70. ročník Matematické olympiády – 2020/2021

Úlohy krajského kola kategorie P

Krajské kolo 70. ročníku MO kategorie P se koná v úterý 19. 1. 2021 v dopoledních hodinách. Na řešení úloh máte 4 hodiny čistého času. V krajském kole MO-P se neřeší žádná praktická úloha, pro zajištění rovných podmínek řešitelů ve všech krajích je použití počítačů při soutěži zakázáno (s výjimkou nahrávání řešení do odevzdávacího systému). I v případě distanční formy soutěže ve vašem kraji je tedy zakázáno psát řešení přímo na počítači. Zakázány jsou rovněž jakékoliv další pomůcky kromě psacích potřeb (např. knihy, výpisy programů, kalkulačky, mobilní telefony).

Řešení každé úlohy vypracujte na samostatný list papíru. V případě distanční organizace krajského kola poté řešení oskenujte či nafoťte a nejpozději do 20 minut po skončení soutěže nahrajte ve formátu PDF do odevzdávacího systému, který najdete na *https://mo.mff.cuni.cz/osmo/*.

Řešení každé úlohy musí obsahovat:

- **Popis řešení,** to znamená slovní popis principu zvoleného algoritmu, *argumenty zdůvodňující jeho správnost* (případně důkaz správnosti algoritmu), diskusi o efektivitě vašeho řešení (časová a paměťová složitost, v úloze P-II-4 též komunikační složitost). Slovní popis řešení musí být jasný a srozumitelný i bez nahlédnutí do samotného zápisu algoritmu (do programu). Není možné odkazovat se na vaše řešení úloh domácího kola, opravovatelé je nemusí mít k dispozici.
- **Zápis algoritmu.** Ve všech úlohách je třeba uvést zápis algoritmu v nějakém dostatečně srozumitelném pseudokódu (případně v programovacím jazyce Pascal, C/C++ nebo Python). Nemusíte detailně popisovat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů, vyhledávání v poli, třídění apod.

Za každou úlohu můžete získat maximálně 10 bodů. Hodnotí se nejen správnost řešení, ale také kvalita jeho popisu a efektivita zvoleného algoritmu. Algoritmy posuzujeme podle jejich časové složitosti, tzn. závislosti doby výpočtu na velikosti vstupních dat. Záleží přitom pouze na řádové rychlosti růstu této funkce. V zadání každé úlohy najdete přibližné limity na velikost vstupních dat. Efektivním vyřešením úlohy rozumíme to, že váš program spuštěný s takovými daty na současném běžném počítači dokončí výpočet během několika sekund.

Vzorová řešení úloh naleznete krátce po soutěži na webových stránkách olympiády *https://mo.mff.cuni.cz/*. Na stejném místě bude zveřejněn i seznam úspěšných řešitelů krajského kola a seznam řešitelů postupujících do ústředního kola.

P-II-1 Let na Mars

V roce 2021 konečně přijde chvíle, kdy Kocourkovská vesmírná agentura vyšle na Mars raketu s lidskou posádkou. Během několika předchozích let procházelo výcvikem n potenciálních kosmonautů (a kosmonautek), z nichž je potřeba na misi vybrat dva. Vybraná dvojice kosmonautů samozřejmě musí být kompatibilní (aby během letu nedocházelo k hádkám), ale zároveň je potřeba, aby byla dostatečně rozdílná (pokud by během mise nastaly nějaké problémy, je dobré mít více různých nápadů a schopností, jak je řešit).

Všichni kosmonauti prošli rozsáhlými psychotesty na jejichž základě dostali *profil,* tj. dvojici nezáporných celých čísel (x, y). Expertní skupina určila, že spolu mohou letět kosmonauti s profily (x, y) a (x', y') právě tehdy, když se obě souřadnice jejich profilů liší nejvýše o k a alespoň jedna souřadnice se liší právě o k. Jinak řešeno, $|x - x'| \leq k$, $|y - y'| \leq k$ a alespoň v jedné z těchto nerovností musí nastat rovnost. (To lze také říci tak, že (x, y) a (x', y') jsou v maximové metrice vzdálené právě k.)

Soutěžní úloha

Na vstupu dostanete přirozené číslo k a n dvojic nezáporných celých čísel (x_1, y_1), …, (x_n, y_n), kde (x_i, y_i) je profil kosmonauta číslo i. Vaším úkolem je zjistit počet dvojic i, j takových, že $1 \leq i < j \leq n$ a kosmonauti i a j spolu mohou letět na Mars.

Formálně řečeno, vaším úkolem je zjistit počet (neuspořádaných) dvojic i, j takových, že $|x_i - x_j| \leq k$ a $|y_i - y_j| \leq k$ a zároveň v alespoň jedné z těchto dvou nerovností nastane rovnost. Můžete předpokládat, že žádní dva kosmonauti nemají přesně stejné profily (tj. pokud $i \neq j$, pak $x_i \neq x_j$ nebo $y_i \neq y_j$).

Formát vstupu

Na prvním řádku dostanete dvě mezerou oddělená nezáporná celá čísla n a k. Na každém z dalších n řádků dostanete dvě mezerou oddělená nezáporná celá čísla x_i a y_i.

Formát výstupu

Na výstup vypište jedno číslo, totiž počet dvojic kosmonautů, kteří spolu na Mars mohou letět.

Příklady

Vstup:	*Výstup:*
3 3 0 0 3 3 1 3	2

Vyhovující dvojice jsou 1, 2 a 1, 3. U kosmonautů 2 a 3 jsou rozdíly obou souřadnic ostře menší než k.

Vstup:	*Výstup:*
5 1 0 0 0 1 1 0 1 1 100 100	6

Každá dvojice složená z prvních čtyř kosmonautů spolu může letět, pátý kosmonaut nemůže letět s nikým.

Vstup:	*Výstup:*
5 1 0 0 0 2 2 0 2 2 1 1	4

Vyhovující dvojice jsou právě ty, které obsahují kosmonauta s profilem $(1, 1)$.

Bodování

Ve všech vstupech platí $n \geq 2$, $1 \leq k \leq 10^9$ a $0 \leq x_i, y_i \leq M$ pro všechna $1 \leq i \leq n$. Následující tabulka popisuje, kolik bodů můžete získat za řešení, která budou předpokládat příslušné horní meze na n a M:

body	n	M
10	10^5	10^9
6	10^5	1000
3	1000	10^9

P-II-2 Laboratorní protokoly

Říša propadá z fyziky. Jeho poslední šance, jak se propadnutí vyhnout, jsou laboratorní protokoly, které musí odevzdat nejpozději dnes večer. V průběhu školního roku jim paní učitelka dala za úkol naměřit n úloh (výsledkem každé z nich je celé číslo) a výslednou známku za protokoly dostanou podle kvadratické odchylky jejich hodnot od hodnot, které naměřila paní učitelka. Čím menší odchylka, tím lepší známka.

Aby si Říša zajistil postup do dalšího ročníku, rozhodl se, že hackne paní učitelce počítač, zjistí, jaké naměřila hodnoty, a podle toho upraví ty své. Aby zůstal nenápadný, rozhodl se, že je upraví jen trochu. To konkrétně znamená, že si předem zvolil celé číslo k a nejvýše k-krát si vybere nějakou svou hodnotu a tu zvýší nebo sníží o 1.

Soutěžní úloha

Na vstupu dostanete dvě celočíselné posloupnosti A a B délky n a přirozené číslo k. Vaším úkolem je nejvýše k-krát vybrat nějaký prvek z B a pokaždé k němu přičíst nebo odečíst jedničku tak, aby po těchto úpravách byl výraz

$$\sum_{i=1}^{n}(A_i - B_i)^2$$

co nejmenší. Stejný prvek posloupnosti můžete vybrat i vícekrát a pokaždé se lze znovu rozhodnout, zda jedničku přičíst či odečíst. Můžete předpokládat, že se hodnota výrazu

$$\sum_{i=1}^{n}(A_i - B_i)^2$$

vejde do proměnné a lze s ní pracovat v konstantním čase.

Zápis se sumou je definován jako

$$\sum_{i=1}^{n}(A_i - B_i)^2 = (A_1 - B_1)^2 + (A_2 - B_2)^2 + \cdots + (A_n - B_n)^2.$$

Formát vstupu

Na prvním řádku dostanete dvě přirozená čísla n, k. Na druhém řádku je n celých čísel $A_1, \ldots, A_n$ a na třetím řádku je n celých čísel $B_1, \ldots, B_n$.

Formát výstupu

Na výstup vypište jedno celé číslo, totiž nejmenší možnou hodnotu výrazu $\sum_{i=1}^{n}(A_i - B_i)^2$, kterou může Říša získat.

Příklady

Vstup:	*Výstup:*
1 9 10 5	0

V tomto vstupu Říša naměřil 5*, zatímco paní učitelka naměřila* 10*. Protože* $k = 9$*, může Říša například pětkrát zvýšit svou hodnotu o* 1 *a pak už nedělat nic, čímž dostane odchylku* 0*. Kdyby ale chtěl, může také svou hodnotu například dvakrát snížit a pak sedmkrát zvýšit, což dá opět odchylku* 0*.*

Vstup:	*Výstup:*
1 5 -10 0	25

Zde je pouze jeden způsob, jak minimalizovat odchylku, totiž pětkrát odečíst jedna od své hodnoty ($0 - 1 - 1 - 1 - 1 - 1 = -5$*), čímž Říša dosáhne odchylky* $(-10 - (-5))^2 = 25$*.*

Vstup:	*Výstup:*
2 5 0 0 10 -2	29

Po úpravách budou Říšovy hodnoty $(5, -2)$*.*

Bodování

Ve všech vstupech platí $n \geq 1$ a $k \geq 0$. Následující tabulka popisuje, kolik bodů můžete získat za řešení, která budou předpokládat příslušné horní meze na n a k:

body	n	k
10	10^5	10^9
7	10^5	10^5
4	1000	1000

Až dva body získáte za správné řešení s libovolnou časovou složitostí. Nezapomeňte na důkaz správnosti!

P-II-3 Vánoční návštěvy

Manželé Novákovi jezdí každý rok na Vánoce na chatu a paní Nováková trvá na tom, že navštíví všechny příbuzné, skrz jejichž vesnici cestou pojedou. Z toho není příliš nadšený pan Novák, jelikož na každé návštěvě musí ochutnat domácí bramborový salát a pak je mu celý zbytek Vánoc zle. Po letošních Vánocích se rozhodl, že se pokusí příští rok naplánovat cestu bezpečně, ale obává se, zda to bude možné.

Pan Novák dělí příbuzné na salámisty a nesalámisty podle toho, zda do salátu dávají salám. Podle pana Nováka salám do salátu samozřejmě nepatří, a zatímco nesalámových salátů snese docela hodně (konkrétně k, potom potřebuje alespoň jednu vesnici bez příbuzných, aby mu slehlo, a pak zase může hodovat), jakmile do sebe jednou musí dostat salám, už nechce žádné jídlo ani vidět.

Aby situace nebyla tak jednoduchá, tetička Gertruda se občas rozhodne a vyrazí na návštěvu k jedněm salámistům. Pan Novák v tuhle chvíli neví, zda na návštěvu pojede, případně kam, ale ví naprosto jistě, že tetičku Gertrudu nechce potkat. Pomůžete mu zjistit, zda má příští rok jistotu, že stráví Vánoce bez žaludečních potíží?

Soutěžní úloha

Na vstupu dostanete n vesnic, z nichž některé dvojice jsou spojené obousměrnými silnicemi. Vesnice jsou očíslované čísly $1, \ldots, n$, Novákovi bydlí ve vesnici číslo 1 a chatu mají ve vesnici číslo n. U každé vesnice navíc dostanete informaci, zda v ní bydlí salámisté, nesalámisté, anebo žádní příbuzní (pro každou vesnici nastane jedna z těchto tří možností a můžete předpokládat, že ve vesnicích 1 a n nebydlí žádní příbuzní). To, ke kterým salámistům přijede tetička Gertruda na návštěvu, pan Novák v tuhle chvíli neví, ale dozví se to před tím, než bude plánovat, jak na chatu pojede.

Pomozte panu Novákovi zjistit, zda bez ohledu na to, kam tetička Gertruda nakonec zamíří, bude pan Novák schopný naplánovat, kudy jet na chatu, aby splnil následující tři podmínky:

- Jakmile Novákovi navštíví salámisty, tak dále už mohou jet pouze vesnicemi, kde žádní příbuzní nebydlí.
- Na cestě nesmí být bezprostředně za sebou více než k vesnic, kde bydlí nějací příbuzní.
- Musí se vyhnout vesnici, kde je tetička Gertruda.

Je možné, že Novákovi některou vesnici navštíví vícekrát. V takovém případě musí zajít na návštěvu pokaždé, když budou vesnicí projíždět. Pokud v žádné vesnici salámisté nebydlí, tetička nikam na návštěvu nepojede.

Formát vstupu

Na prvním řádku dostanete tři přirozená čísla, n, m a k. Na druhém řádku dostanete posloupnost $v_1, \ldots, v_n$ čísel z množiny $\{0, 1, 2\}$, kde $v_i = 0$, pokud ve vesnici číslo i nebydlí žádní příbuzní, $v_i = 1$, pokud ve vesnici číslo i bydlí nesalámisté a $v_i = 2$, pokud ve vesnici číslo i bydlí salámisté. Na každém z m dalších řádků

dostanete dvě čísla x_i a y_i značící, že mezi vesnicemi x_i a y_i vede (obousměrná) silnice.

Formát výstupu

Na výstup vypište `ANO`, pokud pan Novák vždy bude schopný naplánovat cestu, aby splňovala všechny tři podmínky, a `NE` v opačném případě.

Příklady

Vstup:	*Výstup:*
5 5 2 0 2 2 0 0 1 2 1 3 2 4 3 4 4 5	ANO

Pokud tetička Gertruda zamíří do vesnice 2, může pan Novák použít cestu $1-3-4-5$, *pokud zamíří do vesnice 3, může pan Novák použít cestu* $1-2-4-5$.

Vstup:	*Výstup:*
5 5 2 0 1 2 2 0 1 2 1 3 2 4 3 4 4 5	NE

Stejné vesnice a silnice jako v předchozím příkladu, ale jiní příbuzní. V tomto případě se může stát, že tetička Gertruda zamíří do vesnice 4 a tehdy se jí pan Novák nedokáže vyhnout.

Vstup:	*Výstup:*
5 4 2 0 1 1 1 0 1 2 2 3 3 4 4 5	NE

Plán musí obsahovat projetí po sobě jdoucích vesnic 2-3-4, což nejde. V tomto vstupu nejsou žádní salámisté, a tudíž ani tetička Gertruda nemá kam přijet na návštěvu.

Vstup:
```
5 4 1
0 1 0 1 0
1 2
2 3
3 4
4 5
```

Výstup:
```
ANO
```

Stejné vesnice a silnice jako v předchozím případě, dokonce $k = 1$, ale mezi vesnicemi 2 a 4 vede cesta skrze vesnici 3, kde žádní příbuzní nebydlí, a proto salát panu Novákovi může trochu slehnout.

Vstup:
```
6 6 2
0 2 2 0 1 0
1 2
1 3
2 4
3 5
4 6
5 6
```

Výstup:
```
NE
```

Pokud půjde tetička Gertruda na návštěvu do vesnice 2, musel by pan Novák použít cestu $1 - 3 - 5 - 6$, aby se jí vyhnul, ale na ní by musel sníst salát poté, co snědl salát se salámem, což není povoleno.

Vstup:
```
6 5 2
0 1 1 0 1 0
1 2
2 3
3 4
3 5
5 6
```

Výstup:
```
ANO
```

Nejkratší cesta by vedla $1 - 2 - 3 - 5 - 6$. Na ní by ale Novákovi navštívili tři vesnice s nesalámisty za sebou, jenže pan Novák vydrží v kuse maximálně dvě. Proto musí jet $1 - 2 - 3 - 4 - 3 - 5 - 6$, tj. z vesnice 3 si zajet do vesnice 4, aby panu Novákovi slehlo, a pak se vrátit zpět do vesnice 3 a pokračovat na chatu.

Bodování

Až plný počet bodů můžete získat za řešení, které efektivně vyřeší všechny vstupy s $2 \leq n \leq 10^5$, $0 \leq m \leq 10^5$ a $1 \leq k \leq 10$. Až pět bodů lze získat za řešení, které navíc předpokládá, že salámisté bydlí v nejvýše 10 vesnicích. Až dva body můžete získat za libovolné funkční řešení.

Část bodů můžete získat také tehdy, když budete předpokládat, že všichni příbuzní jsou salámisté, nebo naopak že všichni příbuzní jsou nesalámisté.

P-II-4 Třídění na disku

Tato soutěžní úloha navazuje na úlohu z domácího kola. Za zadáním úlohy najdete studijní text, který je totožný se studijním textem ze zadání domácího kola. Podúlohy jsou nezávislé, můžete je řešit v libovolném pořadí.

V obou podúlohách budeme třídit posloupnost položek $x_1, \ldots, x_n$ uložených na disku. Předpokládejte, že položky mají jednotkovou velikost, tedy do jednoho bloku se jich vejde přesně B. Dvě položky dokážeme porovnat v konstantním čase.

Pro jednoduchost můžete předpokládat, že počet položek n je násobkem velikosti bloku B a že kromě vstupu je na disku neomezeně mnoho pracovního prostoru.

Podúloha A (3 body)

Třídění výběrem se říká následujícímu třídicímu algoritmu:

Algoritmus TŘÍDĚNÍVÝBĚREM
Vstup: Posloupnost $x_1, \ldots, x_n$
1. Pro k od 1 do n:
2. Prohodíme x_k s nejmenším z prvků $x_k, x_{k+1}, \ldots, x_n$.

Výstup: Setříděná posloupnost $x_1, \ldots, x_n$

Upravte tento algoritmus, aby fungoval pro data uložená na disku. Analyzujte jeho časovou složitost a I/O složitost.

Podúloha B (7 bodů)

Navrhněte efektivní algoritmus pro třídění na disku. Pokuste se dosáhnout co nejmenší komunikační složitosti.

Studijní text

V tomto ročníku olympiády se budeme zabývat počítáním s obrovským množstvím dat. Budeme zpracovávat vstupy, které se nám ani celé nevejdou do hlavní paměti počítače. Proto naše programy budou muset pracovat s daty uloženými na disku a vyrovnat se s tím, že disk je mnohem pomalejší než hlavní paměť. Budeme přitom uvažovat následující zjednodušený model disku.

Náš počítač má k dispozici *vnitřní a vnější paměť*. Pro zjednodušení budeme vnitřní paměti říkat prostě *paměť* a té vnější *disk*.

K datům ve (vnitřní) pamětí můžeme přistupovat přímo – tak, jak jsme zvyklí v běžných algoritmech. Tato paměť ale má jenom omezenou kapacitu: M položek, kde M je nějaký parametr, který určíme později. Do každé položky můžeme uložit jedno „rozumně velké“ číslo. Tím myslíme čísla, která nejsou řádově větší než čísla na vstupu.

Disk je neomezeně velký. Je rozdělený na *bloky* o velikosti B položek (další parametr). S těmito položkami ale nelze počítat přímo. Můžeme pouze načíst celý blok do paměti, anebo naopak B po sobě jdoucích položek paměti zapsat do jednoho bloku na disku. Čtení budeme v programech zapisovat jako volání funkce READ(b), která dostane pořadové číslo bloku b a vrátí obsah bloku. Podobně WRITE(b, *obsah*) zapíše blok na disk. Jak čtení, tak zápis trvají čas $\mathcal{O}(B)$.

Efektivitu algoritmů posuzujeme pomocí dvou druhů složitosti: *časové složitosti* výpočtů v paměti a *komunikační složitosti,* což je počet přenosů bloků (čtení a zápisů) mezi pamětí a diskem.

Budeme se snažit hledat algoritmy, které dosahují stejné časové složitosti jako na klasickém počítači, a k tomu mají co nejlepší komunikační složitost. Nebude-li řečeno jinak, naše algoritmy by měly fungovat pro libovolné hodnoty parametrů M a B. Můžeme přitom předpokládat, že $\mathcal{O}(B)$ položek se do paměti vejde. Speciálně nám tedy paměť vystačí na konstantní počet číselných proměnných. Ale pokud bychom vytvořili rekurzivní funkci, musíme si dát pozor na prostor potřebný na zásobník.

Příklad: Sekvenční průchod

Práci s diskem si vyzkoušíme na jednoduchém příkladu. Na disku dostaneme posloupnost N celých čísel. Čísla budou rozdělena do bloků: v prvním bloku prvních B čísel, pak dalších B atd. Celkem tedy vstup zabírá $\lceil N/B \rceil$ bloků. Naším úkolem bude najít mezi zadanými čísly to největší.

Půjdeme na to jednoduše: budeme postupně procházet bloky vstupu jeden po druhém. Každý blok načteme do paměti, projdeme všechny jeho prvky, přičemž si udržujeme průběžné maximum. Poté už blok nebudeme potřebovat, takže tutéž paměť můžeme využít na další blok. V pseudokódu by to vypadalo takto:

```
1. i ← 0                                  ◁ Aktuální pozice ve vstupu
2. m ← −∞                                 ◁ Průběžné maximum
3. Dokud i < N:                           ◁ Ještě něco zbývá
4.     b ← Read(i/B)                      ◁ Přečteme další blok vstupu
5.     n ← min(B, N − i)                  ◁ Kolik v něm je položek?
6.     Pro j od 0 do n − 1:               ◁ Projdeme položky
7.         m ← max(m, b[j])
8.     i ← i + z
9. Vypíšeme m.
```

Časová složitost tohoto algoritmu je jistě lineární s velikostí vstupu, tedy $\mathcal{O}(N)$. Jak je to s komunikační složitostí? Každý blok vstupu načteme právě jednou, takže přeneseme $\lceil N/B \rceil \leq N/B + 1$ bloků. To je jistě nejlepší možné, protože na každý blok se musíme alespoň jednou podívat.

Ještě musíme ověřit, kolik potřebujeme vnitřní paměti. Spotřebujeme konstantní počet položek na proměnné a B položek na právě zpracovávaný blok. To se určitě vejde do $\mathcal{O}(B)$, a tím pádem i do paměti.

Příklad: Test symetrie

Opět bude zadána nějaká posloupnost čísel. Tentokrát budeme chtít zjistit, zda je symetrická, tedy zde první prvek je roven poslednímu, druhý předposlednímu a tak dále.

Kdyby byla délka vstupu dělitelná B, stačilo by porovnat první blok s posledním (první má být zrcadlovým obrazem posledního), druhý s předposledním atd.

Přitom bychom každý blok přečetli právě jednou.

Pokud délka vstupu dává po dělení B nějaký nenulový zbytek $z = N \bmod B$, je to trochu složitější: zrcadlový obraz prvního bloku bude zčásti v z položkách posledního bloku a zčásti v $B - z$ položkách předposledního.

Představíme si proto, že spustíme současně dva programy: jeden bude číst vstup od začátku do konce, druhý od konce k začátku. Každý z nich si bude udržovat svůj aktuální blok a čas od času přečte nový. Pokaždé porovnáme aktuální položku přečtenou oběma programy.

První program se bude chovat stejně jako sekvenční čtení z minulého příkladu, druhý bude číst tytéž bloky pozpátku. Dohromady tedy přečtou nejvýše $2\lceil N/B \rceil \leq 2N/B + 2$ bloků. Dokonce se mohou zastavit po zpracování $N/2$ párů prvků, čímž se komunikační složitost zlepší na $N/B + 2$. Časová složitost je stále $\mathcal{O}(N)$.

Dodejme ještě, že zapisování komunikační složitosti pomocí $\mathcal{O}$-čkové notace je poněkud zrádné. Funkci $2N/B + 2$ totiž nelze jen tak zjednodušit na $\mathcal{O}(N/B)$ – pro funkce dvou (a více) proměnných už neplatí, že přičítanou konstantu můžeme jen tak „schovat do $\mathcal{O}$“.* Nejjednodušší možný zápis tedy je $\mathcal{O}(N/B + 1)$.

Příklad: Otočení posloupnosti

Pokračujeme v předchozím příkladu: co kdybychom chtěli posloupnost čísel otočit? Tedy přesunout první prvek na poslední místo, druhý na předposlední a tak dále.

Opět nejprve uvažujme případ, kdy je délka posloupnosti dělitelná velikostí bloku. Tehdy můžeme postupovat po blocích. Nejprve načteme do paměti první a poslední blok, otočíme obsah obou bloků a zapíšeme je zpět na disk v opačném pořadí. Pak provedeme totéž s druhým a předposledním blokem atd. A nakonec, má-li posloupnost lichý počet bloků, přečteme prostřední blok, otočíme ho a zapíšeme zpět.

Všimněte si, že takto každý blok jednou přečteme a jednou zapíšeme, tudíž komunikační složitost činí opět $\mathcal{O}(N/B)$.

V případě, kdy zbytek $N \bmod B$ není nulový, použijeme osvědčenou úvahu se současně běžícími programy, které si vyměňují prvky. Komunikační složitost pak bude $\mathcal{O}(N/B + 1)$.

* Notace $f(n) = \mathcal{O}(g(n))$ totiž znamená, že existuje konstanta c taková, že pro všechna n kromě konečně mnoha výjimek platí $f(n) \leq c \cdot g(n)$. U funkcí více proměnných definujeme $\mathcal{O}$ obdobně a povolíme konečně mnoho výjimek pro každou proměnnou. Jenže zlomek N/B může být libovolně malý i po zakázání konečně mnoha hodnot N i B, takže $+1$ nepřebijeme sebevětší konstantou c.