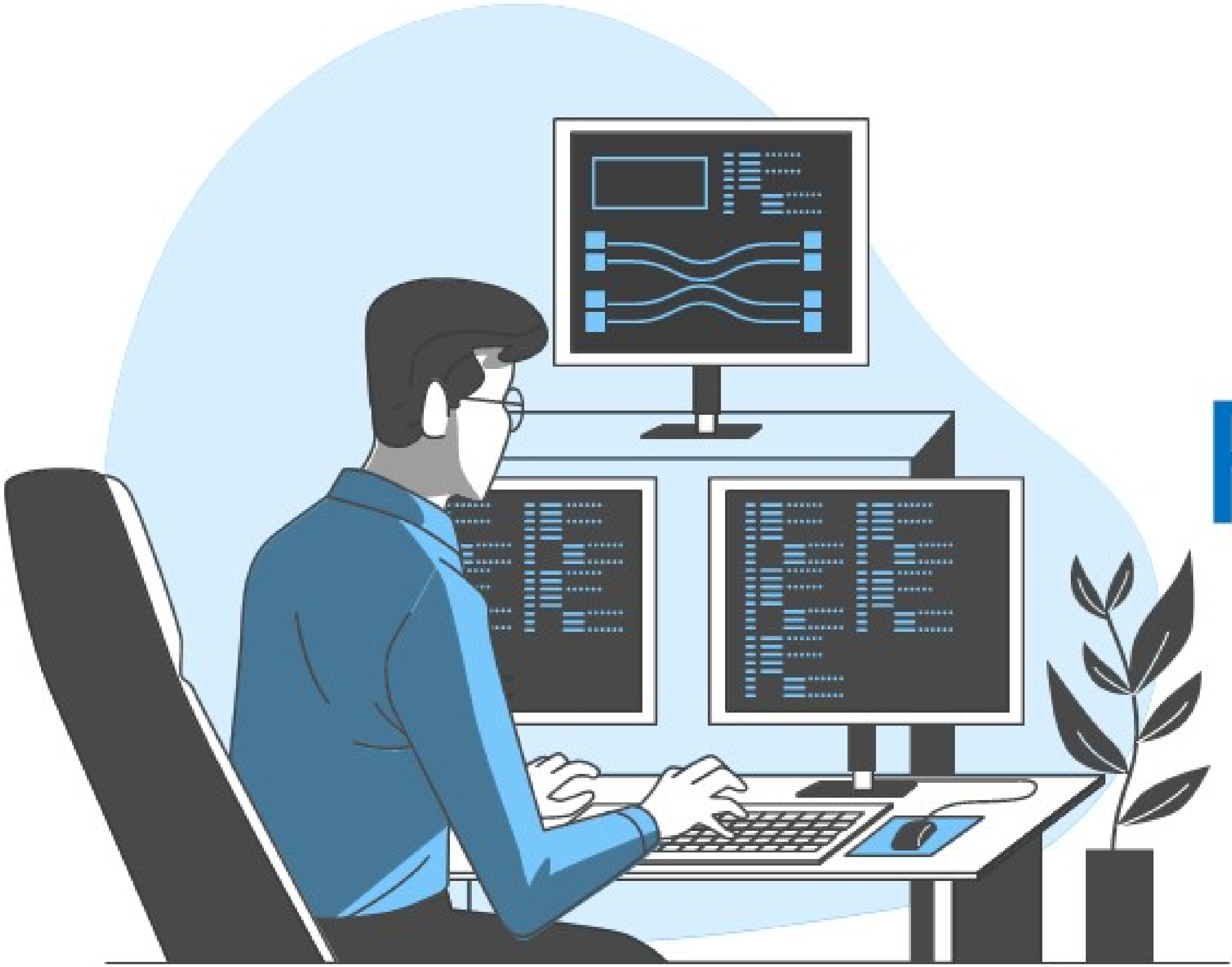


Algoritmi Fundamentali

Lector dr.
Dorin IORDACHE

Cursul nr. 8

sortare

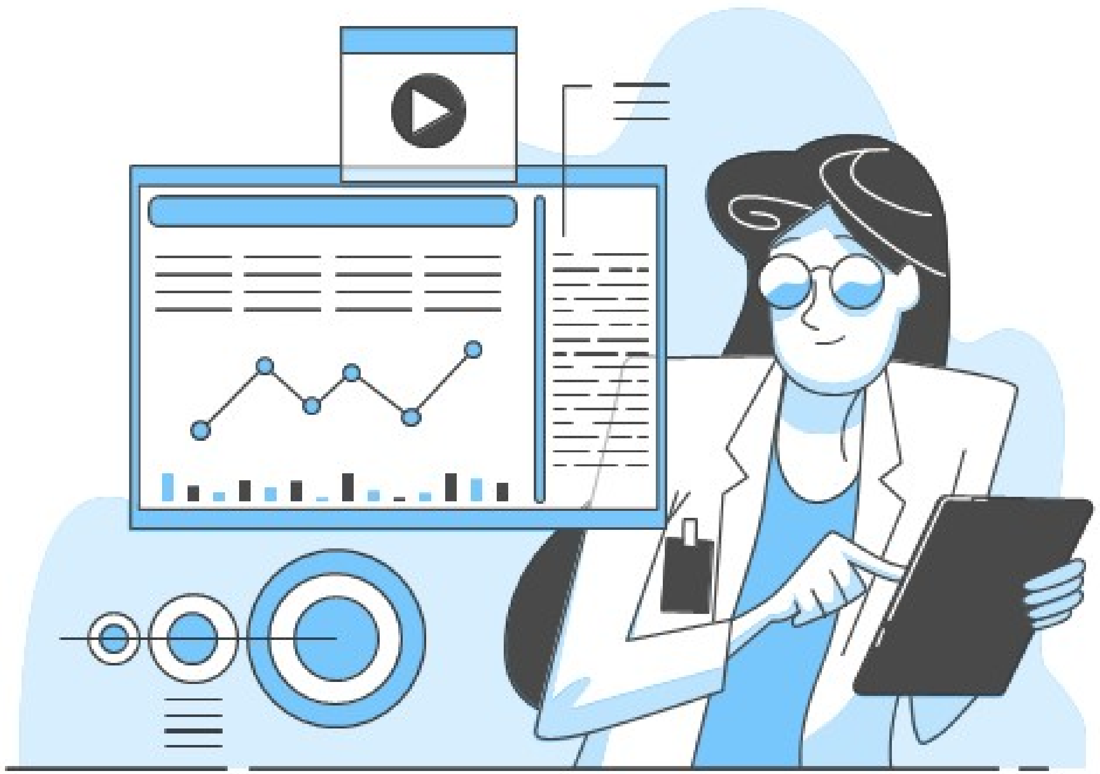
Agenda

01 Problema sortării

02 Sortarea prin insertie

03 Sortarea prin selecție

04 Sortarea prin interschimbare



01

Problema sortarii

Scop sortare

Tehnica generală de calcul:

- Preprocesează datele pentru a face operațiunile ulterioare (nu doar Abstract Data Type - ADT) mai rapid

Exemplu: Sortați datele astfel încât să puteți

- Găsiți cel mai mare al k element în timp constant pentru orice k
- Efectuați o căutare binară pentru a găsi elemente în timp logaritmic

Beneficiile sortării depind de:

- Cât de des se vor schimba datele
- Câte date sunt

Problema sortarii

Consideram:

- O secventa de “entitati” (numere, structuri continand mai multe informatii etc)

- Fiecare entitate poseda o caracteristica numita cheie de sortare. Valorile posibile ale cheii de sortare apartin unei multimi pe care exista o relatie totala de ordine

- Sortarea secventei = aranjarea elementelor astfel incat valorile cheii de sortare sa fie in ordine crescatoare (sau descrescatoare)

Problema sortarii

Exemple:

1. Secventa de numere: (5,8,3,1,6)
   Cheia de sortare este chiar elementul din secventa Rezultatul sortarii crescatoare este (1,3,5,6,8)

2. Secventa de entitati (numite inregistrari sau articole) ce contin doua campuri: nume si nota
   ((Paul,9), (Ioana, 10), (Victor,8), (Ana,9))

In acest caz exista doua criterii posibile de sortare:

nume si nota

Sortare crescatoare dupa nume:

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

Sortare descrescatoare dupa nota:

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

Problema sortarii

Ceva mai formal…

Ordonarea (crescatoare) a secventei $(x_1,x_2,\dots,x_n)$ este echivalenta cu gasirea unei permutari $(p(1),p(2),\dots,p(n))$ indicilor astfel incat:

$$cheie(x_{p(1)}) <= cheie(x_{p(2)}) <= \dots <= cheie(x_{p(n)})$$

In cele ce urmeaza vom considera cheia de sortare ca fiind reprezentata de intreg elementul (secventa este constituita din valori apartinand unei multimi pe care exista o relatie de ordine)

In aceasta ipoteza secventa este considerata ordonata daca satisface:

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

Problema sortarii

Alte ipoteze:

- Presupunem ca secventa este stocata sub forma unui tablou astfel ca este asigurat accesul rapid la oricare dintre elementele ei. Aceasta inseamna ca este vorba despre sortare interna

  Sortarea externa corespunde cazului in care nu se poate accesa simultan intreaga secventa (de exemplu secventa este foarte mare astfel ca nu poate fi incarcata in intregime in memoria interna a calculatorului) ceea ce necesita metode specifice de sortare

- Vom analiza doar metode care transforma tabloul curent fara a construi un alt tablou (spatiul aditional de memore are dimensiunea de ordin O(1)).

Proprietati ale metodelor de sortare

1. Simplitate

   Metoda de sortare ar trebui sa fie simplu de inteles si de implementa

2. Eficienta

   Metoda de sortare ar trebui sa necesite un volum rezonabil de resurse (timp de executie)

3. Naturalete

   O metoda de sortare este considerata naturala daca numarul de operatii necesare este proportional cu gradul de “dezordine” al secventei initial (care poate fi masurat prin numarul de inversiuni din permutarea corespunzatoare secventei sortate)

4. Stabilitate

   O metoda de sortare este stabila daca conserva ordinea relativa a elementelor avand aceeasi valoare a cheii de sortare

Stabilitate

Exemplu:

Configuratia initiala:

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

Sortare stabila:

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

Sortare instabila:

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

Metode elementare de sortare

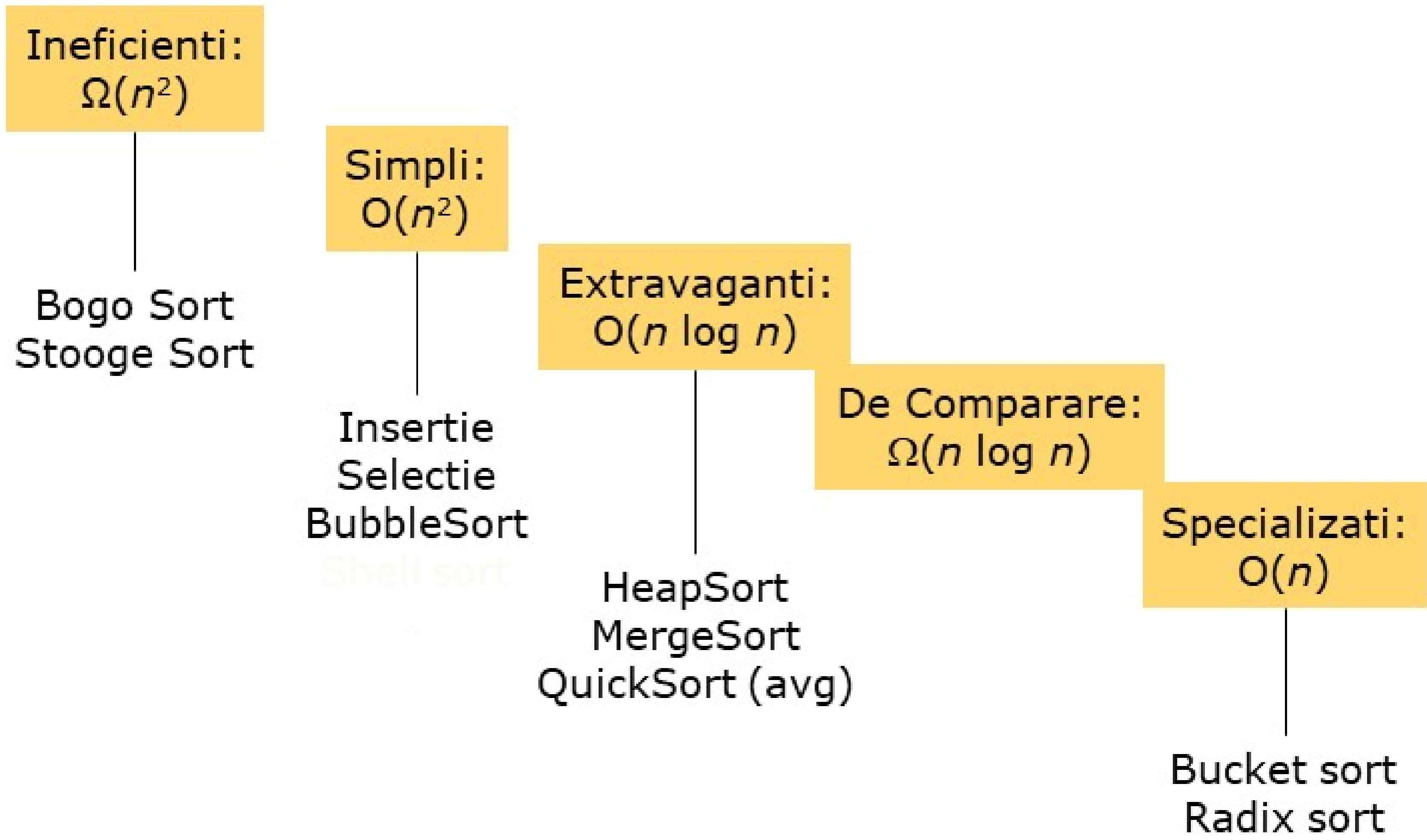
Sunt simple, intuitive dar nu foarte eficiente…

Reprezinta, totusi, puncte de pornire pentru metodele mai avansata.

Exemple:

- Sortare prin insertie
- Sortare prin selectie
- Sortare prin interschimbare

Sortare: algoritmi



Sortare: algoritmi

Studiu individual:
cum sa nu folositi alg. de sortare

Ineficienti: $\Omega(n^2)$
- Bogo Sort
- Stooge Sort

Simpli: $O(n^2)$
- Insertie
- Selectie
- BubbleSort

Extravaganti: $O(n \log n)$
- HeapSort
- MergeSort
- QuickSort (avg)

De Comparare: $\Omega(n \log n)$

Specializati: $O(n)$
- Bucket sort
- Radix sort

Sortare: algoritmi

Ineficienti:
$\Omega(n^2)$

- Bogo Sort
- Stooge Sort

Simpli:
$O(n^2)$

- Insertie
- Selectie
- BubbleSort
- Shell sort

Extravaganti:
$O(n \log n)$

- HeapSort
- MergeSort
- QuickSort (avg)

De Comparare:
$\Omega(n \log n)$

Specializati:
$O(n)$

- Bucket sort
- Radix sort

02

Sortarea prin insertie

Sortare prin insertie

Idee de baza:

Fiecare element al tabloului, incepand cu al doilea, este inserat in subtabloul care il preceda astfel incat acesta sa ramana ordonat:

$$(x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots x_n)$$

Subtablou deja ordonat
(subtablou destinatie)

Element de inserat

Sortarea prin insertie

Problema:
Sortarea crescătoare a celor n elemente v[0], v[1], . . . , v[n-1] ale unui vector v.

Algoritm:
1. Considerăm subşirul v[0], care este deja ordonat.
2. Pentru fiecare i de la 1 la n - 1, subşirul v[0]; v[1]; ... ; v[i - 1] este sortat şi se determină poziţia corespunzătoare a elementului v[i]: acel k ∈ { 0; 1; ... ; i - 1}, cu proprietatea că v[k - 1] ≤ v[i] < v[k]; elementele v[k]; ... ; v[i - 1] se mută cu o poziţie la dreapta, iar elementul curent, v[i], se inserează pe poziţia k.

Sortare prin insertie – algoritm

Structura generala

```
FOR i ← 2,n DO
<insereaza x[i] in subtabloul
    x[1..i-1] astfel incat x[1..i]
    sa 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 insertie – la stanga

WHILE (j>=1) AND (aux<x[j])

FOR i←2,n

i	j		i-1	i-2	i-3	i-4
2	8 5 9 8’ 1	aux=5	8 8 9 8’ 1	5 8 9 8’ 1		
3	5 8 9 8’ 1	aux=9	5 8 9 8’ 1			
4	5 8 9 8’ 1	aux=8’	5 8 9 9 1	5 8 8’ 9 1		
5	5 8 8’ 9 1	aux=1	5 8 8’ 9 9	5 8 8 8’ 9	5 8 8 8’ 9	5 5 8 8’ 9
						1 5 8 8’ 9

Sortare prin insertie – varianta

```
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 zona de manevra):

```
Insertion(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←2                          {x[1..i-1] e sortat}
WHILE i<=n
   aux ← x[i]
    j ← i-1                  {x[1..j] e sortat,  aux<=x[j+1]<=..<=x[i]}
    WHILE (j>=1) AND (aux<x[j]) DO
        x[j+1] ← x[j]        {x[1..j-1] e sortat, aux<x[j]=x[j+1]<= … <=x[i]}
        j ← j-1              {x[1..j] e sortat, aux<x[j+1]=x[j+2]<= … <=x[i]}
   Este satisfacuta fie {aux>=x[j]  si
    x[1]<=…x[j]<=aux<x[j+1]=x[j+2]<=…<=x[i]}
   f ie {(j=0)  si  aux<x[1]=x[2]<=…<=x[i]}
    ENDWHILE
    x[j+1] ← aux   {x[1]<=x[2]<=…x[j+1]<=x[j+2]<=…<=x[i]} (x[1..i] e sortat)
    i ← i+1        {x[1..i-1]  e sortat}
ENDWHILE
 RETURN  x[1..n]
```

Sortare prin insertie – verificare corectitudine

Deci am obtinut ca…

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

Invariant pentru ciclul interior:

{x[1..j] e sortat, aux<=x[j+1]=x[j+2]<=..<=x[i]}

Ambele cicluri sunt finite:

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

functie de terminare pentru ciclul interior:

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

Sortare prin insertie – analiza eficientei

```
for i ← 2, n do
    aux ← v[i]
    j ← i − 1
    while aux < v[j] AND j >= 0 do
        v[j + 1] ← v[j]
        j ← j − 1
    end while
    v[j + 1] ← aux
end for
```

Dimensiune problema: n

Operatii:

- Comparitii ($T_C(n)$)
- Mutari ale elementelor ($T_M(n)$)

Pentru fiecare i ($2 <= i <= n$):

$$1 <= T_C(n) <= i$$

Pentru toate valorile lui i:

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

Sortare prin insertie – analiza eficientei

```
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 problema: n

Operatii:

- Comparatii ($T_C(n)$)
- Mutari ale elementelor($T_M(n)$)

Pentru fiecare i (2 <=i <= n):

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

Pentru toate valorile lui i:

$$2 (n-1) <= T_M(n) <= (n+1)(n+2)/2-3$$

Sortare prin insertie – analiza eficientei

Comparatii:

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

Mutari:

$$2(n-1) <= T_M(n) <= (n+1)(n+2)/2-3$$

Total:

$$3(n-1) <= T(n) <= n^2+2n-3$$

Clase de complexitate (eficienta):

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

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

Sortare prin insertie – stabilitate

8 5 9 8’ 1 ⟶ 5 8 9 8’ 1

5 8 9 8’ 1 ⟶ 5 8 9 8’ 1

5 8 9 8’ 1 ⟶ 5 8 8’ 9 1

5 8 8’ 9 1 ⟶ 1 5 8 8’ 9

Sortarea prin insertie este stabila

03

Sortarea prin selectie

Sortare prin selectie

Ideea de baza:

Pentru fiecare pozitie , i (incepand cu prima) se cauta minimul din subtabloul x[i..n] si acesta se interschimba cu elementul de pe pozitia i.

$$(x_1, x_2, \ldots, x_{i-1}, \boxed{x_i}, x_{i+1}, \ldots x_n)$$

Subtablou deja sortat

Subtablou unde se cauta minimul

Pozitia pe care se plaseaza minimul

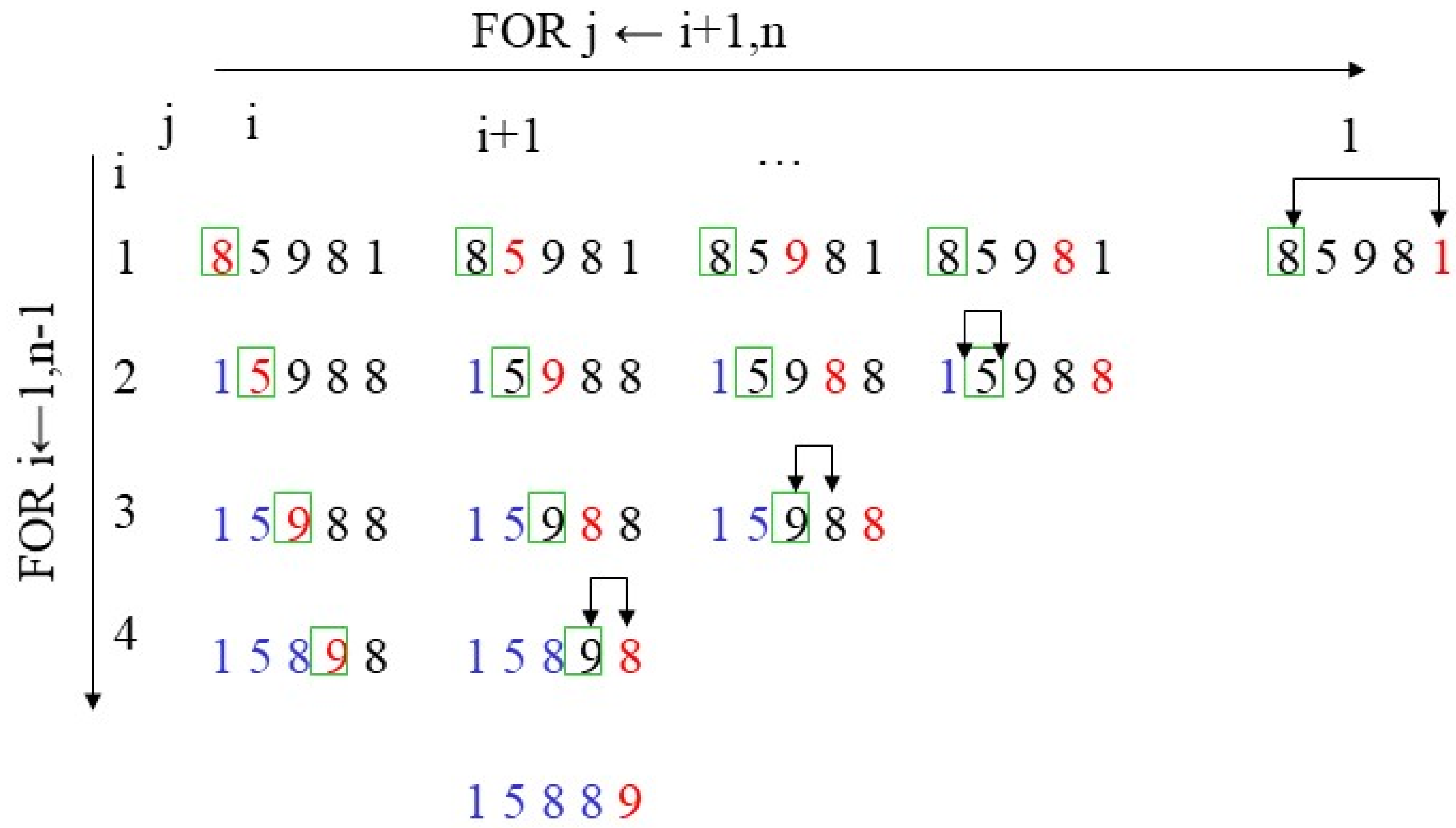
Sortarea prin selectie

Problema:
Sortarea crescătoare a celor n elemente v[0], v[1], . . . , v[n-1] ale unui vector v, pentru fiecare i de la 0 la n - 2.

Algoritm:

1. Se determină valoarea minimă din şirul v[i]; ... ; v[n-1], rămas neordonat.
2. Valoarea minimului se interschimbă cu elementul de la poziţia i.

Sortare prin selectie



Sortare prin selectie

Structura generala

```
FOR i ← 1,n-1 DO
        <se cauta minimul lui
        x[i..n] si se interschimba
        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 ← 1                              {x[1..i-1] sortat si x[i-1]<=x[j], j=i..n}
WHILE i<=n-1 DO
   k ← i
   j ← i+1                  {x[k]<=x[r]  for all r=i+1..j-1}
   WHILE j<=n  DO
     IF x[k]>x[j] THEN  k ← j ENDIF
     j ← j+1
   ENDWHILE                         {x[k]<=x[r]  for all r=i+1..n}
   IF k<>i THEN  x[i]<->x[k]  {x[1..i] sortat si x[i]<=x[j], j=i+1..n}
   i ← i+1
ENDWHILE                        {x[1..i-1] sortat si x[i-1]<=x[j], j=i..n}
RETURN  x[1..n]
```

Sortare prin selectie – verificare corectitudine

Deci am obtinut ca…

Invariant pentru ciclul exterior poate fi considerat:

{x[1..i-1] sortat si x[i-1]<=x[j], j=i..n}

Invariant pentru ciclul interior poate fi considerat:

{x[k]<=x[r] for all r=i+1..j-1}

Ambele cicluri sunt finite:

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

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

Sortare prin selectie – analiza eficientei

```
for i ←1, n − 1 do
    poz_min← i
    for j ←i + 1, n do
        if v[j] < v[poz_min] then
            poz_min← j
        end if
    end for
    if poz_min ≠ i then
        v[poz_min] ↔ v[i]
    end if
end for
```

Dimensiune problema: n

Operatii:

- Comparatii ($T_C(n)$)
- Mutari ale elementelor($T_M(n)$)

Pentru fiecare i (1 <=i <= n-1):

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

Pentru toate valorile lui i:

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

Sortare prin selectie – analiza eficientei

Comparatii:

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

Mutari:

$0 <= T_M(n) <= 3(n-1)$

Total:

$n(n-1)/2 <= T(n) <= n(n-1)/2+3(n-1)$

Clasa de eficienta (complexitate): $T(n) \in O(n^2)$

Sortare prin selectie - stabilitate

8 5 9 8’1   8 5 9 8’1   8 5 9 8’1   8 5 9 8’1   8 5 9 8’1

1 5 9 8’8   1 5 9 8’8   1 5 9 8’8   1 5 9 8’8

1 5 9 8’8   1 5 9 8’8   1 5 9 8’8

1 5 8’9 8   1 5 8’9 8

1 5 8’8 9

Sortarea prin selectie este stabila

04

Sortarea prin interschimbare

Sortare prin interschimbarea elementelor vecine

Idee de baza:

Tabloul este parcurs de la stanga spre dreapta si elementele adiacente sunt comparate. Daca nu sunt in ordinea dorita atunci se interschimba. Procesul este repetat pana cand tabloul e ordonat

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

Subtablou deja ordonat

Sortare prin interschimbarea elementelor vecine

FOR j ← 1,i-1

FOR i ← n,2,-1

i \ j	1	2	…	4	
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 8’ 9			

Sortare prin interschimbarea elementelor vecine

Structura generala

FOR i ← n,2,-1 DO

< se parcurge x[1..i-1], se compara elementele adiacente si se interschimba daca 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 corectutidine

```
Bubblesort(x[1..n])
i ← n        {x[i+1..n] este sortat si x[i+1]>=x[j], j=1..i}
WHILE i>=2 DO
  j ← 1                                   {x[j]>=x[k], k=1..j-1}
  WHILE j<=i-1 DO
      IF x[j]>x[j+1]  THEN  x[j]<->x[j+1]   {x[j+1]>=x[k], k=1..j}
      j ← j+1                              {x[j]>=x[k], k=1..j-1}
  ENDWHILE                                 {x[i-1]>=x[j], j=1..i-1}
         {x[i..n] este sortat si x[i]>=x[j], j=1..i-1}
  i ← i-1
ENDWHILE           {x[i+1..n] este sortat si x[i+1]>=x[j], j=1..i}
RETURN  x[1..n]
```

Bubble sort – analiza corectutidine

Deci am obtinut ca …

Invariant pentru ciclul exterior poate fi:

{x[i+1..n] este sortat si x[i+1]>=x[j], j=1..i}

Un invariant pentru ciclul interior poate fi:

{x[j]>=x[k], k=1..j-1}

Ambele cicluri sunt finite:

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

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

Bubble sort – analiza eficientei

```
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 problema: n

Operatii:

Comparatii ($T_C(n)$)

Mutari ale elementelor ($T_M(n)$)

Pentru fiecare i (1 <=i <= n-1):

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

Pentru toate valorile lui i:

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

Bubble sort – analiza eficientei

```
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 problema: n

Operatii:

Comparatii ($T_C(n)$)

Mutari ale elementelor ($T_M(n)$)

Pentru fiecare i ($2 <= i <= n$):

$$0 <= T_M(n,i) <= 3(i-1)$$

Pentru toate valorile lui i:

$$0 <= T_M(n) <= 3n(n-1)/2$$

Bubble sort – analiza eficientei

Comparatii:

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

Mutari:

$$0 <= T_M(n) <= 3n(n-1)/2$$

Total:

$$n(n-1)/2 <= T(n) <= 2n(n-1)$$

Clasa de eficienta (complexitate): $T(n)$ ⍰⍰(n^2)

Obs. Aceasta varianta de implementare este cea mai putin eficienta. Variantele mai bune evita executia de (n-1) ori a ciclului exterior, oprind prelucrarea cand tabloul este sortat deja

Bubble sort – alte variante

Idee: se reia parcurgerea tabloului cat timp a fost necesara cel putin o interschimbare la parcurgerea anterioara

```
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]
```

$n-1 <= T_C(n) <= n(n-1)$

Idee: se parcurge tabloul doar pana la pozitia ultimei interschimbari efectuate la parcurgerea anterioara

```
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]
```

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

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 8’ 9

Bubble sort este stabila

Sortarea prin insertie

Idea: La pasul k, setare al k^{-lea} element in pozitia corecta printre primele kelemente

Cu alte cuvinte:

- Sortam primul element
- Apoi inserare al 2-lea element in ordine
- Apoi inserare al 3-lea element in ordine
- Apoi inserare al 4-lea element in ordine
- …

Continuare repetitie:

Cand indexul de bucla este i, primele i elemente sunt ordonated

Time?

Best: ______ Worst: ______ Average: ______

Sortarea prin insertie

Idea: La pasul k, setare al k-lea element in pozitia corecta printre primele kelemente

Cu alte cuvinte:
- Sortam primul element
- Apoi inserare al 2-lea element in ordine
- Apoi inserare al 3-lea element in ordine
- Apoi inserare al 4-lea element in ordine
- …

Continuare repetitie:
Cand indexul de bucla este i, primele i elemente sunt ordonated

Deja sau aproape sortat — *Sortat invers*

Timp: Best: $O(n)$ Worst: $O(n^2)$ Average: $O(n^2)$

Stabil si in acelasi spatiu

Sortarea prin selectie

Idea: La pasul k, gasirea celui mai mic element dintre elementele nesortate si plasarea lui in pozitia k

Cu alte cuvinte:

- Cautare cel mai mic element, punerea lui pe pozitia 1
- Cautare urmatorului cel mai mic element, punerea lui pe pozitia 2
- Cautare urmatorului cel mai mic element, punerea lui pe pozitia 3
- …

Continuare repetitie:

Cand indexul de bucla este i, primele i elemente sunt primele cele mai mici i elemente ordonate

Timp?

Best: ______ Worst: ______ Average: ______

Sortarea prin selectie

Idea: La pasul k, gasirea celui mai mic element dintre elementele nesortate si plasarea lui in pozitia k

Cu alte cuvinte:

- Cautare cel mai mic element, punerea lui pe pozitia 1
- Cautare urmatorului cel mai mic element, punerea lui pe pozitia 2
- Cautare urmatorului cel mai mic element, punerea lui pe pozitia 3
- …

Continuare repetitie:

Cand indexul de bucla este i, primele i elemente sunt primele cele mai mici i elemente ordonate

Time: Best: $O(n^2)$ Worst: $O(n^2)$ Average: $O(n^2)$

relatia de recurenta: $T(n) = n + T(N-1)$, $T(1) = 1$

Stabil si in acelasi spatiu

Link-uri utile…

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

Mulțumesc

pentru atenție!

dorin.iordache@365.univ-ovidius.ro