

# Problema 1

---

Scrive o metoda care primeste un array de  $N$  elemente si un numar  $K$ , si intoarce al  $K$ -lea cel mai mic element (e.g. `find_kth(3, {5,4,3,1,2,1,2,7,6,9,4,5})` trebuie sa intoarca "2". (sirul sortat ar fi 1,1,2,2,3,4,4,...), al 3-lea element din sirul sortat e "2"). Gaseste un algoritm de complexitate mai mica de  $O(n \log n)$  (ex: mai rapid decat "sortare + indexare").

**Solutia 1** (solutia de nota 10): Radix sort. Complexitate  $O(n)$ .

Vectorul de la intrare este sortat in timp liniar folosind metoda radix sort si se returneaza cel de-al  $k$ -lea element din vectorul sortat.

Radix Sort (Sorting array  $A[\text{size}]$ )

```
Create all of the bins.
From the least significant digit to the most significant digit
{
    For each element (from the first to the last)
    {
        Isolate the value of the significant digit.
        Store the element in the bin with the matching significant digit
value.
    }
    For each bin (from the first to the last)
    {
        Retrieve all of the elements and store them back into the array.
    }
}
Destroy all of the bins.
```

Daca nu va amintiti exact algoritmul, mai multe detalii pe : [http://en.wikipedia.org/wiki/Radix\\_sort](http://en.wikipedia.org/wiki/Radix_sort). Un exemplu practic gasiti pe [http://en.wikipedia.org/wiki/Radix\\_sort#An\\_example](http://en.wikipedia.org/wiki/Radix_sort#An_example).

**Solutia 2** (solutie de nota 9): Partition-based selection algorithm. Complexitate timp mediu  $O(n)$ , dar  $O(n^2)$  in cazul cel mai defavorabil.

Se foloseste functia auxiliara partition care imparte un vector de elemente in doua parti: una in care toate elementele sunt mai mici decat un numar, si alta in care elementele sunt mai mari decat numarul dat.

```
function partition(list, left, right, pivotIndex)
    pivotValue := list[pivotIndex]
    swap list[pivotIndex] and list[right] // Move pivot to end
    storeIndex := left
    for i from left to right-1
        if list[i] < pivotValue
            swap list[storeIndex] and list[i]
            storeIndex := storeIndex + 1
    swap list[right] and list[storeIndex] // Move pivot to its final place
    return storeIndex
```

Functia care rezolva problema este:

```
function select(list, left, right, k)
  loop
    select pivotIndex between left and right
    pivotNewIndex := partition(list, left, right, pivotIndex)
    if k = pivotNewIndex
      return list[k]
    else if k < pivotNewIndex
      right := pivotNewIndex-1
    else
      left := pivotNewIndex+1
```

Mai multe detalii pe: [http://en.wikipedia.org/wiki/Selection\\_algorithm#Partition-based\\_general\\_selection\\_algorithm](http://en.wikipedia.org/wiki/Selection_algorithm#Partition-based_general_selection_algorithm) . Tot aici este prezentata si descrierea unei metode folosind mediana medianelor care elimina posibilitatea aparitiei cazului defavorabil de complexitate  $O(n^2)$ .

**Solutia 3** (solutia de nota 5): se parcurge sirul de k ori si la fiecare pas se elimina cel mai mic numar. Complexitate  $O(n*k)$ .

```
function select(list[1..n], k)
  for i from 1 to k
    minIndex = i
    minValue = list[i]
    for j from i+1 to n
      if list[j] < minValue
        minIndex = j
        minValue = list[j]
    swap list[i] and list[minIndex]
  return list[k]
```

---

# Problema 2

---

Sa se calculeze numarul  $T(n)$  al cuvintelor de lungime  $n$  folosind alfabetul  $\{A,B,C,D\}$  care contin secventa ABBD.

**Solutia 1** (neoptimizata): Complexitate de timp  $O(n)$ , complexitate spatiala  $O(n)$ .

```
public static long compute(int n) {
    if (n <= 0) {
        System.out.println("Invalid value for n");
        System.exit(1);
    }

    long[] T = new long[n + 1];
    T[1] = 0;
    T[2] = 0;
    T[3] = 0;
    T[4] = 1;
    T[5] = 8;

    for (int i = 6; i <= n; i++)
        T[i] = 8 * T[i - 1] - 16 * T[i - 2] - T[i - 4] + 4 * T[i - 5];

    return T[n];
}
```

Explicatie:

Fie  $S(n)$  numarul cuvintelor de lungime  $n$  din alfabetul dat care NU contin subsirul ABBD. Atunci  $S(n)+T(n)=4^n$ . Pornim de la observatia ca un sir din  $T(n)$  se poate obtine in doua feluri:

- De la un sir de lungime  $n-1$  care deja contine ABBD si la care adaugam o litera oarecare (in 4 moduri posibile)
- De la un sir care nu contine ABBD (de lungime  $n-4$ ) si la sfarsitul caruia adaugam ABBD.

Rezulta relatia:

$T(n) = 4 * T(n-1) + S(n-4) = 4 * T(n-1) + 4^{n-4} - T(n-4)$ . Scriem aceeasi relatie si pentru  $T(n-1)$ :

$T(n-1) = 4 * T(n-2) + 4^{n-5} - T(n-5)$ . Inmultim a doua relatie cu 4 (pentru a avea la ambele  $4^{n-4}$ ) si le scadem.

Rezulta relatia:

$T(n) - 4 * T(n-1) = 4 * T(n-1) - 16 * T(n-2) - T(n-4) + 4 * T(n-5)$ . De aici recurenta din solutie .

**Solutia 2** (optimizata): Complexitate de timp  $O(n)$ , complexitate spatiala  $O(1)$ , merge pentru numere oricat de mari.

```
public static BigInteger compute2(long n) {
    if (n <= 0) {
        System.out.println("Invalid value for n");
        System.exit(1);
    }
    if (n < 4) {
        return BigInteger.ZERO;
    }
    if (n == 4) {
        return BigInteger.ONE;
    }
    if (n==5) {
        return new BigInteger("8");
    }

    BigInteger FOUR = new BigInteger("4");
    BigInteger EIGHT = new BigInteger("8");
    BigInteger SIXTEEN = new BigInteger("16");

    BigInteger a = null;
    BigInteger b, c, d, e, f;
    d = e = f = BigInteger.ZERO;
    c = BigInteger.ONE;
    b = new BigInteger("8");

    for (long i = 6; i <= n; i++) {
        a = b.multiply(EIGHT).add(c.multiply(SIXTEEN).negate()).add(
            e.negate()).add(f.multiply(FOUR));

        f = e;
        e = d;
        d = c;
        c = b;
        b = a;
    }
    return a;
}
```

# Problema 3

---

Se dă un sir de  $N \leq 10000$  numere naturale distincte. Se cere găsirea unui set oarecare de 4 numere din acest sir, fie ele  $a, b, c, d$  astfel încât  $a + b + c = d$ . Ex: pentru sirul „21 5 7 13 9 6 3 2” o soluție corectă este setul ( $a=6, b=13, c=2, d=21$ ) – problema oferită de InfoArena

**Soluție:** Calcularea sumei și diferenței (pozitive) între oricare 2 elemente ale sirului. Complexitate  $O(n^2)$ .

Se tin două map-uri sums și diffs care au ca și cheie suma, respectiv diferența (pozitivă) dintre două numere din vector iar ca valoare perechea de indici (eventual o lista de perechi de indici) care au contribuit la suma/diferența. Când găsim o valoare de suma egală cu una de diferență ( $a+b=d-c$ ) am găsit soluția cerută.

```
for i=2,n do
    for j=1, i-1 do
        sums.put(v[i]+v[j], (i,j))

        if v[i]<v[j] then
            diffs.put(v[j]-v[i], (j,i))
        else
            diffs.put(v[i]-v[j], (i,j))

        if v[i]+v[j] in diffs then
            v[i]+v[j]=v[k]-v[l]
        if i,j,k,l distinct then
            print solution (v[i], v[j], v[k], v[l])
            exit
        if |v[i]-v[j]| in sums then
            v[i]-v[j]=v[k]+v[l] or v[j]-v[i]=v[k]+v[l]
        if i,j,k,l distinct then
            print solution (v[k], v[l], v[i], v[j]) or (v[k], v[l], v[j], v[i])
            exit
```

Trebuie observat că soluția aceasta propune calcularea sumelor și diferențelor în paralel. Această variantă este mai bună decât cea în care se calculează întâi toate sumele sau toate diferențele, deoarece în prima variantă există posibilitatea găsirii unei soluții fără a fi nevoie să procesăm toate numerele.

Hint: lista de perechi de indici trebuie menținută dacă dorim să ne asigurăm că valorile  $a, b, c, d$  sunt oricare 2 diferite. Puteti să vă gândiți de ce este suficient să păstrăm maxim 3 elemente în fiecare listă din hash-ul sums și cum poate fi modificat pseudocodul pentru asta.

# Problema 4

---

Pentru un numar natural nenul  $N$  se noteaza cu  $S(N)$  cifra obtinuta prin urmatorul procedeu: se calculeaza suma cifrelor lui  $N$  si se obtine un numar, apoi se calculeaza suma cifrelor acestui numar si se obtine alt numar, s.a.m.d pana cand rezultatul insumarii cifrelor este mai mic strict decat 10. (ex.:  $N=2481$  suma cifrelor este  $2+4+8+1=15$ , iar  $1+5=6 < 10$ , deci  $S(2481)=6$ ). Sa se implementeze un algoritm eficient (complexitate temporala si spatiala optime) care primeste ca intrare vectorul de cifre ale unui numar natural  $N$  si calculeaza  $S(N)$ .

**Solutie:**

```
public static byte recursiveSum(byte[] v) {
    byte ret = 0;

    for(byte b: v)
        ret = (byte) ((ret + b) % 9);

    if (ret == 0)
        return 9;

    return ret;
}
```

**Explicatie:**

Se porneste de la ideea ca un numar si suma cifrelor sale dau acelasi rest la impartirea prin 9 (echivalent cu a spune ca diferenta intre un numar si suma cifrelor sale este mereu multiplu de 9). De exemplu  $abc = (a+b+c) = 99a + 9b$ , cu generalizarea evidenta. Mai mult, rezulta ca un numar  $n$ , suma cifrelor sale, suma sumei cifrelor sale, s.a.m.d pana cand dam de  $S(n)$ , toate dau acelasi rest la impartirea prin 9.

Deci, daca facem suma cifrelor venite la intrare modulo 9 avem valoarea lui  $S(n)$ , cu observatia ca daca restul este 0, de fapt  $S(n)$  este 9 (pentru ca nu se poate sa tot aduni cifre si sa-ti dea 0);  $n=0$  nu se da ca intrare.

**Nota:** Solutia in care se calculeaza efectiv suma cifrelor, suma sumei cifrelor s.a.m.d nu este optima. Este suficient sa se insumeze numai cifrele primite la intrare si la fiecare pas sa se faca modulo 9.