Примерни решения на домашно № 2, 11 април 2021 г.

Задача 1: Дефиницията на "попълнено двоично дърво" от лекции е в сила. *Последният връх* в попълнено двоично дърво е най-дясното листо на последното ниво. *Свръхпирамида* е попълнено двоично дърво, в което всеки връх u, може би с изключение на последния връх, има наредена двойка ключове $(u.\ell, u.r)$, такива че $u.\ell \leq u.r$. Последният връх v или има наредена двойка ключове $(v.\ell, v.r)$, като $v.\ell \leq v.r$, или има единствен ключ $v.\ell$. Приемете, че ключовете са естествени числа. Нека x е произволен връх, който не е коренът, и p е родителят на x. Иска се:

- ако x не е последният връх, то $p.\ell \leq x.\ell \leq x.r \leq p.r$,
- ако x е последният връх и x има (наредена) двойка ключове, то $p.\ell \leq x.\ell \leq x.r \leq p.r$,
- ако x е последният връх и x има един единствен ключ, то $p.\ell \leq x.\ell \leq p.r$.

Всяка свръхпирамида има поне един връх, който има поне един ключ. Забележете, че всяко $n \in \mathbb{N}^+$ може да бъде брой на ключове в свръхпирамида, като,

- ако n е четно, то всеки връх има точно два ключа и броят на върховете е точно $\frac{n}{2}$,
- ако n е нечетно, то всеки връх без последния има точно два ключа, последният връх има точно един ключ, и броят на върховете е точно $\lceil \frac{n}{2} \rceil$.

Ще представяме свръхпирамида с n ключа чрез масив от $\lceil \frac{n}{2} \rceil$ **наредени двойки** от ключове. Възможно изключение е последният елемент, който може да съдържа единствен ключ и тогава той е число, а не наредена двойка, но това е така само ако броят на ключовете е нечетно число. Ако свръхпирамидата е A, то A.size е броят на елементите на масива; тоест, $\lceil \frac{n}{2} \rceil$. Ако броят на ключовете е нечетен, то $A[A.\text{size}].r$ има стойност Undef.

Съвкупността от всички ключове на свръхпирамидата е някакво мултимножество от естествени числа, понеже повторения на ключове са възможни.

1 т.

- Покажете една възможна свръхпирамида върху следното мултимножество от ключове:

  $$\{0, 0, 4, 5, 6, 6, 6, 6, 7, 8, 8, 11, 15, 19, 20, 30, 35, 38, 40\}_M$$

  Може да я нарисувате като дърво, с двойките ключове вътре или до върховете.

1 т.

- Да кажем, че свръхпирамида е представена чрез масив A. Предложете колкото е възможно по-ефикасни имплементации на функции Min(A) и Max(A), които връщат съответно минимален и максимален ключ от свръхпирамидата, без да променят нищо в нея. Накратко обосновете коректността на функциите.

23 т.

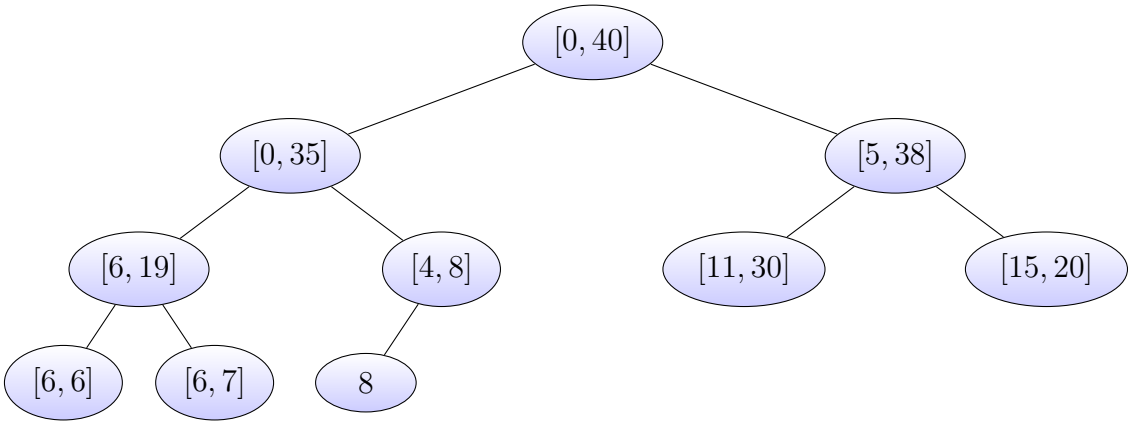
- Предложете колкото е възможно по-ефикасна имплементация на функция INSERT(A, x), която добавя нов ключ x към досегашното мултимножество от ключове. Допуснете, че масивът $A[1, \ldots, A.\text{size}]$ е подмасив на някакъв достатъчно голям масив $A[1, \ldots, N]$, така че, ако инкрементирате A.size, то няма опасност да излезете извън масива. Напишете ясен псевдокод на функцията и го изследвайте детайлно и формално прецизно за коректност и сложност по време.

  Забележка: INSERT не създава свръхпирамида. Допуснете, че свръхпирамидата вече е създадена и съдържа поне един ключ.

Решение: Световно приетата конвенция е такива пирамиди да се наричат *интервални пирамиди*, като всеки връх е асоцииран със затворен интервал, който интервал съдържа всеки от интервалите на децата. Очеведно интервалът на корена съдържа всеки от интервалите, които се появяват в дървото.

Последният връх, тоест най-дясното листо на последния ред е възможно изключение: то и само то може да се асоциира с едно число, а не с интервал. Тук очевидно различаваме интервала $[a, a]$ от числото a; ако последния връх е именуван с $[a, a]$, то това е интервал и в този случай дървото има четен брой ключове, а ако е именуван с a, то това не е интервал и дървото има нечетен брой ключове.

В първото подусловие ключовете са деветнадесет на брой, така че последното листо трябва да се асоциира с число, а не с интервал. Естествено, това число трябва да се съдържа в интервала на родителя. Възможните решения са много. Ето едно от тях:



По второто подусловие: щом интервалът $[u, v]$ на корена съдържа всеки друг интервал в дървото, то u е минимален ключ, а v е максимален ключ. Това е в сила тстк ключовете са поне два. Ако ключът е един единствен, то числото в корена е и минимален, и максимален елемент. В някакъв смисъл—игнорирайки факта, че може да има връх с един единствен ключ—една интервална пирамида представя едновременно две обикновени пирамиди с еднаква големина, едната от които е от тип max, а другата е от тип min, като долните граници на интервалите са ключовете на min пирамидата, а десните, на max пирамидата.

```
MIN(A: интервална пирамида)
1  return A[1].ℓ
```

```
MAX(A: интервална пирамида)
1  if A.size = 1
2       return A[1].ℓ
3  else
4       return A[1].r
```

По третото подусловие: най-общо казано, ако последното листо има само един ключ, правим от него и x интервал, а ако последното листо има два ключа, добавяме нов връх с един единствен ключ, което при представяне с масив е същото като увеличаване на A.size и записваме на x в $A[A.\text{size}].\ell$; след което трябва да се извършат, колкото може по-икономично, модификации на интервалите чрез разместванията на ключове, така че мултимножеството от ключовете да не се промени, но необходимите отношения между интервалите да бъдат в сила.

```
INSERT(A: интервална пирамида, x: нов ключ)
 1  gosmall ← FALSE, gobig ← FALSE, k ← A.size
 2  if A[k].r = UNDEF
 3      y ← A[k].ℓ
 4      A[k].r ← max {x, y}
 5      A[k].ℓ ← min {x, y}
 6      if k > 1 and A[k].ℓ < A[PARENT(k)].ℓ
 7          gosmall ← TRUE
 8      else if k > 1 and A[k].r > A[PARENT(k)].r
 9          gobig ← TRUE
10      if gosmall
11          while k > 1 and A[PARENT(k)].ℓ < A[k].ℓ do
12              swap(A[k].ℓ, A[PARENT(k)].ℓ)
13              k ← PARENT(k)
14      if gobig
15          while k > 1 and A[PARENT(k)].r > A[k].r do
16              swap(A[k].r, A[PARENT(k)].r)
17              k ← PARENT(k)
18  else
19      k++, A.size++
20      A[k].r ← UNDEF
21      A[k].ℓ ← x
22      if k > 1 and A[k].ℓ < A[PARENT(k)].ℓ
23          gosmall ← TRUE
24      else if k > 1 and A[k].ℓ > A[PARENT(k)].r
25          gobig ← TRUE
26      if gosmall
27          while k > 1 and A[PARENT(k)].ℓ < A[k].ℓ do
28              swap(A[k].ℓ, A[PARENT(k)].ℓ)
29              k ← PARENT(k)
30      if gobig
31          swap(A[k].ℓ, A[PARENT(k)].r)
32          k ← PARENT(k)
33          while k > 1 and A[PARENT(k)].r > A[k].r do
34              swap(A[k].r, A[PARENT(k)].r)
35              k ← PARENT(k)
```

Ето аргументация за коректност. $A[k].r = \textsc{Undef}$ на ред 2 е същото като броят на ключовете да е нечетен, което е същото като последният връх да съдържа число, а не интервал. Да допуснем, че последният връх съдържа число. На редове 3–5 алгоритъмът коректно създава интервал в последния връх, като $A[k].\ell$ не надвишава $A[k].r$. Ако $k \leq 1$, алгоритъмът не прави нищо повече, което е правилно. Ако $k > 1$, проверките на редове 6 и 8 се извършват. Забележете, че е невъзможно и двете булеви условия да са истина, защото няма как хем $\min\{x, y\} < A[\textsc{Parent}(k)].\ell$, хем $\max\{x, y\} > A[\textsc{Parent}(k)].r$, имайки предвид, че $A[\textsc{Parent}(k)].\ell \leq y \leq A[\textsc{Parent}(k)].r$.

- Ако текущият интервал $[A[k].\ell, A[k].r]$ се съдържа в интервала $\big[A[\textsc{Parent}(k)].\ell, A[\textsc{Parent}(k)].r\big]$, то нито условието на ред 6 е истина, нито условието на ред 8 е истина, така че алгоритъмът терминира и, от друга страна, интервалната пирамида е коректна.

- Да допуснем, че $A[k].\ell < A[\textsc{Parent}(k)].\ell$. В такъв случай текущият $A[k].\ell$ е новодобавеният x, а $A[k].r$ съдържа стойността на $A[k].\ell$ от началото на алгоритъма. Да мислим за интервалната пирамида като за съвкупност от две обикновени двоични пирамиди – min пирамида и max пирамида. В момента max пирамидата е "в ред", но min пирамидата не е. Променливата gosmall става истина, променливата gobig става лъже и се изпълнява **while**-цикълът на редове 11–13. Но това е част от кода на наивното построяване на обикновена пирамида от min тип, а именно, придвижването чрез размени на един единствен елемент от листо нагоре до правилното му място. На лекции сме доказали колектността на практически същия код, само че за пирамида от max тип, така че няма смисъл сега да доказваме колектността на същото нещо.

  Знаем, че в най-лошия случай елементът се "качва" от листо до корена и сложността по време е логаритмична във размера на пирамидата.

- Да допуснем, че $A[k].r > A[\textsc{Parent}(k)].r$. С дуални съображения виждаме, че min пирамидата е "в ред", но max пирамидата не е. Изпълнява се **while**-цикълът на редове 15–17, който коректно поставя "в ред" max пирамидата във време, което, в най-лошия случай, е логаритмично в размера на пирамидата.

Сега да допуснем, че последният връх съдържа интервал. Доказателството за коректност е почти същото, с малката разлика, че сега (коректно) инкрементираме A.size, понеже се налага да добавим нов връх, асоцииран с число; това, че е асоцииран с число, а не с интервал, точно отговаря на факта, че $A[k].\ell$ става x, а $A[k].r$ е недефинирана. Алгоритъмът на редове 22–35 почти повтаря редове 6–17, с малката разлика, че ако gobig е истина, първата размяна с родител е на $A[k].\ell$ с $A[\textsc{Parent}(k)].r$, а след това е на $A[k].r$ с $A[\textsc{Parent}(k)].r$. Причината е очевидна – $A[k].r$ за $k = A$.size не е дефинирано.

Задача 2: Решете формално рекурентното уравнение

$$T(n) = 2T(n-1) + (\lg n)T(\lfloor \lg n \rfloor) + 1$$

Решение: Отгатваме, че решението е $T(n) = \Theta(2^n)$. Ще докажем това по индукция.

Ще докажем, че $T(n) = O(2^n)$. По определение, това е същото като да докажем, че съществува положителна константа c, такава че за всички достатъчно големи стойности на аргумента е вярно, че $T(n) \leq c2^n$. Ще докажем дори нещо по-силно:

$$\exists c, b > 0 : T(n) \leq c2^n - bn^2 \tag{1}$$

От индуктивните предположения, които са за стойности на аргумента n_0, n_0+1, $\ldots$, $n-1$, имаме:

$$T(n-1) \leq c2^{n-1} - b(n-1)^2 = \frac{1}{2}c2^n - bn^2 + 2bn - b \tag{2}$$

$$\begin{aligned} T(\lfloor \lg n \rfloor) &\leq c2^{\lfloor \lg n \rfloor} - b(\lfloor \lg n \rfloor)^2 \leq c2^{\lg n} - b(\lfloor \lg n \rfloor)^2 \\ &= cn - b(\lfloor \lg n \rfloor)^2 \leq cn - b((\lg n) - 1)^2 \\ &= cn - b(\lg n)^2 + 2b(\lg n) - b \end{aligned} \tag{3}$$

Заместваме десните страни на (2) и (3) в дефиницията на $T(n)$ и получаваме

$$\begin{aligned} T(n) &\leq 2\left(\frac{1}{2}c2^n - bn^2 + 2bn - b\right) + (\lg n)\big(cn - b(\lg n)^2 + 2b(\lg n) - b\big) + 1 \\ &= c2^n - 2bn^2 + 4bn - 2b + cn\lg n - b(\lg n)^3 + 2b(\lg n)^2 - b(\lg n) + 1 \qquad (4) \\ &= c2^n - bn^2 + \underbrace{(-bn^2 + 4bn - 2b + cn\lg n - b(\lg n)^3 + 2b(\lg n)^2 - b(\lg n) + 1)}_{A} \qquad (5) \end{aligned}$$

Ако успеем да покажем, че $A \leq 0$, то имаме право да твърдим, че $T(n) \leq c2^n - bn^2$. Но

$$A = -bn^2 + cn\lg n + 4bn - b(\lg n)^3 + 2b(\lg n)^2 - b(\lg n) + 1 - 2b$$

Тъй като bn^2 доминира над $cn\lg n + 4bn - b(\lg n)^3 + 2b(\lg n)^2 - b(\lg n) + 1 - 2b$, в асимптотичния смисъл, то заради отрицателния знак пред bn^2 със сигурност има положителни константи b и c, такива че за всички достатъчно големи стойности на аргумента е вярно, че $A < 0$. Доказахме (1).

Ще докажем, че $T(n) = \Omega(2^n)$. По определение, това е същото като да докажем, че съществува положителна константа d, такава че за всички достатъчно големи стойности на аргумента е вярно, че $T(n) \geq d2^n$. От индуктивните предположения, които са за стойности на аргумента n_0, n_0+1, $\ldots$, $n-1$, имаме:

$$T(n-1) \geq c2^{n-1} = \frac{1}{2}d2^n \tag{6}$$

$$T(\lfloor \lg n \rfloor) \geq d2^{\lfloor \lg n \rfloor} \geq d2^{(\lg n)-1} = \frac{1}{2}dn \tag{7}$$

Заместваме десните страни на (6) и (7) в дефиницията на $T(n)$ и получаваме

$$\begin{aligned} T(n) &\geq 2 \cdot \frac{1}{2} \cdot d2^n + (\lg n)\frac{1}{2}dn \\ &= d2^n + \frac{d}{2}n\lg n + 1 \\ &\geq d2^n \end{aligned}$$

Задача 3: В алгоритъма SELECT, изучаван на лекции, на редове 4, 6 и 8 (според псевдокода в лекционните записки) има константа 5. Анализирайте алгоритъма, ако тази константа не е 5, а е

- 4,
- 7.

Решение: Първо да допуснем, че константата е 4. Това означава, че разбиваме масива на $\lceil \frac{n}{4} \rceil$ подмасива B_1 и така нататък, всеки с големина 4, може би без последния, който е с големина от 1 до 3. Сортираме ги в $\Theta(n)$ общо време, намираме им медианите и правим масив C от медианите. Очевидно големината на C е $\lceil \frac{n}{4} \rceil$. Намираме с рекурсивно викане медианата на медианите m и изпълняваме кода на редове 13–19. За това изпълнение стойността 4 на константата от началото е без значение. Коректността на алгоритъма не се променя.

Това, което може да се промени, е сложността по време. Тъй като масивите B_i са с големина 4, медианата на кой да е от тях не може да е средният елемент, защото такъв няма. В този контекст не може да дефинираме медиана на масив с четен размер като средно артитметично от средните елементи (вторият и третият в този случай). Медианата трябва да е елемент на оригиналния масив. Така че дефинираме медианата или като втория, или като третия елемент на сортирания B_i, за всяко i. Как точно ще сторим това е без значение за сложността в най-лошия случай. Най-лошият случай се получава, когато условието на ред 13 е лъжа и правим едното (само едното!) от рекурсивните викания на ред 17 или ред 19, като това става върху подмасив с максимален брой елементи.

Какъв е този максимален брой? В най-лошия случай, това са всички елементи на **входния** масив, които се намират от едната страна на m след викането на PARTITION. Освен това, за да е най-лош случаят, k-то от входа е такова, че се налага да викаме рекурсивно именно върху този подмасив (вляво или вдясно от m), който има такъв максимален брой елементи. Максималният брой такива елементи се получава, когато m е по-голяма (или по-малка, за най-лошия случай няма значение) само от тези елементи, от които "се налага" да е по-голяма (по-малка) от съображението, че е медиана на масива от медианите. Когато константата в началото беше 5, m се налагаше да е по-голяма (по-малка) от 3 елемента в половината B_i-та (и от още два елемента, но това е без значение за асимптотиката); тъй като B_i-тата имат по $n/5$ елемента, тази сметка водеше до извода, че m е по-голяма (по-малка) от $3n/10$ елемента, което оставя $7n/10$ елемента за подмасива, върху който викаме в най-лошия случай. Сега сметката е друга: m е по-голяма (по-малка) от 2 елемента на половината B_i-та. Тъй като B_i-тата са $n/4$ на брой, виждаме, че се налага m да е по-голяма (по-малка) от $2\frac{\frac{n}{4}}{2} = \frac{n}{4}$ елемента. Тогава, в най-лошия случай, се налага да викаме рекурсивно върху масив от $\frac{3n}{4}$ елемента. Тогава

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + n$$

Това рекурентно уравнение има решение $T(n) \asymp n \lg n$, което може да покажем по индукция, а може и чрез теоремата на Akra-Bazzi. Имаме $a_1 = a_2 = 1$, $b_1 = \frac{1}{4}$, $b_2 = \frac{3}{4}$ и $g(n) = n$. Числото p е уникалното число, такова че

$$a_1 \cdot b_1^p + a_2 \cdot b_2^p = 1 \;\leftrightarrow \left(\frac{1}{4}\right)^p + \left(\frac{3}{4}\right)^p = 1$$

Не ни трябва Maple, за да сметнем, че $p = 1$. Оттук

$$T(n) \asymp n\left(1 + \int_1^n \frac{g(u)}{u^2}\,du\right) = n\left(1 + \int_1^n \frac{du}{u}\right)$$
$$= n\,(1 + \ln n) \asymp n \lg n$$

Ако обаче $n = 7$, съображенията за сложността се променят така. Тогава m е поголяма (по-малка) от 4 елемента в половината B_i-та, (почти) всяко от които е с размер $n/7$, което дава около $4\frac{n}{14} = \frac{2n}{7}$. Ерго, в най-лошия случай викането накрая е върху масив с размер $\frac{5n}{7}$. Тогава

$$T(n) = T\left(\frac{n}{7}\right) + T\left(\frac{5n}{7}\right) + n$$

Тъй като $\frac{1}{7} + \frac{5}{7} < 1$, решението е $T(n) \asymp n$ с почти същото решение като в лекционните записки. Константата p е друга, но отново е вярно, че $0 < p < 1$, така че асимптотиката излиза същата.

Задача 4: Докажете, че HEAPSORT има сложност по време в най-добрия случай $\Theta(n \lg n)$, ако елементите от входа са уникални (няма повторения).

Решение: Това, че сложността в най-добрия случай е $O(n \lg n)$, е очевидно (тривиално следва от това, че сложността в най-лошия случай е $O(n \lg n)$). Ще покажем, че в най-добрия случай сложността е $\Omega(n \lg n)$. За тази цел е достатъчно да покажем, че за $\Omega(n)$ върхове е вярно, че всеки от тях участва в $\Omega(\lg n)$ размени. Пишейки $\frac{n}{k}$, имаме предвид $\left\lfloor \frac{n}{k} \right\rfloor$ или $\left\lceil \frac{n}{k} \right\rceil$; това е без значение за асимптотиката.

Нека $A[1, \ldots, n]$ е входният масив. Неко S е множеството от входните елементи, като $|S| = n$, понеже във входа няма повторения. Нека $X, Y \subseteq S$, като $\forall x \in X\, \forall y \in Y$: $x < y$ и $|X| = \frac{n}{2}$ и $|Y| = \frac{n}{2}$.

Да разгледаме пирамидата Π, построена от входния A с процедурата BUILD HEAP. Тъй като коренът на Π е от Y и за всеки връх $y \in Y$ е вярно, че по уникалния път от y до корена всички върхове са от Y, лесно се вижда, че върховете от Y индуцират едно единствено поддърво D на пирамидата (за разлика от върховете от X, които в общия случай индуцират множество дървета), което съдържа корена.

Най-много $\frac{n}{4}$ елемента от Y може да са листа на Π, понеже в двоично дърво листата не може да са повече от половината от върховете. А D е двоично дърво с $\frac{n}{2}$ върхове и листата на D са подмножество на листата на Π.

Нека L_i са върховете в Π с височина i, които са от Y. Току-що показахме, че $|L_0| \leq \frac{n}{4}$. Нека h е височината на Π. Принадлежността към $L_0, \ldots, L_h$ генерира разбиване на Y, като

$$|L_0| + |L_1| + \cdots + |L_h| = \frac{n}{2}$$

Предвид това, че $|L_0| \leq \frac{n}{4}$, имаме

$$|L_1| + |L_2| + \cdots + |L_h| \geq \frac{n}{4}$$

Ако допуснем, че $|L_1| < \frac{n}{8}$, следва, че

$$|L_2| + \cdots + |L_h| > \frac{n}{8}$$

което е очевидно невъзможно, защото $L_2, \ldots, L_h$ са пълни нивá в П. Доказахме, че $|L_1| \geq \frac{n}{8}$. Конкретната стойност не е важна, важното е, че $|L_1| = \Omega(n)$.

Припомняме си, че HEAPSORT гради сортирания масив отдясно наляво. В крайния сортиран масив, елементите от X са преди тези от Y, така че HEAPSORT слага Y по местата им в дясната половина на масива **преди** да започне да слага елементите от X по местата им в лявата половина на масива. Ключовото наблюдение следното.

- Всеки елементи от L_0 **може** да бъде сложен на окончателното си място само с две размествания: първо "скача" във върха на пирамидата чрез $\text{swap}(A[1], A[i])$ на ред 4, после $\text{HEAPIFY}(A, 1)$ не размества нищо, понеже този елемент е достатъчно голям, и следващият $\text{swap}(A[1], A[i])$ на ред 4 ще го "изхвърли" извън текущия $A[1, \ldots, A.\text{size}]$ на окончателното му място в $A[1, \ldots, n]$.

- За разлика от елементите от L_0, тези от L_1 **трябва** да "пропътуват" цялото разстояние до корена, преди да бъдат "изхвърлени" извън текущия $A[1, \ldots, A.\text{size}]$ на окончателните си места в $A[1, \ldots, n]$ чрез $\text{swap}(A[1], A[i])$ на ред 4.

  За да се убедим, че е така, достатъчно е да забележим, че $L_1 \subset Y$ и елементите от L_1 трябва да бъдат сложени по окончателните си места още докато i е по-голяма или равна на $\frac{n}{2}$. Но единственият начин това да стане е чрез $\text{swap}(A[1], A[i])$ на ред 4. Тъй като i е прекалено голяма, за да бъде индекса на елемент от L_1, той трябва преди това, някак си, да се е придвижил от началната си позиция в П чак до корена $A[1]$. Това придвижване е станало в резултат на изпълнения на HEAPIFY, която разменя върхове от **съседни** нивá. Ерго, всеки такъв връх е бил участник в $\Omega(\lg n)$ размени.

Щом всеки от $\Omega(n)$ върха участва в $\Omega(\lg n)$ размени, алгоритъмът работи в $\Omega(n \lg n)$ време винаги.