

CĂȘET DE PRACTICĂ

TEMA: ALGORITMI DE SORTARE

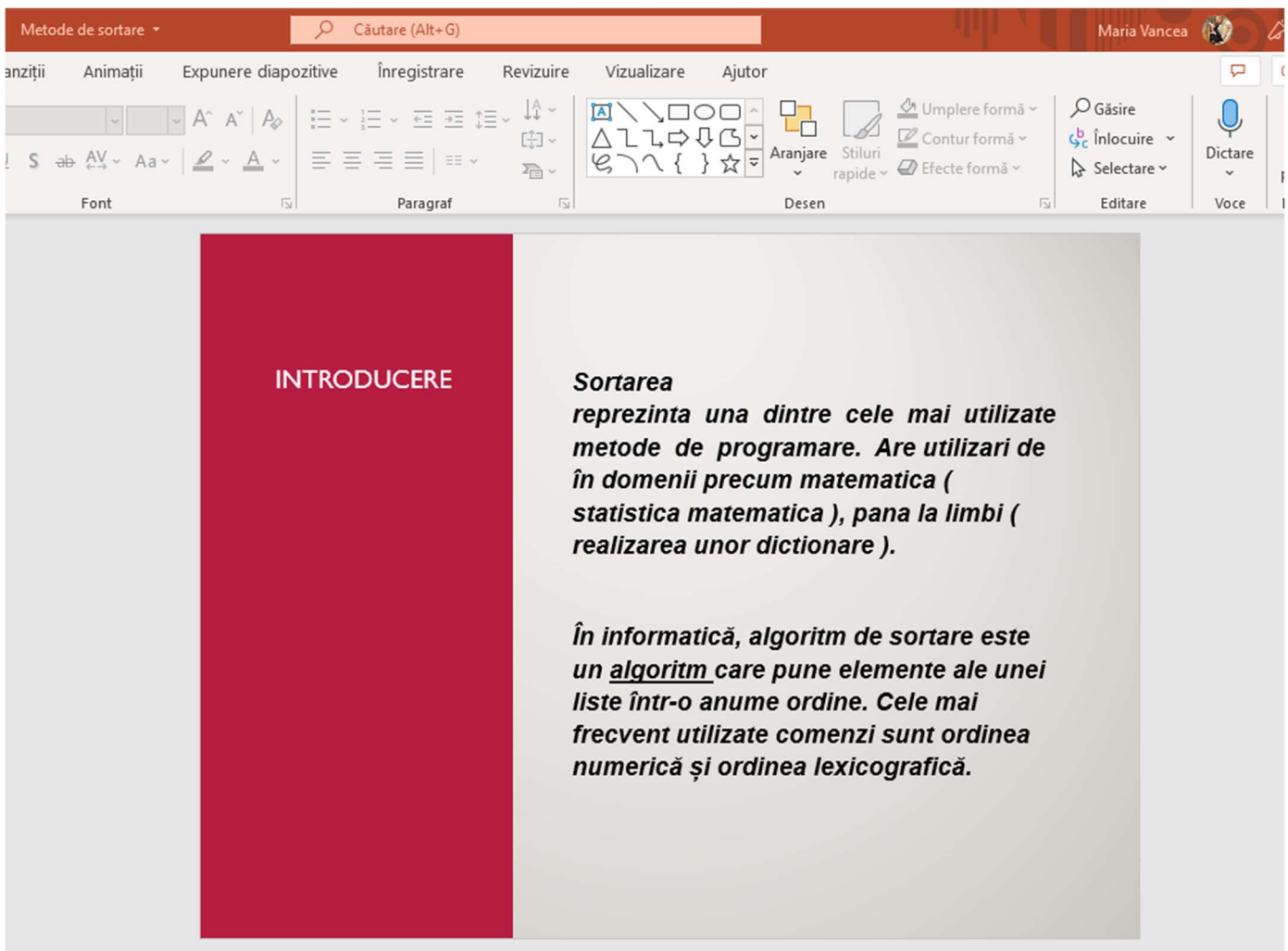
Specializarea: matematica-informatică/informatică

Echipa formată din: Vancea Maria și Mărcuș Denis

PREZENTAREA I :

În cadrul primei prezentări am realizat următoarele:

- ♣ Am făcut o scurtă introducere cu ceea ce reprezintă un algoritm de sortare



- ♣ Am clasificat metodele de sortare, acestea fiind împărțite în două categorii: directe și avansate.

The image is a screenshot of a presentation slide from a software application. The slide has a red vertical bar on the left side with the word "CLASIFICARE" in white capital letters. The main content area is light gray and contains the following text:

Metodele de sortare se clasifica in:

- ☐ **metode directe**
 - se bazeaza,, pe algoritmi de dificultate redusa

Exemplu: SelectSort, InsertionSort, BubbleSort, CombSort
- ☐ **metode avansate**
 - se bazeaza pe algoritmi putin mai complicati

Exemplu: MergeSort, QuickSort, TreeSort ,Shell Sort, HeapSort

The presentation software interface is visible at the top, showing a menu bar with options like "Animații", "Expunere diapozitive", "Înregistrare", "Revizuire", "Vizualizare", and "Ajutor". Below the menu bar is a ribbon with various icons for text, paragraph, and drawing formatting. The top right corner shows the user's name "Maria Vances" and a search bar.

- ♣ La fiecare metodă am dat câteva exemple de algoritmi, cărora le-am descris, pe scurt, principiul și modul de implementare.

5

METODE DIRECTE

❖ SelectSort

- Selecția directă este una dintre cele mai simple metode de sortare deoarece fiecare element este mutat cel mult o dată.
- Algoritmul presupune ca la fiecare pas "i" să se găsească elementul minim dintre $a[i+1]$, $a[i+2]$... $a[n]$ și se interșchimbe cu $a[i]$.

METODE DIRECTE

❖ InsertionSort

- Considerăm elementele $a[1]$... $a[i-1]$ ca fiind sortate și inserăm $a[i]$ în locul ce îi revine.
- Elementele $a[1]$... $a[i]$ sunt sortate prin inserarea lui $a[i]$ între lista elementelor sortate $a[1]$... $a[i-1]$.
- Vectorul este sortat complet când indexul ajunge la capatul drept al vectorului.

❖ BubbleSort

- Această metodă se rezumă la a compara fiecare element cu celelalte, făcându-se interșchimbarea dacă elementul mai mare are indexul mai mic. Este cea mai simplă metodă de sortare și nu necesită cunoștințe detaliate a limbajului de programare.

7

METODE AVANSATE

❖ QuickSort

- Algoritmul este de tip divide et impera și sortează o secvență a tabloului (inițial întreg tabloul), astfel:
- se alege un element special al listei, numit **pivot**;
- se ordonează elementele listei, astfel încât toate elementele din stânga pivotului să fie mai mici sau egale cu acesta, și toate elementele din dreapta pivotului să fie mai mari sau egale cu acesta;
- se continuă recursiv cu secvența din stânga pivotului și cu cea din dreapta lui.

❖ MergeSort

- **Sortarea prin Interclasare**, sau **MergeSort** este o metodă eficientă de sortare a elementelor unui tablou, bazată pe următoarea idee:
- ✓ dacă prima jumătate a tabloului are elementele sortate și a doua jumătate are de asemenea elementele sortate, prin interclasare se va obține tabloul sortat.

8

METODE AVANSATE

❖ TreeSort

- Tree Sort este un algoritm de sortare care construiește un arbore de căutare binară de la elementele de intrare care urmează să fie sortate, și apoi trece copac, în ordine, astfel încât elementele să iasă în ordine sortată. Pași:
- Preia elementele de intrare într-o matrice
- Creează un arbore de căutare binară prin inserarea elementelor de date din matrice în copac
- Efectuează în ordine traversarea pe copac pentru a obține elementele în ordine sortată

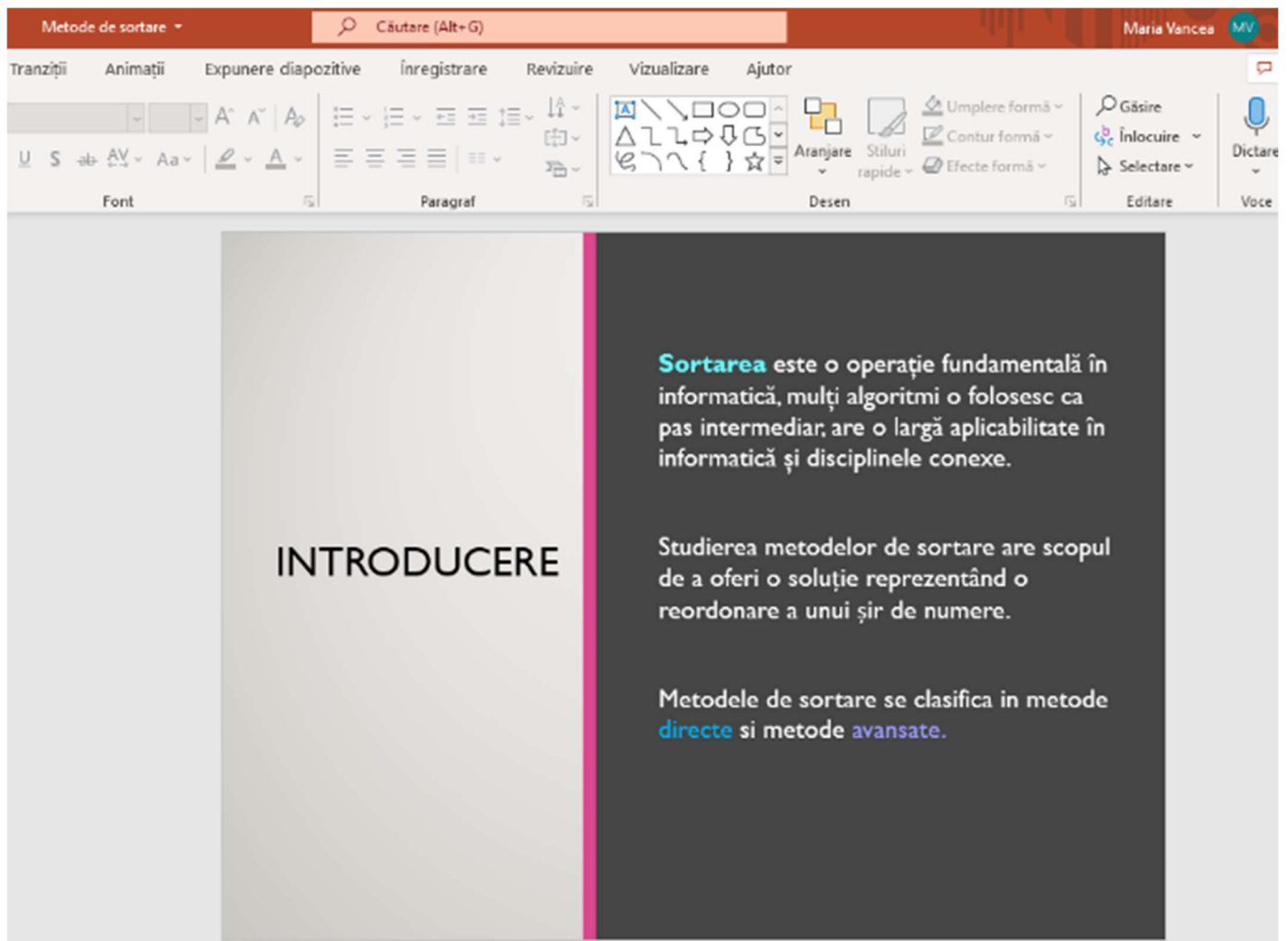
❖ Shell Sort

- Sort Shell este o versiune generalizată a algoritmului de sortare a inserării. Acesta sortează mai întâi elementele care sunt departe unul de celălalt și reduce succesiv intervalul dintre elementele care urmează să fie sortate. Intervalul dintre elemente este redus pe baza secvenței utilizate.

PREZENTAREA II:

În cadrul celei de-a doua prezentări am realizat următoarele:

- ♣ Am făcut din nou o scurtă introducere despre reprezentarea unui algoritm de sortare



- ♣ Am realizat o prezentare generală a 3 algoritmi de sortare, pe care i-am folosit în rezolvarea unor probleme

PREZENTAREA GENERALĂ A ALGORITMILOR UTILIZAȚI

SELECTSORT, **INSERTIONSORT** și **BUBBLESORT** fac parte din categoria metodelor de sortare directe, fiind *algoritmi de dificultate redusă, ușor de găsit și de înțeles*.

SELECTSORT - Sortarea prin selecție este un algoritm de sortare care selectează cel mai mic element dintr-o listă nesortată în fiecare iterație și plasează acel element la începutul listei nesortate.

INSERTIONSORT - Sortarea inserției este un algoritm de sortare care plasează un element nesortat la locul potrivit în fiecare iterație.

BUBBLESORT - Sortarea cu bule este un algoritm de sortare care compară fiecare element cu celelalte, făcându-se interschimbarea dacă elementul mai mare are indexul mai mic.

- ♣ Cele trei probleme rezolvate și prezentate:

1. Cerință: Se dă un vector cu n elemente, numere naturale și un număr k . Ordonați crescător primele k elemente ale vectorului și descrescător elementele rămase. (SELECT SORT)
2. Cerință: Se citește un vector cu n elemente, numere naturale. Ordonați crescător elementele care sunt pătrate perfecte. (INSERTION SORT)
3. Cerință: Se citește un vector cu n elemente. Ordonați descrescător elementele acestuia. (BUBBLE SORT)

PROBLEMA 1

Salvare automată Metode de sortare Căutare (Alt+G) Maria Vancea

Fișier Pornire Inserare Desenare Proiectare Tranzii Animații Expunere diapozitive Înregistrare Revizuire Vizualizare Ajutor

Lipire Diapozitiv nou Reutilizati diapozitive Reinițializare Aspect Reinițializare Secțiune

Clipboard Font Paragraf Desen Editare

Umplere formă Contur formă Efecte formă

Găsiți Înlocuire Selectare

Dictare Voce

Idee de proiectare

14

PROBLEMA I (SELECT SORT)

Pas I (exemplificare):

15

PROBLEMA I

Pas II:

16

PROBLEMA I

Pas II (exemplificare):

Faceți clic pentru a adăuga note

PROBLEMA I (SELECT SORT)

Cerință:

Se dă un vector cu n elemente, numere naturale și un număr k . Ordonează crescător primele k elemente ale vectorului și descrescător elementele rămase.

Pas I:

```
n = int(input())
k = int(input())
lista = list(map(int, input().strip().split()))[:n]
# impartim lista initiala in doua liste
crescator = [] # e in prima vom adauga primele k elemente
for i in range(k):
    crescator.append(lista[i])
print("Lista cu primele k = ", k, "numere este:", crescator)
descrescator = [] # e in a doua parte, adica n-k elemente
for i in range(k, n):
    descrescator.append(lista[i])
print("Lista cu elementele ramase este:", descrescator)
```

PROBLEMA 2

Salvare automată Metode de sortare Căutare (Alt+G) Maria Vancea

Fișier Pornire Inserare Desenare Proiectare Tranzii Animații Expunere diapozitive Înregistrare Revizuire Vizualizare Ajutor

Lipire Diapozitiv nou Reutilizati diapozitive Reinițializare Aspect Reinițializare Secțiune

Clipboard Font Paragraf Desen Editare

Umplere formă Contur formă Efecte formă

Găsiți Înlocuire Selectare

Dictare Voce

19

PROBLEMA II (INSERTIONSORT)

Cerință: Se citește un vector cu n elemente, numere naturale. Ordonează crescător elementele care sunt pătrate perfecte.

Pas I:

```
if __name__ == "__main__":
    list_elemente = []
    list_patrata_perfecte = []
    fisier = open("lista_elemente.txt", "r")
    numar_elemente = fisier.readline()
    n = int(numar_elemente)
    linie = fisier.readline().strip().split(' ')
    for i in range(len(linie)):
        list_elemente.append(int(linie[i]))
    fisier.close()

    for k in range(len(list_elemente)):
        if math.sqrt(list_elemente[k]) == int(math.sqrt(list_elemente[k])):
            list_patrata_perfecte.append(list_elemente[k])

    insertionSort(list_patrata_perfecte)
```

20

PROBLEMA II

Pasul II:

Faceți clic pentru a adăuga note

PROBLEMA II (INSERTIONSORT)

Cerință: Se citește un vector cu n elemente, numere naturale. Ordonează crescător elementele care sunt pătrate perfecte.

Pas I:

PROBLEMA 3

PROBLEMA III (BUBBLESORT)

Cerință: Se citește un vector cu n elemente. Ordonezi descrescător elementele acestuia.

```

1  bubbleSort(list_elemente)
2  n = len(list_elemente)
3  has_swapped = True
4  while(has_swapped):
5      has_swapped = False
6      for i in range(n-1):
7          if list_elemente[i] < list_elemente[i + 1]:
8              list_elemente[i], list_elemente[i + 1] = list_elemente[i + 1], list_elemente[i]
9              has_swapped = True
10
11 if __name__ == "__main__":
12     list_elemente = []
13     fisier = open("lista_elemente.txt", "r")
14     numar_elemente = fisier.readline()
15     n = int(numar_elemente)
16     linie = fisier.readline().strip().split(' ')
17     for i in range(len(linie)):
18         list_elemente.append(int(linie[i]))
19     fisier.close()
20
21     bubbleSort(list_elemente)
22
23     scriere_fisier = open("lista_fisier_out.txt", "w")
24     for j in range(len(list_elemente)):
25         scriere_fisier.write(str(list_elemente[j]))
26         scriere_fisier.write(' ')
27     scriere_fisier.close()
    
```

- ♣ *La final am expus o scurtă comparație cu avantaje si dezavantaje a celor 3 algoritmi folosiți*

COMPARAREA METODELOR

METODA DE SORTARE	AVANTAJE	DEZAVANTAJE
SELECTSORT	Sortare pe loc Numărul de interschimbări este redus, fiind $O(N^2)$ în toate cazurile.	Complexitatea timpului în toate cazurile este $O(N^2)$, nu există cel mai bun scenariu.
INSERTIONSORT	Poate fi ușor de calculat. Numărul de interschimbări este mai redus decât la sortarea cu bule.	Este utilizat în general atunci când valoarea lui N este mică. Pentru valori mai mari ale lui N , este inefficient.
BUBBLESORT	Este cea mai simplă abordare de sortare. Folosind o abordare optimizată, poate detecta o matrice deja sortată la prima trecere cu complexitatea de timp a $O(N)$	Sortarea cu bule este un algoritm mai lent față de insertionsort.

PREZENTAREA III:

În cadrul celei de-a treia prezentări am realizat următoarele:

- ♣ Folosind modelul dat de domnul profesor am exemplificat pas cu pas (step-by-step) cum funcționează 2 algoritmi de sortare și anume: Bubble Sort și Selection Sort

BUBBLE SORT

The screenshot shows a presentation slide titled "BubbleSort" with the logo of "UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA" in the top right corner. Below the title, there is a table representing an array of numbers:

0	1	2	3
12	4	8	31

Below the table, a Python code snippet is displayed, illustrating the bubble sort algorithm:

```
def bubbleSort(lista_elemente):  
    n=len(lista_elemente)  
    has_swapped = True  
    while (has_swapped):  
        has_swapped = False  
        for i in range(n-1):  
            if lista_elemente[i] > lista_elemente[i+1]:  
                lista_elemente[i], lista_elemente[i+1] =  
                lista_elemente[i+1], lista_elemente[i]  
                has_swapped = True
```

On the right side of the code, the initial list is defined: `lista_elemente=[12,4,8,31]`.

The slide is part of a presentation titled "BubbleSort-Step-by-Step" by Maria Vancea, as indicated by the top bar. The bottom right corner of the slide shows the number "2".

SELECTION SORT

SelectionSort- step by step

Căutare (Alt+G)

Maria Vancea

Tranziții Animații Expunere diapozitive Înregistrare Revizuire Vizualizare Ajutor

Font Paragraf Desen Editare Voce

SelectionSort

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

0	1	2	3
12	4	8	31

```
def selectSort(list_elemente):  
    for i in range(len(list_elemente)):  
        min_index = i  
  
        for j in range(i+1, len(list_elemente)):  
            if list_elemente[j] < list_elemente[min_index]:  
                min_index = j  
        (list_elemente[i], list_elemente[min_index]) =  
        (list_elemente[min_index], list_elemente[i])
```

List elemente=[12,4,8,31]

2

Salvare automată

SelectionS

Fișier Pornire Inserare Desenare Proiectare Tranziții

Lipire Diapozitiv nou Reutilizați diapozitive Aspect Reinițializare

Clipboard Diapozitive

3

SelectionSort

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

0	1	2	3
12	4	8	31

```
def selectSort(list_elemente):  
    for i in range(len(list_elemente)):  
        min_index = i  
  
        for j in range(i+1, len(list_elemente)):  
            if list_elemente[j] < list_elemente[min_index]:  
                min_index = j  
        (list_elemente[i], list_elemente[min_index]) =  
        (list_elemente[min_index], list_elemente[i])
```

3

4

SelectionSort

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

0	1	2	3
12	4	8	31

```
def selectSort(list_elemente):  
    for i in range(len(list_elemente)):  
        min_index = i  
  
        for j in range(i+1, len(list_elemente)):  
            if list_elemente[j] < list_elemente[min_index]:  
                min_index = j  
        (list_elemente[i], list_elemente[min_index]) =  
        (list_elemente[min_index], list_elemente[i])
```

4

5

SelectionSort

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

0	1	2	3
12	4	8	31

```
def selectSort(list_elemente):  
    for i in range(len(list_elemente)):  
        min_index = i  
  
        for j in range(i+1, len(list_elemente)):  
            if list_elemente[j] < list_elemente[min_index]:  
                min_index = j  
        (list_elemente[i], list_elemente[min_index]) =  
        (list_elemente[min_index], list_elemente[i])
```

5

- ♣ *Apoi am făcut o demonstrație a eficacității a 3 metode de sortare directă și o metodă de sortare avansată.*

Pentru asta am efectuat următoarele:

- *am generat random 2000 de numere și le-am pus într-o listă, din care în urma rulării fiecarui algoritm să scrie într-un alt fișier lista sortată*

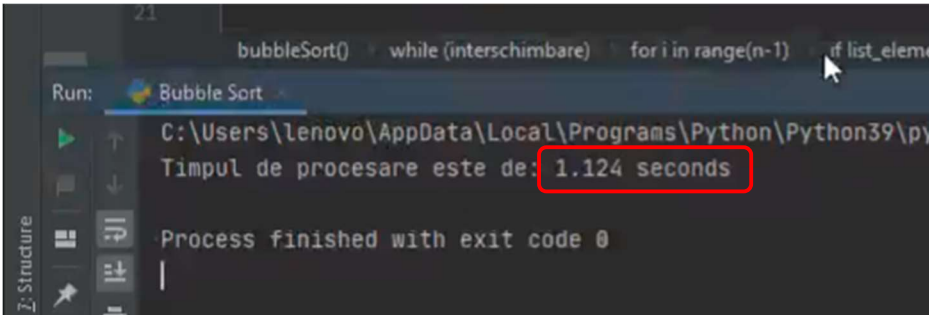
```
1      import random
2
3      lista = []
4      for i in range(2000):
5          a = str(random.randint(1, 1500))
6          lista.append(a)
7
8      scriere_fisier = open("lista_elemente.in", "w")
9      for i in range(len(lista)):
10         scriere_fisier.write(str(lista[i]))
11         scriere_fisier.write(" ")
12     scriere_fisier.close()
```

- *pe lângă asta să se afișeze și timpul execuție a fiecărui algoritm (aici BubbleSort)*

```
start_time = time.time()
bubbleSort(list_elemente)
print("Timpul de procesare este de: %.3f seconds " % (time.time() - start_time))
```

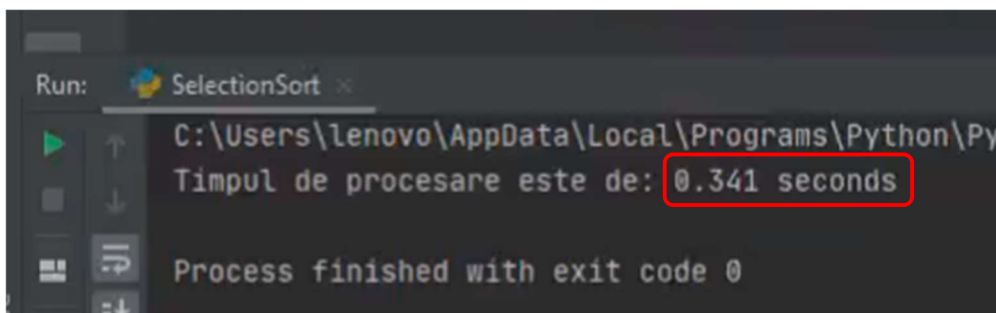
♣ Rezultatele obținute în urma rulării programelor:

► Bubble Sort



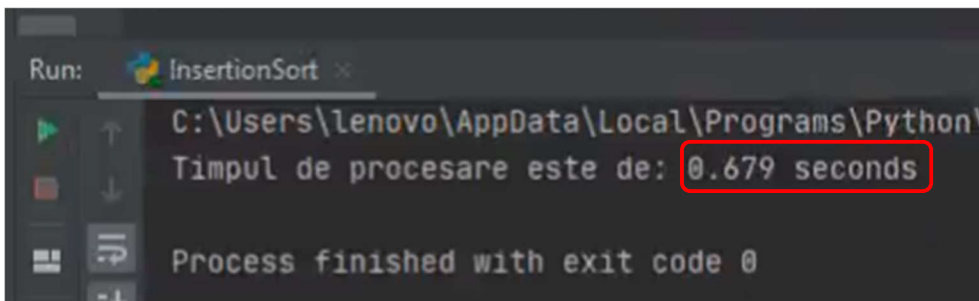
```
Run: Bubble Sort
C:\Users\lenovo\AppData\Local\Programs\Python\Python39\py
Timpul de procesare este de: 1.124 seconds
Process finished with exit code 0
```

► Selection Sort



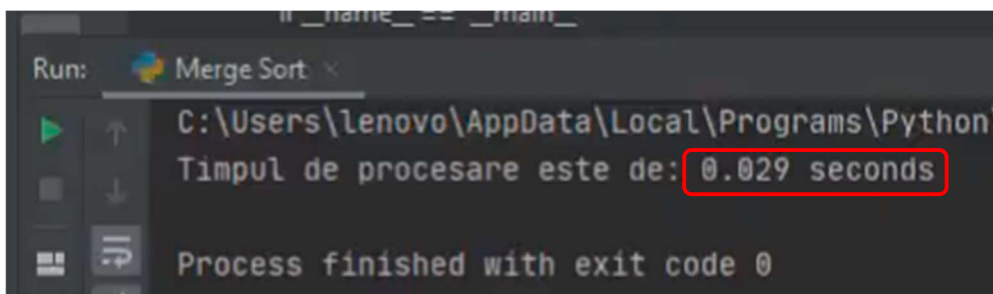
```
Run: SelectionSort
C:\Users\lenovo\AppData\Local\Programs\Python\Py
Timpul de procesare este de: 0.341 seconds
Process finished with exit code 0
```

► Insertion Sort



```
Run: InsertionSort
C:\Users\lenovo\AppData\Local\Programs\Python\
Timpul de procesare este de: 0.679 seconds
Process finished with exit code 0
```

► Merge Sort



```
Run: Merge Sort
C:\Users\lenovo\AppData\Local\Programs\Python\
Timpul de procesare este de: 0.029 seconds
Process finished with exit code 0
```

PREZENTAREA IV

În cadrul celei de-a patra prezentări am realizat următoarele:

- ♣ Folosind modelul dat de domnul profesor am exemplificat pas cu pas (step-by-step) cum funcționează 2 algoritmi de sortare și anume: Shell Sort și Merge Sort

SHELL SORT

The screenshot shows a presentation slide titled "ShellSort" with the logo of "UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA" in the top right corner. The slide displays an array of numbers: 21, 17, 30, 5, 12, indexed from 0 to 4. Below the array, a Python code snippet defines the Shell Sort algorithm. The code is as follows:

```
def shellSort(array):  
    n = len(array)  
    interval = n // 2  
    while interval > 0:  
        for i in range(interval, n):  
            temp = array[i]  
            j = i  
            while j >= interval and array[j - interval] > temp:  
                array[j] = array[j - interval]  
                j -= interval  
            array[j] = temp  
        interval //= 2
```

On the right side of the slide, the initial array is defined: `array = [21, 17, 30, 5, 12]`. The slide is part of a presentation titled "ShellSort - step by step" and is on slide number 2.

MERGE SORT

MergeSort- step-by-step

Căutare (Alt+G)

Maria Vancea

Proiectare Tranziții Animații Expunere diapozitive Înregistrare Revizuire Vizualizare Ajutor

Font Paragraf Desen Editare Voce Proiect

MergeSort

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

0	1	2	3	4
21	17	30	5	12

```
def mergeSort(array):  
    if len(array) > 1:  
        mid = len(array) // 2  
        l = array[:mid]  
        r = array[mid:]  
        mergeSort(l)  
        mergeSort(r)  
        i = j = k = 0  
        while i < len(l) and j < len(r):  
            if l[i] < r[j]:  
                array[k] = l[i]  
                i += 1  
            else:  
                array[k] = r[j]  
                j += 1  
            k += 1  
        while i < len(l):  
            array[k] = l[i]  
            i += 1  
            k += 1  
        while j < len(r):  
            array[k] = r[j]  
            j += 1  
            k += 1  
array = [21, 17, 30, 5, 12]
```

2

- o Timpul de execuție a Shell Sort în sortarea celor 2000 de elemente:

▶ Shell Sort

```
Run: Shell Sort x  
C:\Users\lenovo\AppData\Local\Programs\Python\Python38-32\python.exe  
Timpul de procesare este de: 0.016 seconds  
Process finished with exit code 0
```

MULTUMM!