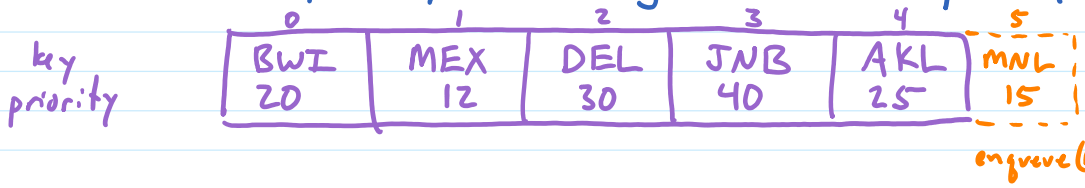
Priority Queue

a collection of items identified by a key and having an associated priority



array

amortized O(1)

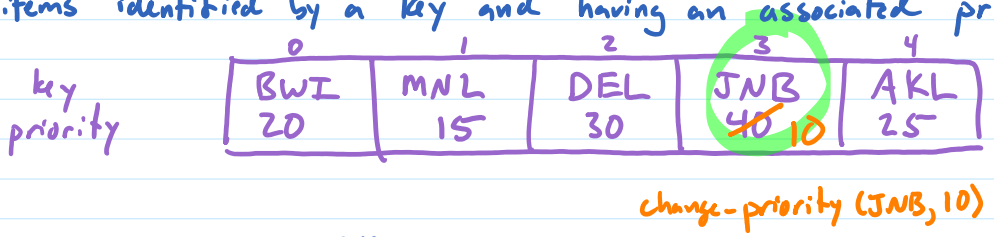
enqueue (key, pri)
add a new item with given key and priority

dequeue ( )
remove and return the item with lowest priority value

change_priority (key, pri)
change the priority of the item with the given key

Priority Queue

a collection of items identified by a key and having an associated priority



key	index
BWI	0
DEL	2
JNB	3
AKL	4
MNL	1

array

amortized

enqueue (key, pri) — O(1)

add a new item with given key and priority

dequeue () — Θ(n)

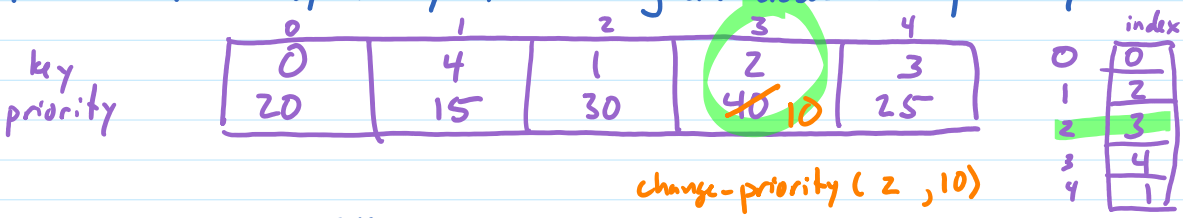
remove and return the item with lowest priority value

change-priority (key, pri) — O(1)

change the priority of the item with the given key

Priority Queue

a collection of items identified by a key and having an associated priority



array

amortized

enqueue (key, pri) — O(1)

add a new item with given key and priority

dequeue ( ) — Θ(n)

remove and return the item with lowest priority value

change-priority (key, pri) — O(1)

change the priority of the item with the given key

Priority Queue

a collection of items identified by a key and having an associated priority

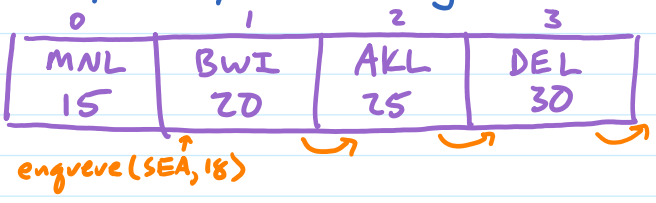
	0	1	2	3
key	BWI	MNL	DEL	AKL
priority	20	15	30	25

key	index
BWI	0
DEL	2
~~JNB~~	~~3~~
AKL	~~4~~ 3
MNL	1

	array	sorted array
	amortized	
enqueue (key, pri)	O(1)	
add a new item with given key and priority		
dequeue ( )	Θ(n)	
remove and return the item with lowest priority value		
change_priority (key, pri)	O(1)	
change the priority of the item with the given key		

Priority Queue

a collection of items identified by a key and having an associated priority

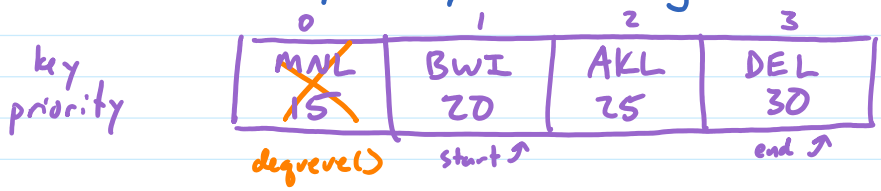


key	index	
BWI	~~1~~	2
DEL	~~3~~	4
AKL	~~2~~	3
MNL	0	
SEA	1	

	array	sorted array
	amortized	worst case
enqueue (key, pri)	$O(1)$	$\Theta(n)$
add a new item with given key and priority		
dequeue ( )	$\Theta(n)$	
remove and return the item with lowest priority value		
change_priority (key, pri)	$O(1)$	
change the priority of the item with the given key		

Priority Queue

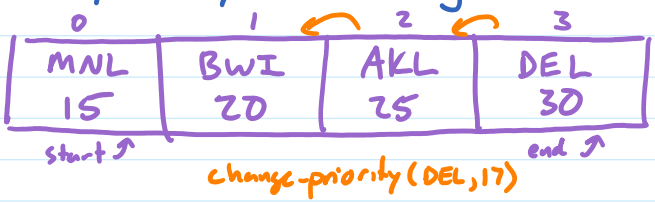
a collection of items identified by a key and having an associated priority



	array	sorted array
	amortized	worst case
enqueue (key, pri) — add a new item with given key and priority	O(1)	Θ(n)
dequeue ( ) — remove and return the item with lowest priority value	Θ(n)	O(1)
change_priority (key, pri) — change the priority of the item with the given key	O(1)	

Priority Queue

a collection of items identified by a key and having an associated priority

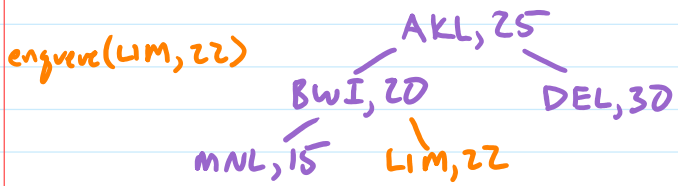


key	index
BWI	~~1~~ 2
DEL	~~3~~ 1
AKL	~~2~~ 3
MNL	0

	array	sorted array
	amortized	worst case
enqueue (key, pri) add a new item with given key and priority	O(1)	θ(n)
dequeue ( ) remove and return the item with lowest priority value	θ(n)	O(1)
change_priority (key, pri) change the priority of the item with the given key	O(1)	worst case θ(n)

Priority Queue

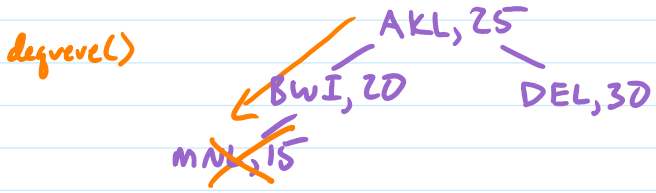
a collection of items identified by a key and having an associated priority



	array	sorted array	balanced binary search tree
	amortized	worst case	worst case
enqueue (key, pri) add a new item with given key and priority	$O(1)$	$\Theta(n)$	$\Theta(\log n)$
dequeue ( ) remove and return the item with lowest priority value	$\Theta(n)$	$O(1)$	
change_priority (key, pri) change the priority of the item with the given key	$O(1)$	worst case $\Theta(n)$	

Priority Queue

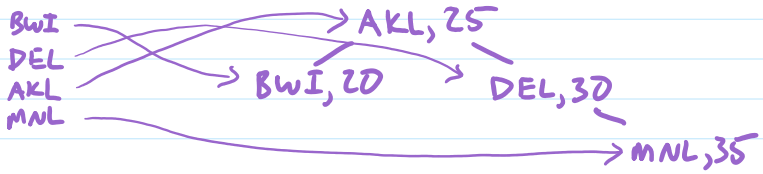
a collection of items identified by a key and having an associated priority



	array	sorted array	balanced binary search tree
enqueue (key, pri)	amortized $O(1)$	worst case $\Theta(n)$	worst case $\Theta(\log n)$
add a new item with given key and priority			
dequeue ( )	$\Theta(n)$	$O(1)$	worst case $\Theta(\log n)$
remove and return the item with lowest priority value			
change_priority (key, pri)	$O(1)$	worst case $\Theta(n)$	
change the priority of the item with the given key			

Priority Queue

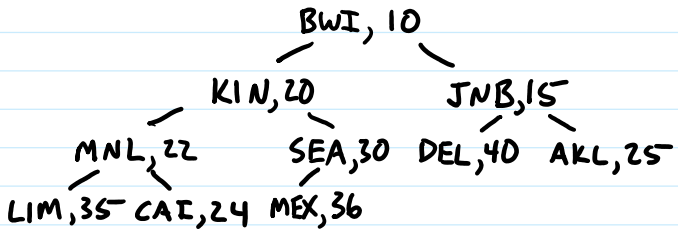
a collection of items identified by a key and having an associated priority



change-priority (MNL, 35)

	array	sorted array	balanced binary search tree	heap
enqueue (key, pri) add a new item with given key and priority	amortized $O(1)$	worst case $\Theta(n)$	worst case $\Theta(\log n)$ →	
dequeue ( ) remove and return the item with lowest priority value	$\Theta(n)$	$O(1)$	worst case $\Theta(\log n)$ →	
change-priority (key, pri) change the priority of the item with the given key	$O(1)$	worst case $\Theta(n)$	worst case $\Theta(\log n)$ →	
TOTAL n enqueues/dequeues + m change-priority operations	$O(n^2 + m)$ $m \in \Theta(n)$ $O(n^2)$ $m \in \Theta(n^2)$ $O(n^2)$	$O(n^2 + nm)$ $O(n^2)$ $O(n^3)$	$O((n+m)\log n)$ $O(n \log n)$ $O(n^2 \log n)$	

Heap

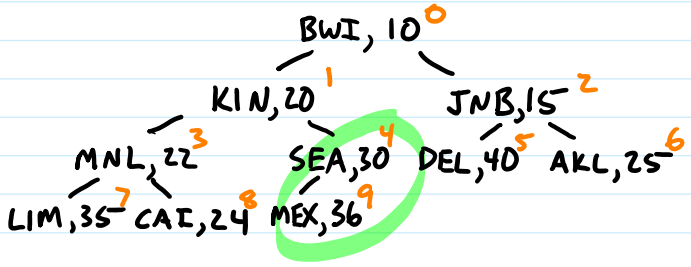


Heap: a binary tree such that

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

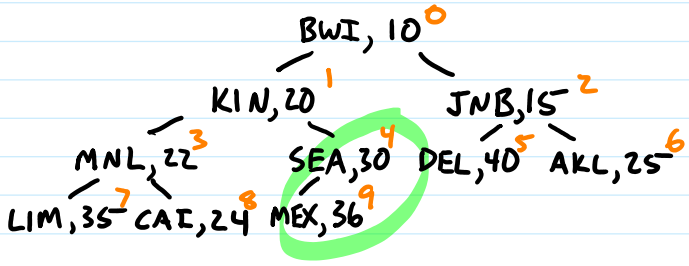


Heap: a binary tree such that

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



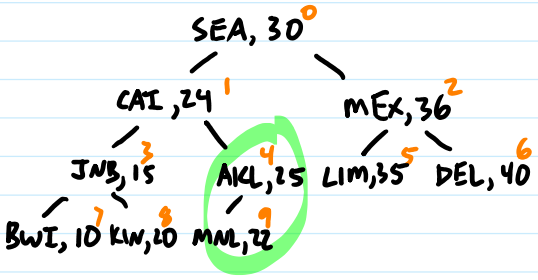
LEFT(i) = 2i + 1
RIGHT(i) = 2i + 2

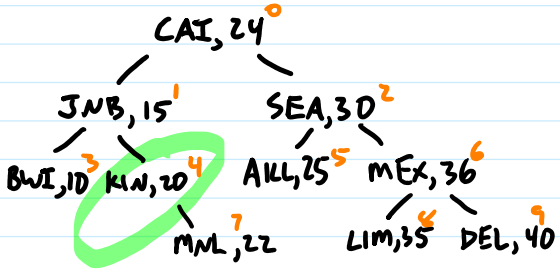
0	1	2	3	4	5	6	7	8	9
BWI 10	KIN 20	JNB 15	MNL 22	SEA 30	DEL 40	AKL 25	LIM 35	CAI 24	MEX 36

Heap: a binary tree such that

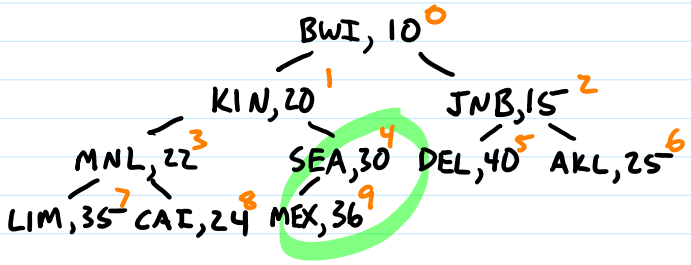
order the value in a node is ≤ the value in any children

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

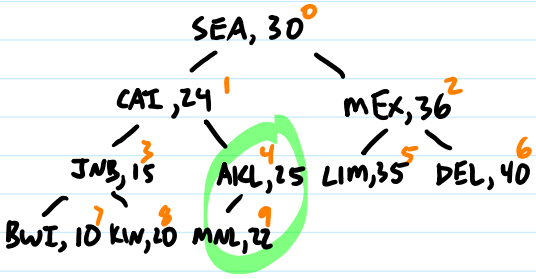
AKL 6
BWI 0
DEL 5
SEA 4
LIM 7
MNL 3
CAI 8
MEX 9
KIN 1
JNB 2

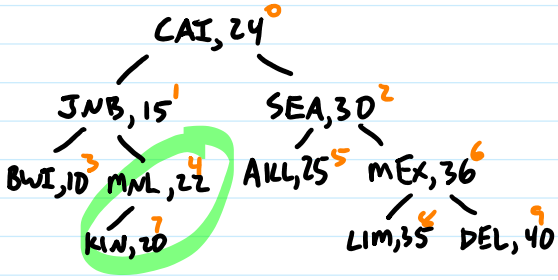
0	1	2	3	4	5	6	7	8	9
BWI 10	KIN 20	JNB 15	MNL 22	SEA 30	DEL 40	AKL 25	LIM 35	CAI 24	MEX 36

Heap: a binary tree such that

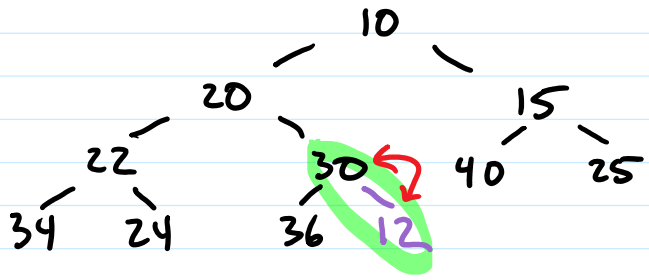
order the value in a node is ≤ the value in any children

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





Heap



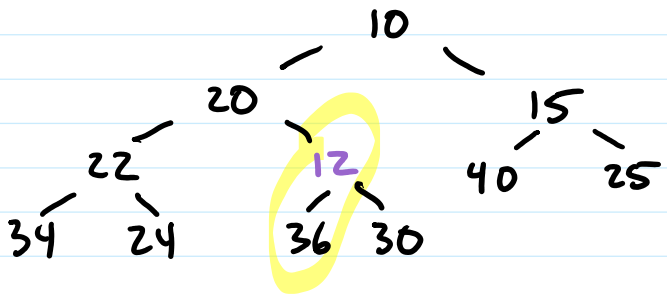
enqueue(key, pri): $i \leftarrow n$

add (key, pri) at location i

while priority at i < priority at PARENT(i)

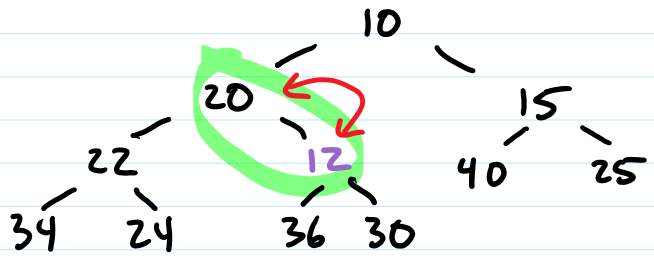
swap i, PARENT(i)

Heap



enqueue(key, pri): $i \leftarrow n$
add (key, pri) at location i
while priority at i < priority at PARENT(i)
swap i, PARENT(i)
$i \leftarrow$ PARENT(i)

Heap



enqueue(key, pri): $i \leftarrow n$

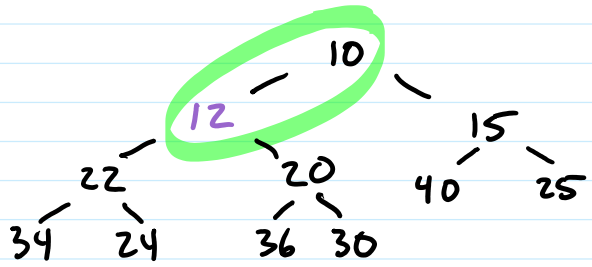
add (key, pri) at location i

while priority at i < priority at PARENT(i)

swap i, PARENT(i)

$i \leftarrow$ PARENT(i)

Heap



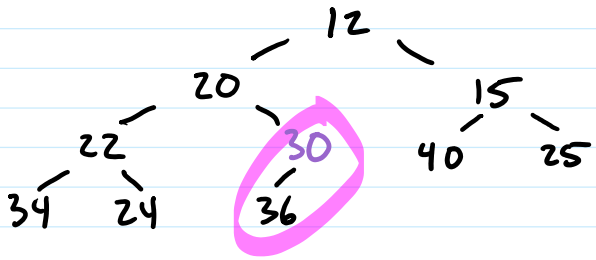
enqueue(key, pri):
worst case $\Theta(\log n)$

$i \leftarrow n$
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)
$n \leftarrow n + 1$

≤ 1 iteration per level
$\Theta(\log n)$ levels

O(1) per iteration

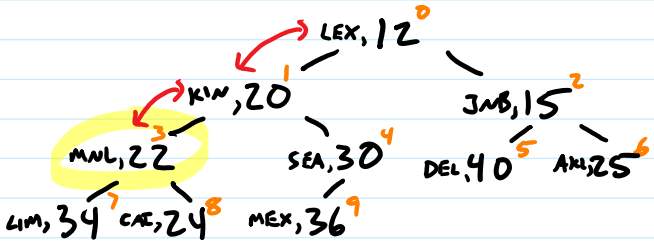
Heap



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
i ← 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
i ← former index of smallest child
O(1) per iteration

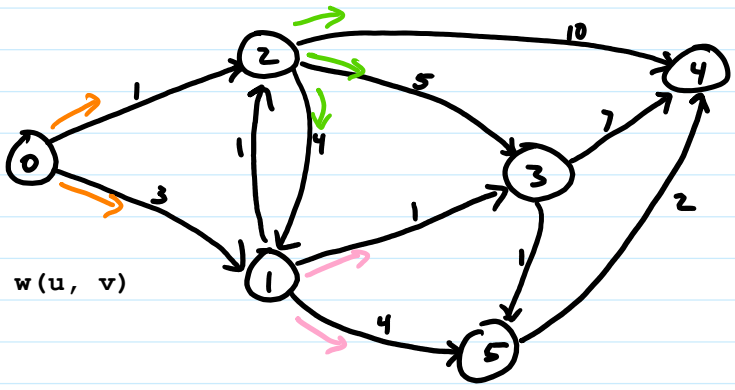
change_priority: find index i of item to change
worst case $\Theta(\log n)$ use loop from dequeue if priority value ↑, loop from enqueue if ↓

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