Algoritmi şi structuri de date (I). Seminar 7: Variante ale algoritmilor de sortare: sortare prin numărare, sortare pe baza cifrelor, shell sort, shaker sort.

Problema 1 *Counting sort - sortare prin numărare.* Să considerăm problema sortării unui tablou $x[1..n]$ având elemente din $\{1, 2, \ldots, m\}$. Pentru acest caz particular poate fi utilizat un algoritm de complexitate liniară bazat pe următoarele idei:

- se construieşte tabelul $f[1..m]$ al frecvenţelor de apariţie a valorilor elementelor tabloului $x[1..n]$;
- se calculează frecvenţele cumulate asociate;
- se foloseşte tabelul frecvenţelor cumulate pentru a construi tabloul ordonat y.

Algoritmul poate fi descris prin:

```
countingsort(integer x[1..n],m)
integer i, f[1..m], y[1..n]
for i = 1, m do f[i] = 0 endfor
for i = 1, n do f[x[i]] = f[x[i]] + 1 endfor
for i = 2, m do f[i] = f[i − 1] + f[i] endfor
for i = n, 1, −1 do
    y[f[x[i]]] = x[i]
    f[x[i]] = f[x[i]] − 1
endfor
for i = 1, n do x[i] = y[i] endfor
return x[1..n]
```

Stabiliţi ordinul de complexitate al algoritmului `countingsort`.

Rezolvare. Dimensiunea problemei este determinată de perechea (m, n). Considerând că operaîile dominante sunt operaţiile efectuate asupra tabloului de frecevnţe şi cele de construire a tabloului sortat, timpul de execuţie este $T(m, n) = 2(m + n) - 1$. Dacă $m \in \mathcal{O}(n)$ atunci algoritmul de sortare prin numărare are complexitate liniară în raport cu numărul de elemente din tabloul de sortat. Dacă însă m este mult mai mare decât n (de exemplu $m \in \mathcal{O}(n^2)$) atunci ordinul de complexitate al algoritmului de sortare este determinat de m. In ceea ce priveşte spaţiul de memorie adiţional, sortarea prin numărare utilizează un spaţiu adiţional de dimensiune $\mathcal{O}(n)$. Atât timp cât ciclul de construire al tabloului y este descrescător (se porneşte cu contorul i de la n), algoritmul este stabil.

Problema 2 *Radix sort - sortare pe baza cifrelor.* Considerăm un tablou $x[1..n]$ având elemente numere naturale constituite din cel mult k cifre. Dacă $m = 10^k$ este semnificativ mai mare decât n atunci sortarea prin numărare nu este eficientă. Pe de altă parte, dacă k este mult mai mic decât n atunci un algoritm de complexitate $\mathcal{O}(kn)$ ar putea fi acceptabil. Un astfel de algoritm este cel bazat pe următoarea idee: se ordonează tabloul în raport cu cifra cea mai puţin semnificativă a fiecarui număr (folosind counting sort în ipoteza că mulţimea valorilor din tabloul de sortat este mulţimea cifrelor, $\{0, 1, \ldots, 9\}$) după care se sortează în raport cu cifra de rang imediat superior ş.a.m.d. până se ajunge la cifra cea mai semnificativă. Deci structura generală a algoritmului este:

```
radixsort(integer x[1..n], k)
integer i
for i = 0, k − 1 do
    x[1..n] = counting2(x[1..n], 9, i)
endfor
return x[1..n]
```

Algoritmul `counting2` este o adaptare a algoritmului de sortare prin numărare în raport cu cifrele de un anumit rang (i):

```
counting2(integer x[1..n],m,c )
integer i, j, f[0..m], y[1..n]
for i = 0, m do f[i] = 0 endfor
for i = 1, n do
    j = (x[i]DIV putere(10, c))MOD 10
    f[j] = f[j] + 1
endfor
for i = 1, m do f[i] = f[i − 1] + f[i] endfor
for i = n, 1, −1 do
    j = (x[i]DIV putere(10, c))MOD 10
    y[f[j]] = x[i]
    f[j] = f[j] − 1
endfor
for i = 1, n do x[i] = y[i] endfor
return x[1..n]
```

Stabiliţi ordinul de complexitate al algoritmului `radixsort`.

Rezolvare. Algoritmul `counting2` se apelează pentru $m = 9$ (care este valoare constantă în raport cu n) deci este de ordinul $\mathcal{O}(n)$. Pe de altă parte pentru ca algoritmul de sortare pe baza cifrelor să funcţioneze corect este necesar ca subalgoritmul de sortare prin numărare apelat să fie stabil (vezi observaţia de la exemplul anterior).

Problema 3 (*Shell sorting - sortare prin inserţie cu pas variabil.*) Unul dintre dezavantajele sortării prin inserţie este faptul că la fiecare etapă un element al şirului se deplasează cu o singură poziţie. O variantă de reducere a numărului de operaţii efectuate este de a compara elemente aflate la o distanţă mai mare ($h \geq 1$) şi de a realiza deplasarea acestor elemente peste mai multe poziţii. De fapt tehnica se aplică în mod repetat pentru valori din ce în ce mai mici ale pasului h, asigurând h-sortarea şirului. Un şir $x[1..n]$ este considerat h-sortat dacă orice subşir $x[i_0], x[i_0 + h], x[i_0 + 2h]$... este sortat ($i_0 \in \{1, \ldots, h\}$). Aceasta este ideea algoritmului propus de Donald Shell în 1959 cunoscut sub numele "Shell sort".

Elementul cheie al algoritmului îl reprezintă alegerea valorilor pasului h. Pentru alegeri adecvate ale secvenţei h_k se poate obţine un algoritm de complexitate $\mathcal{O}(n^{3/2})$ în loc de $\mathcal{O}(n^2)$ cum este în cazul algoritmului clasic de sortare prin inserţie.

Exemplu. Exemplificăm ideea algoritmului în cazul unui şir cu 15 elemente pentru următoarele valori ale pasului: $h = 13$, $h = 4$, $h = 1$ (care corespund unui şir h_k dat prin relaţia $h_k = 3h_{k-1} + 1$, $h_1 = 1$):

Etapa 1: pentru $h = 13$ se aplică algoritmul sortării prin inserţie subşirurilor $x[1], x[14]$ şi $x[2], x[15]$, singurele subşiruri cu elemente aflate la distanţa h care au mai mult de un element:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
14	8	10	7	9	12	5	15	2	13	6	1	4	3	11

şi se obţine

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	8	10	7	9	12	5	15	2	13	6	1	4	14	11

Etapa 2: pentru $h = 4$ se aplică algoritmul sortării prin inserţie succesiv subşirurilor: $x[1], x[5], x[9], x[13]$, $x[2], x[6], x[10], x[14]$, $x[3], x[7], x[11], x[15]$, $x[4], x[8], x[12]$. După prima subetapă (prelucrarea primului subşir) prin care se ordonează subşirul constituit din elementele marcate:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	8	10	7	9	12	5	15	2	13	6	1	4	14	11

se obţine:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	8	10	7	3	12	5	15	4	13	6	1	9	14	11

La a doua subetapă se aplică sortarea prin inserţie asupra subşirului constituit din elementele marcate:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	[8]	10	7	3	[12]	5	15	4	[13]	6	1	9	[14]	11

obţinându-se aceeaşi configuraţie (subşirul este deja ordonat crescător) din care se prelucrează acum subşirul constituit din elementele marcate:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	8	[10]	7	3	12	[5]	15	4	13	[6]	1	9	14	[11]

obţinându-se

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	8	[5]	7	3	12	[6]	15	4	13	[10]	1	9	14	[11]

Se aplică acum sortarea prin inserţie asupra subşirului constituit din elementele marcate:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	8	5	[7]	3	12	6	[15]	4	13	10	[1]	9	14	11

obţinându-se

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	8	5	1	3	12	6	7	4	13	10	15	9	14	11

Se aplică acum sortarea prin inserţie asupra întregului şir. Pentru acest exemplu aplicarea directă a sortării prin inserţie conduce la execuţia a 68 de comparaţii pe când aplicarea sortarii cu pas variabil de inserţie conduce la execuţia a 34 de comparaţii.

Prelucrarea specifică unei etape (o valoare a pasului de inserţie) poate fi descrisă prin:

```
insertie_pas(real x[1..n], integer h)
integer i, j
real aux
for i = h + 1, n do
    aux = x[i]
    j = i − h
    while j ≥ 1 AND aux < x[j] do
        x[j + h] = x[j]
        j = j − h
    endwhile
    x[j + h] = aux
endfor
return x[1..n]
```

Această prelucrare se efectuează pentru diferite valori ale lui h. Un exemplu de şir de valori pentru h (care s-a dovedit a conduce la o variantă eficientă a algoritmului) este cel bazat pe relaţia de recurenţă: $h_k = 3h_{k-1} + 1$, $h_1 = 1$. Structura generală a algoritmului va consta în:

```
shellsort(real x[1..n])
integer h
h = 1
while h ≤ n do
    h = 3 ∗ h + 1
endwhile
repeat
    h = hDIV3
    x[1..n] = insertie_pas(x[1..n], h)
until h = 1
return x[1..n]
```

O altă variantă de alegere a pasului h, pentru care s-a demonstrat că algoritmul are complexitatea $\mathcal{O}(n\sqrt{n})$ este $h_k = 2^k - 1$.

Problema 4 (*Shaker sort.*) Sortarea prin interschimbarea elementelor vecine asigură, la fiecare pas al ciclului exterior, plasarea câte unui element (de exemplu maximul din subtabloul tratat la etapa respectivă) pe poziţia finală. O variantă ceva mai eficientă ar fi ca la fiecare etapă să se plaseze pe poziţiile finale câte două elemente (minimul respectiv maximul din subtabloul tratat la etapa respectivă). Pe de altă parte prin reţinerea indicelui ultimei interschimnări efectuate, atât la parcurgerea de la stânga la dreapta cât şi de la dreapta la stânga se poate limita regiunea analizată cu mai mult de o poziţie atât la stânga cât şi la dreapta (când tabloul conţine porţiuni deja sortate).

Structura algoritmului este:

```
shakersort(real x[1..n])
integer s, d, i, t
s = 1; d = n
repeat
    t = 0
    for i = s, d − 1 do
        if x[i] > x[i + 1] then x[i] ↔ x[i + 1]; t = i endif
    endfor
    if t ≠ 0 then
        d = t; t = 0
        for i = d, s + 1, −1 do
            if x[i] < x[i − 1] then x[i] ↔ x[i − 1]; t = i endif
        endfor
        s = t
    endif until t = 0 OR s = d
return x[1..n]
```

Probleme suplimentare

1. Propuneţi un algoritm care sortează crescător elementele de pe poziţiile impare ale unui tablou şi descrescător cele de pe poziţii pare.

   Indicaţie. Se aplică un algoritm de 2-sortare crescătoare (folosind tehnica sortării prin inserţie) începând cu primul element al tabloului şi un algoritm de 2-sortare descrescătoare începând cu al doilea element al tabloului.

2. Adaptaţi algoritmul de sortare prin numărare pentru ordonarea descrescătoare a unui tablou.

   Indicaţie. Cea mai simplă variantă de modificare a algoritmului de sortare crescătoare în sortare descrescătoare este cea prin care la completarea tabloului x (ultimul ciclu `for` din `countingsort` tabloul y este parcurs de la ultimul element către primul element.

3. Se consideră un set de n valori din $\{1, \ldots, m\}$. Preprocesaţi acest set folosind un algoritm de complexitate $\mathcal{O}(\max\{m, n\})$ astfel încât răspunsul la întrebarea "câte elemente se află în intervalul $[a, b]$, $a, b \in \{1, \ldots, m\}$ să poată fi obţinut în timp constant.

   Indicaţie. Se construieşte tabloul $fc[1..m]$ care conţine frecvenţele cumulate ($fc[i]$ conţine numul de elemente din tabloul iniţial care sunt mai mici sau egale cu i) - similar cu modul de construire a acestui tablou în cadrul algoritmului `countingsort`. Pentru a determina numărul de elemente din intervalul $[a, b]$ este suficient să se calculeze $fc[b] - fc[a] + f[a]$ ($f[a]$ reprezintă numărul de elemente din set care au valoarea a şi trebuie adăugat la diferenţă pentru că $fc[b] - fc[a]$ reprezintă numărul de elemente din $(a, b]$ (strict mai mari decât a şi mai mici sau egale decât b).

4. Propuneţi un algoritm de complexitate liniară pentru ordonarea crescătoare a unui tablou constituit din n valori întregi aparţinând mulţimii $\{0, 1, 2, \ldots, n^2 - 1\}$.

   Indicaţie. Se vor interpreta valorile ca fiind reprezentate în baza n (în baza n cifrele pot fi 0, 1, ..., $n - 1$, deci elementele tabloului au cel mult doua "cifre"). Este suficient să se modifica algoritmul `radixsort` considerând $k = 1$ iar în loc de 9 în apelul lui `counting2` se consideră $n - 1$. In aceste condiţii numărul de operaţii este de ordinul $2n$ adică algoritmul va fi de complexitate $\mathcal{O}(n)$.

5. Se consideră un tablou constituit din triplete de trei valori întregi corespunzătoare unei date calendaristice (zi, luna, an). Propuneţi un algoritm de ordonare crescătoare după valoarea datei.

   Indicaţie. Se poate utiliza succesiv un algoritm *stabil* (în cazul în care n este de ordinul miilor sau mai mare se poate utiliza sortarea pe baza cifrelor): se sortează prima dată după an, apoi după lună şi în final după zi.

6. Se consideră o succesiune de $2n$ discuri de culori alternate: un disc roşu este urmat de un disc albastru, iar un disc albastru este urmat de un disc roşu (ARARAR... sau RARARA...). Se pune problema ca toate discurile de aceeaşi culoare să fie grupate (de exemplu AAARRR sau RRRAAA) în condiţiile în care singura operaţie permisă este interschimbarea a două discuri vecine. Propuneţi un algoritm de rezolvare a problemei şi stabiliţi ordinul de complexitate al algoritmului.

   Indicaţie. Se poate aplica ideea de la sortarea prin interschimbarea elementelor vecine (sau shaker sort) obţinându-se un algoritm de complexitate $\mathcal{O}(n^2)$.

7. Se consideră un set de n puncte în plan specificate prin coordonatele lor carteziene ($\{(x_1, y_1)$, $(x_2, y2)$, ..., $(x_n, y_n)\}$). Să se ordoneze crescător setul de puncte în funcţie de distanţa euclidiană faţă de originea sistemului de axe de coordonate. *Indicaţie.* Distanţa euclidiană de la un punct de coordonate (x_i, y_i) la originea sistemului de coordonate este $\sqrt{x_i^2 + y_i^2}$.

8. Se consideră un set de n localităţi plasate de-a lungul unui drum drept, astfel că poziţiile localităţilor pot fi specificate ca fiind coordonatele $0 \leq x_1 < x_2 < \ldots < x_n$. Se pune problema selectării unei localităţi în care să se plaseze o şcoală astfel încât: (i) să fie minimizată *distanţa maximă* dintre localitatea unde e plasata şcoală şi fiecare dintre celelalte localităţi; (ii) să fie minimizată *media distanţelor* dintre localitatea unde e plasată şcoală şi fiecare dintre celelalte localităţi.

Indicaţie. (i) Având în vedere că $x[1..n]$ este ordonat crescător este suficient să se parcurgă tabloul şi să se determine indicele $imin$ cu proprietatea că $\max(x[imin] - x[1], x[n] - x[imin]) \leq \max(x[i] - x[1], x[n] - x[i])$ pentru fiecare $i \in \{1, \ldots, n\}$. Algoritmul corespunzător va avea ordinul de complexitate $\Theta(n)$. (ii) In varianta bazată pe metoda forţei brute se calculează pentru fiecare element din tablou, $x[i]$, media distanţelor faţă de celelalte elemente: $\frac{1}{n-1} \sum_{j=1}^{n} |x[i] - x[j]|$ şi se determină indicele $imin$ pentru care această medie este minimă. Un astfel de algoritm va avea ordinul de complexitate $\Theta(n^2)$. O altă abordare se bazează pe proprietatea funcţiei $f(y) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - y)^2$ de a atinge valoarea minimă pentru $y = \frac{1}{n} \sum_{i=1}^{n} x_i$. In acest caz este suficient să se calculeze media elementelor din $x[1..n]$ (ordin de complexitate $\Theta(n)$) şi să se determine elementul din x cel mai apropiat de această medie (ordin de complexitate $\Theta(n)$).