Metoda Divide et Impera

Capitolul 4

- **Prezentarea metodei**
- **Implementări sugerate**
- **Probleme propuse**
- **Soluțiile problemelor**

4.1. Prezentarea metodei

Divide et Impera (*Împarte şi Stăpâneşte)* este o metodă de programare care se aplică problemelor care pot fi descompuse în subprobleme independente, *similare* problemei inițiale, de dimensiuni mai mici şi care pot fi rezolvate foarte uşor.

Metoda presupune:

Descompunerea problemei inițiale în subprobleme independente;

Rezolvarea subproblemelor;

Construirea rezultatului prin *compunerea soluțiilor* problemelor de dimensiuni mici.

Această metodă se poate implementa atât iterativ cât şi recursiv. Datorită faptului că problemele se împart în subprobleme în mod recursiv, de obicei împărțirea se realizează până când şirul obținut prin împărțire este de lungime 1, caz în care rezolvarea subproblemei este foarte uşoară, chiar banală.

Fie un vector $X = (x_1, x_2, \ldots, x_n)$ asupra căruia se aplică o prelucrare. Pentru orice i, $j \in \{1, 2, \ldots, n\}$ astfel încât $i < j$, există o valoare p, astfel încât prin prelucrarea secvențelor $\{x_i, x_{i+1}, \ldots, x_p\}$ şi $\{x_{p+1}, x_{p+2}, \ldots, x_j\}$ se obțin soluțiile corespunzătoare subşirurilor care prin combinare conduc la soluția prelucrării secvenței $\{x_i, x_{i+1}, \ldots, x_j\}$.

```
Subalgoritm Divide(i,j,rez):
  dacă i < j atunci
    Împarte(i,j,p)
    Divide(i,p,rez1)
    Divide(p+1,j,rez2)
    rez=Combină(rez1,rez2)
  altfel
    Rezolvă(rez)
sfârşit subalgoritm
```

3.2. Implementări sugerate

1. recapitularea algoritmului căutării binare şi realizarea implementării recursive;
2. calcularea produsului a *n* numere cu metoda *Divide et Impera*;
3. determinarea celui mai mare (celui mai mic) număr într-un şir dat cu metoda *Divide et Impera*;
4. plierea unui vector (Se consideră un vector de lungime *n*. Definim plierea vectorului prin suprapunerea unei jumătăţi, numite *donatoare* peste cealaltă jumătate, numită *receptoare*. Dacă numărul de elemente este impar, elementul din mijloc este eliminat. În acest mod se ajunge la un subşir ale cărui elemente au numerotarea corespunzătoare jumătăţii receptoare. a) Fiind dat un indice *i*, să se determine dacă al *i*-lea element poate fi element final ca rezultat al unor plieri succesive. b) Să se precizeze toate elementele finale posibile. c) Fiind dat un element *i* final, să se determine o succesiune de plieri prin care se ajunge la el);
5. sortare prin interclasare (*merge-sort*);
6. sortare rapidă (*quick-sort*);
7. determinarea minimului dintr-un şir, precum şi a indicelelui uneia dintre valorile minime;
8. verificarea dacă un şir de numere este ordonat crescător;
9. determinarea celui mai mare divizor comun al elementelor unui şir;
10. determinarea celui mai mare număr par dintr-un şir dat;
11. înmulţirea a două polinoame folosind metoda *Divide et Impera*;
12. calcularea valorii unui polinom într-un punct dat;
13. ***Raftul de cărţi***. Fie *n* cărţi de matematică, fizică, literatură şi informatică (având ca simboluri M, F, L, şi I) aşezate pe raftul *A* într-o ordine oarecare. Există, de asemenea, două rafturi *B* şi *X* la dispoziţie. Să se transfere cărţile pe rafturile *LF*, *LMI*, *LL* după următoarele reguli:
    - pe rafturile *A* şi *B* se pot pune sau lua oricâte cărţi;
    - pe raftul *X* se poate pune sau lua o singură carte;
    - pe rafturile *LF*, *LMI*, *LL* se pot pune cărţi şi nu se pot lua;
    - la orice mutare avem acces la o singură carte, şi anume la cea din vârful teancului de pe raftul *A*;
    - tipul cărţii se poate citi numai pe raftul *B* şi numai când aici se află o singură carte;
    - în final cărţile de pe rafturile *LF*, *LMI*, *LL* trebuie să fie în aceeaşi ordine în care erau pe raftul A.
14. ***Produsul a două matrice***. Produsul a două matrice presupune un număr mare de operaţii (n^3 înmulţiri şi $n^2 \cdot (n-1)$ adunări, astfel încât complexitatea algoritmului clasic de înmulţire este $O(n^3)$. Folosind metoda Divide et Impera se reduce complexitatea, dacă se foloseşte următoarea formulă:

– $\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$, unde:
– $C_{11} = A_{11} B_{11} + A_{12} B_{21}$, $C_{12} = A_{11} B_{12} + A_{12} B_{22}$,
– $C_{21} = A_{21} B_{11} + A_{22} B_{21}$, $C_{22} = A_{21} B_{12} + A_{22} B_{22}$.
– Se obține astfel matricea rezultat în urma înmulțirii a 8 matrice de ordin $n/2$.

4.3. Probleme propuse

4.3.1. Minim

Determinați valoarea minimă dintr-un şir de numere, folosind metoda *Divide et Impera*.

Date de intrare

Prima linie a fişierului de intrare **MINIM.IN** va conține un număr întreg *n*, care reprezintă numărul de elemente din şir. Pe următoarea linie se află *n* numere întregi, separate prin câte un spațiu, reprezentând valorile elementelor din şir.

Date de ieşire

Fişierul de ieşire **MINIM.OUT** va conține o linie pe care se va afla valoarea minimă care a fost găsită în şir.

Restricții şi precizări

- $1 \le n \le 10000$;
- Valorile din şir sunt numere întregi mai mici decât 30000.

Exemplu

MINIM.IN
```
7
720 212 10 13 150 15 17
```

MINIM.OUT
```
10
```

4.3.2. Produsul numerelor prin metoda *Divide et Impera*

Calculați produsul a *n* numere folosind metoda *Divide et Impera*.

Date de intrare

Prima linie a fişierului de intrare **PRODUS.IN** va conține un număr întreg *n*, care reprezintă numărul de elemente din şir. Pe următoarea linie se află, separate prin spații, *n* numere reale, reprezentând valorile numerice ale elementelor din şir.

Date de ieşire
Fişierul de ieşire **PRODUS.OUT** va conține o linie pe care se va afla numărul real, cu două zecimale exacte, reprezentând produsul elementelor.

Restricții şi precizări
- $1 \le n \le 100$.

Exemplu

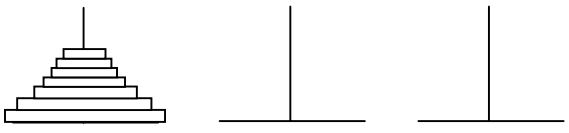
PRODUS.IN	**PRODUS.OUT**
7 10 0.1 100 2 0.1 3 0.1	6.00

4.3.3. Turnurile din Hanoi

Se dau trei tije verticale *A*, *B* şi *C*. Pe tija *A* se găsesc discuri de diametre diferite, perforate la mijloc, aranjate în ordine descrescătoare a diametrelor discurilor, de la bază spre vârf. Celelalte tije sunt goale.

Se cere să se găsească o strategie de mutare a discurilor de pe tija *A* pe tija *B* respectând următoarele reguli:

- La un moment dat se va muta un singur disc (cel care se află deasupra celorlalte discuri pe o tijă).
- Un disc poate fi aşezat doar peste un alt disc având diametru mai mare decât al său, sau pe o tijă goală.



Date de intrare
Fişierului de intrare **HANOI.IN** conține un singur număr natural *n*, care reprezintă numărul de discuri existente pe tija *A*.

Date de ieşire
Fişierul de ieşire **HANOI.OUT** va conține pe fiecare linie două litere separate de un spațiu. Aceste litere reprezintă mutarea care se realizează la un moment dat, şi anume prima literă reprezintă tija de pe care se ia discul, iar a doua literă reprezintă tija pe care se aşează discul respectiv.

Restricții şi precizări
- $0 < n \le 10$.

Exemplu

HANOI.IN
```
3
```

HANOI.OUT
```
A -> B
A -> C
B -> C
A -> B
C -> A
C -> B
A -> B
```

4.3.4. Sortare prin interclasare (*merge-sort*)

Implementați metoda sortării prin interclasare pentru un şir de numere dat.

Date de intrare

Fişierului de intrare **MERGE.IN** conține pe prima linie un număr natural *n***,** care reprezintă numărul de elemente ale şirului. Pe a doua linie, separate prin câte un spațiu, se află cele *n* numere din şir.

Date de ieşire

Fişierul de ieşire **MERGE.OUT** va conține numerele din şirul dat (separate prin câte un spațiu) în ordine crescătoare.

Restricții şi precizări

- $0 \le n \le 10000$;
- Numerele sunt întregi.

Exemplu

MERGE.IN
```
8
23 8 7 17 90 3 17 29
```

MERGE.OUT
```
3 7 8 17 17 23 29 90
```

4.3.5. Sortare rapidă (*quick-sort*)

Implementați metoda *sortării rapide* pentru un şir de numere dat.

Date de intrare

Fişierul de intrare **QUICK.IN** conține pe prima linie numărul *n* al elementelor şirului. Pe a doua linie se află cele *n* numere din şir, separate prin câte un spațiu.

Date de ieşire

Fişierul de ieşire **QUICK.OUT** va conține numerele din şir în ordine crescătoare, separate de câte un spațiu.

Restricții și precizări
- $0 \le n \le 10000$;
- Numerele sunt întregi.

Exemplu

QUICK.IN
```
9
8 5 4 10 12 6 3 9 7
```

QUICK.OUT
```
3 4 5 6 7 8 9 10 12
```

4.3.6. Serbare

Pentru serbarea şcolii profesorul de dans a adus un costum de urs polar care poate fi îmbrăcat doar de un copil care are înălțimea adecvată. El încearcă să găsească elevul potrivit pentru a purta costumul. Profesorul, observând că nu există doi elevi cu aceeaşi înălțime, cere elevilor să se aşeze în ordinea crescătoare a înălțimii lor. Speră astfel ca să găsească mai uşor elevul potrivit.

Cunoscând numărul elevilor, precum şi înălțimile lor în ordine crescătoare, scrieți un program care să determine dacă există un elev cu înălțimea cerută şi care este locul lui în şirul elevilor. Dacă nu există un astfel de elev, determinați locul elevului care are cea mai mică înălțime şi este mai înalt decât înălțimea cerută – adică locul pe care ar sta un elev cu înălțimea potrivită.

Date de intrare

Prima linie a fişierului de intrare **SERBARE.IN** va conține două numere întregi pozitive: înălțimea căutată h în centimetri şi numărul de elevi n. Aceste numere vor fi separate printr-un spațiu. Pe următoarea linie se află n numere întregi, separate prin câte un spațiu, în ordine strict crescătoare, reprezentând înălțimea elevilor în centimetri.

Date de ieşire

Fişierul de ieşire **SERBARE.OUT** va conține o linie pe care se va afla răspunsul `'NU'`, dacă nu există un elev de înălțime potrivită, sau `'DA'` dacă acesta există. Pe următoarea linie se va scrie poziția elevului în şirul ordonat fie că este cel găsit, fie că este a elevului imediat mai mare.

Restricții și precizări
- $60 \le h \le 200$;
- $1 \le n \le 1000$.

Exemple

SERBARE.IN
```
150 7
110  112  140  137  150  151  5
167
```

SERBARE.OUT
```
DA
5
```

```
SERBARE.IN                    SERBARE.OUT
131 9                         NU
110  112  120  137  153  151  4
155 158 167
```

4.3.7. Joc „Ghicirea numărului ascuns”

Deseori o persoană cere alteia să ghicească un număr. După prima încercare se poate ca cel care a ascuns numărul să zică „este prea mare”, sau „este prea mic”, sau chiar „felicitări – ai ghicit”.

Se cere să se scrie un program care simulează acest joc, şi anume o bibliotecă externă generează numărul ascuns şi programul îl ajută pe cel care încearcă să ghicească numărul. La sfârşit să se afişeze numărul de paşi în care s-a găsit numărul ascuns.

Descrierea bibliotecii

Pentru a utiliza biblioteca externă, în program va trebui să includeți instrucțiunea **uses** `nr;` (pentru limbajul Pascal), respectiv `#include "nr.h"`.

Biblioteca vă pune la dispoziție rutine pentru:

- inițializare
- ghicirea numărului

Antetele funcțiilor şi procedurilor:

```
procedure Init;
void Init(void);
```

Acesta este folosit pentru inițializarea bibliotecii.

```
function Incercare(nr:Integer):Integer;
int Incercare(int nr);
```

Aceasta se foloseşte pentru a testa `nr`. Dacă s-a returnat 1, înseamnă că `nr` este prea mare, 2 înseamnă că `nr` este prea mic, 0 înseamnă că s-a ghicit numărul ascuns.

4.3.8. Patinoarul

Un patinoar de formă dreptunghiulară prezintă defecte ale gheții datorită temperaturilor ridicate, dar şi a antrenamentelor numeroase. Proprietarii doresc să evite accidentele prin delimitarea unei zone fără probleme pentru patinatori, astfel încât să acopere o suprafață cât mai mare din cea inițială. Laturile zonei îngrădite pot fi realizate doar prin garduri paralele cu marginile patinoarului.

Se cunosc coordonatele în plan ale patinoarului, precum şi ale zonelor care pot provoca accidente, considerate punctiforme. Sunteți rugați să ajutați proprietarii să determine coordonatele unei zone dreptunghiulare de arie maximă care nu prezintă nici un risc, precum şi aria acestei zone.

Date de intrare
Prima linie a fişierului de intrare **PATINA.IN** va conține coordonatele patinoarului (ordonata şi abscisa colțului stânga-sus, respectiv dreapta-jos), separate prin câte un spațiu. Pe următoarea linie se află numărul *n* a punctelor cu probleme de pe suprafața patinoarului. Pe următoarele *n* linii se află câte două numere întregi, separate de un spațiu, reprezentând coordonatele punctelor nesigure din zonă.

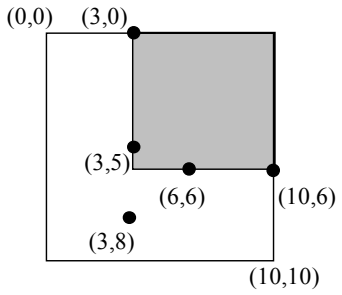
Date de ieşire
Fişierul de ieşire **PATINA.OUT** va conține o linie pe care se vor afla cinci numere, reprezentând aria şi coordonatele (ordonata şi abscisa) colțului stânga-sus, respectiv colțului dreapta-jos a zonei sigure. Aceste numere vor fi separate printr-un spațiu.

Restricții şi precizări
- Coordonatele sunt numere naturale;
- $0 \le n \le 200$;
- Dacă există mai multe soluții, în fişier se va scrie una singură;
- Zonele cu probleme se consideră punctiforme şi pot fi pe marginea împrejmuirii.

Exemplu

PATINA.IN
```
0 0 10 10
3
3 5
3 8
6 6
```

PATINA.OUT
```
42 3 0 10 6
```



4.3.9. Ora de sport

Profesorul de sport doreşte să aleagă un elev care să conducă un grup dintr-o clasă de *n* elevi la încălzirea de la ora de sport. Deoarece el ar vrea ca la fiecare oră, în principiu, să fie alt elev care are acest rol, la începutul orei îi aranjează pe elevi la întâmplare şi le împarte tricouri numerotate de la 1 la *n*. Apoi dă comenzi una după alta, rostind cuvintele „stânga" sau „dreapta", urmând ca după fiecare comandă jumătatea „opusă" comenzii date din şirul de elevi să plece spre zona de încălzire, iar dintre cei rămaşi să aleagă mai departe. Dacă numărul grupului de elevi împărțit la un moment dat este impar, elevul din mijloc pleacă şi el la încălzire. Alegerea se repetă până când rămâne un singur elev pe care profesorul îl numeşte *conducător* pentru ziua respectivă.

Într-o zi profesorul se întreabă oare câți elevi au şansa să devină conducători şi care ar fi aceştia? Dar el ar vrea să mai ştie şi dacă un elev purtând un tricou cu un anumit

număr poate sau nu deveni conducător şi, în caz afirmativ, ce comenzi trebuie să spună pentru ca acesta să ajungă conducător?

Scrieți un program care determină elevii care pot fi conducători, cunoscând numărul total al elevilor din clasă. Apoi, pe baza numărului de pe tricoul unui elev dat, stabiliți dacă acesta poate fi în ziua respectivă conducător sau nu. În caz afirmativ, determinați o succesiune de comenzi *stânga-dreapta* în urma cărora el devine conducător.

Date de intrare
Fişierul de intrare **SPORT.IN** conține două numere naturale *n* şi *t*, separate printr-un spațiu, reprezentând numărul de elevi din clasă, respectiv numărul de ordine de pe tricoul unui elev.

Date de ieşire
Fişierul de ieşire **SPORT.OUT** va conține pe prima linie numerele de pe tricourile elevilor care pot deveni conducători, separate prin câte un spațiu. Pe a doua linie se va afla cuvântul `'DA'` sau cuvântul `'NU'`, după cum elevul cu numărul *t* pe tricou poate să fie conducător sau nu. În caz afirmativ, pe a treia linie veți scrie un şir de caractere format din literele mici `'s'` şi `'d'` care reprezintă succesiunea de comenzi necesară. Dacă pe al doilea rând ați scris `'NU'`, în fişier nu se mai scrie nimic.

Restricții şi precizări
- $0 < n \leq 10000$:
- $0 < t \leq n$.

Exemple

SPORT.IN
```
7 3
```

SPORT.OUT
```
1 3 5 7
DA
sd
```

SPORT.IN
```
7 4
```

SPORT.OUT
```
1 3 5 7
NU
```

4.4. Soluțiile problemelor propuse

4.4.1. Minim

Din enunțul problemei se deduce clar care sunt cerințele şi cum trebuie determinată soluția. Dacă nu s-ar fi specificat metoda de rezolvare, probabil că o parcurgere a şirului de la primul element la ultimul, cu determinarea minimului local ar fi fost o soluție care nu ridica nici un fel de probleme. Dar metoda este impusă, şi anume *Divide et Impera*.

Metoda presupune împărțirea problemei în subprobleme care se rezolvă, iar apoi rezultatele subproblemelor se combină şi se obține soluția problemei date.

În cazul determinării minimului dintr-un şir de numere, descompunerea problemei revine la a împărți şirul în două subşiruri. De exemplu, dacă avem şirul $(x_1, x_2, \ldots, x_n)$ el poate fi împărțit în subşirul $(x_1, \ldots, x_m)$ şi în $(x_{m+1}, \ldots, x_n)$, unde n este numărul de elemente din şir, iar $m = [(1 + n) / 2]$ este indicele elementului din mijloc. După determinarea minimului pentru fiecare dintre cele două subşiruri (min_1 respectiv min_2), combinarea rezultatelor se realizează prin compararea valorilor celor două minimuri ($min_1 < min_2$). În funcție de rezultat se obține rezultatul final al valorii minime din şir, şi anume dacă min_1 este mai mic decât min_2, vom avea $min = min_1$, respectiv $min = min_2$, dacă min_1 este mai mare sau egal cu min_2.

Rezultatul obținut este corect, deoarece, dacă min_1 este valoarea minimă dintre elementele şirului $(x_1, \ldots, x_m)$, înseamnă că orice altă valoare din acest şir este mai mare sau egală cu min_1. Analog se poate raționa pentru min_2 şi şirul $(x_{m+1}, \ldots, x_n)$. Atunci, dacă $min_1 < min_2$, acest lucru implică $min_1 < x_j$, pentru $j \in \{m+1, \ldots, n\}$, în plus prin rezolvarea subproblemelor, min_1 este determinat ca fiind mai mic sau egal decât elementele x_k, unde $k \in \{1, \ldots, m\}$. Deci min_1 conține valoarea minimă din întreg şirul. Analog, dacă rezultatul comparației $min_1 < min_2$ este fals, se obține că min_2 conține valoarea minimă din întreg şirul.

Dar determinarea lui min_1 pentru primul subşir, respectiv a lui min_2 pentru al doilea subşir este o problemă de rezolvat la fel ca problema inițială pentru întreg şirul. Se va împărți subşirul $\{x_1, \ldots, x_m\}$ în alte două subşiruri şi, la rândul lor, cele două subşiruri în altele, până când subşirul care se obține este de dimensiune 1, adică are un singur element, care este valoarea minimă din acest subşir de lungime 1.

```
Subalgoritm Minim(st,dr):
  dacă st < dr atunci
    mij ← (st+dr) div 2        { se determină indicele elementului din mijloc }
    min1 ← Minim(st,mij)
    min2 ← Minim(mij+1,dr)
    dacă min1 < min2 atunci
      minim ← min1
    altfel
      Minim ← min2
    sfârşit dacă
  altfel
    Minim ← x[st]   { dacă st=dr atunci minimul este singurul element din şir }
  sfârşit dacă
sfârşit subalgoritm
```

4.4.2. Produsul numerelor prin metoda Divide et Impera

Din enunțul problemei se deduce clar care sunt cerințele şi cum trebuie determinată soluția. Dacă nu s-ar fi specificat metoda de rezolvare, o parcurgere a şirului de la primul element la ultimul ar fi fost o soluție care nu ridică nici un fel de probleme, dar trebuie să lucrăm cu metoda *Divide et Impera*.

Metoda presupune împărțirea problemei în subprobleme care se rezolvă, apoi rezultatele subproblemelor se combină şi se obține soluția.

Pentru a calcula produsul a n numere, se calculează produsul numerelor din prima jumătate a şirului, produsul numerelor din a doua jumătate şi apoi rezultatul final se obține prin înmulțirea celor două produse calculate. De exemplu, şirul $(x_1, x_2, \ldots, x_n)$ poate fi împărțit în subşirurile $(x_1, \ldots, x_m)$ şi $(x_{m+1}, \ldots, x_n)$, unde n este numărul de elemente din şir, iar $m = [(1 + n) / 2]$.

Calculul produsului numerelor din prima jumătate se poate realiza de asemenea folosind metoda *Divide et Impera* şi astfel primul subşir se împarte în două subşiruri, şi aşa mai departe până când, prin împărțirea în subşiruri, se obțin şiruri de lungime 1. Aceste subşiruri conțin un singur element şi în acest caz „produsul numerelor" este egal cu numărul respectiv.

```
Subalgoritm Produs(st,dr):
  dacă st=dr atunci
    Produs ← x[st]
  altfel
    m ← (st+dr) div 2          { se determină indicele elementului din mijloc }
    p1 ← Produs(st,mij)
    p2 ← Produs(mij+1,dr)
    Produs ← p1 * p2
  sfârşit dacă
sfârşit subalgoritm
```

4.4.3. Turnurile din Hanoi

Se observă că există reguli stricte privind mutarea unui disc.

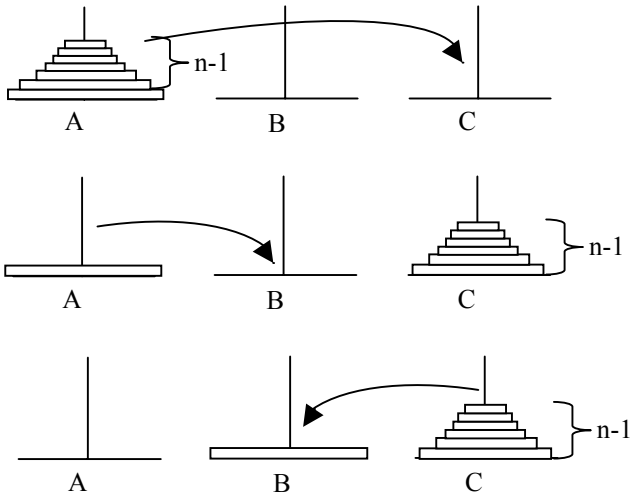
De asemenea, se observă că dacă pe tija A ar exista un singur disc, unica mutare necesară ar fi $A \rightarrow B$.

Pentru două discuri existente pe tija A, mutările ar fi: $A \rightarrow C$, $A \rightarrow B$ şi $C \rightarrow B$. Pentru un număr mai mare de discuri, evident va fi un număr mult mai mare de mutări.

Împărțim problema în subprobleme astfel:

- se mută primele $n - 1$ discuri de pe tija A pe tija C,
- se mută ultimul disc (cel care are diametrul cel mai mare) de pe tija A pe tija B,
- se mută cele $n - 1$ discuri de pe tija C pe tija B.

Acum au apărut două probleme mai „simple” şi anume mutarea a $n-1$ discuri de pe tija A pe tija C, respectiv cele $n-1$ discuri de pe tija C pe tija B. La rândul lor, aceste operaţii se pot împărţi în subprobleme până când rămâne de mutat un singur disc. Astfel, problema se rezolvă folosind metoda *Divide et Impera*.



La un transfer al unui disc acesta este în vârful tijei şi este aşezat tot în vârful unei tije, astfel că o mutare este definită prin numele tijelor de pe care se ia un disc, respectiv cea pe care se aşează. Pentru a asigura precizarea lor pentru fiecare subproblemă care trebuie rezolvată, vom folosi trei parametri (pentru tija sursă, tija destinaţie şi tija auxiliară) care vor primi valori în momentul autoapelării subalgoritmului.

```
Subalgoritm Hanoi(n,A,B,C):
  dacă n=1 Mută un disc de pe tija A pe tija B
    Hanoi(n-1,A,C,B)
    Mută un disc de pe tija A pe tija B
    Hanoi(n-1,C,B,A)
  sfârşit dacă
sfârşit subalgoritm
```

Combinarea soluţiilor subproblemelor nu este necesară în cazul rezolvării acestei probleme.

4.4.4. Sortare prin interclasare (*merge-sort*)

Un şir de valori poate fi ordonat cu diverse metode (în celebra carte a lui *D. Knuth*, *Arta programării calculatoarelor* sunt prezentate peste 30 de metode de ordonare).

De data aceasta se cere explicit să se rezolve această problemă cu metoda de sortare prin interclasare.

Metoda presupune împărțirea şirului în *subşiruri ordonate* şi apoi interclasarea lor, obținându-se astfel un şir ordonat.

Prima etapă, şi anume împărțirea şirului în şubşiruri ordonate, începe prin împărțirea şirului inițial în două subşiruri (cu atenție, astfel încât elementul din mijloc să nu apară în ambele). Este foarte posibil ca cele două subşiruri obținute să nu fie ordonate, dar la rândul lor şi acestea vor fi împărțite în două subşiruri şi aşa mai departe. Subşirurile se vor împărți până când subşirul obținut are un singur element, şi acesta este sigur ordonat. Urmează etapa a doua în care interclasăm subşirurile.

O astfel de rezolvare se încadrează în metoda *Divide et Impera*. Împărțirea problemei în subprobleme este realizată prin divizarea unui şir în subşiruri, iar combinarea rezultatelor parțiale se realizează prin interclasarea celor două subşiruri ordonate.

Astfel, la un pas al prelucrării vom avea subşirul (x_{st}, …, x_{dr}) care se va împărți în două subşiruri (x_{st}, …, x_m) şi (x_{m+1}, …, x_{dr}), unde $m = [(st + dr) / 2]$. Aceste subşiruri se ordonează, apoi urmează interclasarea lor pentru a obține subşirul inițial cu elementele în ordinea cerută.

```
Subalgoritm Ordonare(st,dr):
  dacă st = dr atunci
    ieşire forțată din subalgoritm          { subşirul conține un singur element }
  altfel
    m ← (st+dr) div 2
    Apel prelucrare pentru şirul de indici st, m
    Apel prelucrare pentru şirul de indici m+1, dr
    Interclasare(st,m,dr)    { interclasarea şirurilor (xst, …, xm) şi (xm+1, …, xdr) }
  sfârşit dacă
sfârşit subalgoritm
```

Interclasarea a două şiruri ordonate înseamnă crearea unui nou şir care conține elementele primelor două şi, în plus, acest nou şir este de asemenea ordonat după acelaşi criteriu ca şirurile inițiale. Trebuie menționat că cele două şiruri inițiale trebuie să fie ordonate după acelaşi criteriu, de exemplu ambele să fie ordonate crescător.

Exemplu

Fie două şiruri ordonate: $a = (1, 3, 5, 17, 21)$ şi $b = (3, 3, 8, 11, 23, 27)$. Prin interclasarea lor se obține şirul ordonat $c = (1, 3, 3, 3, 5, 11, 17, 21, 23, 27)$.

Pentru cele două subşiruri (x_{st}, …, x_m) şi (x_{m+1}, …, x_{dr}) care sunt ordonate crescător, interclasarea presupune parcurgerea lor relativ simultan cu ajutorul a doi indici şi trecerea în al treilea şir a elementului minim dintre cele două elemente comparate la un moment dat.

Reamintim subalgoritmul de interclasare, sub forma clasică, actualizată pentru ordonare prin interclasare prezintă câteva modificări. Acum toate elementele fac parte din *acelaşi şir x*. În cadrul algoritmului clasic se „consuma" pe rând câte un element din cele două şiruri şi se trecea în al treilea şir – cel care conţinea şirurile interclasate. Acum al treilea şir trebuie să fie unul auxiliar, altfel s-ar suprascrie valori din şirul iniţial. La sfârşitul subalgoritmului vom copia în *x* (de fapt în subşirul (x_{st}, …, x_{dr}) valorile din şirul auxiliar.

```
Subalgoritm Interclasare(st,m,dr):
  i ← st
  j ← m + 1
  k ← 0
  cât timp (i ≤ m) şi (j ≤ dr) execută:
    dacă xi < xj atunci          { dacă xi (i≤ m) este mai mic decât xj (j ≤ dr) }
      k ← k + 1                { îl copiem în şirul auxiliar a pe poziţia curentă k }
      ak ← xi
      i ← i + 1                                  { avansăm în subşirul din stânga }
    altfel                    { dacă xj (j ≤ dr) este mai mic sau egal cu xi (i ≤ m) }
      k ← k + 1                { îl copiem în şirul auxiliar a pe poziţia curentă k }
      ak ← xj
      j ← j + 1                                 { avansăm în subşirul din dreapta }
    sfârşit dacă
  sfârşit cât timp
  cât timp i ≤ m execută:  { dacă în subşirul din stânga mai există elemente }
    k ← k + 1                                                  { le copiem în a }
    ak ← xi
    i ← i + 1
  sfârşit cât timp
  cât timp j ≤ dr execută: { dacă în subşirul din dreapta mai există elemente }
      k ← k + 1                                                { le copiem în a }
      ak ← xj
      j ← j + 1
  sfârşit cât timp
  pentru i=1,k execută:
        { în şirul a avem k = dr – st + 1 elemente care trebuie copiate în xst, ..., xdr }
      xst+i-1 ← ai
sfârşit subalgoritm
```

4.4.5. Sortare rapidă (*quicksort*)

Problema ordonării unui şir poate fi soluţionată prin diverse metode de ordonare. Acum se cere explicit rezolvarea cu ajutorul metodei de sortare rapidă.

Exemplu

Fie următorul şir de numere: 8, 7, 6, 10, 4, 11, 2, 5, 9. Ne propunem să-l ordonăm crescător.

Primul element se compară cu toate celelalte din şir. Ordonarea porneşte prin compararea primului element cu ultimul din şir.

8, 7, 6, 10, 4, 11, 2, 5, **9**
↑ ↑

Aceste două elemente sunt în ordinea cerută. În această situaţie se trece la compararea primului număr cu penultimul.

8, 7, 6, 10, 4, 11, 2, **5**, 9
↑ ↑

Cele două elemente nu sunt în ordinea cerută, deci vor fi interschimbate şi se obţine şirul:

5, 7, 6, 10, 4, 11, 2, **8**, 9

Deoarece a avut loc o interschimbare, se va continua compararea lui **8** (aflat acum pe penultima poziţie) cu numărul **7** (al doilea, situat după elementul care la pasul anterior a participat la interschimbare), şi având în vedere că **7** este pe o poziţie anterioară lui **8** în şir şi este adevărată relaţia **7 < 8** cele două numere rămân neschimbate.

5, **7**, 6, 10, 4, 11, 2, **8**, 9
↑ ↑

Din aceleaşi considerente şi numerele **6** şi **8** rămân în locaţiile lor după comparare.

5, 7, **6**, 10, 4, 11, 2, **8**, 9
↑ ↑

Urmează să se compare **10** cu **8** şi, datorită faptului că nu sunt în relaţia de ordine cerută, (adică în ordine crescătoare) cele două valori se interschimbă.

5, 7, 6, **10**, 4, 11, 2, **8**, 9 devine: 5, 7, 6, **8**, 4, 11, 2, **10**, 9
↑ ↑

Urmează să se compare tot numărul **8,** dar cu **2**. Datorită faptului că **8 > 2**, cele două numere trebuie schimbate între ele.

5, 7, 6, **8,** 4, 11, **2**, 10, 9 devine: 5, 7, 6, **2**, 4, 11, **8**, 10, 9
↑ ↑

Următoarea comparaţie se va face între **8** şi **4**. Cele două numere rămân la locurile lor.

5, 7, 6, 2, **4**, 11, **8**, 10, 9
↑ ↑

O ultimă comparaţie se realizează între **8** şi **11**. Cele două numere trebuie interschimbate şi se obţine:

5, 7, 6, 2, 4, **8**, 11, 10, 9

După această parcurgere, numărul **8**, cel care a fost în şirul iniţial pe prima poziţie, se află acum pe poziţia sa finală în şir. Toate elementele din subşirul din stânga lui **8** sunt mai mici decât acesta şi toate cele din subşirul din dreapta lui **8** sunt mai mari de-

cât el. Primul număr din şir, în cazul de faţă **8**, a fost comparat cu toate elementele din şir şi prin interschimbări cu cele care sunt mai mici decât el au ajuns în stânga sa, iar cele mai mari au ajuns în subşirul din partea dreaptă.

Se obţin astfel două subşiruri (care trebuie ordonate) şi un element care se află pe poziţia sa finală în şir.
(5, 7, 6, 2, 4), **8**, (11, 10, 9)

Cele două subşiruri obţinute se împart la rândul lor după cum urmează:
(**5**, 7, 6, 2, **4**) → (4, **7**, 6, 2, **5**) → (4, **5**, 6, 2, **7**) → (4, **5**, 6, **2**, 7) → (4, 2, 6, **5**, 7) → (4, 2, **6**, **5**, 7) → (4, 2, **5**, 6, 7) → (4, 2), **5**, (6, 7)
(**11**, 10, **9**) → (9, 10, **11**) → (9, **10, 11**) → (9, 10), **11**

Metoda presupune împărţirea şirului în două subşiruri, astfel încât toate elementele din primul subşir să fie mai mici decât toate elementele din al doilea subşir.

Împărţirea şirului în subşiruri se realizează până când se obţin subşiruri de lungime 1, caz în care se consideră că acel subşir este ordonat.

Metoda sortării rapide lucrează cu metoda *Divide et Impera*, deoarece problema iniţială se împarte în subprobleme independente unele de altele, care se rezolvă mai uşor. Datorită faptului că subşirurile se ordonează direct, combinarea rezultatelor parţiale nu se mai realizează.

Astfel, la o împărţire a unui subşir [x_{st}, …, x_{dr}] se poate scrie algoritmul:

```
Subalgoritm Quick(st,dr):
  dacă st < dr atunci
    Împarte(st,dr,p) { se determină poziţia p care reprezintă locul unde se taie şirul în }
                     { două subşiruri după ce un element a ajuns pe poziţia sa finală p }
    Quick(st,p-1)        { se procedează similar pentru subşirul (x_st, ..., x_p-1) }
    Quick(p+1,dr)                          { şi pentru subşirul (x_p+1, ..., x_dr) }
  sfârşit dacă
sfârşit subalgoritm
```

Cea mai importantă parte a algoritmului o reprezintă cea în care se împarte şirul în subşiruri, procedură în care se realizează şi o ordonare a elementelor faţă de primul număr din şir.

Trebuie ca în fiecare moment să se păstreze cei doi indici ai elementelor care se compară. La fiecare pas se modifică unul dintre cei doi indici, fie cel din dreapta, fie cel din stânga.

Se vor folosi două variabile auxiliare *ii* şi *jj* (deplasamente), cu ajutorul cărora se va şti în fiecare moment care este indicele care creşte (respectiv scade). Pentru început primul număr se compară cu elemente luate de la sfârşitul şirului avansând spre început. Deci primul indice (*i*) rămâne constant şi al doilea (*j*) scade. Astfel $ii \leftarrow 0$, iar $jj \leftarrow -1$. După prima interschimbare de valori, primul indice (*i*) creşte şi al doilea (*j*)

rămâne constant, adică *ii* ← 1 şi *jj* ← 0. Operaţiile de comparare şi trecerea la pasul următor se realizează până când *i* devine egal cu *j*. În situaţia *i* = *j*, elementul care a fost iniţial pe prima poziţie din şir este acum pe poziţia *i*, care este poziţia sa finală în şir, şi s-au obţinut cele două subşiruri.

```
Subalgoritm Împarte(st,dr,p):
  i ← st
  j ← dr
  ii ← 0
  jj ← -1
  cât timp i < j execută:
    dacă x_i > x_j atunci          { dacă elementele nu sunt în ordinea dorită }
      aux ← x_i                                  { se interschimbă elementele }
      x_i ← x_j
      x_j ← aux
      aux ← ii                  { totodată se schimbă şi sensul de comparare }
      ii ← -jj
      jj ← -aux
    sfârşit dacă
    i ← i + ii
    j ← j + jj
  sfârşit cât timp
sfârşit subalgoritm
```

Comparativ cu alte metode de ordonare, în cazul sortării rapide, un singur element este comparat cu toate celelalte, după care se obţin două subşiruri mai scurte ca lungime şi acest fapt conduce la un timp mai mic de execuţie a algoritmului.

4.4.6. Serbare

Dacă şirul nu ar fi ordonat, profesorul ar fi nevoit să încerce să găsească elevul potrivit luând elev după elev şi verificând înălţimea sa. În momentul în care a găsit elevul căruia i se potriveşte costumul, căutarea profesorului ar lua sfârşit. Copilul căutat poate fi primul, dar tot aşa de bine poate fi şi ultimul.

Având în vedere faptul că înălţimea elevilor este dată în ordine crescătoare, o soluţie prin care se găseşte elevul potrivit ar trebui să ţină cont de acest fapt.

O strategie posibilă este ca profesorul să încerce să găsească elevul potrivit studiind înălţimea elevului aflat pe poziţia din mijloc al şirului. Dacă elevul căutat nu este cel din mijloc, profesorul îşi poate da seama dacă elevul este în prima sau în cea de-a doua jumătate a şirului. Şirul este ordonat crescător, iar dacă elevul din mijloc este mai mic decât cel căutat, este sigur că elevul căutat nu se găseşte în prima jumătate a şiru-

lui. Dacă elevul din mijloc este mai mare decât cel căutat, este de asemenea sigur că elevul căutat nu se găseşte în cea de-a doua jumătate. În cele ce urmează, profesorul trebuie să continue căutarea doar în grupul format din jumătate dintre elevi, grup care la rândul lui este un şir ordonat.

Acest subşir va fi împărţit în două (la fel ca şirul iniţial) şi eventual (dacă elevul căutat nu este cel din mijlocul subşirului) se continuă căutarea în jumătatea corespunzătoare a subşirului. Această metodă se repetă până când este găsit elevul cu înălţimea potrivită sau în cazul în care nu există un astfel de elev până când lungimea subşirului este egală cu 1. În acest caz răspunsul la problema profesorului este: „Nu se află în şir nici un elev cu înălţimea potrivită".

Această strategie aplică algoritmul de *căutare binară*. Algoritmul constă în căutarea unei valori *val* într-un şir ordonat de elemente. Astfel, dacă trebuie găsită valoarea *val* în şirul $x_1, x_2, \ldots, x_n$, se testează dacă *val* este egal cu x_m, unde $m = [(n + 1)/2]$.

Dacă cele două valori sunt egale, căutarea se opreşte cu succes.

Dacă cele două valori nu sunt egale, nu înseamnă că nu vom găsi în viitor valoarea *val*. S-ar putea ca valoarea căutată să se găsească într-unul dintre cele două subşiruri obţinute: în subşirul $x_1, \ldots, x_{m-1}$ în situaţia în care $x_m > val$ sau în subşirul $x_{m+1}, \ldots, x_n$, în situaţia în care $x_m < val$.

Împărţirea şirului în două subşiruri arată că problema face parte din categoria celor rezolvate cu ajutorul metodei *Divide et Impera*, deoarece împărţirea unui astfel de subşir se poate realiza *independent* de împărţirea celorlalte subşiruri. În concluzie, problema se împarte în subprobleme ***independente***. Metoda de rezolvare *Divide et Impera* în majoritatea cazurilor conduce la algoritmi având complexitate logaritmică.

Problemele care se rezolvă folosind această metodă presupun posibilitatea de a le împărţi în subprobleme distincte care se pot rezolva mai uşor şi soluţiile acestor subprobleme se combină pentru a obţine rezultatul final al problemei de rezolvat.

În cazul căutării binare împărţirea problemei în subprobleme se realizează prin determinarea celor două subşiruri. Astfel, dacă se caută o valoare în şirul $(x_{st}, \ldots, x_{dr})$, împărţirea problemei constă în stabilirea subşirurilor $(x_{st}, \ldots, x_{m-1})$ şi $(x_{m+1}, \ldots, x_{dr})$, unde $m = [(st + dr) / 2]$.

```
Subalgoritm CăutareBinară(st,dr,h):
  mij ← (st + dr) div 2          { stabilirea mijlocului subşirului curent }
  dacă h = x[mij] atunci
    scrie DA, mij
  altfel
    dacă h < x[mij] atunci
      CăutareBinară(st,mij-1,h)
    altfel
      CăutareBinară(mij+1,dr,h)
sfârşit subalgoritm
```

În rezolvarea acestei probleme nu există compunerea soluţiilor subproblemelor pentru a obţine soluţia finală. Practic găsim valoarea căutată în „ultima" clipă sau aflăm că nu există soluţie.

4.4.7. Joc „Ghicirea numărului ascuns"

Pentru a ghici un număr dintr-un anume interval, prima tendinţă este de a încerca valoarea din mijloc. Dacă este prea mare se va reduce intervalul în care se caută la prima jumătate. De exemplu, dacă intervalul iniţial este [0, 200] se va încerca cu valoarea 100. Apoi se încearcă cu 50 şi la fel se obţine jumătate din ultimul interval considerat, dacă 50 nu a fost cumva numărul ascuns.

După câte se observă strategia de găsire a numărului ascuns se bazează pe metoda Divide et Impera. În cazul de faţă metoda este folosită de persoana care utilizează programul (încearcă să ghicească numărul).

Algoritmul de găsire a numărului ascuns se numeşte căutare binară şi este cel mai rapid algoritm de căutare a unei valori într-un şir ordonat de numere. Numărul maxim de căutări va fi de ordinul $\log_2 n$, unde n este numărul valorilor din şir.

4.4.8. Patinoarul

Dacă patinoarul nu are zone punctiforme cu probleme în interiorul său, atunci zona maximă coincide cu patinoarul iniţial.

Dacă ar avea un singur punct problemă de coordonate (x, y), problema s-ar reduce la a împărţi patinoarul în zone mai mici, delimitate de punctul respectiv ca în figura 2, respectiv ca în figura 3, dreptunghiuri care nu conţin acest punct, el fiind pe marginea zonei. Se obţin astfel 4 zone dreptunghiulare. Zona căutată este una dintre cele patru zone, cea de arie maximă.

În situaţia în care patinoarul conţine mai multe zone nesigure este foarte probabil ca între cele patru dreptunghiuri obţinute prin împărţire, să existe unele care au la rândul lor puncte cu probleme. Aceste noi dreptunghiuri, care au puncte cu probleme se vor trata la fel ca dreptunghiul iniţial şi vor fi împărţite în alte zone mai mici. Împărţirea se repetă până când dreptunghiurile obţinute nu conţin nici un punct interior şi dintre acestea soluţia este dată de dreptunghiul de arie maximă.

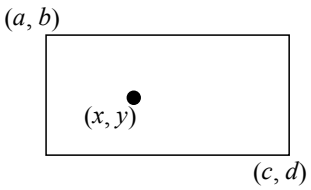


Fig. 1

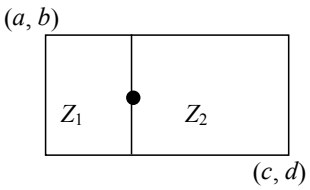


Fig. 2

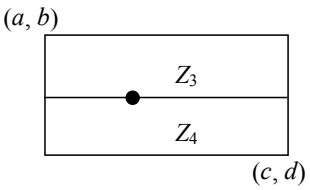


Fig. 3

Dacă se cunosc coordonatele dreptunghiului de împărțit ((*a*, *b*), (*c*, *d*)) şi coordonatele punctului curent (*x*, *y*), se pot calcula foarte uşor coordonatele noilor dreptunghiuri care se formează:

Z_1: colțul stânga-sus rămâne (*a*, *b*), colțul dreapta-jos devine (*c*, *y*)

Z_2: colțul stânga-sus devine (*a*, *y*), colțul dreapta-jos rămâne (*c*, *d*)

Z_3: colțul stânga-sus rămâne (*a*, *b*), colțul dreapta-jos devine (*x*, *d*)

Z_4: colțul stânga-sus devine (*x*, *b*), colțul dreapta-jos rămâne (*c*, *d*)

Aria unei zone de coordonate (*a*, *b*), (*c*, *d*) se calculează cu formula:

$A = (c - a) \cdot (d - b)$.

Pentru a determina dacă un dreptunghi conține puncte cu probleme se poate folosi o funcție care returnează indicele unui punct din tabloul în care sunt memorate, punct cu probleme din interiorul dreptunghiului ales la un moment dat. Dacă acesta nu există returnează 0. Condiția ca un punct de coordonate (*x*, *y*) să se afle în interiorul unui dreptunghi având coordonatele colțurilor în (*a*, *b*), respectiv (*c*, *d*) este ca ($a < x$, $x < c$, $b < y$ şi $y < d$).

```
Subalgoritm Există(a1,b1,c1,d1,i):
                    { caută un punct interior dreptunghiului de coordonate (a1, b1) }
  i ← 1                                        { şi (c1, d1) (nu şi pe margini) }
  găsit ← fals
  cât timp (i ≤ n) şi nu găsit execută:
    dacă (a1 < x[i]) şi (x[i] < c1) şi
         (b1 < y[i]) şi (y[i] < d1) atunci
      găsit ← adevărat              { se va returna indicele i a punctului găsit }
    altfel
      i ← i + 1
    sfârşit dacă
  sfîrşit cât timp
  dacă nu găsit atunci               { dacă nici un punct nu se află în interior }
    i ← 0
  sfârşit dacă
sfârşit subalgoritm
```

Rezolvarea este realizată astfel prin metoda *Divide et Impera*. Împărțirea problemei în subprobleme se realizează prin determinarea a patru dreptunghiuri. Rezolvarea subproblemelor revine la a căuta dacă există puncte nesigure în fiecare dintre dreptunghiurile nou obținute. Dacă există astfel de puncte se trece la o nouă împărțire.

Pentru un dreptunghi fără puncte cu probleme se calculează aria sa şi se compară cu aria maximă obținută deja.

```
Subalgoritm Împart(a1,b1,c1,d1):
  Există(a1,b1,c1,d1,i)       { se caută un punct în interiorul dreptunghiului }
```

```
  dacă i = 0 atunci              { dacă în zonă nu există nici un punct nesigur }
    aria ← (c1-a1)*(d1-b1)                              { se calculează aria }
                          { dacă aria este mai mare decât maximul găsit până acum }
    dacă aria > max atunci
      max ← aria   { se păstrează aria şi coordonatele dreptunghiului respectiv }
      am ← a1
      bm ← b1
      cm ← c1
      dm ← d1
    altfel                  { se împarte terenul în cele patru dreptunghiuri noi }
      Împart(a1,b1,x[t],d1)
      Împart(x[t],b1,c1,d1)
      Împart(a1,b1,c1,y[t])
      Împart(a1,y[t],c1,d1)
    sfârşit dacă
  sfârşit dacă
sfârşit subalgoritm
```

4.4.9. Ora de sport

Să observăm că enunțul precizează explicit modelul după care se desfăşoară activitatea profesorului care urmăreşte descoperirea/stabilirea conducătorului grupului de elevi.

– Nu există criterii de inteligență (profesorul ar putea fi subiectiv...);

– Nu există considerente privind înălțimea, greutatea sau culoarea ochilor... (şirul nu este ordonat după nici un criteriu), ci doar după numărul de pe tricouri. Rezultă că aceste numere vor fi considerate numere de ordine (indici) în şir.

Ceea ce „alege" profesorul este jumătatea care ***rămâne*** (de aici rezultând şi jumătatea care pleacă)! Rezultă că rezolvarea, adică stabilirea răspunsului la cerința problemei este dirijată de comenzile profesorului. Se observă că profesorul dă comenzi până când în şirul (grupul) de elevi rămâne unul singur! În concluzie, în urma comenzilor profesorului, grupul se înjumătățeşte, rămânând în continuare de împărțit jumătatea „preferată", adică cea care nu a plecat.

Se observă că faptul că dacă la un moment dat pleacă, de exemplu, jumătatea din stânga din şir, nu are nici o legătură cu o comandă viitoare imediată sau ulterioară a profesorului; el alege să plece o parte sau alta din grupul existent la momentul respectiv. Altfel spus, faptul că dintre copiii c_1, c_2, ..., c_n la un moment dat vor rămâne copiii c_1, c_2, ..., c_q, unde $q = [(1 + n)/2]$ sau, în funcție de paritatea lui n, $q = [(1 + n)/2] - 1$, nu are nici o legătură cu faptul că la un pas viitor se va alege jumătate dintr-un şir c_p, c_{p+1}, ..., c_r.

Rezultă că împărțirea unui astfel de subşir se poate face *independent* de împărțirea celorlalte subşiruri. În concluzie, problema se împarte în subprobleme ***independente***. În astfel de probleme metoda de rezolvare recomandată este *Divide et Impera*.

Stabilită fiind metoda, să definim o subproblemă: la un moment dat avem elevii cu numerele de ordine *st*, *st* + 1, ..., *dr* într-un subşir care urmează să fie împărţit. La comanda profesorului, acest şir se împarte în două subşiruri, în funcţie de paritatea lungimii şirului (*st* – *dr* + 1):

1. Dacă *st* – *dr* + 1 este număr par, rezultă subşirurile formate din copiii având numerele de ordine *st* + 1, *st* + 2, …, *mij* şi *mij* + 1, *mij* + 2, …, *dr*, unde *mij* este [(*st* + *dr*)/2]
2. Dacă *st* – *dr* + 1 este număr impar, vor rezulta subşirurile (*st* + 1, *st* + 2, …, *mij* – 1) şi (*mij* + 1, *mij* + 2,…, *dr*), urmând ca cel cu numărul de ordine *mij* să meargă direct la încălzire.

În funcţie de comandă rămâne în continuare în „prelucrare" fie subşirul din stânga, fie cel din dreapta.

Să observăm că dacă numărul *n* este putere a lui 2, vor putea deveni conducători toţi copiii. Adică, dacă $n = 2^k$, la prima împărţire obţinem două subşiruri, ambele de lungimi puteri ale lui 2, (care, evident, sunt numere pare) şi aşa mai departe, fiecare împărţire dă naştere la subşiruri având număr par de elevi. De exemplu, dacă *n* = 16, prima împărţire conduce la subşirurile formate din elementele având indice între 1..8 şi 9..16, Dacă s-a comandat *stânga*, se reţine primul subşir (1..4). Dacă urmează din nou o comandă *stânga*, rămân copiii cu numerele de ordine 1 şi 2. În funcţie de următoarea comandă conducător va fi fie 1 fie 2. Dar la fel s-ar împărţi şi celelalte subşiruri în funcţie de comenzile profesorului.

Să vedem ce se întâmplă dacă *n* nu este putere a lui 2, dar este număr par. Evident, împărţind un astfel de număr la 2, putem obţine, fie un număr par fie unul impar. De exemplu, dacă *n* = 10, şi prima comandă este *dreapta*, rămâne subşirul 6..10. O comandă nouă *dreapta* dă naştere subşirului 9..10, deoarece *mij* ar avea valoarea 8, dar copilul care are acest număr de ordine pleacă la încălzire deoarece în subşirul 6..10 au fost 5 (un număr impar) de elevi.

Rezultă că dacă avem subşirul *st*, *st* + 1, ..., *dr*, unde numărul de elevi (*dr* – *st* +1) este impar, elevul având numărul de ordine *mij* = [(*st* + *dr*)/2] nu poate să fie conducător. Aşa pare, că împărţind pur şi simplu numărul *n* în mod repetat la 2, am putea răspunde la prima întrebare (*oare câţi elevi au şansa să fie conducătorii şi care ar fi aceştia?*). Dar nu este aşa! Trebuie să rezolvăm aceeaşi subproblemă pentru subşiruri rezultate atât în partea stângă, cât şi în partea dreaptă!

Cea de-a doua întrebare (*un elev purtând un tricou cu un anumit număr poate sau nu deveni conducător?*) într-un fel este "inversa" primei întrebări. Adică, îndepărtând dintre cei *n* elevi pe cei care nu pot fi conducători, rămân cei care pot fi. Rezultă că vom păstra numerele de ordine care în urma divizării repetate rămân singure într-un subşir. Din cauză că şirul nu se împarte exact în două, ci în funcţie de paritatea elementelor, cel din mijloc se elimină din prelucrare, este nevoie de o variabilă suplimentară `mij1` în care păstrăm numărul de ordine al mijlocului şirului, urmând ca în funcţie de paritatea elementelor în subşirul actual să stabilim limitele subşirurilor rezultate.

```
Subalgoritm Com1(st,dr):
                    { afişează toate numerele elevilor care pot deveni conducători }
  dacă st < dr atunci
    mij ← (st+dr) div 2          { se determină numărul de ordine din mijloc }
    mij1 ← mij + 1
    dacă (dr-st+1) este număr impar atunci
      mij ← mij - 1                        { elevul din mijloc este eliminat }
    sfârşit dacă
    Com1(st,mij)                 { repetarea comenzii pentru prima jumătate }
    Com1(mij1,dr)                { repetarea comenzii pentru a doua jumătate }
  altfel
    scrie st     { st=dr, adică în şir a rămas un singur elev, acesta se afişează }
  sfârşit dacă
sfârşit subalgoritm
```

În a treia întrebare ni se cer *comenzile care trebuie spuse pentru ca un anumit elev să ajungă conducător*. Pentru aceasta vom determina mijlocul subşirului, apoi în funcție de valoare, vom elimina (depăşi) sau nu acest punct de mijloc. Dacă numărul de ordine este mai mic, îl vom căuta în continuare în stânga, ceea ce necesită comanda *stânga*, altfel în dreapta, corespunzător unei comenzi *dreapta*. Comenzile (caracterele corespunzătoare) le vom păstra concatenate într-o variabilă de tip **string**.

```
Subalgoritm Com2(st,dr):{ se determină dacă numărul căutat este conducător }
       { şi succesiunea de comenzi necesară pentru ca acesta să devină conducător }
  dacă st < dr atunci
    mij ← (st+dr) div            { se determină numărul de ordine din mijloc }
    t1 ← mij
    mij1 ← mij + 1
    dacă (dr-st+1) este număr impar atunci
      mij ← mij - 1                        { elevul din mijloc este eliminat }
    sfârşit dacă
    dacă t ≤ t1 atunci
      comanda ← comanda+'s'          { căutăm numărul de ordine în stânga }
      Com2(st,mij)
    altfel
      comanda ← comanda+'d'         { căutăm numărul de ordine în dreapta }
      Com2(mij1,dr)
    sfârşit dacă
  altfel
    dacă t = st atunci găsit ← adevărat     { t este un posibil conducător }
    sfârşit dacă
  sfârşit dacă
sfârşit subalgoritm
```