AVANSAT · OLIMPIADĂ · LECȚIA 2

Complexitate, doi pointeri și fereastra glisantă

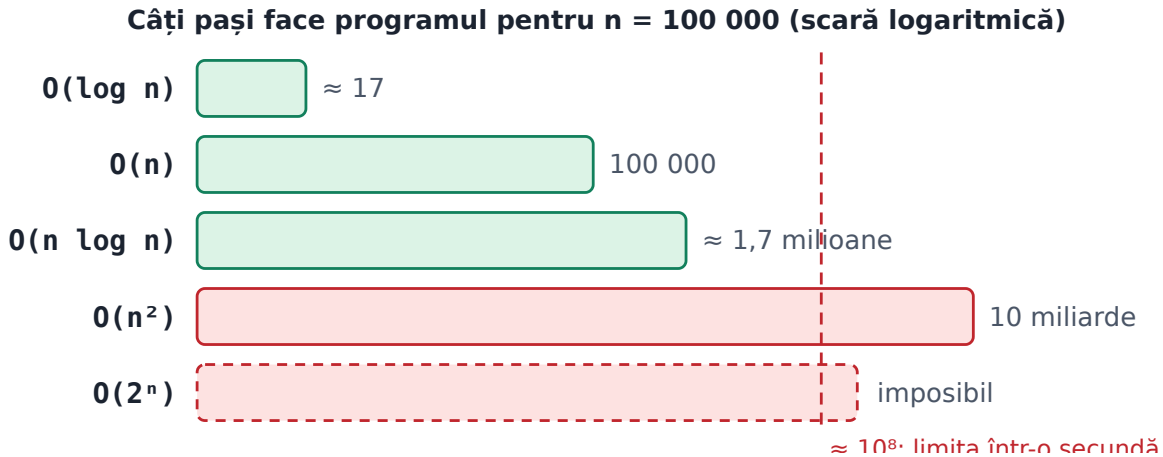
Durată: 75 min Nivel: olimpiadă / informatică intensiv Înainte: lecțiile co-01, x-07

CE VEI ȘTI DUPĂ LECȚIE

- Estimezi din limitele problemei ce complexitate este acceptabilă (și respingi din start soluțiile prea lente).
- Aplici tehnica celor doi pointeri pe un vector sortat.
- Folosești fereastra glisantă pentru secvențe cu o proprietate (sumă, număr de valori distincte).
- Argumentezi de ce un algoritm cu doi pointeri este liniar, deși are o buclă în buclă.

1. Cât de repede e „destul de repede”?

La olimpiadă, o soluție corectă dar lentă primește **punctaj parțial**. Înainte să scrii cod, trebuie să afli *cât de multe operații* are voie să facă programul. Regula practică: un calculator face în jur de **10^8 operații simple pe secundă**. Limita de timp este de obicei 1–2 secunde.



Creșterea numărului de pași

Din **limita lui n** din enunț deduci complexitatea necesară:

n maxim	Complexitate acceptată	Exemple de algoritmi
≤ 10	O(n!) sau O($2^n \cdot n$)	backtracking pe permutări
≤ 20	O(2^n)	submulțimi, bitmask
≤ 500	O(n^3)	Floyd–Warshall
≤ 5 000	O(n^2)	două bucle imbricate simple
≤ 10^5 – 10^6	O(n log n)	sortare, set, căutare binară
≤ 10^7	O(n)	parcurgere, doi pointeri, ciur
≤ 10^{18}	O(log n) sau O(1)	exponențiere rapidă, formule

> **DE CE AȘA?**
>
> **De ce contează doar „ordinul de mărime”?** Pentru $n = 10^5$, $O(n^2)$ înseamnă 10^{10} pași (aproximativ 100 de secunde), iar $O(n \log n)$ înseamnă aproximativ $1{,}7 \cdot 10^6$ pași (sub o centisecundă). Diferența este de peste 5 000 de ori, mult mai mare decât orice optimizare „de detaliu” (un `+` în plus, un `int` în loc de `long long`). De aceea, **întâi se alege algoritmul, apoi se scrie cod**.

Cum numeri pașii. O buclă de la 1 la n costă n. Două bucle imbricate, fiecare de la 1 la n, costă n^2. O buclă care înjumătățește de fiecare dată (căutarea binară) costă $\log_2 n$. Ordinul cel mai mare „câștigă”: $n^2 + n \log n + n$ este $O(n^2)$.

2. Un exemplu: perechea cu suma S

Problemă. Se dă un vector de n numere și o sumă S. Există două elemente (de pe poziții diferite) cu suma S?

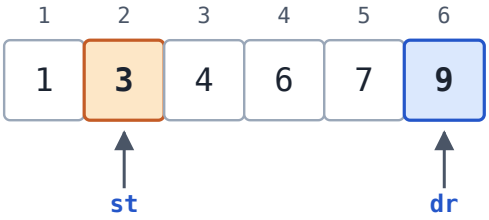
Soluția simplă. Două bucle: pentru fiecare i, pentru fiecare j > i, verifici `v[i] + v[j] == S`. Costă $O(n^2)$: pentru $n = 10^5$ este prea lent.

Idee. Dacă vectorul este **sortat**, putem pune câte un „deget” (pointer) la fiecare capăt:

- suma `v[st] + v[dr]` este **prea mică** → trebuie ceva mai mare: mutăm `st` spre dreapta (valoare mai mare);
- suma este **prea mare** → mutăm `dr` spre stânga (valoare mai mică);
- suma este **egală** cu S → am găsit perechea.



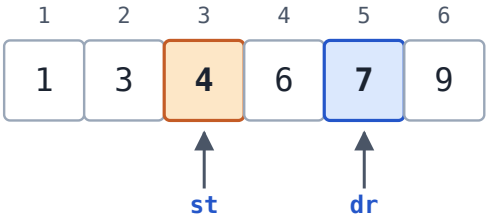
1 + 9 = 10 < 11 → st++



3 + 9 = 12 > 11 → dr--



3 + 7 = 10 < 11 → st++



4 + 7 = 11 = S → gata

Doi pointeri

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

int main()
{
    int n;
    long long S;
    cin >> n >> S;
    vector<long long> v(n);
    for (auto &x : v) cin >> x;
    sort(v.begin(), v.end());
    int st = 0, dr = n - 1;                 // un capăt la început, altul la sfârșit
    while (st < dr)
    {
        long long sum = v[st] + v[dr];
        if (sum == S) { cout << v[st] << ' ' << v[dr]; return 0; }
        if (sum < S) st++;                  // suma e prea mică: mărim cel mai mic
        else dr--;                          // suma e prea mare: micșorăm cel mai mare
    }
    cout << "nu exista";
    return 0;
}
```

Pe vectorul sortat 1 3 4 6 7 9 cu S = 11: (1, 9) → 10 prea mic; (3, 9) → 12 prea mare; (3, 7) → 10 prea mic; (4, 7) → 11. Găsit.

DE CE AȘA?

De ce putem arunca elementele pe care „sărim”? Când `v[st] + v[dr] < S`, perechea lui `v[st]` cu **orice** element mai mic sau egal cu `v[dr]` este și mai mică, deci `v[st]` nu mai are nicio șansă cu elementele rămase: îl eliminăm (`st++`). Raționamentul simetric pentru `dr--`. **Fiecare pas elimină un element**, deci sunt cel mult n pași după sortare: **O(n log n)** în total, din care sortarea este partea cea mai scumpă.

3. Fereastra glisantă

Unele probleme cer o **secvență de elemente consecutive** (un „interval” `[st, dr]`) care respectă o proprietate. Metoda: menținem o „fereastră” și o deplasăm de la stânga la dreapta.

Problemă. Vector de numere **pozitive**, un număr S. Care este lungimea celei mai lungi secvențe cu suma **cel mult S**?

Idee. Pentru fiecare capăt dreapta `dr`:

1. adăugăm `v[dr]` la sumă (fereastra se extinde);
2. cât timp suma depășește S, scoatem `v[st]` și mărim `st` (fereastra se strânge din stânga);
3. acum `[st, dr]` este cea mai lungă secvență validă care se termină în `dr`; actualizăm răspunsul.



fereastra [st, dr] (verde) se extinde la dreapta și se strânge din stânga; cea mai lungă are 4 elemente

Fereastra glisantă

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

int v[100005];

int main()
{
    int n;
    long long S;
    cin >> n >> S;
    for (int i = 1; i <= n; i++) cin >> v[i];
    long long sum = 0;                      // suma ferestrei [st, dr]
    int st = 1, best = 0;
    for (int dr = 1; dr <= n; dr++)
    {
        sum += v[dr];                       // fereastra se extinde la dreapta
        while (sum > S && st <= dr)         // prea mare: o strângem din stânga
        {
            sum -= v[st];
            st++;
        }
        best = max(best, dr - st + 1);      // fereastra [st, dr] are suma <= S
    }
    cout << best;
    return 0;
}
```

DE CE AȘA?

De ce merge doar cu numere pozitive? Aici folosim faptul că, dacă scoatem un element din stânga, suma **scade**; deci „prea mare" se repară doar strângând fereastra. Dacă ar exista și numere negative, scoaterea unui element ar putea să *mărească* suma și metoda s-ar strica (pentru acele cazuri folosești sumele parțiale, în lecția 3). **De ce nu este O(n²), deși `while` stă într-un `for`?** `st` doar **crește** și nu depășește n; `dr` doar crește și nu depășește n. În total, cele două capete fac cel mult 2n deplasări: **O(n)**. Aceasta este analiza **amortizată**: nu contează cât face o singură iterație, ci totalul.

4. Fereastra cu valori distincte

Aceeași idee, cu altă proprietate: cea mai lungă secvență cu **cel mult k valori distincte**. Ținem un vector de frecvențe `cnt[]` pentru valorile din fereastră și un contor `distincte`.

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

int v[100005], cnt[100005];

int main()
{
    int n, k;
    cin >> n >> k;
    for (int i = 1; i <= n; i++) cin >> v[i];
    int distincte = 0, st = 1, best = 0;
    for (int dr = 1; dr <= n; dr++)
    {
        if (cnt[v[dr]]++ == 0) distincte++;          // valoare nouă în fereastră
        while (distincte > k)
        {
            if (--cnt[v[st]] == 0) distincte--;      // o valoare a ieșit complet
            st++;
        }
        best = max(best, dr - st + 1);
    }
    cout << best;
    return 0;
}
```

Pe șirul 1 2 1 3 3 2 2 1 cu k = 2, cea mai lungă secvență cu cel mult două valori distincte este 3 3 2 2 (4 elemente).

DE CE AȘA?

De ce `cnt[v[dr]]++ == 0`? Expresia compară valoarea **dinainte** de incrementare cu 0: dacă era 0, valoarea tocmai a apărut în fereastră și `distincte` crește. La scoatere, `--cnt[...] == 0` verifică valoarea **de după** decrementare: dacă a ajuns la 0, nicio apariție nu a mai rămas în fereastră. **Ce facem dacă valorile sunt mari (până la 10^9)?** Înlocuim vectorul `cnt` cu un `map<int,int>` (lecția 1): complexitatea devine O(n log n).

5. Cum recunoști problemele de tip „doi pointeri”

Indiciu din enunț	Tehnica
„perechi cu o sumă dată”, „perechi cu diferența d” (vector sortat)	doi pointeri la capete sau în aceeași direcție
„cea mai lungă / cea mai scurtă secvență cu proprietatea P”	fereastră glisantă
„numără secvențele care …” cu elemente pozitive	fereastră glisantă; pentru fiecare dr, câte valori st sunt bune
interclasarea a două șiruri sortate	doi pointeri, câte unul în fiecare șir
proprietatea se **păstrează** când fereastra se **micșorează**	merge fereastra glisantă

DE ȘTIUT

Condiția-cheie: dacă o secvență `[st, dr]` este bună, atunci orice subsecvență a ei este bună (**monotonie**). Pentru „suma ≤ S” cu numere pozitive și pentru „cel mult k valori distincte” acest lucru este adevărat; pentru „suma exact S” cu numere pozitive se poate adapta, dar cu numere oarecare nu.

6. Greșeli frecvente

Greșeala	Ce se întâmplă	Corect
doi pointeri pe vector **nesortat**	se pierd perechi	sortezi mai întâi
`while (st <= dr)` în loc de `st < dr` la perechi	un element se împerechează cu el însuși	`st < dr`
suma perechii în `int`	depășire pentru valori mari	`long long`
uiți `st <= dr` în bucla de strângere	`st` poate trece de `dr` când `v[dr] > S`	condiție dublă
fereastră glisantă cu valori negative	suma nu mai este monotonă	sume parțiale
numeri pașii pe o iterație, nu în total	crezi greșit că e $O(n^2)$	analiza amortizată
`best = dr - st` (fără +1)	lungime cu 1 mai mică	`dr - st + 1`

7. La olimpiadă

Subiecte pentru clasele IX–XI cer des „cea mai lungă secvență cu …”, „numărul de perechi cu …” sau „cel mai mic interval care conține …”. Soluția de 100 de puncte este aproape întotdeauna $O(n)$ sau $O(n \log n)$ și folosește una dintre tehnicile din lecție. Soluția $O(n^2)$ aduce de obicei 30–50 de puncte: nu o abandona, dar caută mereu pasul de optimizare.

8. Exersează pe hârtie

1. Câți pași (aproximativ) face O(n log n) pentru n = 10^6? Încape într-o secundă?

2. Un program are trei bucle: una de la 1 la n, apoi două imbricate de la 1 la n. Ce complexitate are în total?

3. Pe vectorul sortat 2 5 8 11 15 20 și S = 26, urmărește pointerii `st` și `dr` și spune ce pereche găsește.

4. Pe același vector și S = 7, spune ce afișează programul și după câți pași.

5. Pentru v = 3 1 2 1 1 și S = 4 (numere pozitive), completează pentru fiecare `dr` valoarea lui `st`, a sumei și a lungimii ferestrei.

6. De ce, în algoritmul cu fereastră glisantă, `st` nu poate depăși `dr + 1`?

7. Cea mai lungă secvență cu cel mult 1 valoare distinctă în 4 4 5 5 5 4: ce lungime are?

8. Modifică algoritmul din secțiunea 3 ca să numere **câte** secvențe au suma ≤ S, nu lungimea celei mai lungi. (Indiciu: pentru fiecare dr, secvențele valide terminate în dr sunt `dr − st + 1`.)

9. De ce O(n^2) este acceptabil pentru n = 3 000, dar nu pentru n = 10^5?

10. Dă un exemplu de vector cu numere negative pe care fereastra glisantă „suma ≤ S" dă un răspuns greșit.

9. Mini-grile

1. Pentru n = $2 \cdot 10^5$ și limita 1 s, care complexitate este potrivită? a) O(n^2) b) O(n log n) c) O(2^n) d) O(n!)

2. Într-un algoritm cu doi pointeri pe un vector sortat, suma `v[st] + v[dr]` este mai mică decât S. Ce faci? a) `dr--` b) `st++` c) `st--` d) te oprești

3. De ce fereastra glisantă este O(n) deși are `while` în `for`? a) pentru că `while` nu se execută niciodată b) pentru că st și dr cresc doar și sunt cel mult n fiecare c) pentru că n este mic d) nu este O(n)

4. Ce pas lipsește pentru perechea cu suma S pe un vector oarecare? a) nimic b) sortarea c) inversarea vectorului d) un al treilea pointer

10. Mai departe (pentru cine vrea mai mult)

- **Trei numere cu suma S:** fixezi un element și aplici doi pointeri pe restul: O(n^2), față de O(n^3) naiv.
- **Interclasare** ca doi pointeri (lecția X.5), și **numărarea perechilor cu diferență d**.
- **Coadă monotonă (deque):** maximul ferestrei de lungime k în O(n) (problema „sliding window maximum").
- **Analiză amortizată** pentru stiva cu `push` și `multipop`.

Rezumat

- Din limita lui n deduci complexitatea necesară; ≈ 10^8 operații pe secundă.

- **Doi pointeri:** pe vector sortat, mișcă capătul care îmbunătățește suma: O(n) după sortare.
- **Fereastră glisantă:** extinzi la dreapta, strângi la stânga; răspunsul se actualizează pentru fiecare dr.
- Funcționează când proprietatea este **monotonă**: dacă secvența e bună, subsecvențele ei sunt bune.
- Analiza **amortizată** explică de ce `while`-ul din `for` nu face algoritmul pătratic.

Probleme de lucru (concurs)

Probleme de pe infoarena, CSES, Kilonova sau pbinfo, în ordinea dificultății. Trimite soluțiile pe platformă și bifează ce ai terminat.

- [ ] CSES **Sum of Two Values** — ușoară
  perechea cu suma x: doi pointeri pe vectorul sortat (cu indici originali)
  https://cses.fi/problemset/task/1640
- [ ] CSES **Playlist** — medie
  cea mai lungă secvență cu valori distincte
  https://cses.fi/problemset/task/1141
- [ ] CSES **Maximum Subarray Sum** — medie
  suma maximă a unei secvențe (O(n))
  https://cses.fi/problemset/task/1643

Temă (concurs)

Mai grele: încearcă-le după ce le-ai terminat pe cele de mai sus.

- [ ] CSES **Subarray Sums II** — dificilă
  ce se întâmplă când valorile pot fi și negative? (sume parțiale + map, lecția 3)
  https://cses.fi/problemset/task/1661
- [ ] CSES **Ferris Wheel** — medie
  greedy + doi pointeri
  https://cses.fi/problemset/task/1090

Citește și exersează în plus

- USACO Guide — Two Pointers — tehnica pe probleme de concurs, cu rezolvări (engleză) https://usaco.guide/silver/two-pointers
- CP-Algorithms — Binary exponentiation și complexitate — colecție de articole pentru algoritmi și analiza lor (engleză) https://cp-algorithms.com/
- Big-O Cheat Sheet — complexitățile structurilor și algoritmilor uzuali într-o singură pagină (engleză) https://www.bigocheatsheet.com/

Răspunsuri

Exersează pe hârtie

1. $\log_2 10^6 \approx 20$, deci $\approx 2 \cdot 10^7$ pași: încape fără probleme.

2. $O(n) + O(n^2) = \mathbf{O(n^2)}$.

3. (2, 20) = 22 < 26 → st++; (5, 20) = 25 < 26 → st++; (8, 20) = 28 > 26 → dr–; (8, 15) = 23 < 26 → st++; (11, 15) = 26: perechea **11 15**.

4. (2, 20) = 22 > 7 → dr–; (2, 15) → dr–; (2, 11) → dr–; (2, 8) = 10 → dr–; (2, 5) = 7: perechea **2 5**, după 5 verificări.

5. dr = 1: st = 1, suma 3, lungime 1 · dr = 2: st = 1, suma 4, lungime 2 · dr = 3: suma 6 > 4 → scot 3 → suma 3, st = 2, lungime 2 · dr = 4: suma 4, st = 2, lungime 3 · dr = 5: suma 5 > 4 → scot 1 → suma 4, st = 3, lungime 3. Cea mai lungă: **3**.

6. Dacă `st = dr + 1`, fereastra este goală (suma 0 ≤ S), deci `while` se oprește; condiția `st <= dr` împiedică scoaterea unui element care nu e în fereastră.

7. 3 (secvența 5 5 5).

8. La fiecare `dr`, după strângere, adaugi `dr - st + 1` la un contor (`long long`): fiecare secvență validă se termină în exact un `dr`.

9. $3\,000^2 = 9 \cdot 10^6$ (cu mult sub 10^8); $(10^5)^2 = 10^{10}$ (de 100 de ori peste limită).

10. v = 4 5 −5 și S = 4. Toată secvența are suma 4 ≤ 4, deci răspunsul corect este 3. Algoritmul: la dr = 2 suma este 9 > 4 și scoate 4, apoi 5 (suma 0, fereastra goală); la dr = 3 are fereastra {−5} de lungime 1, deci răspunde 1. Cu numere negative, strângerea din stânga nu mai este „sigură”: un element scos ar fi putut fi compensat de unul negativ de mai departe.

Mini-grile

1. b) O(n log n). **2.** b) `st++`. **3.** b). **4.** b) sortarea.