Bubble Sort Algorithm

This algorithm works by comparing adjacent elements, swapping them if they are not in order.

Advantages:

Simple to implement and easy to understand.

Disadvantages:

Slow in comparison to other algorithms and highly inefficient for large collections.

Dimitri Vasiliev

Practical way

```
public static int[] bubbleSort (int ArrayInt[]){
    int Temp = 0;
    boolean Swapping = true;
    while(Swapping){
        Swapping = false;
        for(int I = 0 ; I < ArrayInt.length-1 ; I++){
            if(ArrayInt[I] > ArrayInt[I+1]){
                Temp = ArrayInt[I];
                ArrayInt[I] = ArrayInt[I+1];
                ArrayInt[I+1] = Temp;
                Swapping = true;
            }
        }
    }
    return ArrayInt;
}
```

Popular way

```
public static int[] bubbleSort (int ArrayInt[]){
    int Temp = 0;
    for(int J = 0 ; J < ArrayInt.length ; J++){
        for(int I = 0 ; I < ArrayInt.length-1 ; I++){
            if(ArrayInt[I] > ArrayInt[I+1]){
                Temp = ArrayInt[I];
                ArrayInt[I] = ArrayInt[I+1];
                ArrayInt[I+1] = Temp;
            }
        }
    }
    return ArrayInt;
}
```

Differences

The ***practical way*** stops iterating when the algorithm is not swapping elements any more. Meaning that our array is sorted.

Whereas the ***popular way*** stops iterating after going through the entire array.

Case example

Array Element	0	1	2	3	5	Comparison	Action
Unsorted	8	3	1	5	0	8 > 3	Swap 8 and 3
1st iteration	3	8	1	5	0	8 > 1	Swap 8 and 1
2nd iteration	3	1	8	5	0	8 > 5	Swap 8 and 1
3rd iteration	3	1	5	8	0	8 > 0	Swap 8 and 0
4th iteration	3	1	5	0	8	3 > 1	Swap 3 and 1
5th iteration	1	3	5	0	8	3 > 5	Don't swap
6th iteration	1	3	5	0	8	5 > 0	Swap 5 and 3
7th iteration	1	3	0	5	8	5 > 8	Don't swap
8th iteration	1	0	3	5	8	1 > 0	Swap 1 and 0
9th iteration	0	1	3	5	8	1 > 3	Don't swap
Sorted	0	1	3	5	8		

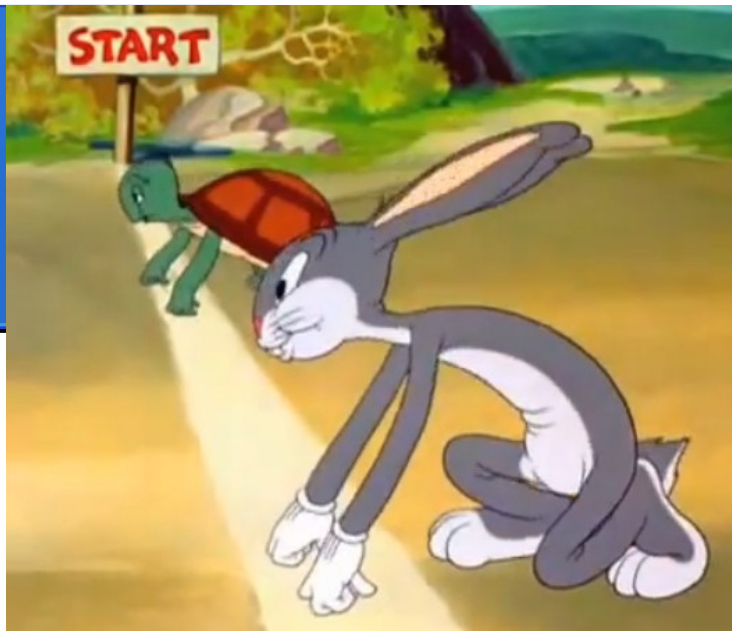
The problem

```
int ArrayInt[] = new int[]{8,3,1,5,0};
```

The Bubble Sort Algorithm will take the first element, which is *8*, in our example, comparing it with the rest of the elements. Since 8 is the *biggest* element in our array, it is going to ascent to its final position quickly.

But *0*, the *smallest* element, is the last one, and it has to be compared to all the other bigger elements to go down. This makes 0 descend to its final position slowly.

Cecil vs Bugs



Rabbits:

Big elements are called rabbits, because they go up fast.

Turtles:

Small elements are called turtles, because they go down slowly.

Conclusion

Bubble Sort is one of the first algorithms studied deeply.

It is easy to understand, study and implement.

It is ideal for small arrays.

It is more efficient than other algorithms when the array is almost sorted.

It should not be implemented in large arrays or in arrays in reverse order.