Priority Queues

"Deadlines aren't bad. They help you organize your time. They help you set priorities. They make you get going when you might not feel like it. "

- *Harvey B. Mackay*

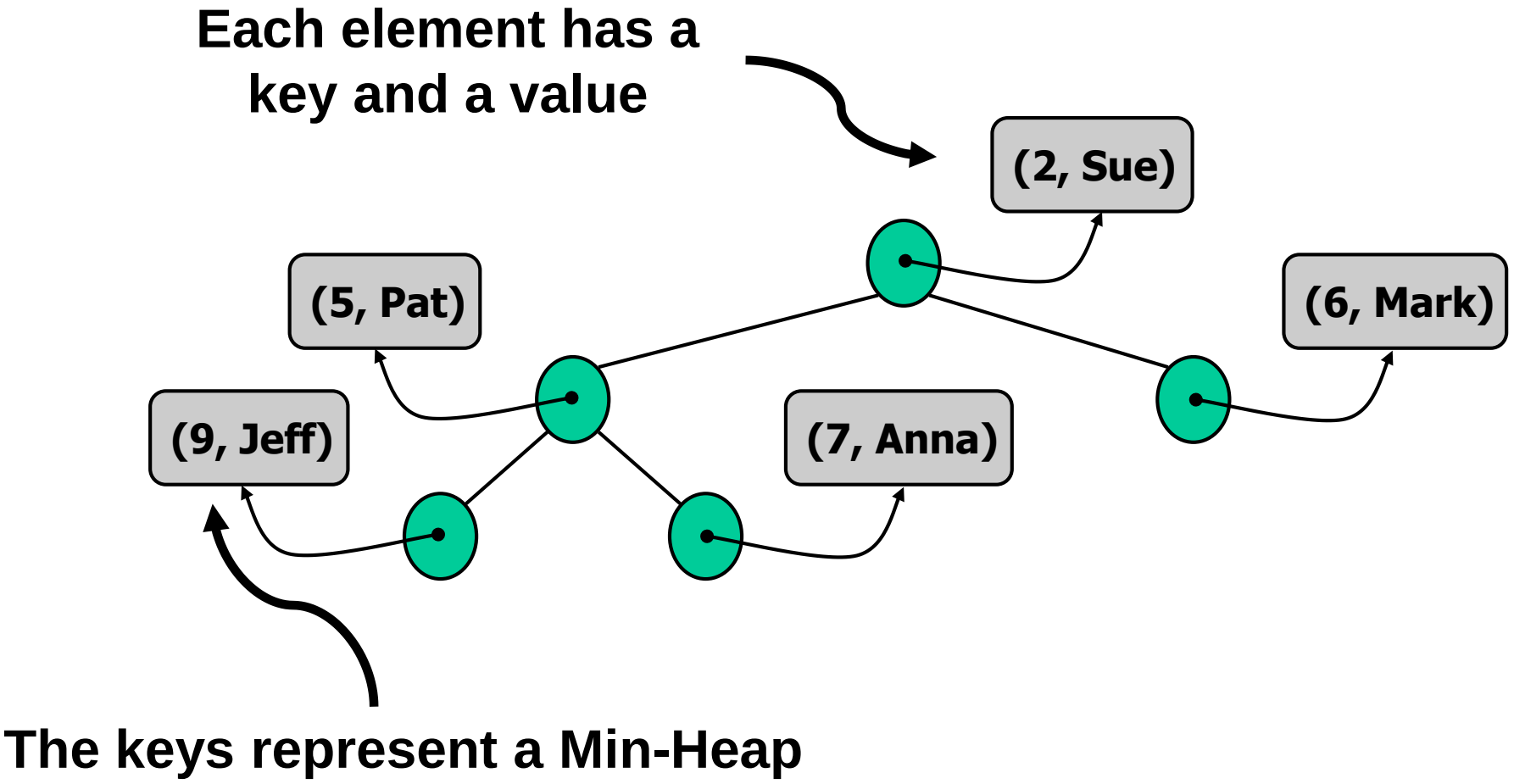
Priority Queue

A priority queue is a queue where elements are accessed according to their priorities:

- *Max-Priority Queue*
  - an element with high priority is served before an element with low priority
- *Min-Priority Queue*
  - an element with low priority is served before an element with high priority

Heaps are usually used to implement priority queues.

Example: Min-Priority Queue



Priority Queues

A *Max-Priority Queue* supports the following operations:

- `Maximum(A)`: returns the element with the largest key
- `Extract-Max(A)`: removes / returns the element with the largest key
- `Increase-Key(A,i,k)`: increases the key of element in position `i` to `k`
- `Insert(A,x,k)`: inserts the element `x` with key `k`

Similarly, a *Min-Priority Queue* supports `Minimum`, `Extract-Min`, `Decrease-Key`, and `Insert` operations.

Max-Heap Implementation

`Maximum(A):`

HEAP-MAXIMUM(A)

return $A[1]$

Time: $\Theta(1)$.

Max-Heap Implementation

`Extract-Max(A):`

HEAP-EXTRACT-MAX(A, n)
if $n < 1$
  then error "heap underflow"
$max \leftarrow A[1]$
$A[1] \leftarrow A[n]$
MAX-HEAPIFY$(A, 1, n-1)$ ▷ remakes heap
return max

Time: $O(\lg n)$.

Max-Heap Implementation

`Increase-Key(A,i,k):`

HEAP-INCREASE-KEY(A, i, key)

if $key < A[i]$

  then error "new key is smaller than current key"

$A[i] \leftarrow key$

while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$

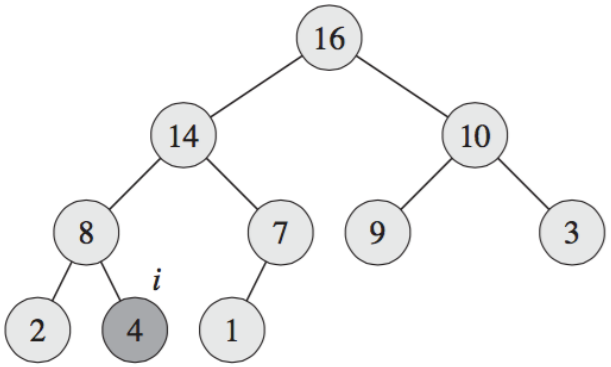
  do exchange $A[i] \leftrightarrow A[\text{PARENT}(i)]$

    $i \leftarrow \text{PARENT}(i)$

Time: $O(\lg n)$.

Example: Increase-Key

`Increase-Key(A,9,15):`



(a)

HEAP-INCREASE-KEY(A, i, key)

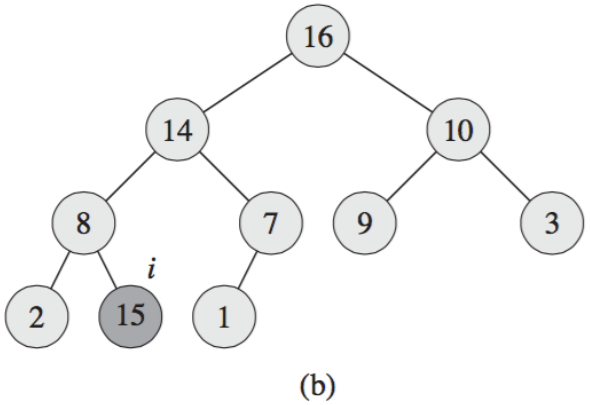
if $key < A[i]$
  then error "new key is smaller than current key"
$A[i] \leftarrow key$
while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$
  do exchange $A[i] \leftrightarrow A[\text{PARENT}(i)]$
    $i \leftarrow \text{PARENT}(i)$

`i = 9, key = 15`
`15 > 4, no error`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	14	10	8	7	9	3	2	4	1	-	-	-	-	-	-

Example: Increase-Key

`Increase-Key(A,9,15):`



(b)

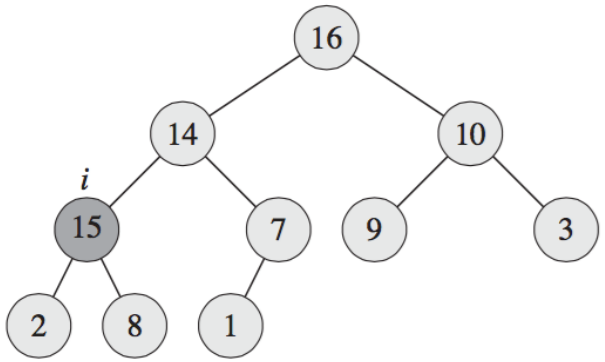
HEAP-INCREASE-KEY(A, i, key)
if $key < A[i]$
  then error "new key is smaller than current key"
$A[i] \leftarrow key$
while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$
  do exchange $A[i] \leftrightarrow A[\text{PARENT}(i)]$
    $i \leftarrow \text{PARENT}(i)$

`A[9] = 15`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	14	10	8	7	9	3	2	15	1	-	-	-	-	-	-

Example: Increase-Key

`Increase-Key(A,9,15):`



(c)

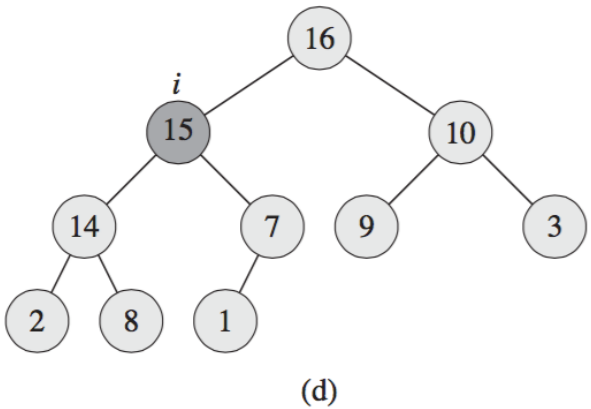
HEAP-INCREASE-KEY(A, i, key)
if $key < A[i]$
  then error "new key is smaller than current key"
$A[i] \leftarrow key$
while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$
  do exchange $A[i] \leftrightarrow A[\text{PARENT}(i)]$
    $i \leftarrow \text{PARENT}(i)$

```
A[4] < A[9]
swap A[9], A[4]
i = 4
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	14	10	15	7	9	3	2	8	1	-	-	-	-	-	-

Example: Increase-Key

`Increase-Key(A,9,15):`



(d)

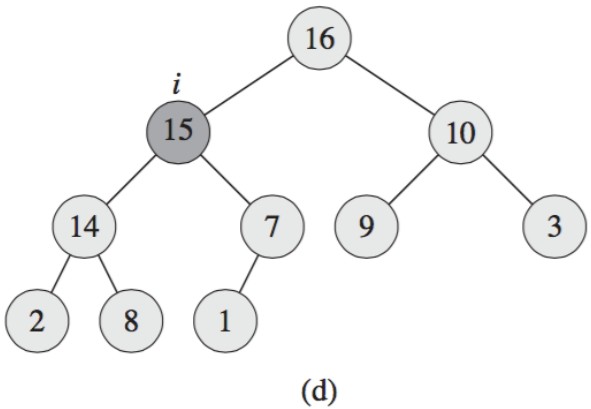
HEAP-INCREASE-KEY(A, i, key)
if $key < A[i]$
  then error "new key is smaller than current key"
$A[i] \leftarrow key$
while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$
  do exchange $A[i] \leftrightarrow A[\text{PARENT}(i)]$
    $i \leftarrow \text{PARENT}(i)$

```
A[2] < A[4]
swap A[4], A[2]
i = 2
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	15	10	14	7	9	3	2	8	1	-	-	-	-	-	-

Example: Increase-Key

`Increase-Key(A,9,15):`



(d)

HEAP-INCREASE-KEY(A, i, key)

if $key < A[i]$
  then error "new key is smaller than current key"
$A[i] \leftarrow key$
while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$
  do exchange $A[i] \leftrightarrow A[\text{PARENT}(i)]$
    $i \leftarrow \text{PARENT}(i)$

`A[1] > A[2]`
`while done`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	15	10	14	7	9	3	2	8	1	-	-	-	-	-	-

Max-Heap Implementation

`Insert(A,x,k):`

MAX-HEAP-INSERT(A, key)

1 $A.heap\text{-}size = A.heap\text{-}size + 1$
2 $A[A.heap\text{-}size] = -\infty$
3 HEAP-INCREASE-KEY$(A, A.heap\text{-}size, key)$

Time: $O(\lg n)$.

Priority Queue

In summary, a heap can support any priority-queue operations in `O(log n)` time.

Exercise 1

- How to implement a standard queue (FIFO) with a priority queue?

Exercise 2

- How to implement a stack (LIFO) with a priority queue?