Data Structures & Algorithms

16: Priority Queue

(Chapter 6 – CLRS)

Dr Ram Prasad Krishnamoorthy

Associate Professor
School of Computing and Data Science

ram.krish@saiuniversity.edu.in

SAI UNIVERSITY

Priority Queue

Priority queue is a data structure in which the elements has an associated priority (*key*).

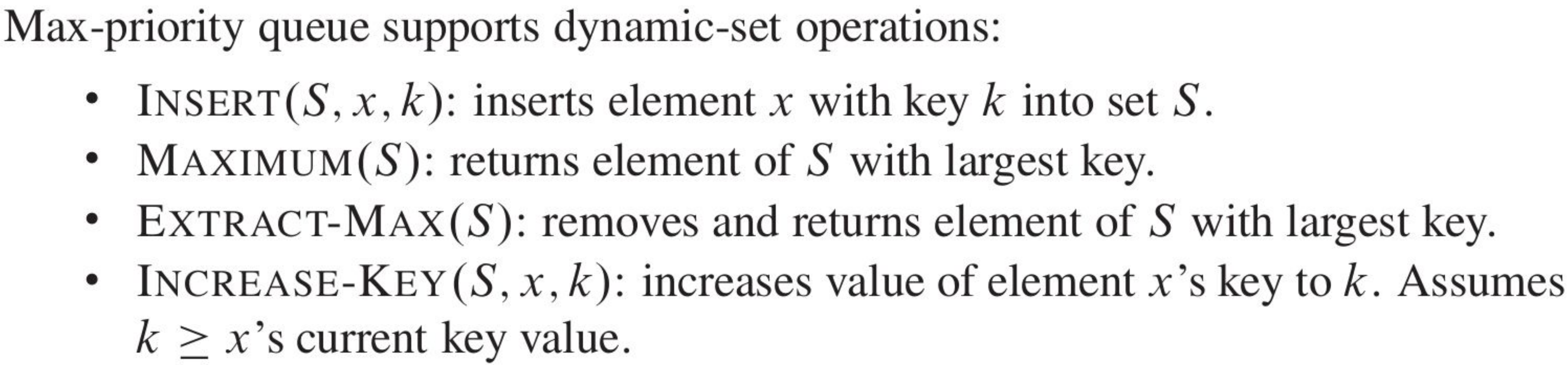
- Elements with high priority are served before elements with low priority.
- They are often implemented using heap.
- *Max-priority* queue and *min-priority* queue can be implemented.

Note: Elements in a priority queue are retrieved based on their priority, not their insertion order such as FIFO or LIFO.

Applications:

- **Job Scheduling in Operating Systems:** Tasks with higher deadlines or dependencies are prioritized in the queue.
- **Network Routing:** Packets might be prioritized based on their type (e.g., real-time video calls over file transfers) to maintain quality of service.

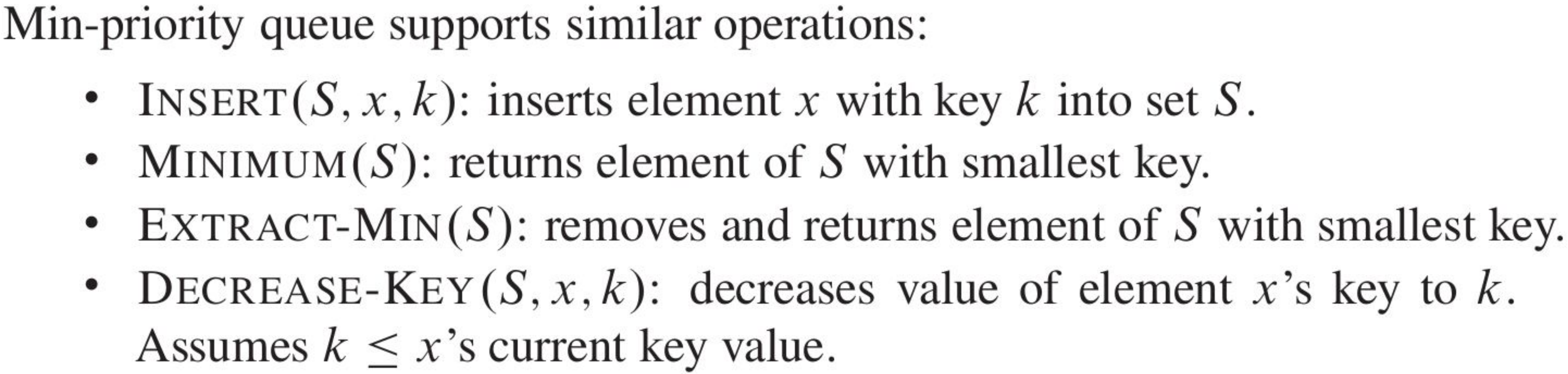
Priority Queue



Max-priority queue supports dynamic-set operations:

- INSERT(S, x, k): inserts element x with key k into set S.
- MAXIMUM(S): returns element of S with largest key.
- EXTRACT-MAX(S): removes and returns element of S with largest key.
- INCREASE-KEY(S, x, k): increases value of element x's key to k. Assumes $k \geq x$'s current key value.

PRIORITY QUEUE



Min-priority queue supports similar operations:

- INSERT(S, x, k): inserts element x with key k into set S.
- MINIMUM(S): returns element of S with smallest key.
- EXTRACT-MIN(S): removes and returns element of S with smallest key.
- DECREASE-KEY(S, x, k): decreases value of element x's key to k. Assumes $k \leq x$'s current key value.

HEAPS

Basic procedures in heap:

1. **MAX-HEAPIFY** → maintains max heap property.

2. **BUILD-MAX-HEAP** → produces max heap from unsorted input array.

3. **HEAPSORT** → sorts an array in place.

4. **MAX-HEAP-INSERT**

   MAX-HEAP-EXTRACT-MAX

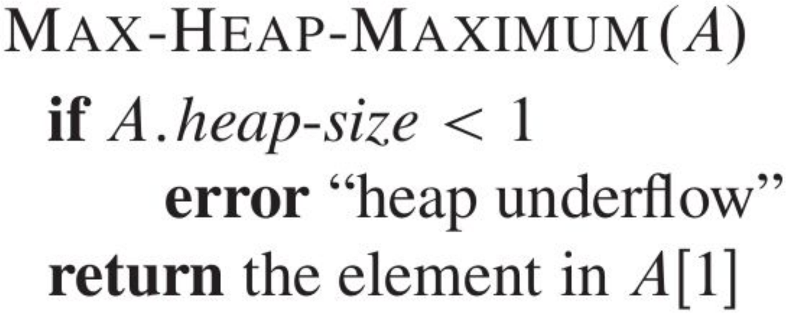
   MAX-HEAP-INCREASE-KEY

   MAX-HEAP-MAXIMUM

Implements Priority Queue

Max-Heap-Maximum

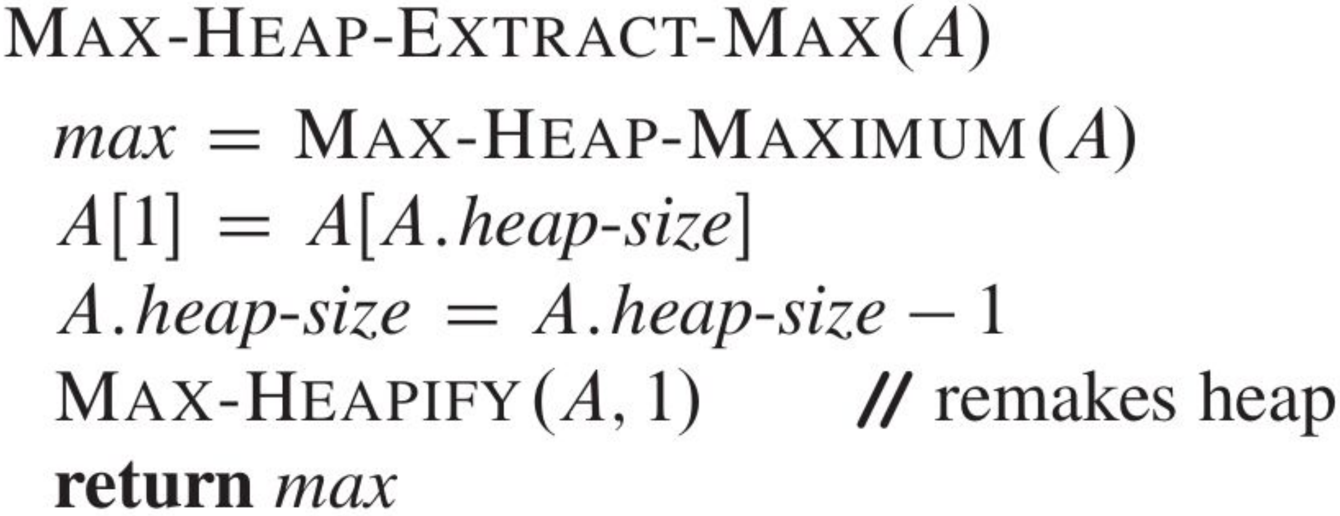
Priority Queue



```
MAX-HEAP-MAXIMUM($A$)
  if $A.heap\text{-}size < 1$
      error "heap underflow"
  return the element in $A[1]$
```

Max-Heap-Extract-Max

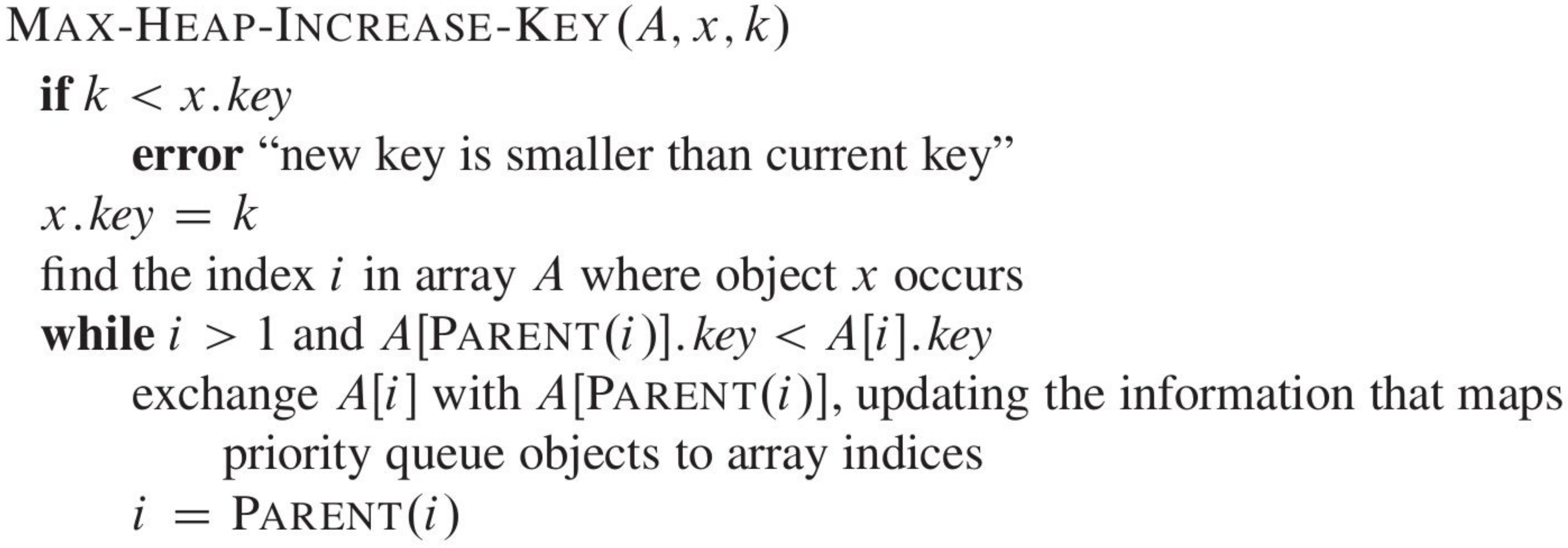
Priority Queue



```
MAX-HEAP-EXTRACT-MAX($A$)
  $max$ = MAX-HEAP-MAXIMUM($A$)
  $A[1] = A[A.heap\text{-}size]$
  $A.heap\text{-}size = A.heap\text{-}size - 1$
  MAX-HEAPIFY($A$, 1)        // remakes heap
  return $max$
```

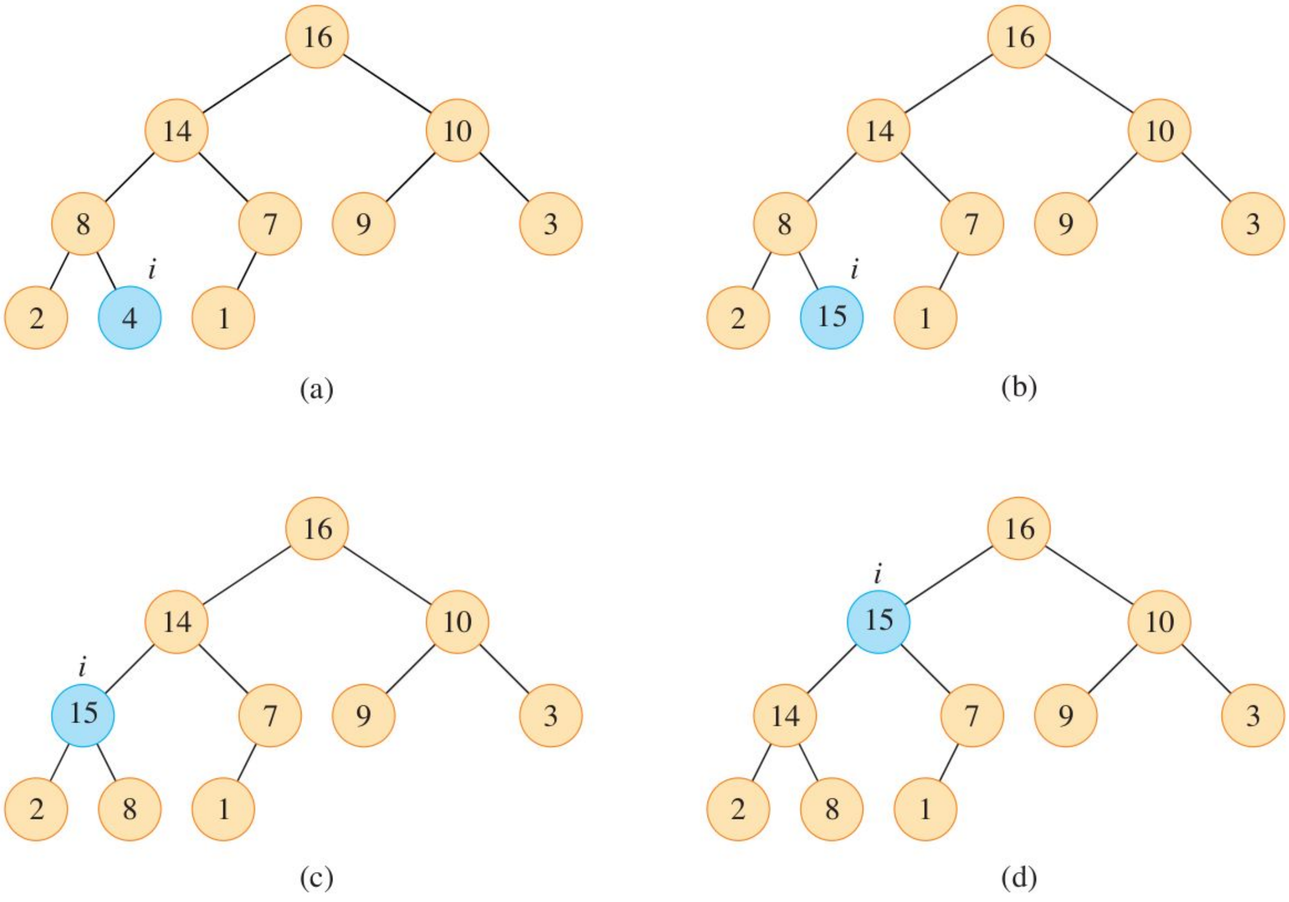
Max-Heap-Increase-Key

Priority Queue



```
MAX-HEAP-INCREASE-KEY$(A, x, k)$
  if $k < x.key$
      error "new key is smaller than current key"
  $x.key = k$
  find the index $i$ in array $A$ where object $x$ occurs
  while $i > 1$ and $A[\text{PARENT}(i)].key < A[i].key$
      exchange $A[i]$ with $A[\text{PARENT}(i)]$, updating the information that maps
          priority queue objects to array indices
      $i = \text{PARENT}(i)$
```

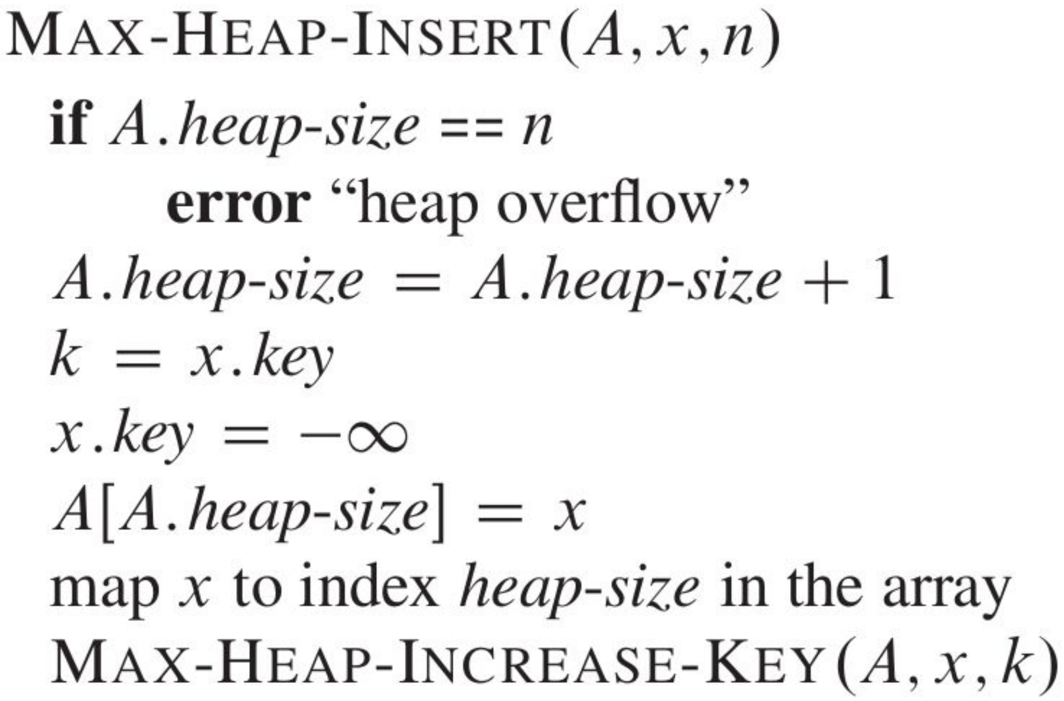
Priority Queue



(a) (b) (c) (d)

Max-Heap-Insert

Priority Queue



```
MAX-HEAP-INSERT$(A, x, n)$
  if $A.heap\text{-}size == n$
      error "heap overflow"
  $A.heap\text{-}size = A.heap\text{-}size + 1$
  $k = x.key$
  $x.key = -\infty$
  $A[A.heap\text{-}size] = x$
  map $x$ to index $heap\text{-}size$ in the array
  MAX-HEAP-INCREASE-KEY$(A, x, k)$
```