INFORMATICĂ · CLASA A XI-A · METODE DE PROGRAMARE · LECȚIA 5

Metoda Divide et Impera: MergeSort, QuickSort, căutare binară recursivă

Durată: 60 min Nivel: intensiv / recomandat și la neintensiv (apare indirect la recursivitate) Înainte: lecțiile xi-04

CE VEI ȘTI DUPĂ LECȚIE

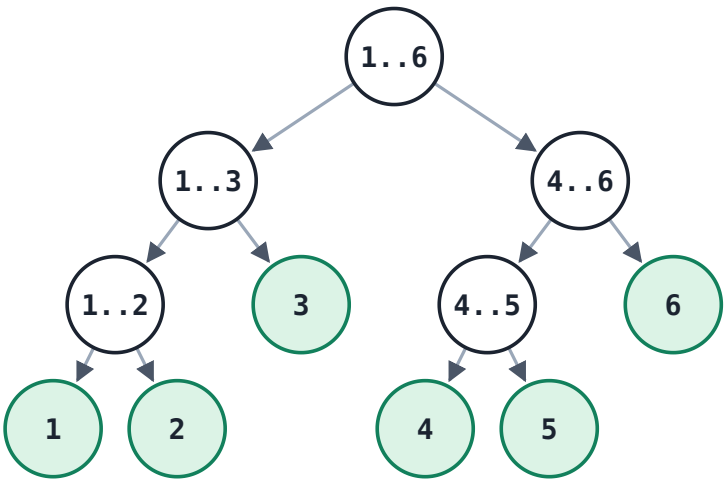
- Înțelegi cei trei pași: împarte, rezolvă subproblemele, combină.
- Scrii funcții divide et impera pe vectori (sumă, maxim, verificări).
- Implementezi căutarea binară recursivă și MergeSort.
- Cunoști ideea QuickSort și problema Turnurilor din Hanoi.

1. Ideea: împarte, rezolvă, combină

Ai de verificat 1000 de lucrări. Dacă le dai în două grămezi unor colegi, iar ei fac la fel mai departe, la final fiecare are doar câteva lucrări, iar tu doar aduni rezultatele. Aceasta este metoda **Divide et Impera** (lat. „împarte și stăpânește"):

1. **Divide:** împarte problema în două (sau mai multe) subprobleme de același tip, mai mici;
2. **Impera:** rezolvă subproblemele recursiv, până la cazuri banale (un singur element);
3. **Combină:** din soluțiile subproblemelor obții soluția întregii probleme.

Pe un vector `a[st..dr]` împărțim la mijloc: `mij = (st + dr) / 2`, adică `a[st..mij]` și `a[mij+1..dr]`.



rde = cazuri de bază (un singur element); în total 6 frunze și 5 împărț

Arborele de apeluri

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

int a[101], n, apeluri;

int suma(int st, int dr)
{
    apeluri++;                                  // numărăm apelurile
    if (st == dr) return a[st];                 // un singur element: banal
    int mij = (st + dr) / 2;
    int s1 = suma(st, mij);                     // subproblema din stânga
    int s2 = suma(mij + 1, dr);                 // subproblema din dreapta
    return s1 + s2;                             // combinare
}

int maxim(int st, int dr)
{
    if (st == dr) return a[st];
    int mij = (st + dr) / 2;
    int m1 = maxim(st, mij), m2 = maxim(mij + 1, dr);
    return m1 > m2 ? m1 : m2;
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    cout << suma(1, n) << ' ' << maxim(1, n) << ' ' << apeluri;
    return 0;
}
```

DE CE AȘA?

De ce atâtea apeluri (2n − 1), dacă o buclă ar face doar n adunări? Pentru suma și maximul metoda D&I nu câștigă nimic: tot n operații se fac. Rostul acestor exemple este să **înțelegi structura**. Câștigul real apare când combinarea e ieftină și problema se împarte bine (căutare binară, sortare), cum vedem mai jos.

LA BAC

Grilă frecventă: „De câte ori se apelează funcția pentru un vector cu n elemente?" Pentru împărțirea în două până la un element, numărul de apeluri este **2n − 1** (n frunze + n − 1 împărțiri). Desenează arborele pentru n = 5 și verifică: 9 apeluri.

Intrare / ieșire

Intrare	Ieșire
`6` `1 2 3 4 5 6`	`21 6 11` (suma, maximul, apelurile)
`1` `7`	`7 7 1`

2. Căutarea binară, recursiv

Aici „combinarea” lipsește: rezolvăm doar **una** dintre jumătăți, fiindcă vectorul este sortat și știm în care se află `x`.

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

int a[101], n;

bool caut(int st, int dr, int x)
{
    if (st > dr) return false;                  // interval vid: nu există
    int mij = (st + dr) / 2;
    if (a[mij] == x) return true;
    if (x < a[mij]) return caut(st, mij - 1, x);
    return caut(mij + 1, dr, x);
}

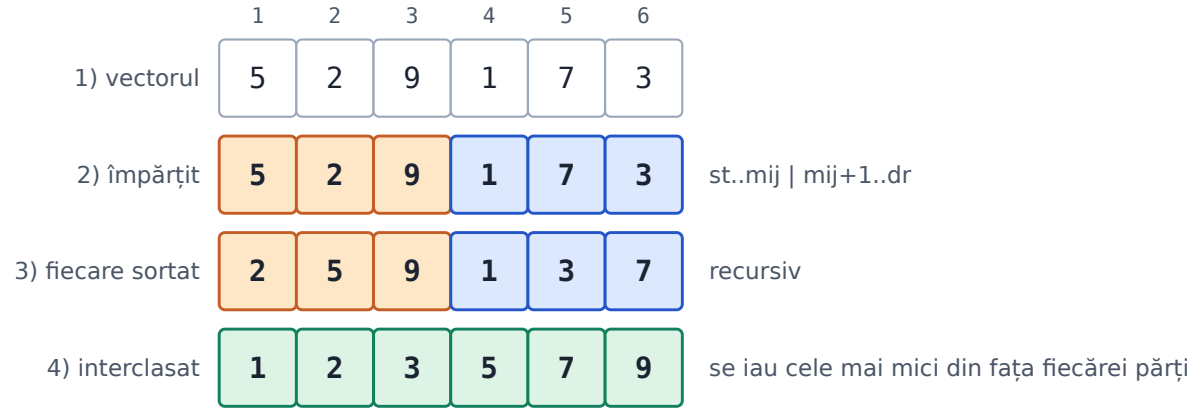
int main()
{
    int x;
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    cin >> x;
    cout << (caut(1, n, x) ? "DA" : "NU");
    return 0;
}
```

Urmărire pentru `x = 12`, `a = (2, 4, 6, 8, 10, 12, 14)`:

apel	st	dr	mij	a[mij]	decizie
`caut(1, 7, 12)`	1	7	4	8	12 > 8: dreapta
`caut(5, 7, 12)`	5	7	6	12	găsit: `true`

3. MergeSort: sortarea prin interclasare

Sortăm cele două jumătăți, apoi le **interclasăm** (ca la lecția X.5): luăm mereu cel mai mic dintre primele elemente ale celor două părți sortate.



MergeSort

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

int a[100001], aux[100001], n;

void mergeSort(int st, int dr)
{
    if (st >= dr) return;                       // 0 sau 1 element: deja sortat
    int mij = (st + dr) / 2;
    mergeSort(st, mij);
    mergeSort(mij + 1, dr);
    int i = st, j = mij + 1, k = 0;             // interclasare în aux
    while (i <= mij && j <= dr)
        if (a[i] <= a[j]) aux[++k] = a[i++];
        else aux[++k] = a[j++];
    while (i <= mij) aux[++k] = a[i++];
    while (j <= dr) aux[++k] = a[j++];
    for (int t = 1; t <= k; t++) a[st + t - 1] = aux[t];    // copiem înapoi
}

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

DE CE AȘA?

De ce `<=` în `if (a[i] <= a[j])` ? La elemente egale luăm întâi pe cel din stânga: sortarea rămâne **stabilă** (elementele egale își păstrează ordinea inițială). **De ce vectorul `aux` ?** Interclasarea nu se poate face pe loc fără să strici elemente încă necitite; scriem rezultatul în `aux` și îl copiem înapoi. **De ce O(n log n)?** Vectorul se împarte de $\log_2 n$ ori (adâncimea arborelui), iar la fiecare nivel interclasările parcurg în total n elemente. Pentru 10^6 elemente sunt ~$2 \cdot 10^7$ pași, față de 10^{12} la bule.

Urmărire pentru `mergeSort(1, 6)` pe `(5, 2, 9, 1, 7, 3)` :

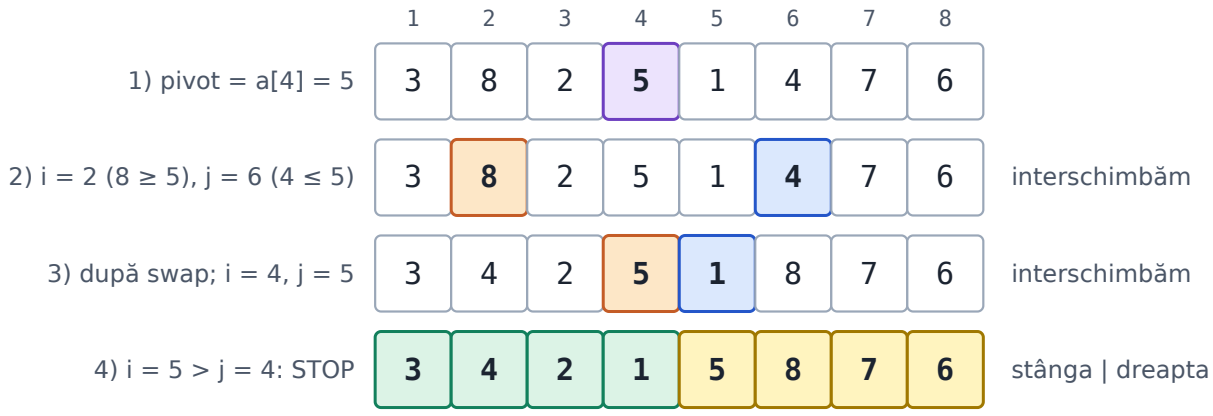
apel	partea sortată după apel
`mergeSort(1, 3)`	`2 5 9`
`mergeSort(4, 6)`	`1 3 7`
interclasare (1, 3) cu (4, 6)	`1 2 3 5 7 9`

Intrare / ieșire

Intrare	Ieșire
`6` `5 2 9 1 7 3`	`1 2 3 5 7 9`
`3` `1 1 1`	`1 1 1`

4. QuickSort (ideea)

Alegem un **pivot**; mutăm spre stânga elementele mai mici decât el și spre dreapta pe cele mai mari (partiționare), apoi sortăm recursiv cele două părți. Aici nu mai e nevoie de combinare: după partiționare, fiecare parte se sortează pe loc.



elementele din stânga sunt ≤ pivot, cele din dreapta ≥ pivot; se sortează recursiv fiecare parte

QuickSort

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

int a[100001], n;

void quickSort(int st, int dr)
{
    int i = st, j = dr, pivot = a[(st + dr) / 2];
    while (i <= j)
    {
        while (a[i] < pivot) i++;               // primul element care NU e mai mic decât pivotul
        while (a[j] > pivot) j--;               // primul (de la dreapta) care NU e mai mare
        if (i <= j) { swap(a[i], a[j]); i++; j--; }
    }
    if (st < j) quickSort(st, j);
    if (i < dr) quickSort(i, dr);
}

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

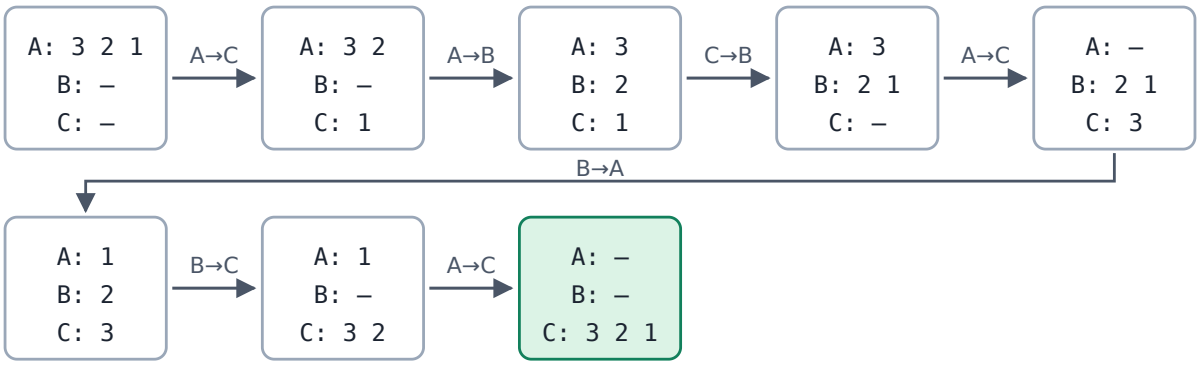
DE CE AȘA?

De ce pivotul din mijloc? Dacă ai alege mereu primul element, un vector deja sortat ar da părți de n − 1 și 0 elemente: O(n²). Pivotul din mijloc se comportă bine pe vectori sortați sau aproape sortați. Pe date „rău intenționate" se alege un pivot aleator. În practică, `sort()` din STL (`<algorithm>`) folosește o variantă îmbunătățită a acestei idei (introsort); la Bac poți folosi `sort`.

	MergeSort	QuickSort
timp (în medie)	O(n log n)	O(n log n)
cel mai rău caz	O(n log n)	O(n²) (rar)
memorie în plus	O(n) (`aux`)	O(log n) (stiva)
stabil	da	nu

5. Turnurile din Hanoi

Mutăm `n` discuri de pe tija A pe C, folosind B, mutând un singur disc o dată și nepunând niciodată un disc mare peste unul mic. Gândește recursiv: pentru a muta cel mai mare disc, trebuie ca celelalte `n - 1` să fie deja pe tija de ajutor.



Hanoi

1. mutăm `n – 1` discuri de pe A pe B (folosind C);
2. mutăm discul cel mare de pe A pe C;
3. mutăm cele `n – 1` discuri de pe B pe C (folosind A).

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

void hanoi(int n, char a, char b, char c)     // mută n discuri de pe a pe c, folosind b
{
    if (n == 0) return;
    hanoi(n - 1, a, c, b);                    // n-1 discuri: a -> b (ajutor: c)
    cout << a << "->" << c << '\n';           // discul cel mare: a -> c
    hanoi(n - 1, b, a, c);                    // n-1 discuri: b -> c (ajutor: a)
}

int main()
{
    int n;
    cin >> n;
    hanoi(n, 'A', 'B', 'C');
    return 0;
}
```

DE CE AȘA?

De ce se inversează rolurile tijelor în apeluri? În primul apel scopul este tija B, iar C devine „ajutorul”; în al doilea, sursa este B și ajutorul este A. Parametrii `(a, b, c)` sunt *roluri* (sursă, ajutor, destinație), nu tije fixe. Numărul de mutări `M(n) = 2·M(n – 1) + 1`, deci $2^n - 1$.

Intrare / ieșire

Intrare	Ieșire
`1`	`A->C`

`2`	`A->B`
	`A->C`
	`B->C`

6. Greșeli frecvente

Greșeala	Efect	Corect
`mij = (st + dr) / 2` cu `st`, `dr` mari și `int`	depășire la valori foarte mari	`st + (dr - st) / 2`
subintervale care se suprapun (`st..mij` și `mij..dr`)	se numără `mij` de două ori	`st..mij` și `mij+1..dr`
lipsește cazul de bază (`st == dr`)	recursie infinită	verifică-l primul
căutare binară pe vector nesortat	rezultat greșit	sortează întâi
`aux` nu se copiază înapoi în `a`	vectorul rămâne nesortat	`for … a[st + t - 1] = aux[t]`
QuickSort cu `<` și `>` greșite (`<=`)	buclă infinită la elemente egale	`<` și `>` stricte

7. La BAC

> **LA BAC**
>
> La Subiectul I apar subprograme D&I cu `st`, `dr`, `mij` pe un vector dat: „ce valoare întoarce apelul?" sau „de câte ori se apelează?". Metoda: desenezi arborele de apeluri cu intervalele `[st, dr]` și completezi de jos în sus. Numărul de apeluri la împărțire în două până la un element: **2n − 1**.

8. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Desenează arborele de apeluri pentru `suma(1, 6)`. Câte apeluri sunt?

2. Câte apeluri face `suma` pentru n = 5? Dar pentru n = 8?

3. Scrie o funcție divide et impera care numără elementele pare dintr-un vector `a[st..dr]`.

4. Câte mutări face `hanoi(5, …)`? Dacă o mutare durează o secundă, cât durează pentru 64 de discuri?

5. Pentru `a = (2, 4, 6, 8, 10, 12, 14)`, scrie apelurile `caut(1, 7, 4)`.

6. Interclasează `(2, 5, 9)` cu `(1, 3, 7)`. Câte comparații se fac între elemente?

7. De ce căutarea binară are nevoie de vector sortat?

8. Estimează numărul de operații pentru MergeSort cu n = 10^6 ($\approx n \cdot \log_2 n$) și compară cu bulele ($\approx n^2/2$).

9. Cum se comportă QuickSort cu pivotul `a[st]` pe un vector deja sortat?

10. (BAC) Ce afișează?

```
int f(int st, int dr, int v[]) { if (st == dr) return v[st]; int mij = (st + dr) / 2;
return f(st, mij, v) + f(mij + 1, dr, v); }
// v = (7, 1, 2, 5, 4) indexat de la 1: cout << f(1, 5, v);
```

9. Mai departe (pentru cine vrea mai mult)

Numărarea inversiunilor. Câte perechi `(i, j)` cu `i < j` și `a[i] > a[j]` are un vector? Soluția naivă e O(n²); cu MergeSort se poate în O(n log n): când la interclasare luăm un element din dreapta, el este mai mic decât **toate** elementele rămase în stânga, deci adăugăm `mij - i + 1` inversiuni.

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

int a[100001], aux[100001], n;
long long inv;

void ms(int st, int dr)
{
    if (st >= dr) return;
    int mij = (st + dr) / 2;
    ms(st, mij); ms(mij + 1, dr);
    int i = st, j = mij + 1, k = 0;
    while (i <= mij && j <= dr)
        if (a[i] <= a[j]) aux[++k] = a[i++];
        else { inv += mij - i + 1; aux[++k] = a[j++]; }       // a[j] e mai mic decât
a[i..mij]
    while (i <= mij) aux[++k] = a[i++];
    while (j <= dr) aux[++k] = a[j++];
    for (int t = 1; t <= k; t++) a[st + t - 1] = aux[t];
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    ms(1, n);
    cout << inv;
    return 0;
}
```

Teorema master (ideea). Dacă o problemă se împarte în 2 subprobleme de mărime n/2 și combinarea costă O(n), timpul este `T(n) = 2T(n/2) + O(n) = O(n log n)`. La căutarea binară: `T(n) = T(n/2) + O(1) = O(log n)`.

Alte aplicații. Înmulțirea rapidă a numerelor mari (Karatsuba), cele mai apropiate două puncte din plan, transformata Fourier rapidă (FFT), căutarea celei de-a k-a valori (quickselect).

Rezumat

- Împarte (`mij`), rezolvă recursiv, combină; cazul de bază: un element.
- Căutare binară: doar o jumătate. MergeSort: ambele jumătăți + interclasare. QuickSort: partiționare, apoi ambele părți.

- Împărțire în două până la un element: $2n - 1$ apeluri.
- MergeSort: O(n log n), stabil, memorie în plus; QuickSort: O(n log n) în medie.
- Hanoi: $2^n - 1$ mutări.

Probleme de lucru (pbinfo)

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

- [ ] #1015 **sumvec** — ușoară
  https://www.pbinfo.ro/probleme/1015/sumvec
- [ ] #1019 **maxim6** — ușoară
  https://www.pbinfo.ro/probleme/1019/maxim6
- [ ] #1155 **cautarebinara** — medie
  https://www.pbinfo.ro/probleme/1155/cautarebinara

Temă (pbinfo)

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

- [ ] #1017 **sumparevec** — ușoară
  https://www.pbinfo.ro/probleme/1017/sumparevec
- [ ] #1018 **cntimpare** — ușoară
  https://www.pbinfo.ro/probleme/1018/cntimpare
- [ ] #1023 **cmmdc3** — ușoară
  https://www.pbinfo.ro/probleme/1023/cmmdc3
- [ ] #1152 **verificareordonatdivimp** — ușoară
  https://www.pbinfo.ro/probleme/1152/verificareordonatdivimp
- [ ] #2527 **hanoi** — ușoară
  https://www.pbinfo.ro/probleme/2527/hanoi
- [ ] #1025 **mergesort** — medie
  https://www.pbinfo.ro/probleme/1025/mergesort

Pentru cei care vor mai mult

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

- [ ] #1024 **quicksort** — medie
  https://www.pbinfo.ro/probleme/1024/quicksort
- [ ] #3206 **nrinversiuni** — medie
  https://www.pbinfo.ro/probleme/3206/nrinversiuni

Grile din subiectele de BAC (pbinfo)

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

- [ ] `#1013` **Exercițiul #1013**
  tablou v cu 8 valori date; subprogram recursiv pe st..dr: ce se obține
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1013

- [ ] `#1014` **Exercițiul #1014**
  tablou a cu 9 valori date; subprogram divide et impera: ce se obține
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1014

- [ ] `#1616` **Exercițiul #1616**
  algoritm în pseudocod cu vector, s și d: secvența de ...
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1616

- [ ] `#1617` **Exercițiul #1617**
  f recursiv pe tabloul (7, 1, 2, 5, 4): valoarea apelului
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1617

- [ ] `#1708` **Exercițiul #1708**
  câte inversiuni conține vectorul A = [5, 3, 2, 4, 1]
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1708

- [ ] `#1350` **Exercițiul #1350**
  f(st, dr, v[]) recursiv: valoarea întoarsă pe un vector dat
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1350

Citește și exersează în plus

- Metoda Divide et Impera https://www.pbinfo.ro/articole/7651/divide-et-impera
- Sortarea prin interclasare – MergeSort https://www.pbinfo.ro/articole/7667/sortarea-prin-interclasare
- Sortarea rapidă – QuickSort https://www.pbinfo.ro/articole/7666/quicksort
- VisuAlgo — Sorting (MergeSort, QuickSort) — animații pas cu pas pentru ambele sortări, cu exemplele tale https://visualgo.net/en/sorting
- Turnurile din Hanoi (joc online) — încearcă singur cu 3, 4, 5 discuri și numără mutările https://www.mathsisfun.com/games/towerofhanoi.html
- CSES — Sorting and Searching — secțiunea „Sorting and Searching”: probleme cu căutare binară și sortări (engleză) https://cses.fi/problemset/list/

Răspunsuri

Exersează pe hârtie

1. Rădăcina `[1, 6]` → `[1, 3]` și `[4, 6]`; `[1, 3]` → `[1, 2]` și `[3, 3]`; `[1, 2]` → `[1, 1]` și `[2, 2]`; `[4, 6]` → `[4, 5]` și `[6, 6]`; `[4, 5]` → `[4, 4]` și `[5, 5]`. Total **11 apeluri** (6 frunze + 5 împărțiri), conform formulei 2·6 − 1.

2. `2·5 − 1 =` **9** apeluri; `2·8 − 1 =` **15**.

3.

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

int a[101], n;

int nrPare(int st, int dr)
{
    if (st == dr) return a[st] % 2 == 0;       // un singur element: 1 dacă e par, altfel 0
    int mij = (st + dr) / 2;
    return nrPare(st, mij) + nrPare(mij + 1, dr);
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    cout << nrPare(1, n);
    return 0;
}
```

4. $2^5 - 1 =$ **31** de mutări. Pentru 64 de discuri: $2^{64} - 1 \approx 1{,}8 \cdot 10^{19}$ secunde ≈ 585 de miliarde de ani.

5. `caut(1, 7, 4)`: `mij = 4`, `a[4] = 8`, `4 < 8` → `caut(1, 3, 4)`: `mij = 2`, `a[2] = 4` → găsit (`true`). Două apeluri.

6. Rezultat `1 2 3 5 7 9`. Comparații: (2, 1) → 1; (2, 3) → 2; (5, 3) → 3; (5, 7) → 5; (9, 7) → 7; apoi partea din dreapta s-a terminat și se copiază restul din stânga (9) fără comparații. Total **5** comparații.

7. Decizia „stânga sau dreapta?" se bazează pe `x < a[mij]`: dacă vectorul nu e sortat, `x` poate fi în ambele părți și jumătatea eliminată ar putea conține elementul căutat.

8. MergeSort: ≈ $10^6 \cdot 20$ = **$2 \cdot 10^7$** operații (milisecunde). Bule: ≈ $10^{12} / 2$ = **$5 \cdot 10^{11}$** operații (minute-ore).

9. Pe un vector sortat, pivotul `a[st]` este cel mai mic element: o parte are 0 elemente, cealaltă n − 1. Adâncimea recursiei devine n și timpul $O(n^2)$, cu risc de stack overflow. Pivotul din mijloc evită cazul acesta.

10. `19`: funcția adună toate elementele (7 + 1 + 2 + 5 + 4 = 19), chiar dacă împarte vectorul în două la fiecare pas.