COMP20005 Workshop Week 8

LAB	Discussion: Sorting & loop invariant Ex 7.2, 7.6, 7.3 • Assignment: Q&A • **Work on Assignment**, AND/OR (if ass1 totally done): • implement array exercises *Important Events*: • **Assignment 1**: due this Friday, at **6pm** [note: GradeScope submission might be over-loaded on Friday afternoon]

A Task: Sorting an array

Input: an array such as `A[]= {5,2,3,2,1}`

The task: re-arrange elements of an array in a particular order.

Example: for an array of n numbers (int, float, dounble)

- order 1: increasing order (or non-decreasing), making

  `A[]= {1,2,2,3,5}`

- order 2: decreasing order (or non-increasing), making

  `A[]= {5,3,2,2,1}`

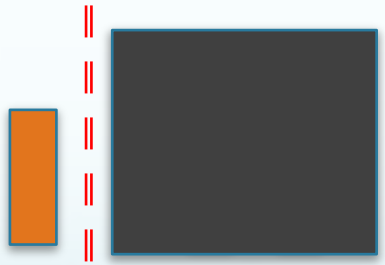
Insertion Sort: understanding

In this algorithm, we explore the input array element-by-element, and
We *keep* ***all explored elements*** *in sorted order*.

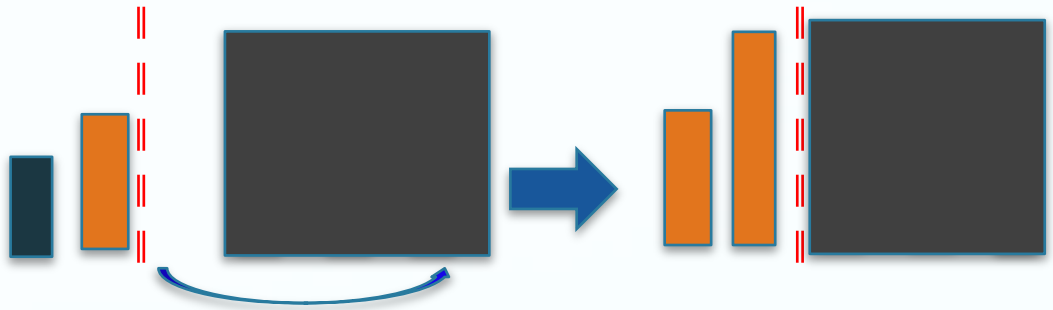
At first we note that:

- Nothing needs to do when examining `A[0]` because `A[0..0]` is sorted

Next step= ?



Insertion Sort: understanding



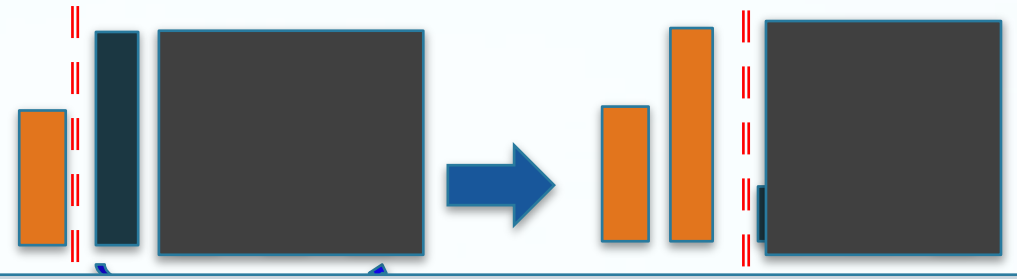
Then: when processing element `A[1]`

- Think of **the left** of `A[1]` as an array of 1 element, with index from `0` to `0`, denote it as `A[0..0]`
- Note that that left array `A[0..0]` is sorted
- Now, insert `A[1]` to the left and hence expand the left to `A[0..1]` insert so that the new left array `A[0..1]` is sorted (by swapping if needed)

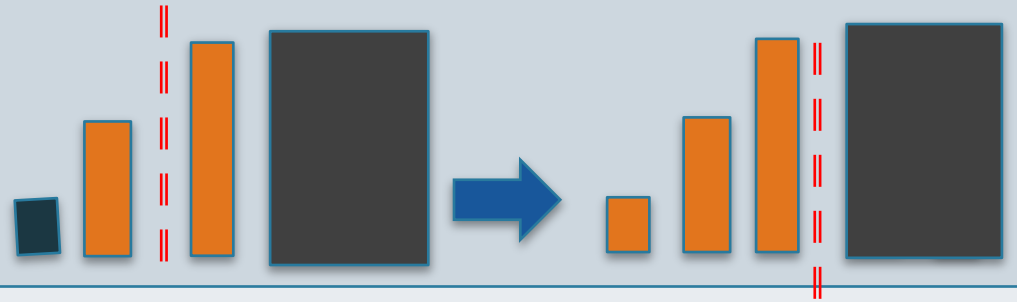
Next step= ?

Insertion Sort: understanding (n=5)

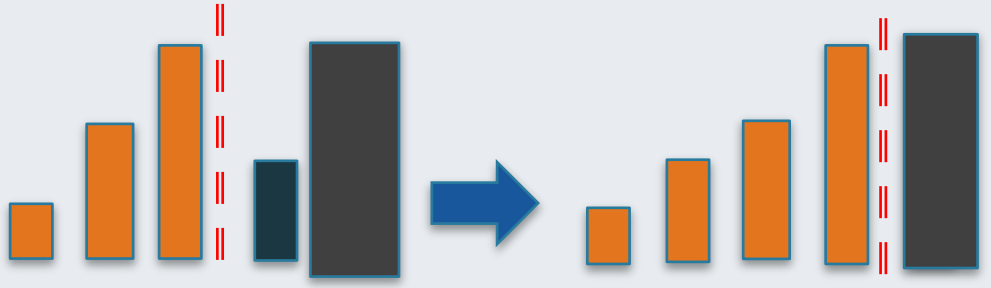
Round 1: process A[1]
- A[0..0] *is sorted*
- *insert* A[1] *to the left so that* A[0..1] *is sorted*



Round 2: process A[2]
- A[0..1] *is sorted*
- *insert* A[2] *to the left so that* A[0..2] *is sorted*



Round i:
insert A[i] to the left so that A[0..i] is sorted



Round n–1: process A[n–1]
- A[0..n–2] *is sorted*
- *insert* A[n] *to the left so that* A[0..n–1] *is sorted*

Insertion Sort: understanding (n=5)

Round 1: process A[1]
- A[0..0] *is sorted*
- *insert* A[1] *to the left so that* A[0..1] *is sorted*

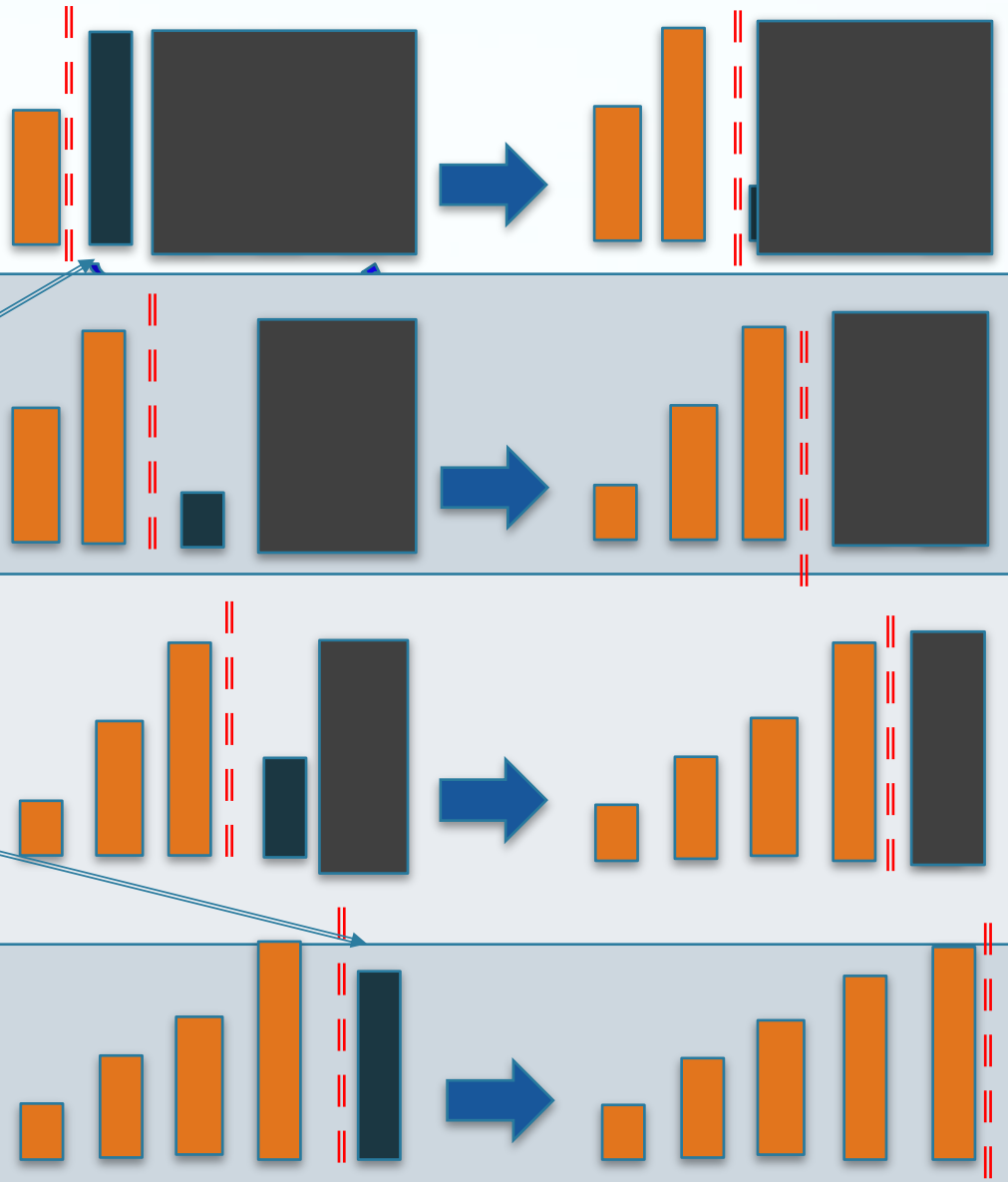
Round 2: process
- A[0..1] *is*
- *insert* A[2]
  A[0..2]

Round i:
insert A[i]
A[0..i] is s

So for the rounds we make a loop:

```
for (i=1;i<n; i++)
```
last value `i= n-1`

Round n-1: process A[n-1]
- A[0..n-2] *is sorted*
- *insert* A[n] *to the left so that* A[0..n-1] *is sorted*



Insertion Sort & loop invariant

`for (i=1;i<n; i++)`

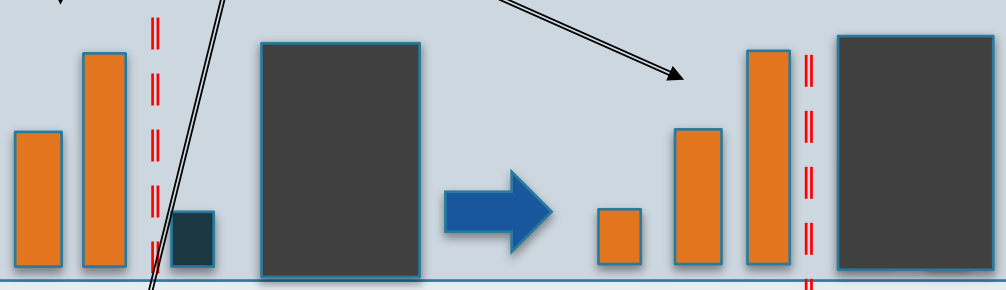
At the start of iteration `i`:

`A[0..i-1]` is sorted

during the iteration, we make sure that that will be true at the start of the next iteration

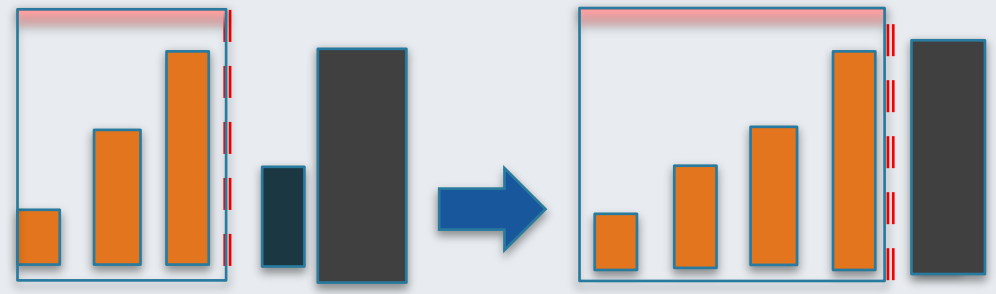
Round `i=3`: process `A[i]`

- ***invariant***: `A[0..i-1]` *is sorted*
- *insert* `A[i]` *to the left so that* `A[0..i]` *is sorted*



Note:

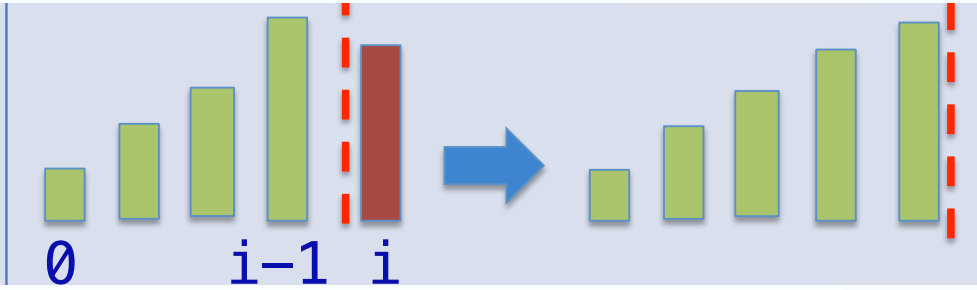
- *loop invariant is also true at the start and end of the loop*
- *understanding loop invariant helps us to write robust loops*



Insertion Sort: Algorithm

Process `A[i]` in round `i`

Round i: consider `A[i]`
- `A[0..i-1]` *is sorted*
- *insert* `A[i]` *to the left so that* `A[0..i]` *is sorted*



How to: insert `A[i]` to the left?
- another loop or single operations?

```
void ins_sort(int A[], int n) {
  int i,j;
  for (i=1; i<n; i++) {
    // loop invariant: A[0..i-1] is sorted
    // now insert A[i] to A[0..i-1] to ensure the loop variant for the next i
    ??
  }
}
```

Ex 7.2

Modify so that the array of values is sorted into decreasing order.

```
    // insertion sort:  sorting elements A[0] to A[n-1] min ascending order

1   for (i= 1; i<n ; i++) {
2     /* swap A[i] left into correct position */
3     for (j= i-1; j>=0 && A[j+1]<A[j]; j--) {
4        /* not there yet */
5        int_swap( &A[j] , &A[j+1] );
6     }
7   }
```

Selection Sort: Ex 7.6

An alternative sorting algorithm is selection sort. It goes like this: scan the array to determine the location of the largest element, and swap it into the last position. Then repeat the process, concentrating at each stage on the elements that have not yet been swapped into their final position.
Write a function to sort an integer array using selection sort.

Selection Sort: understanding

Selection sort: scan the array to determine the location of the largest element, and swap it into the last position. Then repeat the process …

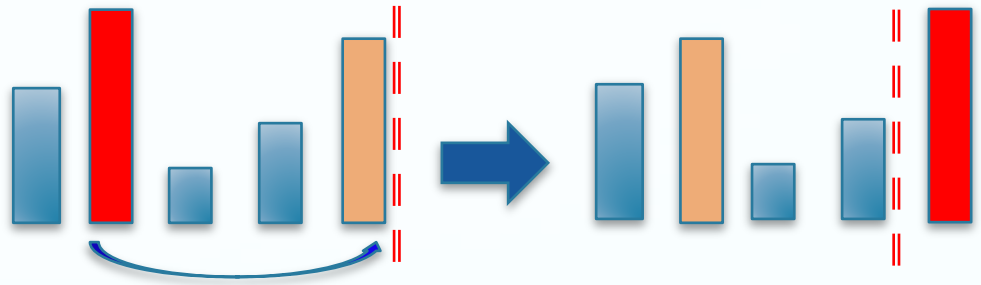
So at first:
scan all un-sorted elements from position `0` *to position* `n-1` *to determine the position of the largest element,*
swap it into the last position, ie. position `n-1`.

What next?

Ex 7.6: Write A Function

?: consider `A[0..n-1]`

- *determine position of the largest*
- *swap with the last, ie.* `A[n-1]`, *then* `...`



```
// sort array A in ascending order using selection sort
void sel_sort(int A[], int n) {

}
```

Do 7.03, finish 7.02, 7.06 using grok

7.03: Modify the insertion sort program so that after the array has been sorted, only the distinct values are retained in the array with variable n suitably reduced.

Assignment 1

Marking Rubric: The importance of Style & Structure

Stage	Presentation	Structure	Execution	max accumulated mark
1	+5	+2	+1	8
2		+4	+4	16
3		+0	+4	20
all	5	6	9	

Failed code could even get 11

Absolutely correct program could get only 9

Correct and good Stage 1+2 alone could get 16

Notes: most of the rubrics have been explained in last week's slides

ASS1 Marking Rubric: a few keys in Presentation [Q&A next week]

use of magic numbers, -0.5;
#defines not in upper case, -0.5;
unhelpful `#define`, -0.5
bad choices for variable names, -0.5;
bad choice for function names, -0.5;

use `#define` for meaningful constants
`#define`-ed names should be in upper case
variable names in lower case (except single-letter array name)
variable names should be expressive

absence of function prototypes, -0.5;

add function prototypes before main()
main() should be the first function body!

inconsistent bracket placement, -0.5;
inconsistent indentation, -0.5;

you can use sample programs in lectures as the model

excessive commenting, -0.5;
insufficient commenting, -0.5;

each function header should have a comment
add comments for non-trivial code segment

lack of whitespace (visual appeal), -0.5;
lines >80 chars, -0.5;
use of constant subscripts in 2d arrays, -1.0;
other issue: minor -0.5, major -1.0
empty program, or helloworld, or etc, -5.0;

- comment at end that says ”programming is fun", +0.5;
- overall care and presentation, +0.5;

DO IT !
Easy way to get back 0.5 or 1 mark

Assignment 1: Some mistakes in Presentation

lack of whitespace (visual appeal), -0.5;

```
some=some*any+1-other;

for (i=0;i<n;i++) {
   if (a+b>c) {
```

✗

```
some= some * any + 1 - other;

for (i= 0; i<n ; i++) {
   if ( a + b > c ) {
```

✓

Assignment 1: Some mistakes in Presentation

lines >80 chars, −0.5;

80-char indicator

```
   printf("hvgvhgv hjjh %d %d fggf %d %d xyuyuyu hghghg hfghghg hghg gfh
%d\n", a, b, c, d, e);

   printf("hvgvhgv hjjh %d %d fggf %d %d xyuyuyu hghghg hfghghg hghg gfh
%d\n",
          a, b, c, d, e);
```

✗

```
   printf("hvgvhgv hjjh %d %d fggf %d %d xyuyuyu hghghg"
          " hfghghg hghg gfh %d\n",
          a, b, c, d, e);
```

✓

Assignment 1: Some mistakes in Presentation

use of constant subscripts in 2d arrays, –1.0;

When the value of row index, column index, or both are constants, we should use named constants, or give enough explanation.

I used a 2D array double marks[CAP][2] to store the marks of A1 and A2 of students in this workshop in the following way:

```
i= 0;
while ( i<CAP && scanf("%lf%lf", &marks[i][0], &marks[i][1]) != 2) {
   i++;
}
```

✗

```
#define A1_IDX 0
#define A2_IDX 1

i= 0;
while ( i<CAP && scanf("%lf%lf", &marks[i][A1_IDX], &marks[i][A2_IDX]) != 2) {
   i++;
}
```

✓

ASS1 Marking Rubric: a few keys in Structure

global variables, -1.0;

Global variables are NOT allowed!

main program too long or too complex, -1.0;
functions too long or too complex, -0.5;
overly complex function argument lists, -0.5;

function should not be long
and should not have too many arguments

insufficient use of functions, -0.5;
duplicate code segments, -0.5;

when having a few line, or a complicated line similarly-duplicated, think about creating a new function!

overly complex algorithmic approach, -0.5;

avoid too many levels of nesting `if and` loops
don't make the markers wonder too much to understand your code!

unnecessary duplication/copying of data, -0.5;

For example, think carefully before you copy an array!

other structural issue: minor -0.5, major -1.0;

Assignment 1: Some mistakes in Structure

duplicate code segments, −0.5;

When? A block of few lines (or only 1 long/complicated line) is structurally repeated.

Example 1:

```
printf("On April 15, I joined a group of 7 friends\n");
...
printf("On April 20, I joined a group of 12 friends\n");
```

Good:

```
print_group(15, 7);
...
print_group(20, 12);
```

```
void print_group(int date, int group_size) {
   printf("On April %d, ... of %d friends\n",
          date, group_size);
}
```

Assignment 1: Some mistakes in Structure

duplicate code segments, −0.5;

When? A block of few lines (or only 1 long/complicated lines) is structurally repeated.

Example 2: different code segments for computing per-class score (stage 1) and per-lecturer score (stage 2)

Example 3:

```
// searching for 15 in A[]
index_15= NOTFOUND;
for (i=0; i<n; i++) {
   if (A[i]==15) {
      index_15= i;
      break;
   }
}
...
// searching for 20 in B[]
index_20= NOTFOUND;
for (i=0; i<m; i++) {
   if (B[i]==20) {
      index_20= i;
      break;
   }
}
```

```
index_15= array_search(15, A, n);
...
index_20= array_search(20, B, m);
```

```
// returns first i that A[i]==x
// or returns NOTFOUND if x not in A
int array_search(int x, int A[], int n) {
   int i;
   for (i=0; i<n; i++) {
      if (A[i]==x) {
         return i;
      }
   }
   return NOTFOUND;
}
```

Assignment 1: Some mistakes in Structure

duplicate code segments, –0.5;

```
// number of festivals in seasons
fest_days[5]= {0,20,5,7,3};
// where index 1 for Spring,
//             2 for Summer, ...

...
for (i=1; i<=4; i++) {
   // i is season, i=1 for Spring
   if (i==1) printf("Spring");
   if (i==2) printf("Summer");
   if (i==3) printf("Autumn");
   if (i==4) printf("Winter");
   printf(" has %d festivals\n",
          fest_days[i]);
}
```

o

```
#define SPRING 1
#define WINTER 4

// number of festivals in seasons
fest_days[5]= {0,20,5,7,3};
// names of seasons
season_names[5][7]={"", "Spring", "Summer",
                    "Autumn", "Winter"};
// where index 1 for Spring,
//             2 for Summer, ...



...

for (i=SPRING; i<=WINTER; i++) {
   // i represents season
   printf("%s has %d festivals\n",
          season_names[i], fest_days[i]);
}
```

Assignment 1: Some mistakes in Structure

overly complex algorithmic approach, –0.5;

When?

- There are chances that a markers spend a reasonable time to read, but cannot understand one or some of your code segments!
- You have too deeply nested control structures (loop, if...)

```
for ( ... ) {
   while ( ... ) {
      if ( ... ) {
         for ( ... ) {
            for ( ... ) {
               if ( ... ) {
                  ...
                  ...
```

➔ consider to form some inner-most parts into a function, or

➔ perhaps to re-develop the algorithm

A1 Q&A, LAB

Assignment 1 : Questions?

make sure to test your code and

read the *verification report* carefully, make sure:

- NO compilation message,
- NO differences with expected output

Know the final submit time in LMS!

Not even finished stage 1?

- ***Learn from solution for 2020.***

Finished or almost finished stage 1?

- ***Check the marking rubric before continuing.***
- ***Rule-following awarded more marks than correctness!***

Programming is fun: Check each of the following exercises, make sure you know how to do it, and implement if not yet done and time permitted.

ARRAYS: *Arrays, and array manipulation, are very important! 15 exercises in C07, for examples:*

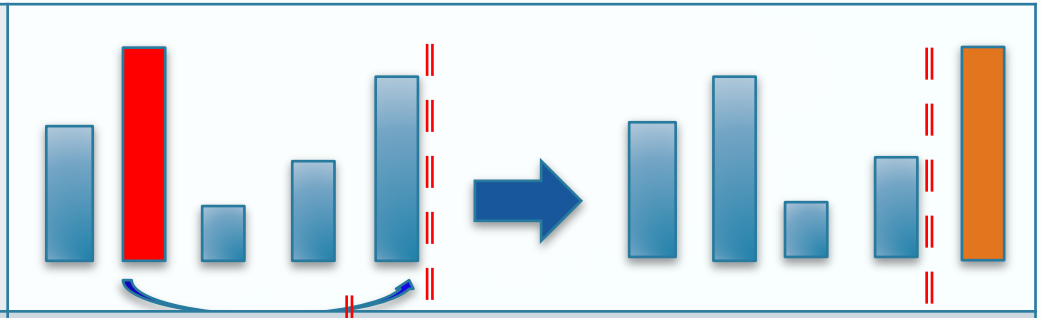
- Sorting-related exercises: 7.5, 7.4 , 7.2, 7.3, 7.6

Additional Materials for self-study

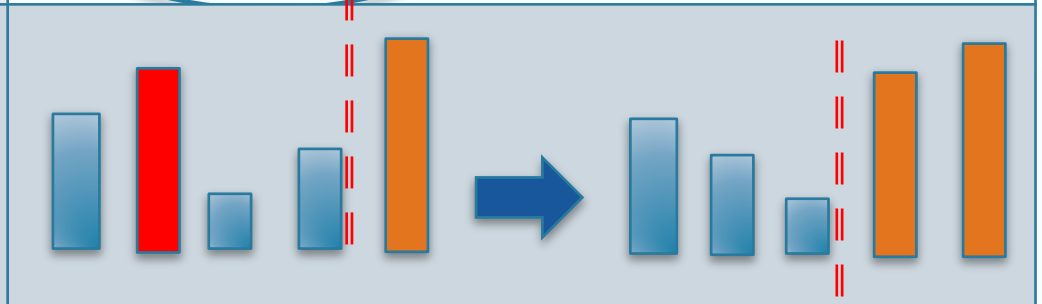
See next pages

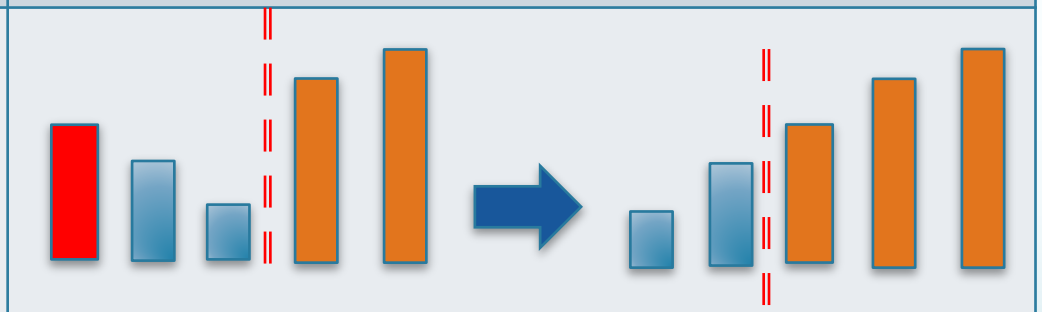
Selection Sort: understanding (n=5)

Round 1: consider `A[0..4]`
- *determine position of the largest*
- *swap with the last, ie.* `A[4]`

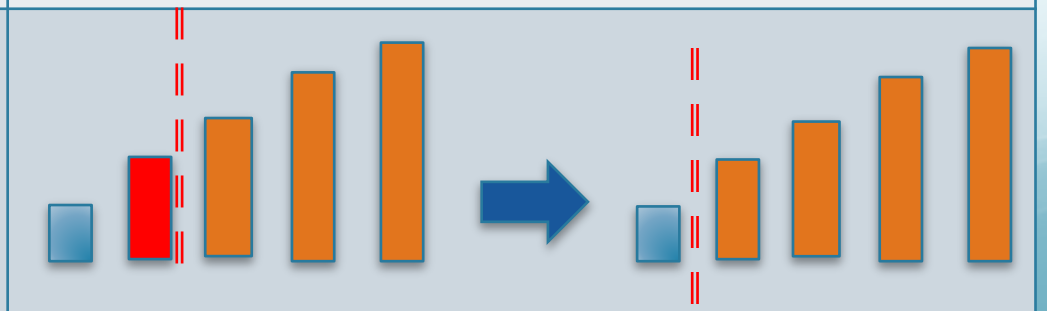


Round 2: consider `A[0..3]`
- *determine position of the largest*
- *swap with the last, ie.* `A[3]`





Round 4: consider `A[0..1]`
- *determine position of the largest*
- *swap with the last, ie.* `A[1]`

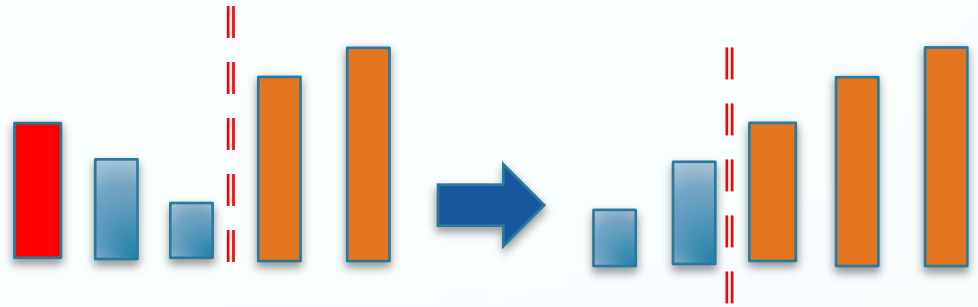


Ex 7.6: Write non-recursive function

First round: examining `A[0..n-1]`, last round: examining `A[0..1]`

Round ?: consider `A[0..i]`

- *determine position of the largest*
- *swap with the last, ie.* `A[i]`



```
void sel_sort(int A[], int n) {



}
```