COMP20005 Workshop Week 8

Preparation:

- open `grok`, `jEdit`, and `minGW` (or `Terminal` if yours is a `Mac`)
- download this slide set (`ws8.pdf`) from github.com/anhvir/c205 if you like

1	Discussion 1: String, Ex 7.12, 7.14
2	Discussion 2: Sorting Algorithms Insertion Sort (exercise 7.2, 7.3) Selection Sort (exercise 7.6)
3	Assignment: Q&A, marking rubric
LAB	• Work on Assignment, AND/OR • group/individual work, • AND/OR implement Ex 7.7, 7.15

Strings and <string.h>

A string is a pointer to char which points to a valid memory location. The string will then be the sequence of chars from that location until the first-seen *zero character* '\0'.

Examples

```
char *s2="1234";        // s2 points to 1 2 3 4 \0
```


```
char s1[10]= "Hello"; //
```


```
char *s= s1; // s1 is string and it can hold 9 chars as maximal
printf("string s= %s  s1= %s\n", s, s1);
```

Important functions:

```
strlen(s)          strcpy(s1, s2)     atoi(s2)
```

Strings: array and pointer notations

```
char *s="1234";
```

Array Notation	Pointer notation

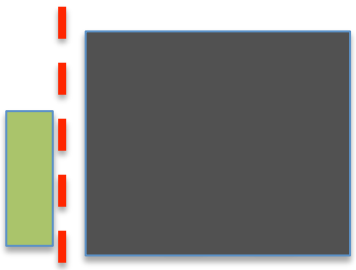
`int n= strlen(s);`
`int i;`
`for (i=0; i<n; i++) {`
`  // process character s[i]`
`  printf("processing %c\n", s[i]);`
`}`
	`char *p;`
	`for (p=s; *p != '\0'; p++) {`
	`  // process character *p`
	`  printf("processing %c\n", *p);`
	`}`

Strings: Exercise 7.12

Write a function `int is_palindrome(char*)` *that returns true if its argument string is a palindrome, that is, reads exactly the same forwards as well as backwards; and false if it is not a palindrome. For example,* `"rats live on no evil star"` *is a palindrome according to this definition, while* `"A man, a plan, a canal, Panama!"` *is not. (But note that the second one is a palindrome according to a broader definition that allows for case, whitespace characters, and punctuation characters to vary.)*

```
int is_palindrome(char *s){



}
```

Insertion Sort: understanding



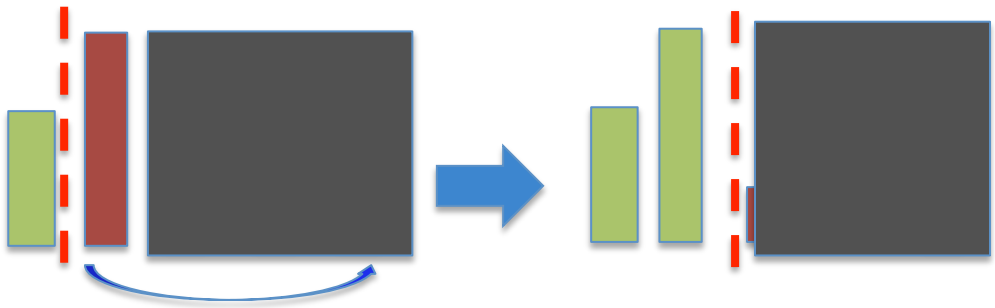
In this algorithm, we process element-by-element, *keeping all processed elements in sorted order*.

At first we note that:

- Nothing needs to do when processing at `A[0]`

Next step= ?

Insertion Sort: understanding



Then:

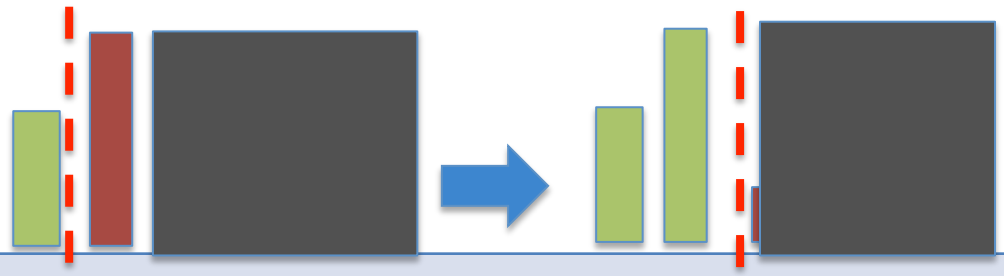
- Consider element `A[1]`
- Think of **the left** of `A[1]` as an array `A[0..0]`
- Note that that left array is sorted
- Now, insert `A[1]` to the left (and hence expand the left) so that the new left array `A[0..1]` is sorted

Next step= ?

Insertion Sort: understanding (n=5)

Round 1: consider `A[1]`

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



Round 2: consider `A[2]`

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

Round i: consider `A[4]`

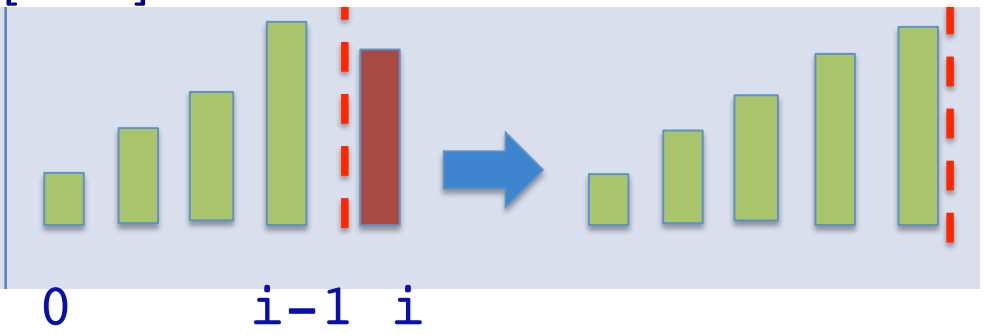
- `A[0..4]` *is sorted*
- *insert* `A[i]` *to the left so that* `A[0..4]` *is sorted*

Insertion Sort: Algorithm

Process `A[1]` in round `1`, ..., `A[n-1]` in round `n-1`

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?

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

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]
                       into non-decreasing order      */
    /* assume that A[0] to A[n-1] have valid values   */
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   }
    /* and that's all there is to it */
```

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

```
void selection_sort (int A[], int n)
```

or, equivalently

```
void selection_sort (int *A, int n)
```

that orders the `n` *elements in array* `A`.

Challenge: write the above function using recursion rather than iteration.

Ex 7.6: understanding

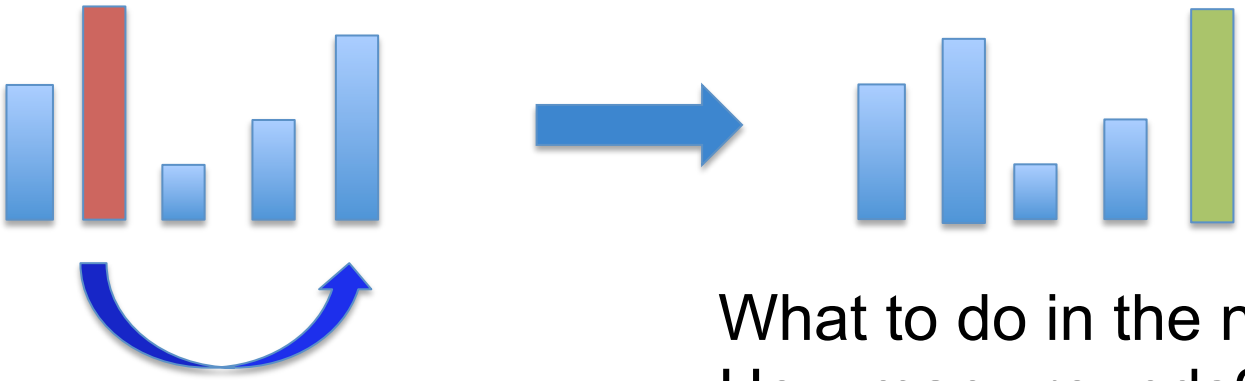
Selection sort: scan the array to determine the location of the largest element, and swap it into the last position. Then repeat the process ...

So in the first round:

- *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`.

After the first round:

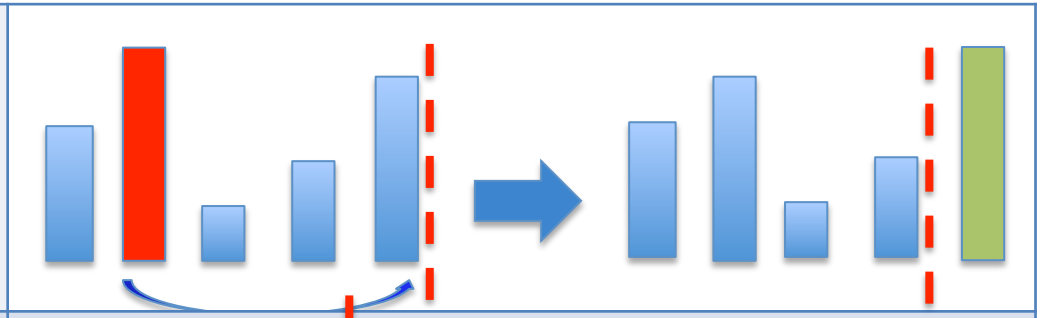
- *Job done for position A[n-1]*
- *The elements in array A[0..n-2] are still unsorted*



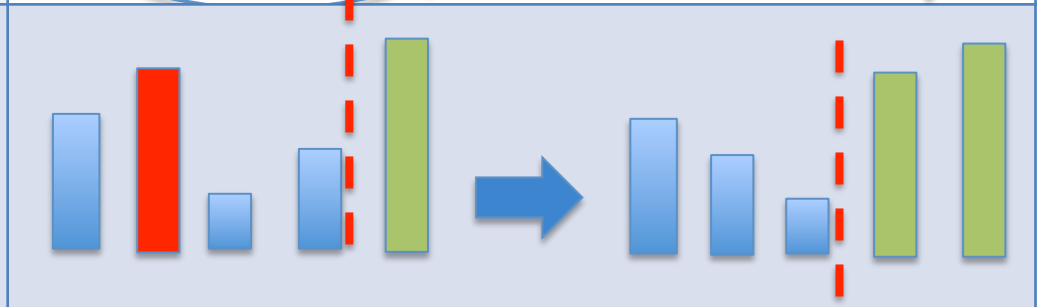
What to do in the next round?
How many rounds?

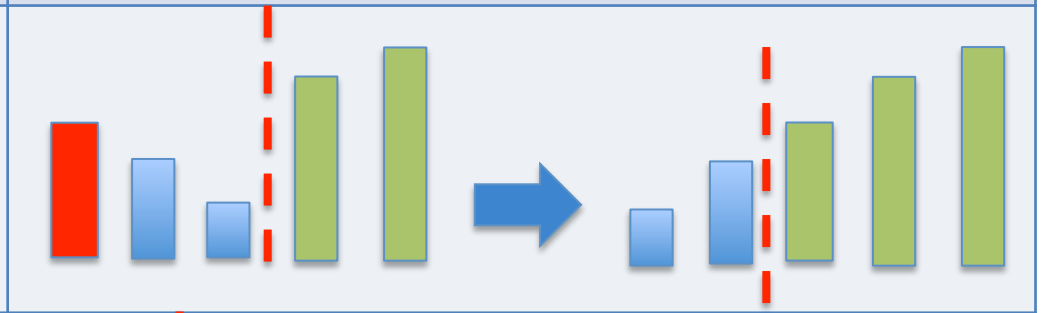
Ex 7.6: understanding (n=5)

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

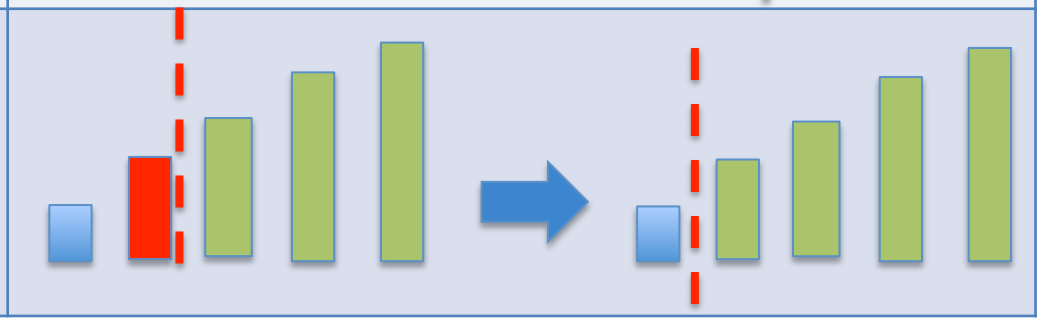


Round 2: consider `A[0..3]`
- *determine position of the largest*
- *swap with the last, ie.* `A[3]`





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

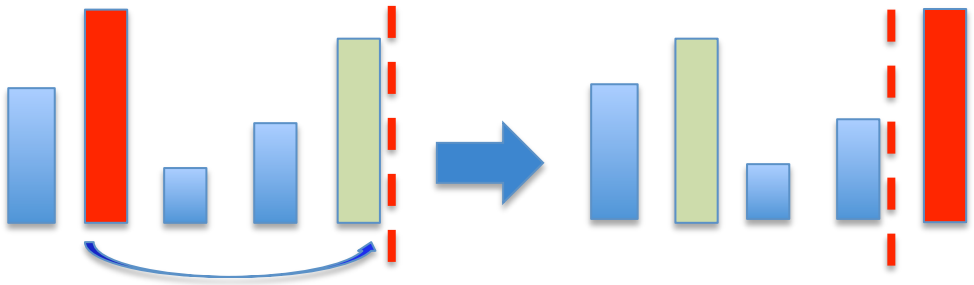


Ex 7.6: Algorithm

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]`

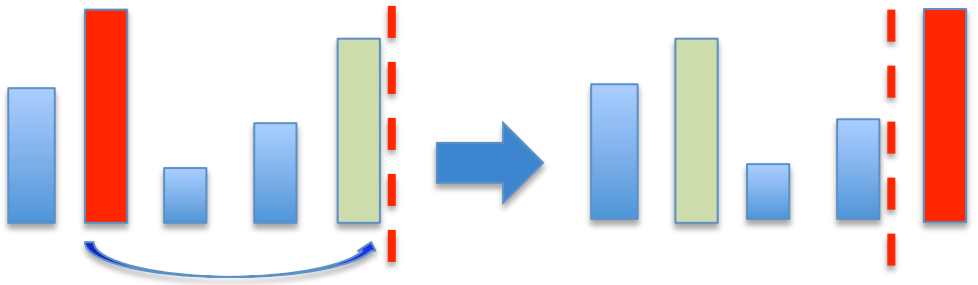


```
void sel_sort(int A[], int n) {



}
```

Ex 7.6: Recursive Algorithm

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

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



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



}
```

Ex 7.3

Write a program that sort an array in increasing order, then after the array has been sorted, only the distinct values are retained in the array (with variable n suitably reduced).

```
./program
Enter as many as 1000 values, ^D to end
1 8 15 3 17 12 4 8 4
9 values read into array
Before:    1   8  15   3  17  12   4   8   4
After :    1   3   4   8  12  15  17
```

Assignment 1: understanding

Understanding the requirements for

- Stage 1
- Stage 2
- Stage 3

TESTING IN YOUR COMPUTER

- *Using redirection when running/testing your program:*

```
./myass1 < wagon0.tsv > my_out0.txt
```

- *Your program's output must be the same as the expected, ie. the command*

```
diff my_out0.txt wagon0-out-dos.txt
```

must give empty output (that is, no difference).

- *Remember that your code might work well on the 2 supplied data sets, but fail on some other…*

SUBMITTING:

- *Wait for the verification report*
- *Read the verification report carefully. Your program might work perfectly in your computer but fail in the testing computer(s).*
- *Try to submib early to avoid unexpected technical problems.*

Assignment 1: marking rubric

use of magic numbers, -0.5; unhelpful #defines, -0.5; #defines not in upper case, -0.5; bad choice for function names, -0.5; bad choices for variable names, -0.5; absence of function prototypes, -0.5; inconsistent bracket placement, -0.5; inconsistent indentation, -0.5; lack of whitespace (visual appeal), -0.5; lines >80 chars, -0.5; excessive commenting, -0.5; insufficient commenting, -0.5; use of constant subscripts in 2d arrays, -1.0; ---------- comment at end of source code that says "programming is fun", +0.5; overall care and presentation, +0.5;	**total mark: up to 3**

Assignment 1: marking rubric

global variables, -0.5; main program too long or too complex, -0.5; other functions too long or too complex, -0.5; overly complex function argument lists, -0.5; insufficient use of functions, -0.5; duplicate code segments, -0.5; overly complex algorithmic approach, -0.5; unnecessary duplication/copying of data, -0.5; other structural issue (minor), -0.5; other structural issue (major), -1.0;	**total mark: up to 5**

Assignment 1: marking rubric

errors in compilation that prevent testing, -3.0; runtime segmentation fault on test1 with no output generated, -1.0; runtime segmentation fault on test2 with no output generated, -1.0; <various deduction for wrong output>	**total mark: up to 3**
----------	**----------**
no Authorship Declaration at top of program, -3.0; incomplete or unsigned Authorship Declaration at top of program, -2.0; significant overlap detected with another submission, -5.0; use of external code without attribution (minor), -1.0; use of external code without attribution (major), -5.0;	**deducted from the whole work**

LAB

Do your assignment, and/or:

- ARRAYS: Exercises 7.2, 7.3, 7.6, 7.7, 7.15
- STRINGS: Exercises 7.12, 7.14

Assignment 1: maximize your score

- Have a look at the 2015 sample solution.
- Test your program with Alistair's datasets, and also with a few special datasets of your own.
- Carefully compare your program's functionality with the specification.
- Carefully assess your program against the marking rubrics.