INFORMATICĂ · CLASA A X-A · TABLOURI UNIDIMENSIONALE · LECȚIA 4

Sortarea vectorilor: bule, selecție, inserție (și sort din STL)

Durată: 60 min Nivel: toate profilurile Înainte: lecțiile x-03

CE VEI ȘTI DUPĂ LECȚIE

- Implementezi metoda bulelor, sortarea prin selecție și prin inserție.
- Urmărești pas cu pas o sortare (exercițiu frecvent la grile).
- Sortezi după un criteriu (suma cifrelor, paritate, descrescător).
- Folosești sort() din biblioteca <algorithm> când e permis.

1. Ideea de sortare

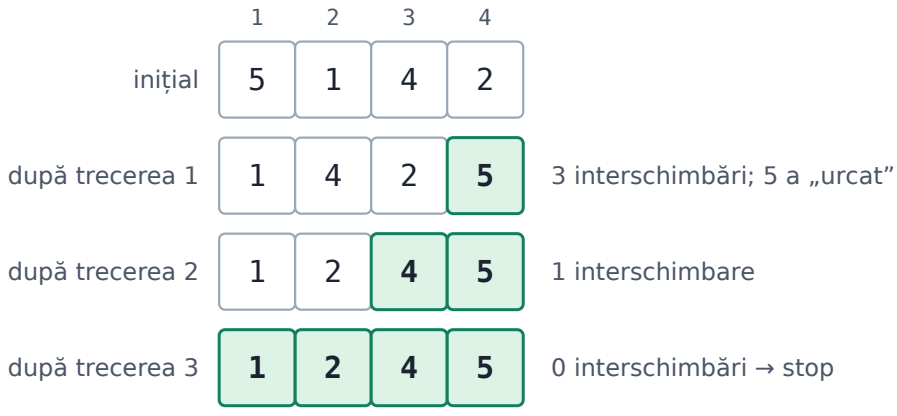
A **sorta** (ordona) un vector înseamnă a-i rearanja elementele crescător (sau descrescător). Toate metodele simple se bazează pe **comparații** și **interschimbări** (`swap`).

DE CE AȘA?

De ce sortăm? Un vector sortat face multe lucruri ușoare: căutarea binară (lecția următoare), eliminarea duplicatelor, găsirea medianei, perechi apropiate, interclasarea. De multe ori „sortează întâi” transformă o problemă $O(n^2)$ într-una $O(n \log n)$.

2. Metoda bulelor (Bubble Sort)

Comparăm perechi de elemente vecine și le interschimbăm dacă sunt în ordine greșită. Repetăm până când nu mai facem nicio interschimbare. Elementele mari „urcă” spre final ca bulele într-un pahar cu sifon.



Metoda bulelor

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int a[1001], n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    bool sortat;
    do
    {
        sortat = true;
        for (int i = 1; i < n; i++)
            if (a[i] > a[i + 1])
            {
                swap(a[i], a[i + 1]);
                sortat = false;
            }
    } while (!sortat);
    for (int i = 1; i <= n; i++)
        cout << a[i] << ' ';
    return 0;
}
```

Urmărire pe `(5, 1, 4, 2)`:

Trecere	Vector după trecere	Interschimbări
1	1 4 2 **5**	3
2	1 2 **4 5**	1
3	1 2 4 5	0 → stop

DE CE AȘA?

De ce se opresc „bulele" după `sortat == true`? Dacă o trecere întreagă nu a găsit nicio pereche în ordine greșită, atunci toate perechile vecine sunt în ordine, deci vectorul este sortat. Fără indicatorul `sortat` ai face mereu `n - 1` treceri, chiar și pe un vector deja sortat. După trecerea `k`, ultimele `k` elemente sunt pe locul lor definitiv.

3. Sortarea prin selecție (Selection Sort)

Pentru fiecare poziție `i`, găsim minimul din restul vectorului și îl aducem pe poziția `i`.



după fiecare pas, prefixul verde este definitiv sortat

Selecție

```
for (int i = 1; i < n; i++)
    for (int j = i + 1; j <= n; j++)
        if (a[j] < a[i])
            swap(a[i], a[j]);
```

(Varianta de mai sus interschimbă la fiecare element mai mic; varianta „clasică” reține poziția minimului și face o singură interschimbare pe pas.)

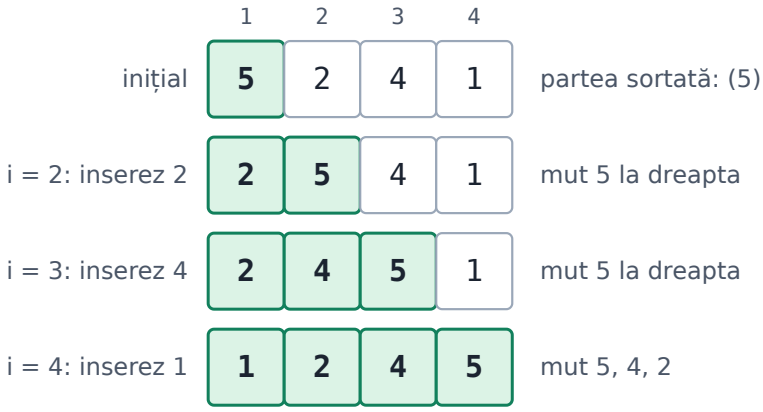
i	j	a[j] < a[i]?	vector după
1	2	2 < 5 da	2 5 4 1
1	3	4 < 2 nu	2 5 4 1
1	4	1 < 2 da	1 5 4 2
2	3	4 < 5 da	1 4 5 2
2	4	2 < 4 da	1 2 5 4
3	4	4 < 5 da	1 2 4 5

DE CE AȘA?

De ce poziția `i` primește mereu minimul? Când `j` termină de parcurs restul, `a[i]` a fost comparat cu fiecare element din dreapta și a „ținut” cea mai mică valoare întâlnită. Prefixul `a[1..i]` este definitiv: conține cele mai mici `i` valori, în ordine.

4. Sortarea prin inserție (Insertion Sort)

Ca la sortarea cărților de joc în mână: luăm pe rând fiecare element și îl inserăm la locul lui în partea deja sortată.



Inserție

```cpp
#include <iostream>
using namespace std;

int a[1001], n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 2; i <= n; i++)
    {
        int x = a[i], j = i - 1;
        while (j >= 1 && a[j] > x)
        {
            a[j + 1] = a[j];     // mutăm spre dreapta
            j--;
        }
        a[j + 1] = x;
    }
    for (int i = 1; i <= n; i++)
        cout << a[i] << ' ';
    return 0;
}
```

DE CE AȘA?

De ce se salvează `x = a[i]` înainte? Mutările la dreapta suprascriu `a[i]` (prima mutare e `a[i] = a[i-1]`). Fără copie pierdem elementul pe care vrem să-l inserăm. Este aceeași idee ca „paharul gol” de la interschimbare.

5. Comparație

Metodă	Comparații (cel mai rău caz)	Observație
bule	~n^2	se oprește devreme dacă vectorul e aproape sortat

Metodă	Comparații (cel mai rău caz)	Observație
selecție	~$n^2/2$	număr fix de comparații, puține interschimbări
inserție	~$n^2/2$	foarte rapidă pe vectori aproape sortați
`sort()` (STL)	~n log n	cea mai rapidă; folosește-o când nu ți se cere explicit metoda

Pentru n = 10^5: metodele de mână fac ~10^{10} operații (zeci de secunde), iar `sort` ~$2 \cdot 10^6$ (instant).

> **LA BAC**
>
> La Bac se acceptă orice metodă de sortare corectă, dar la III.3 (eficiență) o sortare $O(n^2)$ pe 10^6 numere nu mai este eficientă. Deseori problema nu cere sortare deloc: caută o soluție cu vector de frecvență sau „din mers".

6. Sortarea după un criteriu

Schimbi doar **condiția** de comparare.

```
// descrescător
if (a[i] < a[i + 1]) swap(a[i], a[i + 1]);

// crescător după suma cifrelor; la egalitate, crescător după valoare
if (sc(a[i]) > sc(a[i + 1]) || (sc(a[i]) == sc(a[i + 1]) && a[i] > a[i + 1]))
    swap(a[i], a[i + 1]);
```

(`sc` = o funcție care calculează suma cifrelor; le învățăm la subprograme. Până atunci poți calcula sumele într-un vector separat și să le interschimbi împreună.)

> **DE CE AȘA?**
>
> **De ce funcționează la fel pentru orice criteriu?** Algoritmul nu „știe" ce înseamnă „mai mic": îl întreabă pe `if`. Orice relație care spune „a trebuie să fie înaintea lui b" e un criteriu valid, cu condiția să fie coerentă (să nu se contrazică).

Sortarea doar a unor elemente

Exemplu: ordonează crescător **doar elementele pare**, celelalte rămân pe loc. Idee: extragi elementele pare într-un vector auxiliar, îl sortezi, apoi le pui înapoi pe pozițiile elementelor pare, în ordine.

7. `sort()` din STL

```
#include <algorithm>
// ...
sort(a + 1, a + n + 1);                    // sortează a[1..n] crescător
sort(a + 1, a + n + 1, greater<int>());    // descrescător
```

DE ȘTIUT

La exercițiile de pe pbinfo în care scrie explicit „folosind metoda bulelor”, scrie metoda de mână: scopul e să înveți algoritmul. La Bac, folosirea lui `sort` este acceptată dacă problema nu cere o anumită metodă.

8. Greșeli frecvente

Greșeala	Efect	Corect
`for (i = 1; i <= n; i++)` cu `a[i+1]`	citești `a[n+1]`	`i < n`
interschimbare fără al treilea element	ambele ajung egale	`swap` sau `aux`
`sort(a, a + n)` pe vector indexat de la 1	sortezi cu `a[0]` inclus	`sort(a + 1, a + n + 1)`
uiți indicatorul `sortat`	merge, dar mereu lent	`sortat = false` la interschimbare
sortare $O(n^2)$ pe 10^6 elemente	timp depășit	`sort()` sau altă idee

9. La BAC

LA BAC

În grile apar „care secvență ordonează descrescător tabloul” și „vectorul după prima trecere”. Fă tabel cu starea vectorului după fiecare pas. La Subiectul III cu sortare, alege `sort()`, dacă enunțul nu impune altceva.

10. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Se aplică metoda bulelor pe `(6, 3, 8, 1, 5)`. Care e vectorul după prima trecere completă?

2. Câte interschimbări face selecția (varianta din secțiunea 3) pe `(4, 3, 2, 1)`?

3. Ordonează un vector astfel încât elementele pare să fie la început (crescător), iar cele impare la final (descrescător).

4. Câte treceri face metoda bulelor (cu indicator) pe `(2, 1, 3, 4, 5)`? Câte pe `(5, 4, 3, 2, 1)`?

5. Ce afișează `for (i = 1; i < n; i++) if (a[i] > a[i + 1]) swap(a[i], a[i + 1]);` pentru `(3, 1, 2)` (o singură trecere)?

6. Care dintre secvențele următoare ordonează descrescător vectorul `a[1..n]`?

a) `for (i=1;i<n;i++) for (j=i+1;j<=n;j++) if (a[i]<a[j]) swap(a[i],a[j]);`
b) `for (i=1;i<n;i++) for (j=i+1;j<=n;j++) if (a[i]>a[j]) swap(a[i],a[j]);`

c) `sort(a+1, a+n+1);`

d) `sort(a+1, a+n+1, greater<int>());`

7. Care este avantajul sortării prin inserție față de bule pe un vector aproape sortat?

11. Mai departe (pentru cine vrea mai mult)

Sortare prin numărare (Counting Sort). Dacă valorile sunt mici ($\leq 10^6$), nu compari deloc: numeri cât de des apare fiecare valoare și le scrii în ordine. Timp O(n + V). Este ideea vectorului de frecvență din lecția 6.

Merge sort și quick sort. Sortările rapide (O(n log n)) se bazează pe **divide et impera**: împart vectorul, sortează părțile, le combină. Interclasarea din lecția următoare este chiar „combinarea" din merge sort. Le vei scrie în clasa a XI-a.

Sortare cu comparator propriu. `sort(a+1, a+n+1, [](int x, int y){ return sc(x) < sc(y); });` sortează după orice criteriu, fără să rescrii algoritmul. Funcțiile lambda sunt o unealtă comună în C++ modern.

Stabilitate. O sortare e *stabilă* dacă elementele egale rămân în ordinea inițială. Bulele și inserția sunt stabile, selecția cu interschimbări nu. Contează când sortezi înregistrări (nume) după un criteriu (nota), păstrând ordinea alfabetică dintre cei cu aceeași notă.

Rezumat

- Bule: vecini + repetare până nu mai e nicio interschimbare.
- Selecție: minimul din rest pe poziția i.
- Inserție: inserezi a[i] în partea sortată din stânga.
- Criteriul de sortare = condiția din `if`. `sort()` pentru viteză.
- Cele trei metode simple sunt $O(n^2)$; `sort` e O(n log n).

Probleme de lucru (pbinfo)

Începe cu acestea, în ordine: sunt alese să urmeze pașii din lecție.

- [ ] `#509` **ordonare** — ușoară
  Scrie bulele de mână, fără sort().
  https://www.pbinfo.ro/probleme/509/ordonare
- [ ] `#129` **sortare** — ușoară
  Descrescător, din fișier.
  https://www.pbinfo.ro/probleme/129/sortare
- [ ] `#510` **sortprime** — ușoară
  https://www.pbinfo.ro/probleme/510/sortprime

Temă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] #511 **ksort** — ușoară
  https://www.pbinfo.ro/probleme/511/ksort
- [ ] #512 **sortmax** — ușoară
  https://www.pbinfo.ro/probleme/512/sortmax
- [ ] #4146 **sort10** — ușoară
  https://www.pbinfo.ro/probleme/4146/sort10
- [ ] #130 **sortcif** — ușoară
  Criteriu: suma cifrelor.
  https://www.pbinfo.ro/probleme/130/sortcif
- [ ] #183 **sortpp** — medie
  Sortezi doar o parte, restul rămân pe loc.
  https://www.pbinfo.ro/probleme/183/sortpp
- [ ] #514 **sortpie** — medie
  https://www.pbinfo.ro/probleme/514/sortpie
- [ ] #515 **sortsd** — medie
  https://www.pbinfo.ro/probleme/515/sortsd

Pentru cei care vor mai mult

Probleme mai grele sau de concurs, pentru cine vrea să meargă dincolo de BAC.

- [ ] #1608 **sortare-divizori** — medie
  https://www.pbinfo.ro/probleme/1608/sortare-divizori
- [ ] #533 **aliniere** — medie
  https://www.pbinfo.ro/probleme/533/aliniere

Grile din subiectele de BAC (pbinfo)

Exerciții de tip grilă din variantele de Bacalaureat, rezolvate online pe pbinfo, cu verificare automată a răspunsului.

- [ ] #1038 **Exercițiul #1038**
  care secvență ordonează descrescător elementele unui tablou
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1038
- [ ] #1036 **Exercițiul #1036**
  parcurgere cu condiție: produsul elementelor pare
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1036
- [ ] #1039 **Exercițiul #1039**
  efectul unor modificări succesive pe un vector
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1039

Citește și exersează în plus

- Sortarea tablourilor https://www.pbinfo.ro/articole/5587/sortarea-tablourilor
- Metoda bulelor https://www.pbinfo.ro/articole/5589/metoda-bulelor
- Sortarea prin selecție https://www.pbinfo.ro/articole/5605/sortarea-prin-selectie
- Sortarea prin inserție https://www.pbinfo.ro/articole/5609/sortarea-prin-insertie
- VisuAlgo — Sorting — animații pas cu pas pentru bule, selecție, inserție și altele; poți introduce propriul vector https://visualgo.net/en/sorting
- Toptal — Sorting Algorithms Animations — compară vizual viteza algoritmilor pe date aleatoare, aproape sortate, inversate https://www.toptal.com/developers/sorting-algorithms
- cppreference — std::sort — referința pentru sortarea din biblioteca standard https://en.cppreference.com/w/cpp/algorithm/sort

Răspunsuri

Exersează pe hârtie

1. `(3, 6, 1, 5, 8)`. Comparări: 6 și 3 (interschimb → 3 6 8 1 5); 6 și 8 (nu); 8 și 1 (interschimb → 3 6 1 8 5); 8 și 5 (interschimb → 3 6 1 5 8).

2. 6 interschimbări: pentru i = 1 sunt 3 (4↔3, 3↔2, 2↔1), pentru i = 2 sunt 2, pentru i = 3 este 1: total 6.

3. Criteriul: `a[i]` trebuie să fie înaintea lui `a[i+1]` dacă ambele sunt pare și `a[i] < a[i+1]`, sau `a[i]` e par și `a[i+1]` impar, sau ambele impare și `a[i] > a[i+1]`. Interschimbi când nu e îndeplinit:

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int a[1001], n;

bool inainte(int x, int y)       // true dacă x trebuie să fie înaintea lui y
{
    bool px = x % 2 == 0, py = y % 2 == 0;
    if (px != py) return px;     // parul înaintea imparului
    if (px) return x <= y;       // pare crescător
    return x >= y;               // impare descrescător
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i < n; i++)
        for (int j = i + 1; j <= n; j++)
            if (!inainte(a[i], a[j]))
                swap(a[i], a[j]);
    for (int i = 1; i <= n; i++)
        cout << a[i] << ' ';
    return 0;
}
```

4. Pe `(2, 1, 3, 4, 5)`: **2** treceri (prima interschimbă 2 cu 1, a doua nu găsește nimic). Pe `(5, 4, 3, 2, 1)`: **5** (patru treceri cu interschimbări + una de verificare).

5. `(1, 2, 3)`: 3 > 1 interschimbare → (1, 3, 2); 3 > 2 interschimbare → (1, 2, 3).

6. a) și d). Varianta a) pune pe poziția `i` maximul; b) și c) sortează crescător.

7. Pe un vector aproape sortat, inserția face foarte puține mutări (fiecare element e deja aproape de locul lui): se apropie de O(n), pe când bulele fac tot câte o trecere de verificare pentru fiecare element aflat „în urmă”.