test-e.ro

CAIET ELEV

Lecția 26

Metoda bulelor (Bubble Sort). Compararea celor 3 metode de sortare

Metoda bulelor compară perechi de elemente vecine și le interschimbă dacă sunt în ordine necorespunzătoare.

DISCIPLINA Informatică **CLASA** a IX-a · mate-info **LIMBAJ** Python 3

NUMELE ELEVULUI .....

CUM FOLOSEȘTI CAIETUL

Rezolvă exercițiile pe hârtie: urmărești execuția pas cu pas, completezi, asociezi și scrii singur soluțiile, apoi le verifici pe platformă. Ce ai voie să folosești: secvențial, if/elif/else, while, for cu range, %, //, liste ([], indexare, len, append), for x in lista, def/return, min/max/sum, import math, slicing, măsurarea timpului, atribuire simultană (permisă de la L20). Nu se folosesc: open, with, class, tkinter, sorted(), .sort() (metodele gata făcute de sortare ar goli lecția de sens). Fără diacritice în șirurile și comentariile din codul Python. Paritatea cu % 2 != 0. Interschimbarea se scrie cu variabilă auxiliară (aux), pentru continuitate cu L24, deși atribuirea simultană e permisă de la L20.

CUPRINS

1 Ce știm despre liste, indici și interschimbare ....
2 Bula care urcă: o singură trecere prin listă .....
3 De la o trecere la algoritmul complet .....
4 Când algoritmul își dă seama singur că a terminat
5 Numărăm operațiile, nu secundele
6 Trei metode, trei situații .....
7 Cod ghidat · cinci probleme .....

SECȚIUNEA 1 · REACTUALIZARE

1 Ce știm despre liste, indici și *interschimbare*

Înainte de lecția nouă, ne reamintim uneltele: cum accesăm un element prin indice, cum aflăm câte elemente are o listă, cum interschimbăm două elemente cu ajutorul unei variabile auxiliare și cum parcurgem o listă privind câte două poziții odată. Nimic nou aici — doar ne încălzim mâinile.

O listă în Python este o colecție ordonată, în care fiecare element are un **indice**: primul element e `a[0]`, al doilea `a[1]`, iar ultimul `a[n - 1]`, unde `n` e numărul de elemente, aflat cu `len(a)`. Indicii încep de la 0, nu de la 1 — o greșeală clasică este să încercăm să citim `a[n]`, care nu există.

Ca să **interschimbăm** două elemente nu putem scrie direct `a[0] = a[2]` urmat de `a[2] = a[0]`: prima atribuire ar distruge valoarea veche a lui `a[0]`, iar a doua ar copia-o pe ea însăși. Avem nevoie de o variabilă auxiliară, care ține valoarea deoparte cât timp mutăm:

```
a = [7, 2, 5]
aux = a[0]
a[0] = a[2]
a[2] = aux
print(a)
```

Rezultatul este `[5, 2, 7]`. Aceeași secvență de trei linii o vom folosi de foarte multe ori în lecția de azi.

O a doua unealtă pe care o reamintim: parcurgerea unei liste **privind câte două poziții odată**. Când vrem să lucrăm cu perechi de elemente aflate una lângă alta, contorul nu merge până la `n - 1`, ci până la `n - 2`, adică `for i in range(n - 1)` — altfel `a[i + 1]` ar ieși în afara listei:

```
n = len(a)
for i in range(n - 1):
    print(a[i], a[i + 1])
```

ATENȚIE

`range(n - 1)` produce indicii `0, 1, …, n - 2`. Ultima pereche examinată este deci `a[n - 2]`, `a[n - 1]` — exact ultimele două elemente, fără să depășim lista.

DE REȚINUT

Din L24 și L25 avem deja două metode de sortare: **selecția minimului** (caută minimul din zona nesortată și îl aduce la locul lui) și **sortarea cu listă de frecvențe** (numără aparițiile fiecărei valori; se poate folosi doar când valorile sunt numere întregi dintr-un interval mic). Le vom compara astăzi cu o a treia.

Convenția de citire folosită în toată lecția: se citește mai întâi `n`, apoi cele `n` numere, **câte unul pe linie**.

A

EXERCIȚIUL A · GRILĂ

Recapitulare: liste, indici, interschimbare

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Pentru a = [4, 7, 2], ce afișează print(len(a))?

   3 | nimic, fiindcă len() se poate aplica doar unei liste deja ordonate crescător | 4 | 2

2. Pentru a = [4, 7, 2], ce afișează print(a[1])?

   7 | apare eroare, pentru că indicii unei liste Python încep de la 1, nu de la 0 | 4 | 2

3. Ce face secvența aux = a[0], apoi a[0] = a[2], apoi a[2] = aux?

   - copiază al treilea element peste primul, iar al treilea rămâne neschimbat
   - șterge primul element și îl adaugă din nou la sfârșitul listei
   - mută toate elementele listei cu o poziție spre stânga, ciclic
   - interschimbă primul element cu al treilea

4. Sortarea cu listă de frecvențe se poate folosi atunci când:

   - valorile sunt numere reale, oricât de mari, dar lista are cel mult 100 de elemente
   - lista este deja ordonată crescător și vrem doar să verificăm acest lucru
   - valorile sunt numere întregi dintr-un interval mic
   - lista conține exclusiv valori distincte, fără nicio repetare

B

EXERCIȚIUL B · DRY-RUN

Poziția celui mai mic element

DRY-RUN

Urmărește bucla pas cu pas. Coloanele i și a[i] sunt date; tu completezi coloana pmin (după pas), adică poziția minimului după ce s-a executat if-ul. Ultimul rând este cel în care condiția i < 4 devine falsă și bucla se oprește.

```
a = [8, 3, 9, 1]
i = 1
pmin = 0
while i < 4:
    if a[i] < a[pmin]:
        pmin = i
    i = i + 1
print(pmin)
```

i	a[i]	pmin (după pas)
1	3	1
2	9	
3	1	
4	–	

HINT *pmin se schimbă DOAR dacă a[i] < a[pmin]. La i = 2 avem 9 < 3, ceea ce e fals, deci pmin rămâne 1.*

✦

SPRIJIN · SCRIE CODUL

Interschimbă primul cu ultimul

SCRIERE COD

Se citește un număr natural n (n ≥ 2), apoi cele n elemente ale unei liste, câte unul pe linie. Interschimbă primul element cu ultimul, folosind o variabilă auxiliară, și afișează lista rezultată. Ex.: pentru n = 5 și lista 5, 3, 8, 1, 9 → se afișează [9, 3, 8, 1, 5].

HINT **Atenție.** *Nu scrie a[0] = a[n - 1] urmat de a[n - 1] = a[0] — prima linie distruge valoarea veche a lui a[0], iar a doua o copiază pe ea însăși. Păstrează-o mai întâi în aux.*

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

Sumele perechilor de vecini

SCRIERE COD

Se citește un număr natural n (n ≥ 2), apoi cele n elemente ale unei liste, câte unul pe linie. Afișează, câte una pe linie, sumele fiecărei perechi de elemente vecine: a[0] + a[1], apoi a[1] + a[2], și așa mai departe până la a[n-2] + a[n-1]. Ex.: pentru n = 4 și lista 1, 2, 3, 4 → se afișează 3, 5 și 7 (adică 1+2, 2+3, 3+4).

HINT *Se afișează exact n − 1 sume, câte una pentru fiecare pereche de vecini.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 2 · IDEEA METODEI BULELOR

2 Bula care urcă: o singură trecere prin *listă*

Metoda bulelor nu caută nimic și nu numără nimic. Se plimbă prin listă de la stânga la dreapta și, la fiecare pas, compară doar două elemente vecine, interschimbându-le dacă sunt în ordine greșită. La sfârșitul unei singure plimbări, cel mai mare element a ajuns deja la capătul din dreapta.

Imaginează-ți un șir de elevi de înălțimi diferite, așezați la întâmplare. Le dăm o regulă foarte simplă: „uită-te la vecinul din dreapta ta; dacă tu ești mai înalt decât el, schimbați locurile”. Aplicăm regula o dată pentru fiecare pereche de vecini, de la stânga la dreapta. Ce se întâmplă? Cel mai înalt elev, oriunde se afla, e împins tot mai la dreapta la fiecare comparație în care participă și ajunge inevitabil la capătul șirului. Exact ca o bulă de aer care urcă la suprafață — de aici numele metodei.

> **METODA BULELOR (BUBBLE SORT)**
>
> Se compară perechi de elemente **vecine** (`a[i]` cu `a[i + 1]`), de la stânga la dreapta, și se interschimbă cele aflate în ordine necorespunzătoare. O parcurgere completă de acest fel se numește **o trecere**.

Iată o trecere completă, scrisă în Python:

```
n = len(a)
for i in range(n - 1):
    if a[i] > a[i + 1]:
        aux = a[i]
        a[i] = a[i + 1]
        a[i + 1] = aux
```

Să urmărim ce face pe lista `[5, 1, 4, 2, 8]`. La i = 0 comparăm 5 cu 1 și interschimbăm: `[1, 5, 4, 2, 8]`. La i = 1 comparăm 5 cu 4 și interschimbăm: `[1, 4, 5, 2, 8]`. La i = 2 comparăm 5 cu 2 și interschimbăm: `[1, 4, 2, 5, 8]`. La i = 3 comparăm 5 cu 8 — sunt deja în ordine, nu facem nimic. Rezultatul trecerii: `[1, 4, 2, 5, 8]`.

Observă două lucruri. Lista **nu** e ordonată după o singură trecere — 4 și 2 sunt încă inversate. Dar 8, cel mai mare element, a ajuns la locul lui **definitiv**. Asta e câștigul garantat al fiecărei treceri, și pe el se va sprijini algoritmul complet din secțiunea următoare.

ATENȚIE

Odată interschimbate, valorile din `a[i]` și `a[i + 1]` s-au schimbat. La pasul următor comparăm noile valori, nu cele de dinainte. De aceea, în trecerea de mai sus, valoarea 5 „călătorește” cu noi de la o comparație la alta.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Ideea „compară doar vecinii” apare peste tot unde datele sunt aproape aranjate și trebuie doar ajustate local: la reașezarea cărților pe un raft de bibliotecă, unde muți o carte doar cu un loc mai încolo; la afișarea clasamentului live într-un concurs, unde după fiecare probă doar câțiva concurenți își schimbă între ei pozițiile vecine; la aplicațiile de mesagerie, care reordonează conversațiile mutând-o pe cea nou-sosită cu o poziție mai sus, pas cu pas, până în vârf.

A

EXERCIȚIUL A · GRILĂ

Ce face o trecere

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Metoda bulelor compară între ele:

   `primul element al listei cu fiecare dintre celelalte elemente, pe rând`

   `fiecare element cu cel mai mic element găsit până în acel moment`

   `două elemente vecine, aflate pe poziții consecutive`

   `nu compară elemente; numără de câte ori apare fiecare valoare din listă`

2. După o singură trecere completă, lista [5, 1, 4, 2, 8] devine:

   `[5, 1, 2, 4, 8]` `[1, 2, 4, 5, 8]` `[1, 4, 2, 5, 8]` `[1, 4, 5, 2, 8]`

3. Ce element ajunge sigur la poziția lui definitivă după prima trecere?

   `elementul din mijlocul listei, indiferent de valoarea lui`

   `cel mai mic element din listă, care ajunge pe prima poziție`

   `niciunul; după prima trecere nu se poate spune nimic sigur despre listă`

   `cel mai mare element din listă`

B

EXERCIȚIUL B · DRY-RUN

O trecere pe [5, 1, 4, 2, 8]

DRY-RUN

Urmărește trecerea pas cu pas. Coloanele a[i] și a[i+1] îți sunt date, cu valorile din momentul comparației (ele se schimbă pe parcurs!). Tu completezi ultima coloană, cu textul DA sau NU. Ultimul rând arată starea în care bucla se oprește.

```
a = [5, 1, 4, 2, 8]
i = 0
while i < 4:
    if a[i] > a[i + 1]:
        aux = a[i]
        a[i] = a[i + 1]
        a[i + 1] = aux
    i = i + 1
print(a)
```

i	a[i]	a[i+1]	interschimbăm? DA/NU
0	5	1	DA
1	5	4	
2	5	2	
3	5	8	
4	–	–	–

HINT *Interschimbăm doar când elementul din stânga e mai MARE decât cel din dreapta. La ultima pereche, 5 < 8, deci nu se interschimbă nimic.*

C

EXERCIȚIUL C · SCRIE CODUL

O singură trecere

SCRIERE COD

Se citește un număr natural n (n ≥ 2), apoi cele n elemente ale unei liste, câte unul pe linie. Efectuează o singură trecere a metodei bulelor (parcurge perechile de vecini de la stânga la dreapta și interschimbă-le dacă cel din stânga e mai mare) și afișează lista rezultată. Ex.: pentru n = 5 și lista 5, 1, 4, 2, 8 → se afișează [1, 4, 2, 5, 8].

HINT *Lista rezultată nu este, în general, ordonată — o singură trecere garantează doar că cel mai mare element a ajuns la sfârșit.*

SOLUȚIA TA · PYTHON

3 SECȚIUNEA 3 · ALGORITMUL COMPLET

De la o trecere la algoritmul *complet*

O trecere așează un element la locul lui. Repetăm trecerile — și, pentru că după fiecare trecere capătul din dreapta e deja aranjat, zona pe care mai lucrăm se scurtează cu o poziție. Cu n − 1 treceri, lista e sigur ordonată.

Din S2 știm că o trecere duce cel mai mare element la capătul din dreapta. Ce se întâmplă la a doua trecere? Cel mai mare element din **restul** listei ajunge pe penultima poziție. La a treia trecere, următorul ajunge pe antepenultima. Se vede tiparul: **după k treceri, ultimele k elemente sunt deja la locul lor definitiv** și nu mai trebuie atinse.

De aici două consecințe practice. Prima: nu are rost să comparăm perechi din zona deja aranjată, deci bucla interioară nu merge până la `n - 1`, ci până la `n - 1 - k`, unde `k` numără trecerile făcute. A doua: n − 1 treceri sunt suficiente — la fiecare trecere fixăm cel puțin un element, iar dacă n − 1 elemente sunt la locul lor, atunci și ultimul rămas e obligatoriu la locul lui.

Algoritmul complet are deci **două bucle imbricate**: cea exterioară numără trecerile, cea interioară face o trecere.

```
n = len(a)
for k in range(n - 1):
    for i in range(n - 1 - k):
        if a[i] > a[i + 1]:
            aux = a[i]
            a[i] = a[i + 1]
            a[i + 1] = aux
```

Pe lista `[5, 1, 4, 2, 8]` (n = 5) se fac 4 treceri, cu 4, 3, 2 și 1 comparări.
Rezultatul: `[1, 2, 4, 5, 8]`.

Același algoritm, scris în pseudocod — forma pe care o folosim când proiectăm, înainte de a alege limbajul:

```
citește n
citește a₀, a₁, …, aₙ₋₁
pentru k ← 0, n-2 execută
    pentru i ← 0, n-2-k execută
        dacă aᵢ > aᵢ₊₁ atunci
            aux ← aᵢ
            aᵢ ← aᵢ₊₁
            aᵢ₊₁ ← aux
        sfârșit dacă
    sfârșit pentru
sfârșit pentru
scrie a₀, a₁, …, aₙ₋₁
```

Compară-le linie cu linie: `pentru k ← 0, n-2 execută` spune exact ce spune `for k in range(n - 1)` — contorul ia pe rând valorile de la 0 la n−2, limitele incluse. Diferența e că pseudocodul închide fiecare structură cu un terminator (`sfârșit pentru`, `sfârșit dacă`), în timp ce Python se bazează doar pe indentare.

> **ATENȚIE**
>
> Dacă bucla interioară merge până la `range(n - k)` în loc de `range(n - 1 - k)`, atunci `a[i + 1]` iese în afara listei la ultimul pas și programul se oprește cu eroare. Scăderea acelui 1 nu e un detaliu: e chiar condiția ca perechea `a[i]`, `a[i + 1]` să existe.

> **DE REȚINUT**

Sortarea crescătoare compară cu `a[i] > a[i + 1]`. Pentru sortare descrescătoare se schimbă **doar** semnul comparației, în `a[i] < a[i + 1]` — restul algoritmului rămâne identic.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Ordonarea completă cu metoda bulelor apare în programe mici, unde simplitatea contează mai mult decât viteza: în firmware-ul unei stații meteo care sortează cele 24 de temperaturi ale zilei pentru a găsi mediana; în programul unui automat de cafea care ordonează cele 8 prețuri pentru afișarea meniului; în aplicația unui profesor de sport care aranjează crescător înălțimile celor 25 de elevi ai clasei înainte de a-i împărți pe echipe.

A

EXERCIȚIUL A · GRILĂ

Cele două bucle

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. De ce bucla interioară merge până la n - 1 - k și nu până la n - 1?

   `pentru că lista se scurtează efectiv cu un element după fiecare trecere efectuată`

   `pentru că după k treceri ultimele k elemente sunt deja la locul lor definitiv`

   `pentru ca numărul total de comparări să fie exact același la fiecare trecere`

   `nu există niciun motiv real; range(n - 1) ar da exact același rezultat, doar mai lent`

2. Câte treceri face varianta neoptimizată pentru o listă cu n = 6 elemente?

   `6` `depinde de valorile din listă, nu se poate spune înainte de a rula programul` `15` `5`

3. Pentru n = 5 și k = 2, câte comparări face bucla for i in range(n - 1 - k)?

   `3` `4` `2` `niciuna, fiindcă n - 1 - k este deja 0 pentru aceste valori`

B

EXERCIȚIUL B · COMPLETARE

Cum se micșorează zona de lucru

COMPLETARE

Completează golurile folosind cuvintele din rezervă (două cuvinte rămân nefolosite).

`k` `depărtate` `locul` `două` `vecine` `o`

La metoda bulelor, bucla interioară compară perechi de elemente _____, iar zona comparată se micșorează cu _____ poziție după fiecare trecere, fiindcă ultimele _____ elemente sunt deja la _____ lor definitiv.

C

EXERCIȚIUL C · SCRIE CODUL

Sortare crescătoare cu metoda bulelor

SCRIERE COD

Se citește un număr natural n (n ≥ 2), apoi cele n elemente ale unei liste, câte unul pe linie. Ordonează lista crescător folosind metoda bulelor, cu două bucle imbricate, și afișează lista ordonată. Ex.: pentru n = 5 și lista 5, 1, 4, 2, 8 → se afișează [1, 2, 4, 5, 8].

HINT **Atenție.** *Nu folosi sorted() și nici metoda .sort() — scopul exercițiului este chiar scrierea algoritmului. Bucla interioară merge până la n - 1 - k, altfel a[i + 1] iese din listă.*

SOLUȚIA TA · PYTHON

4 SECȚIUNEA 4 · OPTIMIZAREA

Când algoritmul își dă seama singur că a *terminat*

Metoda bulelor are un avantaj pe care selecția minimului nu-l are: poate observa că lista e deja ordonată. Dacă o trecere completă se încheie fără nicio interschimbare, toate perechile de vecini sunt în ordine — deci nu mai are rost să continuăm.

Varianta din S3 face mereu n − 1 treceri, chiar și dacă lista era ordonată de la bun început. Pentru `[1, 2, 3, 4, 5]` face 4 treceri și 10 comparări, fără nicio interschimbare — muncă degeaba.

Aici intervine o observație simplă și puternică: dacă într-o trecere completă **nu s-a făcut nicio interschimbare**, înseamnă că fiecare element e mai mic sau egal cu vecinul din dreapta lui. Dar asta e chiar definiția unei liste ordonate crescător! Deci putem opri algoritmul pe loc.

Ca să știm dacă s-a interschimbat ceva, folosim o variabilă **martor**, pe care o numim `schimbat`. O punem pe 0 la începutul fiecărei treceri și pe 1 în interiorul `if`-ului, de fiecare dată când facem o interschimbare. La sfârșitul trecerii, dacă `schimbat` e încă 0, ne oprim. Bucla exterioară nu mai poate fi un `for`, fiindcă nu știm dinainte câte treceri vom face — devine un `while`.

> **METODA BULELOR CU OPRIRE TIMPURIE**
>
> Un martor `schimbat` se pune pe 0 la începutul fiecărei treceri și pe 1 la fiecare interschimbare. Bucla exterioară continuă cât timp `schimbat == 1`; prima trecere fără interschimbări încheie algoritmul.

```
n = len(a)
k = 0
schimbat = 1
while schimbat == 1:
    schimbat = 0
    for i in range(n - 1 - k):
        if a[i] > a[i + 1]:
            aux = a[i]
            a[i] = a[i + 1]
            a[i + 1] = aux
            schimbat = 1
    k = k + 1
```

Pe lista `[3, 1, 2, 4]` (n = 4) se fac exact 2 treceri: prima cu 3 comparări și 2 interschimbări, care duce lista la `[1, 2, 3, 4]`; a doua cu 2 comparări și 0 interschimbări, care doar confirmă că totul e în ordine. Pe lista deja ordonată `[1, 2, 3, 4, 5]` se face o singură trecere, cu 4 comparări în loc de 10.

> **ATENȚIE**
>
> `schimbat = 0` trebuie pus **la începutul fiecărei treceri**, în interiorul buclei `while`, nu înainte de ea. Dacă îl scrii înaintea `while`-ului, martorul rămâne 1 după prima interschimbare și bucla nu se mai oprește niciodată — program blocat.

> **UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ**
>
> Oprirea timpurie e principiul „nu mai munci dacă nu e nimic de făcut”, și apare peste tot: aplicația de e-mail care verifică dacă lista mesajelor s-a schimbat înainte de a redesena ecranul; catalogul electronic al școlii, care reordonează lista elevilor doar dacă s-a înscris

cineva nou; programul de backup care se oprește imediat ce constată că niciun fișier nu s-a modificat de ultima dată.

A

EXERCIȚIUL A · GRILĂ

Rolul martorului

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Ce rol are variabila schimbat?

   numără câte interschimbări s-au făcut în total, de la începutul algoritmului

   memorează dacă s-a făcut cel puțin o interschimbare în trecerea curentă

   reține poziția ultimei perechi de elemente care a fost interschimbată

   memorează valoarea celui mai mare element găsit în trecerea curentă

2. Pentru o listă deja ordonată crescător, câte treceri face varianta optimizată?

   niciuna, fiindcă algoritmul verifică ordinea listei înainte de prima trecere

   2

   1

   n − 1

3. Ce se întâmplă dacă mutăm schimbat = 0 înaintea buclei while, în loc să-l lăsăm în interiorul ei?

   nu se schimbă nimic; poziția atribuirii nu influențează felul în care rulează bucla

   rezultatul rămâne corect, dar se fac cu exact o trecere mai multe decât e nevoie

   algoritmul se oprește după prima trecere, chiar dacă lista nu e ordonată

   după prima interschimbare, bucla nu se mai oprește niciodată

B

EXERCIȚIUL B · DRY-RUN

Câte treceri pe [3, 1, 2, 4]

DRY-RUN

Urmărește varianta optimizată, trecere cu trecere. Coloana comparări îți e dată; tu completezi interschimbări (număr) și continuă? (textul DA sau NU). Ultimul rând e trecerea în care martorul rămâne 0 și bucla while se oprește.

```
a = [3, 1, 2, 4]
n = 4
k = 0
schimbat = 1
while schimbat == 1:
    schimbat = 0
    for i in range(n - 1 - k):
```

```
        if a[i] > a[i + 1]:
            aux = a[i]
            a[i] = a[i + 1]
            a[i + 1] = aux
            schimbat = 1
    k = k + 1
print(a)
```

trecerea	comparări	interschimbări	continuă? DA/NU
1	3	2	DA
2	2		

HINT *În trecerea 1 se interschimbă (3,1) și apoi (3,2) — două interschimbări. În trecerea 2 lista e deja [1, 2, 3, 4], deci nicio pereche nu mai e în ordine greșită.*

C

EXERCIȚIUL C · SCRIE CODUL

Sortare cu oprire timpurie

SCRIERE COD

Se citește un număr natural n (n ≥ 2), apoi cele n elemente ale unei liste, câte unul pe linie. Ordonează lista crescător folosind metoda bulelor cu oprire timpurie (martorul schimbat) și afișează lista ordonată. Ex.: pentru n = 5 și lista 5, 1, 4, 2, 8 → se afișează [1, 2, 4, 5, 8].

HINT **Atenție.** *Pune schimbat = 0 la începutul fiecărei treceri, în interiorul buclei while. Dacă îl pui înainte de while, programul intră în buclă infinită.*

SOLUȚIA TA · PYTHON

PERFORMANȚĂ · SCRIE CODUL

Câte treceri au fost necesare

SCRIERE COD

Se citește un număr natural n (n ≥ 2), apoi cele n elemente ale unei liste, câte unul pe linie. Afișează numărul de treceri efectuate de varianta cu oprire timpurie a metodei bulelor, numărând și trecerea finală, cea care nu produce nicio interschimbare și oprește algoritmul. Ex.: pentru n = 4 și lista 3, 1, 2, 4 → se afișează 2: prima trecere ordonează lista, a doua doar confirmă că totul e în ordine.

HINT *Pentru o listă deja ordonată crescător răspunsul e mereu 1: o singură trecere e de ajuns ca algoritmul să constate că nu are ce schimba.*

HINT **Atenție.** *Nu confunda trecerile cu comparările. O trecere e o parcurgere completă a zonei necomparate; ea conține mai multe comparări.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 5 · CÂT COSTĂ

5 Numărăm operațiile, nu *secundele*

Ca să comparăm cinstit două metode de sortare nu măsurăm secunde — ele depind de calculator. Numărăm **operațiile**: câte comparări și câte interschimbări face fiecare metodă. Pentru metoda bulelor neoptimizată numărul de comparări nu depinde deloc de date și se poate calcula exact.

La varianta neoptimizată, prima trecere face n − 1 comparări, a doua n − 2, a treia n − 3, și așa mai departe până la ultima, care face o singură comparare. Suma lor este suma primelor n − 1 numere naturale:

```
(n - 1) + (n - 2) + ... + 2 + 1 = n * (n - 1) // 2
```

Pentru n = 5 sunt 10 comparări; pentru n = 10 sunt 45; pentru n = 100 sunt 4 950; pentru n = 1 000 sunt 499 500. Observă ce se întâmplă când n crește de 10 ori: de la 45 la 4 950, adică de aproximativ **100 de ori** mai multe comparări. Când n se înmulțește cu 10, numărul de operații se înmulțește cu 10², adică 100. De asta spunem că metoda are complexitatea **O(n²)**: numărul de operații crește proporțional cu pătratul numărului de elemente.

Numărul de **interschimbări**, în schimb, depinde de date. În cazul cel mai favorabil (listă deja ordonată) este 0. În cazul cel mai defavorabil (listă ordonată invers) fiecare comparare produce o interschimbare, deci tot n(n−1)/2 — pentru `[5, 4, 3, 2, 1]` sunt exact 10 interschimbări. Pe o listă amestecată obișnuită, ca `[5, 1, 4, 2, 8]`, sunt 4.

Optimizarea din S4 nu schimbă complexitatea în cazul defavorabil — tot O(n²). Ce schimbă e **cazul favorabil**: pe o listă deja ordonată face o singură trecere, adică n − 1 comparări, deci O(n).

DE REȚINUT

Comparările variantei neoptimizate sunt **mereu** n(n−1)/2, indiferent de date. Interschimbările sunt între 0 (listă ordonată) și n(n−1)/2 (listă ordonată invers).

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Numărarea operațiilor e felul în care inginerii aleg ce cod ajunge într-un produs: dezvoltatorii unui joc pe telefon calculează câte comparări încap în cele 16 milisecunde dintre două cadre; o platformă de comerț online estimează câte operații cere reordonarea a 100 000 de produse înainte de a o porni pe serverul de vânzări; un programator de aplicații medicale verifică dacă sortarea a 50 000 de analize se termină în timpul de răspuns garantat pacientului.

A

EXERCIȚIUL A · GRILĂ

Câte operații

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Pentru n = 10, câte comparări face varianta neoptimizată a metodei bulelor?

   `90` `100`

   `depinde de valorile din listă; nu se poate spune înainte de a rula programul` `45`

2. Complexitatea metodei bulelor în cazul cel mai defavorabil este:

   `O(n)` `O(n + k)` `O(n²)`

   `O(1), deoarece nu se folosește memorie suplimentară în afara variabilei aux`

3. Dacă numărul de elemente crește de 10 ori, numărul de comparări crește de aproximativ:

   `rămâne același, fiindcă zona comparată se micșorează la fiecare trecere` `10 ori`

   `100 de ori` `20 de ori`

B

EXERCIȚIUL B · COMPLETARE

Formula costului

COMPLETARE

Completează golurile folosind cuvintele din rezervă (două cuvinte rămân nefolosite).

`O(n)` `n-1` `n(n-1)/2` `O(n+k)` `O(n²)` `n²`

Numărul de comparări al metodei bulelor neoptimizate este _____, de aceea spunem că metoda are complexitatea _____. Pe o listă deja ordonată, varianta cu oprire timpurie face doar _____ comparări, adică _____.

C

EXERCIȚIUL C · SCRIE CODUL

Numărul de interschimbări

SCRIERE COD

Se citește un număr natural n (n ≥ 2), apoi cele n elemente ale unei liste, câte unul pe linie. Ordonează lista crescător cu metoda bulelor neoptimizată și afișează câte interschimbări au fost făcute în total. Ex.: pentru n = 5 și lista 5, 1, 4, 2, 8 → se afișează 4.

HINT *Pentru o listă ordonată invers, ca 5, 4, 3, 2, 1, răspunsul este 10 — adică exact numărul de comparări, fiindcă fiecare comparare produce o interschimbare.*

HINT **Atenție.** *Contorul se incrementează în interiorul if-ului, nu în bucla interioară — altfel ai număra comparările, nu interschimbările.*

SOLUȚIA TA · PYTHON

6 SECȚIUNEA 6 · COMPARAREA CELOR 3 METODE

Trei metode, trei *situații*

Acum avem trei metode de sortare. Niciuna nu e „cea mai bună” în general — fiecare e cea mai bună într-un anumit context. Alegerea se face după două întrebări: ce fel de valori sortăm și cât de multe sunt.

Să le punem față în față pe același set de date, `[5, 1, 4, 2, 8]`, cu numerele măsurate direct în Python:

METODĂ	COMPARĂRI	INTERSCHIMBĂRI	COMPLEXITATE	MEMORIE SUPLIMENTARĂ
Selecția minimului	10	2	O(n²)	două variabile (`pmin`, `aux`) — O(1)
Metoda bulelor, neoptimizată	10	4	O(n²)	o variabilă (`aux`) — O(1)
Metoda bulelor, cu oprire timpurie	9	4	O(n²), dar O(n) pe listă ordonată	două variabile (`aux`, `schimbat`) — O(1)
Lista de frecvențe	nu compară elemente	nu interschimbă	O(n + k)	o listă de k + 1 contoare — O(k)

Cele două metode O(n²) fac același număr de comparări. Diferența e de stil: selecția minimului **caută** înainte de a muta, deci face puține interschimbări (cel mult n − 1, una per pas); metoda bulelor **mută des**, dar câte o poziție. Avantajul unic al bulelor e oprirea timpurie: e singura dintre cele trei care „vede” că lista era deja ordonată.

Lista de frecvențe joacă alt joc: nu compară nimic, doar numără. De aceea e mult mai rapidă — O(n + k), unde k e numărul de valori distincte posibile. Prețul: se poate folosi **doar** pentru numere întregi dintr-un interval mic. Pentru 1 000 000 de coduri de gravitate cu valori 1–5 e imbatabilă; pentru 20 de prețuri cu zecimale e inutilizabilă.

Ultima coloană a tabelului spune și ea ceva. Primele trei metode folosesc un **număr fix** de variabile în plus, oricât de lungă ar fi lista: două, sau una, sau două — dar niciodată mai multe. Spunem că au complexitate de spațiu **O(1)**. Lista de frecvențe are nevoie de memorie proporțională cu *intervalul* valorilor, adică **O(k)**: pentru coduri 1–5 sunt 6 contoare, dar pentru valori până la un milion ar fi un milion de contoare. Viteza ei nu e gratuită — se plătește în memorie.

DE REȚINUT — REGULA DE ALEGERE

Valori întregi într-un interval mic, oricâte → **lista de frecvențe**. Listă mică, orice fel de valori → **selecția minimului** sau **metoda bulelor**. Listă mică și aproape ordonată → **metoda bulelor cu oprire timpurie**. Listă foarte mare cu valori oarecare → niciuna dintre cele trei; e nevoie de o metodă mai eficientă, pe care o vei învăța mai târziu.

ATENȚIE

O(n + k) nu e mereu mai bun decât O(n²). Dacă sortăm 10 numere întregi din intervalul 1–1 000 000, lista de frecvențe ar avea nevoie de un milion de contoare pentru 10 valori — k este uriaș, iar metoda devine absurdă. Contează mărimea **intervalului**, nu doar faptul că valorile sunt întregi.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Alegerea metodei potrivite e o decizie de inginerie luată zilnic: la triajul din camera de urgență, pacienții se ordonează după coduri de gravitate 1–5, deci lista de frecvențe; la o companie de lifturi, cererile de mentenanță se ordonează după tipul defecțiunii, coduri 1–8, iar la un milion de cereri lista de frecvențe e singura opțiune rezonabilă; la clasamentul celor 12 finaliști ai unui concurs de talente, cu scoruri zecimale, selecția minimului sau metoda bulelor rezolvă totul instantaneu.

A

EXERCIȚIUL A · GRILĂ

Ce metodă alegem

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Pentru sortarea a 1 000 000 de coduri de gravitate cu valori întregi 1–5, metoda potrivită este:

   `oricare dintre cele trei; la un milion de elemente diferența dintre ele nu se simte`

   `selecția minimului, fiindcă face cele mai puține interschimbări din cele trei`

   `metoda bulelor cu oprire timpurie, care se adaptează datelor`

   `lista de frecvențe, cu complexitate O(n + k)`

2. Pentru sortarea a 20 de prețuri cu zecimale, metoda potrivită este:

   `lista de frecvențe, după rotunjirea prealabilă a tuturor prețurilor la întregi`

   `lista de frecvențe, cu un contor separat pentru fiecare preț distinct din listă`

   `niciuna; prețurile cu zecimale nu se pot ordona cu metodele învățate`

   `metoda bulelor sau selecția minimului`

3. Care dintre cele trei metode „vede” singură că lista era deja ordonată și se oprește imediat?

   `lista de frecvențe, care obține direct contorii în ordinea corectă`

   `selecția minimului, care nu găsește nimic de mutat la niciun pas`

   `niciuna; toate trei parcurg lista de la început până la sfârșit în același număr de pași`

   `metoda bulelor cu oprire timpurie`

B

EXERCIȚIUL B · ASOCIERE

Situația reală și metoda potrivită

ASOCIERE

Asociază fiecare situație cu metoda de sortare potrivită. Coloana din dreapta e amestecată. Scrie în căsuța din fața fiecărui element din stânga litera variantei potrivite. Fiecare literă se folosește o singură dată.

2 000 de coduri de gravitate din triajul medical, valori întregi 1–5

12 scoruri cu zecimale ale finaliștilor unui concurs, unde fiecare mutare a unui concurent în clasament trebuie afișată pe ecran, deci e costisitoare

30 de note aproape ordonate, adăugate una câte una la sfârșitul listei

10 000 000 de prețuri cu zecimale ale unui magazin online

A Selecția minimului

B Metoda bulelor cu oprire timpurie

C Niciuna dintre cele trei — e nevoie de o metodă mai eficientă

D Lista de frecvențe

C

EXERCIȚIUL C · SCRIE CODUL

Triajul pacienților

SCRIERE COD

La camera de urgență, fiecare pacient primește un cod de gravitate, număr întreg între 1 și 5. Se citește un număr natural n (n ≥ 1), apoi cele n coduri, câte unul pe linie. Afișează codurile în ordine crescătoare, folosind metoda listei de frecvențe (nu compara codurile între ele, ci numără-le aparițiile). Ex.: pentru n = 6 și codurile 3, 1, 5, 1, 4, 2 → se afișează, câte unul pe linie, 1, 1, 2, 3, 4, 5.

HINT *Aici lista de frecvențe are nevoie de doar 5 contoare, indiferent cât de mare e n. De aceea e metoda potrivită pentru un spital care primește mii de pacienți pe lună.*

HINT **Atenție.** *Codurile sunt între 1 și 5, deci contorii pot sta pe pozițiile 1…5 ale unei liste de 6 elemente (poziția 0 rămâne nefolosită). Așa nu mai trebuie să scazi 1 din fiecare cod.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 7 · COD GHIDAT

7 Scrie cod care *trece testele*

Cinci probleme. Scrie soluția pe hârtie, urmărește-o mental pe exemplul dat și abia apoi tastează-o pe platformă, unde e verificată automat pe 5 cazuri de test.

PROBLEMA 7.1 · COD GHIDAT

1 *Clasamentul finaliștilor*

SCRIERE COD

La finala unui concurs de talente, fiecare finalist a primit un punctaj, număr natural. Se citește un număr natural n (n ≥ 2), apoi cele n punctaje, câte unul pe linie. Afișează punctajele ordonate descrescător, folosind metoda bulelor. Ex.: pentru n = 5 și punctajele 72, 95, 88, 60, 95 → se afișează [95, 95, 88, 72, 60].

HINT *Față de sortarea crescătoare se schimbă doar semnul comparației: a[i] < a[i + 1] în loc de a[i] > a[i + 1].*

SOLUȚIA TA · PYTHON

2

PROBLEMA 7.2 · COD GHIDAT

Era deja ordonată?

SCRIERE COD

Se citește un număr natural n (n ≥ 2), apoi cele n elemente ale unei liste, câte unul pe linie. Afișează pe primul rând DA dacă lista citită era deja ordonată crescător sau NU în caz contrar, iar pe al doilea rând lista ordonată crescător cu metoda bulelor. Ex.: pentru n = 5 și lista 5, 1, 4, 2, 8 → se afișează pe primul rând NU, iar pe al doilea [1, 2, 4, 5, 8].

HINT **Atenție.** *Verificarea trebuie făcută înainte de sortare — după sortare lista e ordonată oricum, iar răspunsul ar fi mereu DA.*

HINT *O listă e ordonată crescător dacă nicio pereche de vecini nu are elementul din stânga mai mare decât cel din dreapta. Valorile egale nu strică ordinea.*

SOLUȚIA TA · PYTHON

3

PROBLEMA 7.3 · COD GHIDAT

Lista după două treceri

SCRIERE COD

Se citește un număr natural n (n ≥ 2), apoi cele n elemente ale unei liste, câte unul pe linie. Efectuează exact două treceri ale metodei bulelor neoptimizate (a doua trecere lucrează pe o zonă mai mică cu o poziție) și afișează lista rezultată. Ex.: pentru n = 5 și lista 5, 4, 3, 2, 1 → se afișează [3, 2, 1, 4, 5]: prima trecere duce 5 la capăt, a doua duce 4 pe penultima poziție.

HINT *Pentru unele liste, două treceri sunt deja de ajuns ca lista să iasă ordonată — dar nu pentru toate.*

HINT **Atenție.** *A doua trecere merge până la n - 2, nu până la n - 1. Rezultatul ar fi același și dacă ai merge până la n - 1: ultimul element e deja maximul listei, deci comparația în plus nu ar schimba niciodată nimic. Diferența e de eficiență, nu de corectitudine — o comparație făcută degeaba. Exact pe economia asta se sprijină bucla for i in range(n - 1 - k) din S3.*

SOLUȚIA TA · PYTHON

4 PROBLEMA 7.4 · REPARĂ CODUL

Repară numărătoarea comparărilor

SCRIERE COD

Programul de mai jos ar trebui să afișeze numărul de comparări făcute de metoda bulelor neoptimizată pentru o listă cu n elemente — adică n(n−1)/2, afirmația demonstrată în secțiunea S5. Programul rulează fără eroare, dar dă un rezultat greșit: pentru n = 5 și lista 5, 1, 4, 2, 8 afișează 16 în loc de 10. Găsește greșeala, corectează-o și afișează numărul corect de comparări.

```
n = int(input())
a = []
for i in range(n):
    x = int(input())
    a.append(x)
c = 0
```

```
for k in range(n - 1):
    for i in range(n - 1):
        c = c + 1
print(c)
```

Ex.: pentru n = 5 și lista 5, 1, 4, 2, 8 → se afișează 10 (comparările sunt 4 + 3 + 2 + 1).

HINT *Rezultatul nu depinde de valorile din listă, doar de n — la varianta neoptimizată se fac mereu aceleași comparări, indiferent de date.*

HINT **Atenție.** *Codul stricat numără n − 1 comparări la fiecare trecere, deci (n−1)·(n−1) = 16 pentru n = 5. Zona comparată însă se micșorează cu o poziție după fiecare trecere.*

PROGRAMUL CORECTAT · PYTHON

PROBLEMA 7.5 · PROVOCARE

Cât economisește optimizarea

SCRIERE COD

Se citește un număr natural n (n ≥ 2), apoi cele n elemente ale unei liste, câte unul pe linie. Afișează pe același rând, separate printr-un spațiu, două numere: câte comparări face varianta neoptimizată a metodei bulelor și câte face varianta cu oprire timpurie **din secțiunea S4** — cea în care zona

comparată se micșorează cu o poziție după fiecare trecere — pe aceeași listă de start. Ex.: pentru n = 5 și lista 1, 2, 3, 4, 5 → se afișează 10 4: varianta neoptimizată face mereu 10 comparări, iar cea optimizată se oprește după prima trecere, cu 4.

HINT *Pe o listă ordonată invers cele două variante fac exact același număr de comparări — optimizarea nu ajută în cazul defavorabil.*

HINT **Atenție.** *Cele două variante trebuie să pornească de la aceeași listă. Prima sortare modifică lista, deci fă-i o copie înainte, cu b = a[:].*

SOLUȚIA TA · PYTHON