Priority Queue

- Properties
  - Each element is associated with a value (priority)
  - The key with the highest (or lowest) priority is extracted first

a priority queue is no longer FIFO!

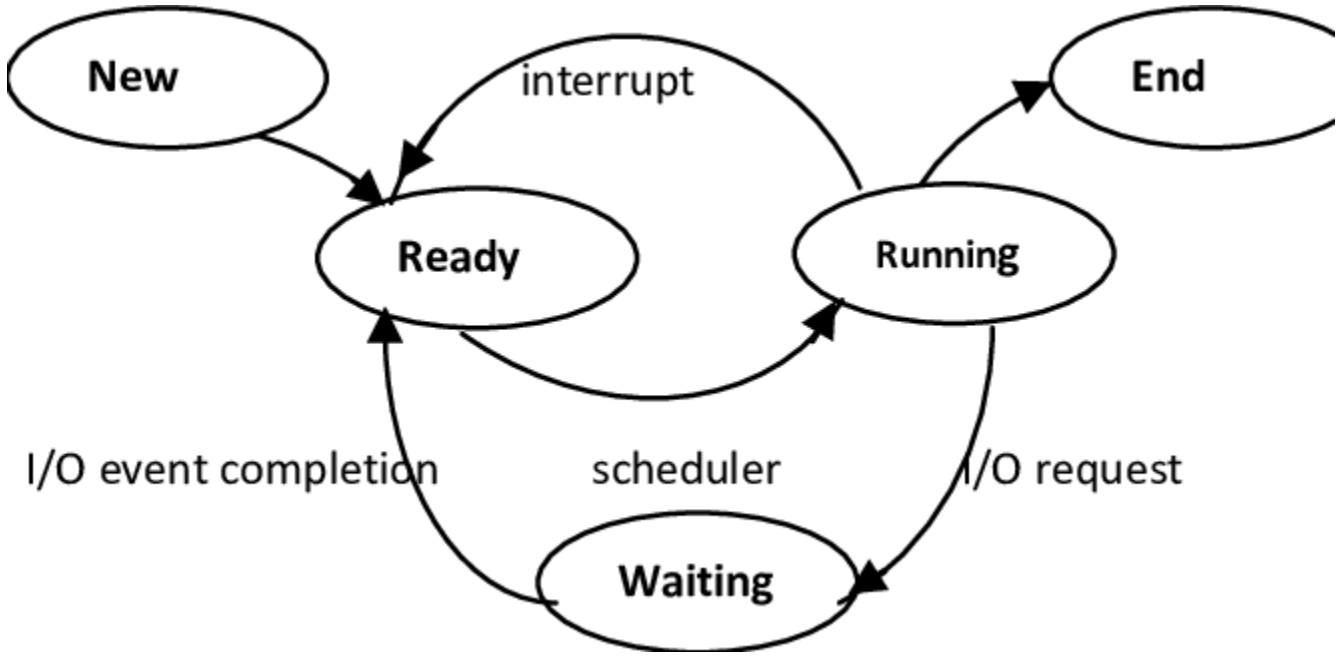
Priority Queues

- Popular & important application of heaps.
- Max and min priority queues.
- Maintains a *dynamic* set *S* of elements.
- Each set element has a *key* (= an associated value.)
- Goal is to support insertion and extraction efficiently

Applications of Priority Queues

- Any event/job management that assign priority to events/jobs
- In Operating Systems
  - Scheduling jobs
  - Ready list of processes in operating systems by their priorities (the list is highly dynamic)
- In Simulators
  - In event-driven simulators to maintain the list of events to be simulated in order of their time of occurrence.

Process state diagram



Basic Operations

- Operations on a max-priority queue:
  - Insert(S, x) : inserts the element x into the set S $S \leftarrow S \cup \{x\}$.
  - Maximum(S) : returns the element of S with the largest key.
  - Extract-Max(S) : removes and returns the element of S with the largest key.
  - Increase-Key(S, x, k): increases the value of element x's key to the new value k.
- Similarly, min-priority queue supports Insert, Minimum, Extract-Min, and Decrease-Key.

HEAP-MAXIMUM

```
HEAP-MAXIMUM(A)
1    return A[1]
```

- Returns the item at the top of the heap
- Runs in $\Theta(1)$ time

Heap-Extract-Max(*A*)

Heap-Extract-Max(*A*)

1. if *heap-size*[*A*] < 1
2. then error “heap underflow”
3. *max* ← *A*[1]
4. *A*[1] ← *A*[*heap-size*[*A*]]
5. *heap-size*[*A*] ← *heap-size*[*A*] - 1
6. MaxHeapify(*A*, 1)
7. return max

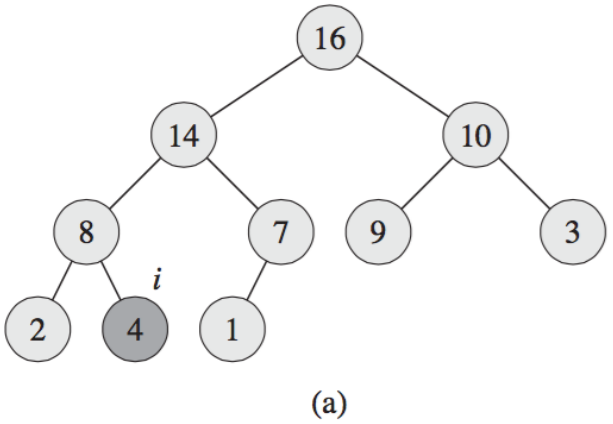
Running time : Dominated by the running time of MaxHeapify
= $O(\log n)$

Heap-Increase-Key(*A*, *i*, *key*)

```
Heap-Increase-Key(A, i, key)
1    If key < A[i]
2       then error "new key is smaller than the current key"
3    A[i] ← key
4    while i > 1 and  A[Parent[i]] < A[i]
5          do exchange A[i] ↔ A[Parent[i]]
6             i ← Parent[i]
```

Example: Heap-Increase-Key

`Heap-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 $A[i] \leftrightarrow A[\text{PARENT}(i)]$

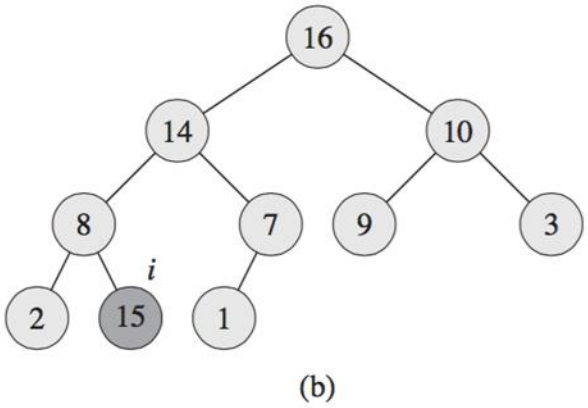
    $i \leftarrow \text{PARENT}(i)$

i = 9, key = 15

15 > 4, no error

Example: Heap-Increase-Key

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



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

if $key < A[i]$

  then error "new key is smaller than current key"

$A[i] \leftarrow key$ ← A[9] = 15

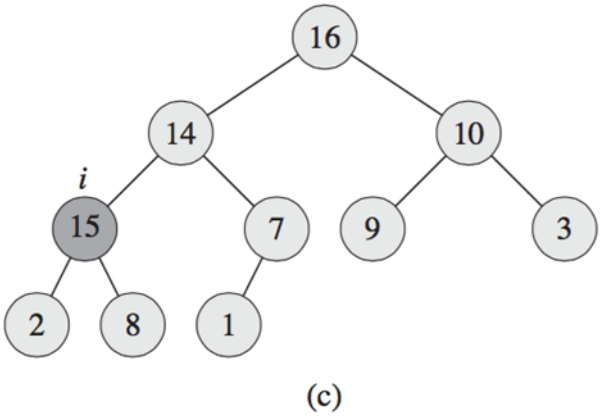
while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$

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

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

Example: Heap-Increase-Key

`Heap-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 $A[\text{PARENT}(i)] < A[i]$

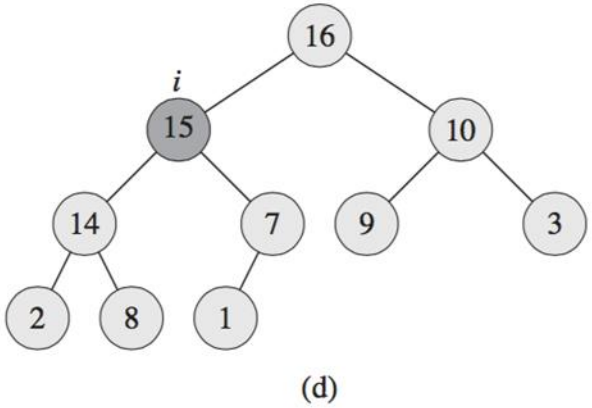
  do exchange $A[i] \leftrightarrow A[\text{PARENT}(i)]$

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

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

Example: Heap-Increase-Key

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



(d)

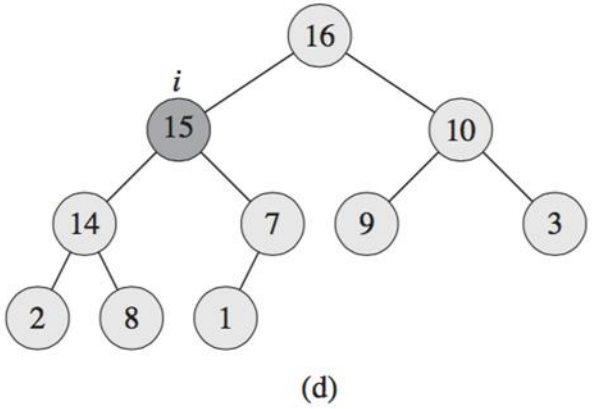
HEAP-INCREASE-KEY(A, i, key)

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

Example: Heap-Increase-Key

`Heap-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

Running time is $O(\log n)$

Heap-Insert(*A*, *key*)

```
Heap-Insert(A, key)
1    heap-size[A] ← heap-size[A] + 1
2    A[heap-size[A]] ← –∞
3    Heap-Increase-Key(A, heap-size[A], key)
```

Running time is $O(\log n)$

Summary

- We can perform the following operations on heaps:
  - Max-Heapify $O(\log n)$
  - BUILD-MAX-HEAP $O(n)$
  - HEAP-SORT $O(n \log n)$
  - HEAP-MAXIMUM $O(1)$
  - Heap-Extract-Max $O(\log n)$
  - Heap-Increase-Key $O(\log n)$
  - Heap-Insert $O(\log n)$

Conclusion

- what a heap is
- how to build a heap
- how to use a heap for sorting
- how to analyze heapsort’s running time
- how to use a heap for priority queues