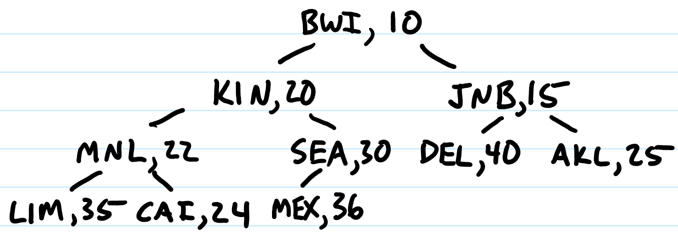
Heap

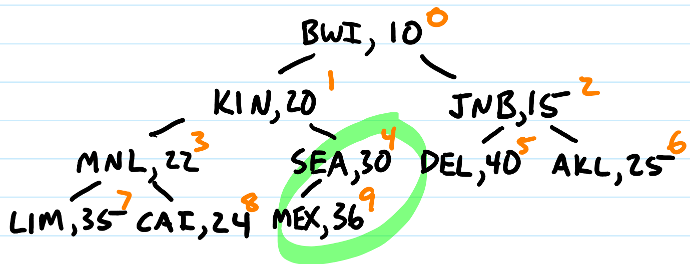


Heap: a binary tree such that

order the valve in a node is $\le$ the valve in any children (min-heap)

shape all levels but possibly the last are full and nodes are as far left as possible

Heap

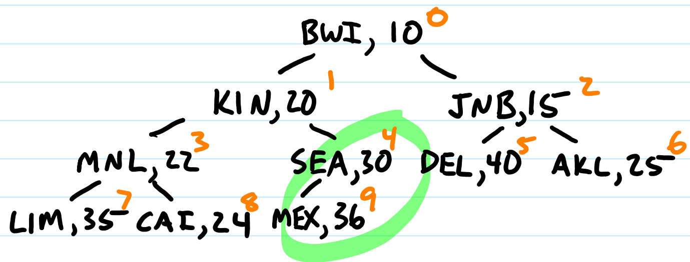


Heap: a binary tree such that

order the value in a node is ≤ the value in any children (min-heap)

shape all levels but possibly the last are full and nodes are as far left as possible

Heap



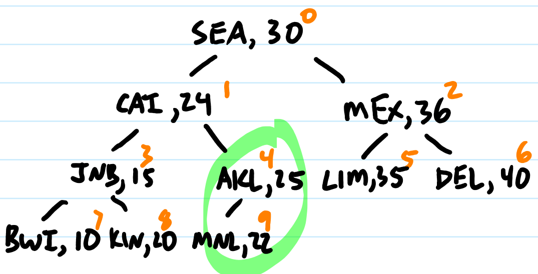
LEFT(i) = 2i + 1
RIGHT(i) = 2i + 2

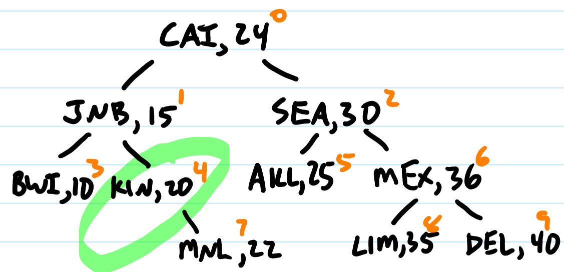
0	1	2	3	4	5	6	7	8	9
BWI 10	KIN 20	JNB 15	MNL 22	SEA 30	DEL 40	AKL 25	LIM 35	CAI 24	MEX 36

Heap: a binary tree such that

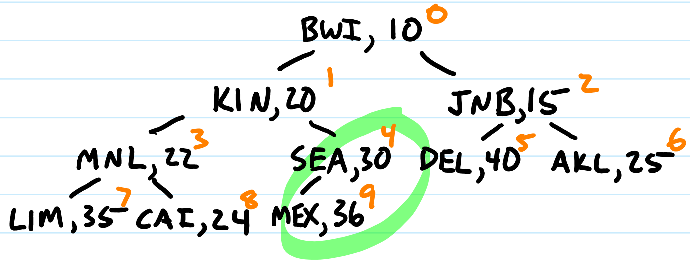
order the value in a node is ≤ the value in any children

shape all levels but possibly the last are full and nodes are as far left as possible





Heap



$LEFT(i) = 2i+1$

$RIGHT(i) = 2i+2$

$PARENT(i) = \lfloor \frac{i-1}{2} \rfloor$

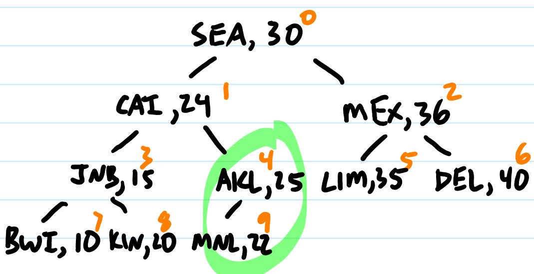
AKL 6
BWI 0
DEL 5
SEA 4
LIM 7
MNL 3
CAI 8
MEX 9
KIN 1
JNB 2

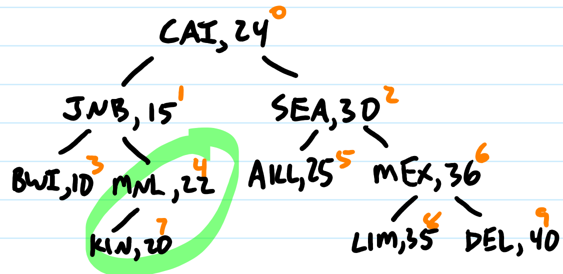
0	1	2	3	4	5	6	7	8	9
BWI 10	KIN 20	JNB 15	MNL 22	SEA 30	DEL 40	AKL 25	LIM 35	CAI 24	MEX 36

Heap: a binary tree such that

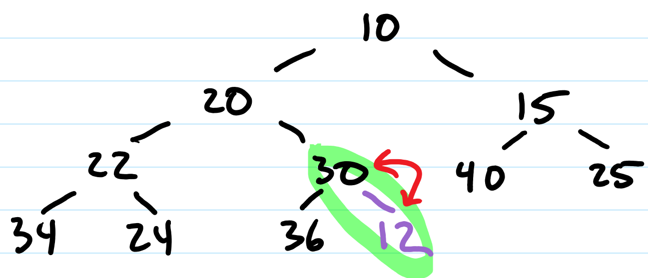
order: the value in a node is ≤ the value in any children

shape: all levels but possibly the last are full and nodes are as far left as possible





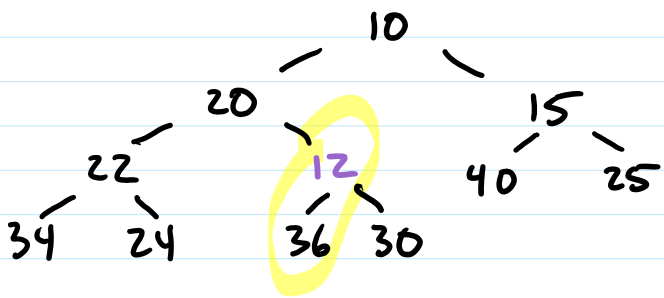
Heap



enqueue(key, pri):

i ← n
add (key, pri) at location i
while priority at i < priority at PARENT(i)
  swap i, PARENT(i)

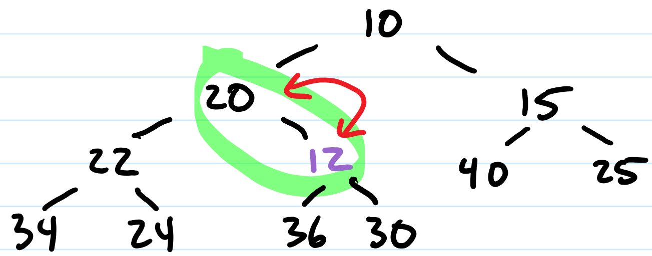
Heap



enqueue(key, pri):

i ← n
add (key, pri) at location i
while priority at i < priority at PARENT(i)
    swap i, PARENT(i)
    i ← PARENT(i)

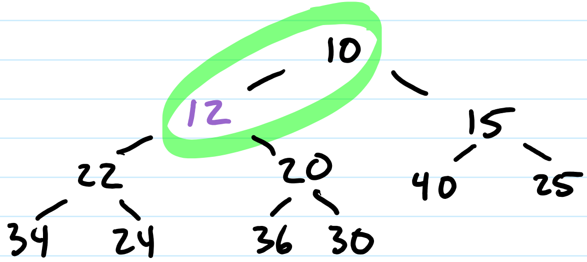
Heap



enqueue(key, pri):

```
i ← n
add (key, pri) at location i
while priority at i < priority at PARENT(i)
    swap i, PARENT(i)
    i ← PARENT(i)
```

Heap



enqueue(key, pri):

worst case $\Theta(\log n)$

$i \leftarrow n$

add (key, pri) at location i

while i > 0 and priority at i < priority at PARENT(i)

  swap i, PARENT(i)

  $i \leftarrow$ PARENT(i)

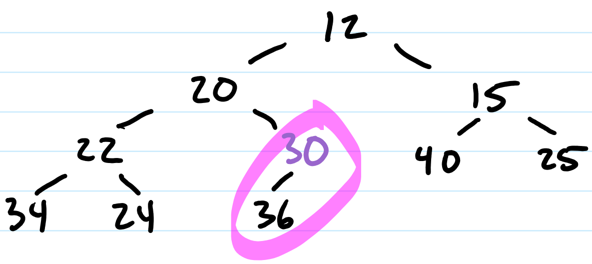
$n \leftarrow n+1$

O(1) per iteration

≤ 1 iteration per level

$\Theta(\log n)$ levels

Heap



dequeue( ) : remember item at root

worst case $\Theta(\log n)$

move last node to root

$n \leftarrow n-1$

$i \leftarrow 0$

while LEFT(i) < n and priority at i > priority at one of i's children

worst case $\Theta(\log n$ iterations)

swap i with smallest child

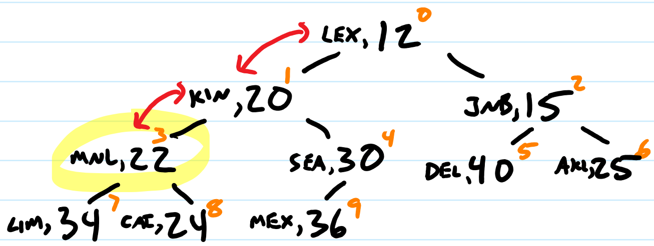
$i \leftarrow$ former index of smallest child

O(1) per iteration

change_priority : find index i of item to change

use loop from dequeue if priority value ↑, loop from enqueue if ↓

Heap



KIN	1
JNB	2
AKL	6
MEX	9
CAI	8
MNL	3
DEL	5
LIM	7
SEA	4
LEX	0

change-priority (MNL, 5)

dequeue(): remember item at root
worst case $\Theta(\log n)$ move last node to root
$n \leftarrow n-1$
$i \leftarrow 0$
worst case ($\Theta(\log n$ iterations)) while LEFT(i) < n and priority at i > priority at one of i's children
  swap i with smallest child — $O(1)$ per iteration
  $i \leftarrow$ former index of smallest child

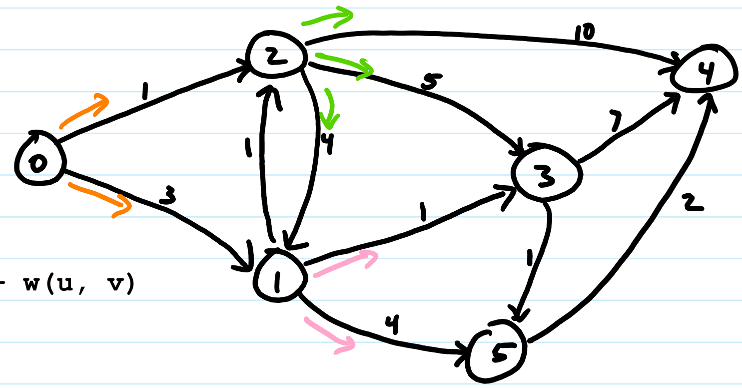
change-priority: find index i of item to change
worst case $\Theta(\log n)$ use loop from dequeue if priority value ↑, loop from enqueue if ↓

Dijkstra's Algorithm

```
for each v
   color[v], pred[v], d[v] ← IN_QUEUE, NIL, ∞
d[s] ← 0

Q ← new PriorityQueue(d)

while Q not empty
   u ← dequeue(Q)
   for each outneighbor v of u
      if color[v] = IN_QUEUE and d[v] > d[u] + w(u, v)
         change_priority(Q, v, d[u] + w(u, v))
         d[v] ← d[u] + w(u, v)
         pred[v] ← u
   color[u] ← DONE
```



vertex	~~0~~	~~1~~	~~2~~	3	4	5
priority(d)	0	~~∞~~ 3	~~∞~~ 1	~~∞~~ ~~6~~ 4	~~∞~~ 11	~~∞~~ 7
predecessor		0	0	2	2	1