Algorithms and Data Structures

Andrzej Pisarski

Plan of the lecture

- Sorting
- The Bubble Sort
- The Insertion Sort
- The Polish Flag
- Sorting Objects
- Invariants and Stability

Sorting

- Why we need sorting?
  1. Arrange names in alphabetical order,
  2. Students by grade,
  3. Customers by zip code,
  4. House sales by price,
  5. Cities in order of increasing population,
  6. …

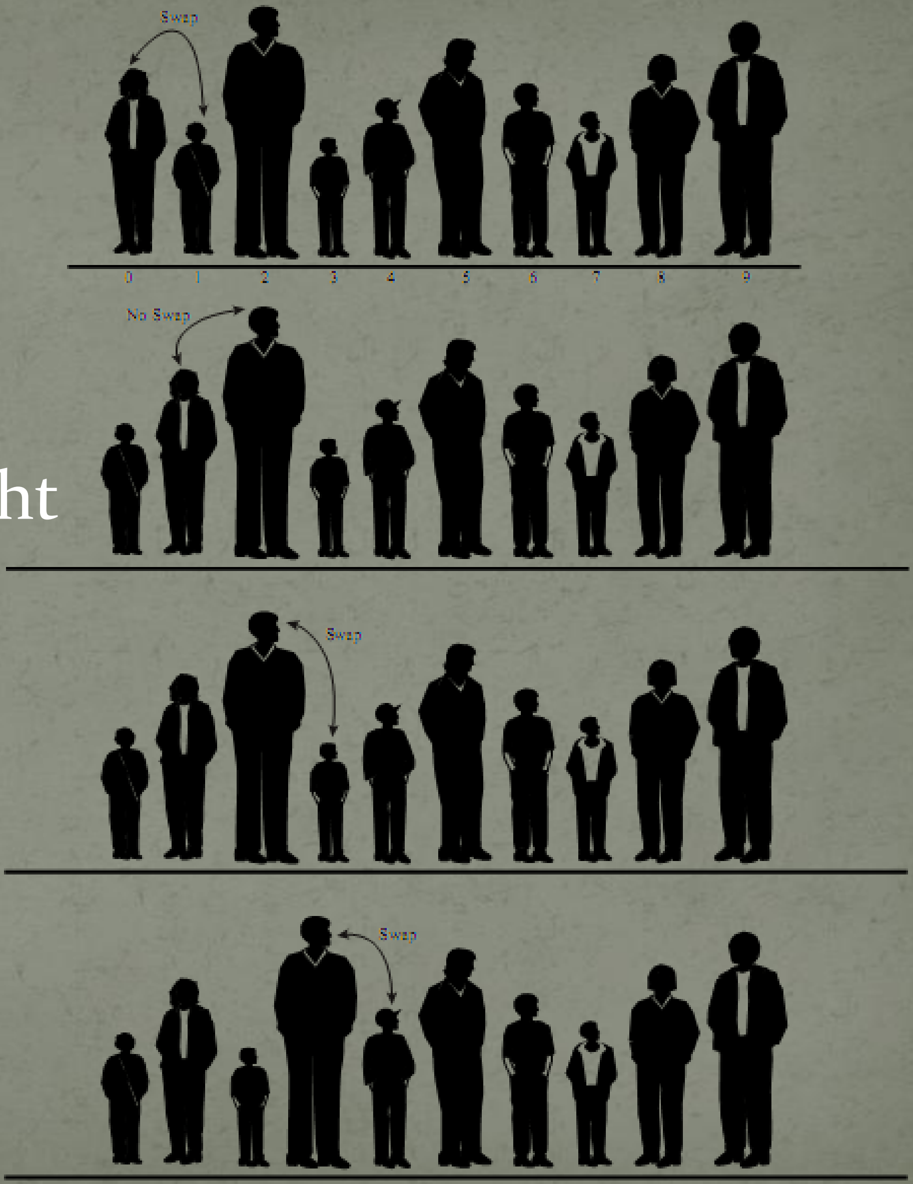
The Bubble Sort

1. Compare two adjacent items
2. If necessary swap items

The Bubble Sort

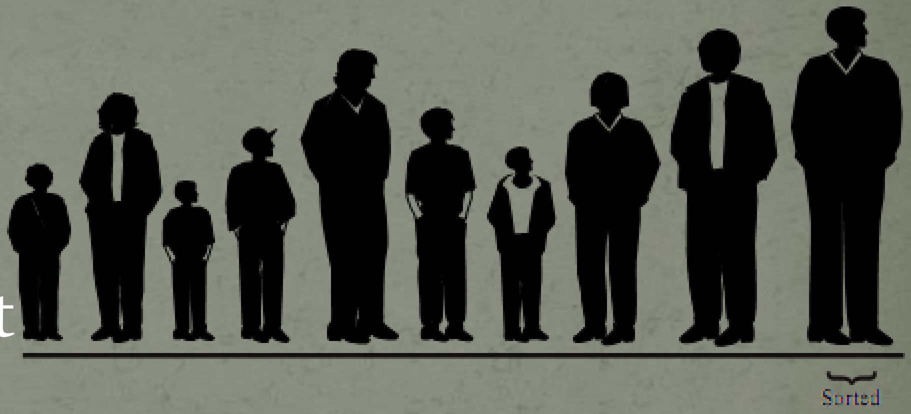
1. Compare two players
2. If the one on the left is taller, swap them
3. Move one position right

(example from R. Lafore book)



The Bubble Sort

1. Compare two players
2. If the one on the left is taller, swap them
3. Move one position right



(example from R. Lafore book)

End of first pass

The Bubble Sort

- Efficiency of the Bubble Sort

  For 10 items we need 9 comparisons on the first pass, 8 on the second, and so on:

$$9+8+7+6+5+4+3+2+1 = 45$$

(N-1) + (N-2) + (N-3) + ... + 1 = N*(N-1)/2

N*(N-1)/2 = 45 for N=10

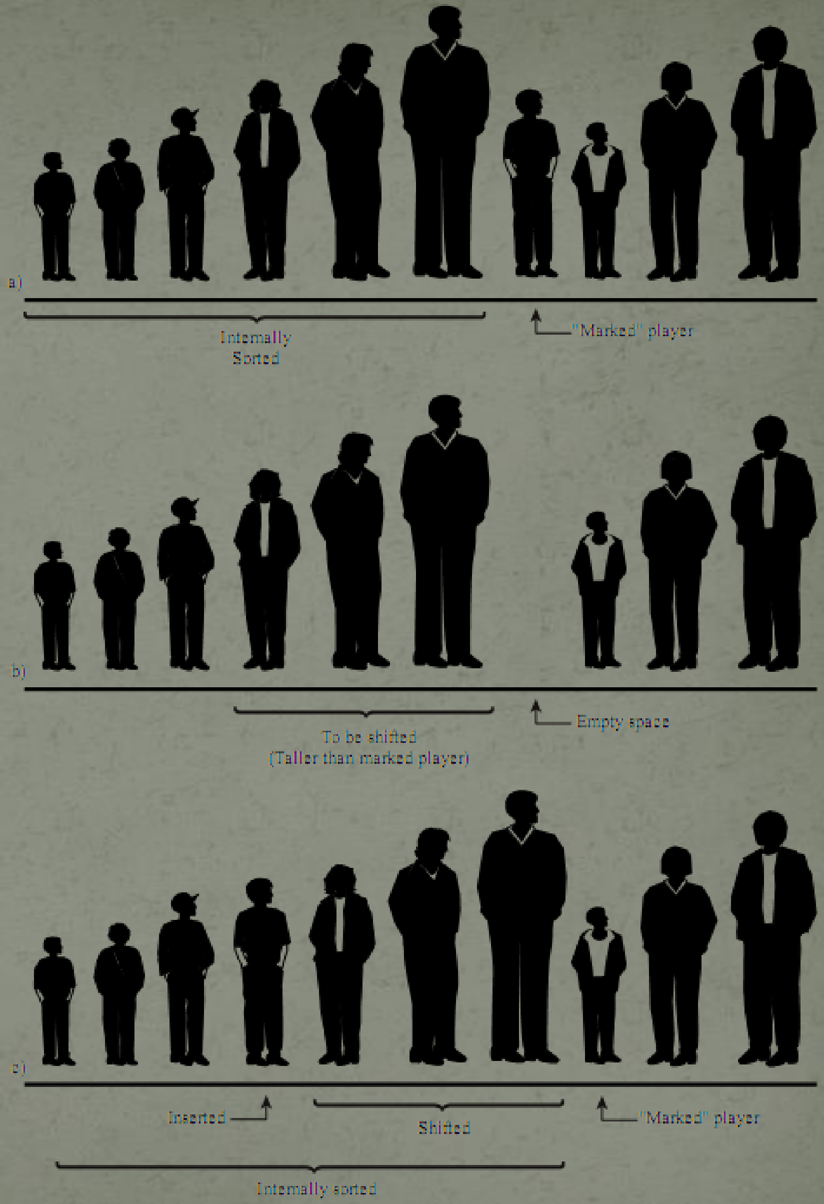
Algorithem makes about (N^2)/2 comparisions.

Big O notations: bubble sort runs in O(N^2) time.

The Insertion Sort

- Start baseball players lined up in random order.
- One player need to be „marked”.
- Palyers on the left side of the marked one are partilally sorted (among themselves).
- Players on the right side of the marked one are unsorted.
- Take the marked player out of line (to make space to tallest player on the left side (sorted))
- Move tallest sorted player one space right (need to apply to players taller than marked one).

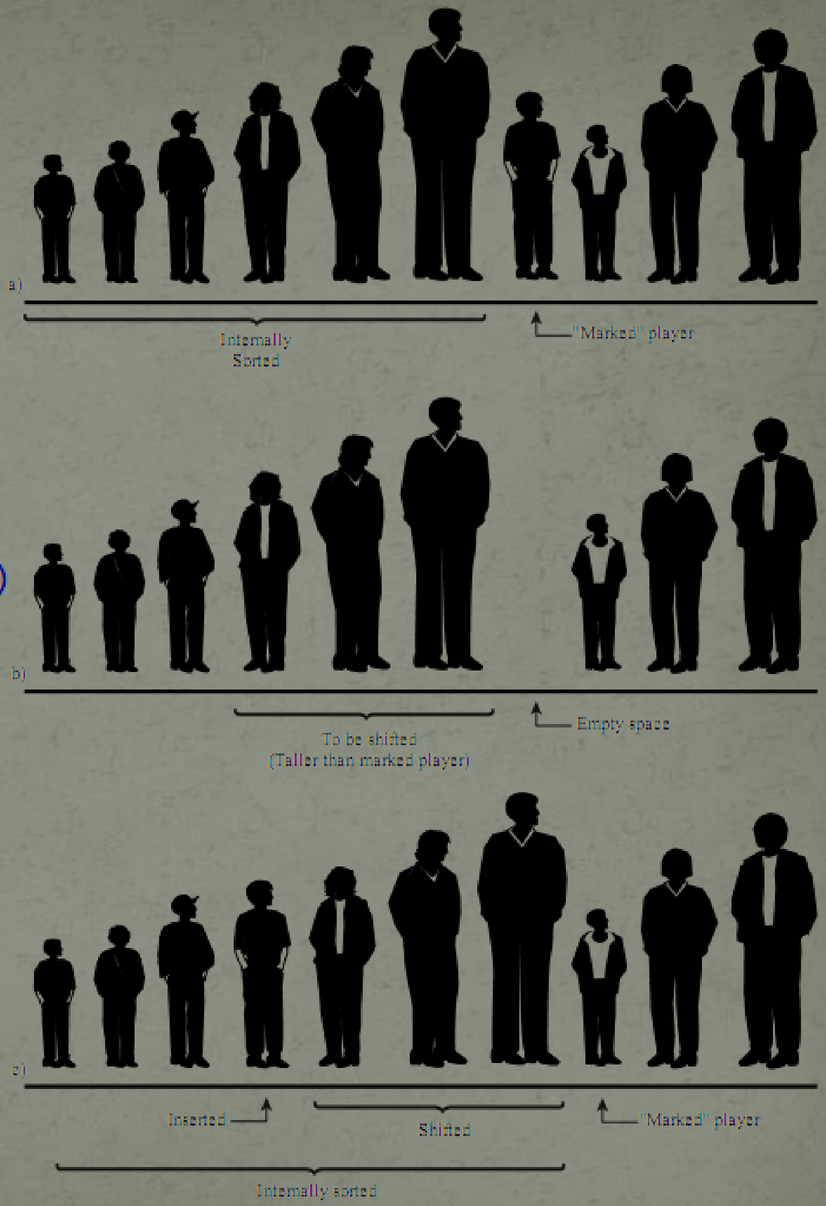
The Insertion Sort



(example from R. Lafore book)

The Insertion Sort

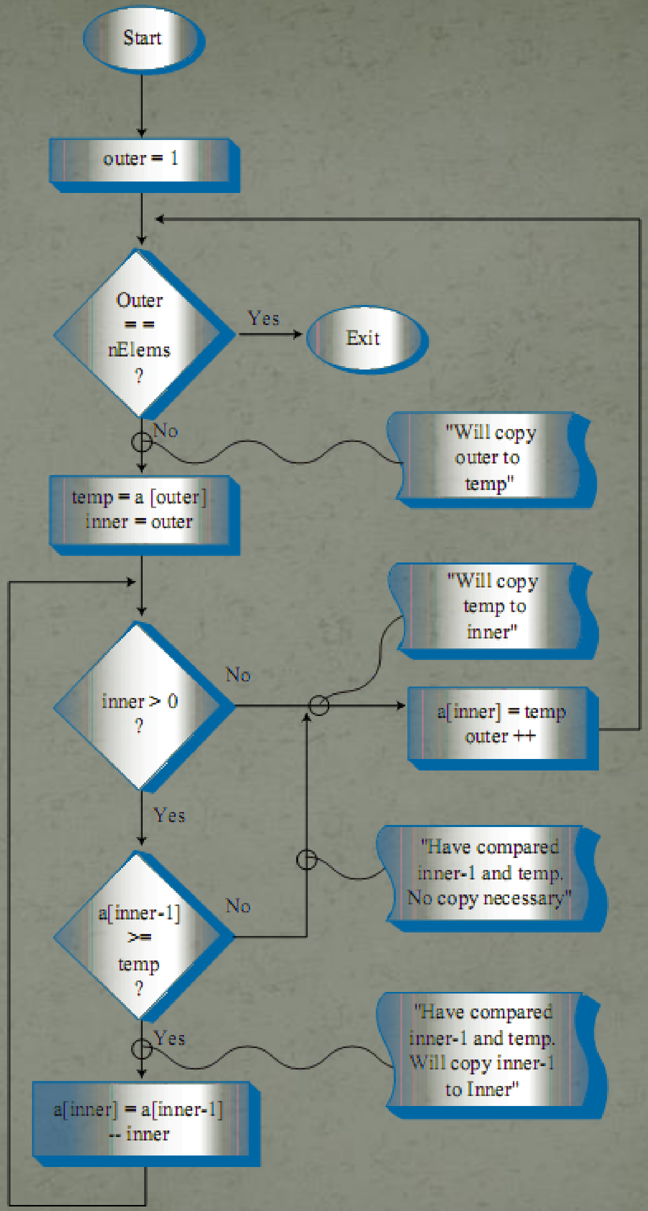
```
void insertionSort()
{
     int in, out;
     for(out=1; out<nElems; out++)
     {
          double temp = v[out];
          in = out;
          while(in>0 && v[in-1] >= temp)
          {
               v[in] = v[in-1];
               --in;
          }
          v[in] = temp;
     }  //end for
}  //end insertionSort()
```



(example from R. Lafore book)

The Insertion Sort

```
void insertionSort()
{
    int in, out;
    for(out=1; out<nElems; out++)
    {
        double temp = v[out];
        in = out;
        while(in>0 && v[in-1] >= temp)
        {
            v[in] = v[in-1];
            --in;
        }
        v[in] = temp;
    }   //end for
}   //end insertionSort()
```



(example from R. Lafore book)

The Insertion Sort

- Efficiency of the Insertion Sort

  How many comaprision an Insertion Sort require?

  Maximum1 comparison on the first pass, maximum 2 on the second, and so on, up to N-1 on the last pass:

$$1 + 2 + 3 + \ldots + N-1 = N*(N-1)/2$$

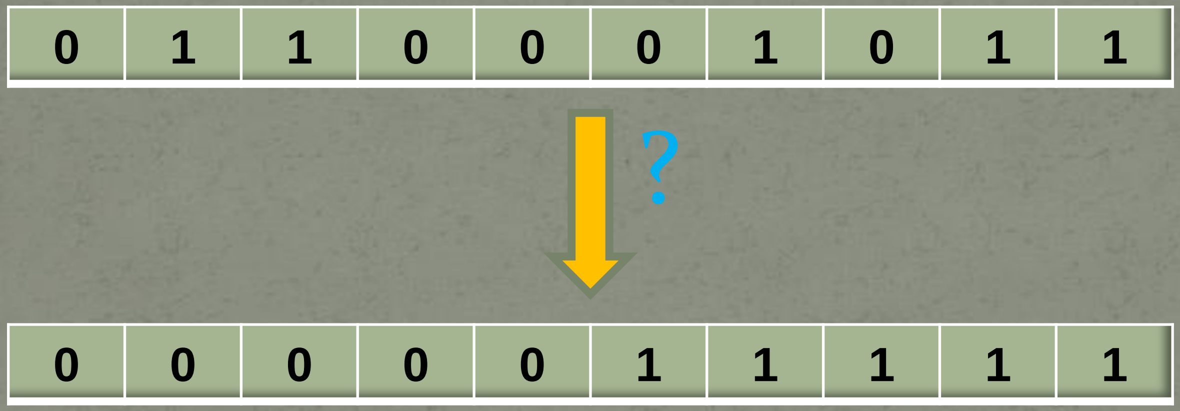
An average comparision is usualy a half of items actually compared

$$N*(N-1)/4$$

Big O notations: insertion sort runs in O(N^2) time.

The Polish Flag

- Problem: find algorithm which can be used to sort random data containing only 0 and 1(we want to get all 0 on the left and 1 on the right side of the series):



0	1	1	0	0	0	1	0	1	1

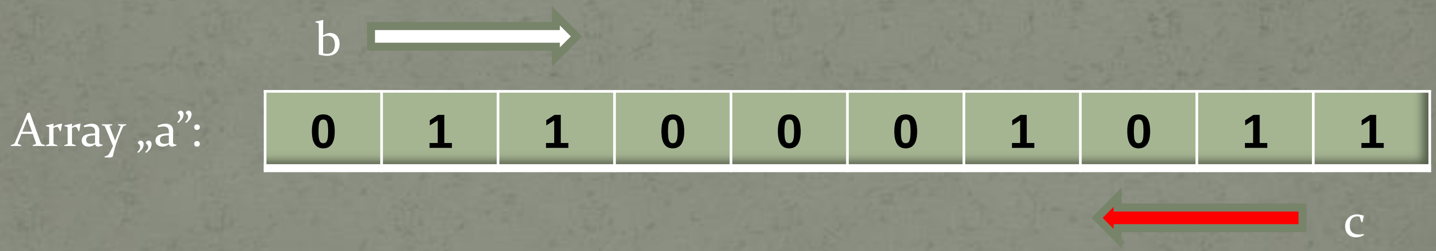
?

0	0	0	0	0	1	1	1	1	1

The Polish Flag

- Problem: find algorithm which can be used to sort random data containing only 0 and 1(we want to get all 0 on the left and 1 on the right side of the series):

We can use two markers: b and c:



Array „a”:

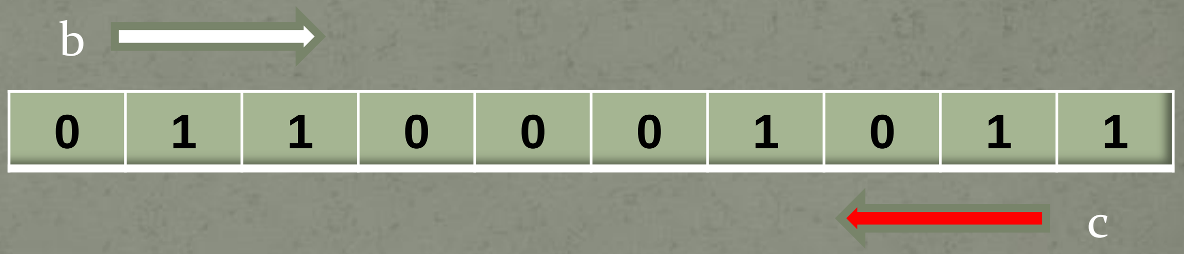
0	1	1	0	0	0	1	0	1	1

Invariant:
elements a[i] for i < b need to be 0 &&
elements a[i] for i > c need to be 1

The Polish Flag

- Problem: find algorithm which can be used to sort random data containing only 0 and 1(we want to get all 0 on the left and 1 on the right side of the series):

We can use two markers: b and c:



Array „a”:	0	1	1	0	0	0	1	0	1	1

Invariant:
elements a[i] for i < b need to be 0 &&
elements a[i] for i > c need to be 1

For b==c array „a” is sorted.

The Polish Flag

- Why Polish flag?

0	0	0	0	0	1	1	1	1	1

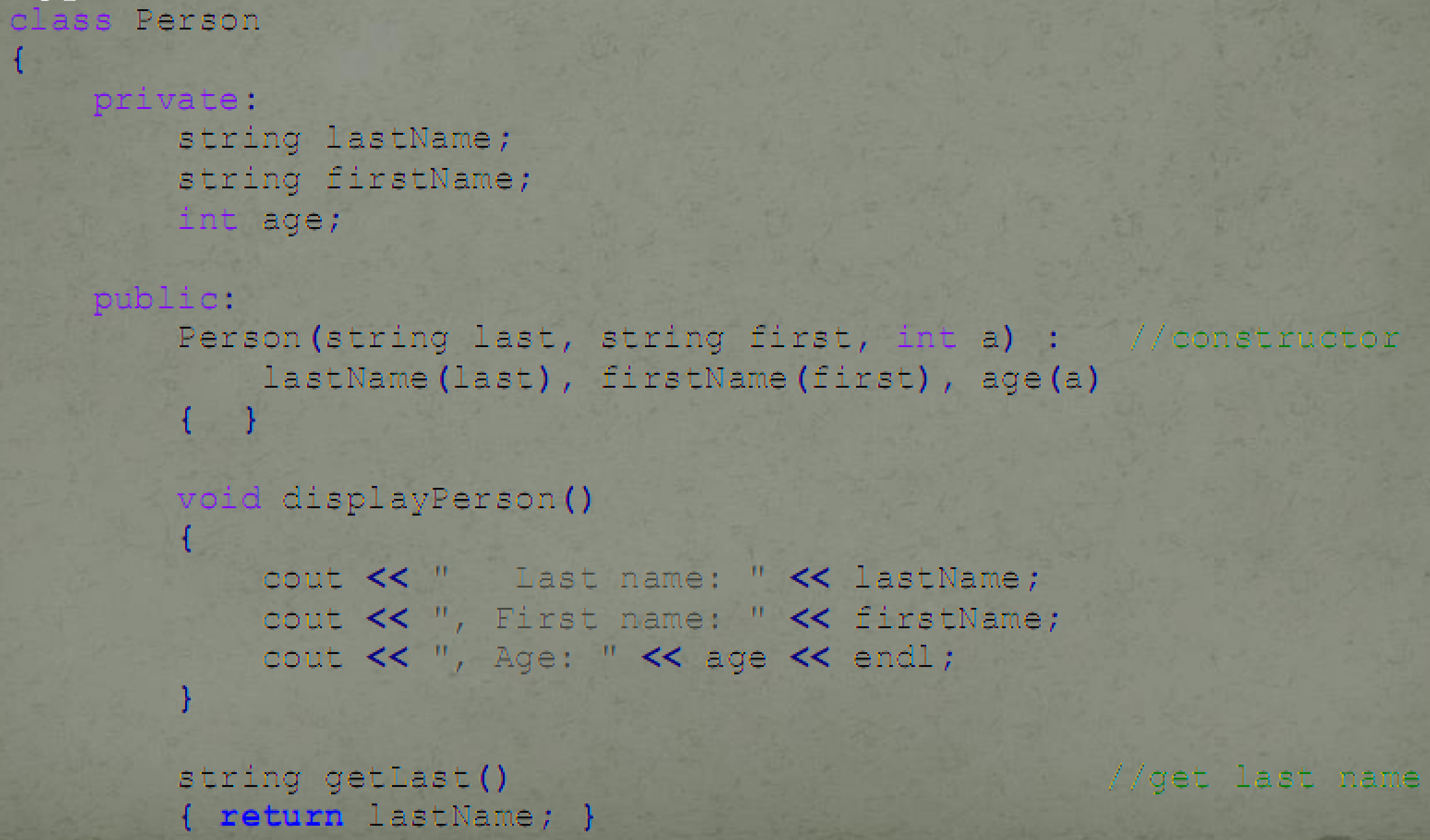
The Polish Flag

- Why Polish flag?

0	0	0	0	0	1	1	1	1	1

Sorting Objects

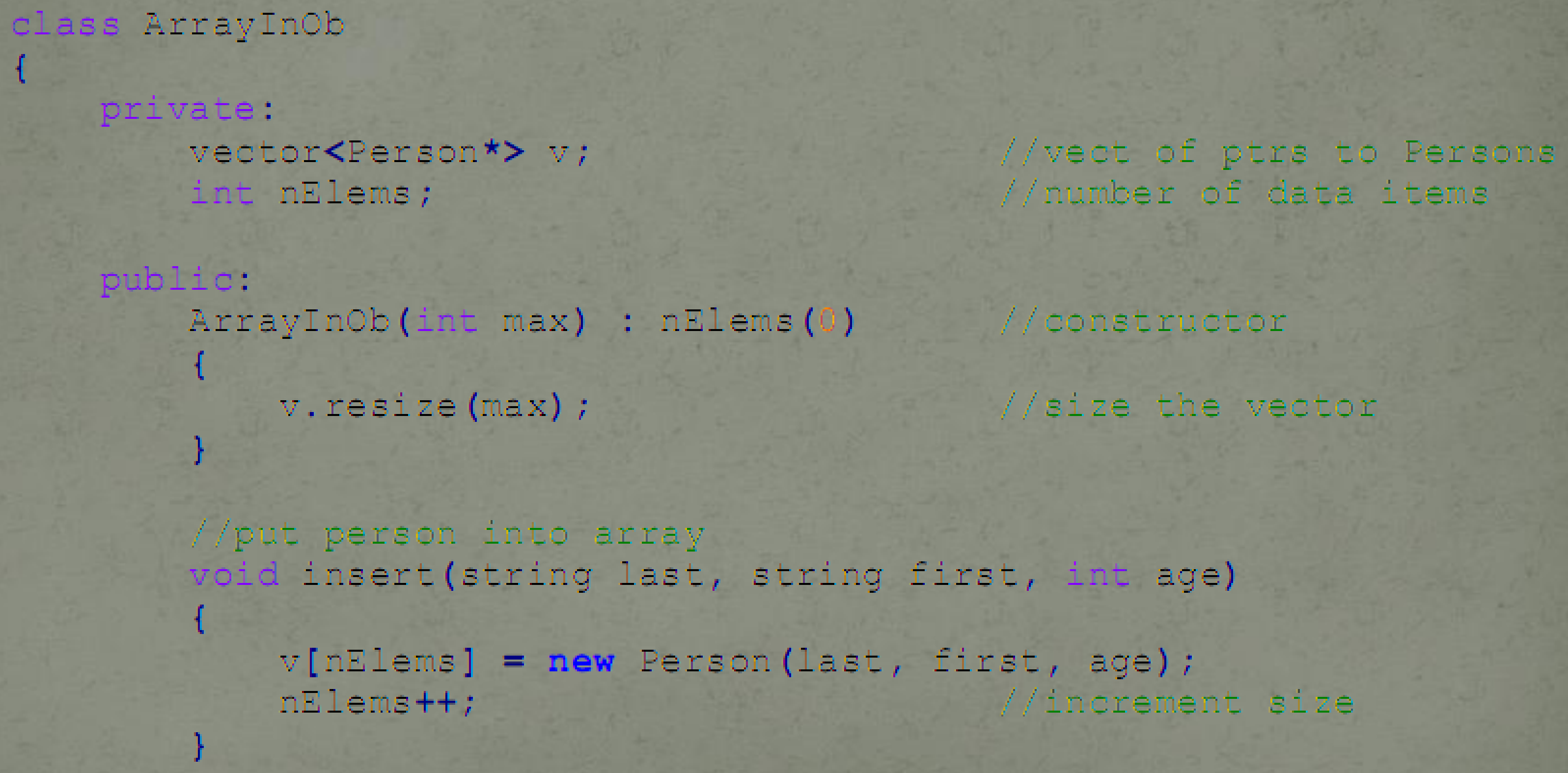
- Sorting can be applied to the objects not only for a primitive data type like: double, int, etc.



```
class Person
{
    private:
        string lastName;
        string firstName;
        int age;

    public:
        Person(string last, string first, int a) :    //constructor
            lastName(last), firstName(first), age(a)
        {   }

        void displayPerson()
        {
            cout << "   Last name: " << lastName;
            cout << ", First name: " << firstName;
            cout << ", Age: " << age << endl;
        }

        string getLast()                                //get last name
        { return lastName; }
```

Sorting Objects

- Sorting can be applied to the objects not only for a primitive data type like: double, int, etc.



```
class ArrayInOb
{
    private:
        vector<Person*> v;                          //vect of ptrs to Persons
        int nElems;                                 //number of data items

    public:
        ArrayInOb(int max) : nElems(0)              //constructor
        {
            v.resize(max);                          //size the vector
        }

        //put person into array
        void insert(string last, string first, int age)
        {
            v[nElems] = new Person(last, first, age);
            nElems++;                               //increment size
        }
```

Invariants and Stability

- Invariants – „are conditions that remaind unchanged as the algorithm proceeds” – R. Lafore.

  In bubble Sort algorithm we can find invariant:

  after first loop data N-1 is sorted,

  after second loop data N-2 and N-1 are sorted,

  …

- Stability – means that algorithm does not change the order of data (e.g.: sorted by name and second time by zip code – for the same zip code should get also data sorted by name. That kind algorithm is *stable*)

  Bubble Sort and Insertion Sort - stable sorting algorithms.

Polish flag

```
procedure sort (n:integer; var a:array);
var x,l,p: integer;
p:=n;
l:=1;
while (l<p) {
        while (a[l]==0) {
                if (l<p)    l:=l+1;

                while (a[p]==1) {
                        if (l<p)    p:=p-1;
                }
        }
        x:=a[p];
        a[p]:=a[l];
        a[l]:=x;
        l:=l+1;
        p:=p-1;
}
```

http://mastalerz.it/algorytmy/