Примерни решения на задачите от семестриалното контролно по ДАА на КН2 & Инф., 13 април 2025 г.

Зад. 1

А) Сравнете по асимптотично нарастване функциите $f(n)$ и $g(n)$, дефинирани така:

$$f(n) = ((n-1)!)^{n!}, \qquad g(n) = (n!)^{(n-1)!}$$

Б) Сравнете по асимптотично нарастване функциите $h(n)$ и $t(n)$, дефинирани така:

$$h(n) = n^{n^n}, \qquad t(n) = (\lg n)^{(\lg n)^{(\lg n)^{\lg n}}}$$

Решение: Първо да сравним $f(n)$ с $g(n)$. Логаритмуваме двете функции:

$$\lg f(n) = \lg\left(((n-1)!)^{n!}\right) = n!\lg(n-1)! \asymp n!(n-1)\lg(n-1)$$
$$\lg g(n) = \lg\left((n!)^{(n-1)!}\right) = (n-1)!\lg n! \asymp (n-1)!n\lg n = n!\lg n$$

Но

$$\lim_{n\to\infty} \frac{n!(n-1)\lg(n-1)}{n!\lg n} = \lim_{n\to\infty} \frac{(n-1)}{\lg n} \cdot \lg(n-1) = \infty$$

понеже и двата множителя клонят към безкрайност. Заключаваме, че $f(n) \succ g(n)$.

Сега да сравним $h(n)$ с $t(n)$. Логаритмуваме двете функции:

$$\lg h(n) = \lg\left(n^{n^n}\right) = n^n \lg n$$
$$\lg t(n) = \lg\left((\lg n)^{(\lg n)^{(\lg n)^{\lg n}}}\right) = (\lg n)^{(\lg n)^{\lg n}} \lg\lg n$$

Логаритмуваме още веднъж:

$$\lg\lg h(n) = \lg\left(n^n\right) + \lg\lg n = n\lg n + \lg\lg n \asymp \underbrace{n\lg n}_{u(n)}$$
$$\lg\lg t(n) = \lg\left((\lg n)^{(\lg n)^{\lg n}}\right) + \lg\lg\lg n = (\lg n)^{\lg n}\lg\lg n + \lg\lg\lg n \asymp \underbrace{(\lg n)^{\lg n}\lg\lg n}_{v(n)}$$

Нека $u(n) = n\lg n$ и $v(n) = (\lg n)^{\lg n}\lg\lg n$. Ще докажем, че $u(n) \prec v(n)$. За целта пак логаритмуваме

$$\lg u(n) = \lg n + \lg\lg n \asymp \lg n$$
$$\lg v(n) = (\lg n)\lg\lg n + \lg\lg\lg n \asymp (\lg n)\lg\lg n$$

Наистина, $\lg n \prec (\lg n)\lg\lg n$, защото $\lg\lg n$ е неограничено растяща функция. Тогава $\lg u(n) \prec \lg v(n)$. Тогава $\lg\lg h(n) \prec \lg\lg t(n)$, откъдето следва, че $h(n) \prec t(n)$.

Зад. 2 За всеки масив $A[1\,..\,n]$ от естествени числа ще казваме, че A *е* k*-забележителен*, ако

- A съдържа точно k четни числа;
- нечетните числа в A са сортирани, тоест, ако $1 \le i < j \le n$ и $A[i]$ и $A[j]$ са нечетни, то $A[i] \le A[j]$.

Нека $A[1\,..\,n]$ е $\left\lceil \frac{n}{\lg n} \right\rceil$-забележителен масив. Опишете алгоритъм със сложност по време $O(n)$, който сортира A. Не е необходимо да пишете подробен псевдокод или да доказвате коректността с инвариант или по индукция. Достатъчно е да опишете идеята ясно и недвусмислено и да дадете съвсем кратка аргументация за коректността и сложността.

Решение: Следният алгоритъм решава задачата.

```
SomeSort(A[1 .. n]: масив от естествени числа, който е ⌈n/lg n⌉-забележителен.)
 1  k ← ⌈n/lg n⌉
 2  създай масиви B[1 .. k], C[1 .. n − k]
 3  b ← 1, c ← 1
 4  for i ← 1 to n
 5      if iseven(A[i])
 6          B[b] ← A[i]
 7          b++
 8      else
 9          C[c] ← A[i]
10          c++
11  Heapsort(B[1 .. k])
12  Merge(B[1 .. k], C[1 .. n − k], A[1 .. n])
13  return A
```

Алгоритъмът Merge на ред 12 е подобен на този, който разгледахме на лекции, с незначителни разлики. За този Merge: **ако** неговите входни масиви $B[1\,..\,k]$ и $C[1\,..\,n-k]$ са сортирани, **то** при терминирането му, масивът $A[1\,..\,n]$ съдържа техните елементи, в сортиран вид. Ерго, входният $A[1\,..\,n]$ се третира от Merge като празен масив, в който се записва резултатът от сливането на B и C.

Коректност. Ефектът от изпълнението на **for**-цикъла (редове 4–10) е, масивът $B[1\,..\,k]$ съдържа четните елементи на входа, и то в реда, в който те се намират във входа, а масивът $C[1\,..\,n-k]$ съдържа нечетните елементи на входа, и то в реда, в който те се намират във входа. Това е напълно очевидно предвид факта, че индексите b и c се инициализират с единици на ред 3.

По условие, нечетните елементи са сортирани във входа. Тогава масивът $C[1\,..\,n-k]$ е сортиран след изпълнението на **for**-цикъла. Знаем, че **Heapsort** е коректен сортиращ алгоритъм. Тогава масивът $B[1\,..\,k]$ е сортиран след изпълнението на ред 11.

Тогава, при викането на Merge на ред 12, както $B[1\,..\,k]$, така и $C[1\,..\,n-k]$ са сортирани. Заключаваме, че на ред 13, масивът A съдържа елементите на $B[1\,..\,k]$ и $C[1\,..\,n-k]$, в сортиран вид. Но елементите на $B[1\,..\,k]$ и $C[1\,..\,n-k]$ са точно елементите на входния A. Заключаваме, че на ред 13, алгоритъмът SomeSort връща масив, състоящ се от елементите на входния A, но в сортиран вид. □

Сложност. **for**-цикълът (редове 4–10) има сложност по време $\Theta(n)$. **Heapsort** на ред 11 работи във време $\Theta(k \lg k)$. Но това е

$$\Theta\left(\left\lceil \frac{n}{\lg n} \right\rceil \lg\left(\left\lceil \frac{n}{\lg n} \right\rceil\right)\right) = \Theta\left(\frac{n}{\lg n} \lg \frac{n}{\lg n}\right) = \Theta\left(\frac{n}{\lg n}(\lg n - \lg\lg n)\right) = \Theta\left(n - n\frac{\lg\lg n}{\lg n}\right) = \Theta(n)$$

тъй като $n > n\frac{\lg\lg n}{\lg n}$. Merge на ред 11 работи във време $\Theta((k) + (n-k))$, тоест, $\Theta(n)$.

Тогава сложността на целия алгоритъм е $\Theta(n) + \Theta(n) + \Theta(n) = \Theta(n)$. Имаме право да кажем, че сложността е $O(n)$.

Зад. 3 Разгледайте рекурентното уравнение

$$T(n) = 2T(n-1) + \frac{n}{(n+1)(n+2)}$$

Иска се да го решите по индукция.

Първо, направете смислено предположение за решението.

Второ, докажете Вашето предположение по индукция.

Решение: Предполагаме, че $T(n) = \Theta(2^n)$. Причината е, че дървото на рекурсията има размер $\Theta(2^n)$, което се вижда кристално ясно, ако се разгледаме само хомогенната част вдясно: $T'(n) = 2T'(n-1)$ има решение $T'(n) = \Theta(2^n)$ – или чрез метода с характеристичното уравнение, или с развиване. Нехомогенната част е намаляваща функция и интуитивно е ясно, че нейното добавяне не може да промени асимптотиката на решението. Дори нехомогенната част да беше константа, или някакво полиномиално нарастване n^k, пак решението щеше да е $T(n) = \Theta(2^n)$, защото такава нехомогенна част дава само единици в мултимножеството от корените (по отношение на метода с характеристичното уравнение), така че 2^n, което идва от корена 2 на характеристичното уравнение, доминира решението.

Ще докажем по индукция, че $T(n) = O(2^n)$. За целта ще докажем, че

$$\exists b, c > 0 \ \forall n \nearrow : T(n) \le c2^n - bn \tag{1}$$

Да допуснем, че

$$T(n-1) \le c2^{n-1} - b(n-1)$$

Тогава

$$\begin{aligned} T(n) &\le 2\left(c2^{n-1} - b(n-1)\right) + \frac{n}{(n+1)(n+2)} \\ &= c2^n - 2b(n-1) + \frac{n}{(n+1)(n+2)} \\ &= c2^n - 2bn + 2 + \frac{n}{(n+1)(n+2)} \\ &= c2^n - bn - bn + 2 + \frac{n}{(n+1)(n+2)} \\ &\le c2^n - bn, \text{ ако } -bn + 2 + \frac{n}{(n+1)(n+2)} \le 0 \end{aligned}$$

Но

$$-bn + 2 + \frac{n}{(n+1)(n+2)} \le 0 \ \leftrightarrow \ -bn(n+1)(n+2) + 2(n+1)(n+2) + n \le 0$$

Последното е вярно за всяко положително b, понеже $n(n+1)(n+2)$ има кубична асимптотика и доминира, в асимптотичния смисъл, събираемото $2(n+1)(n+2) + n$.

Заключаваме, че за всеки положителни b и c и за всички достатъчно големи n, (1) е в сила. ✓

Ще докажем по индукция, че $T(n) = \Omega(2^n)$. За целта ще докажем, че

$$\exists d > 0 \ \forall n \nearrow : T(n) \ge d2^n \tag{2}$$

Да допуснем, че

$$T(n-1) \ge d2^{n-1}$$

Тогава

$$\begin{aligned} T(n) &\ge 2d2^{n-1} + \frac{n}{(n+1)(n+2)} \\ &= d2^n + \frac{n}{(n+1)(n+2)} \\ &\ge d2^n, \text{ за всяко } d > 0 \end{aligned}$$

И така, за всяко $d > 0$, (2) е в сила. ✓

Зад. 4 Нека $n \geq 2$. Даден е масив $A[(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)]$ от наредени двойки от цели числа. Известно е, че

$$\forall i \in \{1, \ldots, n\} : (1 \leq x_i < n^2 \ \wedge \ 1 \leq y_i < n^2)$$

За всяко $i \in \{1, \ldots, n\}$ дефинираме, че $x_i + n^{y_i}$ е *степента на* (x_i, y_i). Опишете алгоритъм със сложност по време $O(n)$, който сортира масива по степените на елементите. Не е необходимо да пишете подробен псевдокод или да доказвате коректността с инвариант или по индукция. Достатъчно е да опишете идеята ясно и недвусмислено и да дадете съвсем кратка аргументация за коректността и сложността.

Внимание! Вашият алгоритъм може да извършва степенувания n^y, било в явен, било в неявен вид, **само** ако y е константа, **но не** и в общия случай.

Решение: Следният алгоритъм, описан само на идейно ниво, решава задачата.

Фаза 1 Създаваме два масива A_1 и A^* от наредени двойки. В A_1 слагаме всички наредени двойки (x, y) от A, такива че $y = 1$, а в A^* слагаме останалите наредени двойки от A.

Фаза 2 Сортираме A_1.

Фаза 3 Сортираме A^*.

Фаза 4 Сливаме A_1 и A^* в един сортиран масив и връщаме него.

Коректността е очевидна при това описание, ако сливането на двата сортирани масива във **Фаза 4** е коректно. Ще въведем няколко дефиниции, ще получим няколко теоретични резултата и ще се върнем на анализа на алгоритъма.

Нека pow е функцията-степенуване от условието, дефинирана по отношение на фиксирано n. Формално, $\text{pow} : \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$, като $\text{pow}(x, y) = x + n^y$.

Да дефинираме следната релация $\preccurlyeq$ върху наредените двойки:

$$(x_i, y_i) \preccurlyeq (x_j, y_j) \ \leftrightarrow \ \text{pow}(x_i, y_i) \leq \text{pow}(x_j, y_j)$$

Желаната сортировка е по отношение на $\preccurlyeq$. Не бъркайте $\preccurlyeq$ с $\preceq$, която е една от петте релации на асимптотично сравнение! Релацията $\preccurlyeq$ е рефлексивна и транзитивна. Дали е антисиметрична? Отговорът е, че в общия случай не е.

Лема 1. *Релацията $\preccurlyeq$ не е антисиметрична.*

Доказателство: Нека $n = 5$. Да разгледаме наредените двойки $p_1 = (24, 1)$ и $p_2 = (4, 2)$. Очевидно $p_1 \neq p_2$, но

$$\text{pow}(p_1) = 24 + 5^1 = 29$$
$$\text{pow}(p_2) = 4 + 5^2 = 29$$

така че $p_1 \preccurlyeq p_2$ и $p_2 \preccurlyeq p_1$. □

Ерго, $\preccurlyeq$ не е частична наредба и в частност не е линейна наредба. $\preccurlyeq$ очевидно не е и симетрична. Можем да кажем, че $\preccurlyeq$ е преднаредба. И то пълна преднаредба, защото няма несравними двойки. Щом е пълна преднаредба, съгласно изучаваното на лекции, можем да сортираме спрямо нея.

Ако обаче вторите елементи на двойките са по-големи от 1, то $\preccurlyeq$ е линейна наредба.

Лема 2. *Релацията $\preccurlyeq$ е силно антисиметрична, ако вторите елементи в двойките са по-големи от едно.*

Доказателство: Да разгледаме наредените двойки $p_1 = (x_1, y_1)$ и $p_2 = (x_2, y_2)$, като $y_1, y_2 \geq 2$ и $p_1 \neq p_2$. Първо да допуснем, че $y_1 = y_2$. Тогава $x_1 \neq x_2$, инак двойките биха били равни. БОО, нека $x_1 < x_2$. Тогава $\text{pow}(p_1) \preccurlyeq \text{pow}(p_2)$, но $\text{pow}(p_2) \not\preccurlyeq \text{pow}(p_1)$, точно както се иска за силна антисиметричност.

Сега да допуснем, че $y_1 \neq y_2$. БОО, нека $y_1 < y_2$. Тогава $y_2 = y_1 + k$, за някое положително k. Твърдим, че $\text{pow}(p_1) \preccurlyeq \text{pow}(p_2)$, но $\text{pow}(p_2) \not\preccurlyeq \text{pow}(p_1)$. Наистина,

$$\text{pow}(p_1) \leq n^2 - 1 + n^{y_1}$$
$$\text{pow}(p_2) \geq 1 + n^{y_2} = 1 + n^{y_1+k} = 1 + n^k \cdot n^{y_1}$$

и очевидно

$$n^{y_1} + n^2 - 1 < 1 + n^k \cdot n^{y_1} \iff n^2 - 2 < (n^k - 1)n^{y_1}$$

предвид това, че $x_1, x_2 \in \{1, \ldots, n^2 - 1\}$, $y_1 \geq 2$, $n \geq 2$ и $k \geq 1$.

Следствие 1. *Ако* $2 \leq y_1 < y_2$, *то* $(x_1, y_1) \preccurlyeq (x_2, y_2)$, *каквито и да са* x_1 *и* x_2.

Връщаме се към анализа на алгоритъма. Разбиването на A на A_1 и A^* във **Фаза 1** може да се реализира с едно линейно премитане на A във време $O(n)$. Това е тривиално.

Сортирането на A_1 във **Фаза 2** е лесно, понеже там са двойките от вида $(x, 1)$. Очевидно се сортират по x. Това може да стане в линейно време чрез Radix Sort по начин, който видяхме на лекции. Става дума за сортиране в линейно време на n числа, всяко от $\{1, \ldots, n^2\}$ от лекции.

Сортирането на A^* във **Фаза 3** може да стане в линейно време, ако ползваме като първичен ключ вторите елементи на двойките (игреците), а вторичен ключ са първите елементи от двойките (хиксовете). Първо сортираме по вторичен ключ със стабилен сортиращ алгоритъм и после по първичен ключ със стабилен сортиращ алгоритъм. Отново, Radix Sort е подходящ, понеже и хиксовете, и игреците се представят с около $2 \lg n$ бита, така че пак ползваме идеята от сортиране в линейно време на n числа, всяко от $\{1, \ldots, n^2\}$ от лекции. Коректността на тази идея се базира на Следствие 1. Забележете, че дотук никъде не извършваме степенуване n^y.

Сливането на A_1 и A^* във **Фаза 4** също може да стане в линейно време.

Забележете, че говорим именно за сливане, а не просто слагане на A_1 преди A^*! Не е вярно, че всеки елемент на A_1 е непременно преди всеки елемент на A^* в сортирания изход. Примерно, $\text{pow}(24, 1) = 29$, но $\text{pow}(1, 2) = 26$, така че $(24, 1)$ е **след** $(1, 2)$.

За да слеем A_1 и A^*, достатъчно е да слеем A_1 с елементите на A^* от вида $(x, 2)$. Масивът A^* е сортиран по първичен ключ вторите елементи на двойките, така че елементите му от вида $(x, 2)$ са в началото. Сливането на A_1 с елементите на A^* от вида $(x, 2)$ може да стане с нормален Merge, който изчислява $\text{pow}(x, y)$ за всеки елемент, подлежащ на сливане, и прави сравненията по тези стойности. Тези изчислявания на степени са разрешени в условието, защото y или е 1, или е 2.

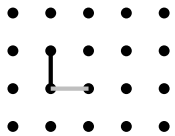
След като слеем A_1 с елементите на A^* от вида $(x, 2)$, останалите елементи на A^* се слагат в изходния масив в реда, в който са в A^*. Основание за това ни дава Следствие 1.

Фаза 4 се изпълнява във време $O(n)$. Откъдето заключаваме, че целият алгоритъм работи във време $O(n)$.

Зад. 5 Нека $n, m \in \mathbb{N}^+$. Представете си точките (i, j) за $1 \le i \le n$ и $1 \le j \le m$. Върху тези точки се играе игра от играчи **А** и **Б**. Те се редуват, като първо играе **А**. Всеки играч, когато е на ход, слага

- или хоризонтална отсечка между (i, j) и $(i, j + 1)$ за някои i и j, такива че $1 \le i \le n$ и $1 \le j < m$,
- или вертикална отсечка между (i, j) и $(i + 1, j)$ за някои i и j, такива че $1 \le i < n$ и $1 \le j \le m$,

но само ако между тези точки досега не е сложена отсечка. Нека отсечките на **А** са черни, а тези на **Б** са сиви. Ако $n = 4$, $m = 5$ и **А** сложи $(2, 2)$—$(3, 2)$, а **Б** сложи $(2, 2)$—$(2, 3)$, нещата изглеждат така:



Целта на **А** е да направи затворена крива (контур) от черни отсечки. Целта на **Б** е да попречи на **А** да стори това. Следният алгоритъм описва играта, без да казва как играчите избират ходовете си:

> Алгоритъм 1
> Докато може да бъдат сложени още отсечки:
>
> - **А** слага отсечка
> - ако може да се сложи отсечка, **Б** слага отсечка

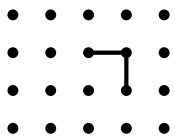
Разгледайте следния алгоритъм за избор на ход на **Б**.

> Алгоритъм 2
> Ако последният ход на **А** е бил (i, j)—$(i, j + 1)$, то,
>
> - ако $i > 1$ и няма отсечка $(i - 1, j + 1)$—$(i, j + 1)$, сложи $(i - 1, j + 1)$—$(i, j + 1)$,
> - в противен случай играй произволно,
>
> В противен случай, нека последният ход на **А** е бил (i, j)—$(i + 1, j)$.
>
> - Ако $j > 1$ и няма отсечка $(i + 1, j - 1)$—$(i + 1, j)$, сложи $(i + 1, j - 1)$—$(i + 1, j)$,
> - в противен случай играй произволно.

Докажете, че Алгоритъм 2 носи успех на **Б**, използвайки инвариант за Алгоритъм 1.

Решение: Първо забелязваме, че Алгоритъм 1 винаги терминира, тъй като максималният брой отсечки, които могат да бъдат сложени, е краен, а именно $2mn - m - n$; веднъж сложена, отсечка нито може да се махне, нито може да се дублира, така че след най-много $2mn - m - n$ добавяния на отсечки алгоритъмът спира.

За всеки две черни отсечки $(i - 1, j)$—(i, j) и (i, j)—$(i, j - 1)$ ще наричаме тяхната съвкупност *обърнато Г в* (i, j). Ето илюстрация на обърнато Г в $(3, 4)$:



Приемаме за абсолютно очевидно, че всеки черен контур има поне едно обърнато Г. От това следва, че ако **А** успее да конструира контур, той ще има поне обърнато Г. Контрапозитивно, ако в края на играта няма нито едно обърнато Г, то **А** не е успял да конструира контур, което означава, че **Б** е успял да попречи на **А** да конструира контур.

Стратегията, която **Б** следва чрез Алгоритъм 2, е проста: той пречи на **А** да образува обърнато Г.

Инвариант за цикъла на Алгоритъм 1 е този.

При всяко достигане на първия ред "Докато може да бъдат сложени още отсечки":

❶ *върху полето с точките няма обърнато Г и*

❷ *за всяка точка* (i, j) *и за всяко възможно добавяне на черна отсечка* x *е вярно, че добавянето на* x *не образува обърнато Г в* (i, j).

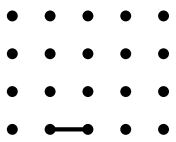
Базата е първото достигане. Тогава няма сложени отсечки. Следователно, в този момент няма обърнато Г, така че ❶ е вярно. Но ❷ също е вярно, тъй като обърнато Г се състои от две отсечки, ерго, както и да се сложи една черна отсечка, обърнато Г няма да се появи.

Да допуснем, че при някое достигане на първия ред на Алгоритъм 1, което не е последното, са в сила и ❶, и ❷.

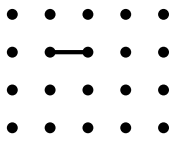
А е на ход. **А** има ход, инак следващо достигане на първия ред не би имало. **А** слага отсечка. Да я наречем x. Съгласно ❷ от допускането, добавянето на x не образува обърнато Г, така че ❶ остава в сила при следващото достигане на първия ред на Алгоритъм 1. Остава да се убедим, че ❷ е в сила при следващото достигане на първия ред на Алгоритъм 1.

Б е на ход. **Б** има ход, инак следващо достигане на първия ред не би имало. Разглеждаме следните две възможности по отношение на x, последно сложената от **А** отсечка. Възможностите са изчерпателни и взаимно изключващи се.

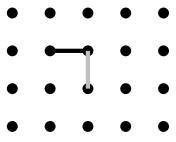
- x е хоризонтална отсечка. x е от вида (i, j)—$(i, j + 1)$.
  - Ако $i = 1$, тоест, x е върху най-долния ред, примерно

    то x не може да бъде част от обърнато Г. Предвид това и индуктивното преположение, ❷ остава в сила след добавянето на x. В този случай **Б** играе произволно. От това, че **Б** слага сива отсечка, не може да се появи възможност за поява на обърнато Г, каквато преди това е нямало. Заключаваме, че ❷ остава в сила при следващото достигане на първия ред на Алгоритъм 1.
  - Нека $i > 1$. Сега x не е върху най-долния ред, примерно

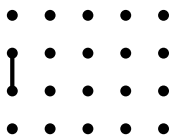
    Ако преди слагането на x е имало сива отсечка $(i, j + 1)$—$(i - 1, j + 1)$:

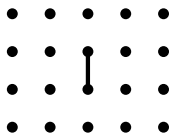
    то x не може да бъде част от обърнато Г. Предвид това и индуктивното преположение, ❷ остава в сила след добавянето на x. В този случай **Б** играе произволно. От това, че **Б** слага сива отсечка, не може да се появи възможност за поява на обърнато Г, каквато преди това е нямало. Заключаваме, че ❷ остава в сила при следващото достигане на първия ред на Алгоритъм 1.

    Ако обаче преди слагането на x не е имало сива отсечка $(i, j + 1)$—$(i - 1, j + 1)$, то **Б** слага сива отсечка $(i, j + 1)$—$(i - 1, j + 1)$. След това x не може да бъде част от обърнато Г. Предвид това и индуктивното преположение, ❷ остава в сила хода на **Б**. Заключаваме, че ❷ остава в сила при следващото достигане на първия ред на Алгоритъм 1.

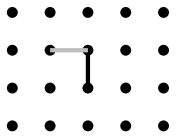
- x е вертикална отсечка. x е от вида (i, j)—$(i + 1, j)$.
  - Ако $j = 1$, тоест, x е върху най-лявата колона, примерно

    то x не може да бъде част от обърнато Г. Предвид това и индуктивното преположение, ❷ остава в сила след добавянето на x. В този случай **Б** играе произволно. От това, че **Б** слага сива отсечка, не може да се появи възможност за поява на обърнато Г, каквато преди това е нямало. Заключаваме, че ❷ остава в сила при следващото достигане на първия ред на Алгоритъм 1.
  - Нека $j > 1$. Сега x не е върху най-лявата колона, примерно

    Ако преди слагането на x е имало сива отсечка $(i + 1, j)$—$(i + 1, j - 1)$:

    то x не може да бъде част от обърнато Г. Предвид това и индуктивното преположение, ❷ остава в сила след добавянето на x. В този случай **Б** играе произволно. От това, че **Б** слага сива отсечка, не може да се появи възможност за поява на обърнато Г, каквато преди това е нямало. Заключаваме, че ❷ остава в сила при следващото достигане на първия ред на Алгоритъм 1.

    Ако обаче преди слагането на x не е имало сива отсечка $(i+1, j)$—$(i+1, j-1)$, то **Б** слага сива отсечка $(i+1, j)$—$(i+1, j-1)$. След това x не може да бъде част от обърнато Г. Предвид това и индуктивното преположение, ❷ остава в сила след хода на **Б**. Заключаваме, че ❷ остава в сила при следващото достигане на първия ред на Алгоритъм 1.

Доказахме инварианта. Разглеждаме терминацията на алгоритъма. Има две възможности: **А** да няма ход и **А** да има ход, но след това **Б** няма ход. Коя от тях ще се реализира е без значение за коректността на алгоритъма. Щом се стигне до ситуация, в която всички $2mn - m - n$ "слотове" са запълнени и, съгласно инварианта, няма обърнато Г, то **Б** е постигнал целта си.