COMP10002 Workshop Week 5

Outlook:

1	Chapters 1-7 & Sample Tests: Questions/Problems
2	Arrays: traversal, searching
3	Discuss: 7.6, 7.7
4	Lab: Implement 7.8, 7.9, 7.10, 7.11

Questions on Chapters 1-7 and Sample Tests?

Functions with arrays as parameters

when defining function:

```
… my_function( int A[], int n) {
    . . .
}
```

Arrays: traversal of array A that has n elements

```
for (i=0; i<n; i++) {
    //visit a[i]
}
```

Example: write function that

- prints all elements of `A`
- returns sum of elements of `A`
- computes `C= A+B` where `A`, `B`, `C` are arrays of the same size

Arrays: finding min, max, …

```
min = A[0];          // use the 1st as candidate
for (i=1; i<n; i++) {
    if (A[i] is a better candidate) {
        min= A[i];
    }
}
```

Examples: For an float array, write function that

1. returns the minimum value
2. swap the largest element with the last element, if there are more than one largest only swap the first one

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)
```

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

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

7.6: understanding

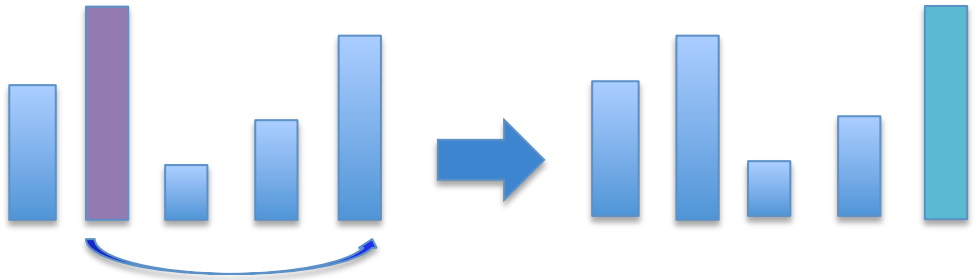
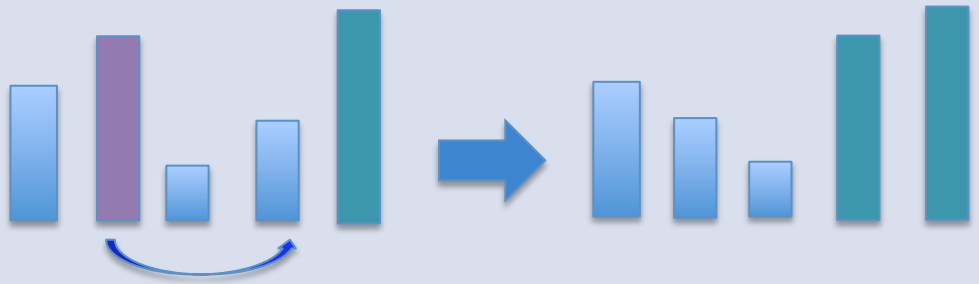
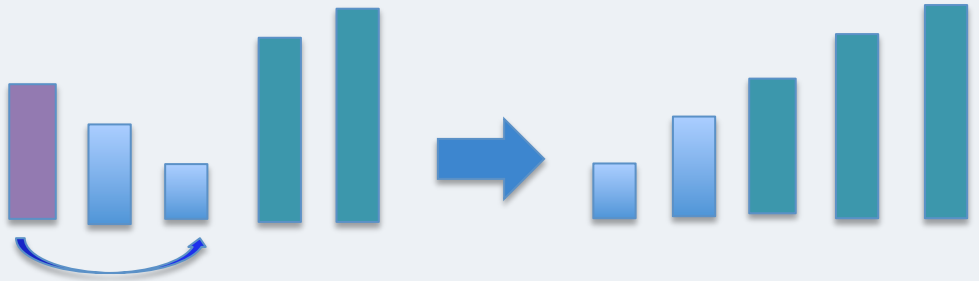
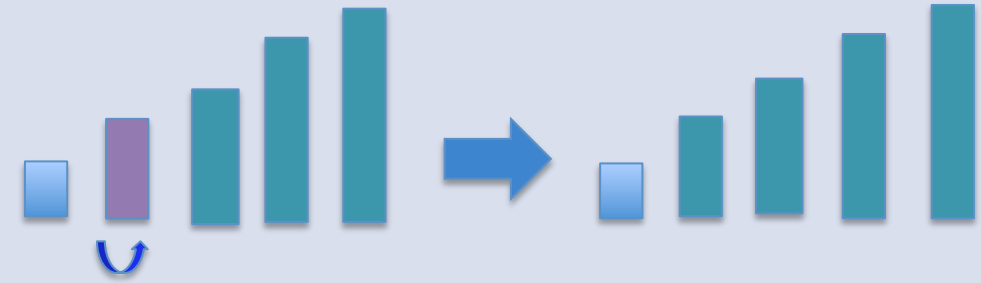
Selection sort: scan the array to determine the location of the largest element, and swap it into the last position. Then repeat the process …

So in the first round:

- *look at all elements from* `A[0]` *to* `A[n-1]`,
- *determine the location (?) of the largest element,*
- *swap it into the last position, ie.* `A[n-1]`.

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

7.6: understanding (n=5)

Round 1: • *determine* `imax`= *index of the largest of* `A[0]` *to* `A[4]`, • *swap* `A[imax]` & `A[4]`.	
Round 2: • *determine* `imax`= *index of the largest of* `A[0]` *to* `A[3]`, • *swap* `A[imax]` & `A[3]`.	
Round 3: • *determine* `imax`= *index of the largest of* `A[0]` *to* `A[2]`, • *swap* `A[imax]` & `A[2]`.	
Round 4: • *determine* `imax`= *index of the largest of* `A[0]` *to* `A[1]`, • *swap* `A[imax]` & `A[1]`.	

7.6: recursive version

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

Recursion:

- When having a task of size `n` (here size is size of data), we:
- convert the task to one or more "same tasks" of smaller size, and
- remember to handle base case (when size `n` is so small like `0`, `1` which make the solution trivial).

7.6: recursive version

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

Typical structure for recursive function:

```
f(n) {
   if (n is a base case) {
      solve the base case and return
   }
   do something to reduce the task to f(n-1)
   f(n-1);
}
```

7.6: recursive version

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

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




}
```

7.7: understanding

7.7: Write a function that takes as arguments an integer array `A` and an integer `n` that indicates how many elements of `A` may be accessed, and returns the value of the integer in `A` that appears most frequently. If there is more than one value in `A` of that maximum frequency, then the smallest such value should be returned. The array `A` may not be modified.

Function header= ?

Basic task=? Is that similar to sorting? to searching?

7.7: approach/design

7.7: Write a function that returns the value of the integer in `A` that appears most frequently. On tie, returns the smallest such value. The array `A` may not be modified.

Lab

Working in group, or Implement 7.7, 7.8, 7.9, 7.10, 7.11

7.8: returns the *k*-smallest of `int A[]`, not modifying `A`

7.9: returns the number of runs in `int A[]`

7.10: returns the number of inversions in `int A[]`

(You can use skeleton and data from last week at a starting point – just to save some time)