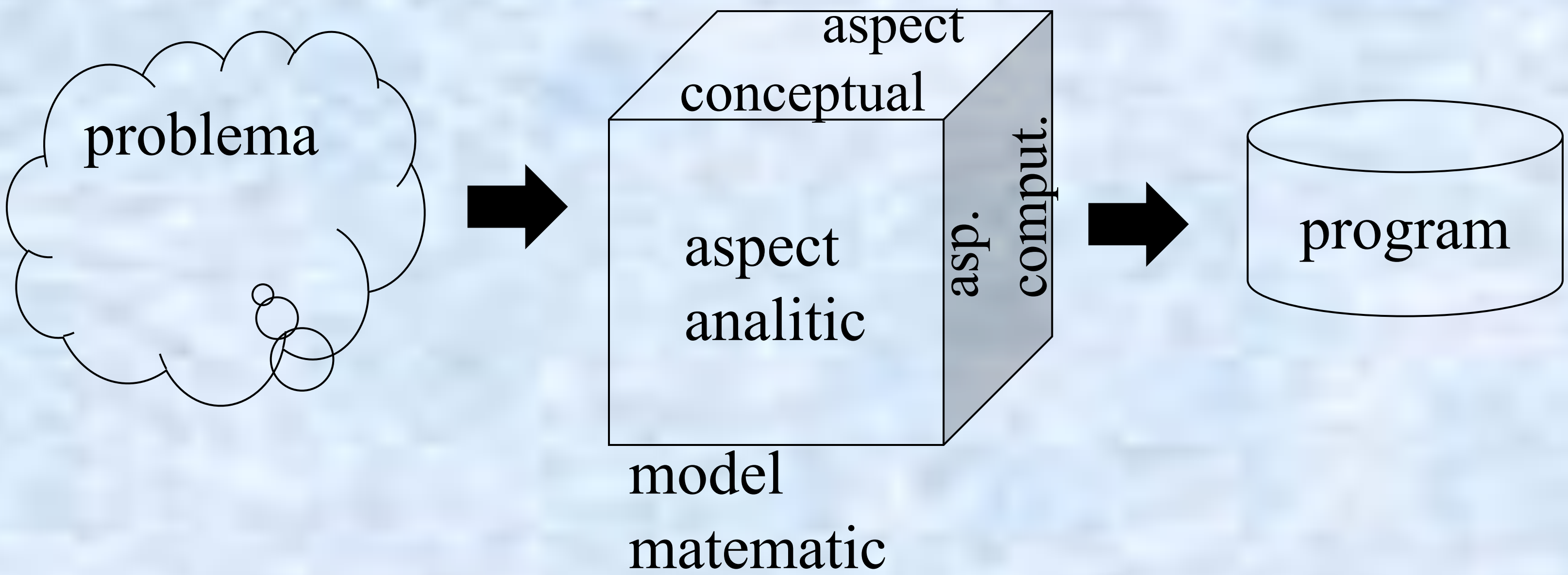
Paradigme de proiectare a algoritmilor

- 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 $P_1(n_1), ..., P_a(n_a)$ cu $n_i \leq n/b$, $b > 1$
    - ***rezolva*** $P_1(n_1), ..., P_a(n_a)$ in aceeasi maniera si obtine solutiile $S_1, ..., S_a$
    - ***asambleaza*** $S_1, ..., S_a$ pentru a obtine solutia 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 $O(n^k)$

$$T(n)=\begin{cases} O(1) & \text{daca} \quad n \le n_0, \\ a\mathrm{T}\left(\dfrac{n}{b}\right) + \mathrm{O}(n^k) & \text{daca} \quad n > n_0. \end{cases}$$

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

Demonstratia pe tabla

Cautare binara

- generalizare: `s[p..q]`
- baza: `p ≥ q`
- divide-et-impera
  - ⇨ divide: `m = [(p + q)/2]`
  - ⇨ subprobleme: daca `a < s[m]` atunci cauta in `s[p..m-1]`, altfel cauta in `s[m+1..q]`
  - ⇨ asamblare: nu exista
  - ⇨ complexitate:
    - aplicind teorema: a = 1, b = 2, k = 0 ⇒ T(n) = O(log n)
    - calculind recurenta:

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

Constructia arborelui binar

- problema
  - ⇨ intrare: o lista ordonata crescator $s = (x_0 < x_1 < \ldots < x_{n-1})$
  - ⇨ iesire: arbore binar de cautare echilibrat care memoreaza s
- algoritm
  - ⇨ generalizare: **`s[p..q]`**
  - ⇨ baza: **`p > q`** $\Rightarrow$ arborele vid
  - ⇨ divide-et-impera
    - divide: **`m = [(p + q)/2]`**
    - subprobleme: **`s[p..m-1]`** $\Rightarrow t_1$, **`s[m+1..q]`** $\Rightarrow t_2$
    - asamblare: construieste 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 ≥ q`**
- divide-et-impera
  - ⇨ divide: **`m = [(p + q)/2]`**
  - ⇨ 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 ≥ q`**
- divide-et-impera
  - ⇨ divide: determina `k` intre `p` si `q` prin interschimbari a.i. dupa determinarea lui `k` avem:
    - **`p ≤ i ≤ k ⇒ a[i] ≤ a[k]`**
    - **`k < j ≤ q ⇒ a[k] ≤ a[j]`**

≤	x	≥

p **x** **k** **x** **q**

⇨ subprobleme: **`a[p..k-1], a[k+1..q]`**

⇨ asamblare: nu exista

Quick sort: partitionare

- initial:
  - ⇨ **`x ← a[p]`** (se poate alege **`x`** arbitrar din **`a[p..q])`**
  - ⇨ **`i ← p+1 ; j ← q`**
- pasul curent:
  - ⇨ daca **`a[i]`** $\leq$ **`x`** atunci **`i ← i+1`**
  - ⇨ daca **`a[j]`** $\geq$ **`x`** atunci **`j ← j-1`**
  - ⇨ daca **`a[i] > x > a[j]`** si **`i < j`** atunci
    - **`swap(a[i], a[j])`**
    - **`i ← i+1`**
    - **`j ← j-1`**
- terminare:

  conditia **`i > j`**

  operatii **`k ← i-1`**

  `swap(a[p], a[k])`

Quick sort: complexitate

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

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

Teorema

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

Selectionare

- problema
  - ⇨ intrare: o lista $\mathbf{a} = (\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1})$
  - ⇨ iesire: cel de-al k+1-lea numar cel mai mic
- algoritm
  - ⇨ pp. $\mathbf{i} \neq \mathbf{j} \Rightarrow \mathbf{a[i]} \neq \mathbf{a[j]}$
  - ⇨ cel de-al k+1-lea numar cel mai mic este caracterizat de:
    - $(\forall \mathbf{i})\mathbf{i} < \mathbf{k} \Rightarrow \mathbf{a[i]} <= \mathbf{a[k]}$
    - $(\forall \mathbf{j})\mathbf{k} < \mathbf{j} \Rightarrow \mathbf{a[k]} <= \mathbf{a[j]}$
  - ⇨ divide-et-impera
    - divide: **partitioneaza(a, p, q, k1)**
    - subprobleme: daca **k1 = k** atunci stop; daca **k < k1** atunci selecteaza din **a[p..k1-1]**, altfel selecteaza din **a[k1+1..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 frecventa: F(ν)
- Transformata Fourier directa:

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

- Transformata Fourier inversa

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

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
  - ⇨ $x_k = f(t_k)$ k=0,…,n-1
  - ⇨ $t_k = kT$, T = perioada de timp la care se fac masuratorile

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

notatie

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

- asociem polinomul

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

Transformata Fourier discreta III

- rolul radacinilor unitatii de ordinul n

$$w_j = e^{\frac{2\pi ij}{n}}$$

$$W_{n-j} = f(w_j)$$

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

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

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

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

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

- a = b = 2, k = 1 $\Rightarrow$ Wj 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)

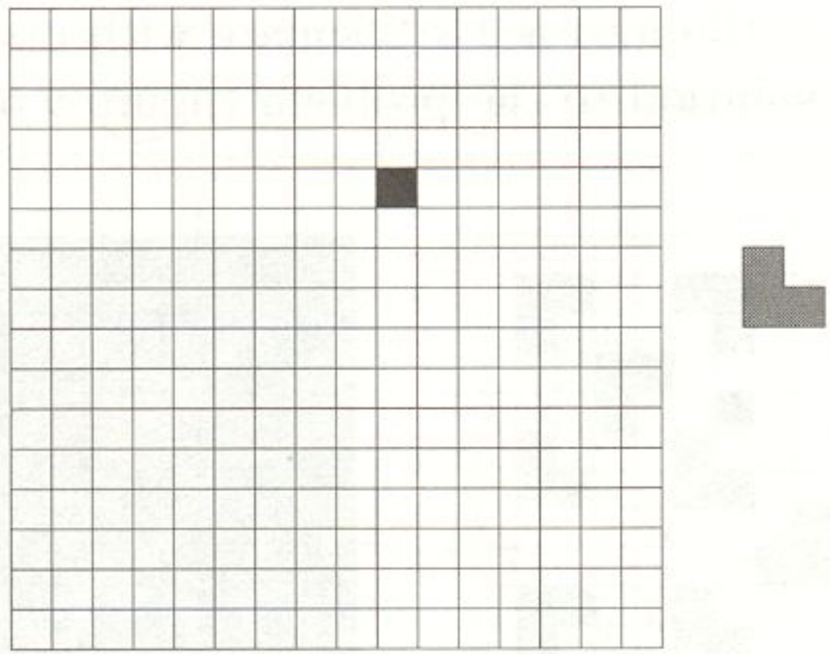


Fig. 4.3. A chess board $2^m \times 2^m$ $(m = 4)$ and a tile

Chess board cover problem

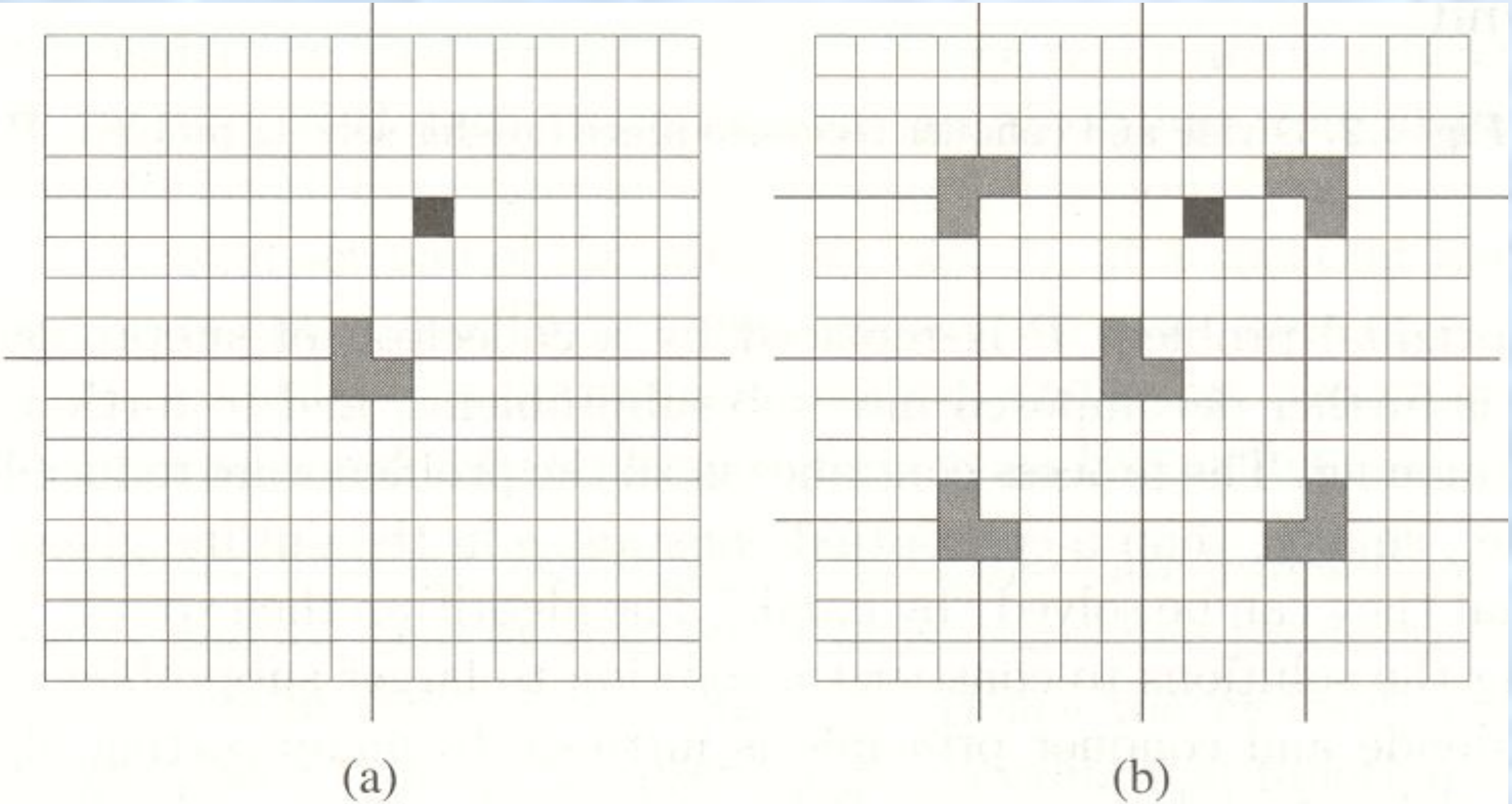


Fig. 4.4. The placement of the first tile (a) and the next four tiles (b)

Chess board cover problem

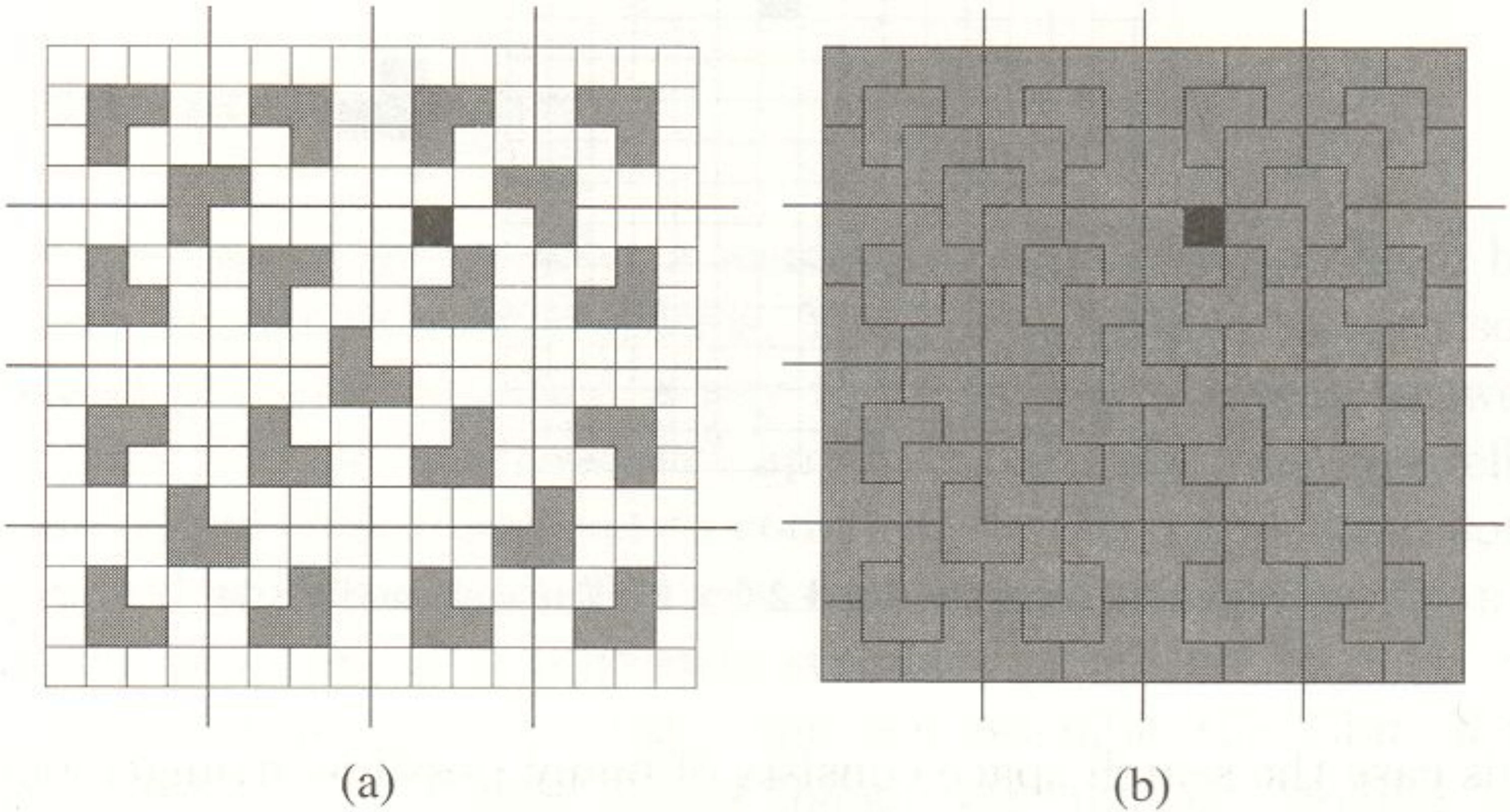


Fig. 4.5. (a) The placement of the next 16 tiles and (b) the final solution

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

Linia orizontului

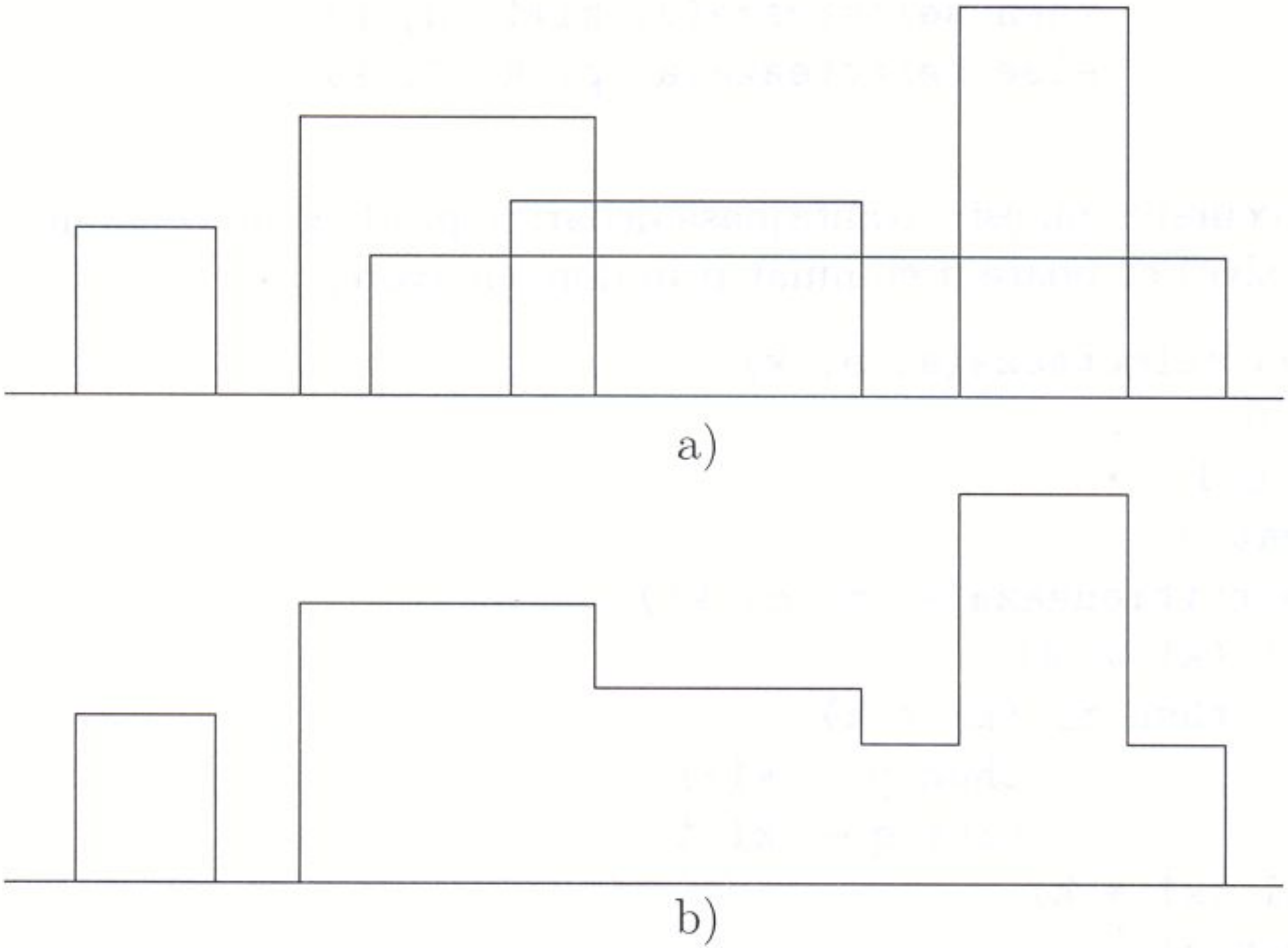


Figura 10.2: Linia orizontului

Linia orizontului

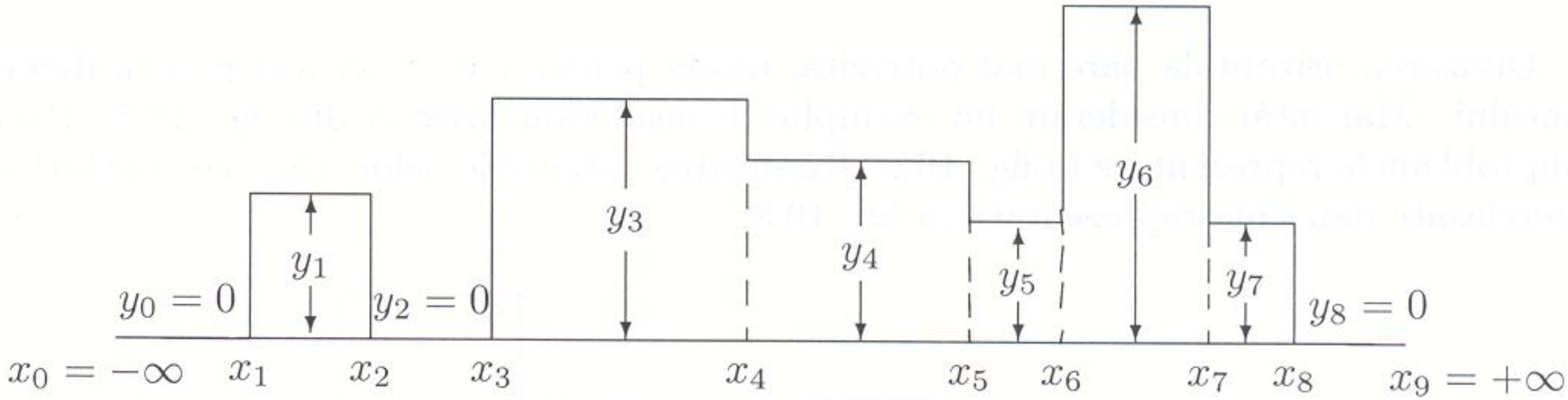


Figura 10.3: Reprezentarea liniei orizontului

Linia orizontului

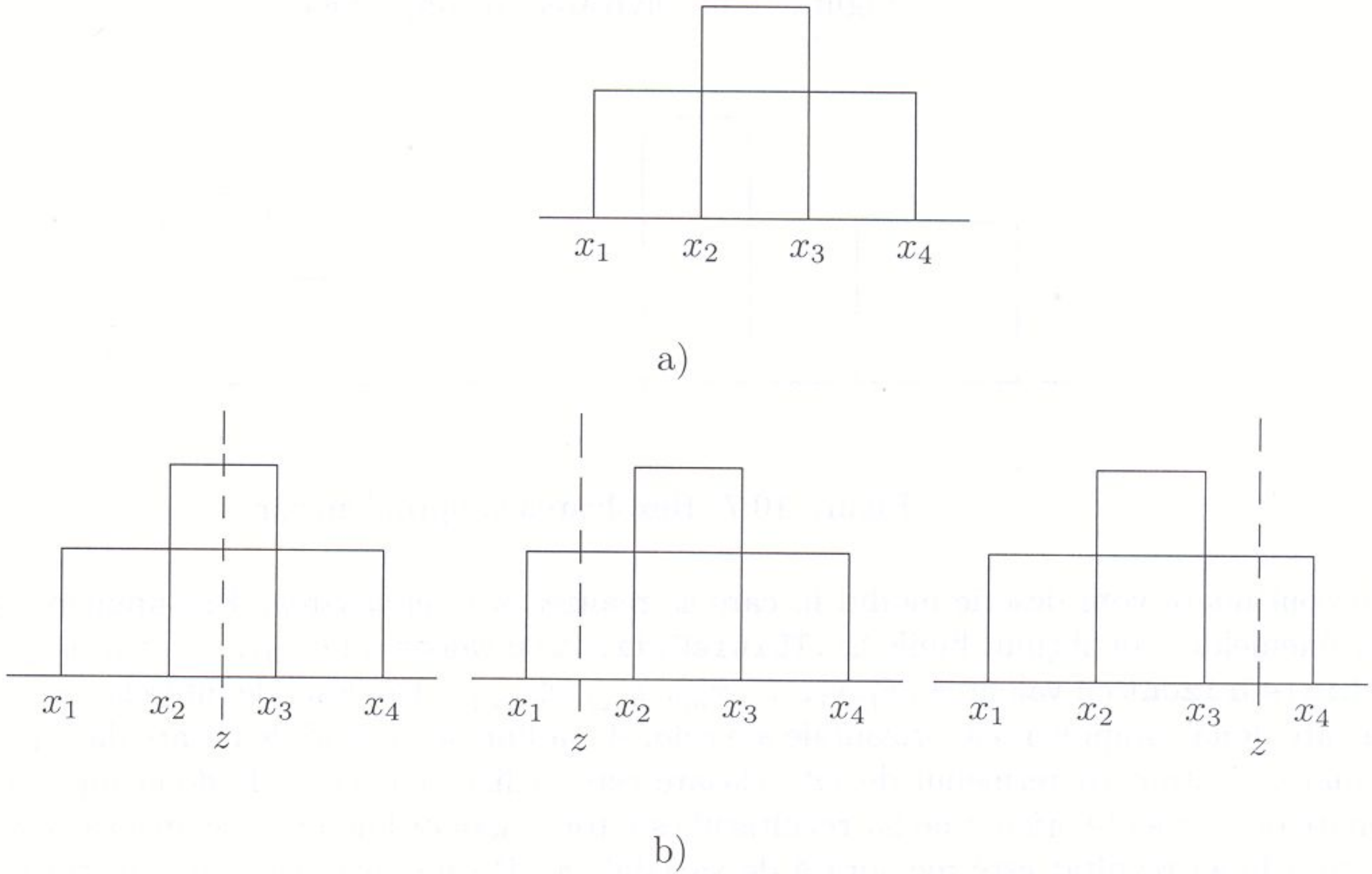


Figura 10.4: Un exemplu când divizarea verticală nu reduce numărul de dreptunghiuri

Linia orizontului

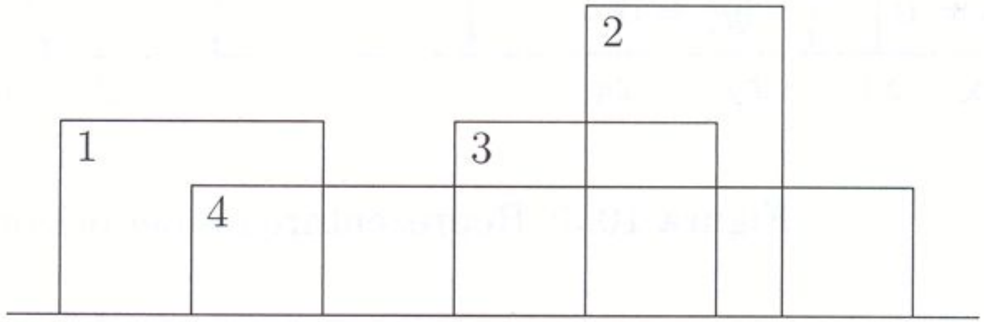


Figura 10.5: Problema iniţială

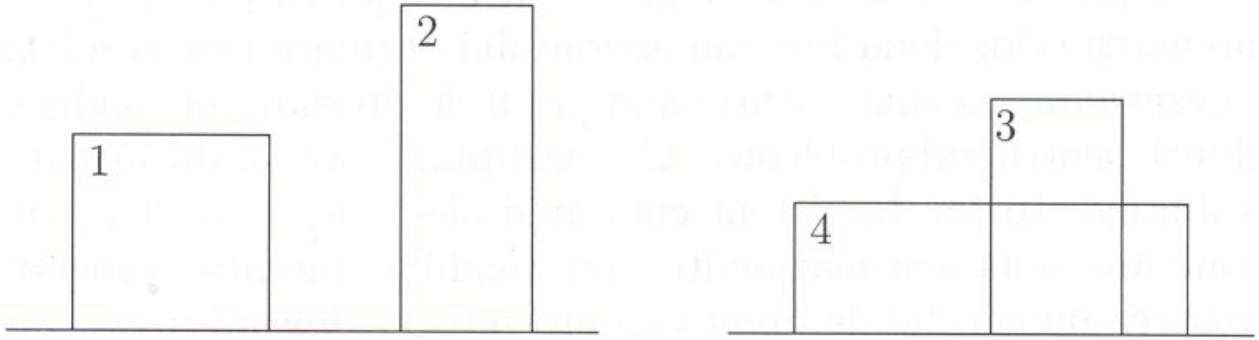


Figura 10.6: Divizarea în subprobleme

Linia orizontului

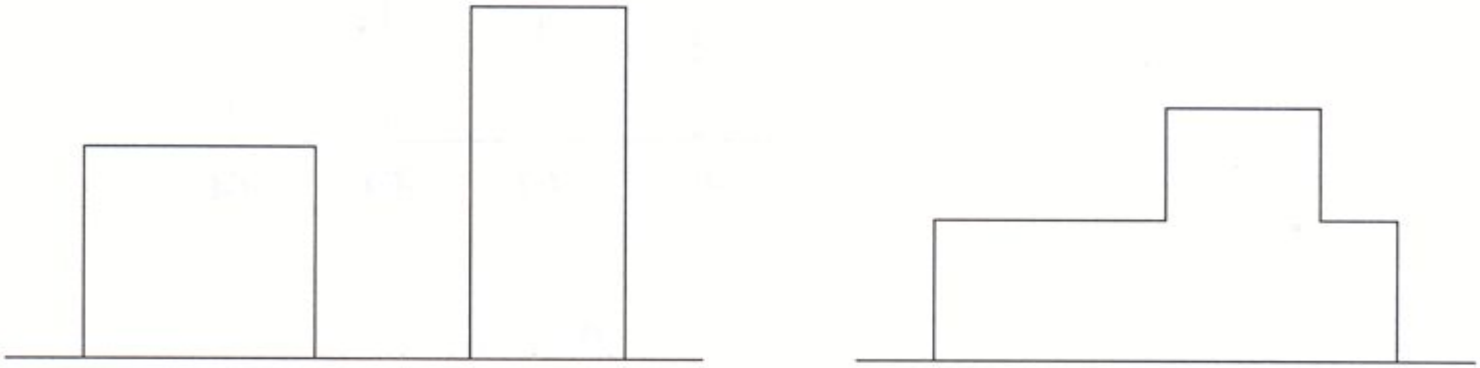


Figura 10.7: Rezolvarea subproblemelor

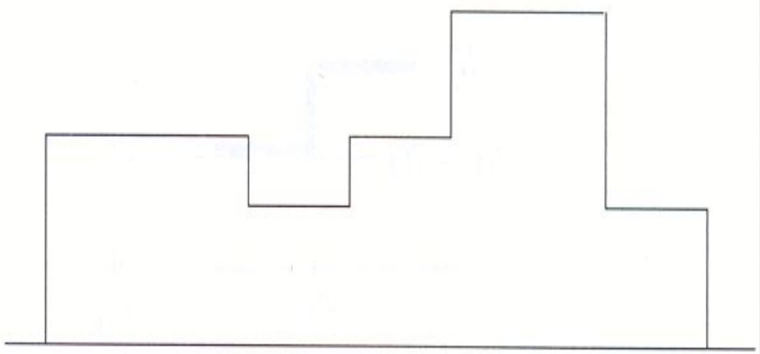


Figura 10.8: Asamblarea soluţiilor

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