

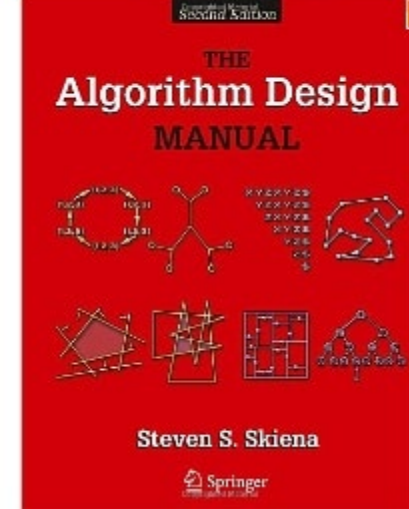
CURS 6:

Analiza metodelor de sortare

Motivatie

S. Skiena – The Algorithm Design Manual

<http://sist.sysu.edu.cn/~isslxm/DSA/textbook/Skienna.-TheAlgorithmDesignManual.pdf>



Interview Problems

- 4-40. [3] If you are given a million integers to sort, what algorithm would you use to sort them? How much time and memory would that consume?
- 4-41. [3] Describe advantages and disadvantages of the most popular sorting algorithms.

Structura

- Problema sortării
- Sortare prin inserție – analiza corectitudinii și a eficienței
- Sortare prin selecție – analiza corectitudinii și a eficienței
- Sortare prin interschimbarea elementelor vecine – analiza corectitudinii și a eficienței

Problema sortării

Considerăm:

- O secvență de “entități” (numere, structuri conținând mai multe informații etc)
- Fiecare entitate posedă o caracteristică numită **cheie de sortare**. Valorile posibile ale cheii de sortare aparțin unei mulțimi pe care există o **relație totală de ordine**
- **Sortarea secvenței** = **aranjarea elementelor** astfel încât valorile cheii de sortare să fie în ordine crescătoare (sau descrescătoare)

Problema sortării

Exemple:

1. Secvența de numere: (5,8,3,1,6)
Cheia de sortare este chiar elementul din secvență
Rezultatul sortării crescătoare este (1,3,5,6,8)
2. Secvența de entități (numite înregistrări sau articole) ce conțin două câmpuri: nume și nota
(Paul,9), (Ioana, 10), (Victor,8), (Ana,9)

In acest caz există două criterii posibile de sortare:
nume și nota

Sortare crescătoare după nume:

((Ana,9),(Ioana,10),(Paul,9),(Victor,8))

Sortare descrescătoare după notă:

((Ioana,10),(Paul,9),(Ana,9),(Victor,8))

Problema sortării

Ceva mai formal...

Ordonarea (crescătoare) a secvenței (x_1, x_2, \dots, x_n) este echivalentă cu găsirea unei permutări $(p(1), p(2), \dots, p(n))$ a indicilor astfel încât:

$$\text{cheie}(x_{p(1)}) \leq \text{cheie}(x_{p(2)}) \leq \dots \leq \text{cheie}(x_{p(n)})$$

În cele ce urmează vom considera cheia de sortare ca fiind reprezentată de întregul element (secvența este constituită din valori aparținând unei mulțimi pe care există o relație de ordine)

În această ipoteză secvența este considerată ordonată crescător dacă satisface:

$$x_{p(1)} \leq x_{p(2)} \leq \dots \leq x_{p(n)}$$

Problema sortării

Alte ipoteze:

- Presupunem ca secvența este stocată sub forma unui tablou astfel că este asigurat accesul rapid la oricare dintre elementele ei. Aceasta înseamnă că este vorba despre **sortare internă**. **Sortarea externă** corespunde cazului în care nu se poate accesa simultan întreaga secvență (de exemplu secvența este foarte mare astfel că nu poate fi încărcată în întregime în memoria internă a calculatorului) ceea ce necesită metode specifice de sortare
- Vom analiza doar metode care **transformă tabloul curent fără a construi un alt tablou** (spațiul adițional de memorie are dimensiunea de ordin $O(1)$).

Proprietăți ale metodelor de sortare

1. Eficiența

Metoda de sortare ar trebui să necesite un volum rezonabil de resurse (timp de execuție)

2. Stabilitate

O metoda de sortare este stabilă dacă păstrează ordinea relativă a elementelor având aceeași valoare a cheii de sortare

3. Simplitate

Metoda de sortare ar trebui să fie simplă de înțeles și de implementat

4. Naturalitate

O metodă de sortare este considerată naturală dacă numărul de operații necesare este proporțional cu gradul de “dezordine” al secvenței inițiale (care poate fi măsurat prin numărul de inversiuni din permutarea corespunzătoare secvenței sortate)

Stabilitate

Exemplu:

Configurația inițială:

$((\text{Ana}, 9), (\text{Ioana}, 10), (\text{Paul}, 9), (\text{Victor}, 8))$

Sortare stabilă:

$((\text{Ioana}, 10), (\text{Ana}, 9), (\text{Paul}, 9), (\text{Victor}, 8))$

Sortare instabilă:

$((\text{Ioana}, 10), (\text{Paul}, 9), (\text{Ana}, 9), (\text{Victor}, 8))$

Metode elementare de sortare

Sunt simple, intuitive dar nu foarte eficiente...

Reprezintă, totuși, puncte de pornire pentru metodele mai avansate.

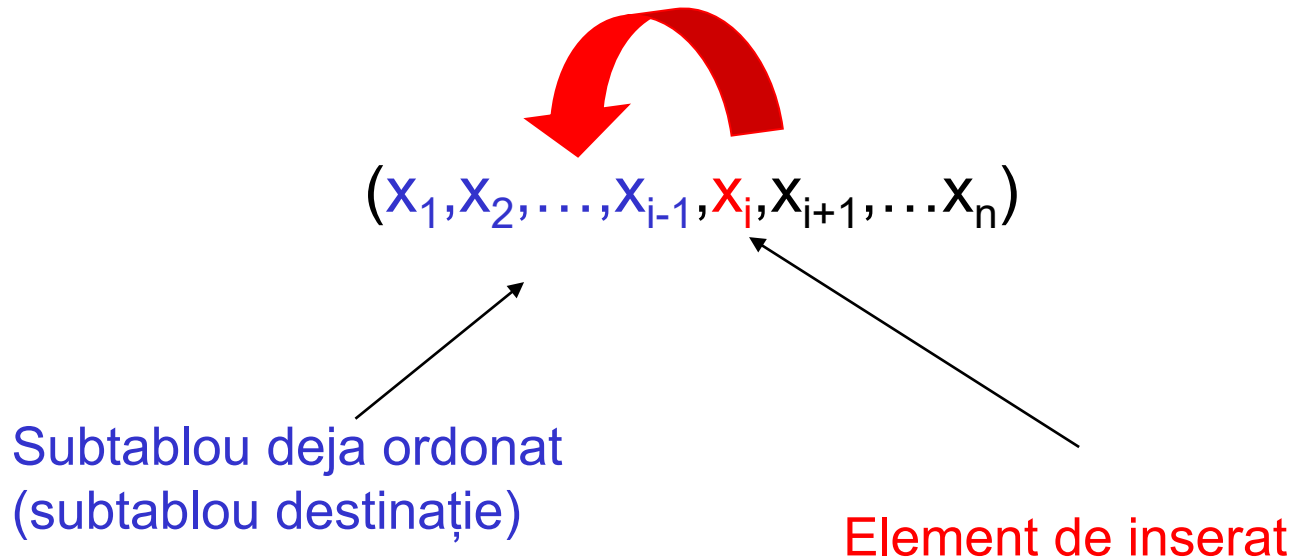
Exemple:

- Sortare prin inserție
- Sortare prin selecție
- Sortare prin interschimbarea elementelor vecine

Sortare prin inserție

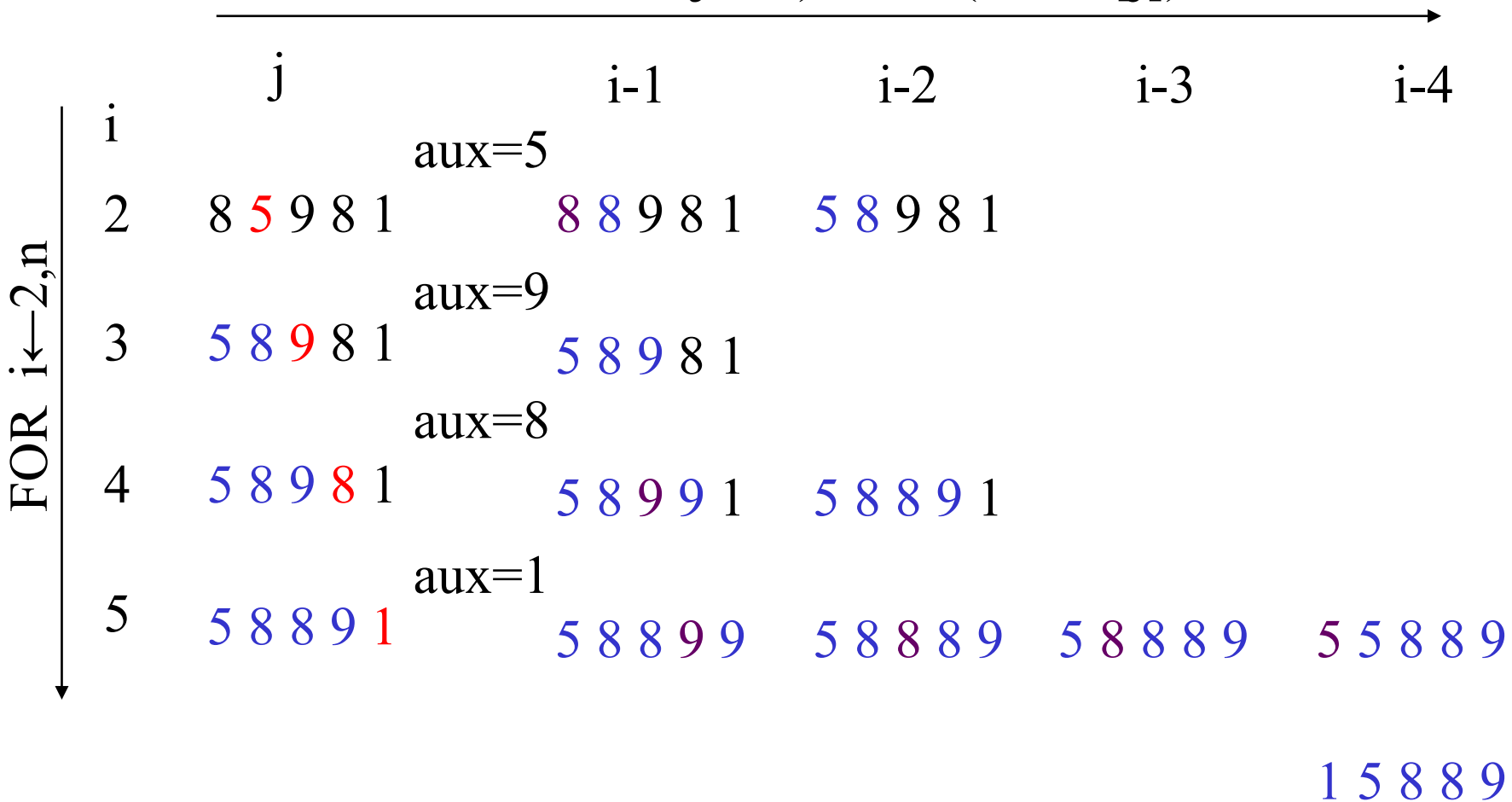
Idee de bază:

Fiecare element al tabloului, începând cu al doilea, este inserat în subtabloul care îl precede astfel încât acesta să rămână ordonat:



Sortare prin inserție - exemplu

WHILE ($j \geq 1$) AND ($aux < x[j]$)



Sortare prin inserție – algoritm

Structura generală

```
FOR i ← 2,n DO  
<insereaza x[i] în subtabloul  
  x[1..i-1] astfel încât x[1..i]  
  să fie sortat>  
ENDFOR
```

Algoritm

```
Insertie(x[1..n])  
FOR i ← 2,n DO  
  aux ← x[i]  
  j ← i-1  
  WHILE (j>=1) AND (aux<x[j]) DO  
    x[j+1] ← x[j]  
    j ← j-1  
  ENDWHILE  
  x[j+1] ← aux  
ENDFOR  
RETURN x[1..n]
```

Sortare prin inserție – variantă

```
Insertie(x[1..n])
FOR i ← 2,n DO
  aux ← x[i]
  j ← i-1
  WHILE (j>=1) AND (aux<x[j]) DO
    x[j+1] ← x[j]
    j ← j-1
  ENDWHILE
  x[j+1] ← aux
ENDFOR
RETURN x[1..n]
```

Varianta (x[0] este folosit ca zonă de manevră):

```
Insertie2(x[0..n])
FOR i ← 2,n DO
  x[0]←x[i]
  j:=i-1
  WHILE (x[0]<x[j]) DO
    x[j+1] ← x[j]
    j ← j-1
  ENDWHILE
  x[j+1] ← x[0]
ENDFOR
RETURN x[1..n]
```

Sortare prin insertie – verificare corectitudine

Insertie($x[1..n]$)

$i \leftarrow 2$

Inv: $\{x[1..i-1] \text{ e sortat}\}$

WHILE $i \leq n$

$aux \leftarrow x[i]$

$j \leftarrow i-1$

 Inv: $\{x[1..j] \text{ e sortat, } aux \leq x[j+1] \leq \dots \leq x[i]\}$

 WHILE $(j > 1) \text{ AND } (aux < x[j])$ DO

$x[j+1] \leftarrow x[j]$

$\{x[1..j-1] \text{ e sortat, } aux < x[j] = x[j+1] \leq \dots \leq x[i]\}$

$j \leftarrow j-1$

$\{x[1..j] \text{ e sortat, } aux < x[j+1] = x[j+2] \leq \dots \leq x[i]\}$

 ENDWHILE

 Este satisfacuta fie $\{aux \geq x[j] \text{ si } x[1] \leq \dots \leq x[j] \leq aux < x[j+1] = x[j+2] \leq \dots \leq x[i]\}$

 fie $\{(j=0) \text{ si } aux < x[1] = x[2] \leq \dots \leq x[i]\}$

$x[j+1] \leftarrow aux$ $\{x[1] \leq x[2] \leq \dots \leq x[j+1] \leq x[j+2] \leq \dots \leq x[i]\}$ ($x[1..i]$ e sortat)

$i \leftarrow i+1$ $\{x[1..i-1] \text{ e sortat}\}$

ENDWHILE

RETURN $x[1..n]$

Sortare prin insertie – verificare corectitudine

Deci am obținut că...

Invariant pentru ciclul exterior poate fi considerat: $\{x[1..i-1] \text{ sortat}\}$

Invariant pentru ciclul interior:

$\{x[1..j] \text{ e sortat, } aux \leq x[j+1] = x[j+2] \leq \dots \leq x[i]\}$

Ambele cicluri sunt finite:

funcție de terminare pentru ciclul exterior: $t(p) = n + 1 - i_p$

funcție de terminare pentru ciclul interior:

$$t(p) = \begin{cases} j_p & \text{dacă } aux < x[j_p] \\ 0 & \text{dacă } aux \geq x[j_p] \text{ sau } j_p = 0 \end{cases}$$

Sortare prin inserție – analiza eficienței

```
Insertie(x[1..n])
FOR i ← 2,n DO
  x[0] ← x[i]
  j ← i-1
  WHILE x[0]<x[j] DO
    x[j+1] ← x[j]
    j ← j-1
  ENDWHILE
  x[j+1] ← x[0]
ENDFOR
RETURN x[1..n]
```

Dimensiune problemă: n

Operații:

- **Comparații** ($T_C(n)$)
- **Mutări ale elementelor** ($T_M(n)$)

Cel mai favorabil caz:

$$x[1] \leq x[2] \leq \dots \leq x[n]$$

Cel mai defavorabil caz:

$$x[1] > x[2] > \dots > x[n]$$

Pentru fiecare i ($2 \leq i \leq n$):

$$1 \leq T_C(n) \leq i$$

Pentru toate valorile lui i :

$$(n-1) \leq T_C(n) \leq n(n+1)/2 - 1$$

Sortare prin inserție – analiza eficienței

```
Insertie(x[1..n])
FOR i ← 2,n DO
  x[0] ← x[i]
  j ← i-1
  WHILE x[0]<x[j] DO
    x[j+1] ← x[j]
    j ← j-1
  ENDWHILE
  x[j+1] ← x[0]
ENDFOR
RETURN x[1..n]
```

Dimensiune problemă: n

Operații:

- **Comparații** ($T_C(n)$)
- **Mutări ale elementelor**($T_M(n)$)

Pentru fiecare i ($2 \leq i \leq n$):

$$0+2 \leq T_M(n) \leq (i-1)+2=i+1$$

Pentru toate valorile lui i :

$$2(n-1) \leq T_M(n) \leq (n+1)(n+2)/2-3$$

Sortare prin inserție – analiza eficienței

Comparații:

$$(n-1) \leq T_C(n) \leq (n+2)(n-1)/2$$

Mutări:

$$2(n-1) \leq T_M(n) \leq (n+1)(n+2)/2 - 3$$

Total:

$$3(n-1) \leq T(n) \leq n^2 + 2n - 3$$

Clase de complexitate (eficiența):

$$T(n) \in \Omega(n)$$

$$T(n) \in O(n^2)$$

Sortare prin inserție – stabilitate

8 5 9 8' 1 \longrightarrow 5 8 9 8' 1

5 8 9 8' 1 \longrightarrow 5 8 9 8' 1

5 8 9 8' 1 \longrightarrow 5 8 8' 9 1

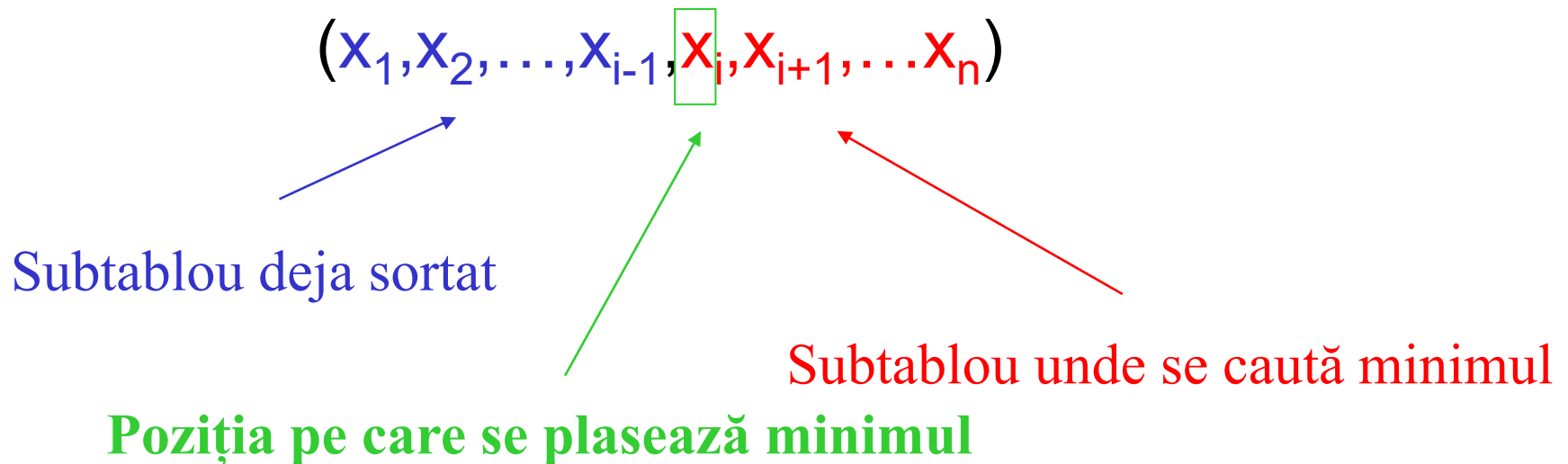
5 8 8' 9 1 \longrightarrow 1 5 8 8' 9

Sortarea prin inserție este stabilă

Sortare prin selecție

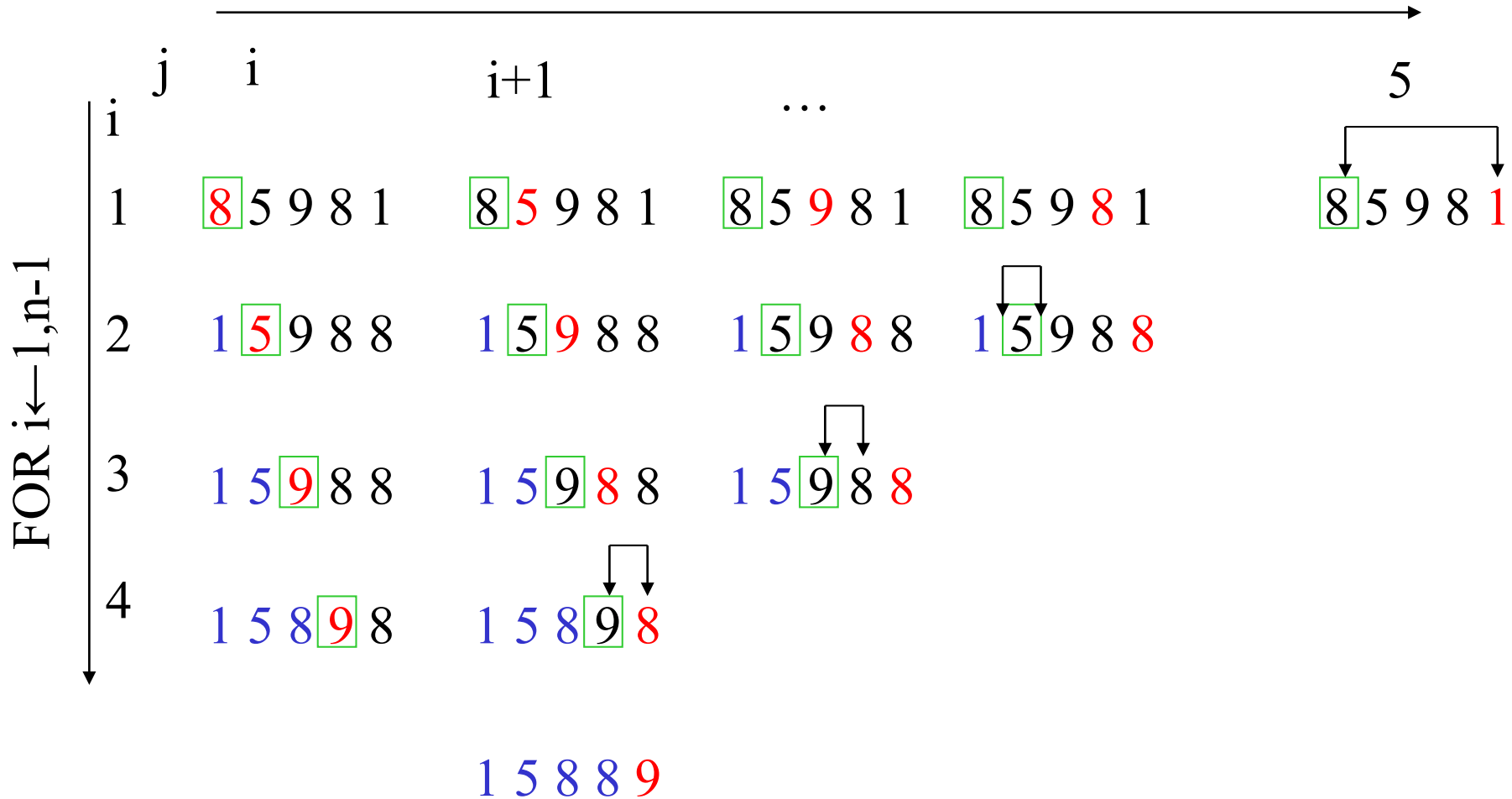
Ideea de bază:

Pentru fiecare poziție , i (începând cu prima) se caută minimumul din subtabloul $x[i..n]$ și acesta se interschimbă cu elementul de pe poziția i .



Sortare prin selecție

FOR j ← i+1, n



Sortare prin selecție

Structura generală

```
FOR i ← 1, n-1 DO
    <se caută minimul lui
    x[i..n] și se interschimbă
    cu x[i]>
ENDFOR
```

Algoritm

```
Selectie (x[1..n])
FOR i ← 1, n-1 DO
    k ← i;
    FOR j ← i+1, n DO
        IF x[k] > x[j] THEN k ← j ENDIF
    ENDFOR
    IF k <> i
        THEN x[i] ↔ x[k]
    ENDIF
ENDFOR
RETURN x[1..n]
```

Sortare prin selectie – verificare corectitudine

Algoritm

Selection ($x[1..n]$)

$i \leftarrow 1$

$\{x[1..i-1] \text{ sortat si } x[i-1] \leq x[j], j=i..n\}$

WHILE $i \leq n-1$ DO

$k \leftarrow i$

$j \leftarrow i+1$

$\{x[k] \leq x[r] \text{ pentru orice } r=i..j-1\}$

WHILE $j \leq n$ DO

IF $x[k] > x[j]$ THEN $k \leftarrow j$ ENDIF

$j \leftarrow j+1$

ENDWHILE

$\{x[k] \leq x[r] \text{ pentru orice } r=i..n\}$

IF $k \neq i$ THEN $x[i] \leftrightarrow x[k]$ $\{x[1..i] \text{ sortat si } x[i] \leq x[j], j=i..n\}$

$i \leftarrow i+1$

ENDWHILE

$\{x[1..i-1] \text{ sortat si } x[i-1] \leq x[j], j=i..n\}$

RETURN $x[1..n]$

Sortare prin selecție – verificare corectitudine

Deci am obtinut că...

Invariant pentru ciclul exterior poate fi considerat:

$\{x[1..i-1] \text{ sortat și } x[i-1] \leq x[j], j=i..n\}$

Invariant pentru ciclul interior poate fi considerat:

$\{x[k] \leq x[r] \text{ pentru orice } r=i..j-1\}$

Ambele cicluri sunt finite:

funcție de terminare pentru ciclul exterior: $t(p)=n-i_p$

funcție de terminare pentru ciclul interior: $t(p)=n+1-j_p$

Sortare prin selecție – analiza eficienței

```
Selectie (x[1..n])
FOR i ← 1,n-1 DO
  k ← i
  FOR j ← i+1,n DO
    IF x[k]>x[j] THEN k ← j
  ENDFOR
  IF k<>i THEN x[i]<->x[k]
ENDFOR
RETURN x[1..n]
```

Dimensiune problemă: n

Operații:

- **Comparații** ($T_C(n)$)
- **Mutări ale elementelor**($T_M(n)$)

Pentru fiecare i ($1 \leq i \leq n-1$):

$$T_C(n,i) = n-i$$

Pentru toate valorile lui i :

$$T_C(n) = n(n-1)/2$$

Sortare prin selecție – analiza eficienței

```
Selectie (x[1..n])
FOR i ← 1,n-1 DO
  k ← i
  FOR j ← i+1,n DO
    IF x[k]>x[j] THEN k ← j
  ENDFOR
  IF k<>i THEN x[i]<->x[k]
ENDFOR
RETURN x[1..n]
```

Dimensiune problemă: n

Operații:

- **Comparații ($T_C(n)$)**
- **Mutări ale elementelor ($T_M(n)$)**

Pentru fiecare i ($2 \leq i \leq n$):

$$0 \leq T_M(n,i) \leq 3$$

Obs: o interschimbare (\leftrightarrow) necesită 3 asignări

Pentru toate valorile lui i :

$$0 \leq T_M(n) \leq 3(n-1)$$

Sortare prin selecție – analiza eficienței

Comparații:

$$T_C(n) = n(n-1)/2$$

Mutări:

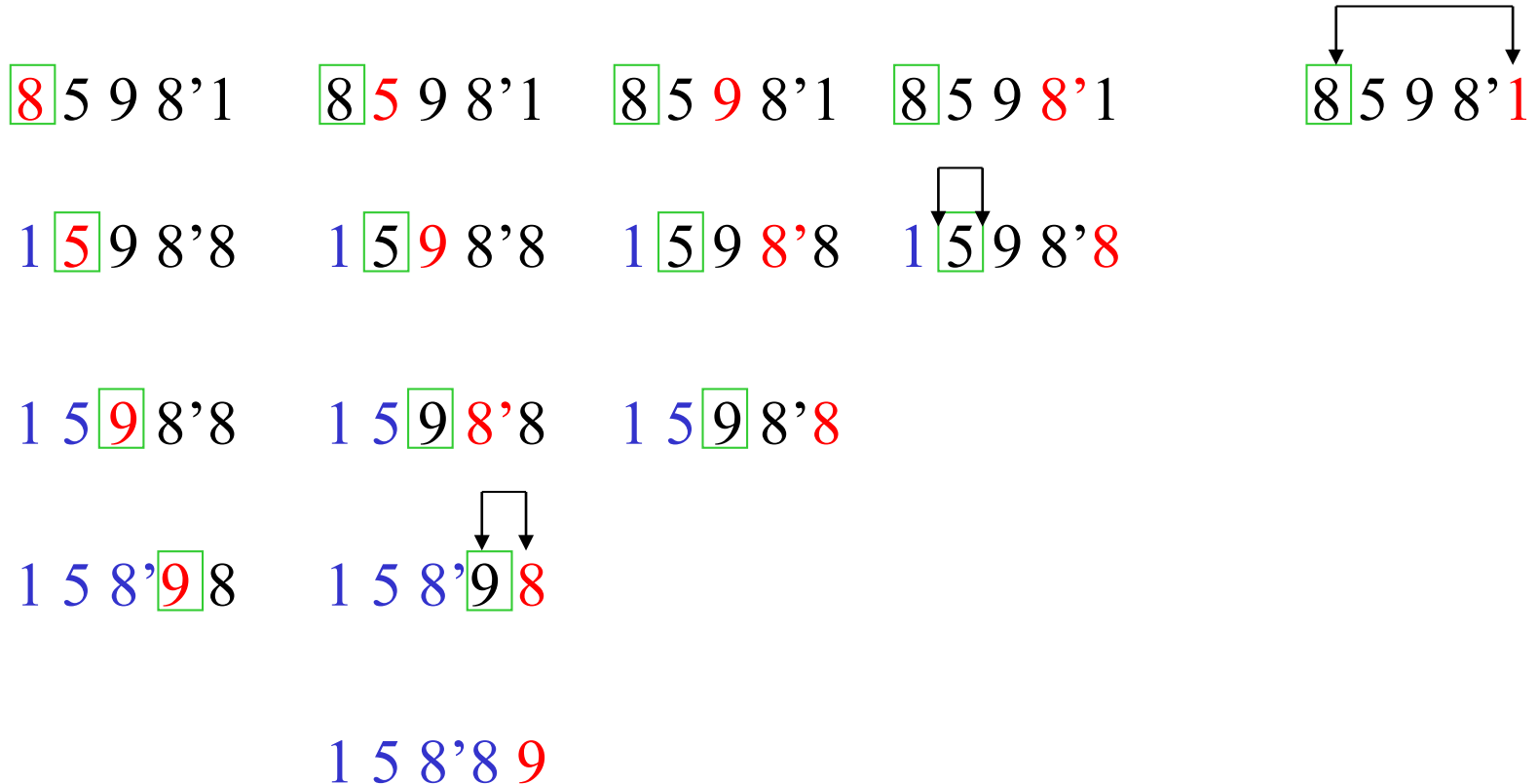
$$0 \leq T_M(n) \leq 3(n-1)$$

Total:

$$n(n-1)/2 \leq T(n) \leq n(n-1)/2 + 3(n-1)$$

Clasa de eficiență (complexitate): $T(n) \in \Theta(n^2)$

Sortare prin selecție - stabilitate




Sortarea prin selecție nu e stabilă

Sortare prin interschimbarea elementelor vecine

Idee de bază:

Tabloul este parcurs de la stânga spre dreapta și elementele adiacente sunt comparate. Dacă nu sunt în ordinea dorită atunci se interschimbă. Procesul este repetat până când tabloul ajunge să fie ordonat

$(x_1, x_2, \dots, x_{m-1}, x_m, x_{m+1}, \dots, x_n)$



Subtablou deja ordonat

Sortare prin interschimbarea elementelor vecine

FOR j ← 1, i-1

		FOR j ← 1, i-1				
		1	2	...	4	
FOR i ← n, 2, -1	i					
	5	8 5 9 8 1	5 8 9 8 1	5 8 9 8 1	5 8 8 9 1	5 8 8 1 9
	4	5 8 8 1 9	5 8 8 1 9	5 8 8 1 9	5 8 1 8 9	
	3	5 8 1 8 9	5 8 1 8 9	5 1 8 8 9		
	2	5 1 8 8 9	1 5 8 9 8			

Sortare prin interschimbarea elementelor vecine

Structura generală

```
FOR i ← n,2,-1 DO  
< se parcurge x[1..i-1], se  
  compară elementele  
  adiacente și se interschimbă  
  dacă este cazul >  
ENDFOR
```

Algoritm

```
Bubblesort(x[1..n])  
FOR i ← n,2,-1 DO  
  FOR j ← 1,i-1 DO  
    IF x[j]>x[j+1]  
      THEN x[j]←>x[j+1]  
    ENDIF  
  ENDFOR  
ENDFOR  
RETURN x[1..n]
```


Bubble sort – analiza corectitudinii

Bubblesort($x[1..n]$)

$i \leftarrow n$ $\{x[i+1..n]$ este sortat si $x[i+1] \geq x[j], j=1..i\}$

WHILE $i \geq 2$ DO

$j \leftarrow 1$ $\{x[j] \geq x[k], k=1..j-1\}$

 WHILE $j \leq i-1$ DO

 IF $x[j] > x[j+1]$ THEN $x[j] \leftrightarrow x[j+1]$ $\{x[j+1] \geq x[k], k=1..j\}$

$j \leftarrow j+1$ $\{x[j] \geq x[k], k=1..j-1\}$

 ENDWHILE $\{x[i-1] \geq x[j], j=1..i-1\}$

$\{x[i..n]$ este sortat si $x[i] \geq x[j], j=1..i-1\}$

$i \leftarrow i-1$

ENDWHILE $\{x[i+1..n]$ este sortat si $x[i+1] \geq x[j], j=1..i\}$

RETURN $x[1..n]$

Bubble sort – analiza corectitudinii

Deci am obținut că ...

Invariant pentru ciclul exterior poate fi:

$\{x[i+1..n] \text{ este sortat și } x[i+1] \geq x[j], j=1..i\}$

Un invariant pentru ciclul interior poate fi:

$\{x[j] \geq x[k], k=1..j-1\}$

Ambele cicluri sunt finite:

funcție de terminare pentru ciclul exterior: $t(p)=i_p-1$

funcție de terminare pentru ciclul interior: $t(p)=i_p-j_p$

Bubble sort – analiza eficienței

```
Bubblesort(x[1..n])
FOR i ← n,2,-1 DO
  FOR j ← 1,i-1 DO
    IF x[j]>x[j+1]
      THEN x[j]<->x[j+1]
    ENDIF
  ENDFOR
ENDFOR
RETURN x[1..n]
```

Dimensiune problemă: n

Operații:

Comparații ($T_C(n)$)

Mutări ale elementelor ($T_M(n)$)

Pentru fiecare i ($1 \leq i \leq n-1$):

$$T_C(n,i) = i-1$$

Pentru toate valorile lui i :

$$T_C(n) = n(n-1)/2$$

Bubble sort – analiza eficienței

```
Bubblesort(x[1..n])
FOR i ← n,2,-1 DO
  FOR j ← 1,i-1 DO
    IF x[j]>x[j+1]
      THEN x[j]<->x[j+1]
    ENDIF
  ENDFOR
ENDFOR
RETURN x[1..n]
```

Dimensiune problemă: n

Operatii:

Comparații ($T_C(n)$)

Mutări ale elementelor ($T_M(n)$)

Pentru fiecare i ($2 \leq i \leq n$):

$$0 \leq T_M(n,i) \leq 3(i-1)$$

Pentru toate valorile lui i :

$$0 \leq T_M(n) \leq 3n(n-1)/2$$

Bubble sort – analiza eficienței

Comparații:

$$T_C(n) = n(n-1)/2$$

Mutări:

$$0 \leq T_M(n) \leq 3n(n-1)/2$$

Total:

$$n(n-1)/2 \leq T(n) \leq 2n(n-1)$$

Clasa de eficiență (complexitate): $T(n) \in \Theta(n^2)$

Obs. Aceasta variantă de implementare este cea mai puțin eficientă.

Variantele mai bune evită execuția de $(n-1)$ ori a ciclului exterior, oprind prelucrarea când tabloul este sortat deja

Bubble sort – alte variante

Idee: se reia parcurgerea tabloului doar dacă a fost necesară cel puțin o interschimbare la parcurgerea anterioară

```
Bubblesort(x[1..n])
sw ← TRUE
WHILE sw=TRUE DO
  sw ← FALSE
  FOR j ← 1, n-1 DO
    IF x[j]>x[j+1]
      THEN x[j]↔x[j+1]
           sw ← TRUE
    ENDIF
  ENDFOR
ENDWHILE
RETURN x[1..n]
```

Idee: se parcurge tabloul doar până la poziția ultimei interschimbări efectuate la parcurgerea anterioară

```
Bubblesort(x[1..n])
t ← n
WHILE t>1 DO
  m ← t; t ← 0
  FOR j ← 1, m-1 DO
    IF x[j]>x[j+1]
      THEN x[j]↔x[j+1]; t ← j
    ENDIF
  ENDFOR
ENDWHILE
RETURN x[1..n]
```

$$n-1 \leq T_c(n) \leq n(n-1)/2$$

$$n-1 \leq T_c(n) \leq n(n-1)$$

Bubble sort - stabilitate

8 5 9 8' 1 5 8 9 8' 1 5 8 9 8' 1 5 8 8' 9 1 5 8 8' 1 9

5 8 8' 1 9 5 8 8' 1 9 5 8 8' 1 9 5 8 1 8' 9

5 8 1 8' 9 5 8 1 8' 9 5 1 8 8' 9

5 1 8 8' 9 1 5 8 9 8'

Bubble sort este stabilă
(cu conditia sa se foloseasca inegalitate
stricta: $x[j] > x[j+1]$)

Link-uri utile...

<http://www.sorting-algorithms.com/>

<http://www.softpanorama.org/Algorithms/sorting.shtml>

<http://www.brian-borowski.com/Software/Sorting/>

http://www.youtube.com/watch?v=INHF_5RIxTE 😊

<http://boingboing.net/2010/08/19/what-does-a-bubble-s.html>



Sumar

- Tehnicile clasice de sortare (inserție, selecție, interschimbarea elementelor vecine) au complexitate pătratică - practice doar pentru tablouri cu număr mic de elemente (de ordinul sutelor).
 - **Inserție**: avantajos dacă tabloul este “aproape sortat”; stabil
 - **Selecție**: util când e necesară sortare parțială (ex: se caută primele $k < n$ elemente în ordine crescătoare; instabil
 - **Bubblesort**: de evitat (în special prima variantă)
- Pentru $n=1000000$ și durata de o nanosecundă/operație ar necesita cca **17 minute**
- Dacă $T(n)$ ar fi din $O(n \log n)$ atunci pentru aceeași valoare a lui n ar fi suficiente **0.019 secunde**

Cursul următor va fi despre...

... tehnici de proiectare a algoritmilor

... tehnica reducerii

... recursivitate

Intrebare de final

Se consideră un tablou x cu 10000 de elemente (numere reale) și se pune problema determinării celui de al 10-lea element în ordine descrescătoare.

De la care dintre algoritmi urmatori ati porni pentru a rezolva problema?

Obs: nu e necesar sa se ordoneze intregul tablou

Variante de răspuns:

- a) Sortare prin inserție
- b) Sortare prin selecție
- c) Sortare prin interschimbarea elementelor vecine
- d) Alt algoritm (precizati care)