Southern Technical University
Technical Institute / Qurna
Dep. of Computer Systems Techniques

Second class
Subject : Data Structures
Lecturer : Israa Mahmood Hayder
Lecture no.20,21

الوحــدة النمطية السادسة عشر

الترتيب(Sorting)

- الاسبوع العشرون-الثالث والعشرون

* الترتيب والبحث sorting and searching. - خوارزميات الترتيب sorting algorithms. - الترتيب بالاختبار selection sort. - ترتيب الفقاعة bubble sort. - الترتيب السريع quick sort.	العشرون-الثالث والعشرون

B// Rationale (مبررات الـوحدة) :-

This unit studies several important methods for sorting lists, both contiguous lists and linked lists. At the same time, we shall develop further tools that help with the analysis of algorithms and apply these to determine which sorting methods perform better under different circumstances.

C// Central Ideas (الـفكـرة المـركـزية):-

- Sorting algorithms
- Selection sort
- Bubble sort
- Quick Sort

D// Objectives (أهداف الـوحـدة):-

After studying this unit, the student will be able to use following ways in sorting:

- Selection sort
- Bubble sort
- Quick sort

1-Sorting

We live in a world obsessed with keeping information, and to find it, we must keep it in some sensible order. Several years ago, it was estimated, more than half the time on many commercial computers was spent in sorting. This is perhaps no longer true, since sophisticated methods have been devised for organizing data, methods that do not require that the data be kept in any special order. Eventually, nonetheless, the information does go out to people, and then it must often be sorted in some way.

Amongst the differing environments that require different methods, the most important is the distinction between ***external*** and ***internal***; that is, whether there are so many records to be sorted that *external and internal sorting* they must be kept in external files on disks, tapes, or the like, or whether they can all be kept internally in high-speed memory. In this chapter, we consider only internal sorting.

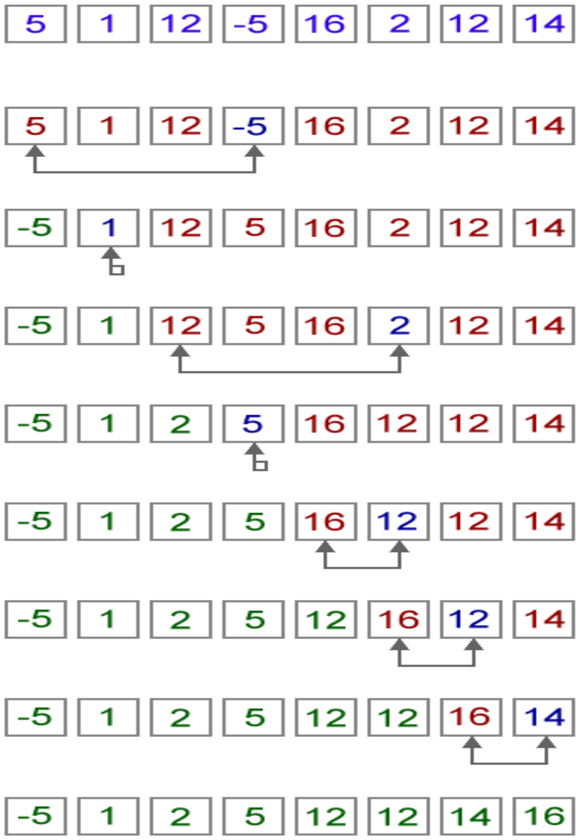
2- Selection Sort

Selection sort is one of the O(n^2) sorting algorithms, which makes it quite inefficient for sorting large data volumes. Selection sort is notable for its programming simplicity and it can over perform other sorts in certain situations (see complexity analysis for more details).

Algorithm

The idea of algorithm is quite simple. Array is imaginary divided into two parts - sorted one and unsorted one. At the beginning, sorted part is **empty**, while unsorted one contains **whole array**. *At every step*, algorithm finds **minimal element** in the unsorted part and adds it to the end of the sorted one. When unsorted part becomes **empty**, algorithm *stops*. When algorithm sorts an array, it swaps first element of unsorted part with minimal element and then it is included to the sorted part. This implementation of selection sort in **not stable**. In case of linked list is sorted, and, instead of swaps, minimal element is linked to the unsorted part, selection sort is **stable**.

Example. **Sort {5, 1, 12, -5, 16, 2, 12, 14} using selection sort.**



chose the smallest & exchange

5	1	12	-5	16	2	12	14
5	1	12	-5	16	2	12	14
-5	1	12	5	16	2	12	14
-5	1	12	5	16	2	12	14
-5	1	2	5	16	12	12	14
-5	1	2	5	16	12	12	14
-5	1	2	5	12	16	12	14
-5	1	2	5	12	12	16	14
-5	1	2	5	12	12	14	16

Ex2) Sort {8, 11, 5, 26, 7} using Bubble sort.

```
8, 11, 5, 26, 7
*        *
8, 11, 5, 26, 7
 *          *
5, 11,8, 26, 7
*            *
5, 11, 8, 26, 7
5, 11, 8, 26, 7

----------------
      *   *
5,|  11, 8, 26, 7
      *        *
5,   8, 11, 7, 26
      *          *
5,   7, 11, 8, 26
5,   7, 11, 8, 26
---------------
         *   *
```

5, 7,| 11, 8, 26

5, 7, $\overset{*}{8}$, 11, $\overset{*}{26}$

5, 7, 8, 11, 26

5, 7, 8, | $\overset{*}{11}$, $\overset{*}{26}$

5,   7, 8, 11, 26

Complexity analysis

Selection sort stops, when unsorted part becomes empty. As we know, on every step number of unsorted elements decreased by one. Therefore, selection sort makes n steps (n is number of elements in array) of outer loop, before stop. Every step of outer loop requires finding minimum in unsorted part. Summing up, n + (n - 1) + (n - 2) + ... + 1, results in O(n^2) number of comparisons. Number of swaps may vary from zero (in case of sorted array) to n - 1 (in case array was sorted in reversed order), which results in O(n) number of swaps. Overall algorithm complexity is O(n^2).Fact, that selection sort requires n - 1 number of swaps at most, makes it very efficient in situations, when write operation is significantly more expensive, than read operation.

C++ code

```
void selectionSort(int arr[], int n) {
    int i, j, minIndex, tmp;
    for (i = 0; i < n - 1; i++) {
        minIndex = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[minIndex])
                minIndex = j;
        if (minIndex != i) {
            tmp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = tmp;
        }  }}
```

- الاسبوع الثاني والعشرون –

Sorting methods

3- Bubble Sort

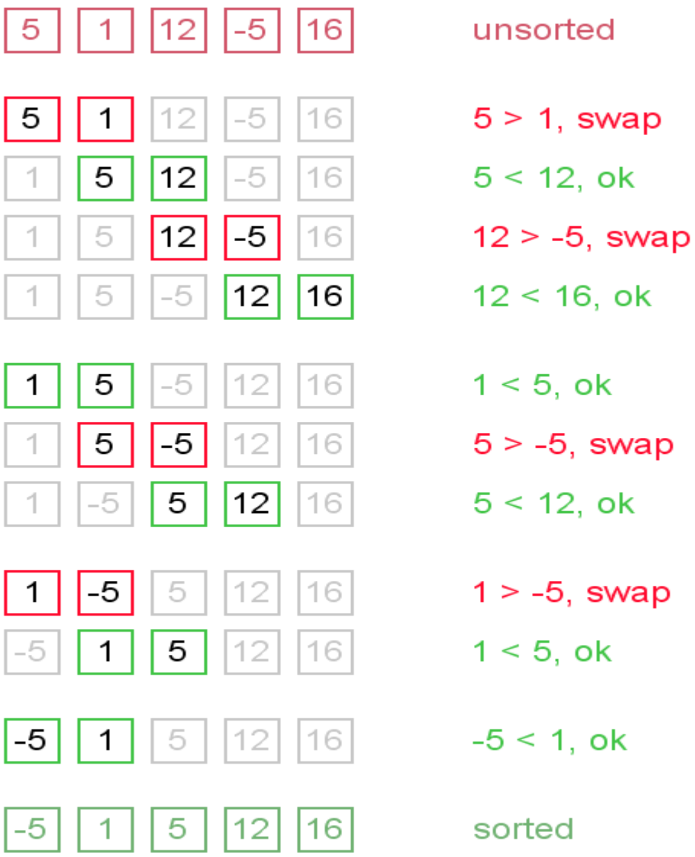
Bubble sort is a simple and well-known sorting algorithm. It is used in practice once in a blue moon and its main application is to make an introduction to the sorting algorithms. Bubble sort belongs to O(n^2) sorting algorithms, which makes it quite inefficient for sorting large data volumes. Bubble sort is **stable** and **adaptive**.

Algorithm

1. Compare each pair of adjacent elements from the beginning of an array and, if they are in reversed order, swap them.
2. If at least one swap has been done, repeat step 1.

You can imagine that on every step big bubbles float to the surface and stay there. At the step, when no bubble moves, sorting stops. Let us see an example of sorting an array to make the idea of bubble sort clearer.

Example. Sort {5, 1, 12, -5, 16} using bubble sort.



5 1 12 -5 16 unsorted

5 1 12 -5 16 5 > 1, swap

1 5 12 -5 16 5 < 12, ok

1 5 12 -5 16 12 > -5, swap

1 5 -5 12 16 12 < 16, ok

1 5 -5 12 16 1 < 5, ok

1 5 -5 12 16 5 > -5, swap

1 -5 5 12 16 5 < 12, ok

1 -5 5 12 16 1 > -5, swap

-5 1 5 12 16 1 < 5, ok

-5 1 5 12 16 -5 < 1, ok

-5 1 5 12 16 sorted

<u>Ex2</u>: Sort {8, 11, 5, 26, 7} using Bubble sort.

```
8, 11, 5, 26, 7
    *   *
8, 11, 5, 26, 7
       *   *
8, 5,11, 26, 7
            *   *
8, 5, 11, 26, 7

8, 5, 11, 7, 26
----------------
*   *
8, 5, 11, 7,| 26
    *   *
5, 8, 11, 7, 26
       *   *
5, 8, 11, 7, 26
---------------
*   *
8, 5, 7, |11, 26
    *   *
5, 8, 7, 11, 26

5, 7, 8, 11, 26
-----------------
*   *
5, 7, | 8, 11, 26
```

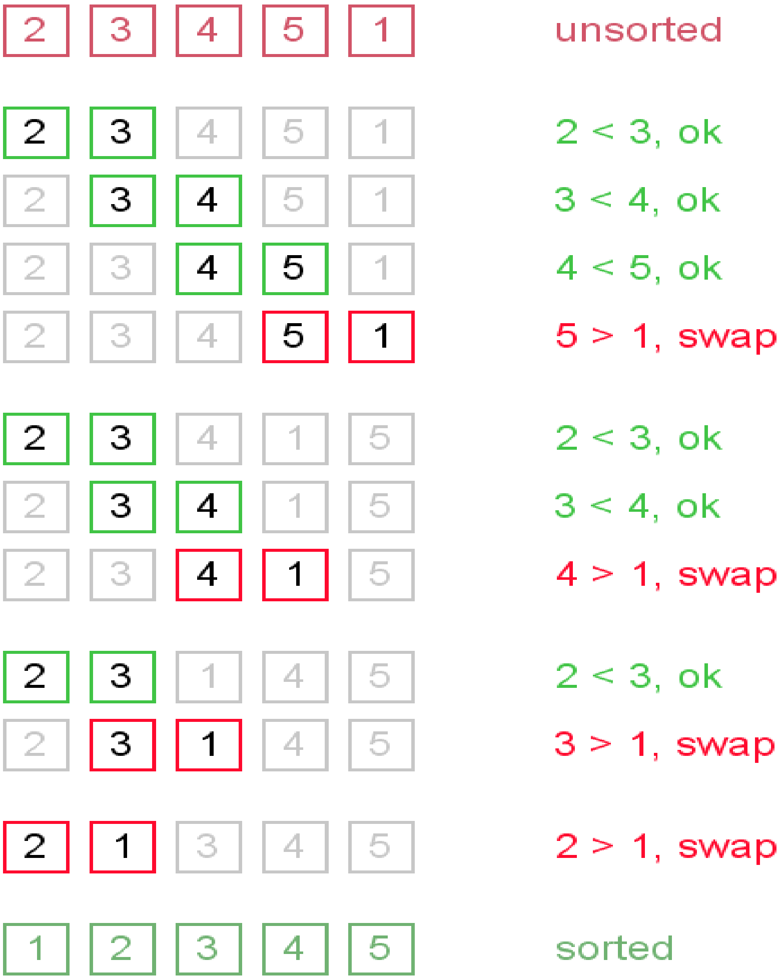
Complexity analysis

Average and worst case complexity of bubble sort is O(n^2). Also, it makes O(n^2) swaps in the worst case. Bubble sort is adaptive. It means that for almost sorted array it gives O(n) estimation. Avoid implementations, which don't check if the array is already sorted on every step (any swaps made). This check is necessary, in order to preserve adaptive property.

Turtles and rabbits

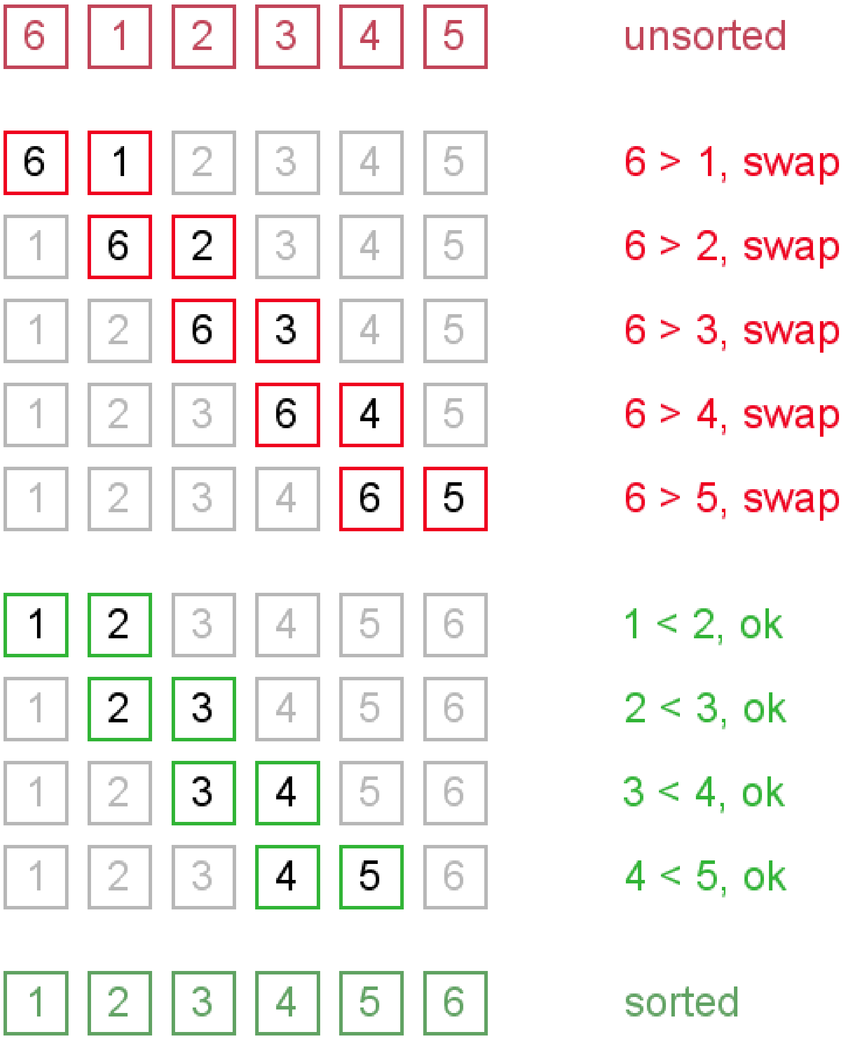
One more problem of bubble sort is that its running time badly depends on the initial order of the elements. Big elements (rabbits) go up fast, while small ones (turtles) go down very slow. This problem is solved in the Cocktail sort.

Turtle example. Thought, array {2, 3, 4, 5, 1} is almost sorted, it takes O(n^2) iterations to sort an array. Element {1} is a turtle.



Rabbit example:

Array {6, 1, 2, 3, 4, 5} is almost sorted too, but it takes O(n) iterations to sort it. Element {6} is a rabbit. This example demonstrates adaptive property of the bubble sort.



There are several ways to implement the bubble sort. Notice, that "swaps" check is absolutely necessary, in order to preserve adaptive property.