Priority Queues with Decreasing Keys

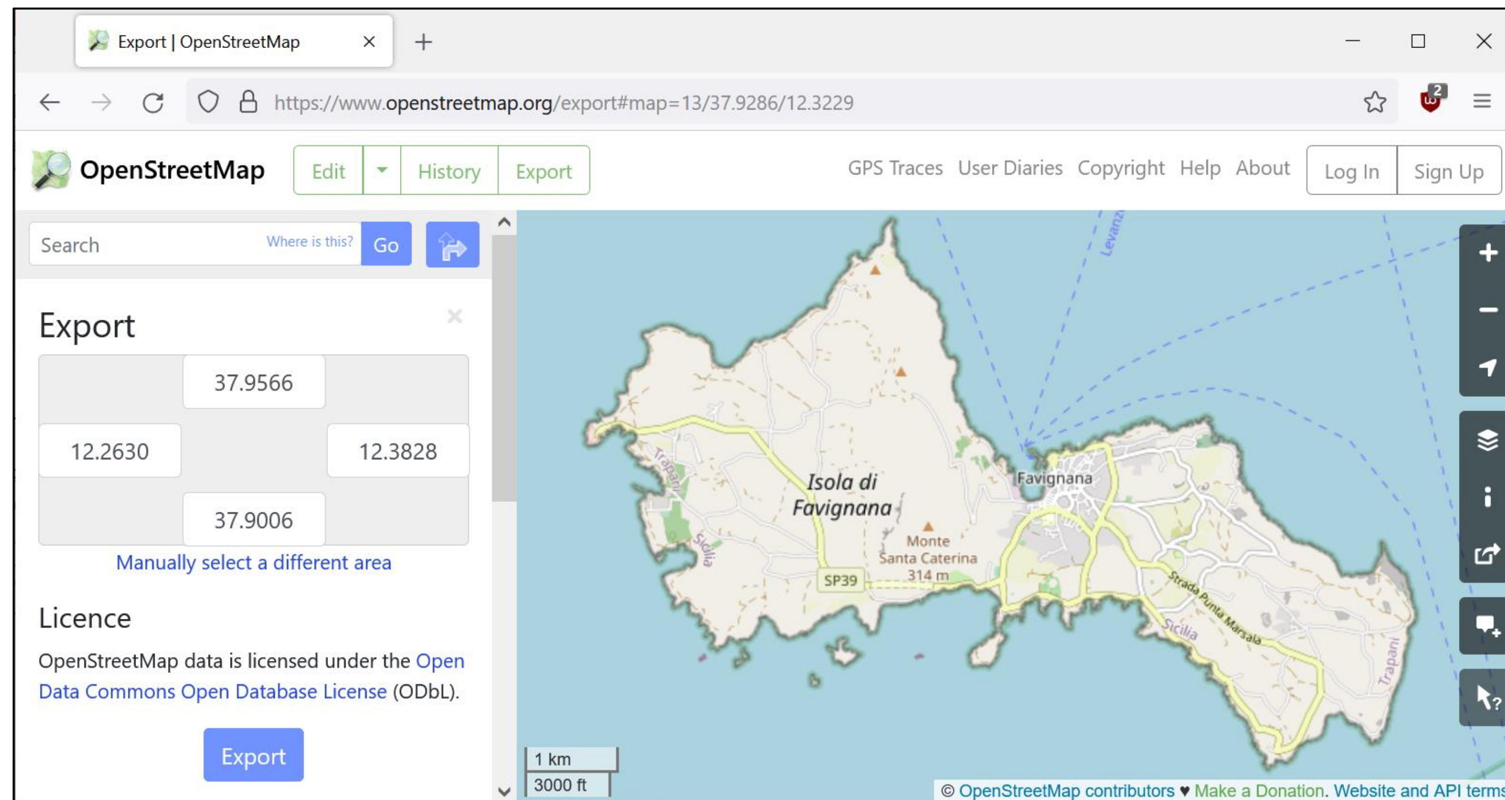
Gerth Stølting Brodal

Aarhus University

11th FUN with Algorithms, Island of Favignana, Sicily, Italy, May 30-June 3, 2022

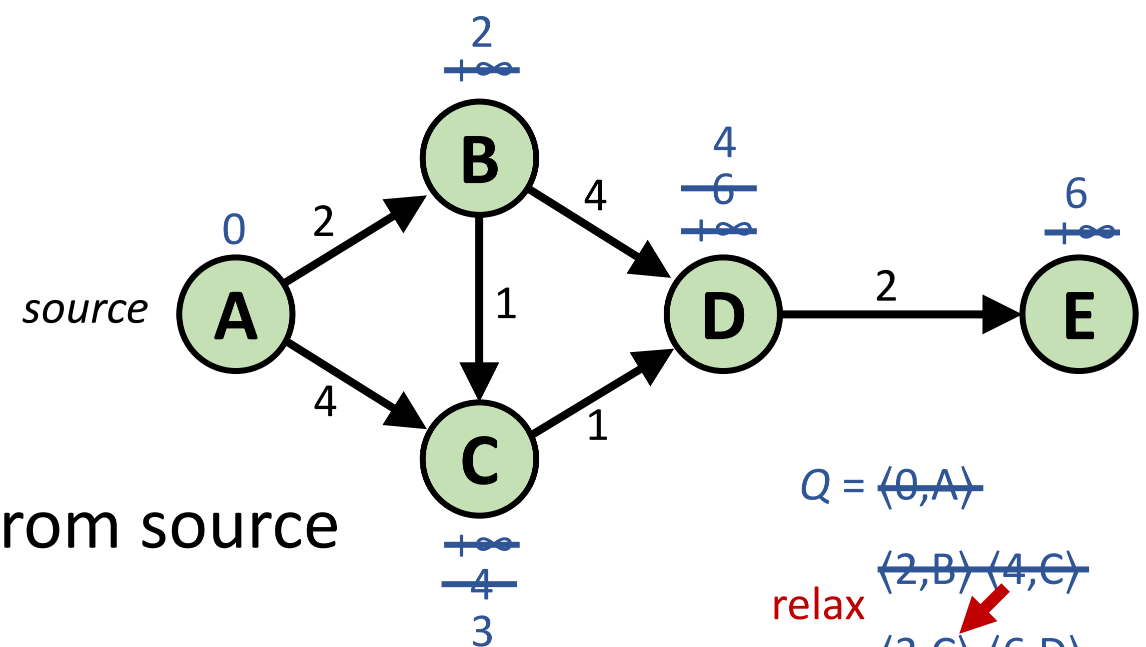
Background

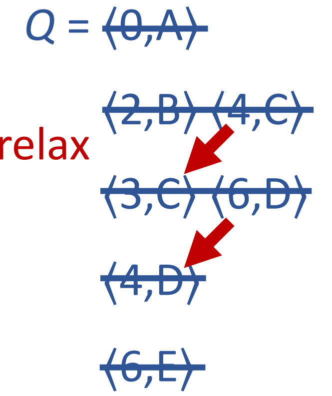
- Bachelorproject = shortest paths on Open Street Map graphs
- Students have trouble implementing Dijkstra's algorithm in Java™



Dijkstra’s algorithm

- Non-negative edge weights
- Visits nodes in increasing distance from source





proc Dijkstra$_1(V, E, \delta, s)$
  $dist[v] = +\infty$ for all $v \in V \setminus \{s\}$
  $dist[s] = 0$
  Insert$(Q, \langle dist[s], s\rangle)$
  while $Q \neq \emptyset$ **do**
    $\langle d, u\rangle =$ ExtractMin(Q)
    for $(u, v) \in E \cap (\{u\} \times V)$ **do**
      if $dist[u] + \delta(u, v) < dist[v]$ **then**
        $dist[v] = dist[u] + \delta(u, v)$
        if $v \in Q$ **then**
          DecreaseKey$(Q, v, dist[v])$
        else
          Insert$(Q, \langle v, dist[v]\rangle)$
  return $dist$

relax

Fibonacci heaps
⇒ $O(m + n \cdot \log n)$

proc Dijkstra$_2(V, E, \delta, s)$
  $dist[v] = +\infty$ for all $v \in V \setminus \{s\}$
  $dist[s] = 0$
  Insert$(Q, \langle dist[s], s\rangle)$
  while $Q \neq \emptyset$ **do**
    $\langle d, u\rangle =$ ExtractMin(Q)
    for $(u, v) \in E \cap (\{u\} \times V)$ **do**
      if $dist[u] + \delta(u, v) < dist[v]$ **then**
        $dist[v] = dist[u] + \delta(u, v)$
        if $v \in Q$ **then**
          Remove(Q, v)
        Insert$(Q, \langle dist[v], v\rangle)$
  return $dist$

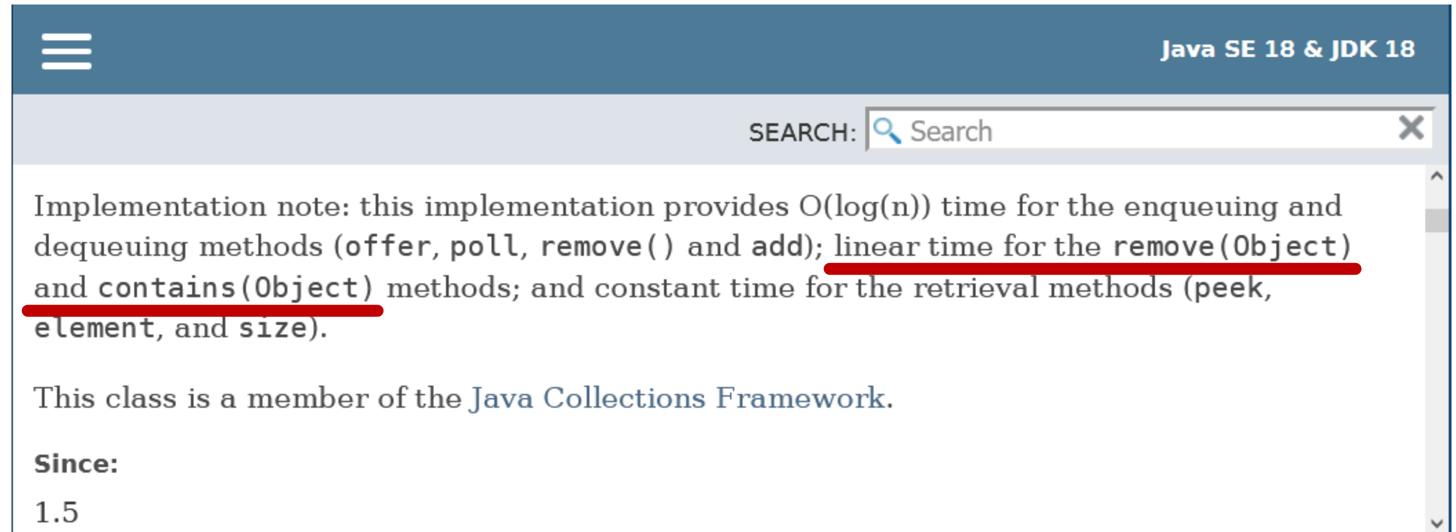
$O(\log n)$ Remove
⇒ $O(m \cdot \log n)$

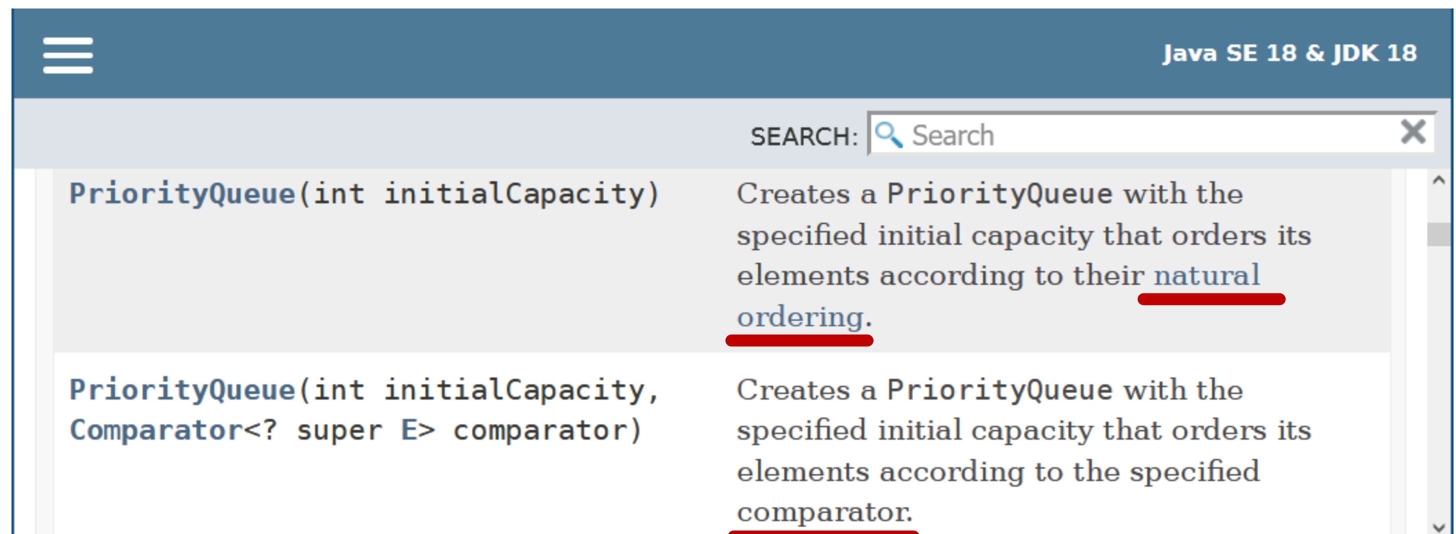
The Challenge - Java's builtin binary heap

- no `decreasekey`
- `remove` $O(n)$ time

  $\Rightarrow$ Dijkstra $O(m \cdot n)$

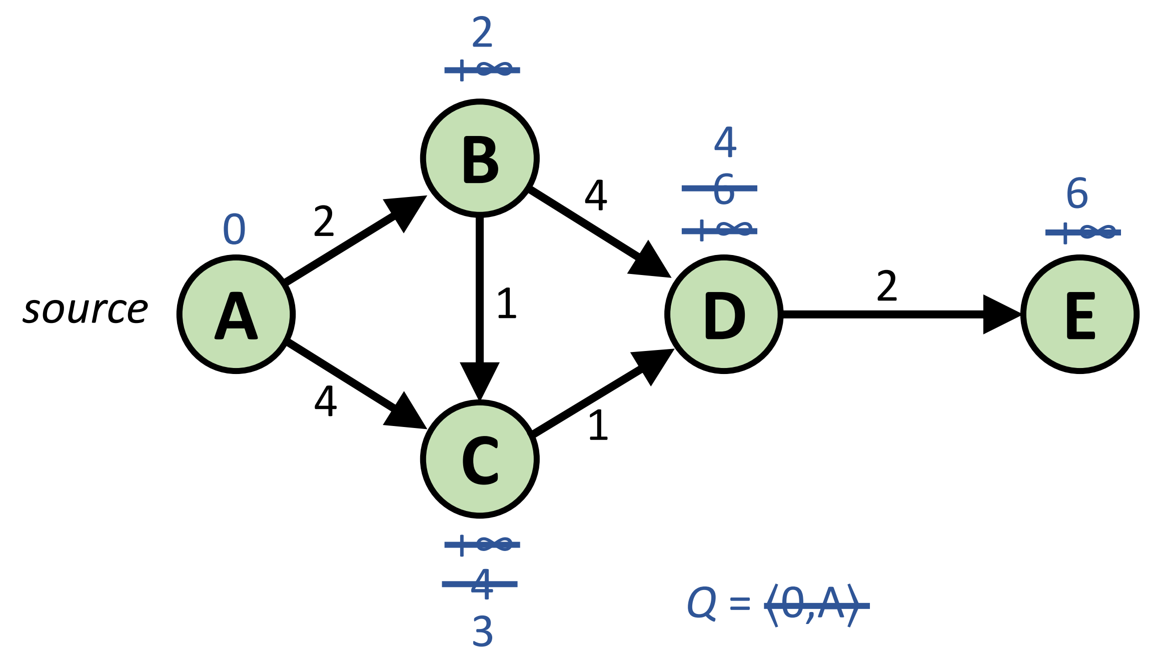
- comparator function





Repeated insertions

- Relax inserts new copies of item
- Skip outdated items



Q = ⟨0,A⟩

⟨2,B⟩ ⟨4,C⟩

⟨3,C⟩ ⟨4,C⟩ ⟨6,D⟩

⟨4,C⟩ ⟨4,D⟩ ⟨6,D⟩

⟨4,D⟩ ⟨6,D⟩

⟨6,D⟩ ⟨6,E⟩

⟨6,E⟩

proc $\texttt{Dijkstra}_3(V, E, \delta, s)$

  $\mathit{dist}[v] = +\infty$ for all $v \in V \setminus \{s\}$

  $\mathit{dist}[s] = 0$

  $\texttt{Insert}(Q, \langle \mathit{dist}[s], s \rangle)$

  while $Q \neq \emptyset$ **do**

    $\langle d, u \rangle = \texttt{ExtractMin}(Q)$

outdated ? → **if** $d = \mathit{dist}[u]$ **then**

      for $(u, v) \in E \cap (\{u\} \times V)$ **do**

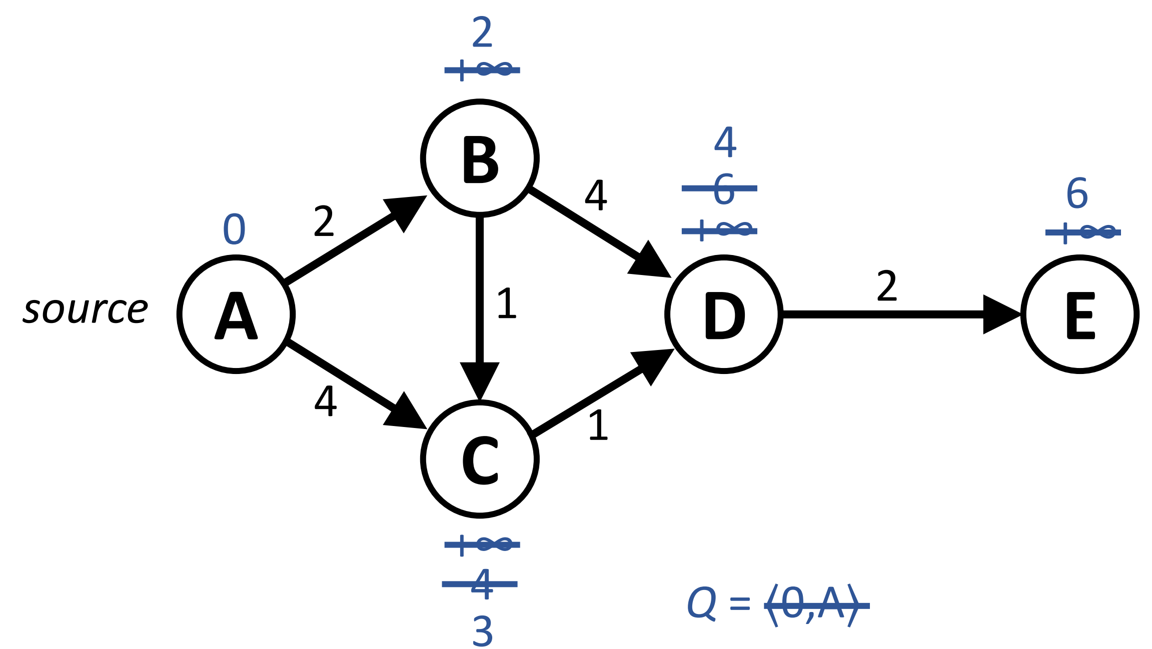
        if $\mathit{dist}[u] + \delta(u, v) < \mathit{dist}[v]$ **then**

          $\mathit{dist}[v] = \mathit{dist}[u] + \delta(u, v)$

relax = reinsert → $\texttt{Insert}(Q, \langle \mathit{dist}[v], v \rangle)$

  return dist

Using a visited set



Q = ~~⟨0,A⟩~~

~~⟨2,B⟩ ⟨4,C⟩~~

~~⟨3,C⟩ ⟨4,C⟩ ⟨6,D⟩~~

~~⟨4,C⟩ ⟨4,D⟩ ⟨6,D⟩~~

~~⟨4,D⟩ ⟨6,D⟩~~

~~⟨6,D⟩ ⟨6,E⟩~~

~~⟨6,E⟩~~

proc $\texttt{Dijkstra}_4(V, E, \delta, s)$

$\quad dist[v] = +\infty$ for all $v \in V \setminus \{s\}$

$\quad dist[s] = 0$

$\quad visited = \emptyset$

$\quad \texttt{Insert}(Q, \langle dist[s], s\rangle)$

$\quad$ **while** $Q \neq \emptyset$ **do**

$\quad\quad \langle d, u\rangle = \texttt{ExtractMin}(Q)$

use bitvector → $\quad\quad$ **if** $u \notin visited$ **then**

$\quad\quad\quad visited = visited \cup \{u\}$

$\quad\quad\quad$ **for** $(u, v) \in E \cap (\{u\} \times V)$ **do**

$\quad\quad\quad\quad$ **if** $dist[u] + \delta(u, v) < dist[v]$ **then**

$\quad\quad\quad\quad\quad dist[v] = dist[u] + \delta(u, v)$

$\quad\quad\quad\quad\quad \texttt{Insert}(Q, \langle dist[v], v\rangle)$

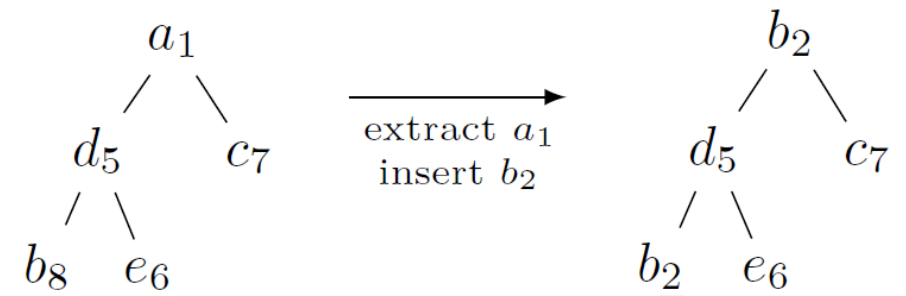
$\quad$ **return** $dist$

A shaky idea...

```
proc Dijkstra_4(V, E, δ, s)
  dist[v] = +∞ for all v ∈ V \ {s}
  dist[s] = 0
  visited = ∅
  Insert(Q, ⟨dist[s], s⟩)
  while Q ≠ ∅ do
    ⟨d, u⟩ = ExtractMin(Q)
    if u ∉ visited then
      visited = visited ∪ {u}
      for (u, v) ∈ E ∩ ({u} × V) do
        if dist[u] + δ(u, v) < dist[v] then
          dist[v] = dist[u] + δ(u, v)
          Insert(Q, ⟨dist[v], v⟩)
  return dist
```

d never used → (the d in $\langle d, u\rangle$ and the $dist[v]$ in $Insert(Q, \langle dist[v], v\rangle)$ are crossed out)

- Only store nodes in Q (save space)
- Comparator
- Key = current distance $dist$



Heap invariants break

Experimental study

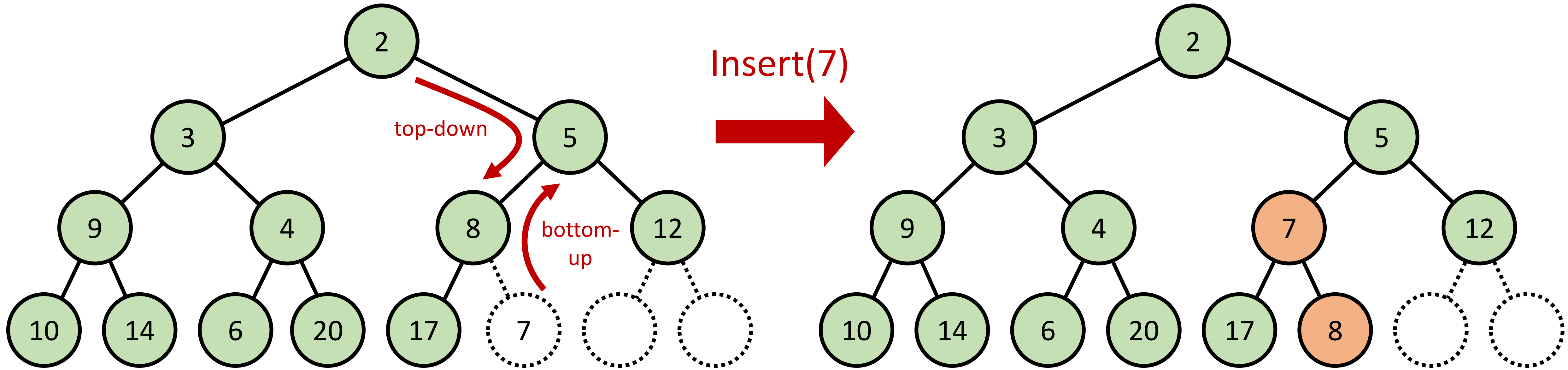
```
visited = set()
Q = Queue()
Q.insert(Item(0, source))
while not Q.empty():
    u = Q.extract_min().value
    if u not in visited:
        visited.add(u)
        for v in G.out[u]:
            dist_v = dist[u] + G.weights[(u, v)]
            if dist_v < dist[v]:
                dist[v] = dist_v
                parent[v] = u
                Q.insert(Item(dist[v], v))
```

- Implemented Dijkstra_4 in Python
- Stress test on random cliques
- Binary heaps failed (default priority queue in Java and Python)
- Skew heaps worked
- Leftist heaps worked
- Pairing heaps worked
- Binomial queues worked
- Post-order heaps worked
- Binary heaps with top-down insertions worked

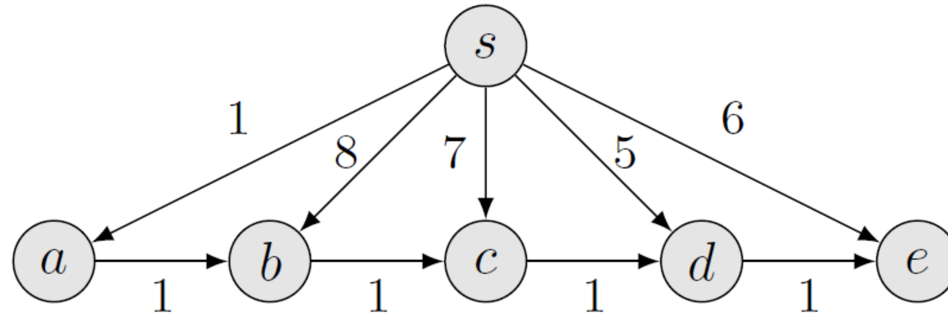
Pointer based: Skew heaps, Leftist heaps, Pairing heaps, Binomial queues

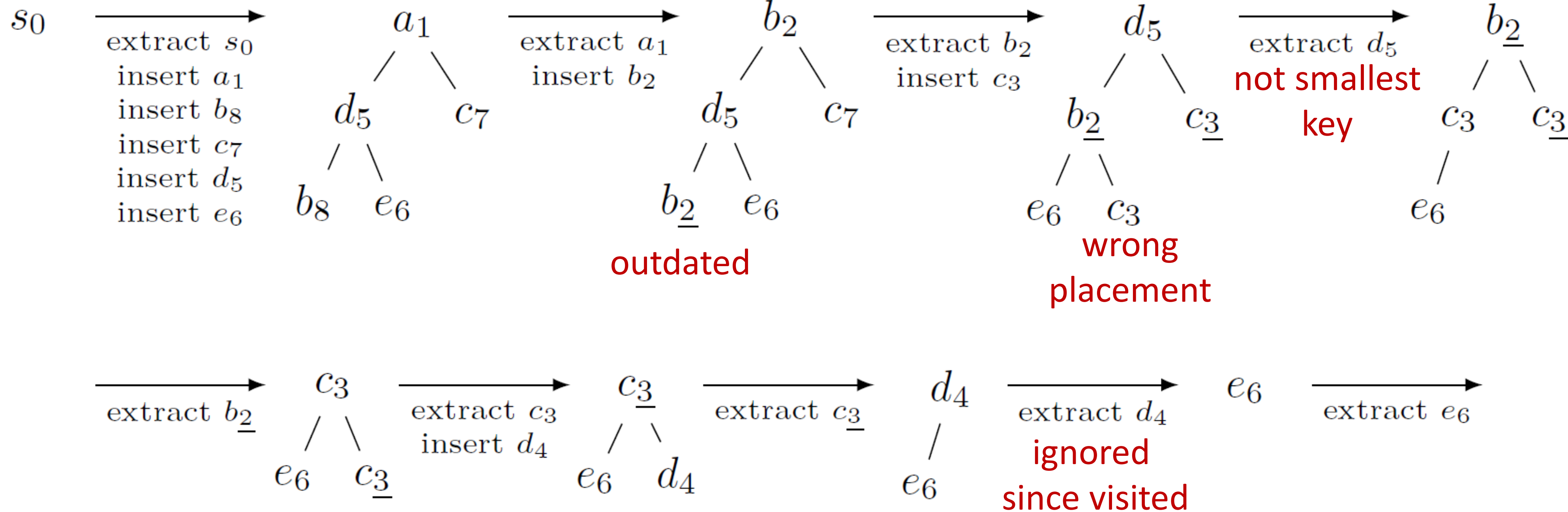
Implicit (space efficient): Post-order heaps, Binary heaps with top-down insertions

Binary heap insertions – bottom-up vs top-down



Binary heaps using *dist* in a comparator fails





Definition
Priority Queues with Decreasing Keys

- Items = ⟨key, value⟩
- Over time keys can decrease – *priority queue is not informed*
- Items are compared w.r.t. their current keys
- The original key of an item is the key when it was inserted

`Insert`(item)

`ExtractMin()` returns an item with current key less than or equal to all original keys in the priority queue

Theorem 1

`Dijkstra`$_4$ correctly computes shortest paths when using $dist$ as current key and a priority queue supporting decreasing keys

Theorem 2

The following priority queues support decreasing keys (out of the box)

- binary heaps with top-down insertions
- skew heaps
- leftist heaps
- pairing heaps
- binomial queues
- post-order heaps

Proof of Theorem 2 - Basic idea

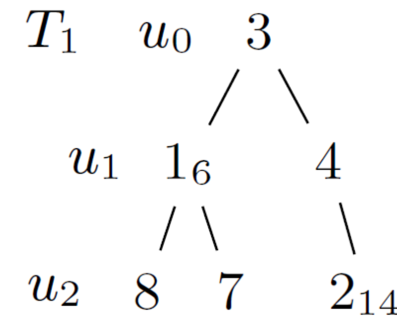
- Decreased heap order

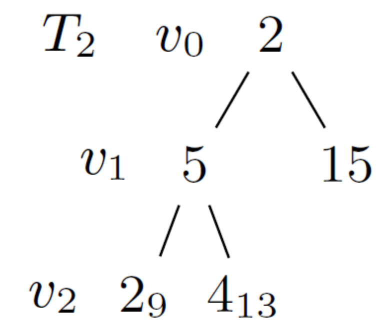
  u ancestor of *v* ⇒
  current key *u* ≤ original key *v*

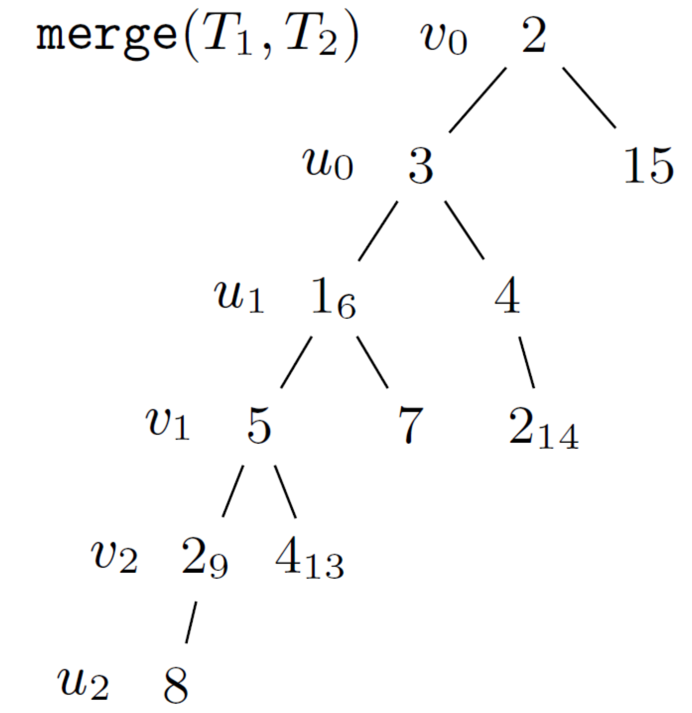
- Root valid item to extract

- Top-down merging two paths preserves decreased heap order

⇒ skew heaps and leftist heaps support decreasing keys



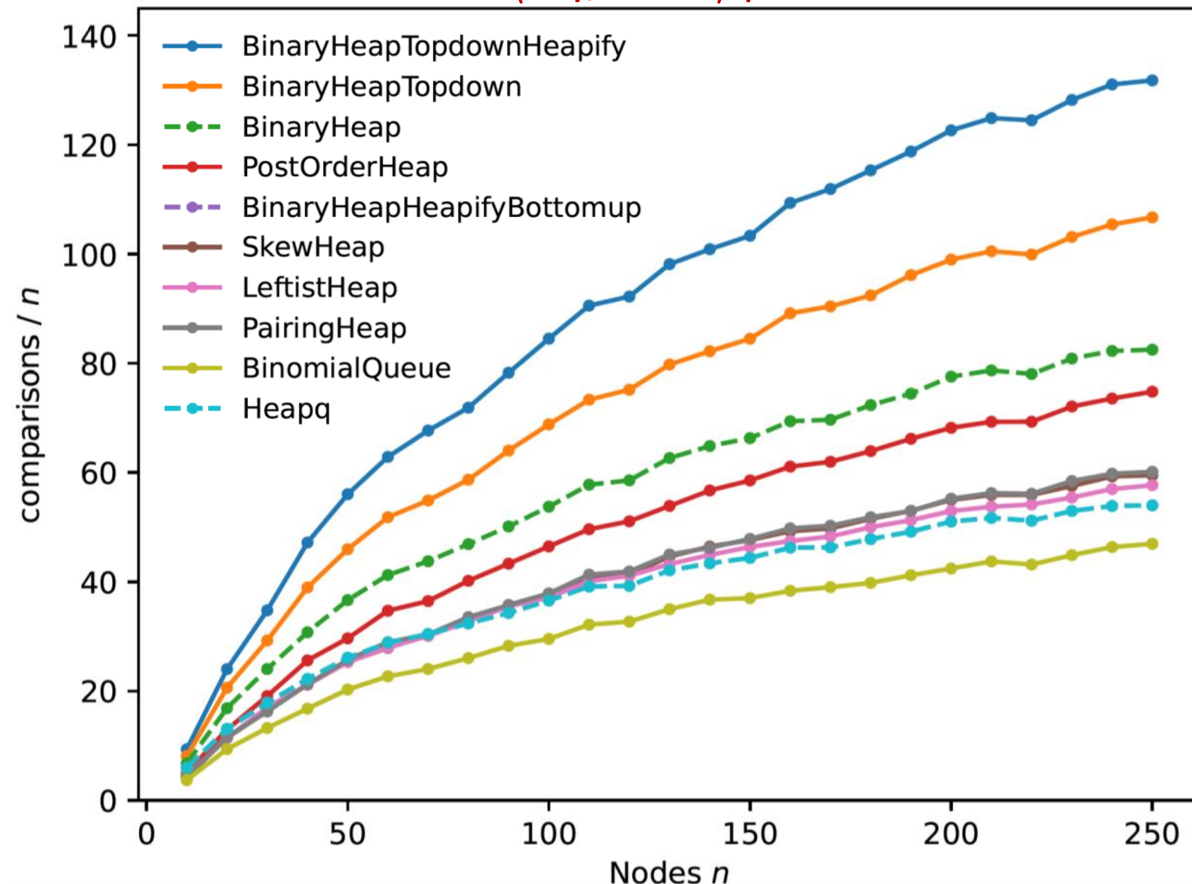




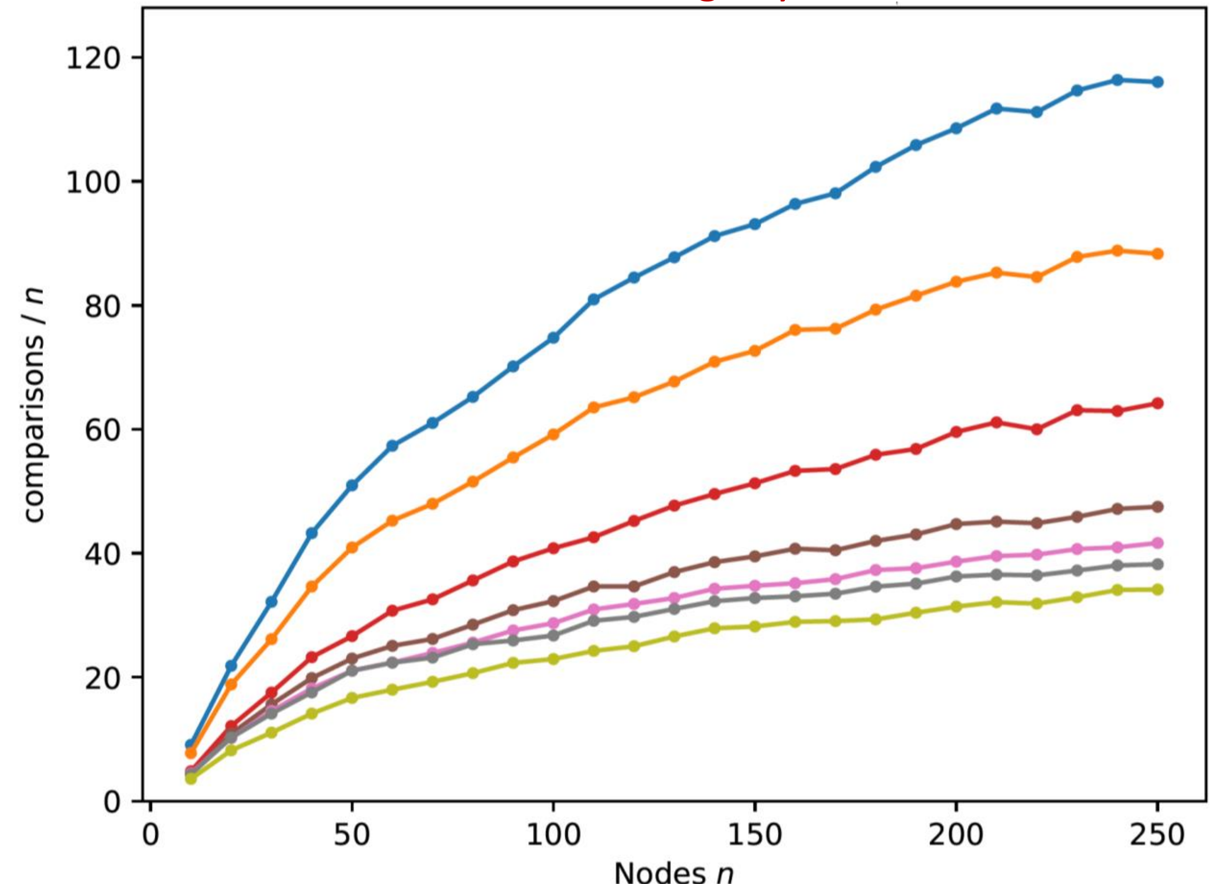
Experimental evaluation of various heaps

- Cliques with uniform random weights
- With decreasing keys less comparisons (outdated items removed earlier)

⟨key, value⟩ pairs

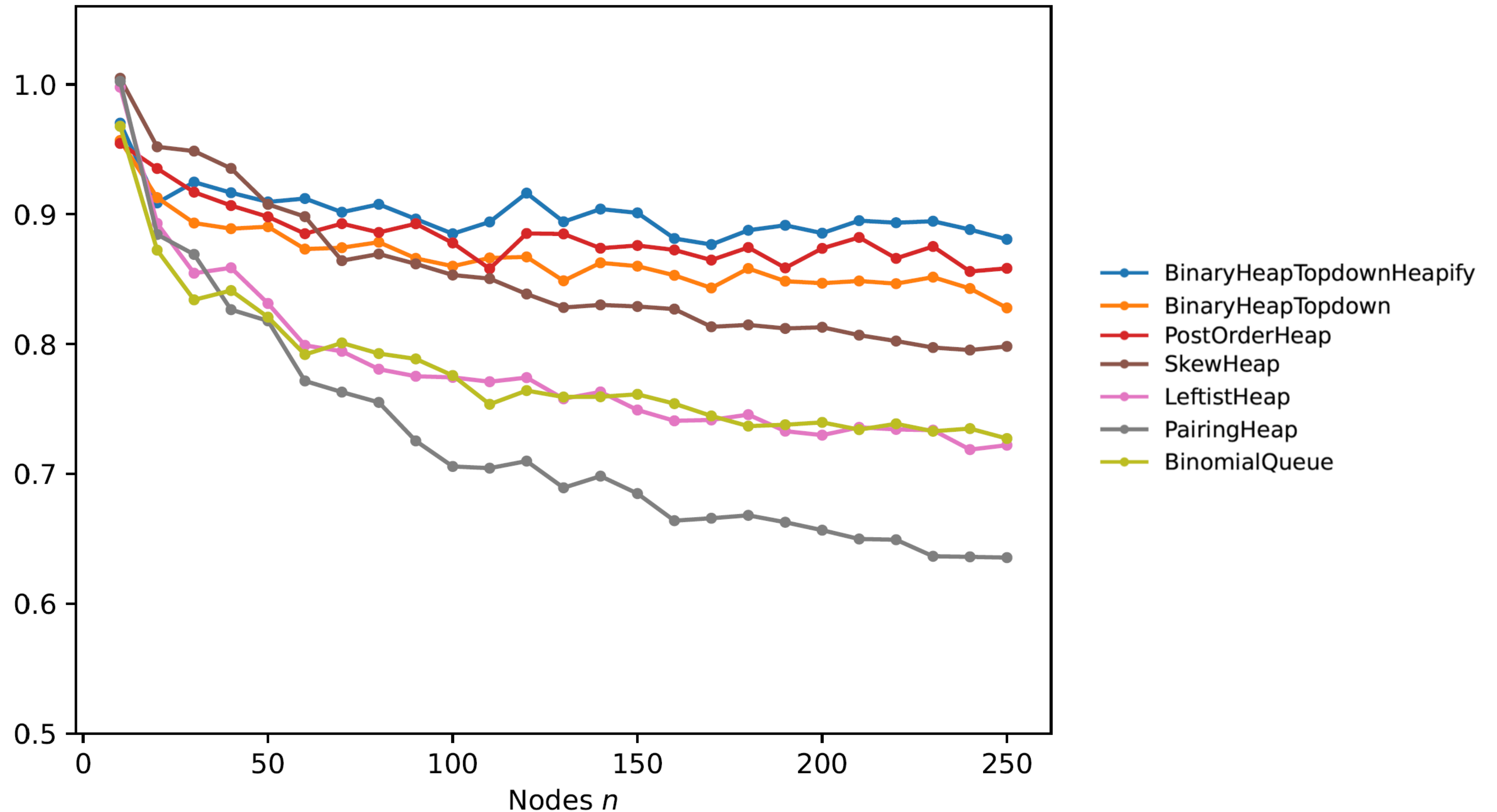


decreasing keys

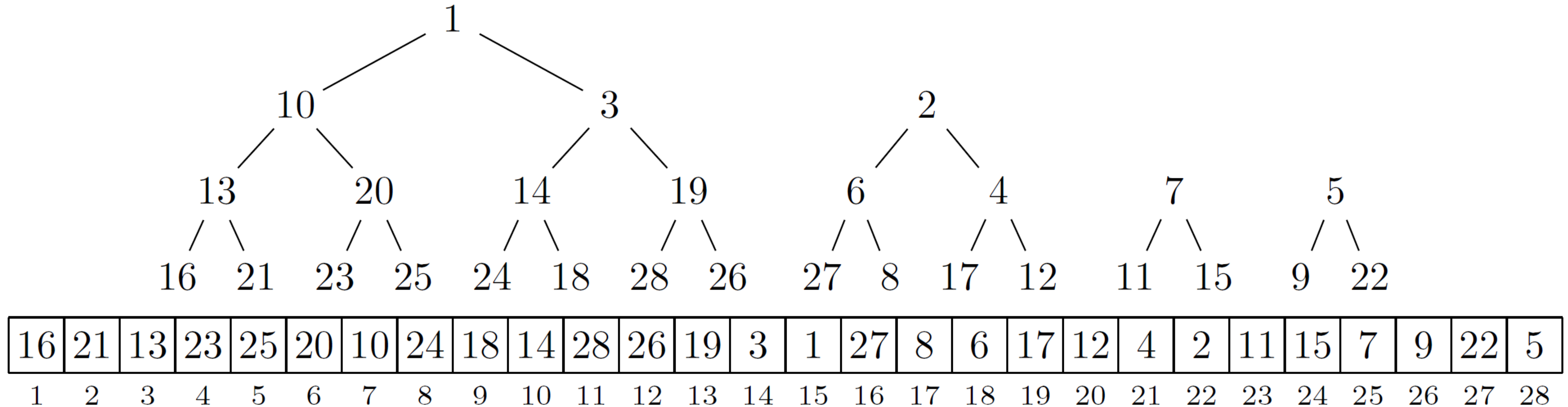


Reduction in comparisons

comparisons decreasing keys / comparisons ⟨key, value⟩ pairs



Postorder heap [Harvey and Zatloukal, FUN 2004]



- `Insert` amortized O(1), `ExtractMin` amortized O(log *n*)
- Implicit (space efficient)
- Best implicit comparison performance (and good time performance)

Conclusion

- Introduced notion of priority queues with decreasing keys
  ... as an approach to deal with outdated items in Dijkstra's algorithm
- Experiments identified priority queues supporting decreasing keys
  ... just had to prove it
- Builtin priority queues in Java and Python are binary heaps
  ... do not support decreasing keys
- Binary heaps with top-down insertions do support decreasing keys
  ... and also

skew heaps, leftist heaps, pairing heaps,
binomial queues, post-order heaps