Algoritmi şi structuri de date (I). Seminar 6: Stabilirea ordinului de complexitate pornind de la numărul de operaţii efectuate. Proprietăţi şi calcule cu ordine de complexitate. Analiza complexităţii în cazul mediu.

Problema 1 Să se determine ordinul de creştere al valorii variabilei x (în funcţie de n) după execuţia algoritmilor `alg1`, `alg2`, `alg3`, `alg4`, `alg5` şi `alg6`.

```
alg1(integer n)
1: x ← 0
2: for i ← 1, n do
3:    for j ← 1, i do
4:       for k ← 1, j do
5:          x ← x + 1
6:       end for
7:    end for
8: end for
9: return x
```

```
alg2(integer n)
1: x ← 0
2: i ← n
3: while i ≥ 1 do
4:    x ← x + 1
5:    i ← i DIV 2
6: end while
7: return x
```

```
alg3(integer n)
1: x ← 0
2: i ← n
3: while i ≥ 1 do
4:    for j ← 1, n do
5:       x ← x + 1
6:    end for
7:    i ← i DIV 2
8: end while
9: return x
```

```
alg4(integer n)
1: x ← 0
2: i ← n
3: while i ≥ 1 do
4:    for j ← 1, i do
5:       x ← x + 1
6:    end for
7:    i ← i DIV 2
8: end while
9: return x
```

Indicaţie. `alg1`:

$$x = \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j} 1 = \sum_{i=1}^{n}\sum_{j=1}^{i} j = \sum_{i=1}^{n} \frac{i(i+1)}{2} = \frac{n(n+1)(n+2)}{6} \in \Theta(n^3)$$

`alg2`: x va conţine numărul de cifre ale reprezentării în baza 2 a lui n, adică $\lfloor \log_2(n) \rfloor + 1 \in \Theta(\lg n)$.

`alg3`: Spre deosebire de exemplul anterior, pentru fiecare valoare a lui i variabila x este mărită cu n, deci valoarea finală va fi $n(\lfloor \log_2(n) \rfloor + 1) \in \Theta(n \lg n)$.

`alg4`: Este similar algoritmului `alg3` însă pe linia 4 limita superioară a ciclului **for** este i în loc de n. Variabila x va conţine suma $n + \lfloor \frac{n}{2} \rfloor + \ldots + \lfloor \frac{n}{2^k} \rfloor$ cu k având proprietatea că $2^k \leq n < 2^{k+1}$. Ordinul de mărime al lui x se poate stabili pornind de la observaţia că $2^{k+1} - 1 \leq x < 2(2^{k+1} - 1)$. Deci $x \in \Theta(2^k) = \Theta(n)$.

```
alg5(integer n)
1: x ← 0
2: i ← 1
3: while i < n do
4:    x ← x + 1
5:    i ← 2 * i
6: end while
7: return x
```

```
alg6(integer n)
1: x ← 0
2: i ← 2
3: while i < n do
4:    x ← x + 1
5:    i ← i * i
6: end while
7: return x
```

`alg5:` Variabila x va conţine cel mai mare număr natural k cu proprietatea că $2^k \leq n$, deci $x = \lfloor \log_2 n \rfloor \in \Theta(\lg n)$.

`alg6:` Variabila x va conţine cel mai mare număr natural k cu proprietatea că $2^{2^k} \leq n$ deci $x = \lfloor \log_2 \log_2 n \rfloor \in \Theta(\lg\lg n)$.

Problema 2 Să se determine termenul dominant şi ordinul de creştere pentru expresiile:

(a) $2\lg n + 4n + 3n\lg n$

(b) $2 + 4 + 6 + \ldots 2n$

(c) $\frac{(n+1)(n+3)}{n+2}$

(d) $2 + 4 + 8 + \ldots + 2^n$

Indicaţie. (a) Termenul dominant în $2\lg n + 4n + 3n\lg n$ este evident $3n\lg n$ iar ordinul de creştere este $n\lg n$.
(b) Termenul dominant al sumei $2 + 4 + 6 + \ldots 2n$ nu este $2n$ ci n^2 întrucât $2 + 4 + 6 + \ldots 2n = n(n+1)$. Deci ordinul de creştere este chiar n^2.
(c) Termenul dominant este $n^2/(n+2)$ iar ordinul de creştere este n.
(d) Întrucât $2 + 4 + 8 + \ldots + 2^n = 2(1 + 2 + 4 + \ldots + 2^{n-1}) = 2(2^n - 1)$, termenul dominant este $2 \cdot 2^n$ iar ordinul de creştere este 2^n.

Problema 3 Să se arate că:

(a) $n! \in \mathcal{O}(n^n)$, $n! \in \Omega(2^n)$

(b) $\lg n! \in \Theta(n\lg n)$

(c) $2^n \in \mathcal{O}(n!)$

(d) $\sum_{i=1}^{n} i\lg i \in \mathcal{O}(n^2\lg n)$

(e) $\lg(n^k + c) \in \Theta(\lg n)$, $k > 0, c > 0$.

Indicaţie. (a) Întrucât $n! \leq n^n$ pentru orice n natural, rezultă imediat că $n! \in \mathcal{O}(n^n)$. Pe de altă parte, $n! \geq 2^{n-1}$ pentru orice n, deci $n! \in \Omega(2^n)$.
(b) Se porneşte de la aproximaţia Stirling $n! \simeq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ care este adevărată pentru valori mari ale lui n. Mai exact, există $c_1, c_2 \in \mathbb{R}_+$ şi $n_0 \in \mathbb{N}$ astfel încât

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \frac{c_1}{n}\right) \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \frac{c_2}{n}\right)$$

pentru orice $n \geq n_0$.

Prin logaritmare se obţine:

$$\ln(2\pi n)/2 + \ln(1 + c_1/n) + n\ln n - n \leq \ln n \leq \ln(2\pi n)/2 + \ln(1 + c_1/n) + n\ln n - n$$

deci $\ln n! \in \Theta(n\ln n)$. Facând abstracţie de baza logaritmului se obţine $\lg n! \in \Theta(n\lg n)$
(c) Cum $2^n < n!$ pentru orice $n \geq 4$ rezultă că $2^n \in \mathcal{O}(n!)$.
(d) $\sum_{i=1}^{n} i\lg i \leq \lg n \sum_{i=1}^{n} i \leq n^2\lg n$, deci $\sum_{i=1}^{n} i\lg i \in \mathcal{O}(n^2\lg n)$.
(e) Pentru valori suficient de mari ale lui n are loc $\lg n^k \leq \lg(n^k + c) \leq \lg(2n^k)$, adică $k\lg n \leq \lg(n^k + c) \leq k\lg(n) + \lg 2$, deci $\lg(n^k + c) \in \Theta(\lg n)$.

Problema 4 Care dintre următoarele afirmaţii sunt adevărate ? Demonstraţi.

(a) $\sum_{i=1}^{n} i^2 \in \Theta(n^2)$, $\sum_{i=1}^{n} i^2 \in \mathcal{O}(n^2)$, $\sum_{i=1}^{n} i^2 \in \Omega(n^2)$

(b) $cf(n) \in \Theta(f(n))$, $f(cn) \in \Theta(f(n))$

(c) $2^{n+1} \in \mathcal{O}(2^n)$, $2^{2n} \in \mathcal{O}(2^n)$?

Indicaţie. (a) Întrucât $\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$ rezultă că $\sum_{i=1}^{n} i^2 \in \Theta(n^3)$, deci $\sum_{i=1}^{n} i^2 \in \Omega(n^2)$ însa celelalte două afirmaţii sunt false.
(b)Întrucât $c_1 f(n) \leq cf(n) \leq c_2 f(n)$ pentru constante c_1 şi c_2 satisfăcând $0 < c_1 \leq c \leq c_2$ rezultă că $cf(n) \in \Theta(f(n))$. In schimb $f(cn) \in \Theta(f(n))$ nu este adevărată pentru orice funcţie f şi orice constantă c. De exemplu $f(n) = \exp(n)$ şi $c > 1$ nu satisfac această proprietate întrucât nu există c' astfel încât $\exp(cn) \leq c' \exp(n)$ pentru valori mari ale lui n.
(c) Întrucât $2^{n+1} = 2 \cdot 2^n$ rezultă că $2^{n+1} \in \Theta(2^n)$ deci implicit şi $2^{n+1} \in \mathcal{O}(2^n)$. Pe de altă parte, $\lim_{n\to\infty} 2^{2n}/2^n = \infty$ prin urmare $2^{2n} \notin \mathcal{O}(2^n)$ dar $2^{2n} \in \Omega(2^n)$.

Problema 5 Propuneţi un algoritm pentru determinarea celor mai mici două valori dintr-un tablou. Determinaţi numărul de comparaţii efectuate asupra elementelor tabloului şi stabiliţi ordinul de complexitate al algoritmului.

Indicaţie. O variantă de algoritm este descrisă în `valori_minime`.

```
valori_minime(integer a[1..n])
if a[1] < a[2] then
   min1 ← a[1]; min2 ← a[2];
else
   min1 ← a[2]; min2 ← a[1];
end if
for i ← 3, n do
   if a[i] < min1 then
      min2 ← min1; min2 ← a[i]
   else if a[i] < min2 then
      min2 ← a[i]
   end if
end for
return min1, min2
```

În cazul cel mai nefavorabil numărul de comparaţii efectuate asupra elementelor tabloului este $T(n) = 1 + 2(n-2) = 2n - 3$ deci algoritmul aparţine lui $\mathcal{O}(n)$.

Problema 6 Propuneţi un algoritm de complexitate liniară pentru a determina tabloul frecvenţelor cumulate pornind de la tabloul frecvenţelor simple. Pentru un tablou $(f_1, \ldots, f_n)$ de frecvenţe, tabloul frecvenţelor cumulate $(fc_1, \ldots, fc_n)$ se caracterizează prin $fc_i = \sum_{j=1}^{i} f_j$.

Rezolvare. Este uşor de văzut că un algoritm care calculează pentru fiecare i valoarea sumei $fc_i = \sum_{j=1}^{i} f_j$ necesită efectuarea a $\sum_{i=2}^{n} \sum_{j=1}^{i} 1 = \sum_{i=2}^{n} i = n(n+1)/2 - 1 \in \Theta(n^2)$. Observând că $fc_i = fc_{i-1} + f_i$ pentru $i = \overline{2,n}$ şi $fc_1 = f_1$ rezultă că frecvenţele cumulate pot fi calculate prin algoritmul `frecvente_cumulate` descris în continuare.

```
frecvente_cumulate(integer f[1..n])
integer i,fc[1..n]
fc[1] ← f[1]
for i ← 2, n do
   fc[i] = fc[i − 1] + f[i]
end for
return fc[1..n]
```

Problema 7 Să se stabilească ordinul de complexitate pentru următorii algoritm ce prelucrează date de volum n.

```
for i ← 1, n do
    . . . (Θ(1))
    for j ← 1, 2i do
        . . . (Θ(1))
        k ← j (Θ(1))
        while k ≥ 0 do
            . . . (Θ(1))
            k ← k − 1
        endwhile
    endfor
endfor
```

Problema 8 Să se stabilească ordinul de complexitate pentru următorul algoritm ce prelucrează date de volum n:

```
h ← 1
while h ≤ n do
    . . . (Θ(1))
    h ← 2 ∗ h
endwhile
```

Problema 9 Să se stabilească ordinul de complexitate pentru următorul algoritm ce prelucrează date de volum n:

```
calcul (x[0..n-1])
k ← 0
for i ← 0, n − 1 do
    for j ← 0, n − 1 do
        y[k] ← x[i] ∗ x[j]
        k ← k + 1
    endfor
endfor return y[1..k]
```

Problema 10 Să se arate că $\sum_{i=1}^{n} i^k \in \Theta(n^{k+1})$.

Problema 11 Ce relaţie există între $\Theta(\log_a n)$ şi $\Theta(\log_b n)$?

Problema 12 Propuneţi un algoritm de complexitate $\Theta(n^2)$ şi unul de complexitate $\Theta(n)$ pentru evaluarea unui polinom de grad n.

Probleme suplimentare/teme

1. Se consideră un tablou $x[1..n]$ şi o valoare $x0$ care este prezentă (pe o singură poziţie) în tablou. Presupunem că avem un algoritm de căutare care verifică elementele tabloului într-o ordine aleatoare (dar verifică fiecare element o singură dată). Procedura este similară celei în care avem într-o cutie $n-1$ bile negre şi o singură bilă albă şi efectuăm extrageri (fără a pune bila extrasă înapoi) până la extragerea bilei albe. Estimaţi numărul mediu de comparaţii (sau extrageri de bile) până la identificarea valorii căutate (extragerea bilei albe).

   Indicaţie. Se presupune că elementele (bilele) disponibile pot fi selectate cu aceeaşi probabilitate.

2. Se consideră un şir de caractere $s[1..n]$ şi un şablon (subşir de caractere) $t[1..m]$ (cu $m < n$). Scrieţi un algoritm care verifică dacă şablonul t este sau nu prezent în s şi stabiliţi ordinul de complexitate al algoritmului (considerând comparaţia dintre elementele şirului şi cele ale şablonului ca operaţie dominantă).

   Indicaţie. Cel mai simplu algoritm (nu şi cel mai eficient) se bazează pe parcurgerea şirului s de la poziţia $i = 1$ până la poziţia $i = n-m$ şi compararea element cu element al lui $s[i..i+m-1]$ cu $t[1..m]$.

3. Se consideră un tablou de valori întregi $x[1..n]$ şi o valoare dată, s. Să se verifice dacă există cel puţin doi indici i şi j (nu neapărat distincţi) care au proprietatea că $x[i] + x[j] = s$. Analizaţi complexitatea algoritmului propus.

4. Care este valoarea variabilei x după execuţia prelucrărilor:

```
x ← 0
j ← n
whilej ≥ 1 do
    for i ← 1, j do
        x ← x + 1
    endfor
    j ← jDIV3
endwhile
```

5. Să se determine ordinul de mărime al variabilei x după execuţia următoarelor prelucrări:

```
x ← 0
j ← n
while j ≥ 1 do
   for i ← 1, j do
      x ← x + 1
   end for
   j ← jDIV3
end while
return x
```