Review and Warmup
CISC4080
CIS, Fordham Univ.

Instructor: X. Zhang

Goal

- Be comfortable with writing bubble sort, selection sort

- Practice basic building blocks (coding patterns)
- Step-wise refinement:
  - Write ideas as comments for a block of code
  - Be specific/accurate about what you are doing
    - Pay attention to boundary condition

    - Code: do what you need to do, exactly

- Next class: bubble sort, selection sort recursively, recursive thinking

List

- a list: a data structure (ADT) that stores a collection of elements (of same type), in which accessing a[i] (i-th element) takes constant amount of time (i.e., accessing a[1], a[2], …a[1000] takes same amount of time)
  - can be a C++ array, C++ STL vector
- **a sublist** a[i…j] where i>=0, j<=n-1, is a contiguous part of a list a[0…n-1]
  - e.g., a[1…8] is a sublist of a[0…9]
  - a[1…1] is a sublist of a[0…9] of length 1
  - a[3…2] is a null list (length is 0)

Can you complete this?

```
/* Search for a target value in list a
@param a: the list
@param n: length of list a
@param v: the value to search for
@return the first position where v appears in a; -1 if not found
*/
LinearSearch (a, n,  v)
{
        loc = -1 //not found yet
        for i = n-1 downto 0

                If (a[i]==v)
                        loc=i

        return loc
}
```

Find largest element

```
/* Find largest element in a sublist
 @param a: a list
 @param first, last: specify the sublist
 @return largest value stored in a[first…last]
*/
FindLargest (a[], first, last)
{

        largest=a[first]  //store the largest value seen so far

        for i=first+1 to last
          //scan through the rest of the list, for each new value seen (a[i])
          // update largest if a[i] is larger than “largest seen so far"
               If (a[i] > largest)
                    largest = a[i]



        return largest;
}
```

Pattern 1:

Scan through the list:

From lower end to higher end

Or from higher end to lower end

Index	0	1	2	3			n-1
A	3	5	2	11	…	…	42

```
for i=0; i<=n-1; i++
    access/processing A[i]
```

```
for i=n-1; i<=0; i—
    access/processing A[i]
```

Is a list sorted?

- Idea: to check if a list is sorted or not, we need to compare all adjacent pairs of element, to see if they are in order
  - All adjacent pairs are in order, then list is sorted
  - One pair in wrong order, then list is not sorted

```
IsSorted (a, n)
{
        for i=0 to n-2  //iterate through all possible I
              If (a[i] > a[i+1])  //compare adjacent pair
                    return false
         return true;
}
```

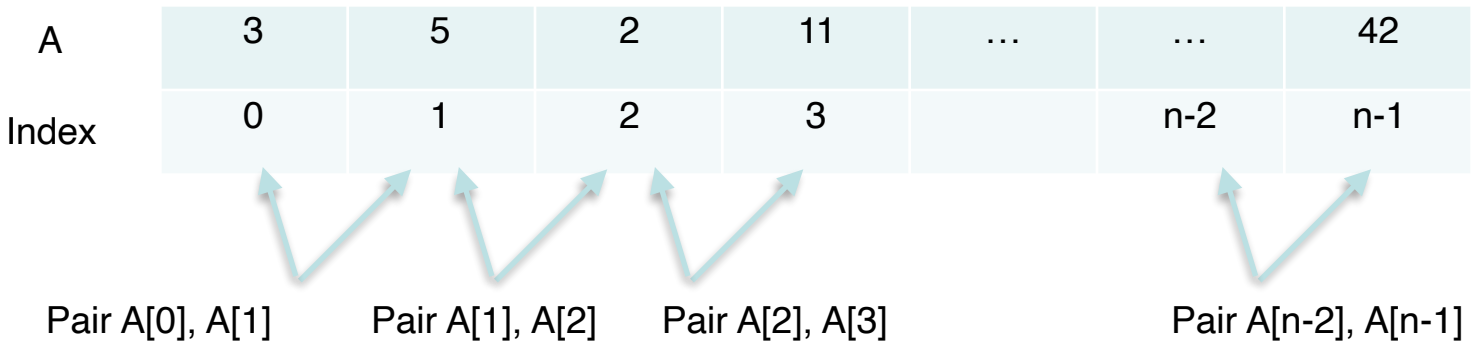
Pattern 2: all adjacent pairs

Scan through the list:

From lower end to higher end

// Or from higher end to lower end

Process adjacent pair: a[i] with the following element a[i+1]




```
for i=0; i<=n-2; i++
    access/processing A[i], A[i+1]
```

```
for i=n-2; i<=0; i—
    access/processing A[i], A[i+1]
```

Does a list contain duplicates?

- To check if a list contains duplicate values or not
  - For each element in the list, check if it appears in other place in the list

```
ContainDuplicate (a,n)
{
        For (int i=0; i<=n-1; i++) //for each element in list
        {
             //does a[i] appears elsewhere in the list?
             for (int j=0; j<=n-1; j++)
             {
                  If (a[i]==a[j] && i!=j) //a[i] appears somewhere else (pos j)
                       return true;
              }
         }
         return false;
}
```

- Pattern: enumerate all pairs in a list

Does a list contain duplicates?

- To check if a list contains duplicate values or not
  - For each element in the list, check if it appears in other place in the list

```
ContainDuplicate (a,n)
{
        For (int i=0; i<=n-1; i++) //for each e
        {
            //does a[i] appears elsewhere in the list?
            for (int j=i+1; j<=n-1; j++)
            {
                If (a[i]==a[j] && i!=j) //a[i] appears at pos j, somewhere else
                    return true;
            }
        }
        return false;
}
```

In previous sol, every pair is checked twice.
a[2] with checked against a[4]:
i=2, j=4; and then i=4,j=2

To check each pair only once: always check a[i] with Elements appear after it
i.e., j iterates through i+1… n-1

- Pattern: enumerate all pairs in a list

Pattern 3: all pairs

Scan through the list:

From lower end to higher end

Pair current element a[i] with each of elements goes after it

A	3	5	2	11	…	…	42
Index	0	1	2	3		n-2	n-1

Pair A[0] with A[1],
With A[2], …A[n-1]

```
for i=0; i<=n-2; i++
      // pair a[i] with each element in a[i+1…n-1]
       for j=i+1; j<=n-1; j++
              processing A[i], A[j] //e.g., if (A[i]==A[j]) …
```

Reverse a list

```
/* Reverse elements stored in the list
@param list:
@param n: length of list */
Reverse (list, n)
{
    int left=0, int right=n-1
    while (left<right) {
        swap (list[left], list[right])
        left+=1
        right-=1
    }
}
```

left, right starts from both ends
They move towards each other by one step, until meeting in the middle

Pattern 4: two indices from two ends

A	3	5	2	11	…	…	42
Index	0	1	2	3		n-2	n-1

Left —> <— right

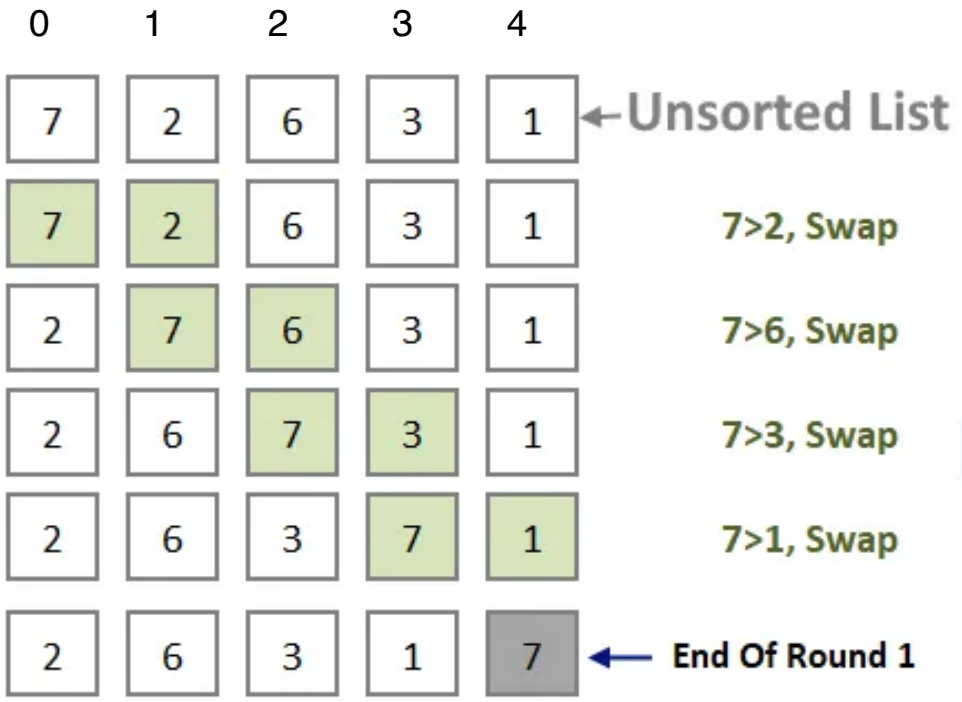
```
//Set left, right to points to two ends
left=0, right=n-1

// both walk to the middle; until meeting or passing each other
while (left<right) {
      Swap (A[left], A[right]) // or other operations…
}
```

bubble sort

- First round: scan list from left to right, compare each adjacent pair of elements, swap them if they are in wrong order

a



1) Define bubble sort function
2) Write comment for first round
3) Implement round 1

one bubbling round?

```
/*Bubble largest element to right as in bubble sort
   @param a: the list
   @param n: length of a
*/
bubbleRound (a, n)
{
        //scan list from left to right, compare each adjacent pair of elements, swap them if they are in wrong order
}
```

one bubbling round?

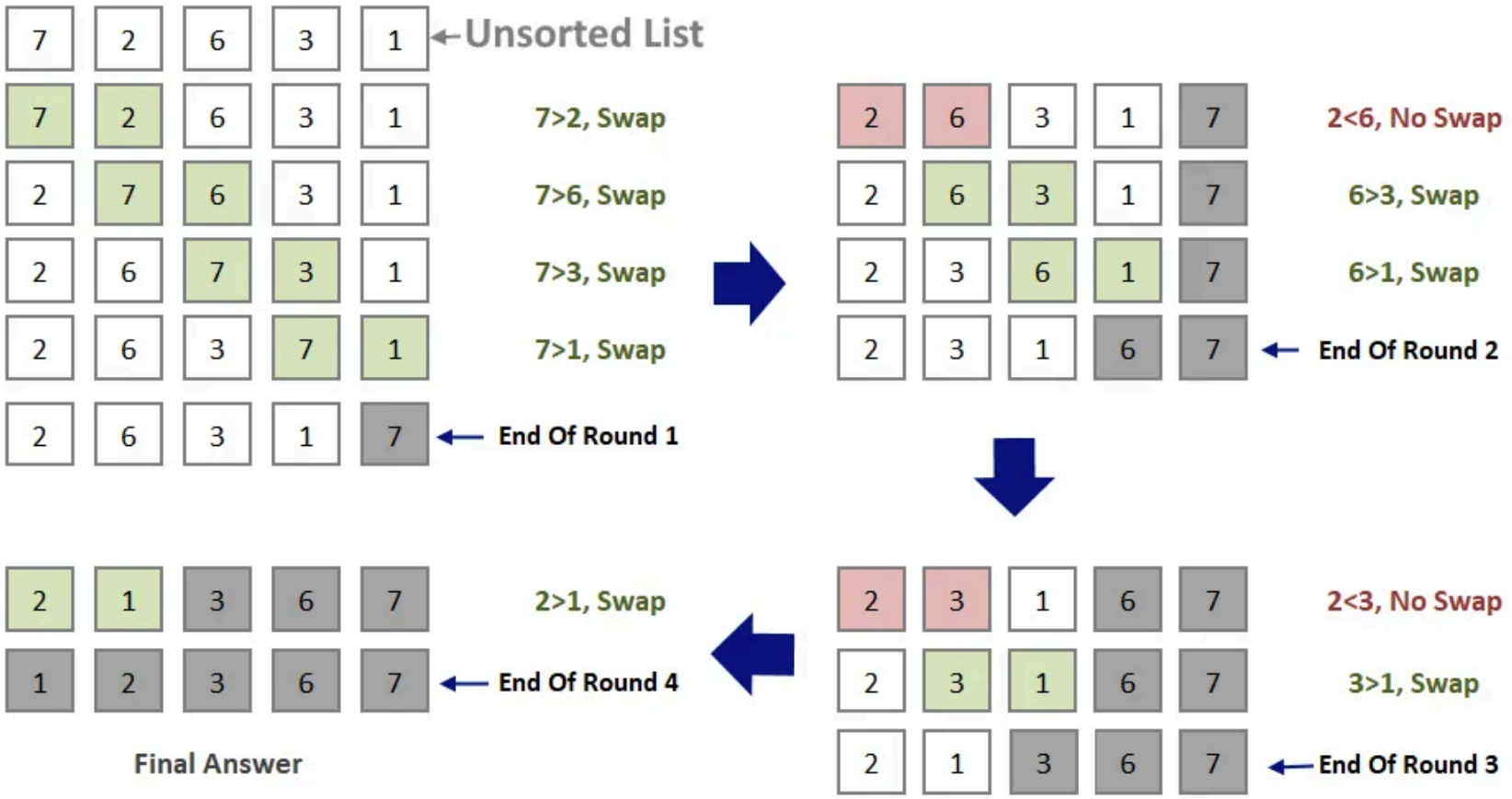
```
/*Bubble largest element to right as in bubble sort
   @param a: the list
   @param n: length of a
*/
bubbleRound (a, n)
{
        //scan list from left to right, compare each adjacent pair of elements, swap them if they are in wrong order
       for (int i=0; i<=n-1;i++)
          If (a[i] > a[i+1])
               swap (a[i], a[i+1])
}
```

Check boundary condition:
Look at boundary value for I, and see what's happens at these boundary condition:
i=0 => compare a[0] with a[1]
i=n-1 => compare a[n-1] with a[n-1+1]

How to fix?

bubble sort

- We can then repeat n-1 rounds to sort whole list
  - or repeat until there is no swap in prev round



BubbleUp

- From Idea to Code …

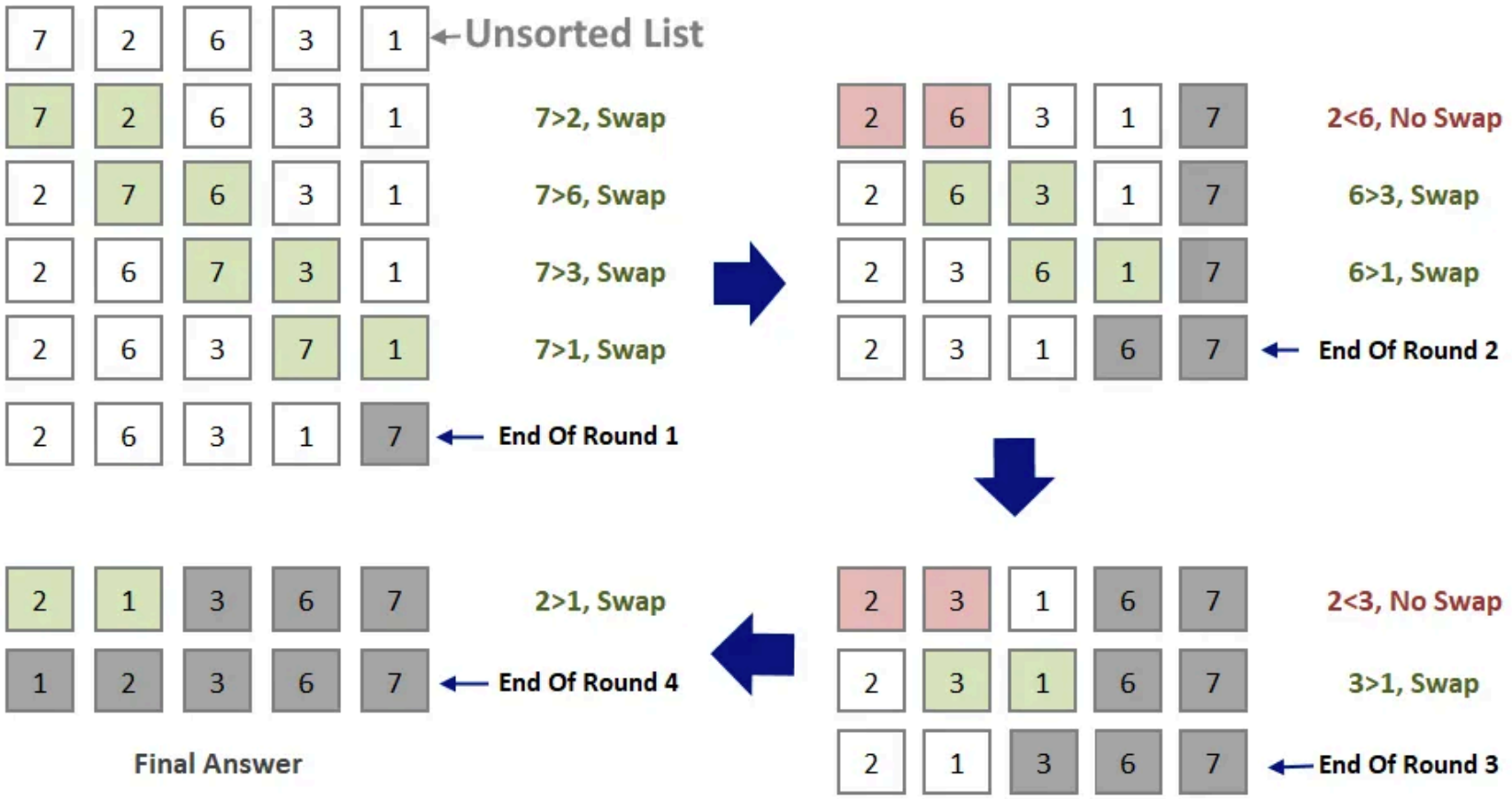
BubbleSort: v1

```
Bubblesort (a,n)
{
     for (int j=0;j<n-1;j++) {
          //performing a bubble round for a[0…n-1]
          for (int i=0;i<=n-2;i++)
             if (a[i]>a[i+1])
                swap (a[i], a[i+1]);
    }
}
```

bubble sort

4) Add outer-loop to repeat for n-1 rounds
5)* ignore gray elements…

- We can then repeat n-1 rounds to sort whole list
  - or repeat until there is no swap in prev round



BubbleSort: v2

```
//the range of bubbleup round shrinks …
Bubblesort (a,n)
{
      for (int j=0;j<n-1;j++) { //j: which round
            //performing a bubble round for a[0…n-1-j]
            for (int i=0;i<n-1-j;i++)
                if (a[i]>a[i+1])
                   swap (a[i], a[i+1]);
     }
}
```

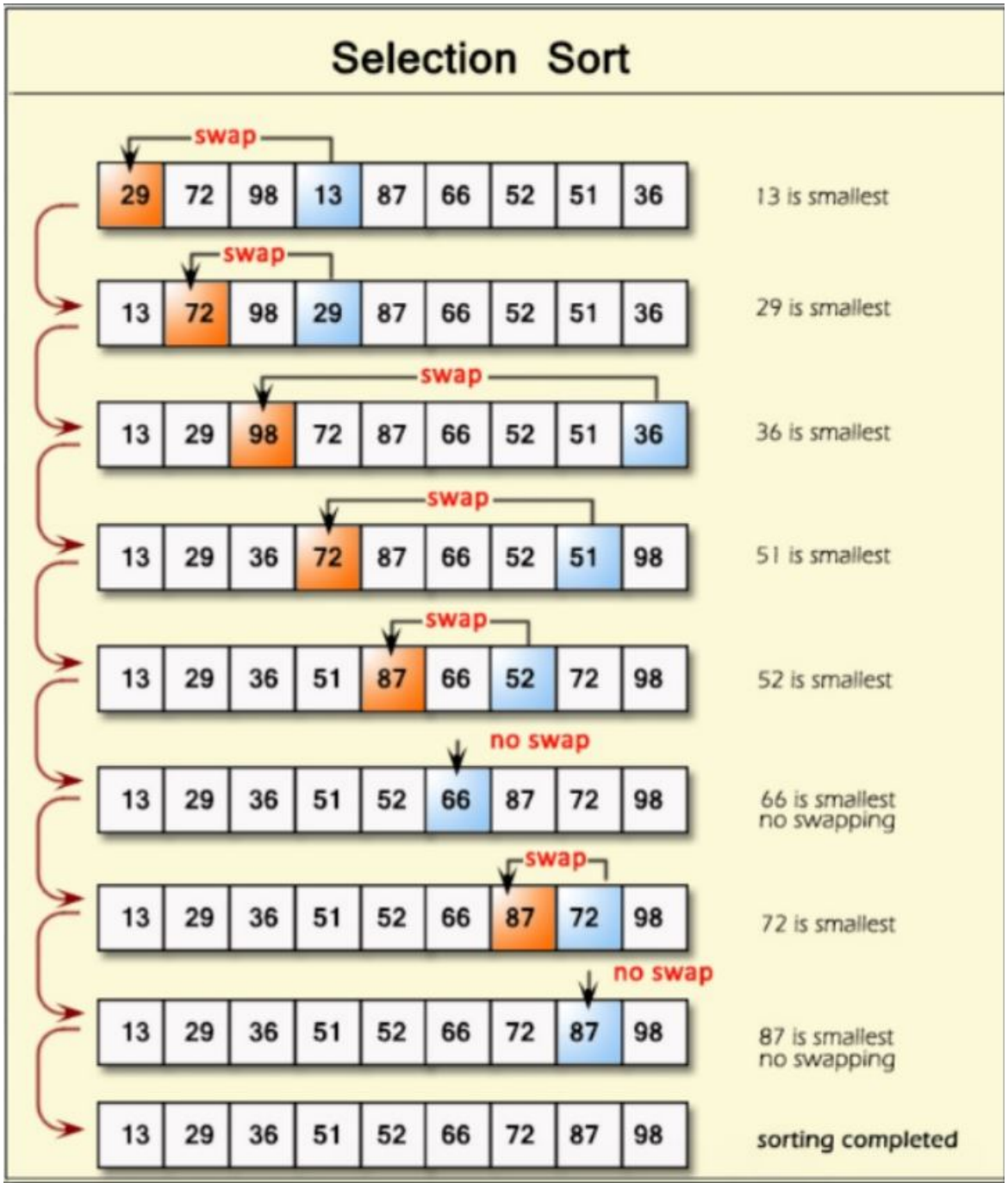
Check outerloop:

When j=0, range is a[0…n-1]
When j=n-2, range is a[0…1]

Check inner loop:
i=0, a[0], a[1] are compared
i=n-1-j-1, a[n-2-j] and a[n-1-j] are compared

BubbleSort: v3

```
//the range of bubbleup round shrinks …
// if there is no swap in a particular round, then the list
// is sorted!
Bubblesort (a,n)
{
     hasSwap;
     for (int j=0;j<n-1;j++) { //j: which round
          hasSwap = false;
          //performing a bubble round for a[0…n-1-j]
          for (int i=0;i<n-1-j;i++)
              if (a[i]>a[i+1]) {
                 swap (a[i], a[i+1]);
                 hasSwap=true;
          }
          If (!hasSwap)
               Return true; //finish a round, in which there is no swap
    }
}
```


Selection Sort
swap
29 72 98 13 87 66 52 51 36
13 is smallest
swap
13 72 98 29 87 66 52 51 36
29 is smallest
swap
13 29 98 72 87 66 52 51 36
36 is smallest
swap
13 29 36 72 87 66 52 51 98
51 is smallest
swap
13 29 36 51 87 66 52 72 98
52 is smallest
no swap
13 29 36 51 52 66 87 72 98
66 is smallest
no swapping
swap
13 29 36 51 52 66 87 72 98
72 is smallest
no swap
13 29 36 51 52 66 72 87 98
87 is smallest
no swapping
13 29 36 51 52 66 72 87 98
sorting completed

SelectionSort

From Idea to Code …