**Algoritmi şi structuri de date (I). Seminar 10:** Aplicaţii ale tehnicii divizării. Aplicaţii ale interclasării.

**Problema 1** Fie $a[1..m]$ şi $b[1..n]$ două tablouri ordonate crescător având elemente nu neapărat distincte. Să se construiască un tablou strict crescător ce conţine elementele distincte din $a$ şi $b$.

*Rezolvare.* Se aplică tehnica interclasării verificând de fiecare dată la completarea în tabloul destinaţie dacă elementul ce ar trebui adăugat este diferit de ultimul element din tablou.

```
interclasare(a[1..m],b[1..n])
i = 1; j = 1; k = 0
if a[i] < b[j] then k = k + 1; c[k] = a[i]; i = i + 1
      else if a[i] > b[j] then k = k + 1; c[k] = b[j]; j = j + 1
         else k = k + 1; c[k] = a[i]; i = i + 1; j = j + 1
      endif
endif
while i ≤ m AND j ≤ n do
   if a[i] < b[j] then
      if a[i] ≠ c[k] then k = k + 1; c[k] = a[i]; i = i + 1
      else i = i + 1 endif
   else if a[i] > b[j] then
      if b[j] ≠ c[k] then k = k + 1; c[k] = b[j]; j = j + 1
      else j = j + 1 endif
      if a[i] ≠ c[k] then k = k + 1; c[k] = a[i]; i = i + 1;j = j + 1
      else i = i + 1;j = j + 1 endif
      endif
   endif
endwhile
while i ≤ m do
   if a[i] ≠ c[k] then k = k + 1; c[k] = a[i]; i = i + 1
   else i = i + 1 endif
endwhile
while j ≤ n do
   if b[j] ≠ c[k] thenk = k + 1; c[k] = b[j]; j = j + 1
   else j = j + 1 endif
endwhile
return c[1..k]
```

Ordinul de complexitate este în acest caz $\mathcal{O}(m+n)$.

**Problema 2** Se consideră două numere întregi date prin tablourile corespunzătoare descompunerilor lor în factori primi. Să se construiască tablourile similare corespunzătoare celui mai mic multiplu comun al celor două valori.

*Rezolvare.* Să considerăm numerele: $3415 = 3 \cdot 5^3 \cdot 7 \cdot 13$ şi $966280 = 2^3 \cdot 5 \cdot 7^2 \cdot 17 \cdot 29$. Primului număr îi corespund tablourile: $(3, 5, 7, 13)$ respectiv $(1, 3, 1, 1)$ iar celui de al doilea număr îi corespund tablourile $(2, 5, 7, 17, 29)$ respectiv $(3, 1, 2, 1, 1)$.

Tablourile corespunzătoare celui mai mic multiplu comun vor fi:

- *factori primi:* tabloul care conţine toţi factorii primi corespunzători celor două numere şi care poate fi obţinut prin interclasare ţinând cont că aceştia trebuie să fie distincţi.
- *exponenţi:* tabloul ce conţine valorile maxime ale exponenţilor.

În cazul exemplului cele două tablouri sunt: $(2, 3, 5, 7, 13, 17, 29)$ respectiv $(3, 1, 3, 2, 1, 1, 1)$

```
interclasare(fa[1..m],pa[1..m],fb[1..n],pb[1..n] )
i = 1; j = 1; k = 0
while i ≤ m AND j ≤ n do
    if fa[i] < fb[j] then
        k = k + 1; fc[k] = fa[i]; pc[k] = pa[i]; i = i + 1
    else if a[i] > b[j] then
        k = k + 1; fc[k] = fb[j]; pc[k] = pb[j]; j = j + 1
            else k = k + 1; fc[k] = fa[i]; pc[k] = max(pa[i], pb[j]); i = i + 1;j = j + 1
        endif
    endif
endwhile
while i ≤ m do
    k = k + 1; fc[k] = fa[i]; pc[k] = pa[i]; i = i + 1
endwhile
while j ≤ n do
    k = k + 1; fc[k] = fb[j]; pc[k] = pb[j]; j = j + 1
endwhile
return fc[1..k], pc[1..k]
```

**Problema 3** *Problema selecţiei celui de al k-lea element.* Fie $a[1..n]$ un tablou, nu neapărat ordonat. Să se determine al $k$-lea element al tabloului, selectat în ordine crescătoare (pentru $k = 1$ se obţine minimul, pentru $k = n$ se obţine maximul etc.).

*Rezolvare.* O primă variantă de rezolvare o reprezintă ordonarea crescătoare a tabloului (prin metoda selecţiei) până ajung ordonate primele $k$ elemente ale tabloului. Numărul de operaţii efectuate în acest caz este proporţional cu $kn$. Pentru $k$ apropiat de $n/2$ aceasta conduce la un algoritm de complexitate pătratică. O variantă mai eficientă este cea în care se utilizează un algoritm de sortare de complexitate $\mathcal{O}(n \log n)$.

Un algoritm de complexitate liniară se poate obţine aplicând strategia de partiţionare de la sortarea rapidă: folosind o valoare de referinţă (de exemplu cea aflată pe prima poziţie în tablou) se partiţionează tabloul în două subtablouri astfel încât elementele aflate în primul subtablou să fie toate mai mici decât valoarea de referinţă iar elementele din al doilea subtablou să fie toate mai mari decât valoarea de referinţă. Procesul de căutare continuă doar în unul dintre cele două subtablouri, în funcţie de relaţia dintre poziţia de partiîonare şi numărul de ordine al elementului căutat.

Algoritmul de partiţionare poate fi cel folosit în cadrul sortării rapide:

```
partiţionare(integer a[li..ls])
v = a[li]; i = li − 1; j = lj + 1;
while i < j do
    repeat i = i + 1 until a[i] ≥ v
    repeat j = j − 1 until a[j] ≤ v
    if i < j then a[i] ↔ a[j] endif
endwhile
return j
```

Dacă poziţia de partiţionare este $q$ iar $k \leq q - li + 1$ atunci problema se reduce la selecţia celui de al $k$-lea element din subtabloul $a[li..q]$. Dacă însă $k > q - li + 1$ atunci problema se reduce la selecţia celui de al $k - (q - li + 1)$ element din subtabloul $a[q + 1..ls]$. Valoarea parametrului $k$ este întotdeauna cuprinsă între 1 şi numărul de elemente din subtabloul prelucrat (în cazul în care $li = ls$ valoarea lui $k$ va fi 1). Algoritmul poate fi descris prin:

```
selectie(a[li..ls], k)
if li = ls then return a[li]
else
    q =partitionare(a[li..ls])
    r = q − li + 1
    if k ≤ r then selectie(a[li..q], k)
        else selectie(a[q + 1..ls], k − r)
    endif
endif
```

În cazul cel mai favorabil, partiţionarea este echilibrată la fiecare etapă (cele două subtablouri au aproximativ acelaşi număr de elemente). Întrucât algoritmul de partiţionare are complexitate liniară putem presupune că numărul de comparaţii efectuate asupra elementelor tabloului satisface următoarea relaţie de recurenţă:

$$T(n) = \begin{cases} 0 & n = 1 \\ T(n/2) + n & n > 1 \end{cases}$$

Aplicând teorema master pentru estimarea ordinului de complexitate (pentru $m = 2$, $d = 1$, $k = 1$) se obţine că, în cazul cel mai favorabil, algoritmul are complexitate liniară. În cazul cel mai defavorabil partiţionarea ar conduce la fiecare etapă la descompunerea într-un subtablou constituit dintr-un singur element şi la un subtablou constituit din celelalte elemente. Relaţia de recurenţă ar fi în acest caz:

$$T(n) = \begin{cases} 0 & n = 1 \\ T(n-1) + n & n > 1 \end{cases}$$

ceea ce conduce la o complexitate pătratică. La fel ca în cazul algoritmului de sortare rapidă, algoritmul selecţiei se comportă în medie similar celui mai favorabil caz, având o complexitate liniară.

**Problema 4** *Determinare mediană.* Mediana unui tablou cu $n$ elemente este elementul aflat pe poziţia $\lfloor (n+1)/2 \rfloor$ în cadrul tabloului ordonat crescător (obs: în cazul în care $n$ este par se consideră uneori ca mediană media aritmetică a valorilor aflate în mijlocul tabloului). Fie $x[1..n]$ şi $y[1..n]$ două tablouri ordonate crescător. Să se determine mediana tabloului $z[1..2n]$ care conţine toate elementele din $x$ şi $y$.

*Rezolvare.* O primă variantă ar fi să se construiască tabloul $z$ prin interclasare şi să se returneze elementul de pe poziţia $n$. Este însă suficient să se realizeze o interclasare parţială până când se obţine elementul de poziţia $n$. În acest scop se poate folosi interclasarea bazată pe valori santinelă mai mari decât elementele din ambele tablouri.

```
interclasare ( x[1..n], y[1..n])
x[n + 1] = |x[n]| + |y[n]|; y[n + 1] = |x[n]| + |y[n]|;
i = 1; j = 1;
for k = 1, n do
    if x[i] < y[j] then z[k] = x[i]; i = i + 1
    else z[k] = y[j]; j = j + 1
    endif
endfor
return z[n]
```

Este evident că algoritmul are complexitate liniară.

**Probleme suplimentare**

1. Fie $a[1..m]$ şi $b[1..n]$ două tablouri ordonate strict crescător. Propuneţi un algoritm de complexitate liniară pentru determinarea mulţimii elementelor comune celor două tablouri.

   *Indicaţie.* Se aplică ideea de la interclasare: la întâlnirea a două elemente $a[i] = b[j]$ se transferă valoarea comună în tabloul care va conţine elementele intersecţiei şi se progresează atât în tabloul $a$

(se incrementează $i$) cât şi în tabloul $b$ (se incrementează $j$). Dacă $a[i] < b[j]$ se progresează doar în tabloul $a$, altfel se progresează doar în tabloul $b$.

2. Se consideră două numere întregi date prin tablourile corespunzătoare descompunerilor lor în factori primi. Să se construiască tablourile similare corespunzătoare celui mai mare divizor comun al celor două valori.

   *Indicaţie.* Tablourile cu factorii primi şi puterile corespunzătoare celui mai mare divizor comun se construiesc în manieră similară modului în care se construieşte intersecţia a două mulţimi, singura diferenţă fiind faptul că pentru fiecare factor preluat se preia şi minimul dintre exponenţii corespunzători factorului respectiv.

3. Se consideră trei tablouri $a[1..m]$, $b[1..n]$ şi $c[1..p]$ cu proprietăîle: $a$ şi $b$ sunt ordonate strict crescător iar $c$ este ordonat strict descrescător. Propuneţi un algoritm de complexitate liniară în raport cu dimensiunile celor trei tablouri care construieşte tabloul crescător $d[1..m+n+p]$ care conţine toate elementele tablourilor $a$, $b$ şi $c$.

   *Indicaţie.* O variantă de rezolvare (care poate fi extinsă pentru un număr arbitrar tablouri) este de a interclasa tablourile $a$ şi $b$ (ordin de complexitate $\mathcal{O}(m+n) = \mathcal{O}(\max(m,n))$), iar rezultatul se interclasează cu $c$ (ordin de complexitate $\mathcal{O}(m+n+p) = \mathcal{O}(\max(m,n,p))$), prin parcurgerea tabloului $c$ de la ultimul element către primul element.

4. Propuneţi un algoritm de complexitate $\mathcal{O}(n \log n)$ pentru a determina numărul de perechi $(a[i], a[j])$ ale unui tablou $a[1..n]$ (tabloul are elemente distincte) având proprietatea că $i < j$ şi $a[i] > a[j]$.

   *Indicaţie.* Se foloseşte ideea de la sortarea prin interclasare (cu $i$ contor în subtabloul din stânga şi $j$ contor în subtabloul din dreapta) iar la fiecare transfer al unui element $a[i]$ din subtabloul din stâga se incrementează cu $j-(m+1)$ numărul de inversiuni (întrucât $a[i]$ este strict mai mare decât toate elementele din subtabloul din dreapta având indici cuprinşi între $m+1$ şi $j-1$. O variantă de implementare (în ipoteza că tabloul $a$ şi contorul $nr$ sunt variabile globale este:

```
def interclasare(s,d,m):
    global nr
    i=s
    j=m+1
    c=[]
    while (i<=m) and (j<=d):
        if a[i]<a[j]:
            c.append(a[i])
            i = i+1
            nr = nr+(j-(m+1))
        elif a[i]==a[j]:
            c.append(a[i])
            c.append(a[j])
            nr = nr+(j-(m+1))
            i = i+1
            j = j+1
        else:
            c.append(a[j])
            j = j+1
    while i<=m:
        c.append(a[i])
        nr = nr + (j-(m+1))
        i=i+1
    while j<=d:
        c.append(a[j])
        j=j+1
```

```
    a[s:d+1]=c

def inv(s,d):
     if s<d:
         m=(s+d)//2
         inv(s,m)
         inv(m+1,d)
         interclasare(s,d,m)
```

5. Se consideră un tablou $a[1..n]$ care conţine numere întregi (atât pozitive cât şi negative). Propuneţi un algoritm de complexitate liniară (şi care foloseşte spaţiu suplimentar de dimensiune constantă) pentru a transforma tabloul iniţial astfel încât toate valorile negative să fie înaintea celor pozitive.

   *Indicaţie.* Se utilizează algoritmul de partiţionare de la sortarea rapidă folosind pe 0 ca valoare a pivotului. O variantă de implementare este:

```
def reorganizare(a):
    n=len(a)
    i=0
    j=n-1
    while i<j:
        while (i<n-1) and (a[i]<0):
            i=i+1
        while (j>0) and (a[j]>0):
            j=j-1
        if i<j:
            a[i],a[j]=a[j],a[i]
    return a
```

6. Se consideră un şir de caractere $x[0..n-1]$ care conţine simboluri din mulţimea $\{a, b, c\}$. Propuneţi un algoritm de complexitate liniară care transformă şirul $a$ astfel încât toate simbolurile "a" să fie grupate la începutul şirului, toate simbolurile "c" să fie la sfârşitul şirului iar simbolurile "b" să fie în mijlocul şirului.

   *Indicaţie.* O variantă bazată pe tehnica forţei brute poate consta în determinarea numărului de elemente egale cu fiecare dintre cele trei simboluri ($nra$, $nrb$ respectiv $nrc$) şi completarea cu $a$ a primelor $nra$ poziţii, cu $b$ a următoarelor $nrb$ poziţii şi cu $c$ a ultimelor $nrc$ poziţii. Algoritmul are ordinul de complexitate $\Theta(n)$ însă necesită parcurgerea de două ori a elementelor. O variantă care se bazează pe interschimbări de elemente şi extinde algoritmul de partiţionare folosit la `quicksort` este:

```
def partitie3(x,val):  #val reprezinta valoarea din mijloc (b)
    k=len(x)-1
    i=0
    j=0
    while j<=k:
        if x[j]<val:
            x[i],x[j]=x[j],x[i]
            i=i+1
            j=j+1
        elif x[j]>val:
            x[j],x[k]=x[k],x[j]
            k=k-1
        else:
            j=j+1
    return x
```

Ideea partiţionării este de a gestiona cele trei contoare ($i$, $j$ şi $k$) astfel încât în $x[0..i-1]$ să fie prima valoare ($a$) în $x[i..j-1]$ să fie a doua valoare ($b$), în $x[k+1..n-1]$ să fie a treia valoare ($c$) iar porţiuna $x[j..k]$ să nu fie încă procesată.