# 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`

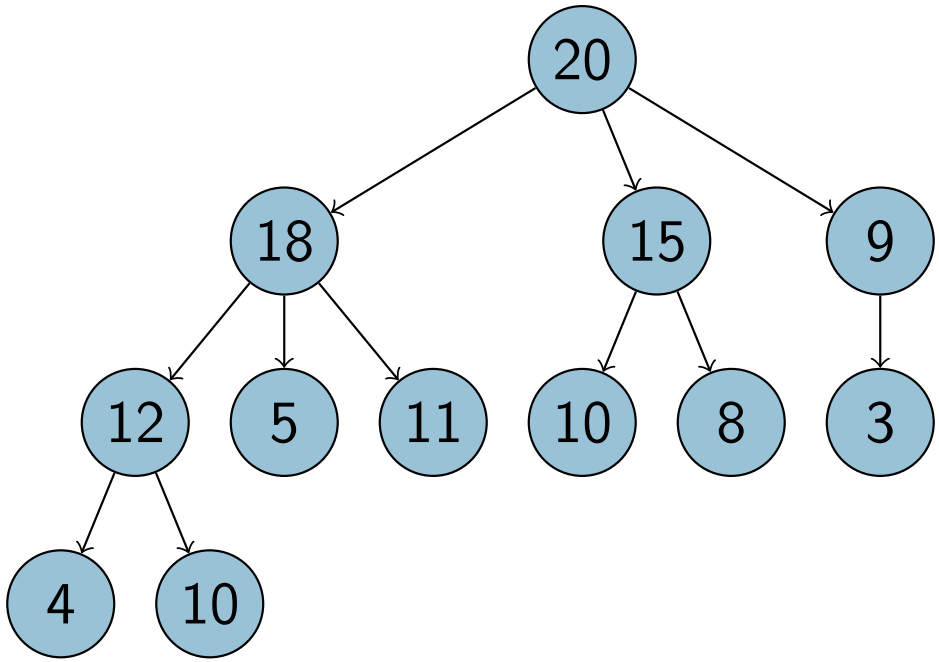
Не е нужно елементите в опашката да са сортирани!

# Пирамида

## Дефиниция (Пирамида)

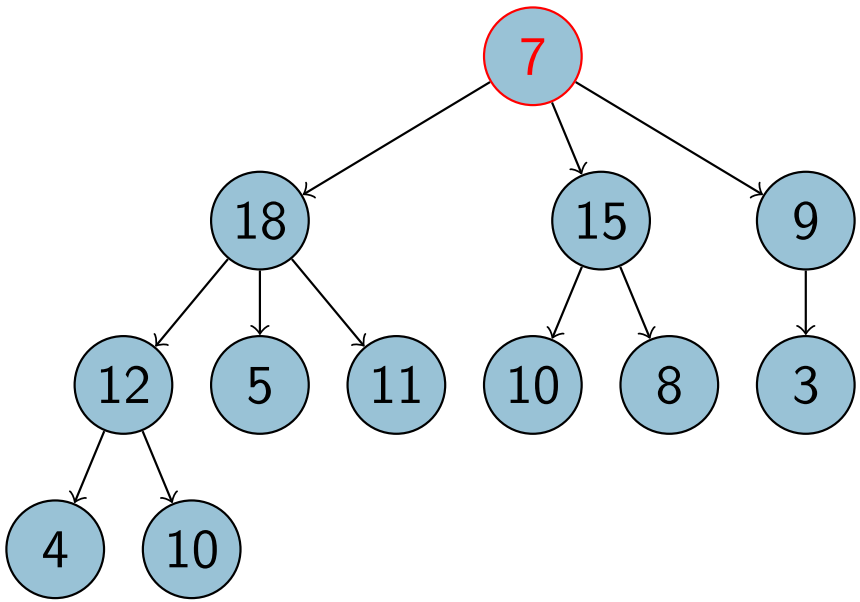
Дърво, в което всеки родител е с по-висок приоритет от децата си.

Пример:



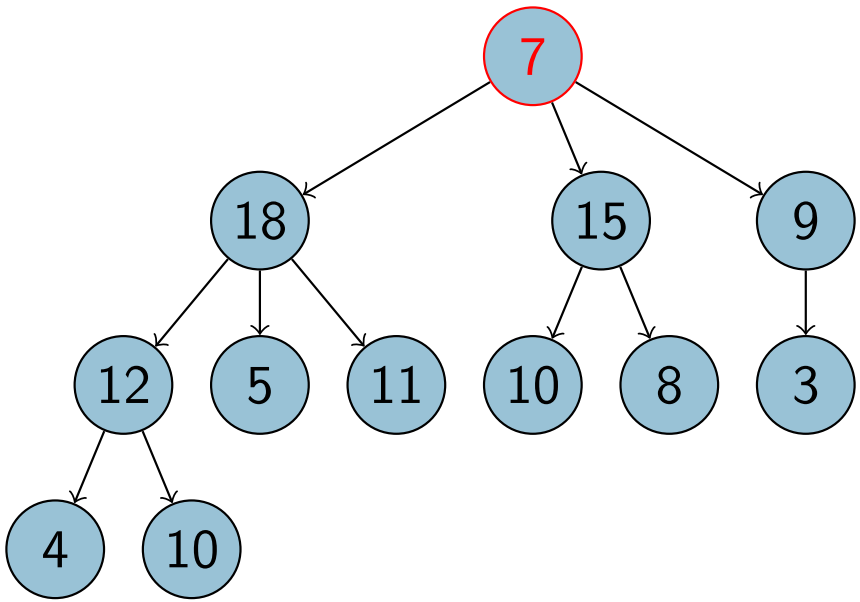
# Пресяване надолу (sift down)

- Да разгледаме елемент, който нарушава пирамидалното свойство



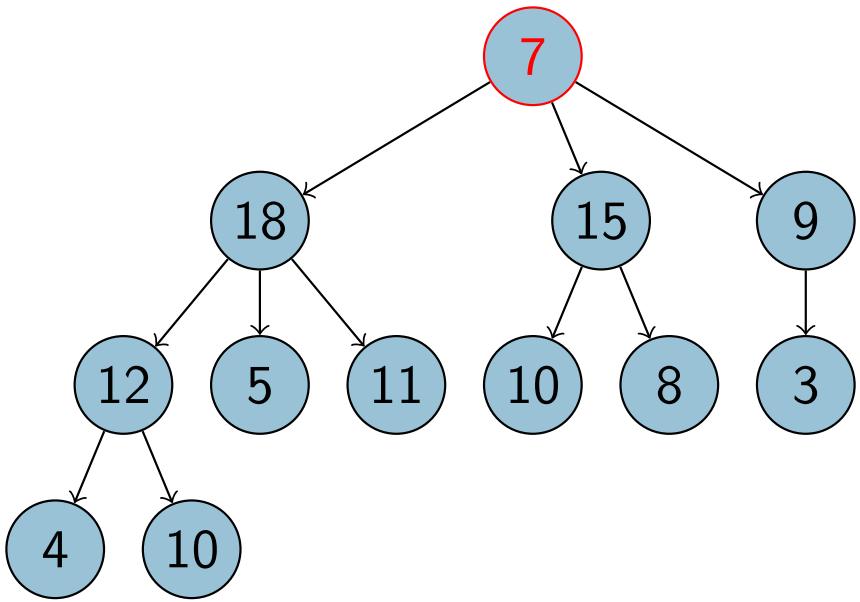
# Пресяване надолу (sift down)

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



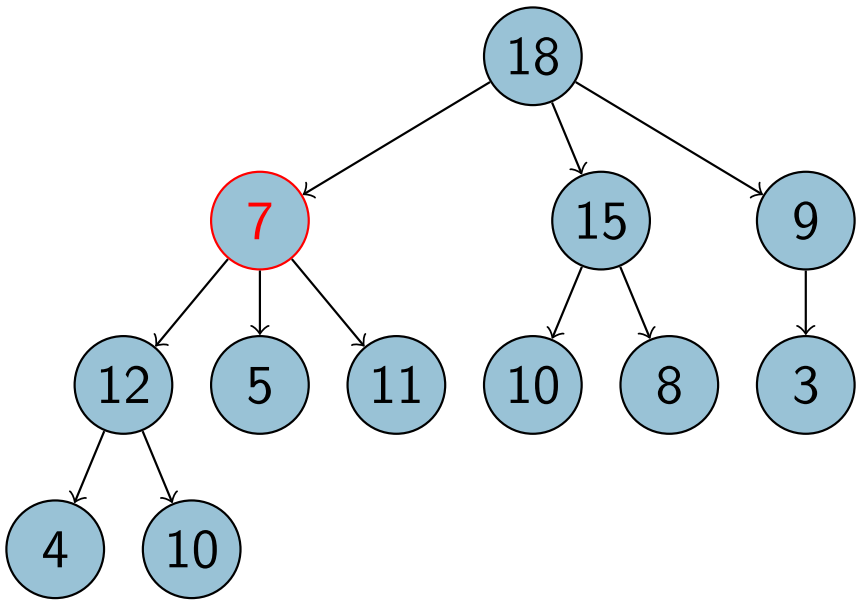
# Пресяване надолу (sift down)

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



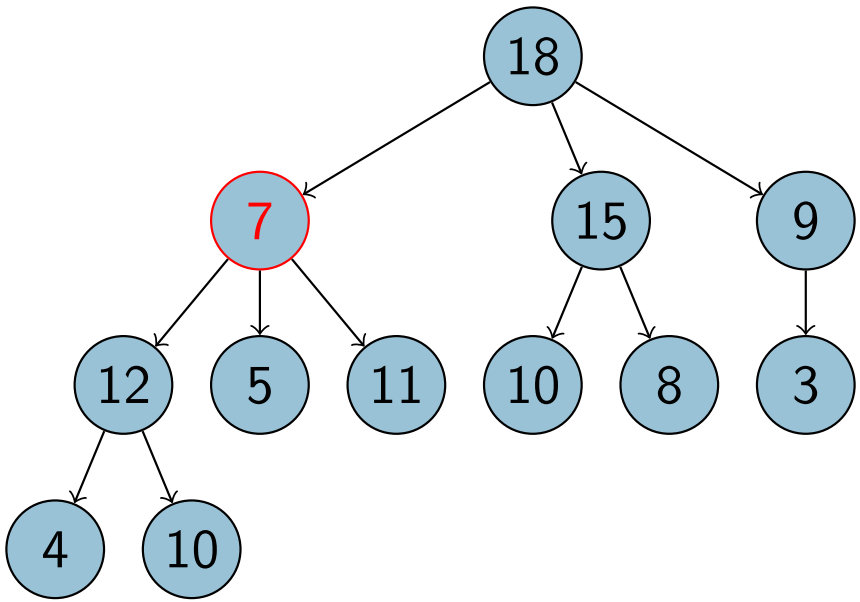
# Пресяване надолу (sift down)

- 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?)



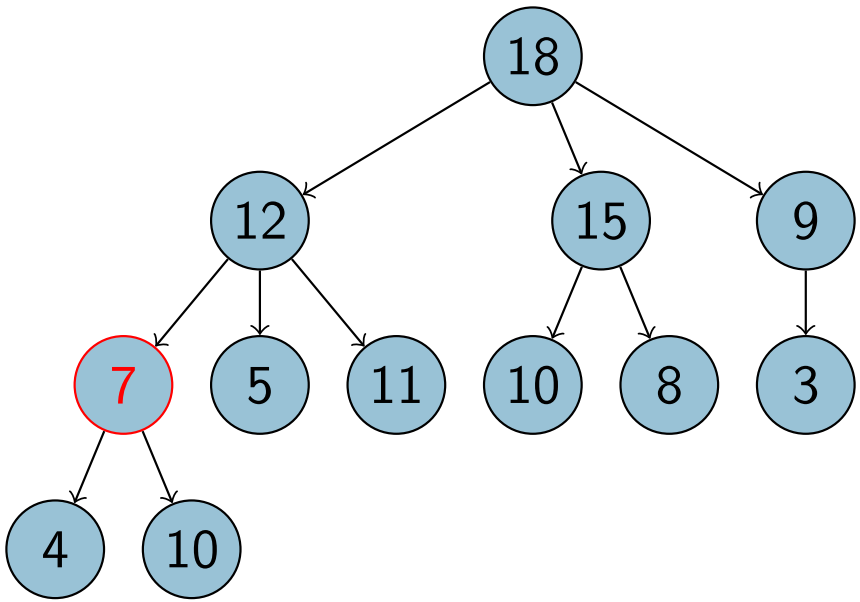
# Пресяване надолу (sift down)

- 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



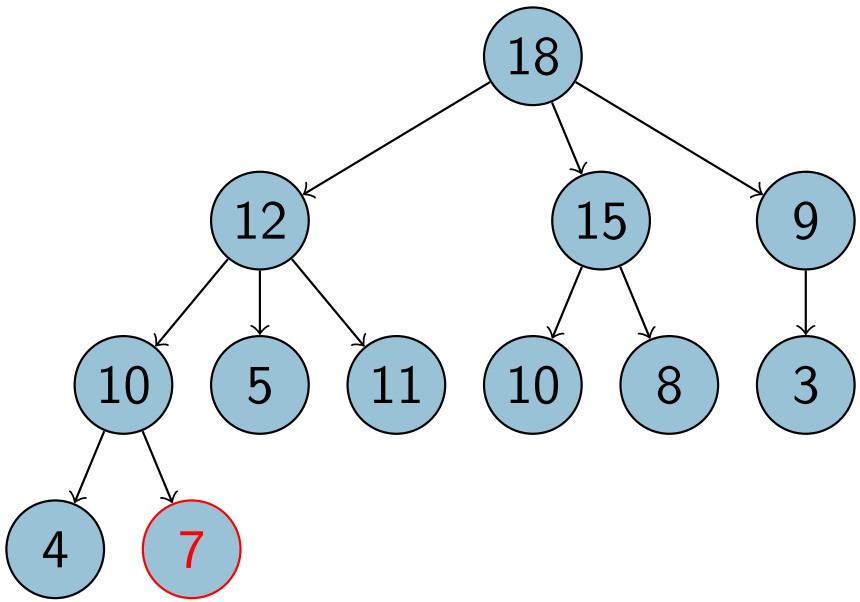
# Пресяване надолу (sift down)

- 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)

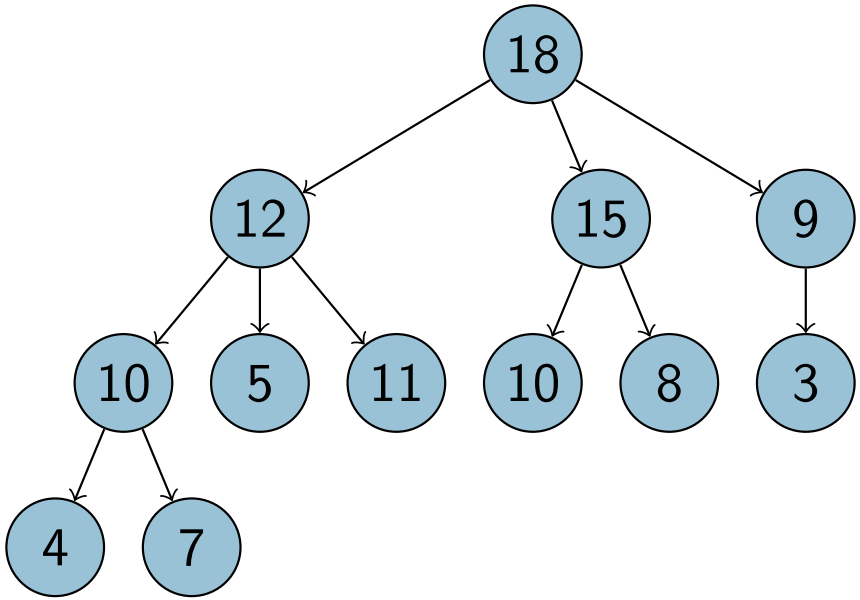
- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. някое от децата му е по-голямо от него.
- Как да възстановим пирамидата?
- Идея: разменяме "нарушителя" с **най-голямото** му дете (защо?)
- Продължаваме докато има нарушение или не стигнем до листо




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

# Пресяване надолу (sift down)

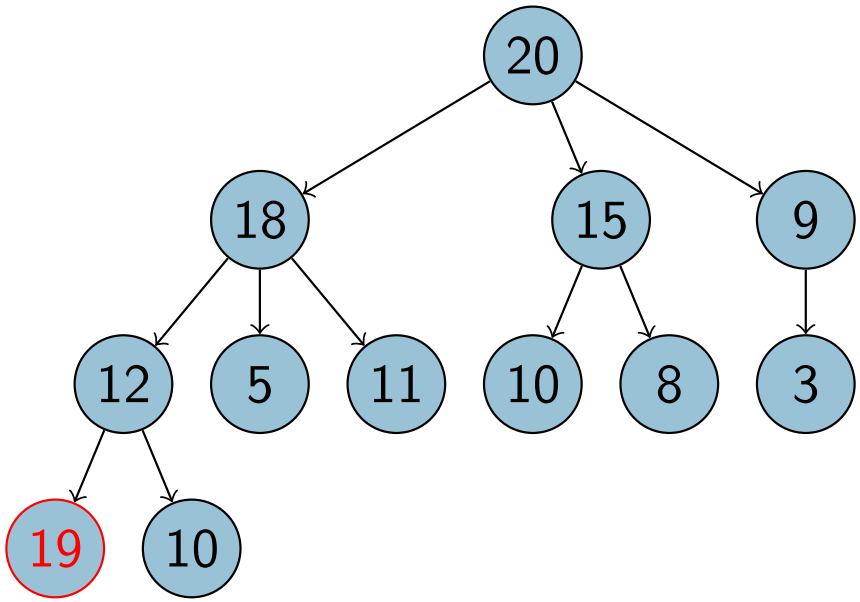
- Да разгледаме елемент, който нарушава пирамидалното свойство
  - т.е. някое от децата му е по-голямо от него.
- Как да възстановим пирамидата?
- Идея: разменяме "нарушителя" с **най-голямото** му дете (защо?)
- Продължаваме докато има нарушение или не стигнем до листо
- Листата никога не нарушават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




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

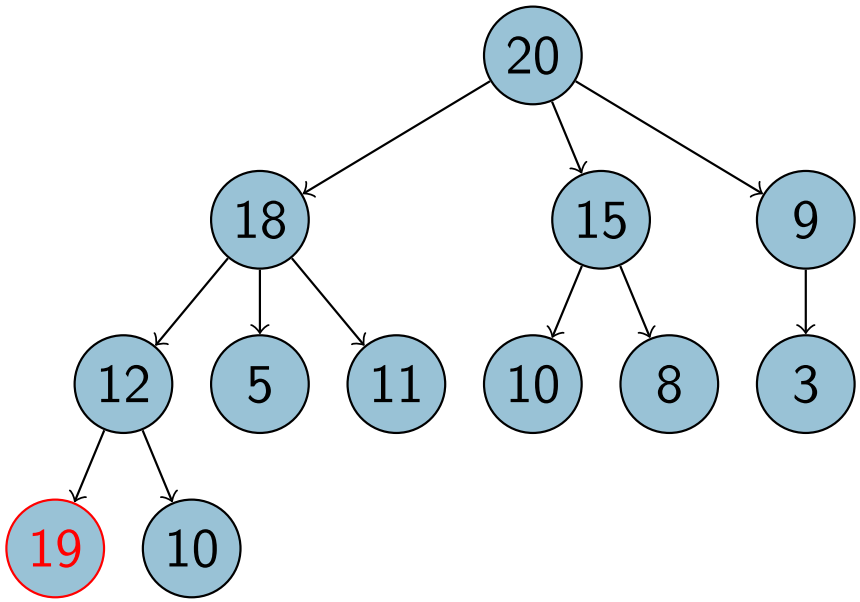
# Пресяване нагоре (sift up)

- Да разгледаме елемент, който нарушава пирамидалното свойство



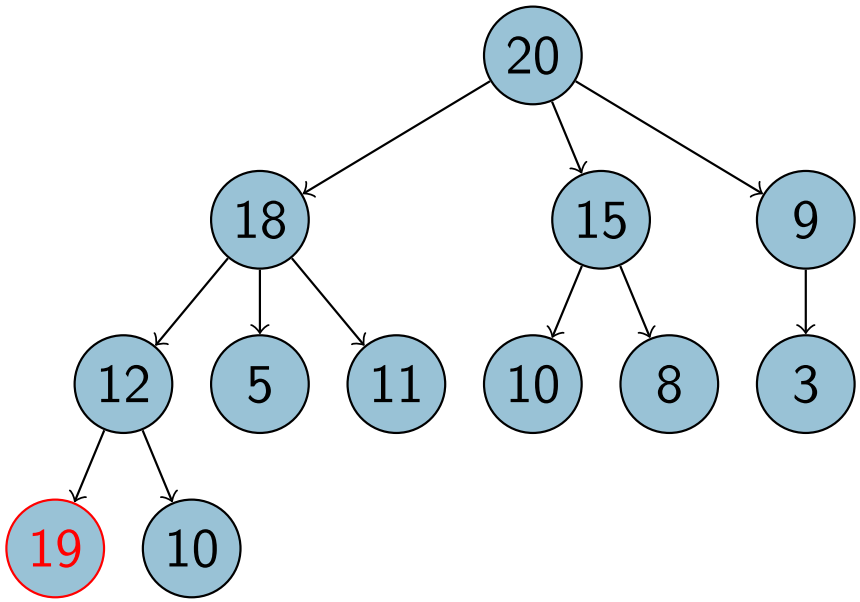
# Пресяване нагоре (sift up)

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



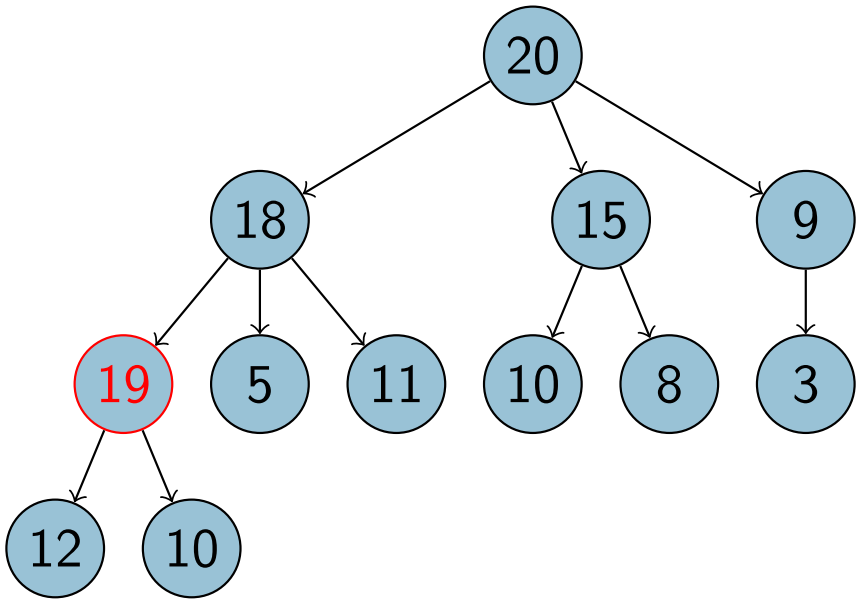
# Пресяване нагоре (sift up)

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



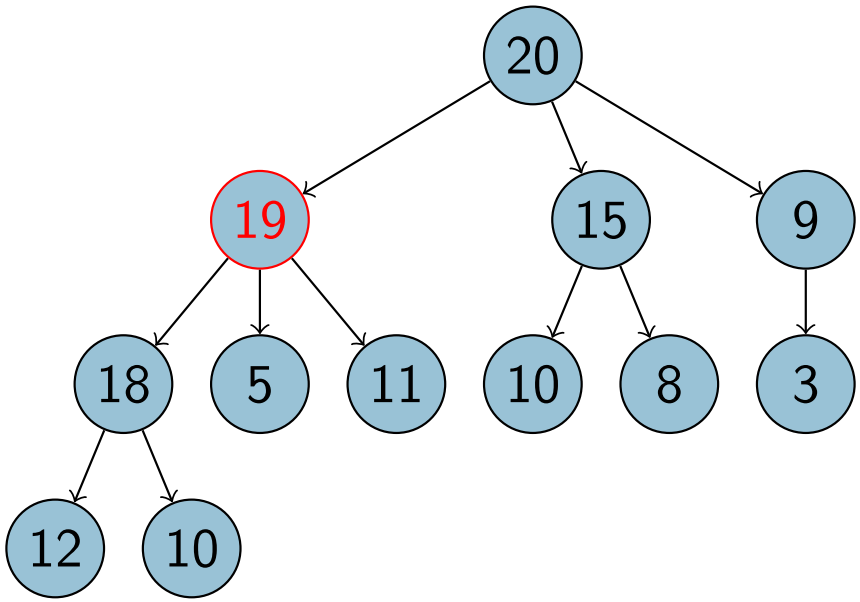
# Пресяване нагоре (sift up)

- 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



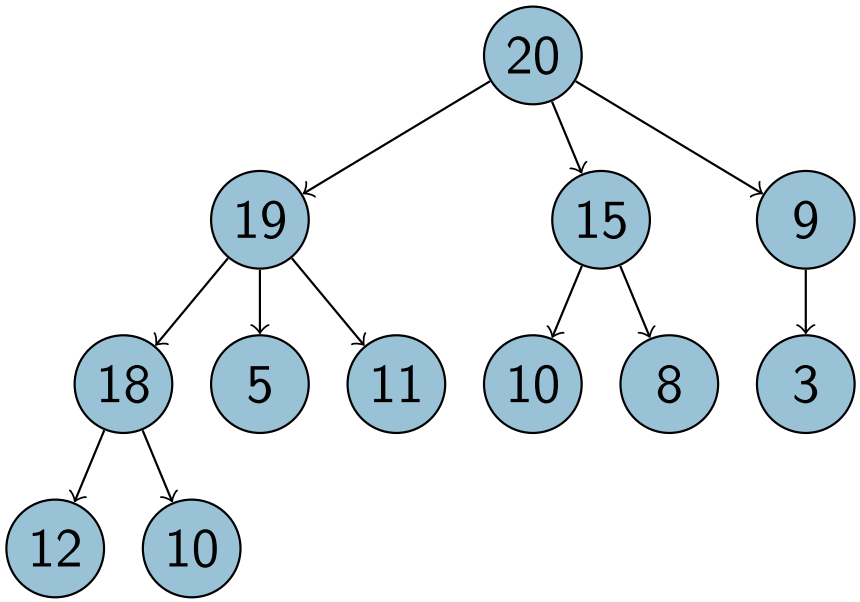
# Пресяване нагоре (sift up)

- 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



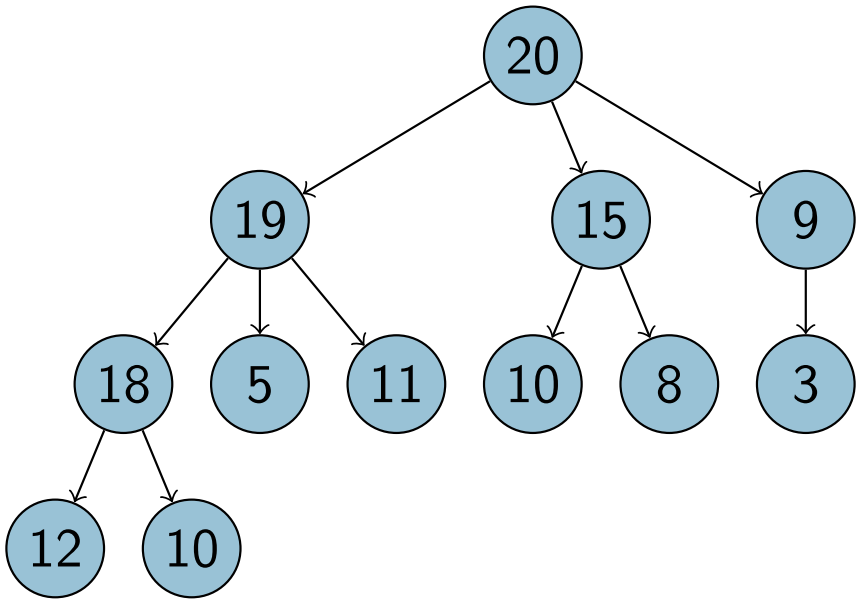
# Пресяване нагоре (sift up)

- 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 надолу — $O(n \log n)$ (отгоре-надолу)
  - с пресяване нагоре — $O(n)$ (отдолу-нагоре)