HEAPSORT

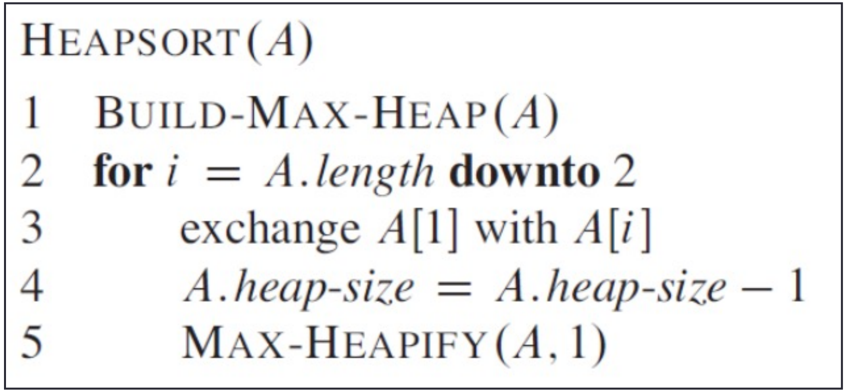
CS340

Heapsort Properties

- Combines the best properties of insertion sort and merge sort
  - O(n lg n) worst case
  - Sorts in place (no extra storage needed)

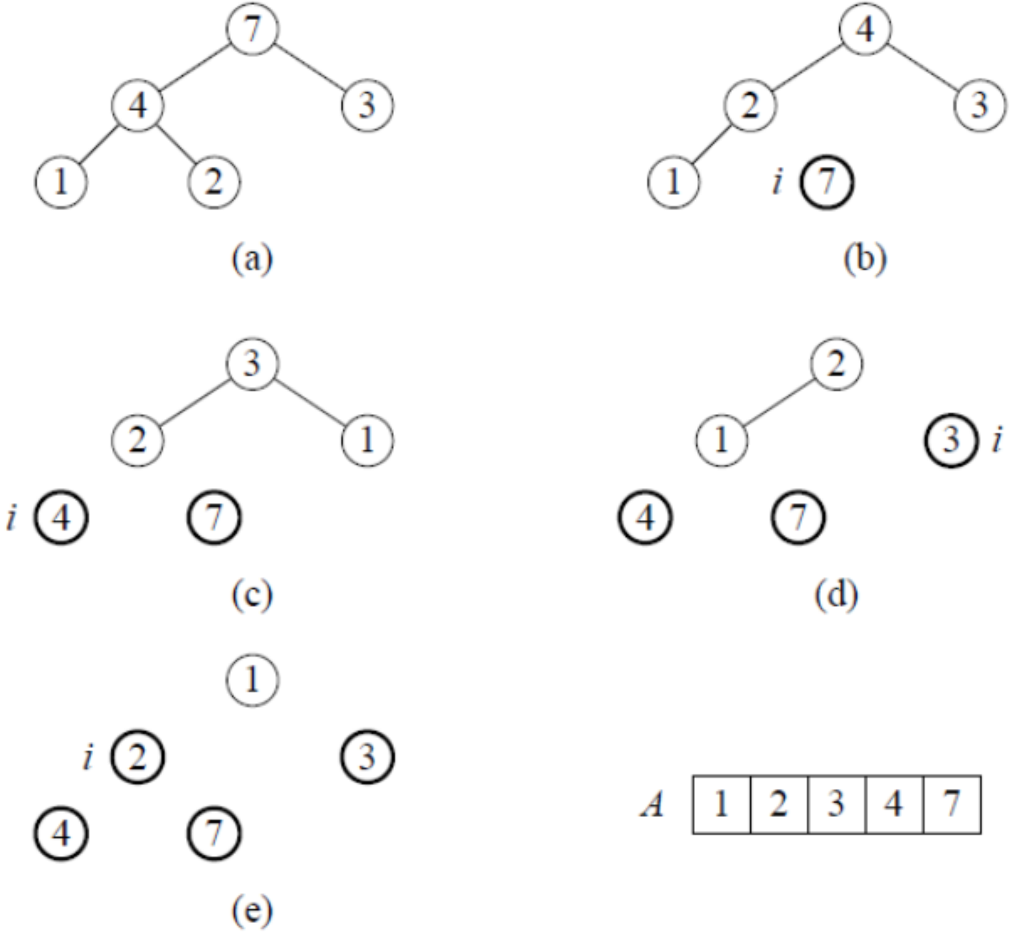
Heapsort

1. Build a max-heap from the array (O(n))
2. Place max element in its correct position by swapping it with the last item (O(1))
3. Decrease heap size by 1
4. Call max-heapify on root O(lg n) each time, n times



```
HEAPSORT($A$)
BUILD-MAX-HEAP($A$)
for $i = A.length$ downto 2
    exchange $A[1]$ with $A[i]$
    $A.heap\text{-}size = A.heap\text{-}size - 1$
    MAX-HEAPIFY($A, 1$)
```

Heapsort example



Priority Queues

- A **priority queue** is a data structure for maintaining a set S of elements, each with an associated value called a **key**.

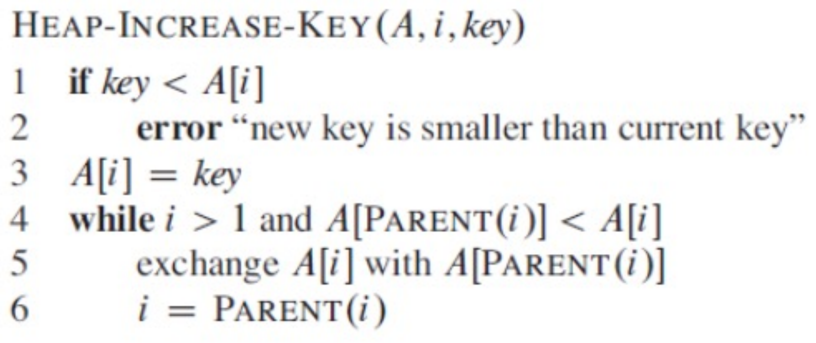
Max-Priority Queue

- A max-priority queue supports the following operations:
  - **INSERT(A, x)** inserts the element x into the array A
  - **MAXIMUM(A)** returns the element of A with the largest key.
  - **EXTRACT-MAX(A)** removes and returns the element of A with the largest key.
  - **INCREASE-KEY(A, i, k)** increases the value of element i’s key to the new value k, which is assumed to be at least as large as i’s current key value.

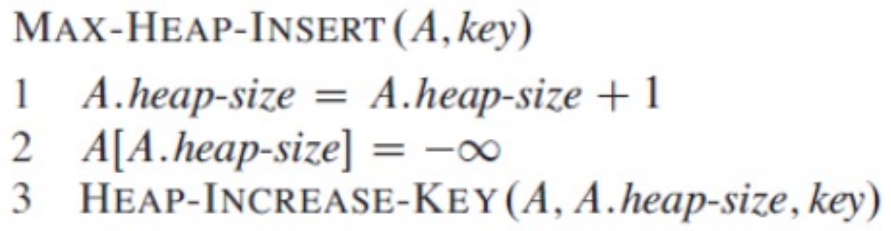
Priority Queue methods

```
<T> T maximum(T[ ] A) {

}
<T> T extractMax(T[ ] A) {

}
```

Priority Queue methods



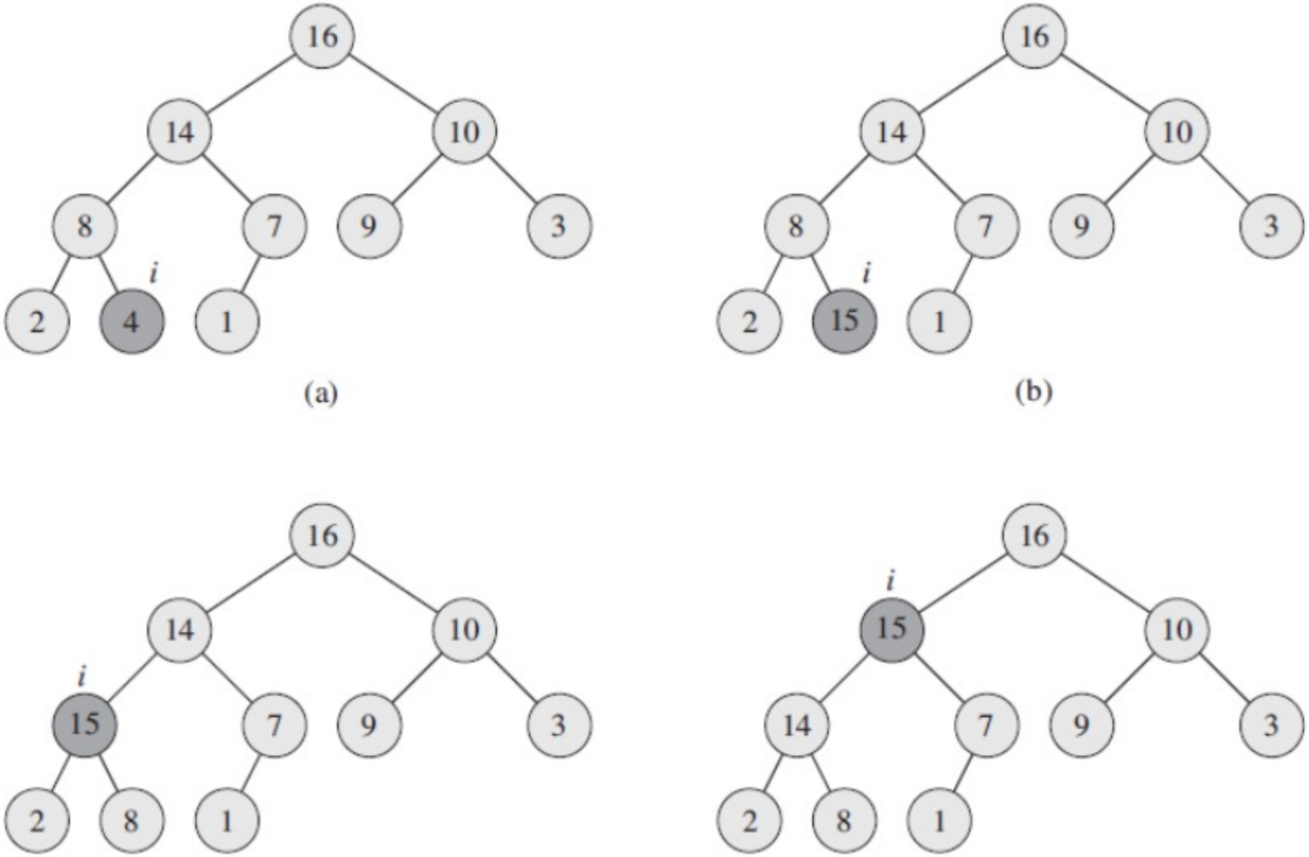
```
HEAP-INCREASE-KEY$(A, i, key)$
if $key < A[i]$
        error "new key is smaller than current key"
$A[i] = key$
while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$
        exchange $A[i]$ with $A[\text{PARENT}(i)]$
        $i = \text{PARENT}(i)$
```



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

Time complexities?

Increase-Key example



(a)

(b)

(c)

(d)