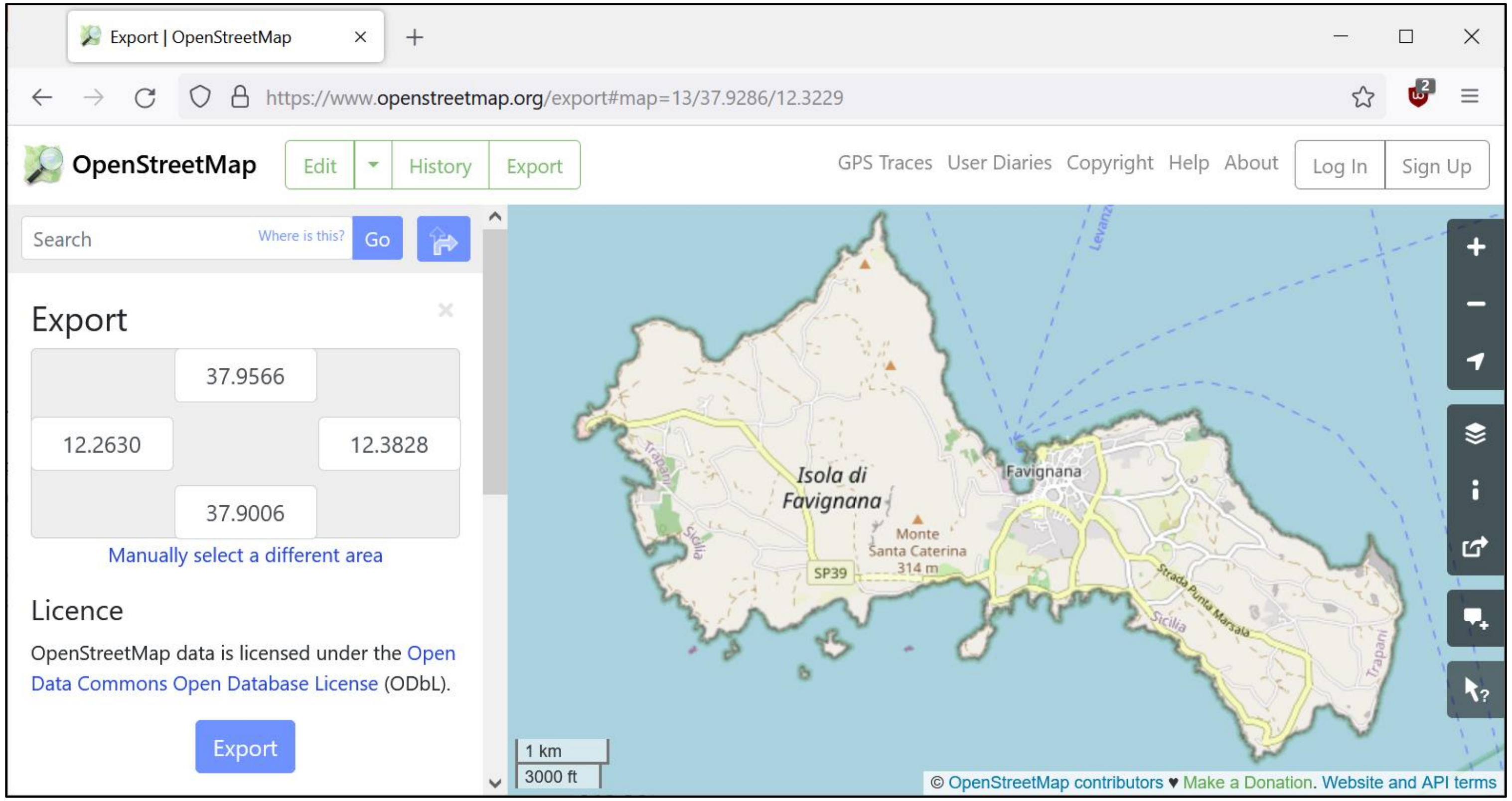
Priority Queues with Decreasing Keys

Gerth Stølting Brodal
Aarhus University

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

Background

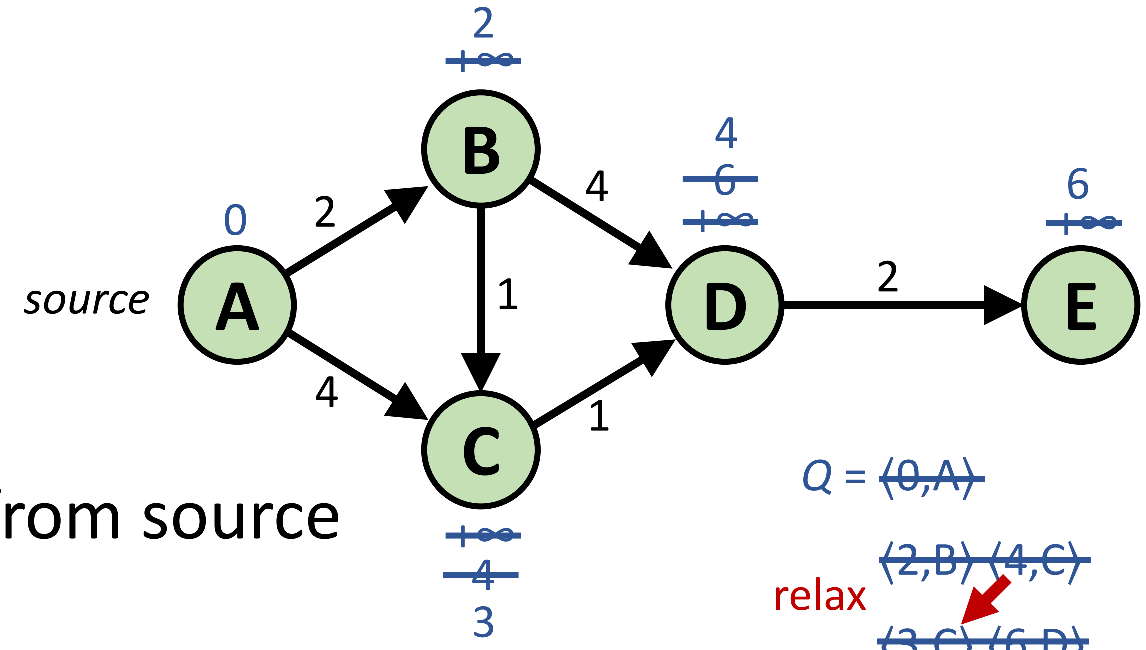
- Bachelorproject = shortest paths on Open Street Map graphs
- Students have trouble implementing Dijkstra's algorithm in Java™



```
 <way id="106231197" visible="true" version="8" changeset="90539127" timestamp="2020-09-07T15:29:03Z" user="vbertola" uid="4347030">
  <nd ref="1222984792"/>
  <nd ref="6789155595"/>
  <nd ref="1222984775"/>
  <nd ref="1222984838"/>
  ...
  <nd ref="4439326163"/>
  <nd ref="4439326143"/>
  <nd ref="1222984725"/>
  <nd ref="6789155593"/>
  <nd ref="1222984792"/>
  <tag k="addr:city" v="Favignana"/>
  <tag k="addr:housenumber" v="29"/>
  <tag k="addr:postcode" v="91023"/>
  <tag k="addr:street" v="Via Giovanni Amendola"/>
  <tag k="contact:facebook" v="www.facebook.com/exstabilimentofloriofavignana"/>
  <tag k="description" v="Museo regionale di storia - apertura stagionale marzo-novembre"/>
  <tag k="heritage:website" v="http://pti.regione.sicilia.it/portal/page/portal/PIR_PORTALE/PIR_LaStrutturaRegionale/PIR_AssBeniCulturali/PIR_BeniCulturaliAmbientali"/>
  <tag k="name" v="Ex stabilimento Florio delle tonnare di Favignana e Formica"/>
  <tag k="operator" v="Regione siciliana"/>
  <tag k="operator:type" v="public"/>
  <tag k="tourism" v="museum"/>
  <tag k="website" v="http://www.visitsicily.info/ex-stabilimento-florio-tonnara-favignana/"/>
 </way>
```

Dijkstra’s algorithm

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



Q = ~~⟨0,A⟩~~

~~⟨2,B⟩ ⟨4,C⟩~~

relax

~~⟨3,C⟩ ⟨6,D⟩~~

~~⟨4,D⟩~~

~~⟨6,E⟩~~

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

relax

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

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

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

The Challenge - Java's builtin binary heap

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

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

- comparator function

Java SE 18 & JDK 18

SEARCH: Search

Implementation note: this implementation provides O(log(n)) time for the enqueuing and dequeuing methods (`offer`, `poll`, `remove()` and `add`); linear time for the `remove(Object)` and `contains(Object)` methods; and constant time for the retrieval methods (`peek`, `element`, and `size`).

This class is a member of the Java Collections Framework.

Since:

1.5

Java SE 18 & JDK 18

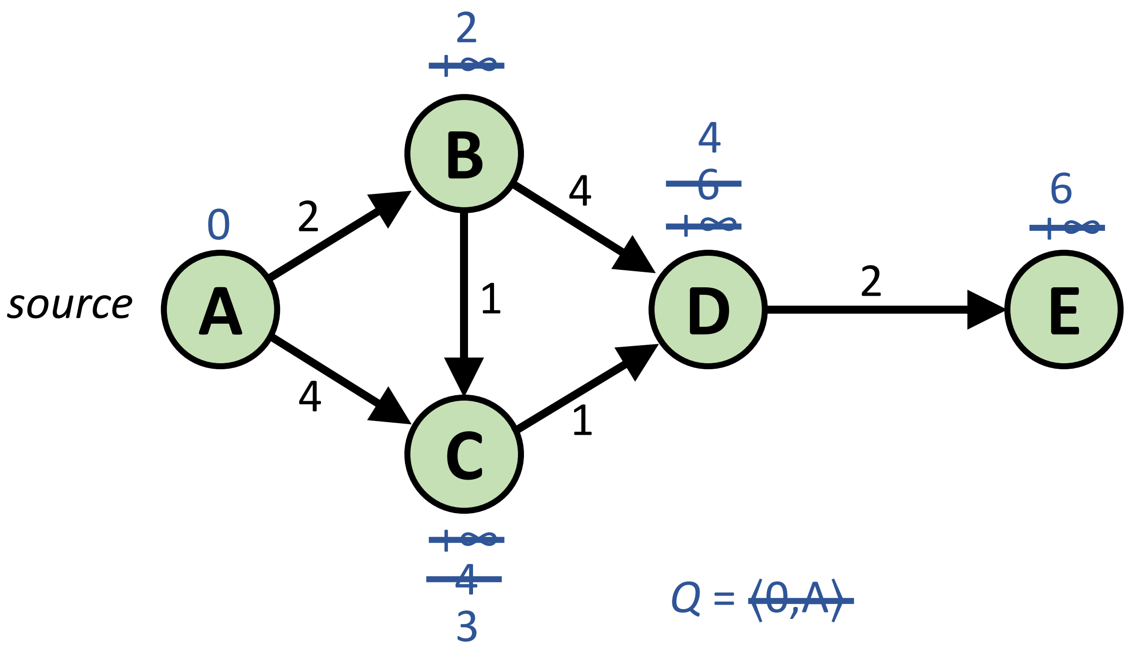
SEARCH: Search

`PriorityQueue(int initialCapacity)`	Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to their natural ordering.
`PriorityQueue(int initialCapacity, Comparator<? super E> comparator)`	Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to the specified comparator.

Repeated insertions

- Relax inserts new copies of item
- Skip outdated items

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



Q = ~~⟨0,A⟩~~

~~⟨2,B⟩ ⟨4,C⟩~~

~~⟨3,C⟩ ⟨4,C⟩ ⟨6,D⟩~~

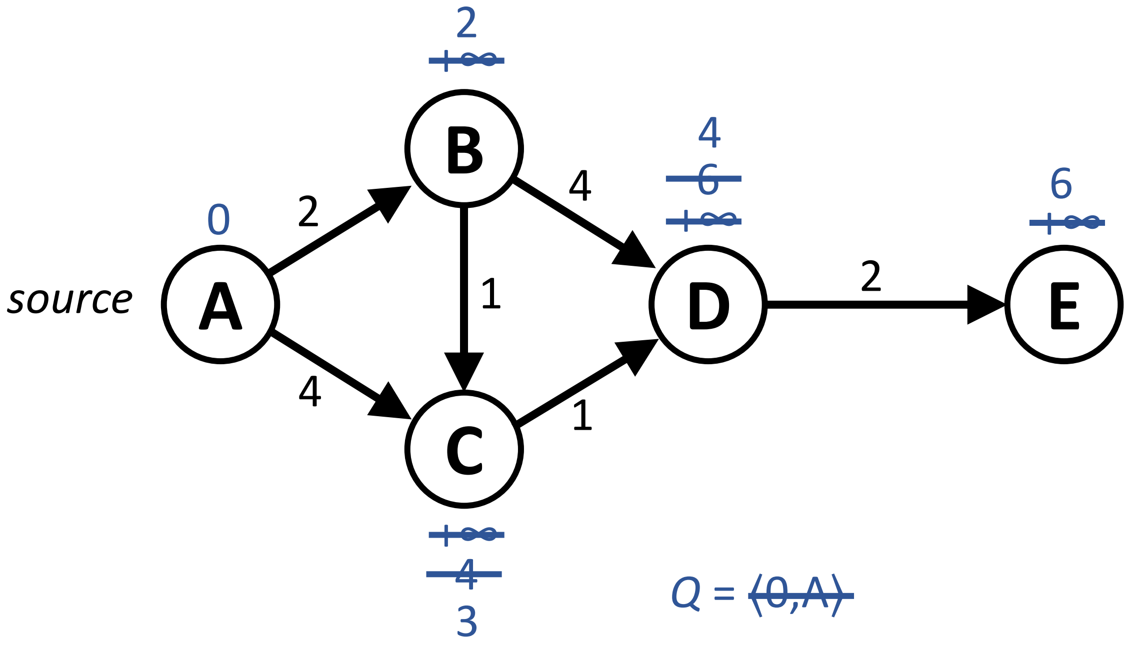
~~⟨4,C⟩ ⟨4,D⟩ ⟨6,D⟩~~

~~⟨4,D⟩ ⟨6,D⟩~~

~~⟨6,D⟩ ⟨6,E⟩~~

~~⟨6,E⟩~~

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⟩~~

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

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

$\quad dist[s] = 0$

$\quad visited = \emptyset$

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

$\quad \textbf{while } Q \neq \emptyset \textbf{ do}$

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

use bitvector ⟶ $\qquad \textbf{if } u \notin visited \textbf{ then}$

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

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

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

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

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

$\quad \textbf{return } dist$

A shaky idea…

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

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

$\quad dist[s] = 0$

$\quad visited = \emptyset$

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

$\quad \textbf{while}\ Q \neq \emptyset\ \textbf{do}$

d never used → $\quad \langle \cancel{d}, u\rangle = \texttt{ExtractMin}(Q)$

$\quad\quad \textbf{if}\ u \notin visited\ \textbf{then}$

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

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

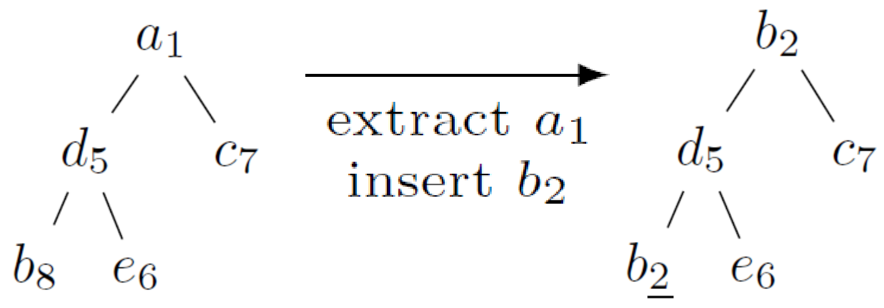
$\quad\quad\quad\quad \textbf{if}\ dist[u] + \delta(u, v) < dist[v]\ \textbf{then}$

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

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

$\quad \textbf{return}\ dist$

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



Heap invariants break

Experimental study

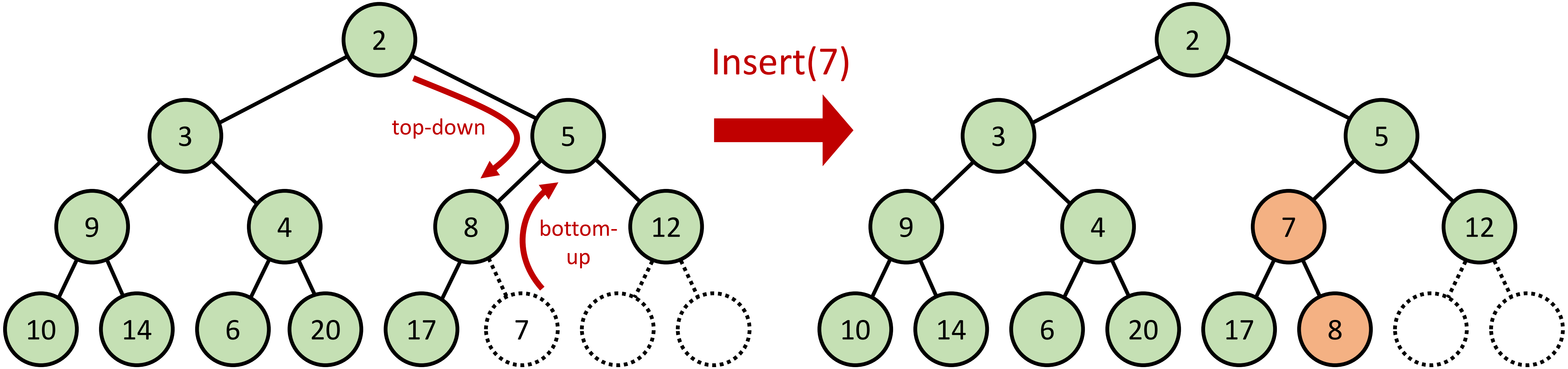
```
visited = set()
Q = Queue()
Q.insert(Item(0, source))
while not Q.empty():
    u = Q.extract_min().value
    if u not in visited:
        visited.add(u)
        for v in G.out[u]:
            dist_v = dist[u] + G.weights[(u, v)]
            if dist_v < dist[v]:
                dist[v] = dist_v
 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

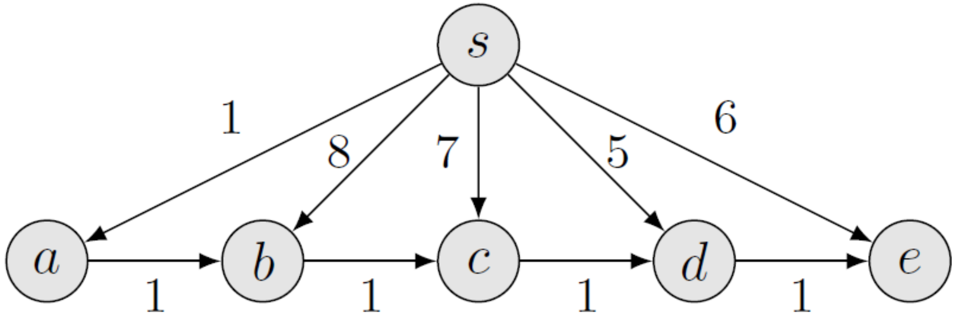
Pointer based

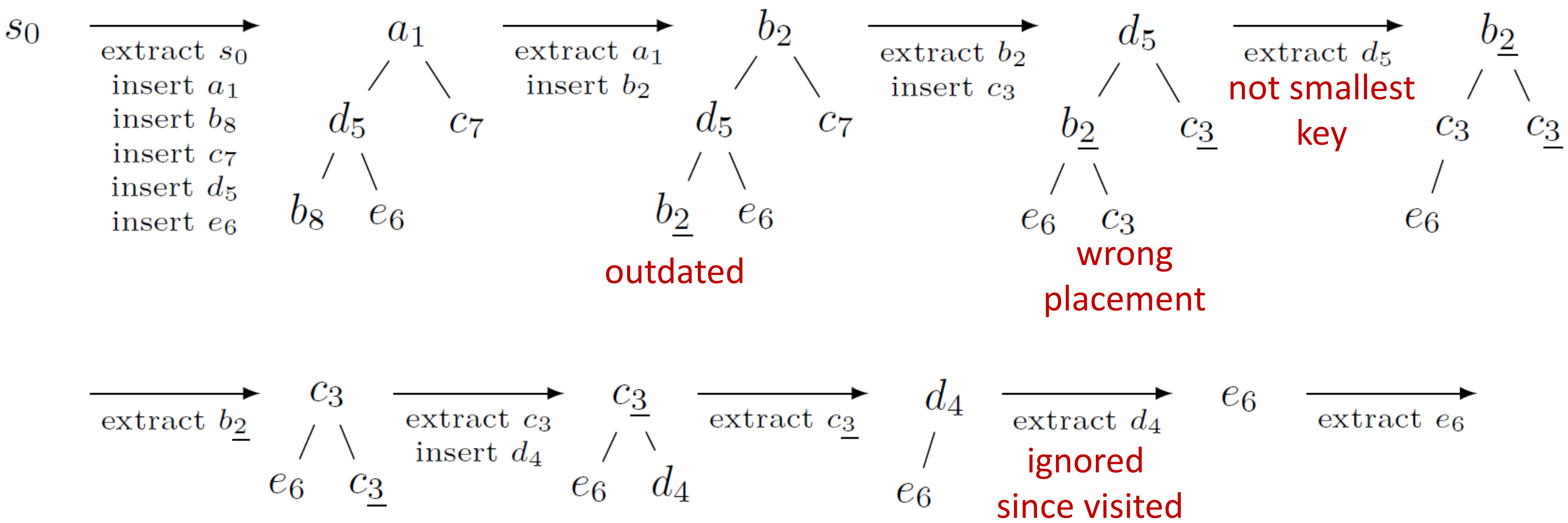
Implicit (space efficient)

Binary heap insertions – bottom-up vs top-down



Binary heaps using *dist* by 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

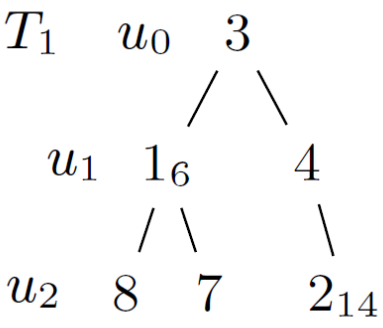
- Decreased heap order

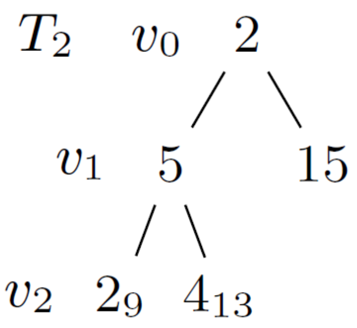
  u ancestor of *v* ⇒
  current key *u* ≤ original key *v*

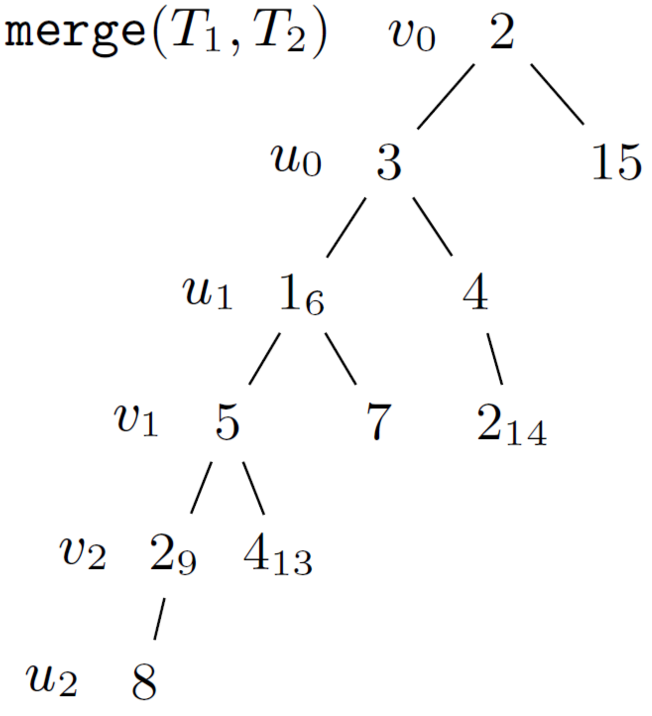
- Root valid item to extract

- Top-down merging two paths preserves decreased heap order

⇒ skew heaps and leftist heaps support decreasing keys



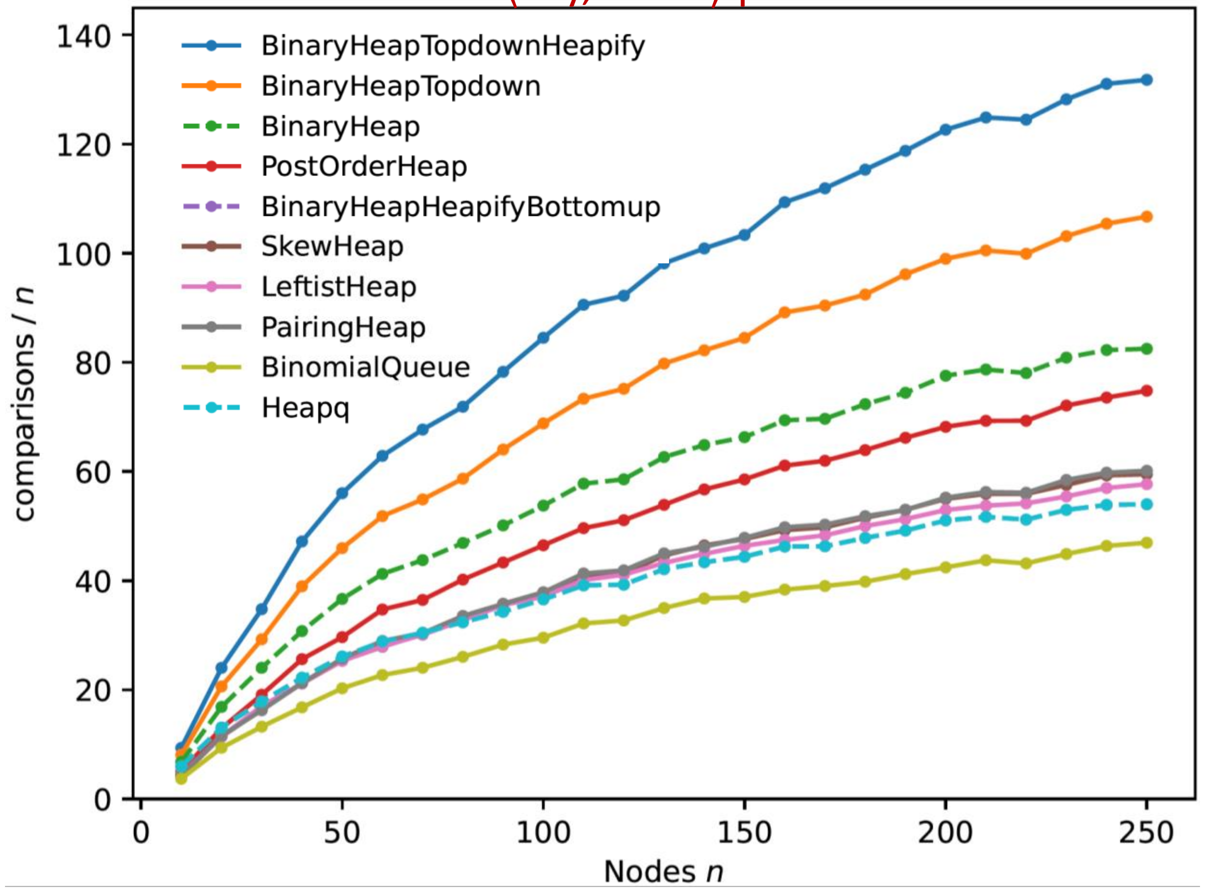




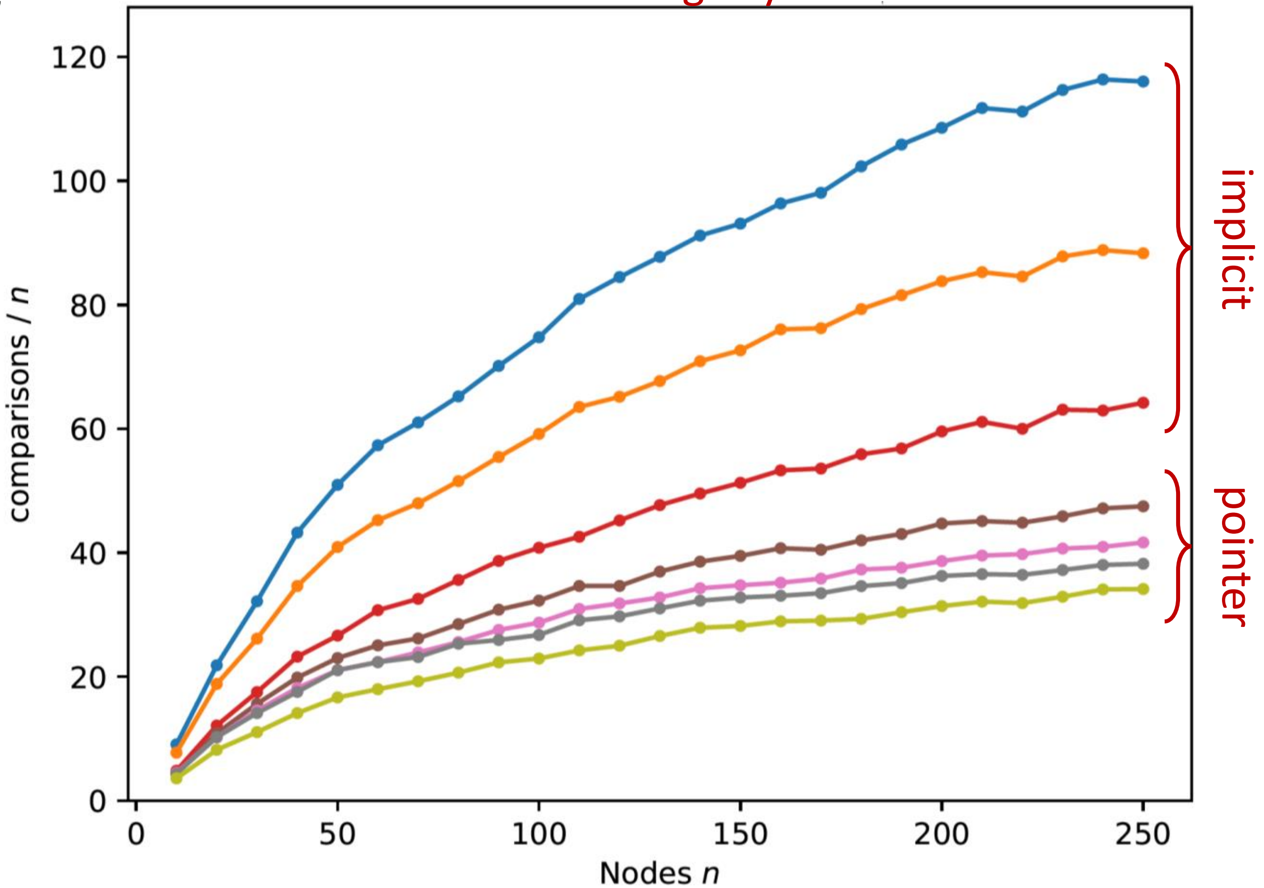
Experimental evaluation of various heaps

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

⟨key, value⟩ pairs

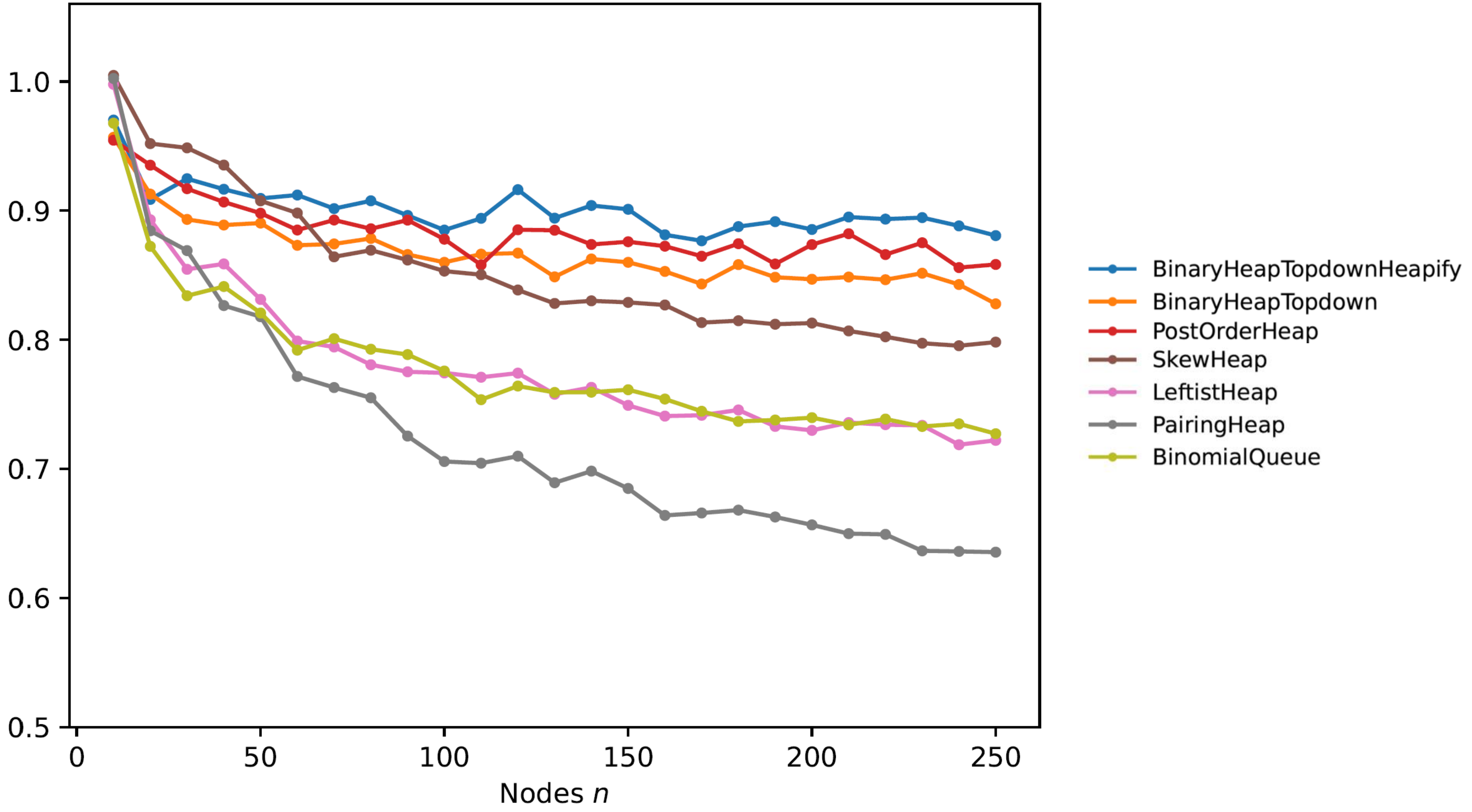


decreasing keys

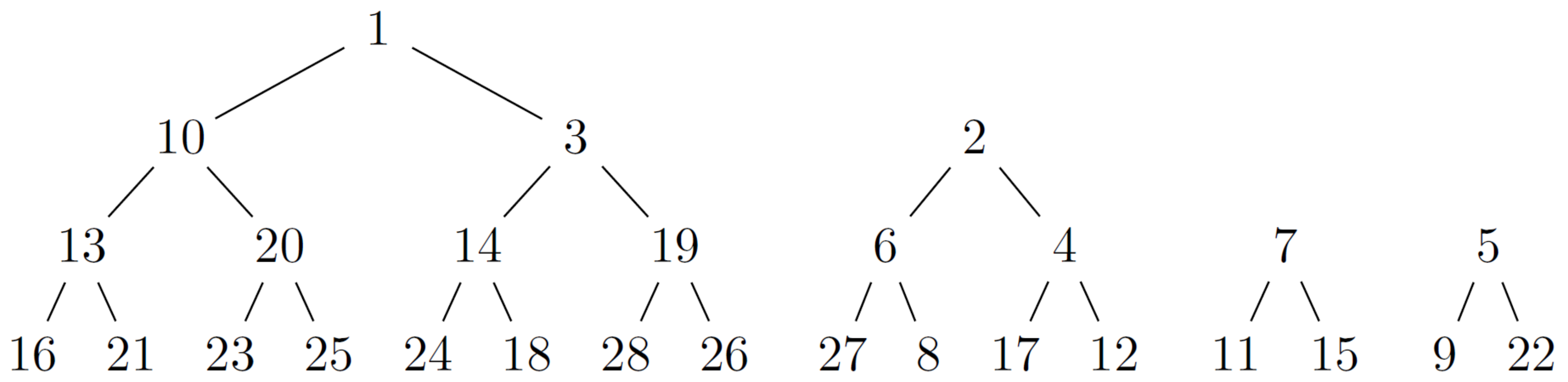


Reduction in comparisons

comparisons decreasing keys / comparisons ⟨key, value⟩ pairs



Postorder heap [Harvey and Zatloukal, FUN 2004]



16	21	13	23	25	20	10	24	18	14	28	26	19	3	1	27	8	6	17	12	4	2	11	15	7	9	22	5
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

- `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