

October 2013

## Bubble and Selection Sorting with Excel

Jan Benacka

*Department of Informatics, Faculty of Natural Sciences, Constantine the Philosopher University, Nitra, Slovakia,*  
jbenacka@ukf.sk

Follow this and additional works at: <http://epublications.bond.edu.au/ejsie>



This work is licensed under a [Creative Commons Attribution-Noncommercial-No Derivative Works 4.0 License](https://creativecommons.org/licenses/by-nc-nd/4.0/).

---

### Recommended Citation

Benacka, Jan (2013) Bubble and Selection Sorting with Excel, *Spreadsheets in Education (eJSiE)*: Vol. 6: Iss. 3, Article 4.  
Available at: <http://epublications.bond.edu.au/ejsie/vol6/iss3/4>

This Regular Article is brought to you by the Bond Business School at [epublications@bond](mailto:epublications@bond.edu.au). It has been accepted for inclusion in *Spreadsheets in Education (eJSiE)* by an authorized administrator of [epublications@bond](mailto:epublications@bond.edu.au). For more information, please contact [Bond University's Repository Coordinator](#).

---

# Bubble and Selection Sorting with Excel

## Abstract

The paper gives a method of learning programming the bubble and selection sorting algorithms through developing interactive applications in Excel and coding the just acquired step of the solution in Delphi Pascal. The method was tried in nine 90 minute lessons with 83 participants, which were gymnasium students, undergraduate, master and PhD students of Teaching Informatics, and university Informatics teachers. The participants were given questionnaires to find out their opinion on the method. The results are discussed.

## Keywords

Bubblesort, Selectsort, spreadsheets, Pascal, Delphi, constructivism

## Distribution License



This work is licensed under a [Creative Commons Attribution-Noncommercial-No Derivative Works 4.0 License](https://creativecommons.org/licenses/by-nc-nd/4.0/).

## Cover Page Footnote

The author is a member of the research team of project PRIMAS (Promoting Inquiry in Mathematics and Science Education across Europe) funded by the EU 7th Framework Programme, grant agreement 244380. The author thanks Ing. Mgr. Alžbeta Danielová and Mr. Milan Holota, the headmasters of Gymnázium in Šurany and Gymnázium in Nové Zámky, for their kind permission to carry out the survey, and Mgr. Alexander Meleg, the head of School Committee for Informatics at Gymnázium in Nové Zámky, for his helpfulness.

# Bubble and Selection Sorting with Excel

## Abstract

The paper gives a method of learning programming the bubble and selection sorting algorithms through developing interactive applications in Excel and coding the just acquired step of the solution in Delphi Pascal. The method was tried in nine 90 minute lessons with 83 participants, which were gymnasium students, undergraduate, master and PhD students of Teaching Informatics, and university Informatics teachers. The participants were given questionnaires to find out their opinion on the method. The results are discussed.

**Keywords:** Bubblesort, Selectsort, spreadsheets, Pascal, Delphi, constructivism

## 1. Introduction

Research in computer science education is focused on aiding and improving teaching and learning programming especially to novices. Since the nineties, there has been significant research into the application of constructivism in computer science education [1] – [4]. A tool that enables easy access to ideas and concepts in computer science in a constructivist way is the spreadsheet [5] – [9].

Educators discovered the educational potential of spreadsheets in mathematics and sciences in the early 1980s. Spreadsheets allow using problem-solving and heuristic methods. Only applications developed in special programming environments offer such ability of analyzing scientific problems. Students are familiar with using spreadsheets so the educators can focus on tutoring and scaffolding the learning process. Baker and Sugden [10] gave a detailed account on spreadsheets in education from 1979 till 2003. They prove on a wide range of research papers about spreadsheets in teaching mathematics, physics and computer science that *“... there is no longer a need to question the potential for spreadsheets to enhance the quality and experience of learning that is offered to students. Traditional barriers (...) need to be removed, either by ensuring that access to computers is improved or by changing assessment methods. Further expansion is needed of the types of topics that can be effectively covered by spreadsheet examples ...”* However, *“Despite the relevance of spreadsheets to today’s workplace, research in computer science education has neglected these versatile modelling tools”* [11].

This article gives a method of learning programming the Bubblesort and Selectsort sorting algorithms through developing interactive applications in Excel and rewriting the just acquired step of the solution in Pascal in Delphi, which is a standard environment for teaching programming at upper secondary schools in Slovakia [12]. The algorithms are an important part of learning programming. As Taherkhani, Korhonen and Malmi’s research showed, 49% of freshmen students would write one of these programs to sort an array of integers before given instruction on sorting (22% for Bubblesort and 27% for Selectsort) [13]. The method was tried in nine 90 minute lessons with 83 participants, which were gymnasium students of age 18-19, undergraduate, master and PhD students of Teaching

Informatics, and university teachers of Informatics. Questionnaires were given to the participants in order to find out whether they found the method interesting, understood everything and gained new knowledge and skills. The Excel and Delphi applications are in sections 2 and 3. The questionnaires and results are in section 4. Conclusions are in section 5.

## 2. Bubblesort

### 2.1. Excel and Delphi implementations

In Bubblesort, the maximum (or minimum) of the unsorted section of the array "bubbles" to the end (or beginning) of the section. This is executed through comparing the adjacent terms and swapping them if necessary [14]. The Excel application is in Fig. 1.

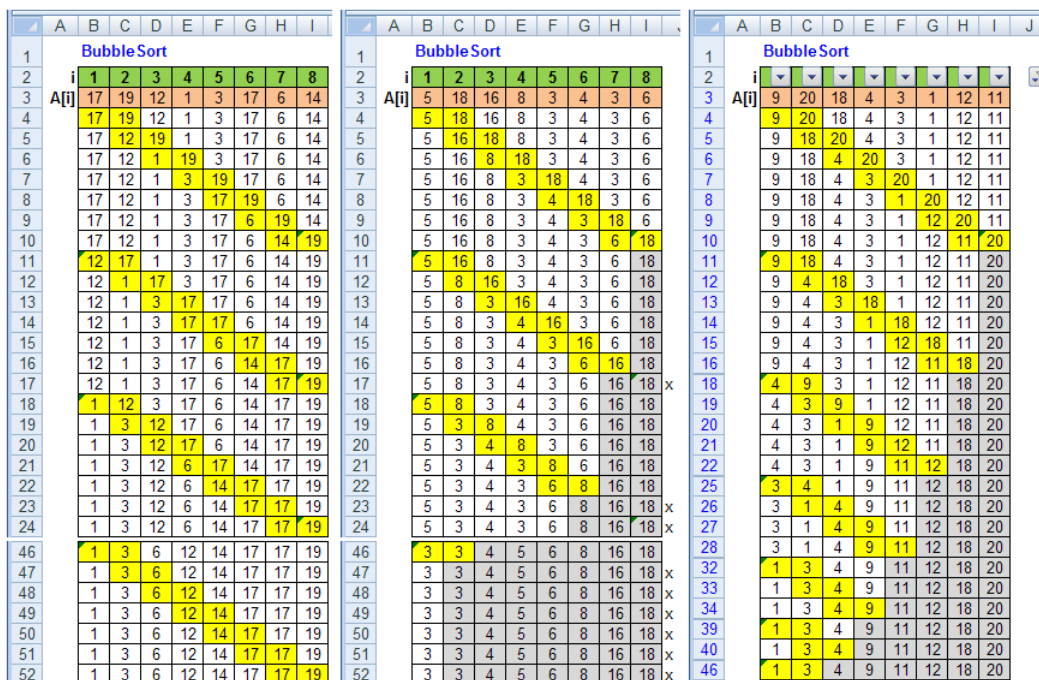


Figure 1. Bubblesort; the simple version (left, middle) and the improved version (right)

The array is in range B3:I3. The indices of the terms are in range B2:I2. Cells B3:I3 contain the function =RANDBETWEEN(1,20) that generates random numbers from 1 to 20. The first round of bubbling is in range B4:I10. The formula =B3 is written in cell B4 and filled right as far as column I. Range B4:I4 is filled down as far as row 10, which produces 7 copies of the generated array. The formulas =IF(B3>C3,C3,B3) and =IF(B3>C3,B3,C3) are written in cells B4 and C4 (yellow ones). The formulas swap the two terms if the former is bigger than the latter. Range B4:C4 is copied and pasted in cells C5, D6, E7, F8, G9 and H10, which makes the biggest term to “bubble” into cell I10. The first round is copied and pasted six times in cells B11, B18, B25, B32, B39, B46, which gives seven rounds of bubbling altogether. The sorted array is in range B52:I52. The first, second, third and seventh round are shown in Fig. 1 left. In Fig. 1 middle, the cells with the sorted terms are coloured in grey, and the redundant rows are labelled with “x”. In Fig. 1 right, the redundant rows are filtered away.

The Pascal code that corresponds to the application in Fig. 1 left is in Fig. 2. The lines are numbered in the succession as they were coded (see section 2.2). The interface is in Fig. 3 left.

```

bubblesort.pas
bubblesort
{ 0a} procedure TForm1.Button1Click(Sender: TObject);
{ 1} var i,j,Help:integer;
{ 2} A:array[1..8] of integer;
{ 0b} begin
{ 3} Randomize;
{ 4} Memo1.Clear;
{ 5} Memo2.Clear;

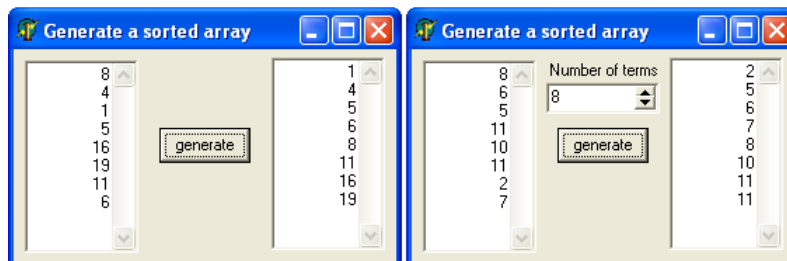
{ 6a} for i:=1 to 8 do //generating the array
{ 6b} begin
{ 7} A[i]:=Random(20)+1;
{ 8} Memo1.Lines.Add(IntToStr(A[i]));
{ 6c} end;

{10} for j:=1 to 7 do //bubblesort
{11} for i:=1 to 7] do
{12} if A[i]>A[i+1] then begin Help:=A[i]; A[i]:=A[i+1]; A[i+1]:=Help end;

{ 9} for i:=1 to 8 do Memo2.Lines.Add(IntToStr(A[i])); //output
{ 0c} end;

```

Figure 2. Delphi Pascal code for generating and bubblesorting 8 term arrays

Figure 3. Delphi application for generating and sorting 8 term arrays (left) and  $N$  term arrays (right)

The code that corresponds to the Excel application in Fig. 1 right is in Fig. 4.

```

bubblesort.pas
bubblesort
{ 6c} end;

{10} for j:=1 to 7 do //bubblesort
{11} for i:=1 to 8-j] do
{12} if A[i]>A[i+1] then begin Help:=A[i]; A[i]:=A[i+1]; A[i+1]:=Help end;

{ 9} for i:=1 to 8 do Memo2.Lines.Add(IntToStr(A[i])); //output

```

Figure 4. Improved Delphi Pascal code for generating and bubblesorting 8 term arrays

The interface of the general version is in Fig. 3 right. The code is in Fig. 5. Number  $N$  of the terms is inputted through a SpinEdit component. Its properties Max, Min and Value are set to 2, 100 and 8.

```

bubblesort.pas
bubblesort
{ 0a} procedure TForm1.Button1Click(Sender: TObject);
{ 1} var i,j,Help,N:integer;
{ 2} A:array[1..100]of integer;
{ 0b} begin
{ 3} Randomize;
{ 4} Memo1.Clear;
{ 5} Memo2.Clear;

{13} N:=SpinEdit1.Value; //generating the array
{ 6a} for i:=1 to N do
{ 6b} begin
{ 7} A[i]:=Random(20)+1;
{ 8} Memo1.Lines.Add(IntToStr(A[i]));
{ 6c} end;

{10} for j:=1 to N-1 do //bubblesort
{11} for i:=1 to N-j do
{12} if A[i]>A[i+1] then begin Help:=A[i]; A[i]:=A[i+1]; A[i+1]:=Help end;

{ 9} for i:=1 to N do Memo2.Lines.Add(IntToStr(A[i])); //output
{ 0c} end;
    
```

Figure 5. Delphi Pascal code for generating and bubblesorting  $N$  term arrays,  $N \leq 100$

If counter  $j$  is taken for the number of the unsorted terms, then it drops from  $N$  to 2, and the latest term to be compared with the next one is the last but one unsorted term (see Fig. 1 right). The code is

```

bubblesort.pas
bubblesort
{ 6c} end;

{10} for j:=N downto 2 do //bubblesort
{11} for i:=1 to j-1 do
{12} if A[i]>A[i+1] then begin Help:=A[i]; A[i]:=A[i+1]; A[i+1]:=Help end;

{ 9} for i:=1 to N do Memo2.Lines.Add(IntToStr(A[i])); //output
    
```

Figure 6. Other Delphi Pascal code for generating and bubblesorting  $N$  term arrays

The program can be improved by checking whether a swap was executed in the “for  $i$ ” loop. If there was no swap then the array has been sorted and the “for  $j$ ” loop can be ended by the Break command, which implies that “repeat” loop should be used instead of the “for” one.

### 2.2. The teaching method

The students were supposed to be able to work with Excel and, in Delphi, to create the interface, generate an array, output it in component Memo, and swap the values of two variables. The applications in Fig. 1 left and Fig. 3 left were shown and demonstrated. The students downloaded the Excel template shown in Fig. 7 and the empty application shown in Fig. 3 left (no written code, just the automatically generated one) from the author’s web site, and the two applications were developed step by step as follows:

	A	B	C	D	E	F	G	H	I
1	BubbleSort								
2	i								
3	A[i]								
4									

Figure 7. Excel template

**Excel:** Ranges A2:I2 (indices) and A3:I3 (array) were created; formula =B3 was written in cell B4 and filled right as far as column I; range B4:I4 was filled down as far as row 10 (7 copies of the array).

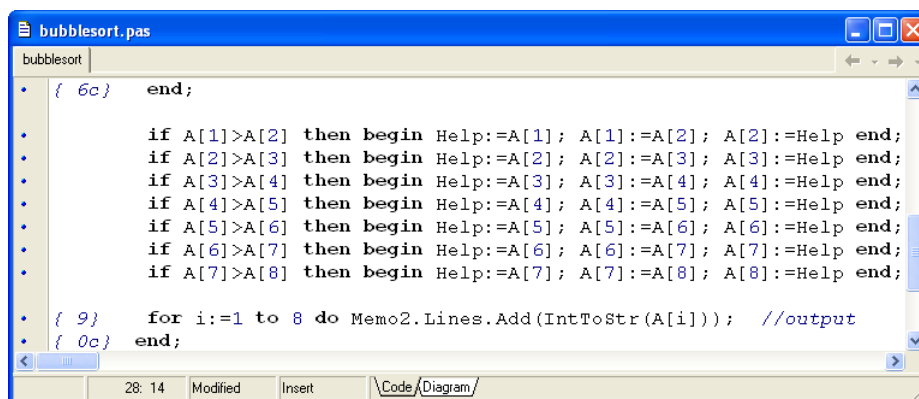
**Delphi:** Lines 0 – 9 were generated or written (unsorted array was obtained in Memo 1 and Memo 2).

**Excel:** Range B4:C4 was created (swapping A[1] and A[2]); the range was coloured in yellow, copied and pasted in cells C5, D6, E7, F8, G9, H10 (first round was accomplished, the maximum bubbled into cell I10).

**Delphi:** The command

```
if A[1]>A[2] then begin Help:=A[1]; A[1]:=A[2]; A[2]:=Help; end;
```

was written between lines 6c and 9, copied and passed 6 times; the indices were rewritten as shown in Fig. 8 (first round was accomplished, the maximum bubbled to the end of the array).



```

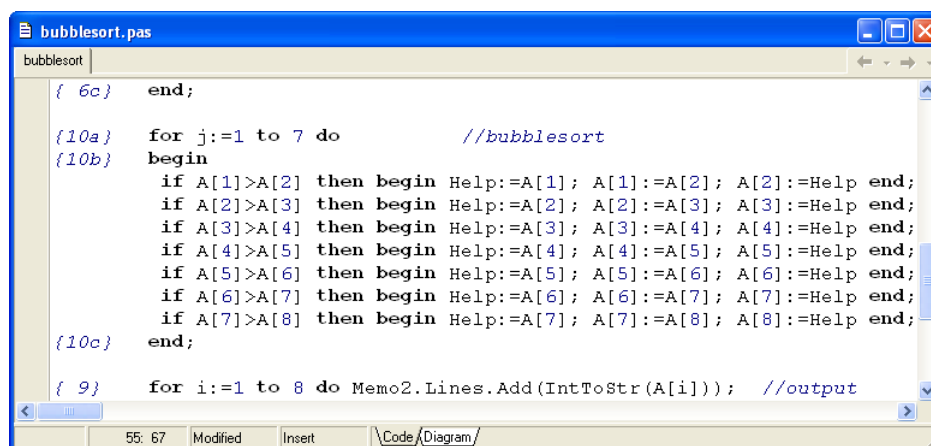
bubblesort.pas
bubblesort
• { 6c} end;
•
•     if A[1]>A[2] then begin Help:=A[1]; A[1]:=A[2]; A[2]:=Help end;
•     if A[2]>A[3] then begin Help:=A[2]; A[2]:=A[3]; A[3]:=Help end;
•     if A[3]>A[4] then begin Help:=A[3]; A[3]:=A[4]; A[4]:=Help end;
•     if A[4]>A[5] then begin Help:=A[4]; A[4]:=A[5]; A[5]:=Help end;
•     if A[5]>A[6] then begin Help:=A[5]; A[5]:=A[6]; A[6]:=Help end;
•     if A[6]>A[7] then begin Help:=A[6]; A[6]:=A[7]; A[7]:=Help end;
•     if A[7]>A[8] then begin Help:=A[7]; A[7]:=A[8]; A[8]:=Help end;
•
• { 9}   for i:=1 to 8 do Memo2.Lines.Add(IntToStr(A[i])); //output
• { 0c} end;
  
```

Figure 8. Delphi Pascal commands that correspond to Excel formulas in the first round yellow cells

**Excel:** The first round was copied and pasted six times in cells B11, B18, B25, B32, B39 and B46 (the application was accomplished, a sorted array was obtained in range B52:I52).

**Delphi:** The sequence of the seven “if” commands was copied and pasted six times to get the code that corresponds to the seven rounds in Excel (the application was accomplished, a sorted array was obtained in Memo 2; the students were warned of the “greenhorn’s construction”).

**Delphi:** The students were invited to think about how to simplify the code. It was found that just the first round would be enough if executed in a “for” loop the counter of which would be the ordinal number of the round; six rounds were erased; lines 10 were coded (Fig. 9).



```

bubblesort.pas
bubblesort
{ 6c} end;
{10a} for j:=1 to 7 do //bubblesort
{10b} begin
    if A[1]>A[2] then begin Help:=A[1]; A[1]:=A[2]; A[2]:=Help end;
    if A[2]>A[3] then begin Help:=A[2]; A[2]:=A[3]; A[3]:=Help end;
    if A[3]>A[4] then begin Help:=A[3]; A[3]:=A[4]; A[4]:=Help end;
    if A[4]>A[5] then begin Help:=A[4]; A[4]:=A[5]; A[5]:=Help end;
    if A[5]>A[6] then begin Help:=A[5]; A[5]:=A[6]; A[6]:=Help end;
    if A[6]>A[7] then begin Help:=A[6]; A[6]:=A[7]; A[7]:=Help end;
    if A[7]>A[8] then begin Help:=A[7]; A[7]:=A[8]; A[8]:=Help end;
{10c} end;
{ 9}   for i:=1 to 8 do Memo2.Lines.Add(IntToStr(A[i])); //output
  
```

Figure 9. Delphi Pascal commands that correspond to Excel formulas in all yellow cells

**Delphi:** The students were invited to think about how to further simplify the code. It was found that just one “if” commands out of the 7 ones would be enough if executed in another “for” loop the counter of which would be the ordinal number of the term that was to be compared with the next term; lines 11 and 12 were coded (Fig. 2).

**Excel:** The cells with the sorted terms were coloured in grey; the redundant rows were revealed, labelled with “x” (Fig. 1 middle) and filtered away (Fig. 1 right) (an improved version of the application was obtained with significantly reduced number of rows).

**Delphi:** Looking at the coloured pattern in Fig. 1 right, the fact was revealed that the index of the last term to be compared with the next term depended on the ordinal number  $j$  of the round; if  $j = 1$ , then it was  $i = 7$ ; if  $j = 2$ , then it was  $i = 6$ , and so on; the sum was always 8, which gave the formula  $i = 8 - j$ ; line 11 was rewritten (a much quicker version of the application was obtained; Fig. 4).

**Delphi:** The application was generalised. Component SpinEdit was added, properties Min, Max and Value were set to 2, 100 and 8. Variable  $N$  was introduced for the number of terms (Fig. 5).

**Delphi:** The students were invited to take counter  $j$  for the number of the unsorted terms, and read the white-yellow pattern in Fig. 1 right in other way. The “for  $j$ ” loop was rewritten as “N downto 2”. It was found that the ordinal number of the last unsorted term was  $j$ , so the latest term to be compared with the next one was the last but one unsorted term the index of which was  $i = j - 1$ . Lines 10 and 11 were rewritten as in Fig. 6.

**Delphi:** The fact was pointed out that it may happen that the array would be sorted as early as in the first round, e.g.; the students were invited to think at home how to find out that the array has been sorted and how to adjust the program to stop.

### 3. Selectsort

#### 3.1. Excel and Delphi implementations

In Selectsort, the maximum (or minimum) of the unsorted section of the array is found and swapped with the latest (or first) unsorted term [14]. The knowledge of finding the maximum (or minimum) of an array is essential. The application is in Fig. 10. The array is in range B3:I3. The indices are in range B2:I2. The formula =B3 is written in cell B4 and filled right as far as column I. Range B4:I4 is filled down as far as row 10, which produces 7 copies of the generated array.

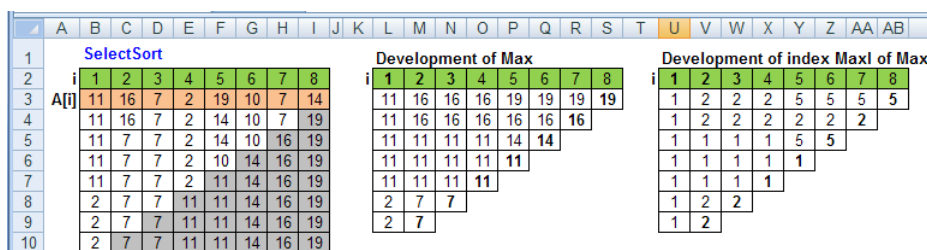


Figure 10. Selectsort in Excel

The first round is in ranges L3:S3 (development of the current maximum Max), U3:AB3 (development of the index MaxI of the current maximum Max) and B4:I4. Cells L3 and U3 contain =B3 and =B2. Cells M3 and V3 contain =IF(C3>L3,C3,L3) and =IF(C3>L3,C\$2,U3). The formulas are filled right. The formula =IF(B\$2=\$AB3,\$I3,B3) is written in cell B4. The formula is copied right as far as cell H4. The formula =S3 is written in cell I4.



The second round is in ranges L4:R4, U4:AA4 and B5:H5. Ranges L4:R4 and U4:AA4 are copies of ranges L3:R3 and U3:AA3. The formula from cell B4 is copied into cell B5, modified to =IF(B\$2=\$AA4,\$H4,B4) and filled right as far as G5. Cell H5 contains the formula =R4. The next rounds are implemented in the same way. The sorted array is in cells B10:I10.

Remark: A version with functions Max and Match is added into the attached Excel file.

The Pascal code that corresponds to the application in Fig. 10 is in Fig. 11. The lines are numbered in the succession as they were coded (see section 3.2). The interface is in Fig. 3 left.

```

selectsort.pas
selectsort
{ 0a} procedure TForm1.Button1Click(Sender: TObject);
{ 1} var i, j, Max, MaxI: integer;
{ 2}     A: array[1..8] of integer;
{ 0b} begin
{ 3}     Randomize;
{ 4}     Memo1.Clear;
{ 5}     Memo2.Clear;

{ 6a}   for i:=1 to 8 do           //generating the array
{ 6b}   begin
{ 7}     A[i]:=Random(20)+1;
{ 8}     Memo1.Lines.Add(IntToStr(A[i]));
{ 6c}   end;

{10a}   for j:=1 to 7 do         //selectsort
{10b}   begin
{11}     Max:=A[1]; MaxI:=1;
{12}     for i:=2 to 9-j do
{13}       if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
{14}     A[MaxI]:=A[9-j]; A[9-j]:=Max;
{10c}   end;

{ 9}   for i:=1 to 8 do Memo2.Lines.Add(IntToStr(A[i])); //output
{ 0c} end;

```

Figure 11. Delphi Pascal commands for generating and selectsorting 8 term arrays

The code for the general version (Fig. 3 right) is

```

selectsort.pas
selectsort
{ 0a} procedure TForm1.Button1Click(Sender: TObject);
{ 1} var i, j, N, Max, MaxI: integer;
{ 2}     A: array[1..100] of integer;
{ 0b} begin
{ 3}     Randomize;
{ 4}     Memo1.Clear;
{ 5}     Memo2.Clear;

{13}   N:=SpinEdit1.Value;       //generating the array
{ 6a}   for i:=1 to N do
{ 6b}   begin
{ 7}     A[i]:=Random(20)+1;
{ 8}     Memo1.Lines.Add(IntToStr(A[i]));
{ 6c}   end;

{10a}   for j:=1 to N-1 do       //selectsort
{10b}   begin
{11}     Max:=A[1]; MaxI:=1;
{12}     for i:=2 to N+1-j do
{13}       if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
{14}     A[MaxI]:=A[N+1-j]; A[N+1-j]:=Max;
{10c}   end;

{ 9}   for i:=1 to N do Memo2.Lines.Add(IntToStr(A[i])); //output
{ 0c} end;

```

Figure 12. Delphi Pascal commands for generating and selectsorting  $N$  term arrays

If  $j$  is taken for the number of unsorted terms in the array, then it drops from  $N$  to 2, and the position  $i$  of the latest unsorted term is  $i = j$ . The code is

```

selectsort
{10a} for j:=N downto 2 do //selectsort
{10b} begin
{11}   Max:=A[1]; MaxI:=1;
{12}   for i:=2 to j do
{13}     if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
{14}   A[MaxI]:=A[j]; A[j]:=Max;
{10c} end;
    
```

Figure 13. Other Delphi Pascal code for selectsorting  $N$  term arrays

### 3.2 The teaching method

The students were supposed to be familiar with Bubblesort (section 2) and to be able to find the maximum of an array. The applications in Fig. 10 and Fig. 3 left were shown and demonstrated. The template shown in Fig. 14 and the application shown in Fig. 3 left with lines 0 – 9 completed were downloaded by the students from the author’s web site, and the applications were developed step by step as follows:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1		SelectSort																										
2	i	1	2	3	4	5	6	7	8																			
3	A[i]	16	13	1	5	17	18	4	7																			
4																												
5																												
6																												
7																												
8																												
9																												
10																												

Figure 14. Excel template for Selectionsort

**Excel:** The formula =B3 was written in cell B4 and filled right as far as column I; range B4:I4 was filled down as far as row 10 (7 copies of the array were obtained).

**Excel:** Range L3:S3 was created (the maximum was found); range U3:AB3 was created (the index of the maximum was found)

**Delphi:** The following commands were written between lines 6c and 9 (maximum and its index were found; Fig. 15):

```

Max:=A[1]; MaxI:=1;
for i:=2 to 8 do if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
    
```

**Excel:** Range B4:I4 was created (the maximum was swapped with the last term); range I4:I10 was coloured (the sorted part was marked in grey).

**Delphi:** The following commands were written below the previous ones (the maximum was swapped with the last term; Fig. 15)

```

A[MaxI]:=A[8]; A[8]:=Max;
    
```

**Excel:** Ranges L4:R4, U4:AA4 and B5:I5 were created (the second maximum and its index were found and swapped with the last but one term).

**Delphi:** The three commands were copied, pasted below and number 8 was rewritten by 7 as follows (the second maximum and its index were found; the second maximum was swapped with the last but one term; Fig. 15).

```

Max:=A[1]; MaxI:=1;
for i:=2 to 7 do if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
A[MaxI]:=A[7]; A[7]:=Max;
    
```

The development went on by repeated pasting and rewriting (Fig.10: a sorted array was obtained in range B10:I10; Fig. 15: a sorted array was obtained in Memo 2; the Delphi code was called “a greenhorn’s construction”, again).



```

selectsort
{ 6c } end;

Max:=A[1]; MaxI:=1;
for i:=2 to 8 do if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
A[MaxI]:=A[8]; A[8]:=Max;

Max:=A[1]; MaxI:=1;
for i:=2 to 7 do if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
A[MaxI]:=A[7]; A[7]:=Max;

Max:=A[1]; MaxI:=1;
for i:=2 to 6 do if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
A[MaxI]:=A[6]; A[6]:=Max;

Max:=A[1]; MaxI:=1;
for i:=2 to 5 do if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
A[MaxI]:=A[5]; A[5]:=Max;

Max:=A[1]; MaxI:=1;
for i:=2 to 4 do if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
A[MaxI]:=A[4]; A[4]:=Max;

Max:=A[1]; MaxI:=1;
for i:=2 to 3 do if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
A[MaxI]:=A[3]; A[3]:=Max;

Max:=A[1]; MaxI:=1;
for i:=2 to 2 do if A[i]>Max then begin Max:=A[i]; MaxI:=i; end;
A[MaxI]:=A[2]; A[2]:=Max;

{ 9 } for i:=1 to 8 do Memo2.Lines.Add(IntToStr(A[i])); //output

```

Figure 15. A simple version of Selectsort for sorting 8 term arrays

**Delphi:** The students were invited to simplify the code. It was found that the upper bound for counter  $i$  in the “for” loop depends on the ordinal number of the round. Variable  $j$  was introduced for the ordinal number of the round. It was found that if  $j=1$ , then the bound was  $i=8$ . If  $j=2$ , then it was  $i=7$ , and so on. The sum of  $i$  and  $j$  was always 9, which implied the formula  $i=9-j$ . It was found that just one sequence out of the seven ones would be enough if executed in an outer “for  $j$ ” loop; lines 10 – 14 were written.

**Delphi:** The application was generalised. Component SpinEdit was added. Properties Min, Max and Value were set to 2, 100 and 8. Variable  $N$  was introduced for the number of terms. The array range in line 2 was adjusted to 1..100 (Fig. 12).

**Delphi:** The students were invited to take variable  $j$  for the number of the unsorted terms, and read the white-grey pattern in Fig. 10 in another way. The “for  $j$ ” loop was rewritten as “N downto 2”. It was found that the index of the latest unsorted term, which was to be replaced by Max, was  $i=j$ . Lines 10a, 12 and 14 were rewritten as in Fig. 13.

#### 4. Survey and results

Nine 90 minute lessons were taught to 83 participants. The groups are listed in Tab. 1. The Bubblesort applications were developed with 15 fourth year (age 18-19) gymnasium (grammar school) students of Gymnazium in Surany in Informatics lesson (group SU), 3 fourth year gymnasium students of Gymnazium in Nove Zamky within Computer modelling club that the author runs at the school (group NZ), 27 first year undergraduates of Teaching Informatics in Programming lesson (group PR), 6 first year master students of Teaching Informatics in Informatics Didactics lesson (group DI), and a group of 2 PhD students and 3 university teachers of Informatics and Teaching Informatics at a workshop within conference ISSEP 2013 (group CO). The Selectsort applications were developed with 17 fourth year gymnasium students of Gymnazium in Surany in Informatics lesson, 3 fourth year gymnasium students of Gymnazium in Nove Zamky, members of the Computer modelling club (group NZ), and 7 first year undergraduates of Teaching Informatics in Programming Seminar (group PS). At the end of the lesson, the participants were given questions with a list of answers to be chosen from. The questionnaire for groups SU, NZ, PR and SP comprised the following questions:

- A) The lesson was (1:very; 2:quite; 3: little; 4:not) interesting;
- B) I understood (1:everything; 2:majority; 3:minority; 4:nothing);
- C) I learned (1:very much; 2:quite much; 3: little; 4:nothing) new in programming;
- D) Developing the Excel application was (1:very; 2:quite; 3: little; 4:not) helpful;
- E) I am a man (1:Yes; 2:No).
- F) I had written a Bubblesort program before this one (1:Yes; 2:No).

The word Bubblesort in question F was replaced with Selectsort when the algorithm was taught. The average of the answers to questions A, B, C and D are in Tab. 1 in columns A, B, C and D. The number of answer "Yes" to questions E and F is in columns E and F.

The questionnaire for groups DI and CO comprised the following questions:

- A) The lesson was (1:very; 2:quite; 3: little; 4:not) interesting;
- D) The Excel application was (1:very; 2:quite; 3: little; 4:not) helpful;
- E) I am a man (1:Yes; 2:No).
- G) I would like to see teaching Selectsort through Excel 1) yes; 2) no;

The average of the answers to questions A and D are in Tab. 1 in columns A and D. The number of answer "Yes" to questions E and G is in columns E and G.

Group	Number	Topic	A	B	C	D	E	F	G
SU	15	Bubble	1.5	1.5	2.1	1.8	13	4	-
NZ	3	Bubble	2.0	1.3	2.0	1.3	3	1	-
PR	27	Bubble	1.6	1.6	2.0	1.6	15	6	-
DI	6	Bubble	1.7	-	-	1.2	6	-	4
CO	5	Bubble	2.0	-	-	1.0	4	-	3
SU	17	Select	2.6	2.0	2.2	2.1	13	0	-
NZ	3	Select	2.0	1.3	1.3	1.3	3	0	-
PS	7	Select	1.4	2.0	2.0	1.9	5	0	-
<b>Total</b>	<b>83</b>		<b>1.7</b>	<b>1.7</b>	<b>2.0</b>	<b>1.7</b>	<b>62</b>	<b>11</b>	<b>7</b>

Table 1: Result of the questionnaires; A – D: average answer; E – G: number of "yes"

The average of the answers of the gymnasium students (groups SU and NZ, twice each) is 1.8 (A), 1.7 (B), 2.1 (C) and 1.9 (D) at the standard deviation of 0.49, 0.68, 0.56 and 0.61, respectively. The relative frequencies are graphed in Fig. 16.

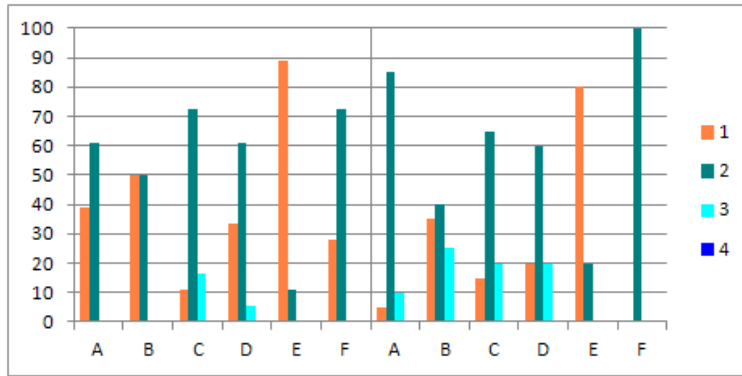


Figure 16: Relative frequency of the answers of the gymnasium students in % for Bubblesort (left) and Selectsort (right).

There was no answer of 4. The result is that: (A) 100% found the Bubblesort lessons very or quite interesting; it was 90% for Selectsort; (B) 100% understood everything or the majority for Bubblesort; it was 75% for Selectsort; (C) 83% learned very or quite much new in Delphi programming for Bubblesort; it was 80% for Selectsort. (D) 94% found developing the Excel application very or quite helpful for Bubblesort; it was 80% for Selectsort; (E) 89% were men and 11% were women for Bubblesort, and 80% were men and 20% for Selectsort; (F) 28% had written a Bubblesort program before and it was 0% for Selectsort.

The average of the answers of the undergraduates (group PR twice and PS) is 1.6 (A), 1.6 (B), 2.0 (C) and 1.6 (D) at the standard deviation of 0.54, 0.59, 0.62, 0.69. The relative frequencies are graphed in Fig. 17. There was no answer of 4.

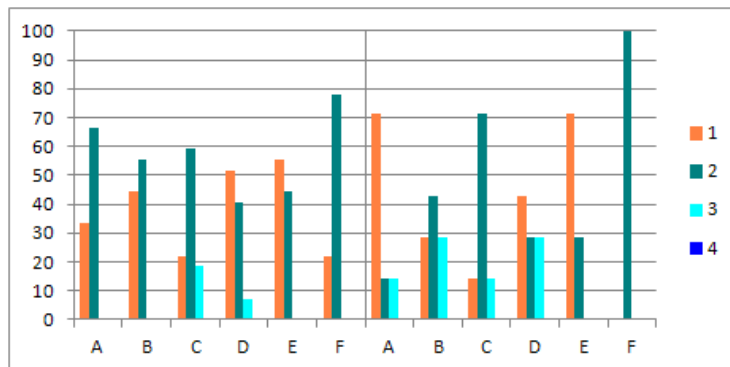


Figure 17: Relative frequency of the answers of the undergraduates in % for Bubblesort (left) and Selectsort (right)

The result is that: (A) 100% found the Bubblesort lessons very or quite interesting and it was 85% for Selectsort; (B) 100% understood everything or the majority for Bubblesort and it was 71% for Selectsort; (C) 81% learned very or quite much new in programming for Bubblesort and it was 86% for Selectsort; (D) 93% found developing the Excel application very or quite helpful for Bubblesort and it was 71% for Selectsort; (E) 56% were man and 44% were women for Bubblesort and it was 71% and 29% for Selectsort; (F) 22% had written a Bubblesort program before the lesson and it was 0% for Selectsort.

The average of the answers of the master students, PhD students and university teachers is 1.8 (A) and 1.1 (D) at the standard deviation of 0.39 and 0.48. There was no answer of 3 or 4 in questions A and D. The relative frequencies are graphed in Fig. 18.

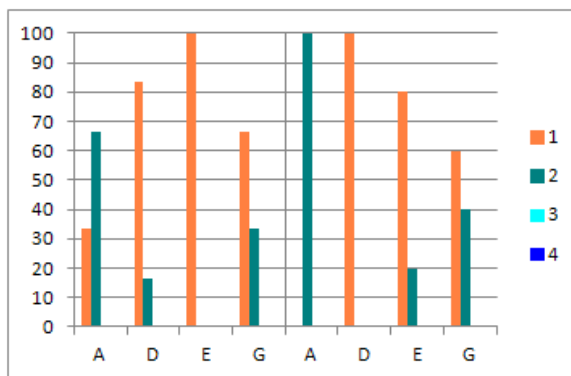


Figure 18: Relative frequency of the answers of the master students (left), and PhD students and university teachers (right) in % for Bubblesort

The result is that: (A) 100% found the lesson very or quite interesting; (D) 100% found developing the Excel application very or quite helpful; (E) 91% were men and 9% were women; (G) 64% would like to see teaching Selectsort through Excel.

The relative frequencies of the answers to questions A, D and E of all participants are graphed in Fig. 19. The result is that: (A) 29% found the lessons very interesting, 67% found them quite interesting and 4% little interesting; (D) 45% found developing the Excel applications very helpful, 45% quite helpful and 10% little helpful; (E) 75% were men.

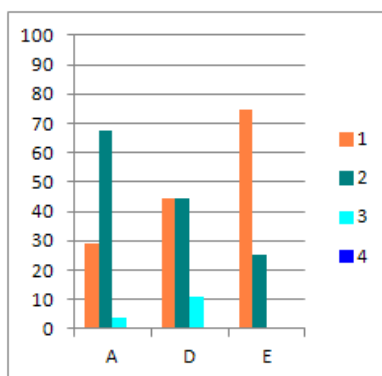


Figure 19: Relative frequency of the answers of all participants in %

### 5. Conclusions

A method of learning programming the bubble and selection sorting algorithms was presented in the paper. The method is based on the idea of simultaneous developing interactive applications in Excel and rewriting the just acquired step of the solution in Delphi Pascal. Nine 90 minute lessons were taught to 83 participants from whom 25% were women. The participants were fourth year gymnasium (grammar school) students (age 18-19), first year undergraduates and first year master students of Teaching Informatics, and PhD students and university teachers of Informatics and Teaching Informatics. Questionnaires were given to the participants after the lessons to find out their opinion of the method. The main outcome was that 96% of the participants found the lessons interesting (29% very and 67% quite) and 90% of the participants found developing the Excel applications helpful (45% very and 45% quite). Relying on the result, it can be concluded that the simultaneous developing interactive spreadsheets and writing the code in a programming language is an interesting and lucid way of programming the two algorithms. Other algorithm that is suitable for the teaching approach is insertion sorting.

## References

- [1] Ben-Ari, M. (1998). Constructivism in computer science education. *ACM SIGCSE Bulletin*, 30(1), 257–261.
- [2] Hadjerrouit, S. (1999). A constructivist approach to object-oriented design and programming. *ACM SIGCSE Bulletin*, 31(3), 171–174.
- [3] Van Gorp, M.J., Grissom, S. (2001). An empirical evaluation of using constructive classroom activities to teach introductory programming. *Computer Science Education*, 11, 247–260.
- [4] Ma, L., Ferguson, J., Roper M., Wood, M. (2011). Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education*, 21(1), 57 – 80.
- [5] Dybdahl, A., Sutinen, E., Tarhio, J. (1998). On animation features of Excel. *ITiCSE '98 Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education: Changing the delivery of computer science education*. *ACM SIGCSE Bulletin*, 30(3), 77-80.
- [6] Fone, W. (2001). Using a familiar package to demonstrate a difficult concept: Using an Excel spreadsheet to explain concepts of neural networks to undergraduates. *ITiCSE '01 Proceedings of the 6th annual conference on Innovation and technology in computer science education*, *ACM SIGCSE Bulletin*, 33(3), 165-168.
- [7] Lovászová, G., Hvorecký, J. (2003). On programming and spreadsheet calculations. *Spreadsheets in Education*, 1(1), article 3.  
<http://epublications.bond.edu.au/ejsie/vol1/iss1/3>
- [8] Benacka, J. (2008). 3D Graphics with spreadsheets. *Spreadsheets in Education*, 3(1).  
<http://epublications.bond.edu.au/ejsie/vol3/iss1/7>.
- [9] Benacka, J. (2012). Central projection in Excel – an introduction to virtual reality. *Spreadsheets in Education*, 6(1). <http://epublications.bond.edu.au/ejsie/vol6/iss1/3>.
- [10] Baker, J.; Sugden, S. (2003). Spreadsheets in education – the first 25 years. In: *Spreadsheets in Education*, 1 (1), article 2. <http://epublications.bond.edu.au/ejsie/vol1/iss1/2>
- [11] Kadijevich, D. (2013). Learning about spreadsheets. In: Kadijevich, D.M., Angeli, C., Schulte, C. (Eds.), *Improving Computer Science Education*. Routledge, New York, 19–33.
- [12] Blaho, A. (2006). *Informatika pre stredné školy. Programovanie v Delphi (Eng.: Informatics for secondary schools. Programming in Delphi)*. SPN - Mladé letá, Bratislava.
- [13] Taherkhani, A., Korhonen, A., Malmi, L. (2012). Categorizing variations of student-implemented sorting algorithms, *Computer Science Education*, 22(2), 109-138.
- [14] Knuth, D. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed. Addison-Wesley, p. 107, 139.