Quick

Se consideră următoarea implementare a algoritmului quicksort:

**int cnt = 0, v[MAX\_N + 1], mai\_mici[MAX\_N], mai\_mari[MAX\_N]; // variabile globale**

int pivot(int st, int dr) {

int nr\_mai\_mici = 0, nr\_mai\_mari = 0;

for (int i = st + 1; i <= dr; i++) {

if (v[i] < v[st])

mai\_mici[++nr\_mai\_mici] = v[i];

else

mai\_mari[++nr\_mai\_mari] = v[i];

**cnt++;**

}

v[st + nr\_mai\_mici] = v[st];

for (int i = 1; i <= nr\_mai\_mici; i++)

v[st + i - 1] = mai\_mici[i];

for (int i = 1; i <= nr\_mai\_mari; i++)

v[st + nr\_mai\_mici + i] = mai\_mari[i];

return st + nr\_mai\_mici;

}

void sorteaza(int st, int dr) {

if (st < dr) {

int p = pivot(st, dr);

sorteaza(st, p-1);

sorteaza(p+1, dr);

}

}

Pentru un șir v cu n elemente stocate pe poziții de la 1 la n, facem apelul sorteaza(1,n);, la finalul căruia șirul v ajunge să fie sortat crescător.

O permutare de n elemente este un șir de n valori naturale cuprinse între 1 și n, distincte două câte două.

Pentru un n dat, sa se determine o permutare de n elemente, pentru care dacă aplicăm algoritmul de mai sus, valoarea variabilei cnt la final este minimă. Dacă există mai multe permutări care satisfac cerința, să se determine permutarea minim lexicografică dintre acestea.

Date de intrare

Fișierul quick.in conține pe prima linie două numere naturale separate prin spațiu, p și n.

Date de ieșire

Pentru p=1 fișierul quick.out conține pe prima linie valoarea minimă obținută în variabila cnt în urma aplicării algoritmului pentru o permutare de lungime n.

Pentru p=2 fișierul quick.out conține pe prima linie n numere naturale distincte, cuprinse între 1 și n, separate prin spații, reprezentând permutarea minim lexicografică de lungime n, corespunzătoare valorii minime calculate în variabila cnt, la sfârșitul aplicării algoritmului de sortare.

Restricții și precizări

* 1 <= n <= 100.000
* Un şir *a1, a2,..., an* este mai mic lexicografic decât un alt şir *b1, b2,..., bn* dacă există un număr întreg *i* (*1 <= i <= n*) astfel încât: *a1 = b1, a2 = b2, ... , ai–1 = bi–1, iar ai < bi*.
* Pentru teste în valoare de 6 puncte p = 1 și n <= 10
* Pentru alte teste în valoare de 9 puncte p = 2 și n <= 10
* Pentru alte teste în valoare de 20 de puncte p = 1 și n <= 5.000
* Pentru alte teste în valoare de 30 de puncte p = 2 și n <= 5.000
* Pentru alte teste în valoare de 14 puncte p = 1

Exemplu

| quick.in | quick.out |
| --- | --- |
| 1 4 | 4 |
| 2 4 | 2 1 3 4 |

Explicație

Pornind de la șirul 2 1 3 4, la apelul sorteaza(1, 4) se fac trei incrementări ale lui cnt. După aceea șirul devine 1 2 3 4. Apoi se mai fac două autoapeluri, sorteaza(1, 1) și sorteaza(3,4). Primul autoapel nu mai efectuează incrementări, iar al doilea efectuează o incrementare, apelând apoi sorteaza(3, 2) si sorteaza(4, 4), care nu mai efectuează alte incrementări. În total avem 4 incrementări.

Similar, pentru șirul 3 2 1 4 se vor efectua tot 4 incrementări: la apelul sorteaza(1, 4) se fac trei incrementări, șirul devine 2 1 3 4, apoi se mai fac două autoapeluri, sorteaza(1, 2) și sorteaza(4,4), ultimul nu mai efectuează incrementări, iar primul efectuează una, șirul devenind 1 2 3 4, apelând mai departe sorteaza(1, 1) si sorteaza(3, 2), care nu mai efectuează alte incrementări. Acest sir (3 2 1 4) nu este o soluție acceptată întrucât nu este mai mic lexicografic decat 2 1 3 4.

Atenție, nu ne interesează momentul în care se sortează șirul ci numărul de incrementări pe care le face algoritmul dat pe permutarea detectată. De exemplu, dacă permutarea obținută era 1,2,3,4 un apel sorteaza(1,4) ar fi făcut trei incrementări, apoi permutarea ar fi rămas aceeași și s-ar fi făcut un autoapel sorteaza(2,4), pentru acesta se mai fac doua incrementări, apoi autoapelul sorteaza(3, 4) care mai face o incrementare, apoi sorteaza(4, 4) care nu mai realizeaza alte incrementări. Așadar numărul de incrementări ar fi fost 6, chiar dacă șirul dat era deja crescător.