and let it be the temporary min value.
for k = $i + 1$ to $N - 1$ {	For each other element in rest,

if (list[k] < minValue){
minValue = list[k]
index = k
}
}
	if element is smaller than the min value, then it will becomes the new min value. So remember its index.

if (i != index)
list.swap(i, index)
}
	Swap (if it is necessary)

Selection Sort

```
for i = 0 to N − 2 {

    index = i
    minValue = list[ i ]

    for k = i + 1  to N − 1  {

        if ( list[k] < minValue ){
            minValue = list[k]
            index = k
        }
    }
    if ( i != index )
        list.swap(i,  index )
}
```

This is the bottleneck (the inner loop).

Selection Sort

for $i = 0$ to $N-2$

for k = $i + 1$ to $N - 1$

{ .... }

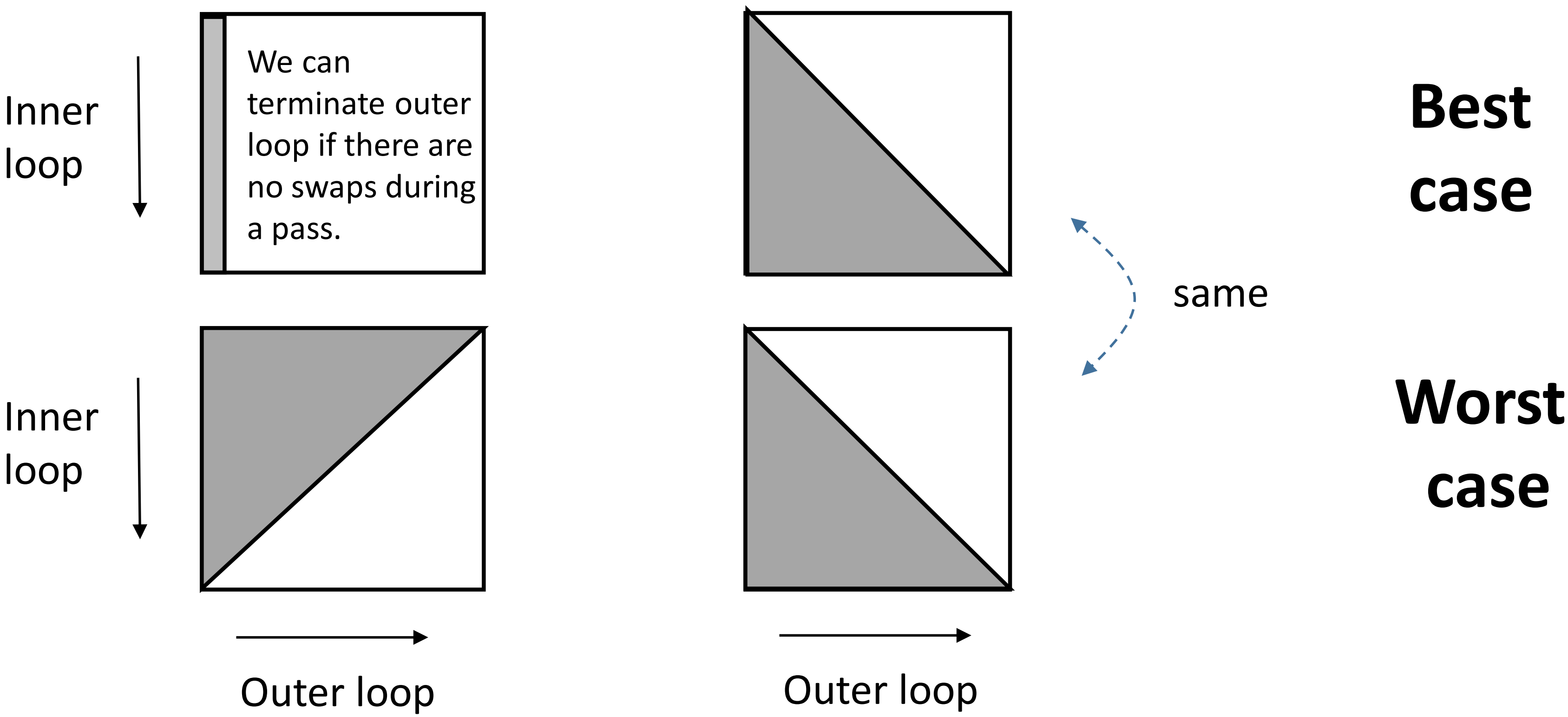
Q: how many times does { .... } get executed?

A: $N-1 \;+ N-2 \;+ N-3 \;+ \;\ldots\; + 2 + 1$

$i = 0$ $\quad i = 1$ $\quad i = 2$ $\quad i = N-2$

= $N\,(N-1\,)\,/\,2$

Bubblesort | Selection sort



Insertion Sort

for i = 1 to $N - 1$ {

Insert element list[i] into its correct position with respect to the list elements at indices 0 to $i - 1$

(At the start of pass i, the elements at indices 0 to $i - 1$ are sorted *only amongst themselves*. This is a weaker condition than in selection sort.)

}

Initial list

0	3
1	17
2	-5
3	-2
4	23
5	4

Initial list

	Initial list	$i = 1$	$i = 2$	$i = 3$
0	3	3	-5	-5
1	17	17	3	3
2	-5	-5	17	17
3	-2	-2	-2	-2
4	23	23	23	23
5	4	4	4	4

(**At the start of pass 3**, the elements at indices 0 to 2 are sorted *amongst themselves*.

Insert element i into its correct position with respect to 0 to i-1

	Initial list	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	3	3	-5	-5	-5
1	17	17	17	3	-2
2	-5	-5	3	17	3
3	-2	-2	-2	-2	17
4	23	23	23	23	23
5	4	4	4	4	4

(**At the start of pass 4**, the elements at indices 0 to 3 are sorted *amongst themselves*.)

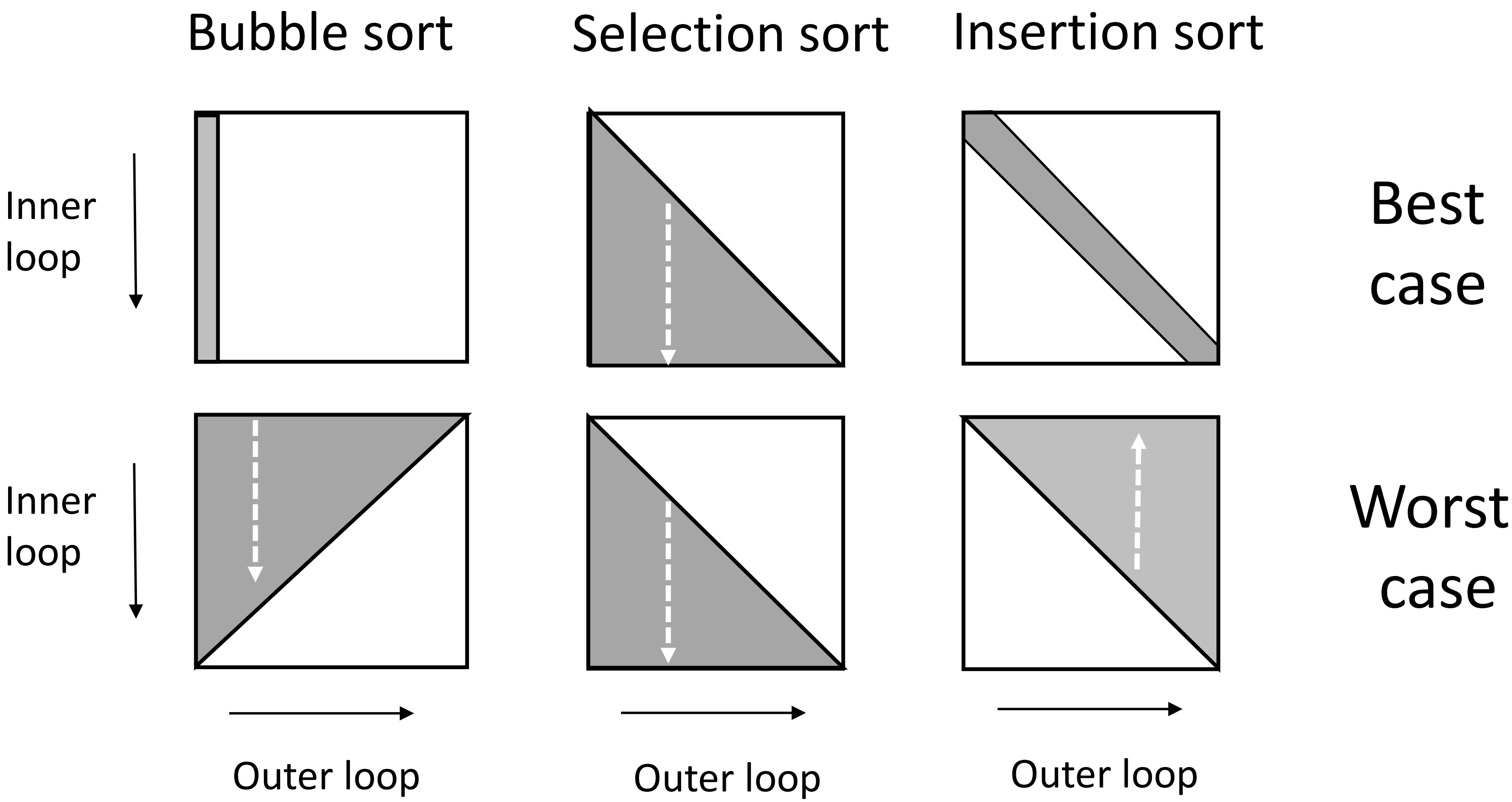
Mechanism is similar to inserting (adding) an element to an array list:

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

Insertion Sort

```
for i = 1 to N − 1{        //  index of element to move
    e =  list[ i ]               //  store as tmp
    k = i
    while (k > 0) and (e  <  list[ k - 1]){
        list[k] =  list[k - 1]              //   move it forward
        k  = k -1
    }
    list[k] = e
}
```

Time Complexity



Best case(s) : bubble and insertion sort are O(N), selection sort is O(N^2).
Worst case : each of the three algorithms is O(N^2).

Sorting Algorithms

- Bubble sort
- Selection sort
- Insertion sort

today $O(N^2)$

- Mergesort
- Heapsort
- Quicksort

lectures 22, 28 $O(N \log N)$

Hector Tutorial TODAY on Zoom at 6 pm