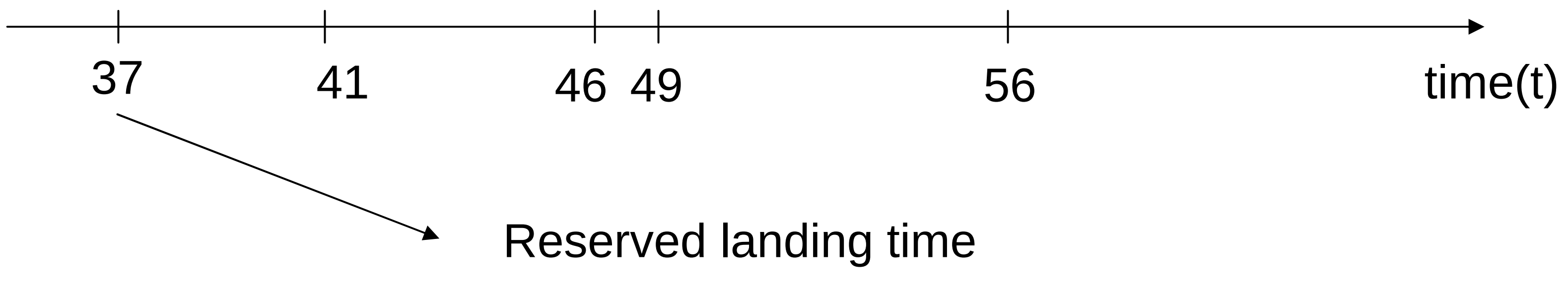
Priority queue & Heap

2008/3/7

Review L3: Runway reservation system

- Problem:
  - Airport with single runway
  - Reservation for future landings (insert)
  - When plane lands, it is removed from the set of pending events (extract-min)



Review: the proposed solutions

- Sorted array

1	2	3	4	5

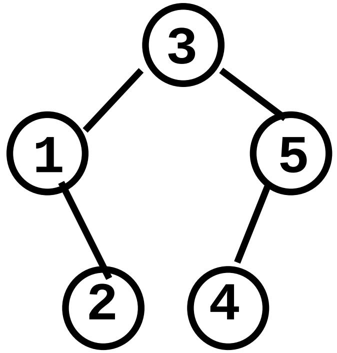
- Performance:
  - Insert: O(n)
  - Extract-min: O(1)

- Dictionary

key	value
1	
2	
3	
4	
5	

- Performance:
  - Insert: O(1)
  - Extract-min: O(n)
  - Good for searching, but not sorting

- BST



- Performance:
  - Insert: O(h)
  - Extract-min: O(h)
  - h=logN

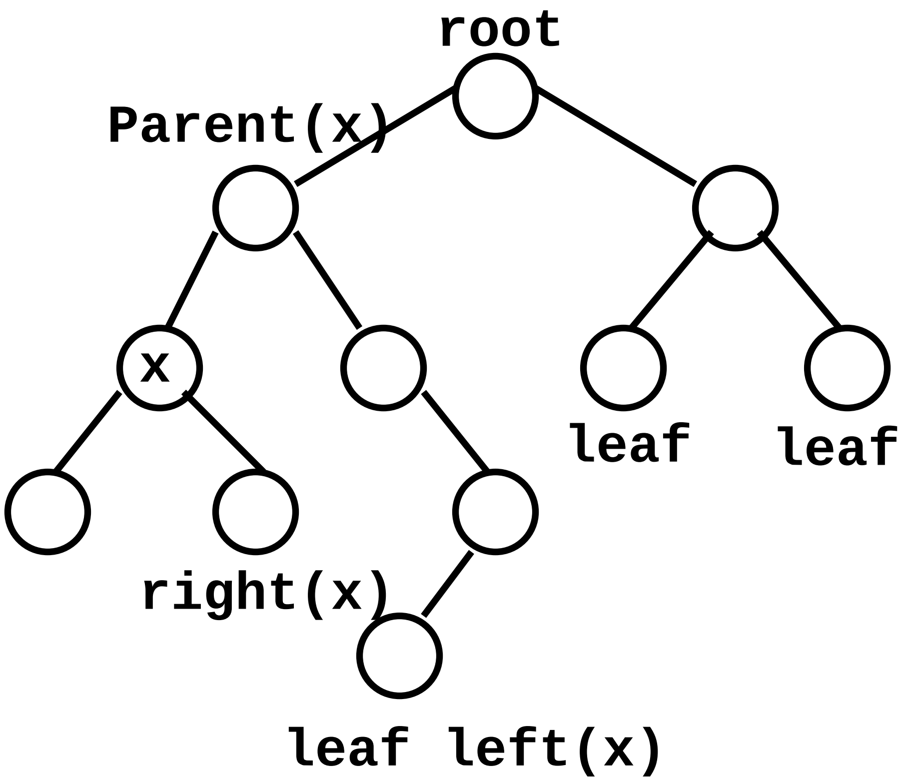
*** One more possible implementation: heap**

Review L8: Binary Heap

- definition: A Nearly complete binary tree
- Implementation: array!
- Property: Max-heap property
- Operations
  - Max-Heapify(A,i)
    - Maintain max-heap property of tree rooted at A[i]
  - Build-max-heap(A)
    - Convert an array A into a max-heap
  - Heapsort (A)
    - Sorting an array A

array

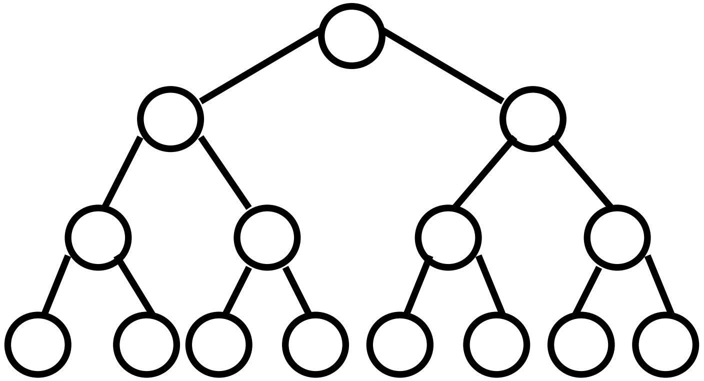
Binary Trees



- Binary search tree is a kind of binary tree.

Complete Binary Trees

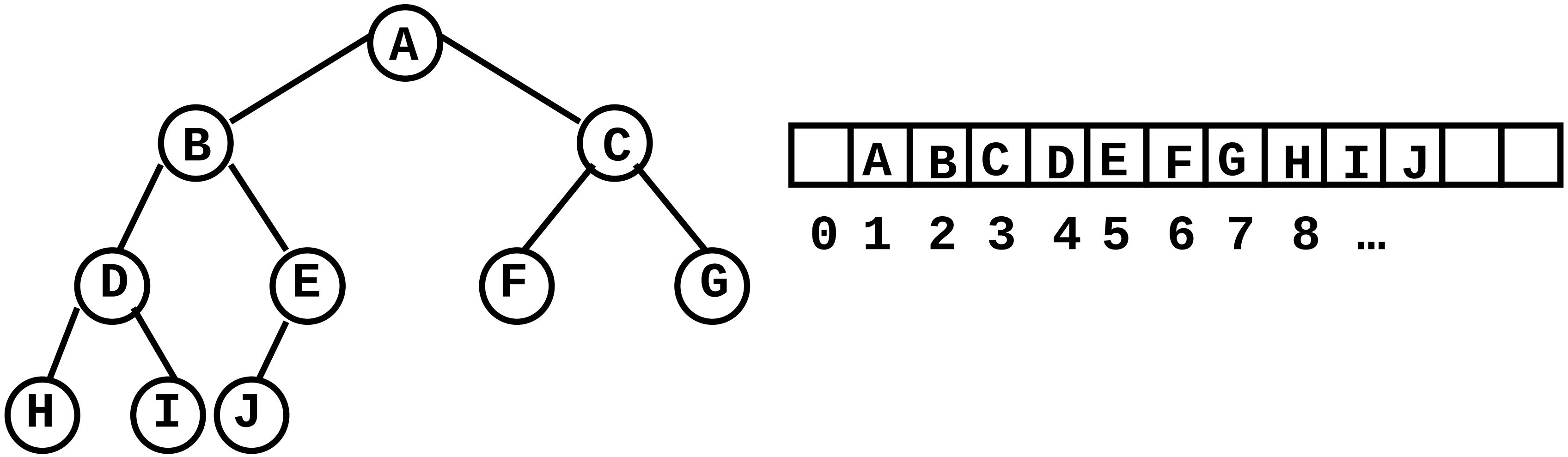
- Where a node can have 0 (for the leaves) or 2 children
- all leaves are at the same depth



height	no. of nodes
0	1
1	2
2	4
3	8
d	$\mathbf{2^d}$

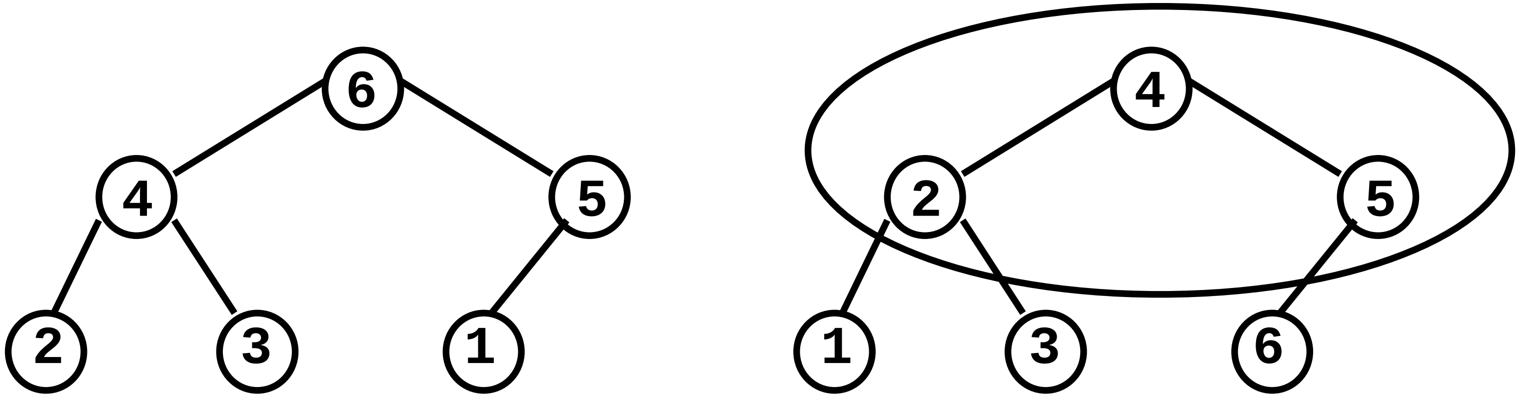
- A complete binary tree with N nodes has height `O(logN)`

Array Implementation of Binary Heap



- For any element in array position `i`
  - The left child is in position `2i`
  - The right child is in position `2i+1`
  - The parent is in position `floor(i/2)`

Max-Heap-order property



A max-heap Not a max-heap

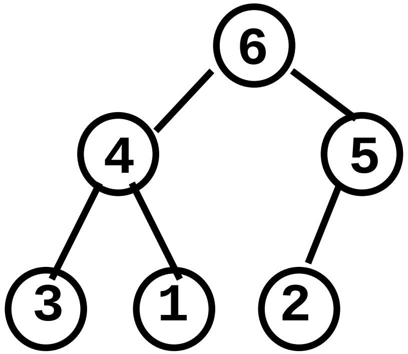
- the value at each node is less than or equal to the values at both its descendants

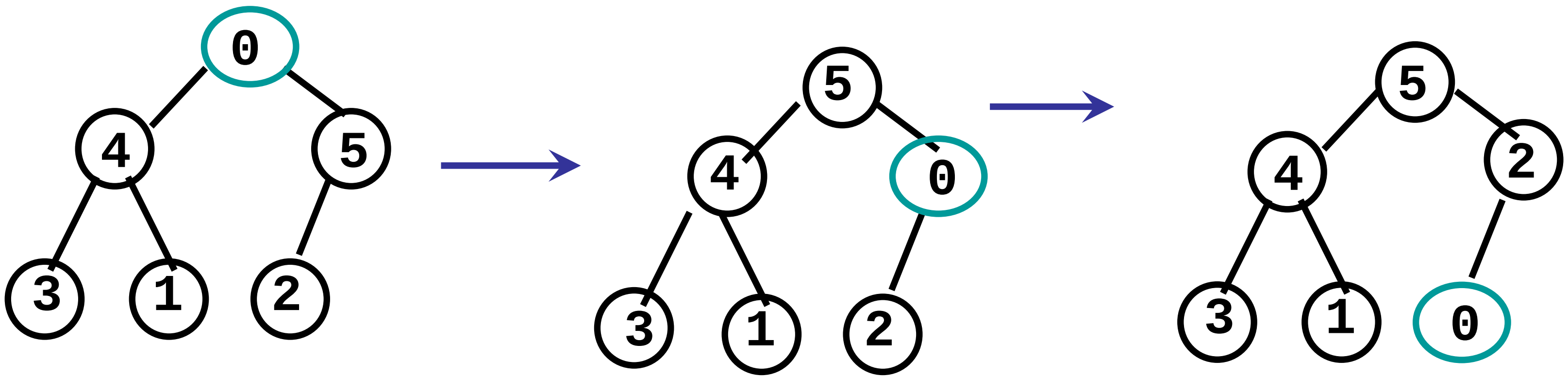
Review L8: Binary Heap

- definition: A Nearly complete binary tree
- Implementation: array!
- Property: Max-heap property
- Operations
  - Max-Heapify(A,i)
    - Maintain max-heap property of tree rooted at A[i]
  - Build-max-heap(A)
    - Convert an array A into a max-heap
  - Heapsort (A)
    - Sorting an array A

array

Max-heapify





Max-heapify(1)

Time: O(logN)

Max-heapify(A,i)

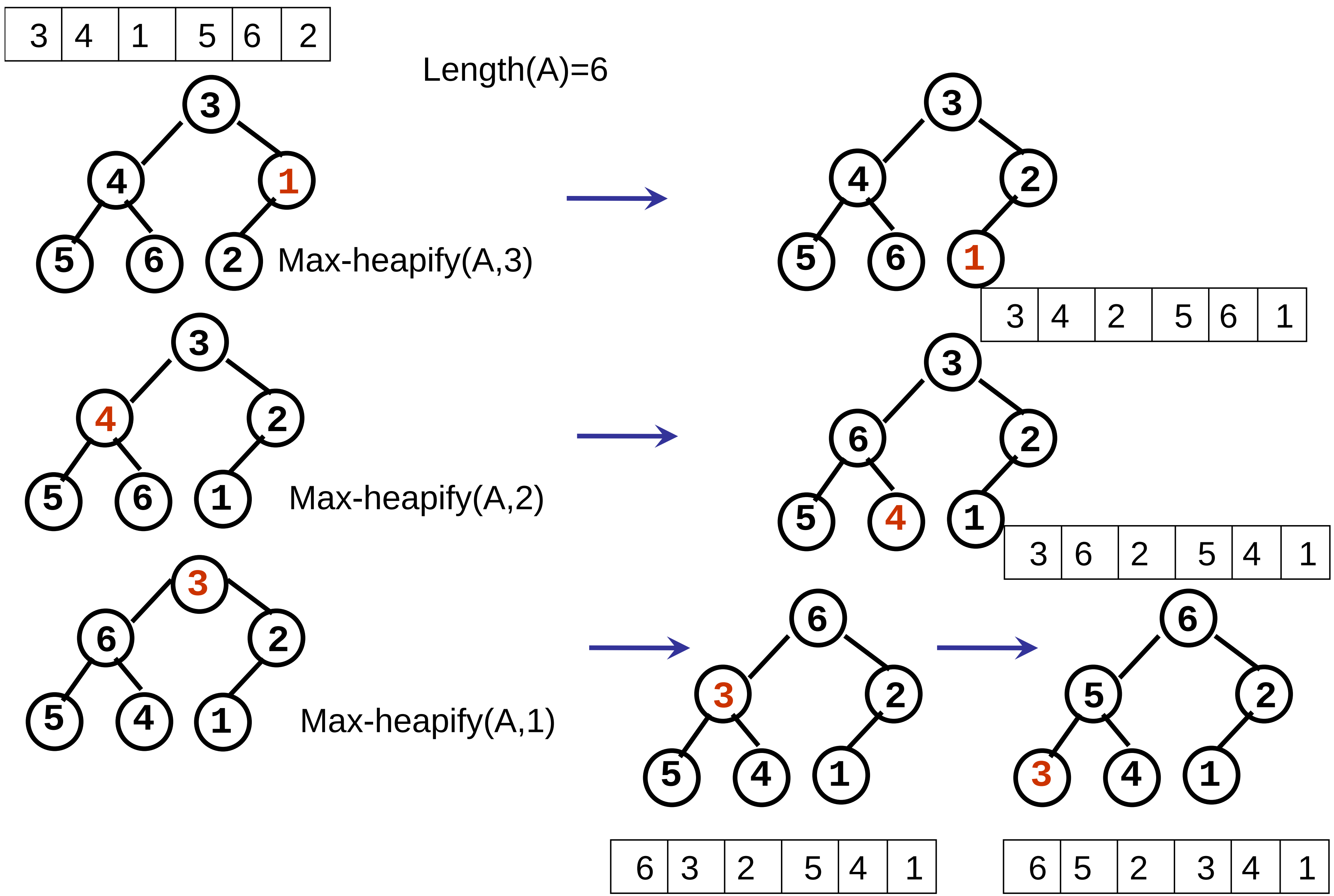
```
Max-heapify(A,i) #maintain max-heap property of tree rooted A[i]
    l<-left(i)
    r<-right(i)
    #compare the value with left child
    if l<=heap-size(A) and A[l]>A[i]
        then largest<-l
        else largest<-i
    #compare the value with right child
    if r<=heap-size(A) and A[r]>A[largest]
        then largest<-r
    # do swap if necessary and hen call max-heapify again
    if largest≠i
        then exchange A[largest]<->A[i]
             Max-heapify(A,largest)
```

Review L8: Binary Heap

- definition: A Nearly complete binary tree
- Implementation: array!
- Property: Max-heap property
- Operations
  - Max-Heapify(A,i)
    - Maintain max-heap property of tree rooted at A[i]
  - Build-max-heap(A)
    - Convert an **array** A into a max-heap
  - Heapsort (A)
    - Sorting an **array** A

array

Build max-Heap



Build-max-Heap(A)

Build-max-Heap(A) #Convert an array A into a max-heap

1. heap-size(A)<-length(A)

2. for i<- length(A)/2 down to 1

3. do Max-heapify(A,i)

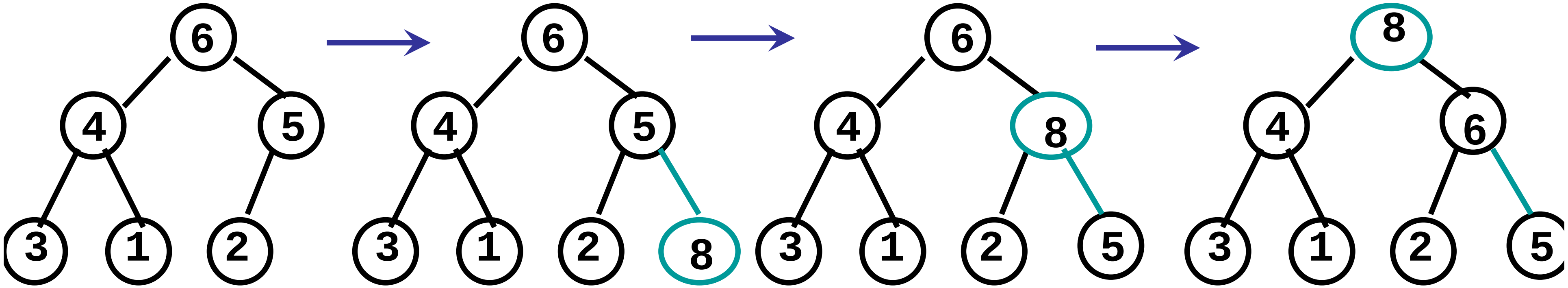
- Running time = O(n)
- * can also start from I to 1 down to length(A)/2, but that may need more calls for the method "max-heapify"

Heap is used to implement a priority queue, and runway-system is a kind of priority queue

Priority queue

- A priority queue is a data structure for maintaining a set of elements with values called keys
- Application:
  - Scheduling system (Runway system)
- Main Operations:
  - Insert (A,k)
    - Insert an element with key k into the **heap** A
  - extract-max(A)
    - Return element with the largest key from **heap** A
  - increase-value(A,I,k)
    - Update the key of element A[i] to a new value , k.
- Implementation
  - binary search tree
  - heap

insert



Insert 8

Insert(A,k)

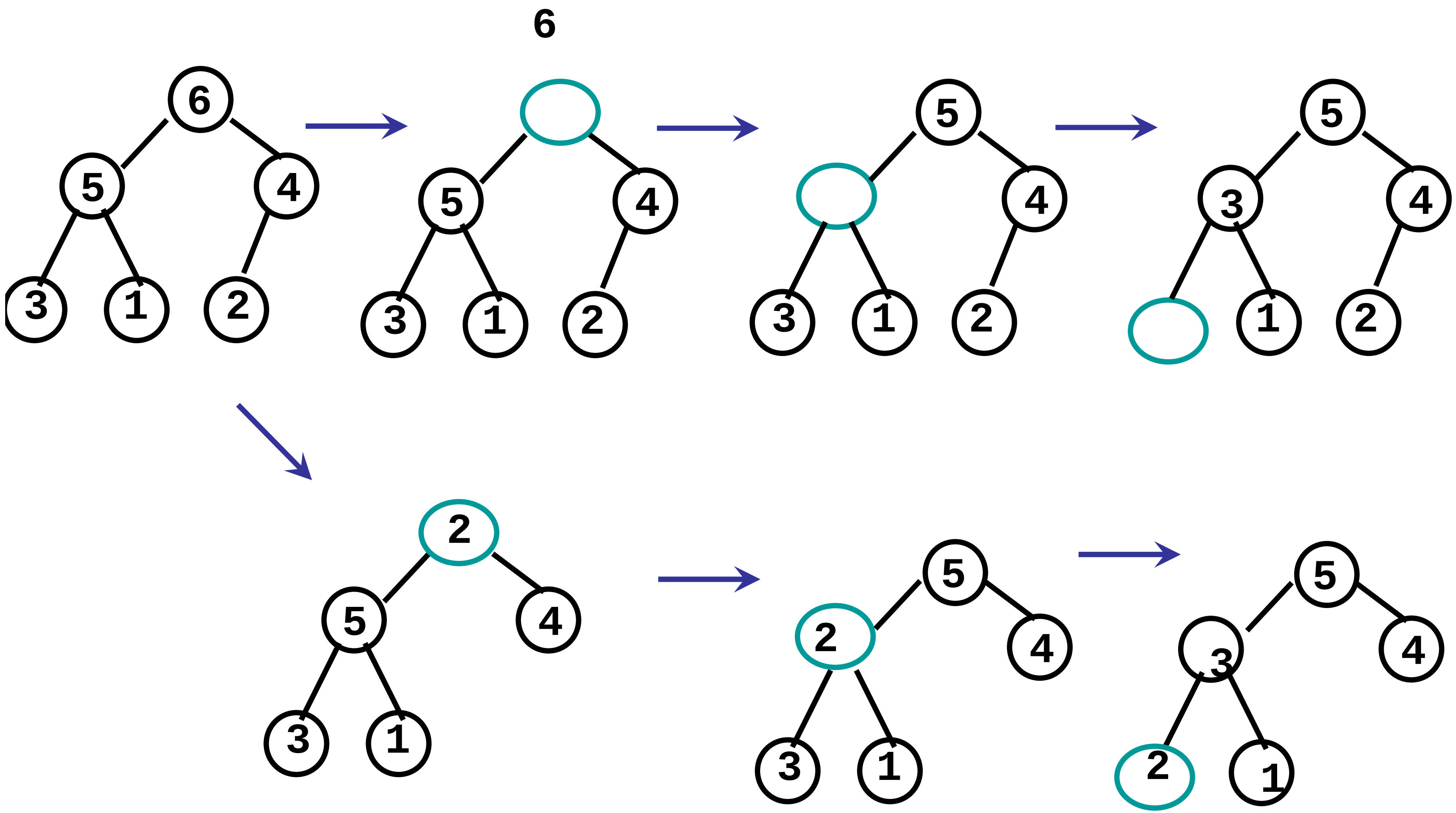
Time: O(logN)

```
insert(A,k) # Insert an element  with key k into heap A

heap-size[A] <- heap-size[A] +1
i <-heap-size[A]
A[i]=k
# Insert it to next available position at the lowest level
While i>1 and A[parent(i)]<A[i]
# traverse a path from this node toward the root to find
a proper place until the max-heap property is maintained
        do exchange A[parent(i)]<->A[i]
        i<-parent(i)
```

Priority queue

- A priority queue is a data structure for maintaining a set of elements with values called keys
- Application:
  - Scheduling system (Runway system)
- Main Operations:
  - Insert (A,k)
    - Insert an element with key k into the heap A
  - extract-max(A)
    - Return element with the largest key from heap A
  - increase-value(A,I,k)
    - Update the key of element A[i] to a new value , k.
- Implementation
  - binary search tree
  - heap

extract-max



extract-max(A)

Time: O(logN)

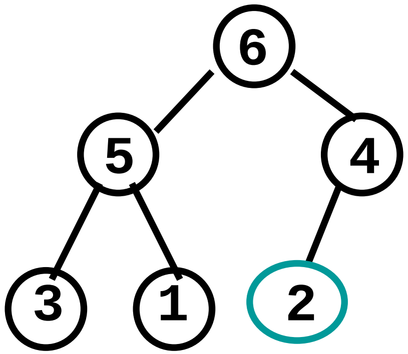
extract-max(A) #Return the largest key from heap A

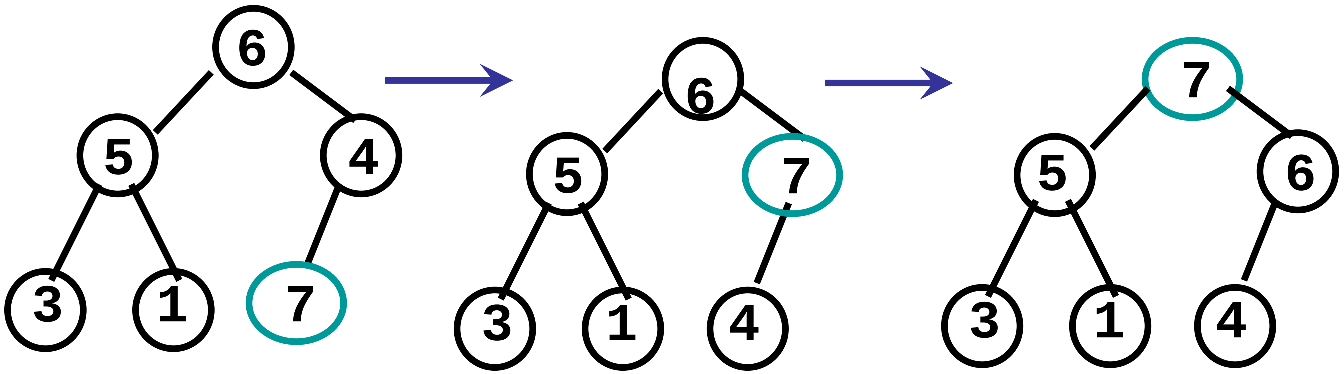
1. **max <- A[1]**
2. **A[1] <- A[heap-size(A)]**
3. **heap-size(A) <- heap-size(A)-1**
4. **Max-heapify(A,1)**
5. **return max**

Priority queue

- A priority queue is a data structure for maintaining a set of elements with values called keys
- Application:
  - Scheduling system (Runway system)
- Main Operations:
  - Insert (A,k)
    - Insert an element with key k into the heap A
  - extract-max(A)
    - Return element with the largest key from heap A
  - increase-value(A,I,k)
    - Update the key of element A[i] to a LARGER value , k.
- Implementation
  - binary search tree
  - heap

Increase-key





increase-key(A,i,k)

Time: O(logN)

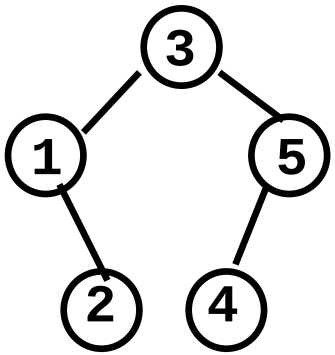
```
increase-key(A,i,k)   # Update the key of A[i]  to a new value , k.

A[i] <-k
While i>1 and A[parent(i)]<A[i]
# traverse a path from this node toward the root to find
a proper place until the max-heap property is maintained
        do exchange A[parent(i)]<->A[i]
        i<-parent(i)
```

*Very similar to insert

*Do max-heapify when decrease-key(A,I,k)

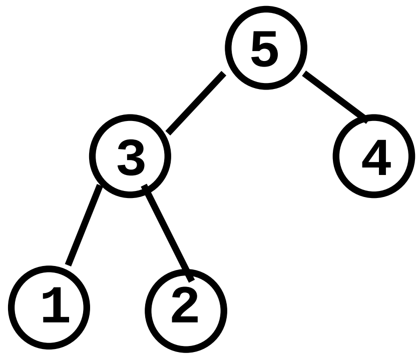
Compare two implementations of priority queue

- ❑ BST



- ❑ Performance:
  - ❑ Insert: O(h)
  - ❑ Extract-min: O(h)
  - ❑ h=logN (if BST is balanced)

- **Heap**



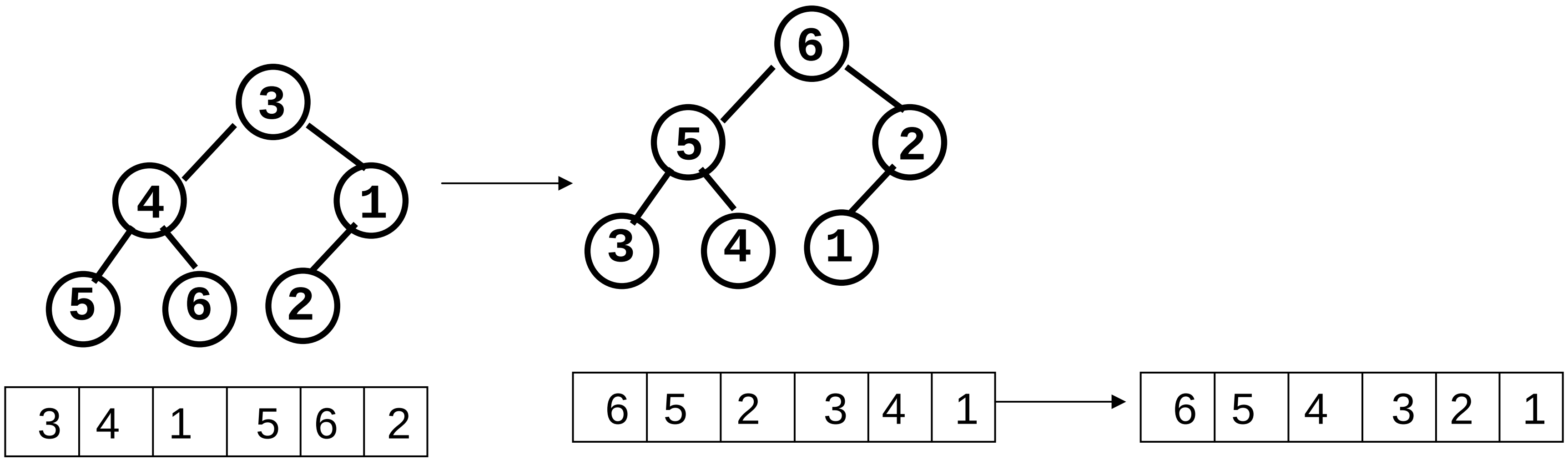
- ❑ Performance:
  - ❑ Insert: O(h)
  - ❑ Extract-min: O(h)
  - ❑ h=logN

Why is heap a better implementation of priority queue than BST?

Heap is used for sorting arrays

Sorting

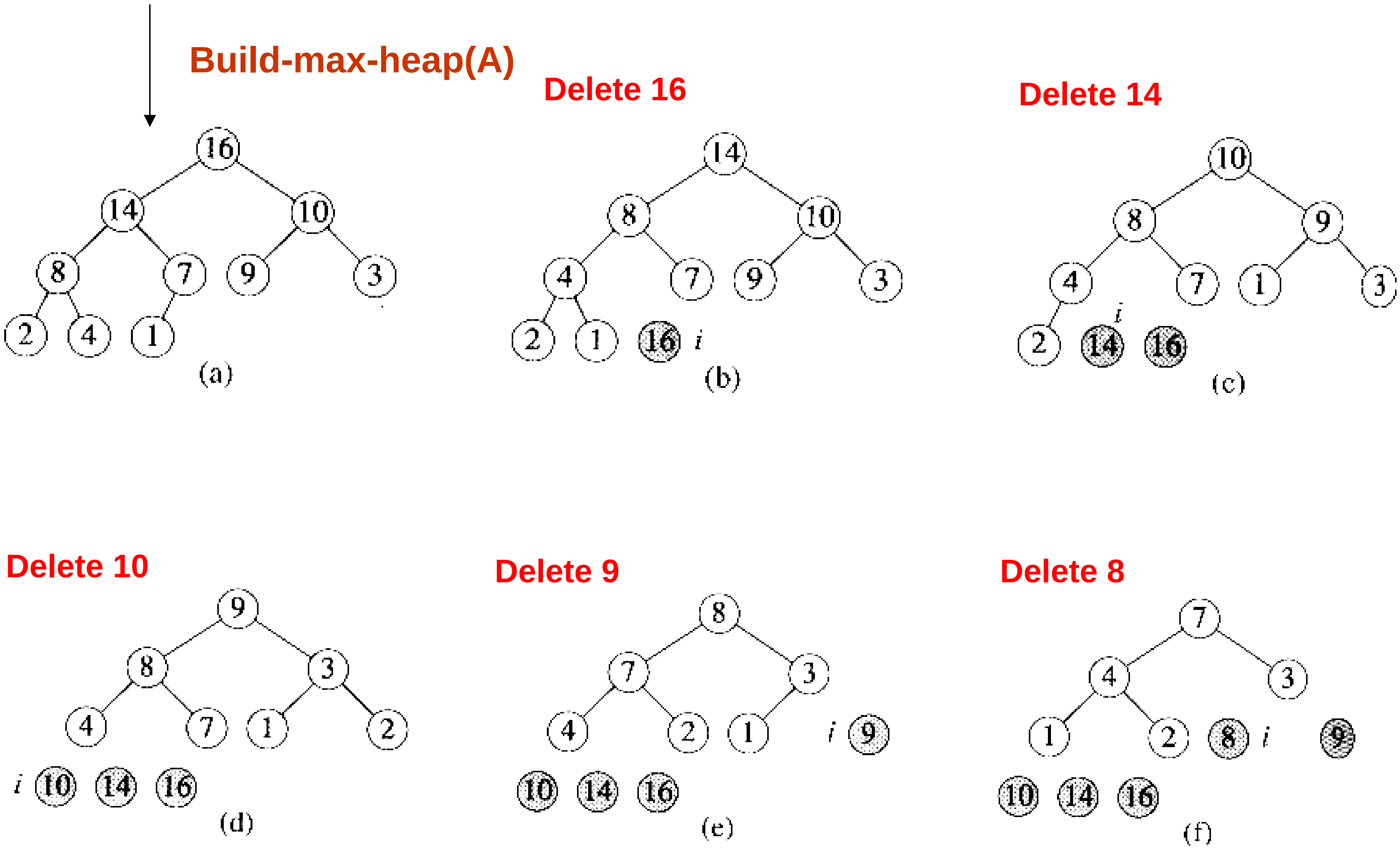
- Problem: Given an array A, return an sorted array



Build-max-heap is not all

Heapsort (A)

[1,2,3,4,7,8,9,10,14,16]



Time: O(NlogN)

Heapsort(A)

```
Heapsort(A) #sort an array A
    Build-max-Heap(A)     # the heap is built in A
    while heap-size(A)>1
        max=extract-max(A)
        A[heap-size(A)]=max
```

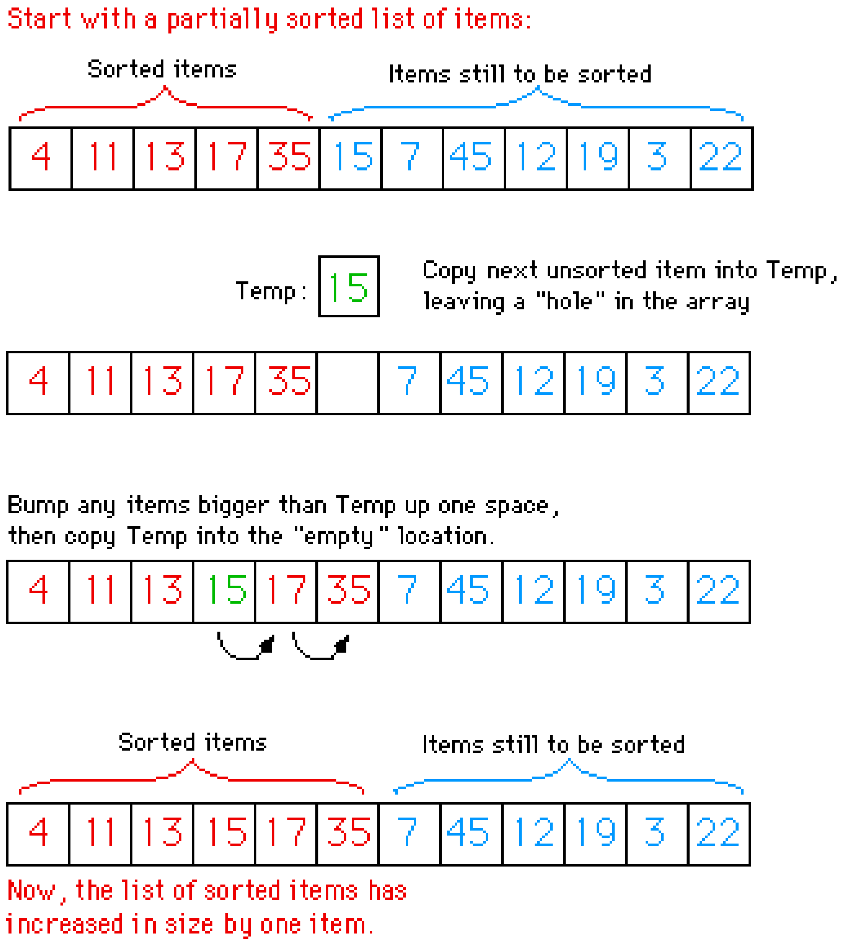
Heapsort(A)

```
Heapsort(A) #sort an array A
1.    Build-max-Heap(A)    # the heap is built in A
2.    for i<- length(A) down to 2
3.        do exchange A[1]<->A[i]
4.            heap-size(A)<-heap-size(A)-1
5.            Max-heapify(A,1)
```

Review: all the sorting algorithms so far

*All sorting algorithm stores data in an array, but heap sort is efficient cause it can be thought as a tree)

Review sorting Alg(1)-insertion sort



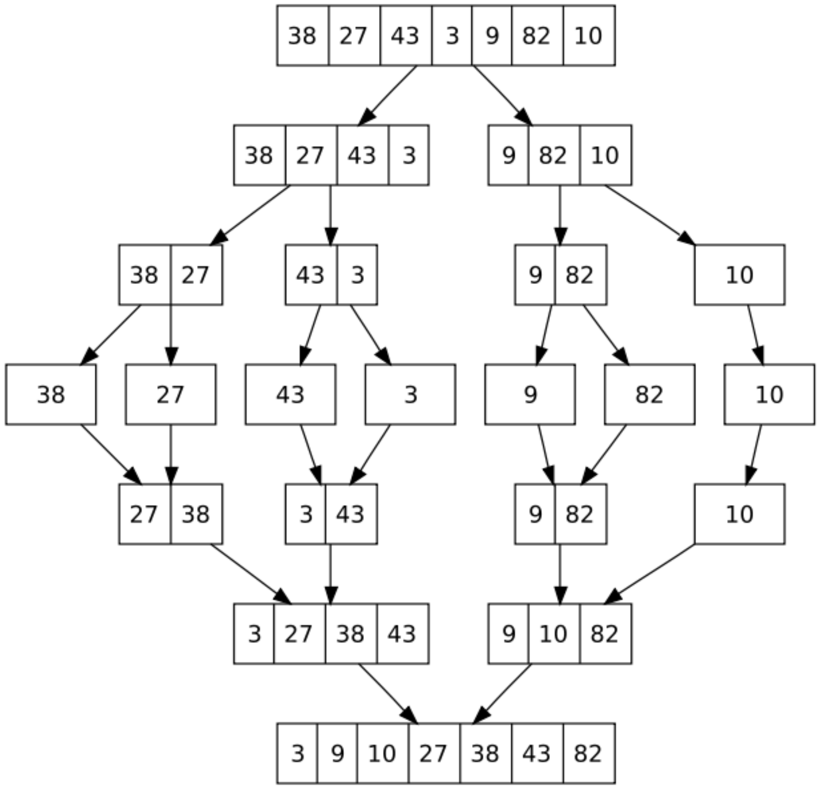
Running Time:

Worst: $(n-1)+(n-2)+\ldots+(1)=(n-1)(n)/2=O(n^2)$

Best: $1+1+1\ldots\ldots+(n-1)=O(n)$

Sorting Alg	Insertion	Merge	selection	Heap
Memory	O(1)	O(n)	O(1)	O(1)
Time	$O(n^2)$	O(nlogn)	$O(n^2)$	O(nlogn)

Review sorting Alg(2)-merge sort

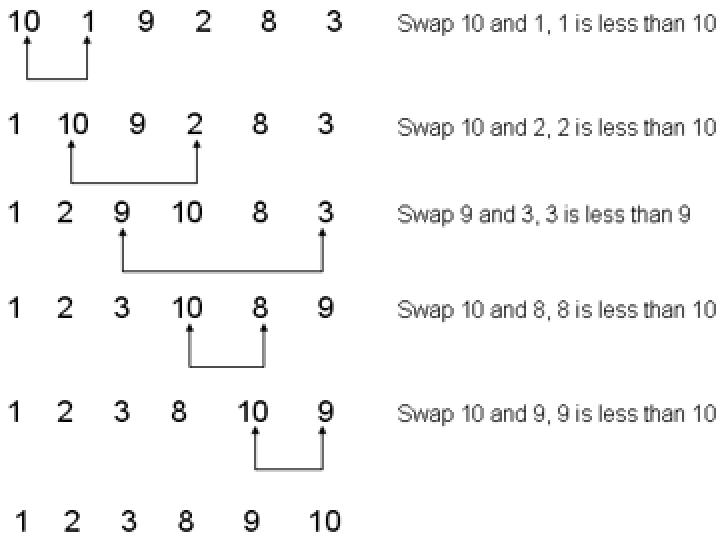


Running Time:

Worst, Best: n+n+n……+(n)=nlogn=O(nlogn)

Sorting Alg	Insertion	Merge	selection	Heap
Memory	O(1)	O(n)	O(1)	O(1)
Time	$O(n^2)$	O(nlogn)	$O(n^2)$	O(nlogn)

Review sorting Alg(3)-selection sort



Given an array A with elements,A[1],…A[n]

```
for i ← 1 to n-1
    do min ← i
    for j ← i+1 to n
        do if A[j]<A[min]
            min ← j
    A[i]↔A[min]
```

Running time:

Worst,best: $(n-1)+(n-2)+\ldots+(1)=(n1)(n)/2=O(n^2)$

Sorting Alg	Insertion	Merge	selection	Heap
Memory	$O(1)$	$O(n)$	$O(1)$	$O(1)$
Time	$O(n^2)$	$O(n\log n)$	$O(n^2)$	$O(n\log n)$

Conclusion

- Heap supports methods
  - Insert and extract-max
    - Applied to priority queue
    - Better than BST, sorted array
  - Heapsort
    - Applied to sorting
    - Better than insertion, merge, selection sort

Reference

- www.cs.ust.hk/~qyang/171/heapsort.ppt
- http://ww3.algorithmdesign.net/handouts/Heap.pdf