Tema 2: Analiza complexităţii algoritmilor. Algoritmi de căutare şi sortare.

Verificare (prin test la seminar): în săptămâna 9-13.12.2019

Submiterea temelor: Se uploadează prin Classroom (cf indicaţiilor primite de la cadrul didactic coordonator de laborator) o arhiva de tip *.zip având numele construit pe baza regulii: NumeStudent_seria_subgrupa_Tema2.zip (de exemplu `AdamEva_IR_sgr3_Tema2.zip`).

Arhiva trebuie să conţină următoarele fişiere:

- Un fişier în format PDF sau DOC care să conţină soluţiile propuse (răspunsuri la întrebări, algoritmi descrişi în pseudocod, explicaţii etc.)
- Câte un fişier cu codul sursă Python corespunzător implementărilor solicitate în enunţ denumit NumeStudent_seria_subgrupa_Tema2_Problema (de exemplu `AdamEva_IR_sgr3_Tema2_Pb1b.py`)

Observaţie: punctajele sunt orientative

1. Se consideră un tablou $x[1..n]$ şi se doreşte construirea unui tablou $m[1..n]$ care conţine pe poziţia i media aritmetică a elementelor din subtabloul $x[1..i]$ ($m[i] = (x[1] + \ldots + x[i])/i$).

   (a)(5p) Propuneţi un algoritm de complexitate $\Theta(n^2)$ pentru construirea tabloului m. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python.

   (b)(10p) Propuneţi un algoritm de complexitate $\Theta(n)$ pentru construirea tabloului m. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python.

2. Se consideră trei tablouri de numere întregi, $a[1..n]$, $b[1..n]$, $c[1..n]$. Se pune problema verificării dacă există cel puţin un element comun în cele trei tablouri. De exemplu tablourile $a = [3, 1, 5, 10]$, $b = [4, 2, 6, 1]$, $c = [5, 3, 1, 7]$ au un element comun, pe când $a = [3, 1, 5, 10]$, $b = [4, 2, 6, 8]$, $c = [15, 6, 1, 7]$ nu au nici un element comun.

   (a)(5p) Propuneţi un algoritm de complexitate $O(n^3)$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python.

   (b)(10p) Propuneţi un algoritm de complexitate $O(n^2)$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python.

   (c)(10p) Presupunând că elementele tablourilor sunt din $\{1, 2, \ldots, m\}$ propuneţi un algoritm de complexitate $O(\max(m, n))$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python. *Indicaţie.* este permisă utilizarea unei zone suplimentare de memorie de dimensiune $\mathcal{O}(n)$.

   (d)(10p) Presupunând că toate cele trei tablouri sunt ordonate crescător propuneţi un algoritm de complexitate $O(n)$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python. *Indicaţie.* Se poate folosi ideea de la tehnica interclasării.

3. Se consideră un tablou, $x[1..n]$, ordonat crescător, v o valoare de acelaşi tip ca elementele tabloului şi algoritmul `alg`.

```
1: alg(x[1..n],v)
2: i ← n
3: while i ≥ 1 and v < x[i] do
4:     i ← i − 1
5: end while
6: return i + 1
```

(a)(5p) Implementaţi algoritmul `alg` în Python şi stabiliţi ce returnează.

(b)(5p) Estimaţi numărul *mediu* de comparaţii efectuate (în ipoteza în care toate clasele de date de intrare au aceeaşi probabilitate de apariţie).

4. (a)(10p) Se consideră o matrice $A = (a_{ij})_{i=\overline{1,m},j=\overline{1,n}}$ cu m linii şi n coloane. Propuneţi (şi implementaţi în Python) un algoritm care rearanjează liniile matricii astfel încât să fie în ordine crescătoare în raport cu norma euclidiană a liniilor (norma euclidiană a liniei i a matricii A este $\sqrt{a_{i1}^2 + a_{i2}^2 + \ldots + a_{in}^2}$). Stabiliţi ordinul de complexitate al algoritmului propus considerând ca operaţii dominante toate operaţiile efectuate asupra elementelor matricii (comparaţii, adunări, înmulţiri).

   (b)(10p) Se consideră o listă cu date de naştere specificate prin triplete de forma (zi, luna, an). Propuneţi (şi implementaţi în Python) un algoritm care ordonează crescător datele de naştere. Stabiliţi ordinul de complexitate al algoritmului propus.

5. Se consideră un tablou $x[1..n]$ ordonat crescător.

   (a)(10p) Implementaţi o funcţie Python care transformă tabloul $x[1..n]$ prin deplasare circulară către dreapta cu $k < n$ poziţii. De exemplu tabloul $[1, 3, 4, 6, 8, 10, 12]$ va fi transformat pentru $k = 3$ în $[8, 10, 12, 1, 3, 4, 6]$.

   (b)(10p) Se consideră că tabloul ordonat $x[1..n]$ a fost transformat prin deplasare circulară la dreapta cu k poziţii. Presupunând că valoarea lui k este cunoscută propuneţi un algoritm de complexitate $\mathcal{O}(\log n)$ care verifică dacă o valoare dată v se află sau nu în tablou. De exemplu se caută valoarea $v = 7$ în $[8, 10, 12, 1, 3, 4, 6]$.

   (c)(20p) Propuneţi un algoritm de complexitate $\mathcal{O}(\log n)$ care să permită rezolvarea problemei de căutare de la punctul anterior şi în cazul în care valoarea k (numărul de deplasări) nu este cunoscută de la început.

6. (20p) Se consideră un tablou $x[1..2n]$ cu elemente din $\{1, 2, \ldots, m\}$ (cu $m < n$). Propuneţi (şi implementaţi în Python) un algoritm care împerechează elementele din tabloul x (se construiesc perechi de forma $(x[i], x[j])$ cu $i < j$ în aşa fel încât fiecare element să fie inclus într-o singură pereche) astfel încât să fie minimizată valoarea maximă a sumei elementelor din perechi. De exemplu pentru $x = [1, 5, 9, 3]$ elementele pot fi grupate în următoarele moduri: $(1, 5)$ şi $(9, 3)$; $(1, 9)$ şi $(5, 3)$; $(1, 3)$ şi $(5, 9)$. In primul caz suma maximă este 12 (maximul dintre 6 şi 12), în al doilea caz este 10 (maximul dintre 10 şi 8), iar în al treilea caz este 14 (maximul dintre 4 şi 14). Prin urmare soluţia este $(1, 9)$ şi $(5, 3)$. Stabiliţi ordinul de complexitate al algoritmului propus.