

Structuri de date și algoritmi

upT Universitatea
Politehnica
Timișoara

P-ța Victoriei nr. 2
RO 300006 - Timișoara
Tel: +4 0256 403000
Fax: +4 0256 403021
rector@rectorat.upt.ro
www.upt.ro

Domeniul de studii: Informatică/ Specializarea: Informatică

SDA – Cursul 6

Ș.I. dr.ing. Adriana ALBU

adriana.albu@upt.ro

<http://www.aut.upt.ro/~adrianaa>

3. Tehnici de sortare (partea a treia)

3.10 Sortări utilizând baze de numerație (radix-sort)

- Metodele de sortare prezentate consideră elementele de sortat (cheile) drept entități, utilizate ca atare în operațiile de comparație și interschimbare asociate procesului de sortare
- Metodele de sortare de tip „radix” iau în considerare **proprietățile digitale** ale numerelor, respectiv posibilitatea de a reprezenta numerele în diferite baze de numerație și de a utiliza în procesul de sortare componentele asociate reprezentării (biți – baza 2, cifre zecimale – baza 10, cifre hexazecimale etc.)
- Reprezentarea în baza m conduce la metode radix- m
- Procesul de sortare **impune operații de acces la componentele reprezentării**, individuale sau grupate funcție de metoda de sortare
 - pentru reprezentarea în baza 2 sunt necesare funcții de acces la un grup contiguu de biți
- Considerând componenta **cea mai semnificativă la stânga** și cea mai puțin semnificativă la dreapta, examinarea componentelor asociate reprezentării se poate face fie de la stânga la dreapta, fie de la dreapta la stânga

3.10.1 Sortare radix prin interschimbare (sortare după ranguri)

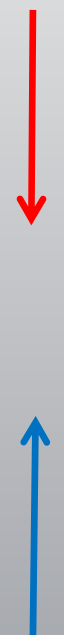
- Se bazează pe observația că **rezultatul comparației** a două elemente este dat în primă fază de valoarea componentei celei mai semnificative, respectiv de valoarea primei componente care diferă
- Elementele sunt procesate de la stânga la dreapta, începând cu componenta cea mai semnificativă
- Principiul metodei radix-2:
 - printr-un procedeu de partiționare asemănător lui quickSort (deci prin interschimbare) se obțin succesiv câte două partiții
 - prima conținând elemente care încep cu 0
 - cea de-a doua elemente care încep cu 1
 - se continuă procesul pentru următorul bit, până la epuizarea tuturor biților asociați reprezentării

3.10.1 Sortare radix prin interschimbare (sortare după ranguri)

```
radix_schimb(int s, int d, int b){ /*b - nr. de biti*/
    int t, i, j;
    if((d>s) && b>=0){
        i=s;
        j=d;
        b=b-1;
        do{
            while((biti(a[i].cheie,b,1)==0) && (i<j)) i=i+1;
            while((biti(a[j].cheie,b,1)==1) && (i<j)) j=j-1;
            t=a[i]; a[i]=a[j];a[j]=t;
        }while(i != j);
        if(biti(a[d].cheie,b,1)==0)
            j=j+1; /*refacere lungime partitie */
        radix_schimb(s, j-1, b-1);
        radix_schimb(j, d, b-1);
    }
}
```

3.10.1 Sortare radix prin interschimbare (sortare după ranguri)

➤ Exemplu: 7, 3, 15, 7, 10, 4, 1



| | | | | | | | | | |
|----|--------------|----|--------------|----|---------------|----|---------------|----|------|
| 7 | 0111 | 7 | 0111 | 7 | 0 1 11 | 1 | 0001 | 1 | 0001 |
| 3 | 0011 | 3 | 0011 | 3 | 0011 | 3 | 0011 | 3 | 0011 |
| 15 | 1 111 | 1 | 0001 | 1 | 0 0 01 | 7 | 01 1 1 | 4 | 0100 |
| 7 | 0111 | 7 | 0111 | 7 | 0111 | 7 | 0111 | 7 | 0111 |
| 10 | 1010 | 10 | 1 010 | 4 | 0100 | 4 | 01 0 0 | 7 | 0111 |
| 4 | 0100 | 4 | 0 100 | 10 | 1010 | 10 | 1010 | 10 | 1010 |
| 1 | 0 001 | 15 | 1111 | 15 | 1111 | 15 | 1111 | 15 | 1111 |

3.10.2 Sortarea radix directă (sortare prin distribuție)

- Se realizează sortarea după un bit, procesând biții de la dreapta la stânga, începând cu cel mai puțin semnificativ
- Sortarea după un bit trebuie să fie stabilă, astfel încât la sortarea bitului de indice i , toți biții cuprinși între i și lungimea elementului să fie deja sortați
- Se utilizează funcția `biti(a[i].cheie, k, 1)`; $k=0, 1, 2, \dots, b-1$
- Sunt necesare b treceri; b = numărul de biți asociați elementului
- Pentru sortarea după un bit se aplică **metoda sortării cu determinarea distribuțiilor**, fiind necesar un **tablou suplimentar** de dimensiune 2^1
- „sortare digitală” sau „sortare prin metoda buzunarelor”
- Numărul de treceri corespunde numărului de biți (ranguri), cu avantajul că sortarea în cadrul unei treceri se face independent de trecerea anterioară, ceea ce simplifică algoritmul

3.10.2 Sortarea radix directă (sortare prin distribuție)

➤ Exemplu: 7, 3, 15, 7, 10, 4, 1 m=1

| | | | | | | | | | |
|-------------|------|----|------|----|------|----|------|----|------|
| 7 | 0111 | 10 | 1010 | 4 | 0100 | 1 | 0001 | 1 | 0001 |
| 3 | 0011 | 4 | 0100 | 1 | 0001 | 10 | 1010 | 3 | 0011 |
| 15 | 1111 | 7 | 0111 | 10 | 1010 | 3 | 0011 | 4 | 0100 |
| 7 | 0111 | 3 | 0011 | 7 | 0111 | 4 | 0100 | 7 | 0111 |
| 10 | 1010 | 15 | 1111 | 3 | 0011 | 7 | 0111 | 7 | 0111 |
| 4 | 0100 | 7 | 0111 | 15 | 1111 | 15 | 1111 | 10 | 1010 |
| 1 | 0001 | 1 | 0001 | 7 | 0111 | 7 | 0111 | 15 | 1111 |
| Distribuții | | | | | | | | | |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | |
| 2 | 5 | 2 | 5 | 3 | 4 | 5 | 2 | | |

3.10.2 Sortarea radix directă (sortare prin distribuție)

- **Reducerea numărului de treceri** se obține prelucrând un grup de m biți în locul unuia singur $\Rightarrow b/m$ treceri și un tablou pentru contorizarea distribuțiilor de dimensiune 2^m ; b – multiplu de m
 - Ex. $m = 2$
- Sortarea radix directă depinde de evaluarea lui m
 - pentru $m=b$ sortarea degenerază în sortare cu determinarea distribuțiilor
- Dezavantaj: pentru sortarea tabloului a este necesar tabloul destinație b , fiind necesară readucerea lui în a înaintea fiecărei treceri
- Variantă: în fiecare trecere să se realizeze 2 pași de sortare:
 - 1) $a \rightarrow b$
 - 2) $b \rightarrow a$

3.10.2 Sortarea radix directă (sortare prin distribuție)

➤ Exemplu: 7, 3, 15, 7, 10, 4, 1 $m=2$

| | |
|----|------|
| 7 | 0111 |
| 3 | 0011 |
| 15 | 1111 |
| 7 | 0111 |
| 10 | 1010 |
| 4 | 0100 |
| 1 | 0001 |

| Distribuții | | | | |
|----------------------|---|---|---|---|
| i | 0 | 1 | 2 | 3 |
| D[i] | 1 | 1 | 1 | 4 |
| D[i] | 1 | 2 | 3 | 7 |
| Indici pentru mutare | | | | |
| | 0 | 1 | 2 | 6 |
| | | | | 5 |
| | | | | 4 |
| | | | | 3 |

| | |
|----|------|
| 4 | 0100 |
| 1 | 0001 |
| 10 | 1010 |
| 7 | 0111 |
| 3 | 0011 |
| 15 | 1111 |
| 7 | 0111 |

| Distribuții | | | | |
|----------------------|---|---|---|---|
| i | 0 | 1 | 2 | 3 |
| D[i] | 2 | 3 | 1 | 1 |
| D[i] | 2 | 5 | 6 | 7 |
| Indici pentru mutare | | | | |
| | 1 | 4 | 5 | 6 |
| | 0 | 3 | | |
| | | 2 | | |
| | | | | |

| | |
|----|------|
| 1 | 0001 |
| 3 | 0011 |
| 4 | 0100 |
| 7 | 0111 |
| 7 | 0111 |
| 10 | 1010 |
| 15 | 1111 |

Analiza sortării radix

- Sortarea radix prin interschimbare are **aceleași performante că și quicksort**
 - $C = n \log_2 n$
- Ambele sortări radix (prin schimburi și directă) utilizează $n * b$ comparații de biți pentru a sorta n elemente după b biți
- Sortarea radix directă poate sorta n elemente de lungime b biți în b/m treceri, utilizând un **spațiu suplimentar** de memorie
 - necesar tabloului de distribuții, cu dimensiunea 2^m
 - un tablou suplimentar, de dimensiunea celui de sortat
- Pornind de la observația „creșterea lui m reduce numărul de treceri”, sortarea radix directă poate conduce la performanțe apropiate de cele oferite de o sortare liniară, pentru un m suficient de mare

3.11 Sortarea indirectă (Sortarea tablourilor cu elemente de mari dimensiuni)

- În cazul în care dimensiunea elementelor de sortat este mare, „**costul**” **mișcării** acestora în cadrul tabloului în cazul metodelor de sortare prezentate poate deveni semnificativ, conducând astfel la diminuarea performanțelor
- Sortarea indirectă presupune accesarea elementelor tabloului de sortat prin intermediul unui **tablou suplimentar** numit tablou **de indici**, care va conține indicatori (pointeri) spre elementele tabloului
- Sortarea se va realiza în tabloul de indici, urmând ca tabloul original să fie sortat ulterior într-o singură trecere
- Considerând tabloul de sortat a și tabloul de indici p , în procesul de sortare se vor compara elementele din a (accesul în a realizându-se prin intermediul indicilor din p) și se vor mișca (muta) elementele din p
- În multe situații este suficientă doar sortarea tabloului de indici, fără a trebui reordonate elementele tabloului

3.12 Sortare externă. Sortarea fișierelor secvențiale

3.12.1 Tehnica sortării prin interclasare (mergeSort)

- **Accesul strict secvențial** la componentele structurii secvență conduce la modificări ale tehnicilor de sortare
 - restricție severă comparativ cu accesul direct specific structurii de tablou
- Principiul de bază al acestor metode de sortare este **interclasarea**
 - presupune combinarea a două sau mai multe secvențe ordonate într-o singură secvență ordonată, prin selecții repetate ale componentelor curent accesibile, în manieră secvențială
 - Ex. : interclasarea secvenței 1,3,4,5,8,9 cu secvența 2,6,7,10,14,17
- Aplicarea metodei de sortare prin interclasare constă în construirea unor secvențe sortate, începând cu secvențe de lungime 1, și interclasarea lor
 - 1,2,3,4,5,6,7,8,9,10,14,17
- Fișierul secvențial supus sortării este simulat printr-un tablou

3.12.1 Tehnica sortării prin interclasare (mergeSort)

➤ E o tehnică divide-et-impera (divide-and-conquer)

➤ Rezultă astfel etapele:

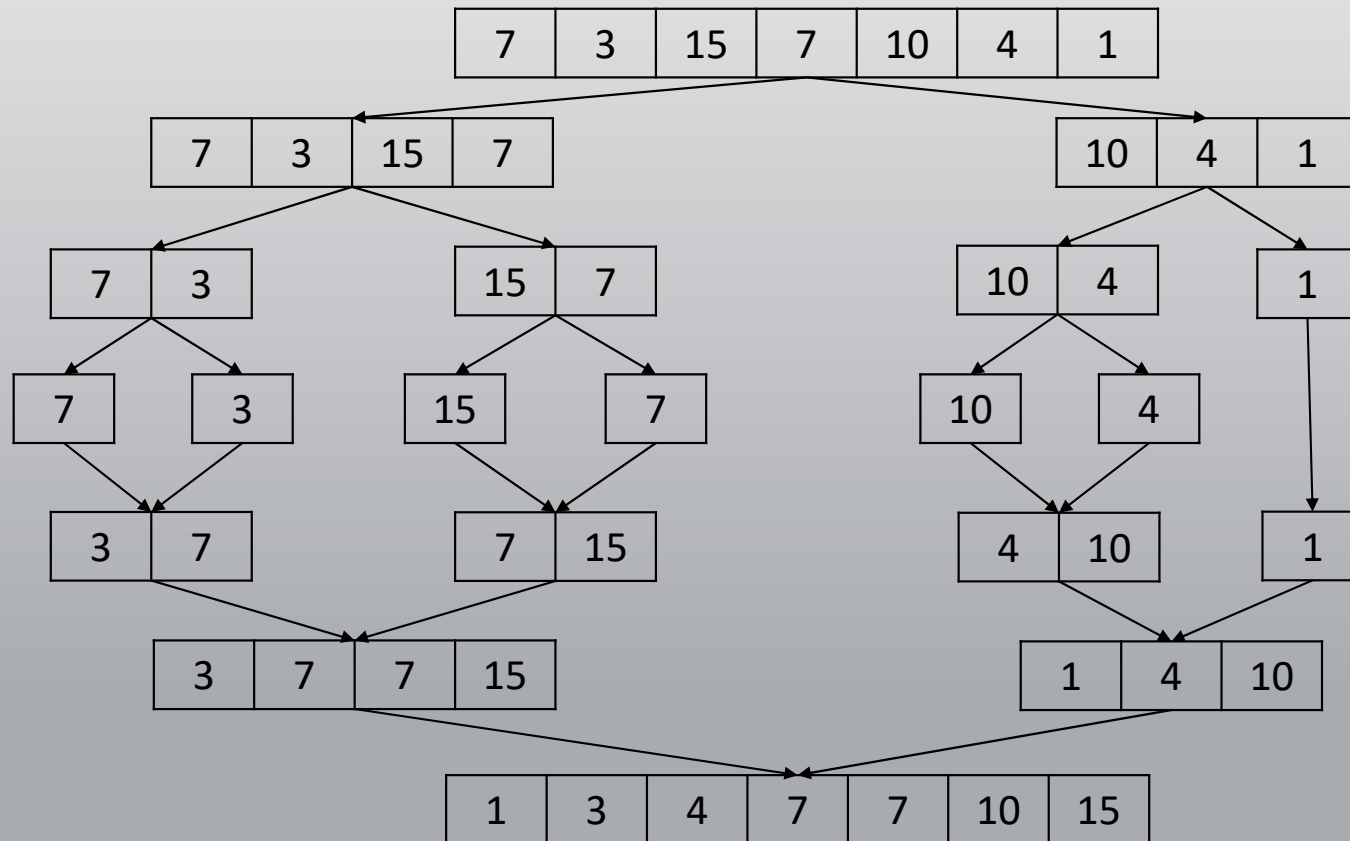
1. *Divide*: Se împarte secvența de interclasat a în două sub-secvențe b și c , egale sau diferind cu 1 ca lungime (defalcare, înjumătățire) \Rightarrow două subprobleme, instanțe ale aceleiași probleme
2. *Conquer*: Se rezolvă recursiv subproblemele
3. *Combine*: Se interclasează soluțiile b și c ale subproblemelor în a , combinând câte un element din fiecare în perechi ordonate

```
mergeSort(A, 0, length(A)-1)
```

```
mergeSort(A, p, r) :  
    if (p > r)  
        return  
    q = (p+r)/2  
    mergeSort(A, p, q)  
    mergeSort(A, q+1, r)  
    merge(A, p, q, r)
```

3.12.1 mergeSort

➤ Exemplan: 7, 3, 15, 7, 10, 4, 1



3.12.1 Tehnica sortării prin interclasare (mergeSort)

- Defalcarea se oprește când sub-secvențele de sortat au dimensiune 1
- Apoi se realizează interclasarea, prin următorii pași:
 - S-a ajuns la sfârșitul uneia dintre sub-secvențe?
 - Nu:
 - Se compară elementele curente din cele două sub-secvențe
 - Se copiază elementul mai mic în secvența de sortat
 - Se mută cursorul după elementul copiat
 - Da:
 - Se copiază restul sub-secvenței nevide

3.12.1 Tehnica sortării prin interclasare (mergeSort)

```
/*Interclasare două sub-secvențe L și M în secvența arr*/  
void merge(int arr[], int p, int q, int r) {  
    /*Creare L ← A[p..q] și M ← A[q+1..r]*/  
    int n1 = q - p + 1;  
    int n2 = r - q;  
    int L[n1], M[n2];  
  
    for (int i = 0; i < n1; i++)  
        L[i] = arr[p + i];  
    for (int j = 0; j < n2; j++)  
        M[j] = arr[q + 1 + j];  
  
    /*Refacere indecsi*/  
    int i, j, k;  
    i = 0;  
    j = 0;  
    k = p;
```

3.12.1 Tehnica sortării prin interclasare (mergeSort)

```
/*Atata timp cat exista elemente  
in ambele sub-secvnte, elementul mai  
mic se plaseaza in arr*/
```

```
while (i < n1 && j < n2) {  
    if (L[i] <= M[j]) {  
        arr[k] = L[i];  
        i++;  
    } else {  
        arr[k] = M[j];  
        j++;  
    }  
    k++;  
}
```

```
/*La epuizarea uneia dintre  
sub-secvnte, se copiaza in arr  
elementele ramase in cealalta sub-  
secvnta*/
```

```
while (i < n1) {  
    arr[k] = L[i];  
    i++;  
    k++;  
}  
  
while (j < n2) {  
    arr[k] = M[j];  
    j++;  
    k++;  
}  
}
```

3.12.1 mergeSort – analiză

- Trecheri: $\log_2 n$
- În fiecare trecere sunt copiate toate cele n elemente, rezultând astfel numărul total de **mișcări**
 - $M = n \log_2 n$
- Numărul total de **comparații** este mai mic decât numărul de mișcări, deoarece operația de copiere a resturilor nu presupune comparații
 - $C \leq M$

3.13 Concluzii privind sortarea tablourilor

- Metodele de sortare prezentate conduc la următoarele clase de performanțe:
 - $O(n^2)$ – metodele directe, simple
 - $O(n \log_2 n)$ – metodele avansate, complexe
 - $O(n)$ – dacă se dispune de informații suplimentare, zone suplimentare de memorie
- Beneficiile inserției binare față de inserția simplă sunt nesemnificative, chiar negative în cazul tablourilor gata sortate
- Inserția prin schimburi (*bubbleSort*)
 - cea mai puțin eficientă în raport cu selecția și inserția, chiar și în varianta sa îmbunătățită ca sortare amestecată
 - performantă în cazul tablourilor gata sortate
- Dacă se ia în considerare dimensiunea elementelor, selecția directă se situează pe primul loc în cadrul metodelor directe, *bubbleSort* pierde din performanță, iar *quickSort* este considerată cea mai rapidă

3.13 Concluzii privind sortarea tablourilor

➤ Tehnica *quickSort*

- superioară tehnicii *heapSort* (factor 2 la 3).
- sortează un tablou sortat invers practic cu aceeași viteză cu care sortează un tablou gata ordonat

➤ În situațiile în care dispunem de informații suplimentare asupra cheilor de sortat, respectiv dacă utilizăm tablouri suplimentare în sortare, performanța crește semnificativ, ajungându-se chiar la $O(n)$

- *binSort* (limitează domeniul cheilor)
- determinarea distribuțiilor se apropie de performanța $O(n)$

➤ *Radix* (interschimburi și directă) concurează *quickSort*

➤ *Radix* cu tablouri suplimentare pentru distribuții poate ajunge la $O(n)$

➤ Sortarea indirectă câștigă performanță prin renunțarea la mișcarea elementelor

Vă mulțumesc!