1. "ALTERNĒJOŠA VIRKNE"

Uzdevumu var atrisināt dažādos veidos, taču visu vairāk vai mazāk efektīvu risinājumu pamatā ir viena ideja: virzīties pa doto skaitļu virkni, skaitot kaut kādus nepieciešamos raksturlielumus. Vieglākais un drošākais ceļš varētu būt – stingri ievērot uzdevuma tekstā doto definīciju. Konkrēti – katram dotās virknes elementam A[i], izņemot pirmo un pēdējo, pārbaudām nosacījumus A[i-1]<A[i]>A[i+1] un A[i-1]>A[i]<A[i+1]. Ja viens no tiem izpildās, tad uzskatām virknes elementu par "labu". Uzdevums kļūst šāds: atrast garumu visgarākajam fragmentam no pēc kārtas esošiem labiem elementiem. Šo procesu vieglāk aprakstīt ar pseidokodu:

```
tekošais_garums ← 0;
maksimālais_garums ← 0;
For i ← 2 To N-1
   If (A[i] ir labs elements)
    Then
      Palielināt tekošais_garums;
      If tekošais_garums>maksimālais_garums
       Then maksimālais_garums ← tekošais_garums;
    Else  // A[i] nav labs, labo elementu virkne ir aprāvusies
      tekošais_garums ← 0;
   End If
End For
```

Šeit `tekošais_garums` ir pēc kārtas esošu labo elementu fragmenta, kas beidzas ar aplūkojamo i-to elementu, garums; `maksimālais_garums` katrā brīdī ir visgarākā pēc kārtas esošu labo elementu fragmenta garums.

Beigās tikai jāizvada vērtība `(maksimālais_garums + 2)`, ja vien `maksimālais_garums > 0`. Turpretī ja `maksimālais_garums = 0`, tad alternējošu fragmentu dotajā virknē nav, un jāizvada skaitlis 0.

Ievērosim, ka vienlaicīgi visus dotos skaitļus glabāt nav nepieciešams – pilnīgi pietiek glabāt tikai 3 skaitļus: A[i-1], A[i] un A[i+1].

Protams, pārbaudi

```
If tekošais_garums>maksimālais_garums Then maksimālais_garums ←
tekošais_garums;
```

ir nepieciešams veikt nevis katrā solī, bet tikai tad, kad izrādās, ka kārtējais pārbaudāmais elements nav labs. Nedrīkst aizmirst veikt šo pārbaudi arī cikla beigās – citādi mēs riskējam pazaudēt alternējošu fragmentu, kas beidzas ar pēdējo dotās virknes elementu. Rezultātā iegūstam šādu pseidokodu:

```
tekošais_garums ← 0;
maksimālais_garums ← 0;
For i ← 2 To N-1
   If (A[i] ir labs elements)
    Then Palielināt tekošais_garums;
    Else  // A[i] nav labs, labo elementu virkne ir aprāvusies
      If tekošais_garums>maksimālais_garums
       Then maksimālais_garums ← tekošais_garums;
      tekošais_garums ← 0;
   End If
End For
If tekošais_garums>maksimālais_garums
 Then maksimālais_garums ← tekošais_garums;
```

2. "CIPARU STARPĪBAS"

Ko mēs gribam? Mēs gribam izveidot lielāko iespējamo skaitli A. Tāpēc sāksim skaitli ar ciparu 9 un pēc tam pievienosim kārtējos ciparus tā, lai sanāktu dotie b_1, b_2 utt. Nē, nesanāk – skaidrs, ka pēc 9 seko mazāks cipars (vai vēl viens 9), un šo ciparu viegli noteikt viennozīmīgi, taču kādu ciparu likt, teiksim, aiz cipara 5, ja nākamais cipars no tā atšķiras, teiksim, par 3? Vai izvēlēties lielāko no iespējamiem cipariem? Protams, ar šādu pieeju sanāks lielāks skaitlis A. Ja, protams, sanāks... Samērā ātri kļūst skaidrs, ka tā mēs varam nonākt strupceļā – no kārtējā cipara nav iespējams virzīties ne uz augšu – pie lielāka cipara, ne arī uz leju – pie mazāka. Varbūt mēs nepareizi izvēlējāmies virzienu kādam no iepriekšējiem cipariem? Varbūt... Bet kuram? Iet atpakaļ un sākt visu no sākuma? Sanāk kaut kāda traka pārlase. Starp citu, varbūt vajadzēja sākt nevis ar ciparu 9, bet ar kādu citu? Pilnīgi iespējams – piemēram, ja B=57. Viegli redzēt, ka ar 9 šajā gadījumā sākt nedrīkst – tad šāds skaitlis A neeksistē. Tai pašā laikā der, piemēram, skaitlis A=381.

Sarežģīti nudien. Tomēr kaut kas sāk noskaidroties. Var gadīties tā, ka, sākot skaitli A ar ciparu 9, mums būs nepieciešamība kaut kad vēlāk, K-tajā vietā, likt tādu ciparu a_K, kas tur nevar atrasties – b_K, b_{K+1}, ... nedod iespēju atrast tādus ciparus a_{K+1}, a_{K+2}, ..., kas nebūtu pretrunā ar ciparu a_K. Kāds cipars vispār var būt K-tajā vietā? Skaidrs, ka jebkurš cipars nederēs – tam var traucēt b_K, b_{K+1}, ..., t.i., nosakot ciparu a_K, jārūpējas par cipariem, kas atrodas pa labi no tā. Bet, nosakot ciparus, kas atrodas pa labi no a_K, jārūpējas par cipariem, kas atrodas vēl vairāk pa labi utt. Kad gan tas beigsies? Pareizi – beigās. Ideja! Noskaidrosim, kādi cipari vispār var atrasties katrā vietā un jau pēc tam izvēlēsimies lielākos. Šo noskaidrošanu sāksim no labā (!) gala.

Tātad, sākam – no labā gala. Pēdējā, (N+1)-ajā vietā var atrasties jebkurš cipars.

Pamati ielikti. ☺ Ko tālāk? Vai K-tajā vietā var atrasties cipars D? Protams, var, ja vien izpildās vismaz viens no diviem nosacījumiem:

$D - b_K >= 0$ un (K+1)-ajā vietā drīkst atrasties cipars $(D - b_k)$, vai arī

$D + b_K <= 9$ un (K+1)-ajā vietā drīkst atrasties cipars $(D + b_k)$.

Izveidosim masīvu C[1..N+1, 0..9]: elementā C[K, D] glabāsim 1, ja K-tajā vietā drīkst atrasties cipars D, un 0, ja nedrīkst (vai True un False, vai kā citādi – kā labāk patīk). Masīvu C viegli aizpildīt, ejot no labās uz kreiso pusi – no N-tās līdz pirmajai vietai. (N+1)-ajā vietā drīkst atrasties jebkurš cipars, t.i., C[N+1, D]=1 visiem D no 0 līdz 9.

Tagad, kad aizpildīts masīvs C, mēs varam sākt veidot lielāko iespējamo A. Vispirms izvēlēsimies ciparu a_1. Tas ir vislielākais cipars D, kuram C[1, D]=1. Bet tālāk ir pavisam vienkārši – secīgi atrodam ciparus a_2, a_3, ..., a_{N+1} šādi: apzīmēsim $a_K + b_K$ ar D, tad – ja C[K+1, D]=1, tad izvēlamies kārtējo ciparu a_{K+1} vienādu ar D, citādi $a_{K+1} = a_K - b_K$ (ievērosim: ja C[K+1, D]=0, tad C[K+1, $a_K - b_K$] noteikti vienāds ar 1, jo kā gan citādi radās vienīgais elementā C[K, a_K]?).

3. "ALTERNĒJOŠA VIRKNE – 2"

Pieņemsim, ka jau esam sadalījuši doto N ciparu virkni $a_1a_2a_3...a_N$ par skaitļiem n_1, n_2, ..., n_K, kas veido alternējošu virkni ar lielāko iespējamo garumu K.

Aplūkosim kādu šīs virknes skaitli n_M, kas mazāks par abiem tā kaimiņiem (vai vienīgo kaimiņu, ja M=1 vai M=K). Pieņemsim, ka skaitlis n_M satur vairāk par vienu ciparu. Tad mēs varam izmainīt virkni šādi: visus skaitļa n_M ciparus, izņemot pēdējo, piekabināsim skaitļa n_{M-1} beigās, bet, ja M=1, tad visus skaitļa n_M ciparus, izņemot pirmo, piekabināsim skaitļa n_2 sākumā. Rezultātā skaitlis n_M samazināsies, bet cits skaitlis palielināsies. Viegli redzēt, ka virkne paliks alternējoša, un tās garums nemainīsies. Atkārtojot šādu operāciju citiem virknes skaitļiem, iegūsim alternējošu virkni ar lielāko iespējamo garumu, kurā visi skaitļi, kas mazāki par saviem kaimiņiem, ir viencipara.

Pieņemsim, ka iegūtajā skaitļu virknē ir kāds piecciparu skaitlis. Šis skaitlis ir lielāks par saviem kaimiņiem, jo visi skaitļi, kas mazāki par saviem kaimiņiem, tika samazināti līdz viencipara skaitļiem. Sadalīsim šo piecciparu skaitli par trim skaitļiem: divciparu, viencipara un vēl vienu divciparu skaitli. Viegli redzēt, ka esam ieguvuši jaunu alternējošu virkni (jebkurš divciparu skaitlis ir lielāks par jebkuru viencipara skaitli, jo nuļļu dotajā ciparu virknē nav), turklāt tās garums ir par 2 lielāks nekā sākotnējās virknes garums. Tas ir pretrunā ar to, ka dotā alternējošā virkne bija garākā iespējamā. Tātad, iegūtajā alternējošā virknē piecciparu skaitļu nav. Saprotams, tajā nav arī garāku skaitļu – tādus skaitļus var tieši tādā pašā veidā sadalīt par trim skaitļiem, saglabājot virkni alternējošu.

Rezultātā nonākam pie secinājuma: starp visām alternējošām virknēm ar lielāko iespējamo garumu eksistē tāda, kurā

- visi skaitļi, kas mazāki par saviem kaimiņiem, ir viencipara un
- katrs skaitlis satur ne vairāk par 4 cipariem.

Tagad risinājuma metodi var formulēt pavisam īsi – dinamiskā programmēšana. Tomēr aplūkosim risinājumu sīkāk.

Virzīsimies ciparu virknē no kreisās uz labo pusi, apstrādājot pa vienam ciparam.

Apzīmēsim ar L_P garumu tādai visgarākajai alternējošai virknei, kas veidota no pirmajiem P cipariem, kurā pēdējais skaitlis sastāv no viena cipara un ir mazāks par iepriekšējo virknes skaitli.

Apzīmēsim ar $G_{P,1}$, $G_{P,2}$, $G_{P,3}$ un $G_{P,4}$ garumu visgarākajai alternējošai virknei, kas veidota no pirmajiem P cipariem, kurā pēdējais skaitlis ir lielāks par iepriekšējo un attiecīgi sastāv no viena, diviem, trim un četriem cipariem.

Viegli redzēt, ka lielumiem L_P, $G_{P,1}$, $G_{P,2}$, $G_{P,3}$ un $G_{P,4}$ izpildās šādas rekurentas sakarības:

$$G_{P,1} = \begin{cases} L_{P-1} + 1, & ja \quad a_P > a_{P-1} \\ -\infty, & ja \quad a_P \le a_{P-1} \end{cases}$$

$$G_{P,2} = L_{P-2} + 1$$

$$G_{P,3} = L_{P-3} + 1$$

$$G_{P,4} = L_{P-4} + 1$$

$$L_P = \begin{cases} \max(G_{P-1,1}, G_{P-1,2}, G_{P-1,3}, G_{P-1,4}) + 1, & ja \quad a_P < a_{P-1} \\ \max(G_{P-1,2}, G_{P-1,3}, G_{P-1,4}) + 1, & ja \quad a_P \ge a_{P-1} \end{cases}$$

Sākumā inicializējam

$L_1=G_{1,1}=1$, $G_{1,2}=G_{1,3}=G_{1,4}=-\infty$, un

$L_P=0$, ja P<=0.

Uzdevuma atrisinājums ir lielākā no vērtībām L_N, $G_{N,1}$, $G_{N,2}$, $G_{N,3}$ un $G_{N,4}$.

Algoritma analīze. Katru no skaitļiem L_P, $G_{P,1}$, $G_{P,2}$, $G_{P,3}$ un $G_{P,4}$ (P=1, 2, …, N) var izrēķināt ar konstantu operāciju skaitu. Tātad algoritma sarežģītība ir O(N).

Piezīme. Nav grūti pierādīt, ka starp visām alternējošām virknēm ar lielāko iespējamo garumu eksistē tāda, kurā

- visi skaitļi, kas mazāki par saviem kaimiņiem, ir viencipara un
- katrs skaitlis satur ne vairāk par 3 cipariem.

Šis apsvērums ļauj nedaudz paātrināt programmas izpildi, lai gan augstāk izklāstītais risinājums strādā pietiekoši ātri.

Turklāt, pēdējā apgalvojumā var aizvietot skaitli 3 ar 2, taču šī fakta pierādījums ir samērā apjomīgs.

4. "VARŽU CIRKS"

Atmetīsim uzreiz gadījumu, kad M<=N, – tad gājienus izdarīt nav izdevīgi un jāizvada skaitlis 0.

Aplūkosim uzdevumu kā grafu teorijas uzdevumu: visi iespējamie varžu novietojumi veido grafa virsotņu kopu; no šī grafa virsotnes V uz virsotni U iet orientēta šķautne tad un tikai tad, ja no varžu novietojuma, kas atbilst virsotnei V, var ar vienu gājienu iegūt varžu novietojumu, kas atbilst virsotnei U; katrai šķautnei pierakstīts tās garums, kas vienāds ar N, ja šķautne atbilst parastam gājienam, un N-M, ja tā atbilst maģiskajam gājienam. Citiem vārdiem, šķautnes garums ir atbilstošā gājiena izdarīšanas **izmaksas**. Ievērosim, ka maģiskas šķautnes garums ir negatīvs (jo M>N) – parasta lieta, ar grafiem gadās ne tas vien.

Sākotnējais varžu novietojums atbilst kādai virsotnei W.

Tagad uzdevumu varam pārformulēt tā: pieņemsim, ka katrai virsotnei V ir zināms īsākā ceļa kopgarums no W uz V. Nepieciešams atrast vismazāko no šiem garumiem.

Uzreiz ievērosim, ka izveidotajā grafā nav ciklu ar negatīvu šķautņu kopgarumu. Patiešām – ja kāda ceļa kopgarums ir negatīvs, tad tas satur vismaz vienu maģisku šķautni. Katra maģiska šķautne samazina laukumā esošo varžu kopskaitu, bet palielināt varžu skaitu nav iespējams – tādu gājienu vienkārši nav. Līdz ar to aplūkojamais ceļš nevar beigties virsotnē, kurā tas ir sācies. Tas savukārt nozīmē, ka, ja kāda virsotne V ir sasniedzama no virsotnes W, tad eksistē arī īsākais ceļš no W uz V (pierādījums pielikumā).

Tā vietā, lai no virsotnes W meklētu īsākos ceļus uz citām virsotnēm, meklēsim ceļus, kas satur vismazāk šķautņu. Pierādīsim, ka jebkura šāda ceļa kopgarums būs mazākais iespējamais. Aplūkosim jebkuru vienu no šiem ceļiem. Pieņemsim, ka tas ved uz kādu virsotni V un satur P parastas šķautnes un Q maģiskas. Tas nozīmē, ka varžu skaits pozīcijā, kas atbilst grafa virsotnei V, ir par Q mazāks nekā sākotnējais varžu skaits (virsotnei W). Tātad **visi** ceļi no virsotnes W uz V satur **tieši Q** maģiskas šķautnes (katra no tām samazina varžu skaitu par 1). Tā kā aplūkojamais ceļš satur vismazāk šķautņu, tad vērtība P ir vismazākā iespējamā – no virsotnes W uz V neeksistē ceļš, kas satur mazāk par P parastām šķautnēm. Tā kā visu parastu šķautņu garumi ir pozitīvi (tie vienādi ar N), tad ceļa kopgarums būs jo mazāks, jo mazāk tas saturēs parastas šķautnes. Bet aplūkojamais ceļš satur vismazāk parasto šķautņu, līdz ar to tā kopgarums ir vismazākais iespējamais.

Ideja ir skaidra – no sākotnējās virsotnes W meklējam tādus ceļus uz visām citām virsotnēm, kas satur vismazāk šķautņu. Katrai virsotnei pierakstām arī atrastā īsākā ceļa kopgarumu. Beigās jāizvada vismazākā no pierakstītajām vērtībām, bet **ar mīnus zīmi**.

Šos ceļus var atrast, izmantojot meklēšanu plašumā: sākam ar virsotni W un atrodam visas virsotnes, kas no tās ir sasniedzamas ar vienu gājienu. Pieņemsim, ka šīs virsotnes veido kopu S_1. Tālāk atrodam visas līdz šim nesasniegtās virsotnes, kas no kopas S_1 ir sasniedzamas ar vienu gājienu, iegūstot kopu S_2. Līdzīgi iegūstam kopas S_3, S_4 utt. Tā turpinām, kamēr nav iespējams sasniegt jaunu virsotni, t.i., kārtējā iegūtā kopa ir tukša.

Risinājuma pseidokods:

1. Inicializācija.

D(W)=0, D(V)=+∞ visām virsotnēm V≠W, A = {W}, B = {}.

2. Cikls.

```
While (saraksts A nav tukšs) Do
{
   U – saraksta A pirmā virsotne;
   Pievienojam virsotni U apstrādāto virsotņu kopai B;
   Katrai neapstrādātai virsotnei V (kas nav kopā B), uz kuru iet šķautne no U, Do
   {
      D(V) = D(U) + s[U,V]
      Pievienojam virsotni V saraksta A beigās
   }
   Izmetam virsotni U no saraksta A
}
```

3. Noslēgums.

Izvadām - $\min_{V} D(V)$.

Atliek precizēt realizācijas momentus. Laukuma izmērs ir 16 rūtiņas, un katrā no tām var atrasties viena varde (bet var arī neatrasties), t.i., jebkura pozīcija var tikt kodēta ar 16 bitiem (0 – tukša rūtiņa, 1 – aizņemta rūtiņa) jeb – ar vienu 16-bitu skaitli. Tā arī darīsim – katru pozīciju kodēsim ar vienu 16-bitu skaitli bez zīmes. Kopējais iespējamo pozīciju skaits nepārsniedz 2^{16} – tas nav daudz, tāpēc ieviesīsim masīvu D(V) glabāšanai (pozīcijai V atbilst elements ar tās 16-bitu kodu). Saraksta A glabāšanai arī var izmantot masīvu ar izmēru 2^{16}. Kopu B vislabāk glabāt *boolean* tipa masīvā ar izmēru 2^{16}: B(V)=True, ja virsotne V ir apstrādāta un False, ja nav. Sākumā visu masīvu inicializējam ar False.

Algoritma analīze. Tā kā katra grafa virsotne var būt tikai vienā kopā S_i, tad katra grafa šķautne tiks aplūkota ne vairāk kā **vienu reizi**. Līdz ar to algoritma sarežģītība ir proporcionāla grafa šķautņu skaitam, kas savukārt nepārsniedz 2^{23}, jo no katras virsotnes (kuru kopskaits ir 2^{16}) katra no ne vairāk kā 16 vardēm var veikt ne vairāk kā 8 gājienus un $2^{16}*16*8=2^{23}$.

Pielikums.

Apgalvojums. Ja grafā nav ciklu ar negatīvu kopgarumu, tad starp katrām divām tā virsotnēm vai nu neeksistē ceļš, vai arī eksistē ceļš ar vismazāko kopgarumu.

Pierādījums. Apzīmēsim ar N grafa virsotņu skaitu. Jebkurš ceļš, kas satur vismaz N šķautnes, noteikti kādā virsotnē ieiet vairākkārt, t.i., satur ciklu. Visu ciklu garumi ir nenegatīvi, tāpēc šo ciklu var izmest no ceļa, un ceļa kopgarums no tā nepalielināsies. Tātad, īsākais ceļš, ja vien tāds eksistē, satur ne vairāk par N-1 šķautni, un tādu ceļu ir galīgs skaits. Tas garantē īsākā ceļa eksistenci starp jebkurām divām virsotnēm, starp kurām vispār eksistē vismaz viens ceļš.

5. "KARTĪTES"

Uzreiz atmetīsim gadījumu N=K – tad vienkārši jāizvada visi dotie skaitļi. Turpmāk uzskatīsim, ka N>K.

Sakārtosim visus uz kartītēm uzrakstītos skaitļus to absolūto vērtību dilšanas secībā: $|a_1| \geq |a_2| \geq |a_3| \geq \ldots \geq |a_N|$.

Iespējami vairāki varianti:

1. Starp pirmajām K kartītēm eksistē kartīte ar skaitli 0 (vai, kas ir tas pats, a_K=0). Tas nozīmē, ka starp dotajiem skaitļiem ir mazāk nekā K nenulles skaitļu, t.i., jebkurš K skaitļu komplekts satur 0. Tādējādi, lielākais iespējamais reizinājums vienāds ar 0 un izvaddatu failā var izvadīt jebkurus K no dotajiem skaitļiem, piemēram, a_1, a_2, … , a_K.

2. Starp pirmajām K kartītēm nulles nav, t.i., $a_K \neq 0$. Šādā gadījumā atradīsim negatīvo skaitļu skaitu starp skaitļiem a_1, a_2, ..., a_K.

2.1. Starp skaitļiem a_1, a_2, … , a_K ir pāra skaits negatīvu skaitļu. Skaidrs, ka šajā gadījumā jāizvada skaitļi a_1, a_2, … , a_K.

2.2. Starp skaitļiem a_1, a_2, … , a_K ir nepāra skaits negatīvu skaitļu.

2.2.1. Visi skaitļi a_1, a_2, … , a_K ir negatīvi.

2.2.1.1. Starp skaitļiem a_{K+1}, a_{K+2}, … , a_N ir vismaz viens nenegatīvs skaitlis. Apzīmēsim ar A pirmo (ar vismazāko numuru) nenegatīvo skaitli starp skaitļiem a_{K+1}, a_{K+2}, … , a_N. Tad jāizvada skaitļi a_1, a_2, … , a_{K-1}, A.

2.2.1.2. Visi skaitļi a_{K+1}, a_{K+2}, … , a_N ir negatīvi. Tas nozīmē, ka visi uz kartītēm uzrakstītie skaitļi ir negatīvi un K ir nepāra skaitlis. Jebkuru K skaitļu reizinājums būs negatīvs, t.i., mums jāizvada skaitļi, kuru reizinājums pēc absolūtās vērtības ir mazākais iespējamais. Citiem vārdiem, jāizvada skaitļi a_{N-K+1}, a_{K+2}, … , a_N.

2.2.2. Starp skaitļiem a_1, a_2, … , a_K ir vismaz viens pozitīvs skaitlis. Apzīmēsim ar B_1 pēdējo (ar vislielāko numuru) pozitīvo skaitli starp skaitļiem a_1, a_2, … , a_K, bet ar C_1 – pēdējo (ar vislielāko numuru) negatīvo skaitli starp skaitļiem a_1, a_2, … , a_K.

2.2.2.1. Visi skaitļi a_{K+1}, a_{K+2}, … , a_N ir negatīvi. Izvadām skaitļus a_1, a_2, … , a_K, iepriekš nomainot šajā rindā skaitli B_1 pret a_{K+1}.

2.2.2.2. Visi skaitļi a_{K+1}, a_{K+2}, … , a_N ir nenegatīvi. Izvadām skaitļus a_1, a_2, … , a_K, iepriekš nomainot šajā rindā skaitli C_1 pret a_{K+1}.

2.2.2.3. Starp skaitļiem a_{K+1}, a_{K+2}, … , a_N ir gan negatīvi, gan nenegatīvi skaitļi. Apzīmēsim ar B_2 pirmo (ar vismazāko numuru) nenegatīvo skaitli starp skaitļiem a_{K+1}, a_{K+2}, … , a_N, bet ar C_2 – pirmo (ar vismazāko numuru) negatīvo skaitli starp skaitļiem a_{K+1}, a_{K+2}, … , a_N. Tad rindā a_1, a_2, … , a_K ir jāsamaina vai nu B_1 pret C_2, vai arī C_1 pret B_2. Jebkurā gadījumā izvēlēto skaitļu reizinājums kļūst pozitīvs, taču pirmajā gadījumā tā absolūtā vērtība mainās $|C_2/B_1|$ reizes, bet otrajā – $|B_2/C_1|$ reizes. Protams, mūs interesē tas variants, kurā šī attiecība ir lielāka. Ievērosim, ka nosacījums $|C_2/B_1| > |B_2/C_1|$ ir ekvivalents nosacījumam $|C_2 \cdot C_1| > |B_2 \cdot B_1|$ vai, ņemot vērā reizinātāju zīmes, $C_2 \cdot C_1 > B_2 \cdot B_1$. Tā, iegūstam:

2.2.2.3.1. Ja $C_2 \cdot C_1 > B_2 \cdot B_1$, tad izvadām skaitļus a_1, a_2, … , a_K, iepriekš nomainot šajā rindā skaitli B_1 pret C_2.

2.2.2.3.2. Ja $C_2 \cdot C_1 \leq B_2 \cdot B_1$, tad izvadām skaitļus a_1, a_2, … , a_K, iepriekš nomainot šajā rindā skaitli C_1 pret B_2.

Algoritma analīze. Algoritma sarežģītību nosaka skaitļu kārtošanas sarežģītība risinājuma sākumā. Visas pārējās operācijas kopā prasa sliktākajā gadījumā vairākas reizes "izskriet" dotajam skaitļu masīvam, un to sarežģītība ir O(N).