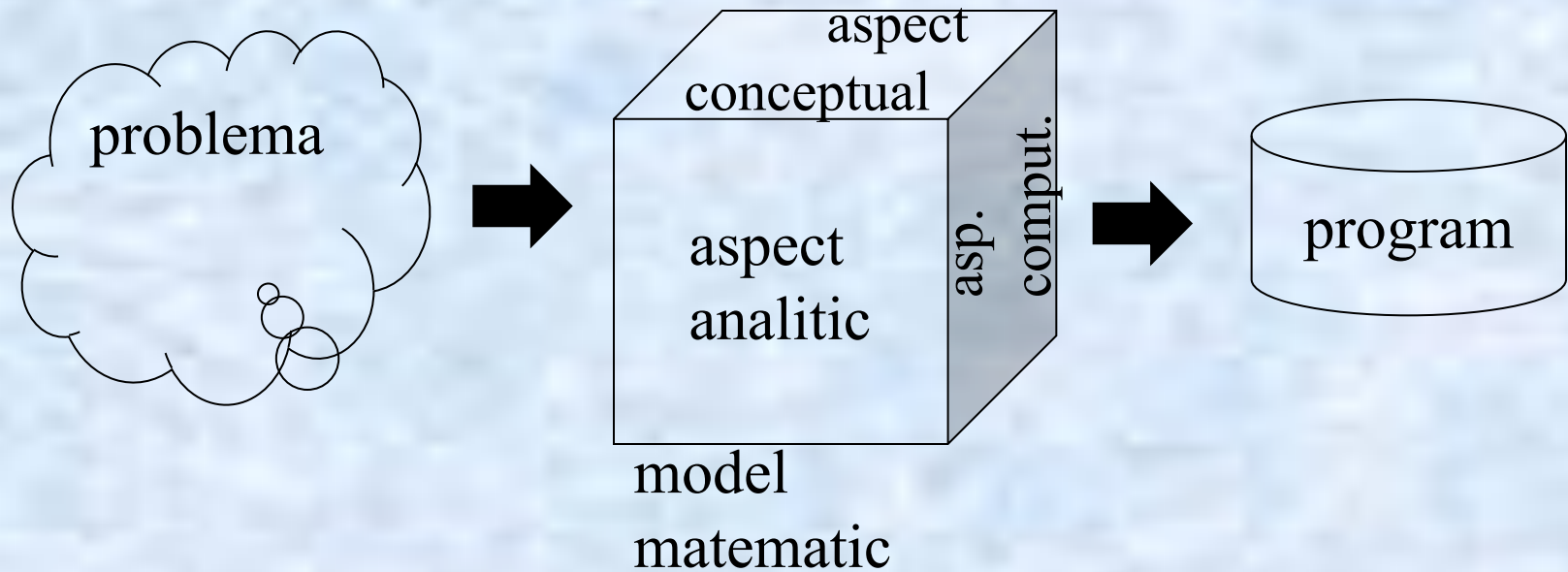


Paradigme de proiectare a algoritmilor

- Despre paradigme de proiectare a algoritmilor
- Paradigma divide-et-impera
 - ⇒ Prezentarea generala a paradigmei
 - ⇒ Studii de caz
 - cautare binara
 - constructia arborelui binar de cautare
 - sortare prin interclasare
 - sortare rapida (quick sort)
 - selectionare
 - transformarea Fourier rapida
 - “chess board cover”
 - linia orizontului

Despre paradigmele de proiectare a algoritmilor



- Avantajele aduse de constructia modelului matematic:
 - ⇒ eliminarea ambiguitatilor si inconsistentelor
 - ⇒ utilizarea intrumentelor matematice de investigare
 - ⇒ diminuarea efortului de scriere a programelor

Paradigma divide-et-impera

□ Modelul matematic

⇒ $P(n)$: problema de dimensiune n

⇒ baza

- daca $n \leq n_0$ atunci rezolva P prin metode elementare

⇒ divide-et-impera

- **divide** P in a probleme $P_1(n_1), \dots, P_a(n_a)$ cu $n_i \leq n/b$, $b > 1$
- **rezolva** $P_1(n_1), \dots, P_a(n_a)$ in aceeași maniera și obține soluțiile $S_1, \dots, S_a$
- **asambleaza** $S_1, \dots, S_a$ pentru a obține soluția S a problemei P

Paradigma divide-et-impera: algoritm

```
procedure DivideEtImpera(P, n, S)
begin
    if (n <= n0)
    then determina S prin metode elementare
    else imparte P in P1, ..., Pa
        DivideEtImpera(P1, n1, S1)
        ...
        DivideEtImpera(Pa, na, Sa)
    Asambleaza(S1, ..., Sa, S)
end
```

Paradigma divide-et-impera: complexitate

- presupunem ca divizarea + asamblarea necesita timpul $O(n^k)$

$$T(n) = \begin{cases} O(1) & \text{daca } n \leq n_0, \\ aT\left(\frac{n}{b}\right) + O(n^k) & \text{daca } n > n_0. \end{cases}$$

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{daca } a > b^k, \\ O(n^k \log_b n) & \text{daca } a = b^k, \\ O(n^k) & \text{daca } a < b^k. \end{cases}$$

Demonstratia pe tabla

Cautare binara

□ generalizare: $s[p \dots q]$

□ baza: $p \geq q$

□ divide-et-impera

⇒ divide: $m = \lfloor (p + q) / 2 \rfloor$

⇒ subprobleme: dacă $a < s[m]$ atunci caută în $s[p \dots m-1]$,
altfel caută în $s[m+1 \dots q]$

⇒ asamblare: nu există

⇒ complexitate:

- aplicind teorema: $a = 1, b = 2, k = 0 \Rightarrow T(n) = O(\log n)$
- calculind recurenta:

$$T(n) = T(n/2) + 2 = T(n/4) + 4 = \dots = T(1) + 2h = 2\log n + 1$$

Constructia arborelui binar

□ problema

- ⇒ intrare: o lista ordonata crescator $s = (x_0 < x_1 < \dots < x_{n-1})$
- ⇒ iesire: arbore binar de cautare echilibrat care memoreaza s

□ algoritm

- ⇒ generalizare: $s[p \dots q]$
- ⇒ baza: $p > q \Rightarrow$ arborele vid
- ⇒ divide-et-impera
 - divide: $m = \lfloor (p + q) / 2 \rfloor$
 - subprobleme: $s[p \dots m-1] \Rightarrow t_1, s[m+1 \dots q] \Rightarrow t_2$
 - asamblare: construiesc arborele binar t cu radacina $s[m]$, t_1 subarbore stinga si t_2 subarbore dreapta.
 - complexitate:
 - aplicam teorema: $a = 2, b = 2, k = 0 \Rightarrow T(n) = O(n)$

Sortare prin interclasare (Merge sort)

- generalizare: $a[p..q]$
- baza: $p \geq q$
- divide-et-impera
 - ⇒ divide: $m = \lfloor (p + q) / 2 \rfloor$
 - ⇒ subprobleme: $a[p..m]$, $a[m+1..q]$
 - ⇒ asamblare: interclaseaza subsecventele sortate $a[p..m]$ si $a[m+1..q]$
 - initial memoreaza rezultatul interclasarii in $temp$
 - copie din $temp[0..p+q-1]$ in $a[p..q]$
- complexitate:
 - ⇒ timp $a = 2, b = 2, k = 1$ $T(n) = O(n \log n)$
 - ⇒ spatiu suplimentar: $O(n)$

Sortare rapida (Quick sort)

□ generalizare: $a[p..q]$

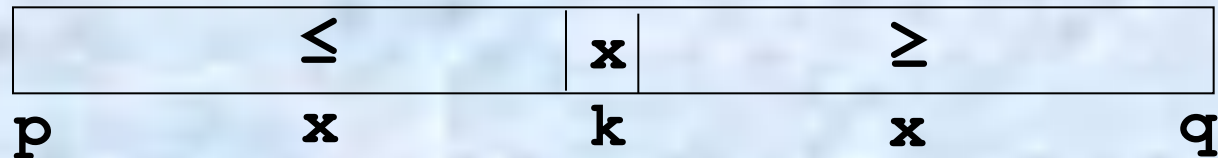
□ baza: $p \geq q$

□ divide-et-impera

⇒ divide: determina k între p și q prin interschimbari a.i.
dupa determinarea lui k avem:

$$\bullet p \leq i \leq k \Rightarrow a[i] \leq a[k]$$

$$\bullet k < j \leq q \Rightarrow a[k] \leq a[j]$$



⇒ subprobleme: $a[p..k-1]$, $a[k+1..q]$

⇒ asamblare: nu exista

Quick sort: partitionare

□ initial:

⇒ $\mathbf{x} \leftarrow \mathbf{a}[\mathbf{p}]$ (se poate alege $\mathbf{x}$ arbitrar din $\mathbf{a}[\mathbf{p} \dots \mathbf{q}]$)

⇒ $\mathbf{i} \leftarrow \mathbf{p} + 1$; $\mathbf{j} \leftarrow \mathbf{q}$

□ pasul curent:

⇒ daca $\mathbf{a}[\mathbf{i}] \leq \mathbf{x}$ atunci $\mathbf{i} \leftarrow \mathbf{i} + 1$

⇒ daca $\mathbf{a}[\mathbf{j}] \geq \mathbf{x}$ atunci $\mathbf{j} \leftarrow \mathbf{j} - 1$

⇒ daca $\mathbf{a}[\mathbf{i}] > \mathbf{x} > \mathbf{a}[\mathbf{j}]$ si $\mathbf{i} < \mathbf{j}$ atunci

- $\mathbf{swap}(\mathbf{a}[\mathbf{i}], \mathbf{a}[\mathbf{j}])$

- $\mathbf{i} \leftarrow \mathbf{i} + 1$

- $\mathbf{j} \leftarrow \mathbf{j} - 1$

□ terminare:

conditia $\mathbf{i} > \mathbf{j}$

operatii $\mathbf{k} \leftarrow \mathbf{i} - 1$

$\mathbf{swap}(\mathbf{a}[\mathbf{p}], \mathbf{a}[\mathbf{k}])$

Quick sort: complexitate

- complexitatea in cazul cel mai nefavorabil: $T(n) = O(n^2)$
- complexitatea medie

$$T^{med}(n) = \begin{cases} (n-1) + \frac{1}{n} \sum_{i=1}^n (T^{med}(i-1) + T^{med}(n-i)) & n \geq 1, \\ 1 & n = 0. \end{cases}$$

Teorema

$$T^{med}(n) = O(n \log n).$$

Selectionare

□ problema

⇒ intrare: o lista $\mathbf{a} = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1})$

⇒ iesire: cel de-al $k+1$ -lea numar cel mai mic

□ algoritm

⇒ pp. $i \neq j \Rightarrow a[i] \neq a[j]$

⇒ cel de-al $k+1$ -lea numar cel mai mic este caracterizat de:

- $(\forall i) i < k \Rightarrow a[i] \leq a[k]$
- $(\forall j) k < j \Rightarrow a[k] \leq a[j]$

⇒ divide-et-impera

- divide: **partitioneaza**($\mathbf{a}, \mathbf{p}, \mathbf{q}, \mathbf{k1}$)
- subprobleme: daca $\mathbf{k1} = \mathbf{k}$ atunci stop; daca $\mathbf{k} < \mathbf{k1}$ atunci selecteaza din $\mathbf{a}[\mathbf{p} \dots \mathbf{k1}-1]$, altfel selecteaza din $\mathbf{a}[\mathbf{k1}+1 \dots \mathbf{q}]$
- asamblare: nu exista

⇒ complexitate: $n + k \log(n/k) + (n-k) \log(n/(n-k))$

Transformata Fourier discreta I

□ descrierea unui semnal

⇒ domeniul timp: $f(t)$

⇒ domeniul frecvență: $F(\nu)$

□ Transformata Fourier directă:

$$F(\nu) = \mathfrak{F}[f(t)] = \int_{-\infty}^{+\infty} f(t) e^{-2\pi i \nu t} dt$$

□ Transformata Fourier inversă

$$f(t) = \mathfrak{F}^{-1}[F(\nu)] = \int_{-\infty}^{+\infty} F(\nu) e^{2\pi i \nu t} d\nu$$

Transformata Fourier discreta - aplicatie

□ *Filtrarea imaginilor*

- ⇒ transformata Fourier a unei functii este echivalenta cu reprezentarea ca o suma de functii sinus
- ⇒ eliminand frecventele foarte inalte sau foarte joase nedorite (adica eliminand niste functii sinus) si aplicand transformata Fourier inversa pentru a reveni in domeniul timp, se obtine o filtrare a imaginilor prin eliminarea “zgomotelor”

□ Compresia imaginilor

- ⇒ o imagine filtrata este mult mai uniforma si deci va necesita mai putini biti pentru a fi memorata

Transformata Fourier discreta II

□ cazul discret

$$\Rightarrow x_k = f(t_k) \quad k=0, \dots, n-1$$

$$\Rightarrow t_k = kT, \quad T = \text{perioada de timp la care se fac masuratorile}$$

$$\mathfrak{F}[f(t)] = T \sum_{k=0}^{n-1} x_k e^{-\frac{2\pi i j k}{n}}$$

notatie

$$W_j = \sum_{k=0}^{n-1} x_k e^{-\frac{2\pi i j k}{n}}$$

□ asociem polinomul

$$f(Y) = x_0 + x_1 Y + \boxtimes + x_{n-1} Y^{n-1}$$

Transformata Fourier discreta III

□ rolul radacinilor unitatii de ordinul n

$$w_j = e^{\frac{2\pi i j}{n}}$$
$$W_{n-j} = f(w_j)$$

radacina de ordinul n
a unitatii

valoarea polinomului
in radacina de ord. n
a
unitatii

Transformata Fourier discreta IV

□ calculul valorilor prin divide-et-impera

$$f(Y) = (x_0 + x_2 Y^2 + \dots) + Y(x_1 + x_3 Y^2 + \dots)$$

$$g(Y) = x_0 + x_2 Y + \dots + x_{n-2} Y^{n/2-1}$$

$$h(Y) = x_1 + x_3 Y + \dots + x_{n-1} Y^{n/2-1}$$

$$f(w^j) = g(w^{2j}) + wh(w^{2j})$$

□ $a = b = 2, k = 1 \Rightarrow W_j$ poate fi calculat cu $O(n \log n)$ inmultiri

Chess board cover problem

There is a chess board with a dimension of 2^m (i.e., it consists of 2^{2m} squares) with a hole, i.e., one arbitrary square is removed. We have a number of L shaped tiles (see figure 4.3) and the task is to cover the board by these tiles. (The orientation of a tile isn't important.)
(Michalewicz&Fogel, How to solve it: Modern heuristics)

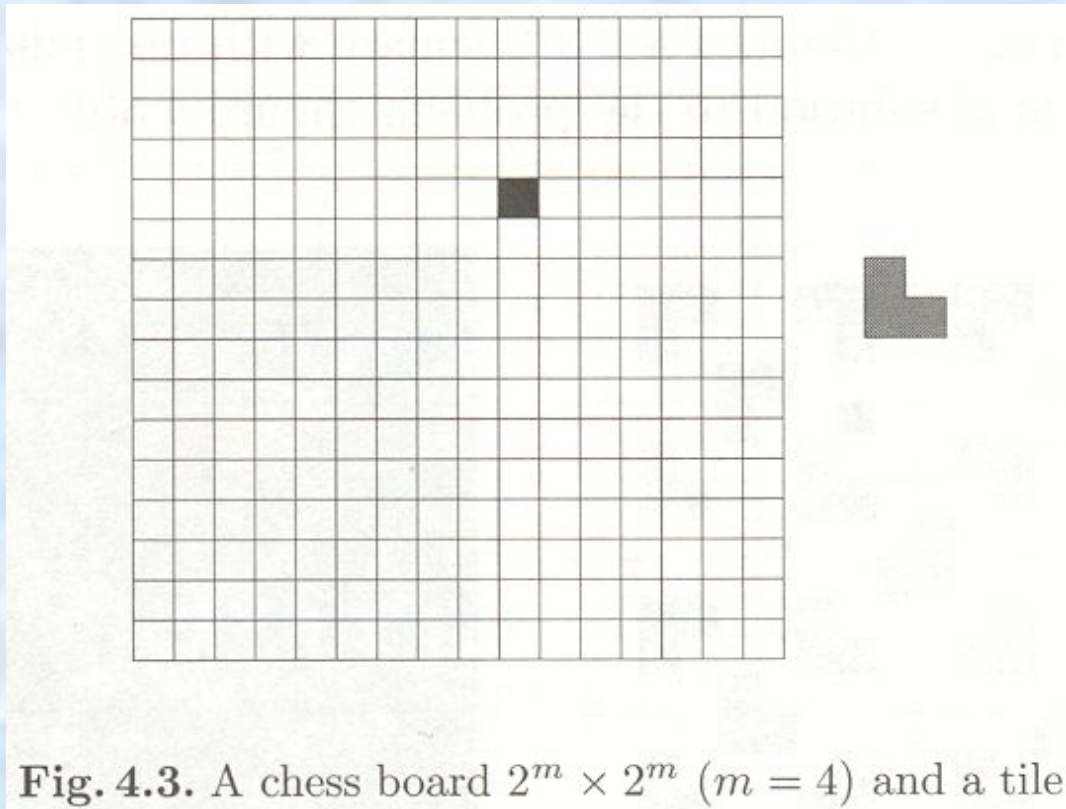


Fig. 4.3. A chess board $2^m \times 2^m$ ($m = 4$) and a tile

Chess board cover problem

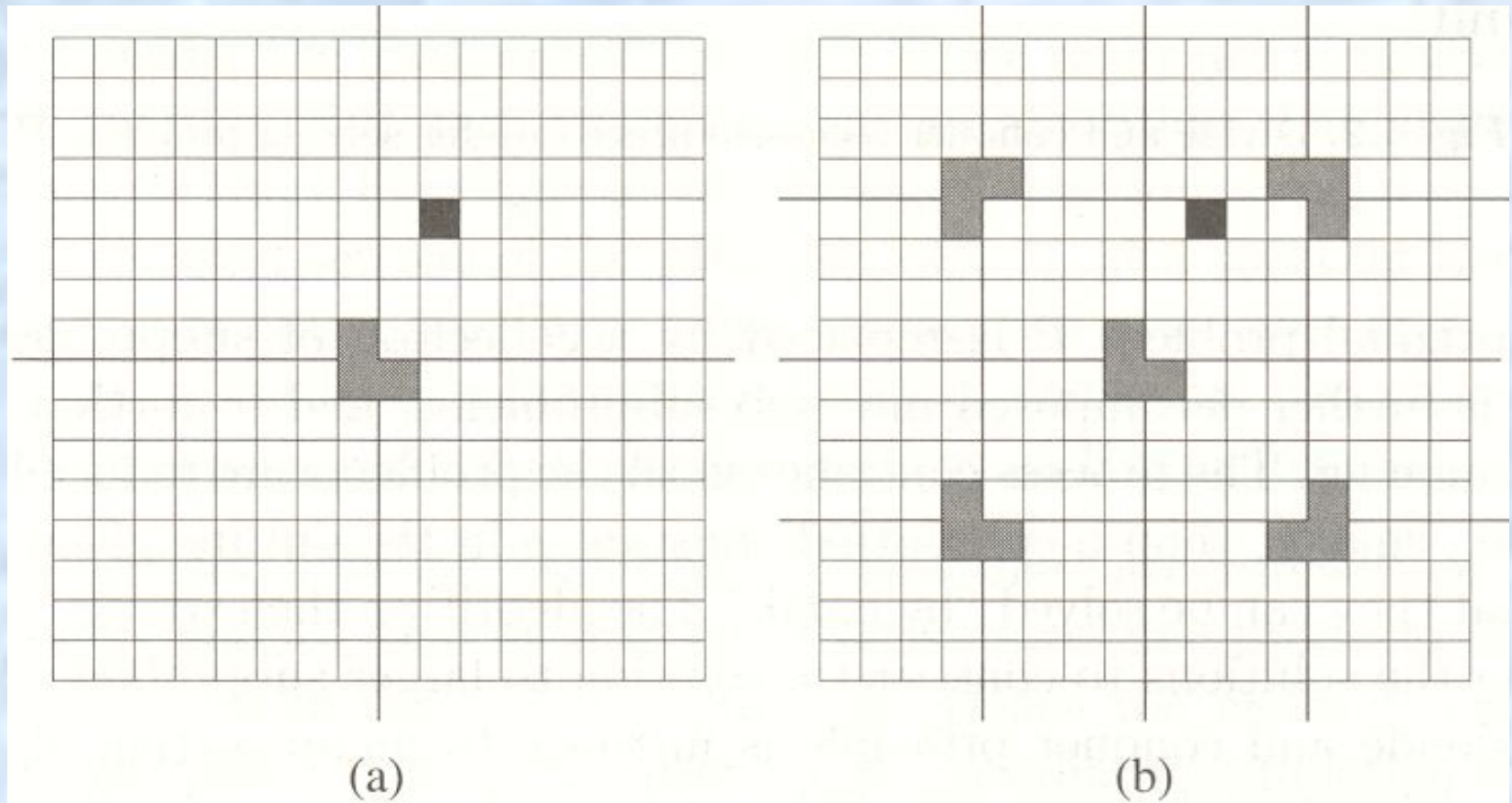
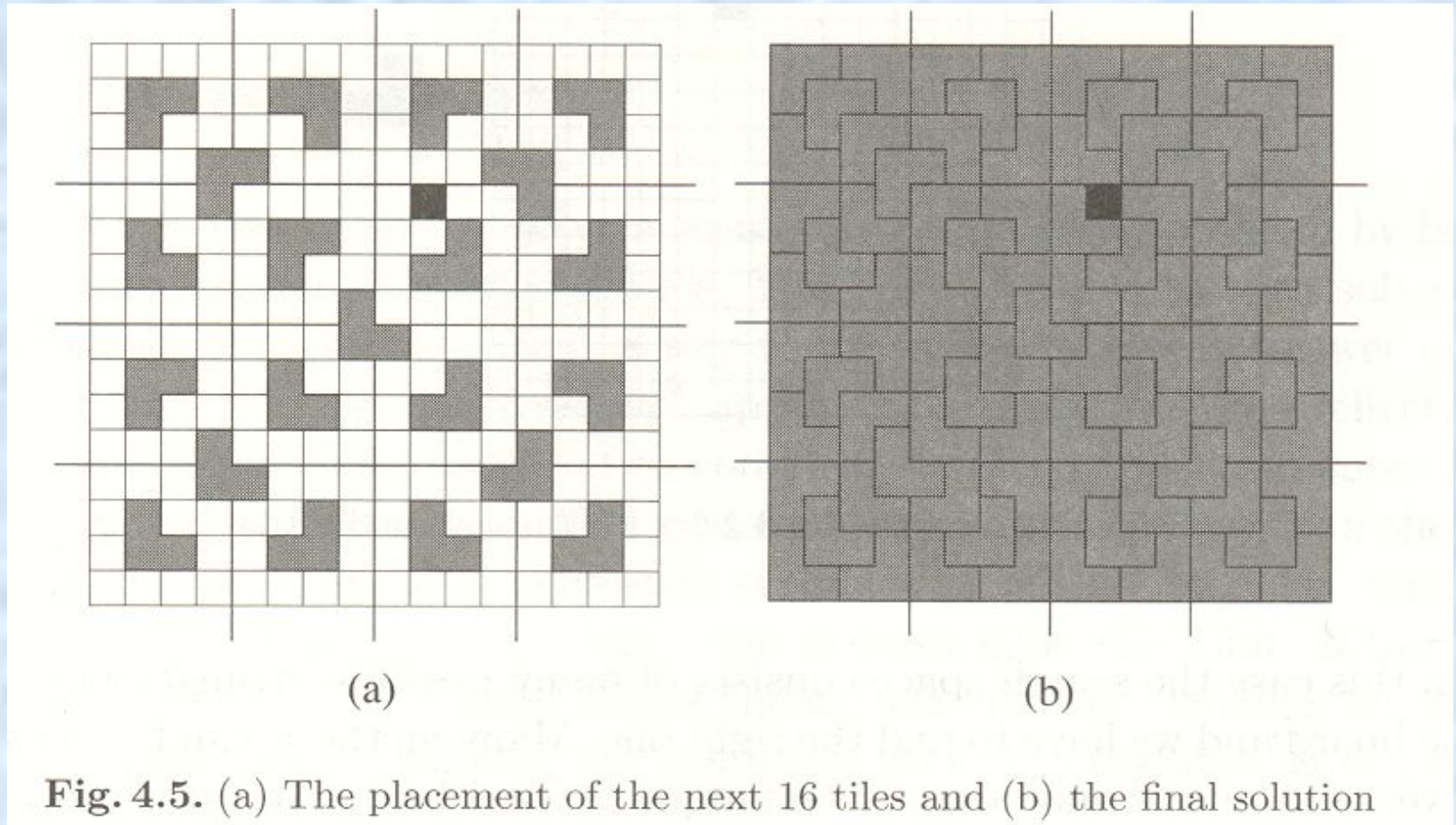


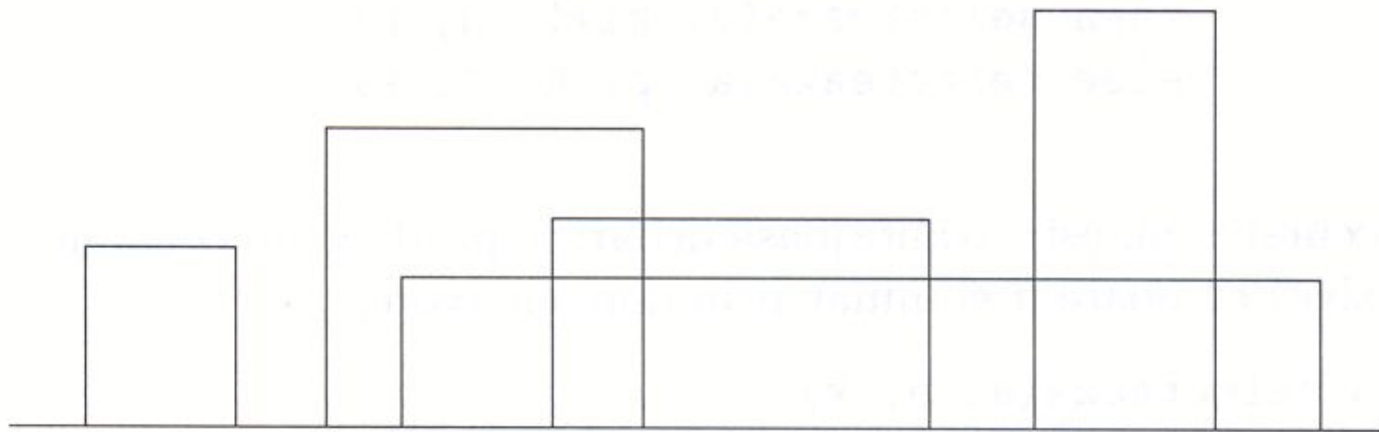
Fig. 4.4. The placement of the first tile (a) and the next four tiles (b)

Chess board cover problem

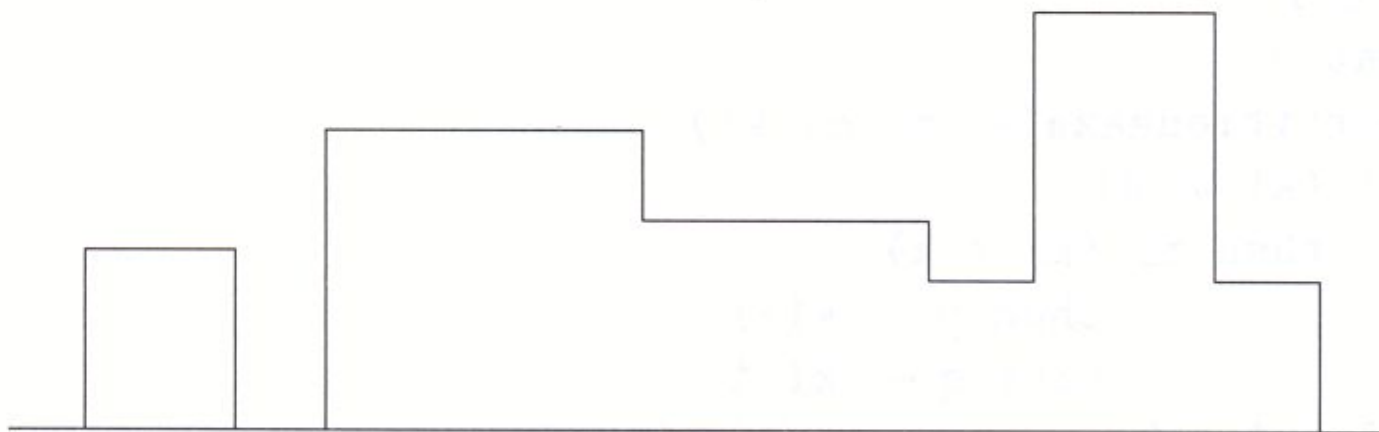


Timp de executie: $a = 4, b = 2, k = 0 \Rightarrow T(n) = O(n^2)$

Linia orizontului



a)



b)

Figura 10.2: Linia orizontului

Linia orizontului

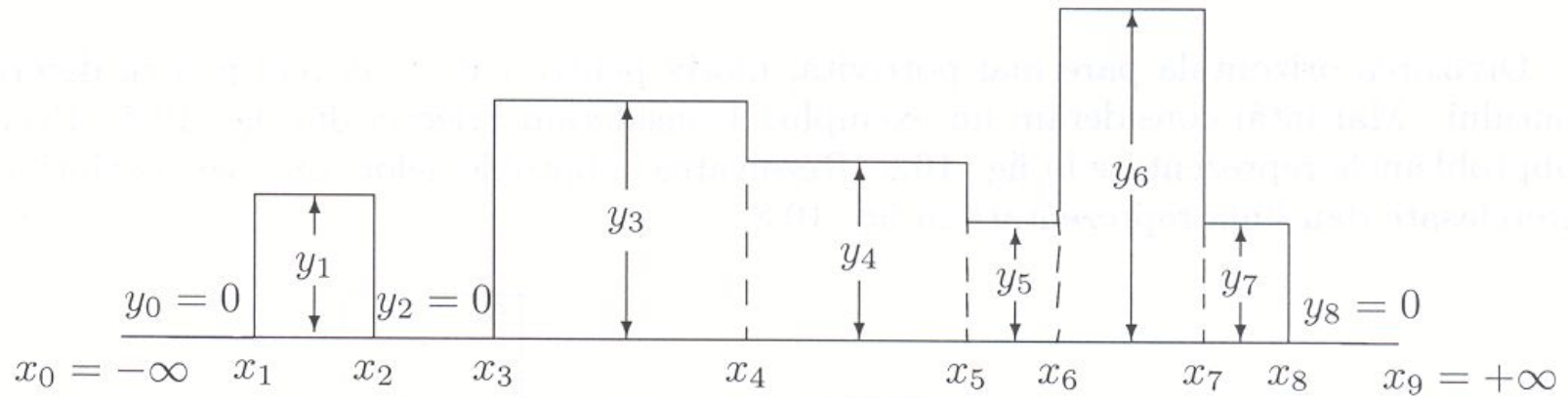
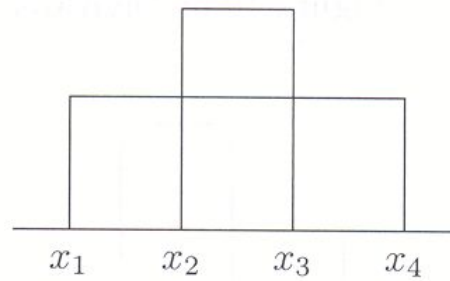
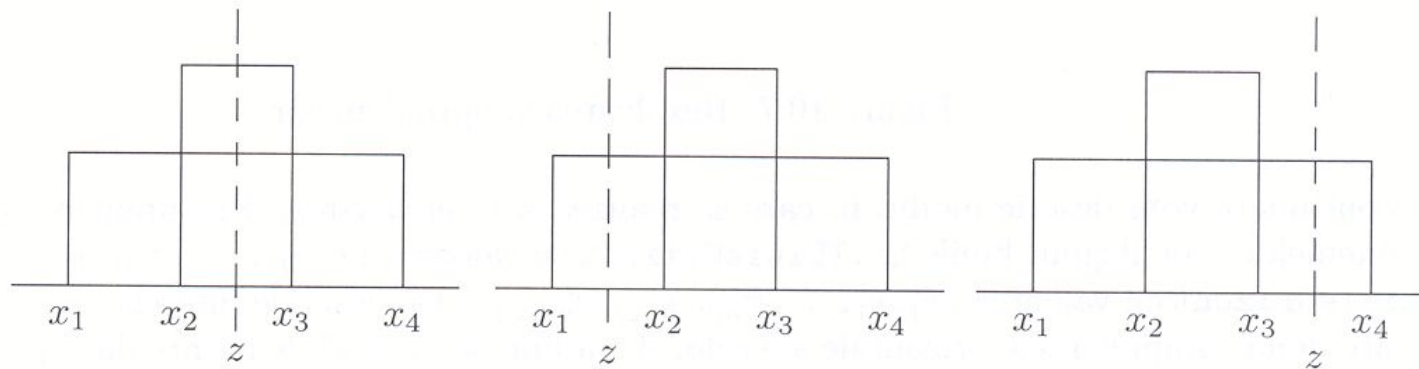


Figura 10.3: Reprezentarea liniei orizontului

Linia orizontului



a)



b)

Figura 10.4: Un exemplu când divizarea verticală nu reduce numărul de dreptunghiuri

Linia orizontului

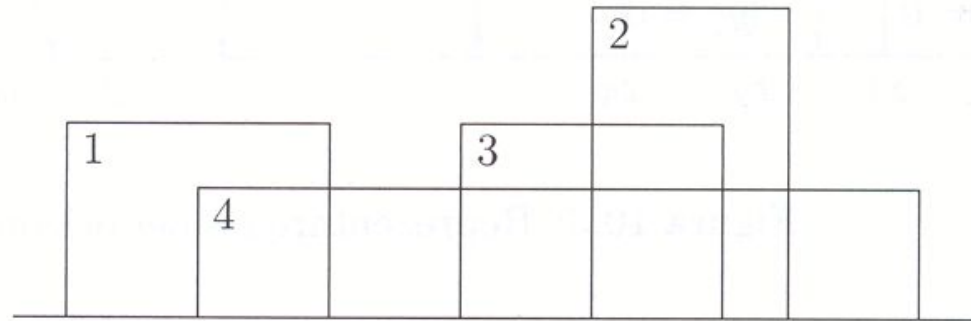
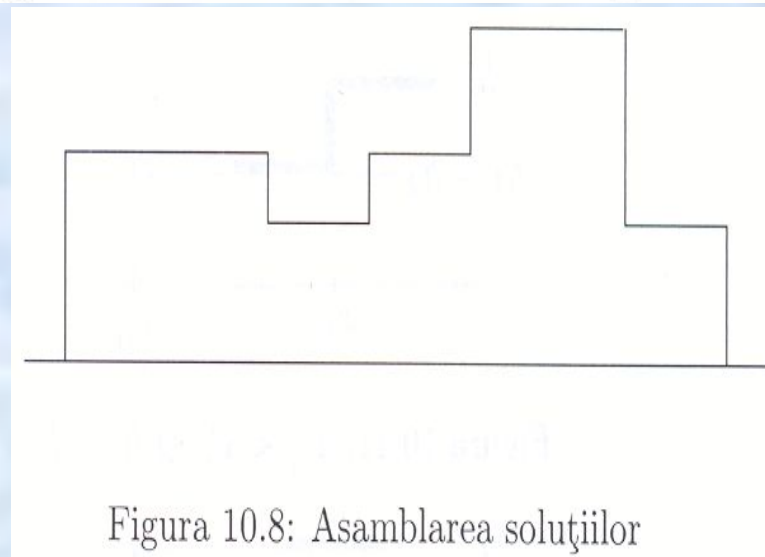
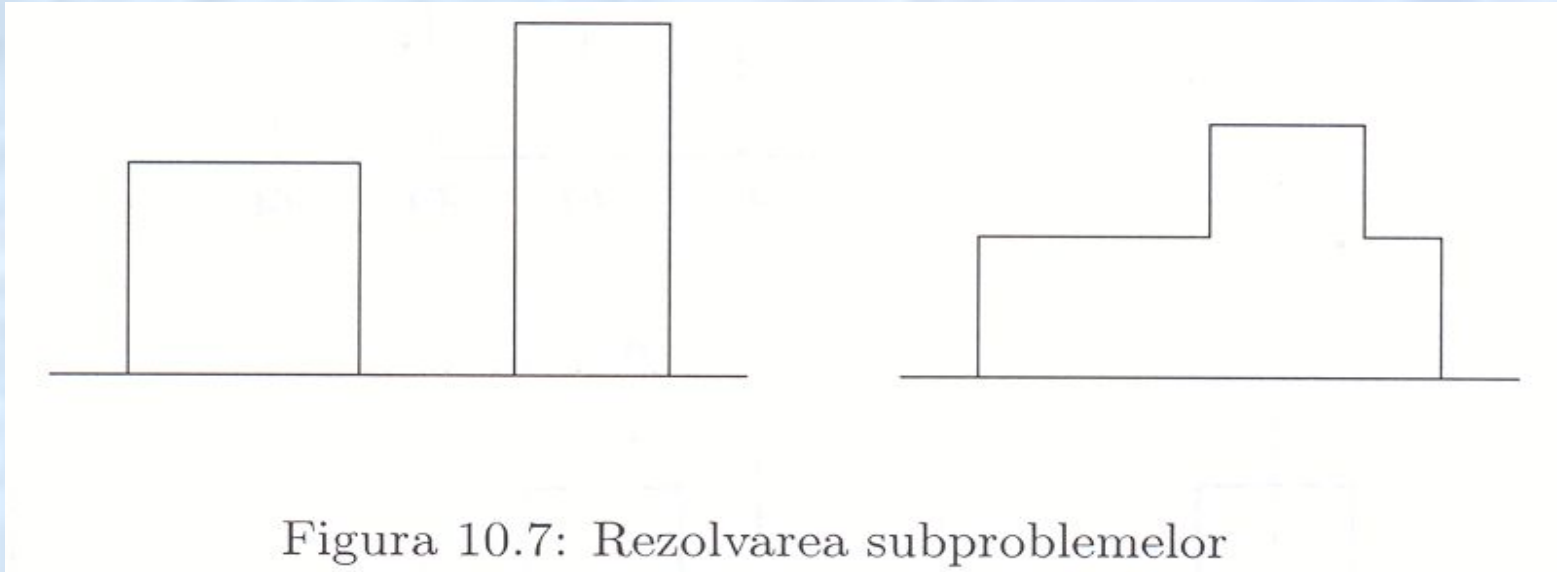


Figura 10.5: Problema inițială



Figura 10.6: Divizarea în subprobleme

Linia orizontului



Timp de executie: $a = 2, b = 2, k = 1 \Rightarrow T(n) = O(n \log n)$