Пирамида

Трифон Трифонов

Структури от данни и програмиране,
спец. Компютърни науки, 2 поток, 2015/16 г.

4 декември 2015 г.

АТД: приоритетна опашка

Опашка, в която елементите са наредени по приоритет и първи се обработва най-приоритетният елемент.

Операции

- `create()` — създаване на празна приоритетна опашка
- `build(l)` — създаване на приоритетна опашка по списък `l` от елементи с приоритет
- `empty()` — проверка за празнота на приоритетна опашка
- `enqueue_prioritized(x, p)` — включване на елемент `x` с приоритет `p` в опашката
- `dequeue_highest()` — изключване на елемента с най-висок приоритет от опашката
- `head()` — достъп до елемента с най-висок приоритет

Сортиран списък като приоритетна опашка

Списък, в който елементите са сортирани низходящо по приоритет.
Сложност на операциите:

- `create()`, `empty()` — $O(1)$
  - съответните операции за списъци

Сортиран списък като приоритетна опашка

Списък, в който елементите са сортирани низходящо по приоритет.
Сложност на операциите:

- `create()`, `empty()` — $O(1)$
  - съответните операции за списъци
- `enqueue_prioritized(x, p)` — $O(n)$
  - обхождане и `insertAfter`

Сортиран списък като приоритетна опашка

Списък, в който елементите са сортирани низходящо по приоритет.
Сложност на операциите:

- `create()`, `empty()` — $O(1)$
  - съответните операции за списъци
- `enqueue_prioritized(x, p)` — $O(n)$
  - обхождане и `insertAfter`
- `dequeue_highest()` — $O(1)$
  - `deleteBegin`

Сортиран списък като приоритетна опашка

Списък, в който елементите са сортирани низходящо по приоритет.
Сложност на операциите:

- `create()`, `empty()` — $O(1)$
  - съответните операции за списъци
- `enqueue_prioritized(x, p)` — $O(n)$
  - обхождане и `insertAfter`
- `dequeue_highest()` — $O(1)$
  - `deleteBegin`
- `head()` — $O(1)$
  - `*begin()`

Сортиран списък като приоритетна опашка

Списък, в който елементите са сортирани низходящо по приоритет.
Сложност на операциите:

- `create()`, `empty()` — $O(1)$
  - съответните операции за списъци
- `enqueue_prioritized(x, p)` — $O(n)$
  - обхождане и `insertAfter`
- `dequeue_highest()` — $O(1)$
  - `deleteBegin`
- `head()` — $O(1)$
  - `*begin()`
- `build(l)` — $O(n^2)$
  - повтаряне на `enqueue_prioritized`

Сортиран списък като приоритетна опашка

Списък, в който елементите са сортирани низходящо по приоритет.
Сложност на операциите:

- `create()`, `empty()` — $O(1)$
  - съответните операции за списъци
- `enqueue_prioritized(x, p)` — $O(n)$
  - обхождане и `insertAfter`
- `dequeue_highest()` — $O(1)$
  - `deleteBegin`
- `head()` — $O(1)$
  - `*begin()`
- `build(l)` — $O(n^2)$
  - повтаряне на `enqueue_prioritized`

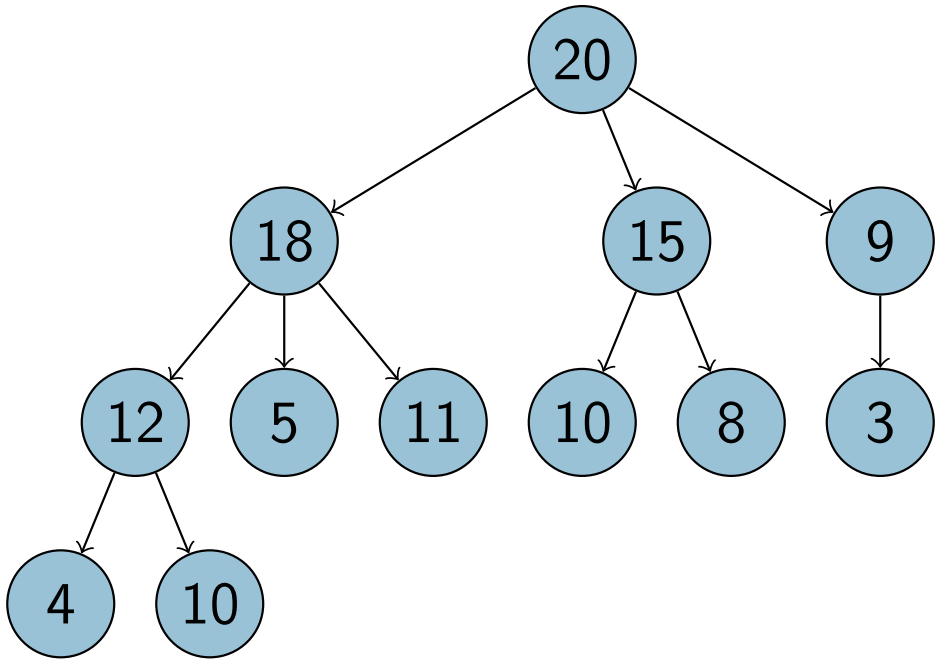
Не е нужно елементите в опашката да са сортирани!

Пирамида

Дефиниция (Пирамида)

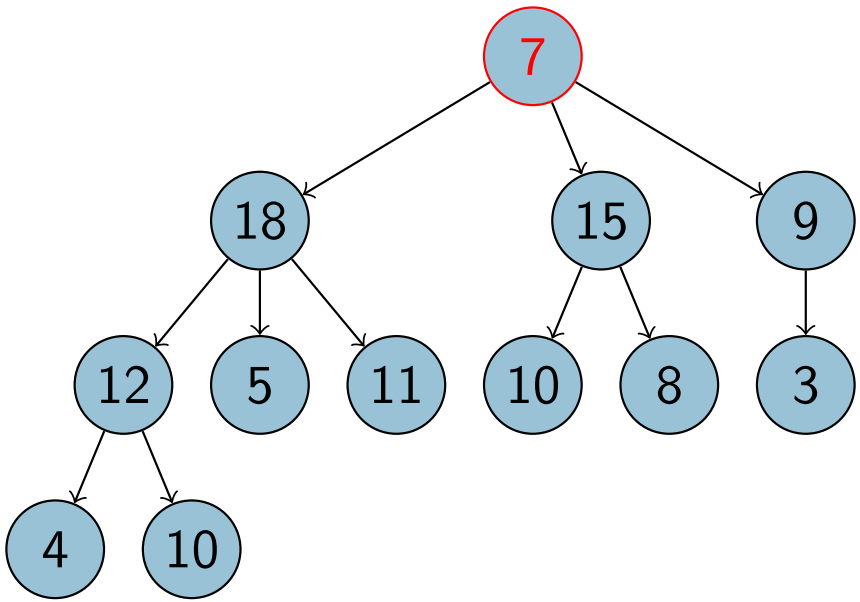
Дърво, в което всеки родител е с по-висок приоритет от децата си.

Пример:



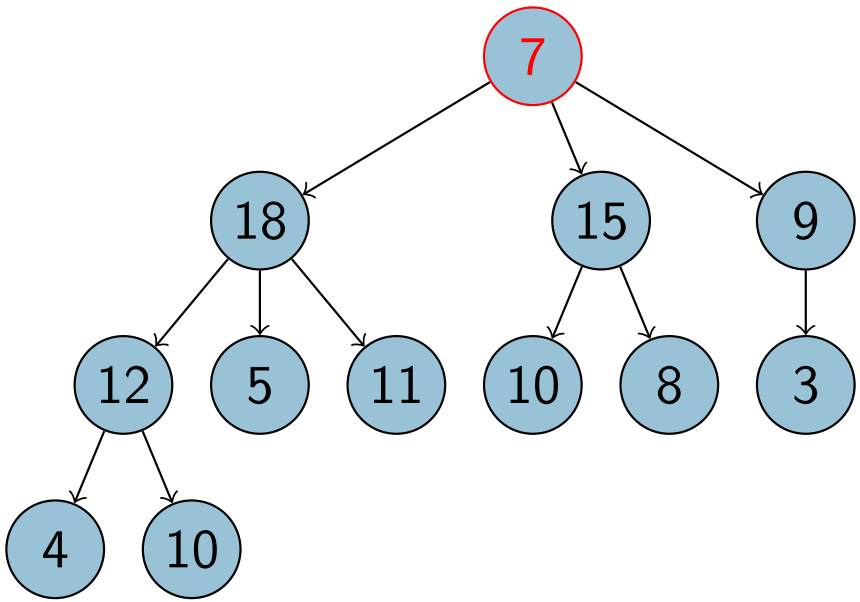
Пресяване надолу (sift down)

- Да разгледаме елемент, който нарушава пирамидалното свойство



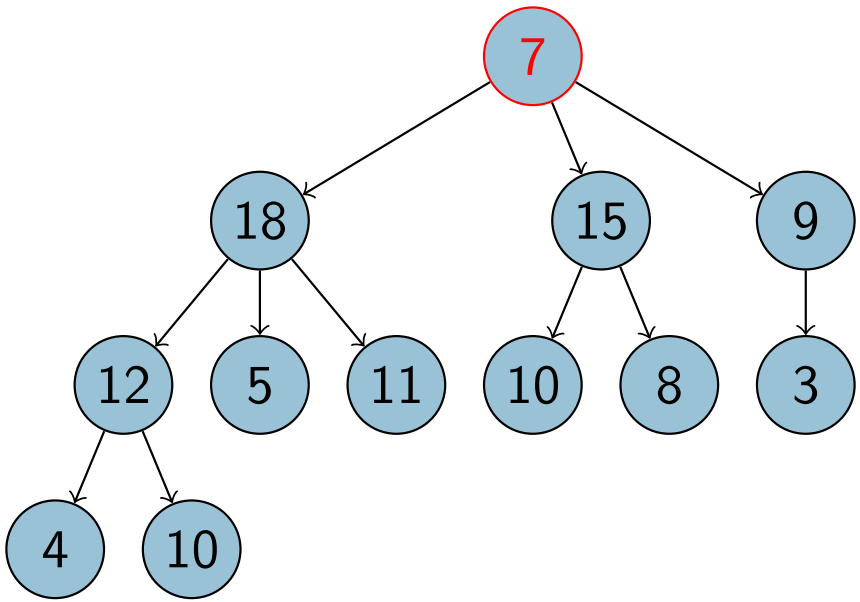
Пресяване надолу (sift down)

- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. някое от децата му е по-голямо от него.



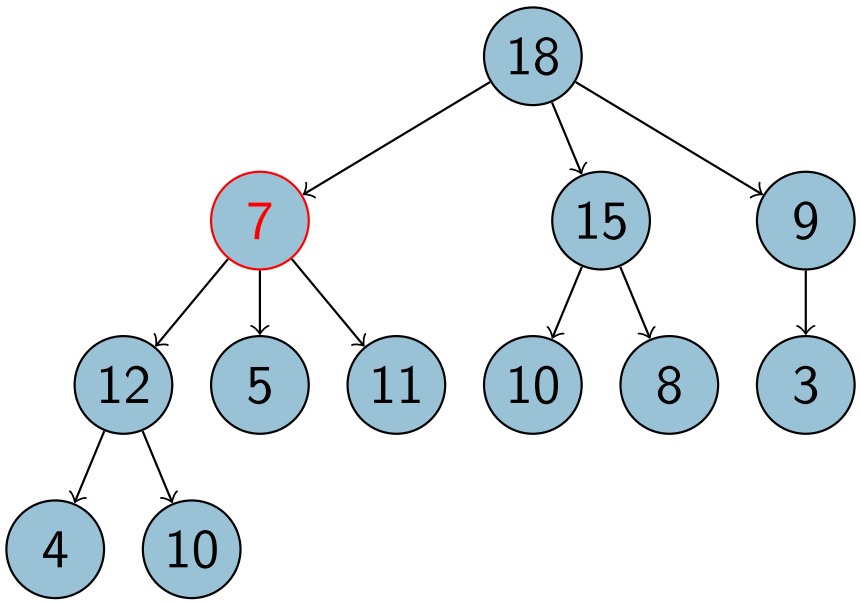
Пресяване надолу (sift down)

- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. някое от децата му е по-голямо от него.
- Как да възстановим пирамидата?



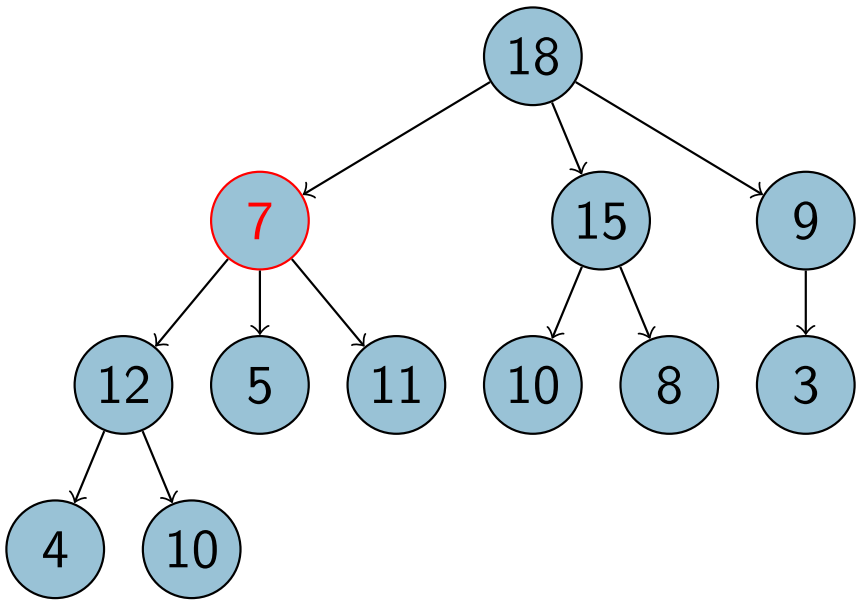
Пресяване надолу (sift down)

- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. някое от децата му е по-голямо от него.
- Как да възстановим пирамидата?
- Идея: разменяме "нарушителя" с **най-голямото** му дете (защо?)



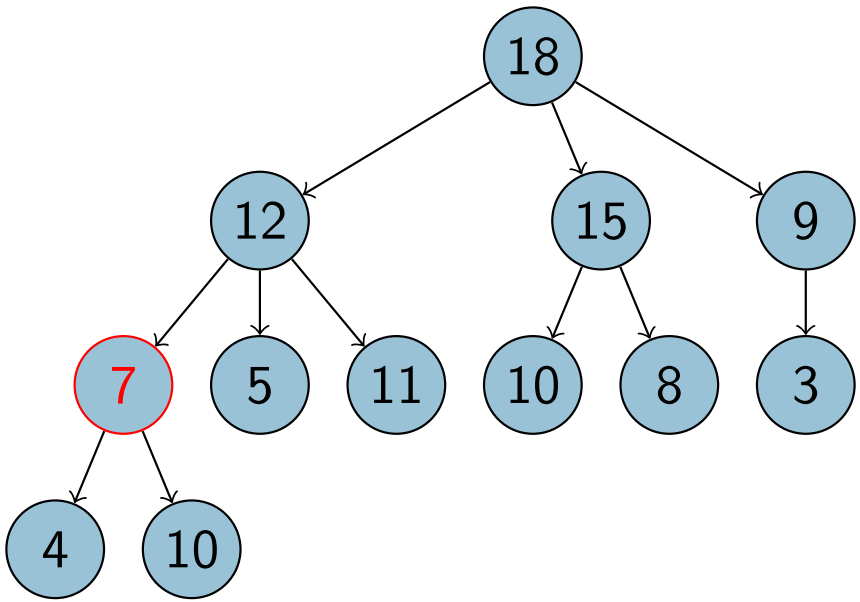
Пресяване надолу (sift down)

- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. някое от децата му е по-голямо от него.
- Как да възстановим пирамидата?
- Идея: разменяме "нарушителя" с **най-голямото** му дете (защо?)
- Продължаваме докато има нарушение или не стигнем до листо



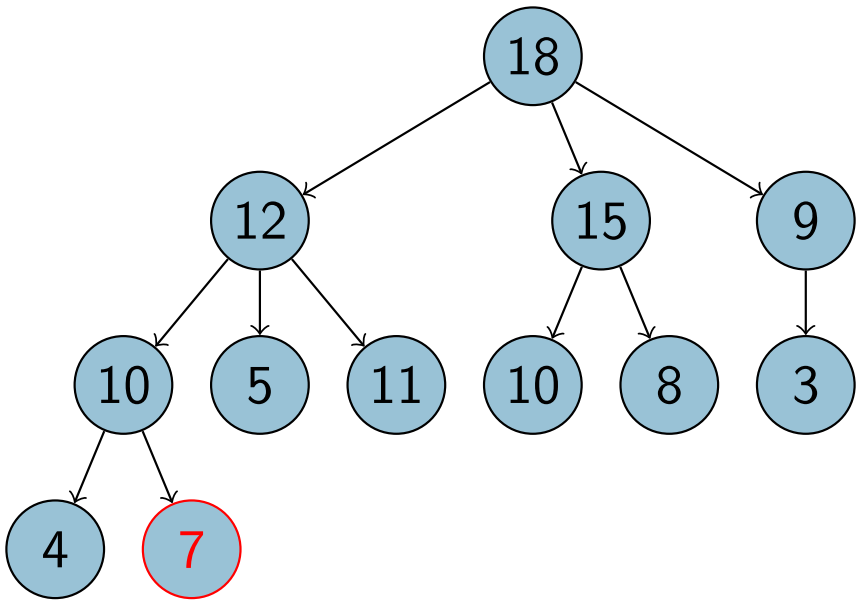
Пресяване надолу (sift down)

- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. някое от децата му е по-голямо от него.
- Как да възстановим пирамидата?
- Идея: разменяме "нарушителя" с **най-голямото** му дете (защо?)
- Продължаваме докато има нарушение или не стигнем до листо



Пресяване надолу (sift down)

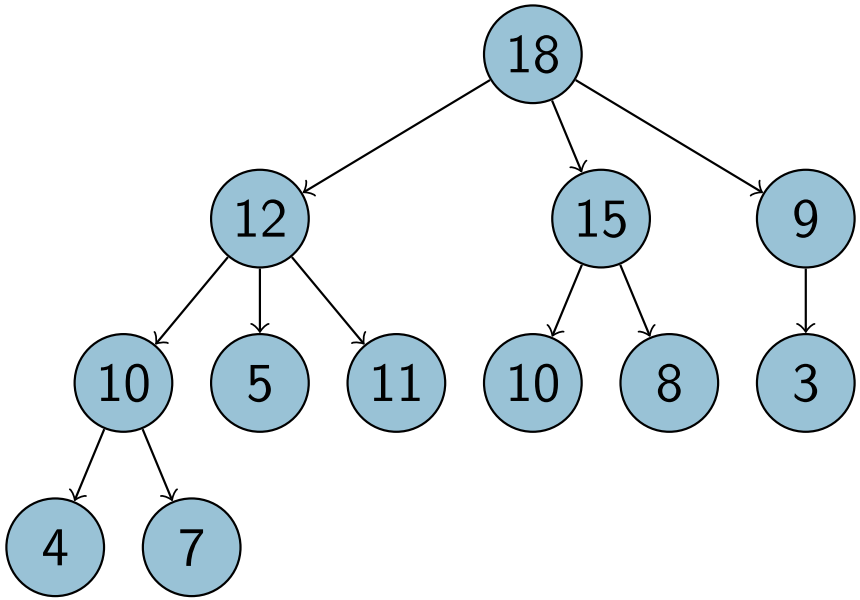
- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. някое от децата му е по-голямо от него.
- Как да възстановим пирамидата?
- Идея: разменяме "нарушителя" с **най-голямото** му дете (защо?)
- Продължаваме докато има нарушение или не стигнем до листо



Сложност: $O(h)$, където h е височината на пирамидата.

Пресяване надолу (sift down)

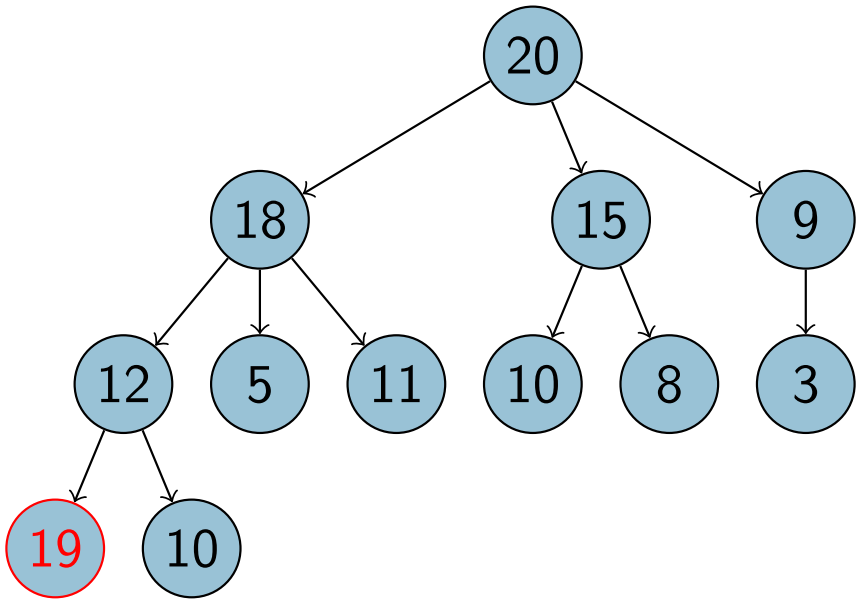
- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. някое от децата му е по-голямо от него.
- Как да възстановим пирамидата?
- Идея: разменяме "нарушителя" с **най-голямото** му дете (защо?)
- Продължаваме докато има нарушение или не стигнем до листо
- Листата никога не нарушават пирамидалното свойство



Сложност: $O(h)$, където h е височината на пирамидата.

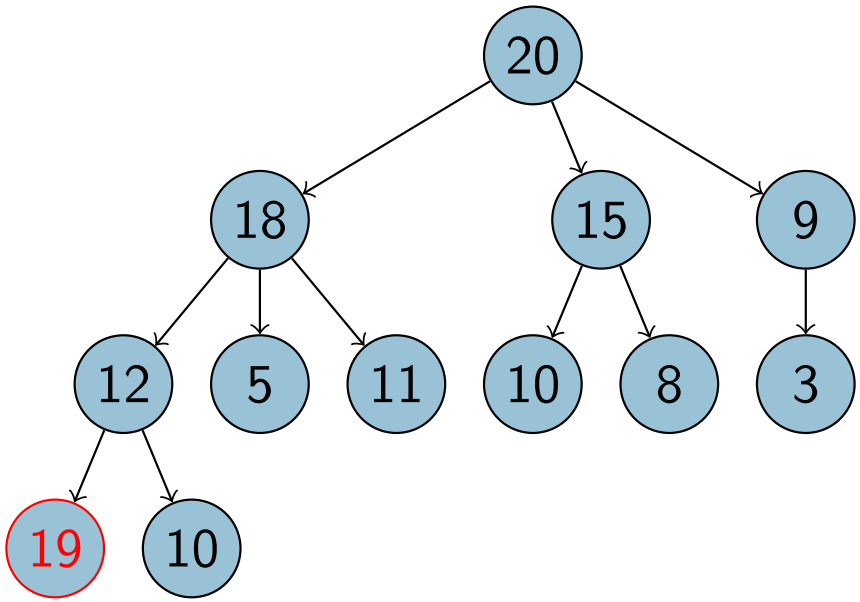
Пресяване нагоре (sift up)

- Да разгледаме елемент, който нарушава пирамидалното свойство



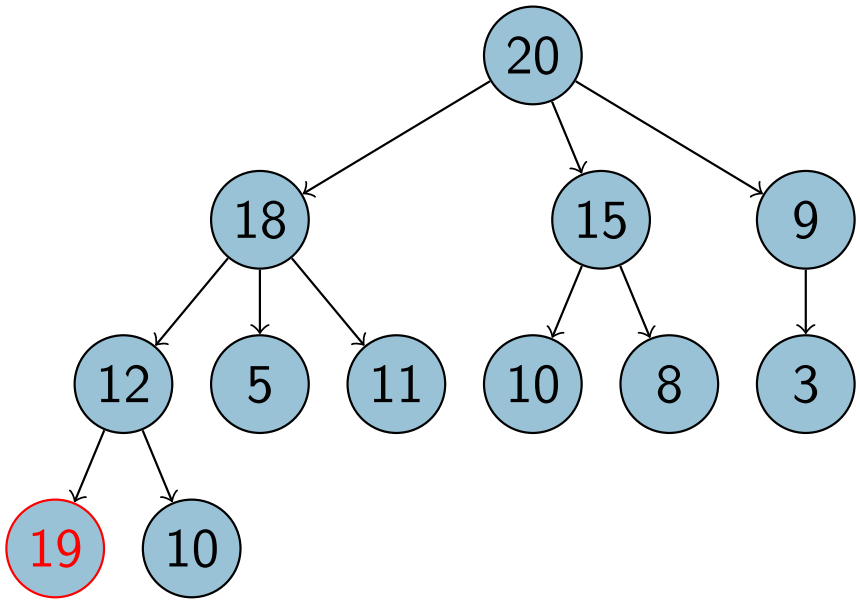
Пресяване нагоре (sift up)

- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. по-голям е от родителя си.



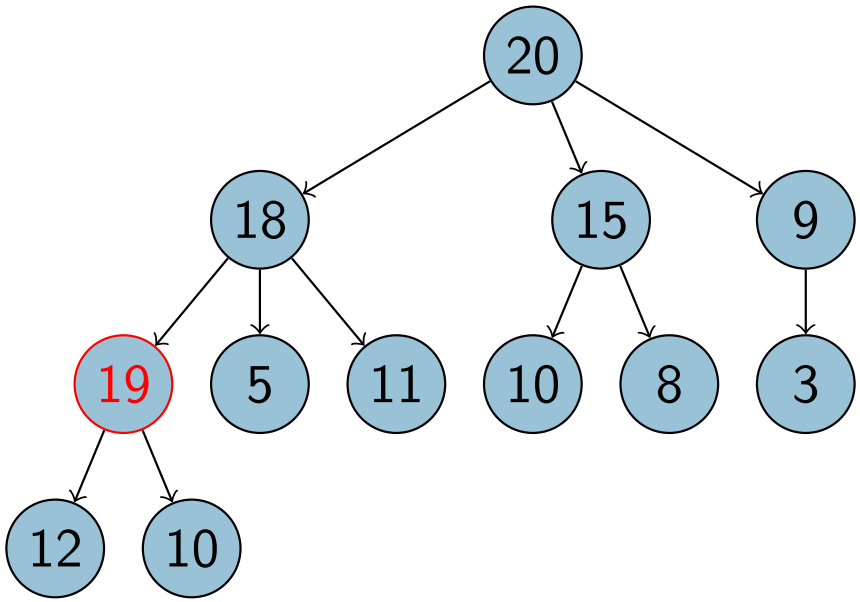
Пресяване нагоре (sift up)

- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. по-голям е от родителя си.
- Как да възстановим пирамидата?



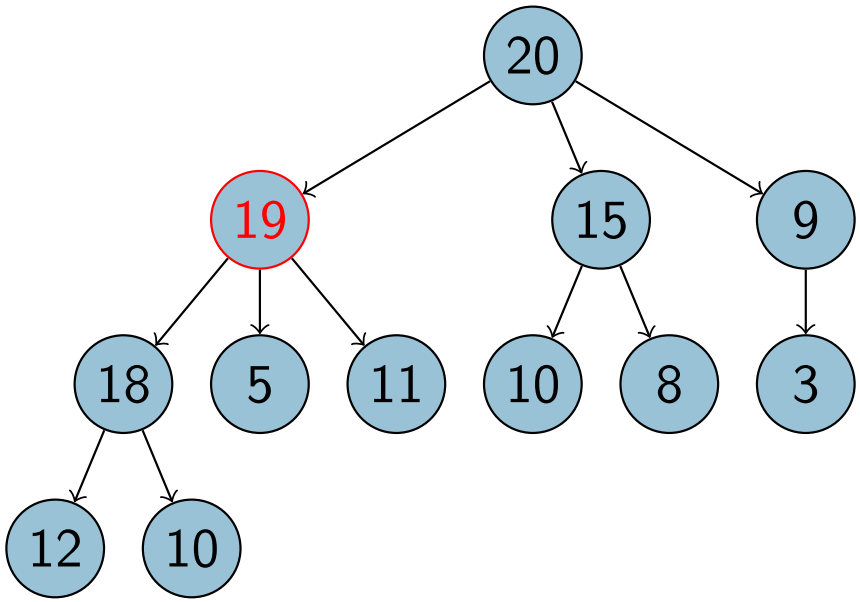
Пресяване нагоре (sift up)

- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. по-голям е от родителя си.
- Как да възстановим пирамидата?
- Идея: разменяме “нарушителя” с родителя си



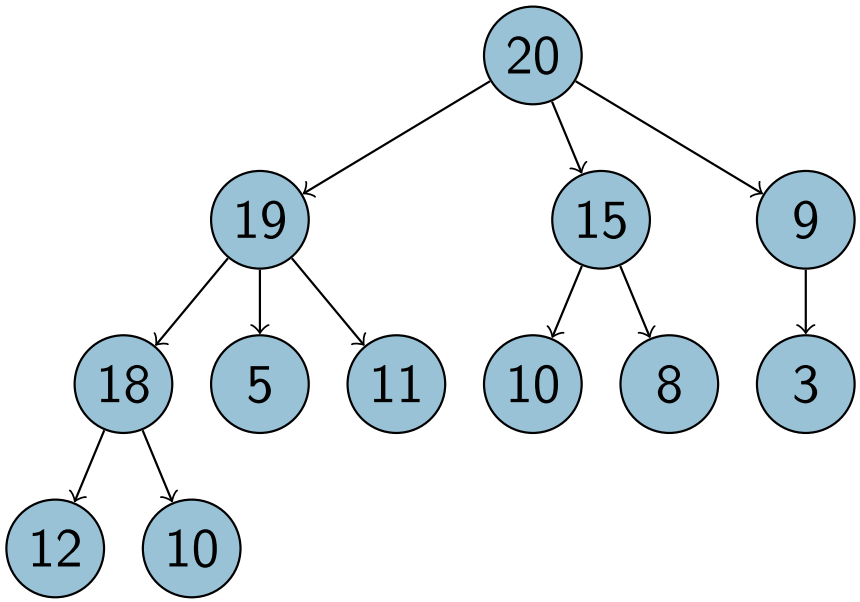
Пресяване нагоре (sift up)

- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. по-голям е от родителя си.
- Как да възстановим пирамидата?
- Идея: разменяме “нарушителя” с родителя си
- Продължаваме докато има нарушение или не стигнем до корена



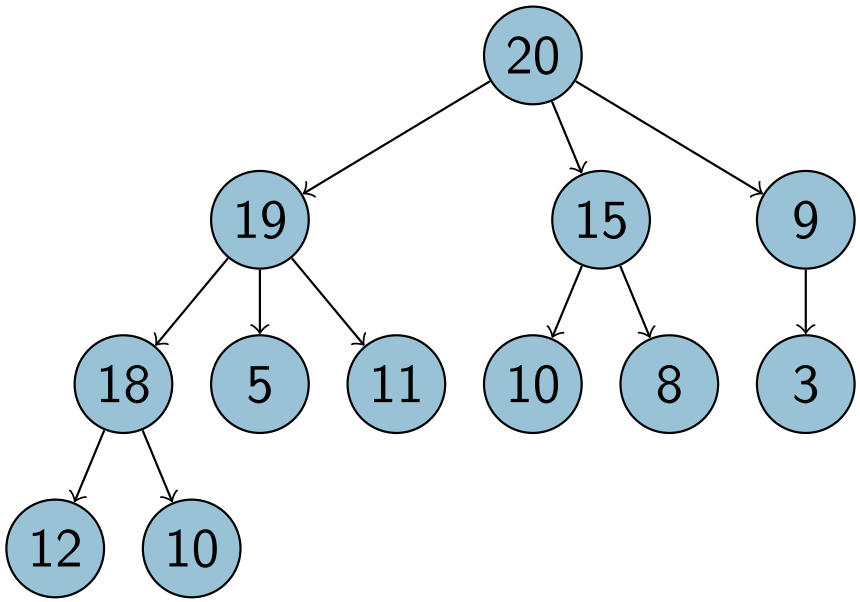
Пресяване нагоре (sift up)

- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. по-голям е от родителя си.
- Как да възстановим пирамидата?
- Идея: разменяме “нарушителя” с родителя си
- Продължаваме докато има нарушение или не стигнем до корена



Пресяване нагоре (sift up)

- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. по-голям е от родителя си.
- Как да възстановим пирамидата?
- Идея: разменяме “нарушителя” с родителя си
- Продължаваме докато има нарушение или не стигнем до корена



Сложност: $O(h)$, където h е височината на пирамидата.

Пирамидата като приоритетна опашка

Операции

- `create()`, `empty()` — $O(1)$
  - съответните операции за дървета

Пирамидата като приоритетна опашка

Операции

- `create()`, `empty()` — $O(1)$
  - съответните операции за дървета
- `enqueue_prioritized(x, p)` — $O(h)$
  - вмъкване на елемента като листо и пресяването му нагоре
  - по възможност без промяна на височината на дървото

Пирамидата като приоритетна опашка

Операции

- `create()`, `empty()` — $O(1)$
  - съответните операции за дървета
- `enqueue_prioritized(x, p)` — $O(h)$
  - вмъкване на елемента като листо и пресяването му нагоре
  - по възможност без промяна на височината на дървото
- `dequeue_highest()` — $O(h)$
  - заместване на корена с някое листо и пресяването му надолу
  - по възможност с листо от най-долно ниво

Пирамидата като приоритетна опашка

Операции

- `create()`, `empty()` — $O(1)$
  - съответните операции за дървета
- `enqueue_prioritized(x, p)` — $O(h)$
  - вмъкване на елемента като листо и пресяването му нагоре
  - по възможност без промяна на височината на дървото
- `dequeue_highest()` — $O(h)$
  - заместване на корена с някое листо и пресяването му надолу
  - по възможност с листо от най-долно ниво
- `head()` — $O(1)$
  - `*root()`

Пирамидата като приоритетна опашка

Операции

- `create()`, `empty()` — $O(1)$
  - съответните операции за дървета
- `enqueue_prioritized(x, p)` — $O(h)$
  - вмъкване на елемента като листо и пресяването му нагоре
  - по възможност без промяна на височината на дървото
- `dequeue_highest()` — $O(h)$
  - заместване на корена с някое листо и пресяването му надолу
  - по възможност с листо от най-долно ниво
- `head()` — $O(1)$
  - `*root()`
- `build(l)`
  - с пресяване нагоре — $O(n \log n)$ (отгоре-надолу)
  - с пресяване надолу — $O(n)$ (отдолу-нагоре)

Пълно двоично дърво

Дефиниция (Пълно двоично дърво)

Двоично дърво, за което:

- всички нива с изключение на последното са пълни
- на последното ниво листата са максимално вляво

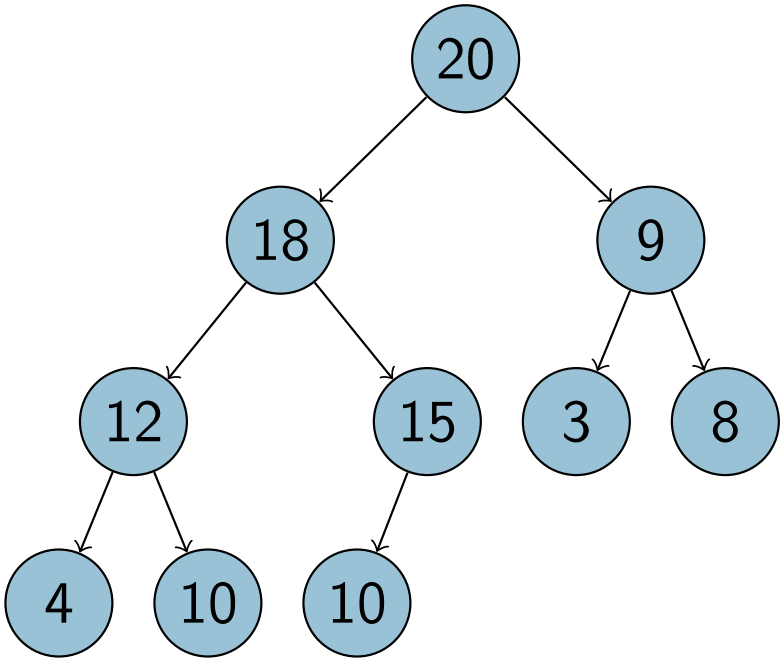
Пълно двоично дърво

Дефиниция (Пълно двоично дърво)

Двоично дърво, за което:

- всички нива с изключение на последното са пълни
- на последното ниво листата са максимално вляво

Пример:



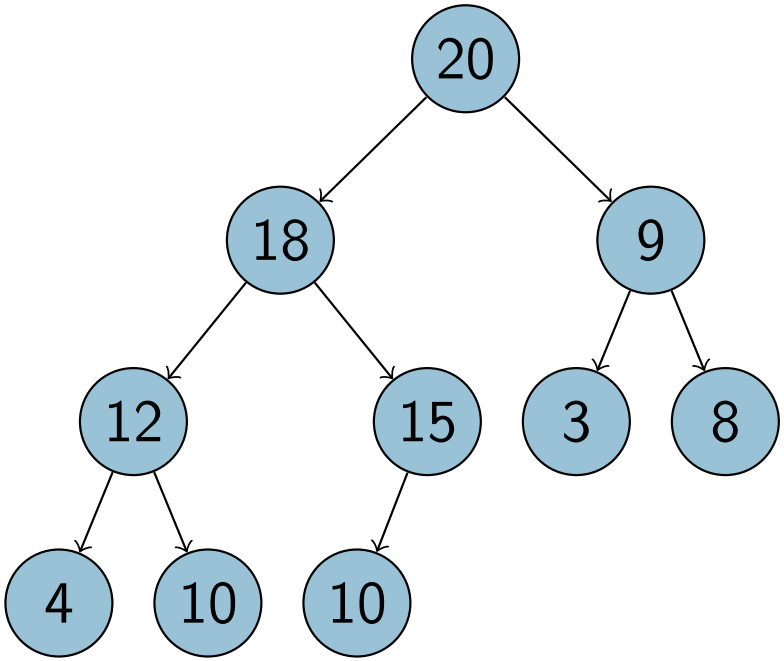
Пълно двоично дърво

Дефиниция (Пълно двоично дърво)

Двоично дърво, за което:

- всички нива с изключение на последното са пълни
- на последното ниво листата са максимално вляво

Пример:



Свойство:

$$2^{h-1} \leq n \leq 2^h - 1$$

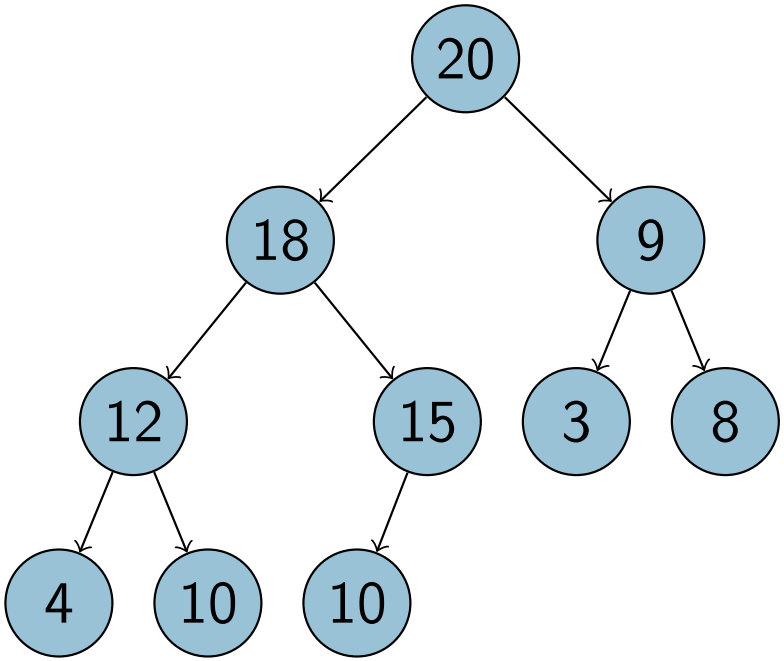
Пълно двоично дърво

Дефиниция (Пълно двоично дърво)

Двоично дърво, за което:

- всички нива с изключение на последното са пълни
- на последното ниво листата са максимално вляво

Пример:



Свойство:

$$2^{h-1} \leq n \leq 2^h - 1$$

$$\log_2(n+1) \leq h \leq \log_2 n + 1$$

Двоична пирамида

Можем да представим пълните двоични дървета чрез масив a, така че

- a[0] е корен

Двоична пирамида

Можем да представим пълните двоични дървета чрез масив a, така че

- a[0] е корен
- a[2*i+1] и a[2*i+2] са ляво и дясно дете на a[i]

Двоична пирамида

Можем да представим пълните двоични дървета чрез масив a, така че

- a[0] е корен
- a[2*i+1] и a[2*i+2] са ляво и дясно дете на a[i]
- съответно a[(j-1)/2] е родителят на a[j]

Двоична пирамида

Можем да представим пълните двоични дървета чрез масив a, така че

- `a[0]` е корен
- `a[2*i+1]` и `a[2*i+2]` са ляво и дясно дете на `a[i]`
- съответно `a[(j-1)/2]` е родителят на `a[j]`

Дефиниция (Двоична пирамида)

Пирамида, която е пълно двоично дърво.

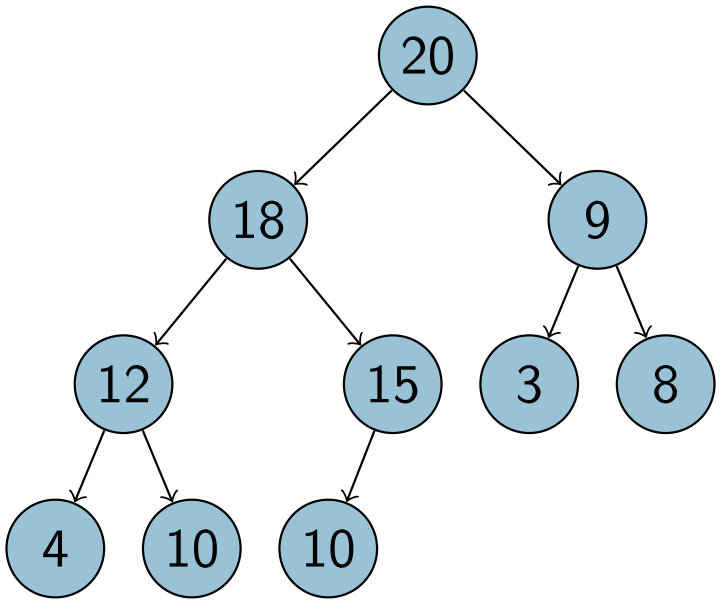
Двоична пирамида

Можем да представим пълните двоични дървета чрез масив a, така че

- `a[0]` е корен
- `a[2*i+1]` и `a[2*i+2]` са ляво и дясно дете на `a[i]`
- съответно `a[(j-1)/2]` е родителят на `a[j]`

Дефиниция (Двоична пирамида)

Пирамида, която е пълно двоично дърво.



Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.

Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.
Алгоритъм:

1. Трансформираме a в пирамида

Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.
Алгоритъм:

1. Трансформираме a в пирамида

2. Разглобяваме пирамидата и получаваме елементите в обратен ред

Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.
Алгоритъм:

1. Трансформираме a в пирамида
   a. Строим пирамидата отдолу-нагоре
2. Разглобяваме пирамидата и получаваме елементите в обратен ред

Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.
Алгоритъм:

1. Трансформираме a в пирамида
   a. Строим пирамидата отдолу-нагоре
   б. Елементите a[n/2],...,a[n-1] са листа и не нарушават пирамидалното свойство
2. Разглобяваме пирамидата и получаваме елементите в обратен ред

Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.
Алгоритъм:

1. Трансформираме a в пирамида
   а) Строим пирамидата отдолу-нагоре
   б) Елементите a[n/2],...,a[n-1] са листа и не нарушават пирамидалното свойство
   в) Обхождаме елементите a[n/2-1],...,a[0] и ги пресяваме надолу
2. Разглобяваме пирамидата и получаваме елементите в обратен ред

Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.
Алгоритъм:

1. Трансформираме a в пирамида
   - а) Строим пирамидата отдолу-нагоре
   - б) Елементите a[n/2],...,a[n-1] са листа и не нарушават пирамидалното свойство
   - в) Обхождаме елементите a[n/2-1],...,a[0] и ги пресяваме надолу
   - г) Сложност: $O(n)$
2. Разглобяваме пирамидата и получаваме елементите в обратен ред

Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.
Алгоритъм:

1. Трансформираме a в пирамида
   - а) Строим пирамидата отдолу-нагоре
   - б) Елементите a[n/2],...,a[n-1] са листа и не нарушават пирамидалното свойство
   - в) Обхождаме елементите a[n/2-1],...,a[0] и ги пресяваме надолу
   - г) Сложност: $O(n)$
2. Разглобяваме пирамидата и получаваме елементите в обратен ред
   - а) Коренът на пирамидата a[0] е най-голямото число в масива

Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.
Алгоритъм:

1. Трансформираме a в пирамида
   а. Строим пирамидата отдолу-нагоре
   б. Елементите a[n/2],...,a[n-1] са листа и не нарушават пирамидалното свойство
   в. Обхождаме елементите a[n/2-1],...,a[0] и ги пресяваме надолу
   г. Сложност: $O(n)$
2. Разглобяваме пирамидата и получаваме елементите в обратен ред
   а. Коренът на пирамидата a[0] е най-голямото число в масива
   б. Разменяме го с последния елемент a[n-1] и вече не го считаме за част от пирамидата

Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.
Алгоритъм:

1. Трансформираме a в пирамида
   - а) Строим пирамидата отдолу-нагоре
   - б) Елементите a[n/2],...,a[n-1] са листа и не нарушават пирамидалното свойство
   - в) Обхождаме елементите a[n/2-1],...,a[0] и ги пресяваме надолу
   - г) Сложност: $O(n)$
2. Разглобяваме пирамидата и получаваме елементите в обратен ред
   - а) Коренът на пирамидата a[0] е най-голямото число в масива
   - б) Разменяме го с последния елемент a[n-1] и вече не го считаме за част от пирамидата
   - в) Пресяваме новия корен надолу

Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.
Алгоритъм:

1. Трансформираме a в пирамида
   - а) Строим пирамидата отдолу-нагоре
   - б) Елементите a[n/2],...,a[n-1] са листа и не нарушават пирамидалното свойство
   - в) Обхождаме елементите a[n/2-1],...,a[0] и ги пресяваме надолу
   - г) Сложност: $O(n)$
2. Разглобяваме пирамидата и получаваме елементите в обратен ред
   - а) Коренът на пирамидата a[0] е най-голямото число в масива
   - б) Разменяме го с последния елемент a[n-1] и вече не го считаме за част от пирамидата
   - в) Пресяваме новия корен надолу
   - г) Повтаряме за елементите a[n-2],...,a[1]

Пирамидално сортиране

Можем да сортираме масив като го превърнем в двоична пирамида.
Алгоритъм:

1. Трансформираме a в пирамида
   - а) Строим пирамидата отдолу-нагоре
   - б) Елементите a[n/2],...,a[n-1] са листа и не нарушават пирамидалното свойство
   - в) Обхождаме елементите a[n/2-1],...,a[0] и ги пресяваме надолу
   - г) Сложност: $O(n)$
2. Разглобяваме пирамидата и получаваме елементите в обратен ред
   - а) Коренът на пирамидата a[0] е най-голямото число в масива
   - б) Разменяме го с последния елемент a[n-1] и вече не го считаме за част от пирамидата
   - в) Пресяваме новия корен надолу
   - г) Повтаряме за елементите a[n-2],...,a[1]
   - д) Сложност: $O(n \log n)$