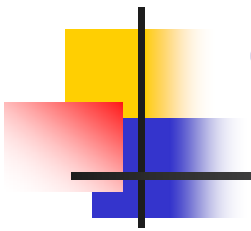
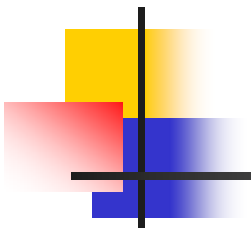


Chapter 6: Binary Heaps

- Data Structures and Algorithms
- Kyuseok Shim
- SoEECS, SNU.

Priority Queue ADT

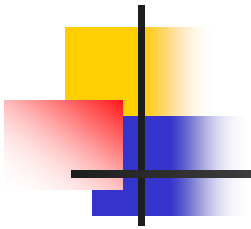


Priority Queue Q supports the following operations

- `Q.Insert(x)`
- `y = Q.DeleteMin()`

Simpler than Dictionary but important application areas

- Discrete Event Simulations (Ex: Airport)
- Process Scheduling



Binary Heaps

Binary Heap: A Complete, Partially Ordered Binary Tree

- Complete Tree: every level of the tree is completely filled, except for the bottom level, which is filled from left to right. (With height h, there are between 2^h and $2^{h+1}-1$ nodes)
- Partially Ordered Tree: the key of each internal node is less than or equal to the keys of its children

Binary Heap can be stored in arrays since it is complete!

Array Implementation of a Heap

Consider an indexing of nodes in heap from 1 to *n*

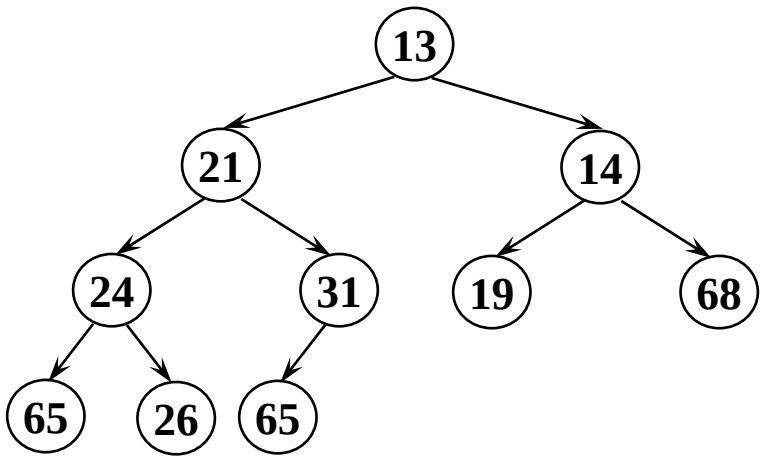
(so the root is numbered 1 and the last leaf is numbered *n*)

There is a simple math. relationship bet'n index of a node and that of its children

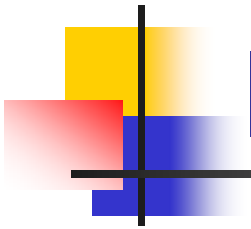
left_child(*i*): $2i$ **(if** $2i \leq n$**)**

right_child(*i*): $2i+1$ **(if** $2i+1 \leq n$**)**

parent(*i*): $\left\lfloor \frac{i}{2} \right\rfloor$ **(if** $i \geq n$**)**



array		13	21	14	24	31	19	68	65	26	32		
index	0	1	2	3	4	5	6	7	8	9	10	11



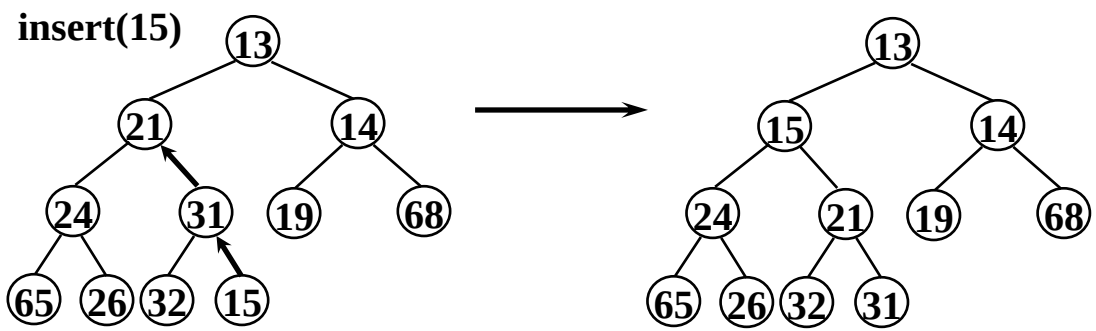
Insertion into a Heap

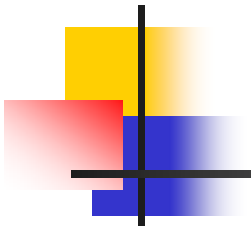
Put into the next available leaf node

(this is simple since it is array)

- Look at the parent of this element and swap if the parent is larger (note that the partial ordering is preserved)
- Repeat with the parent (*sifting up*)

Insertion into a Heap

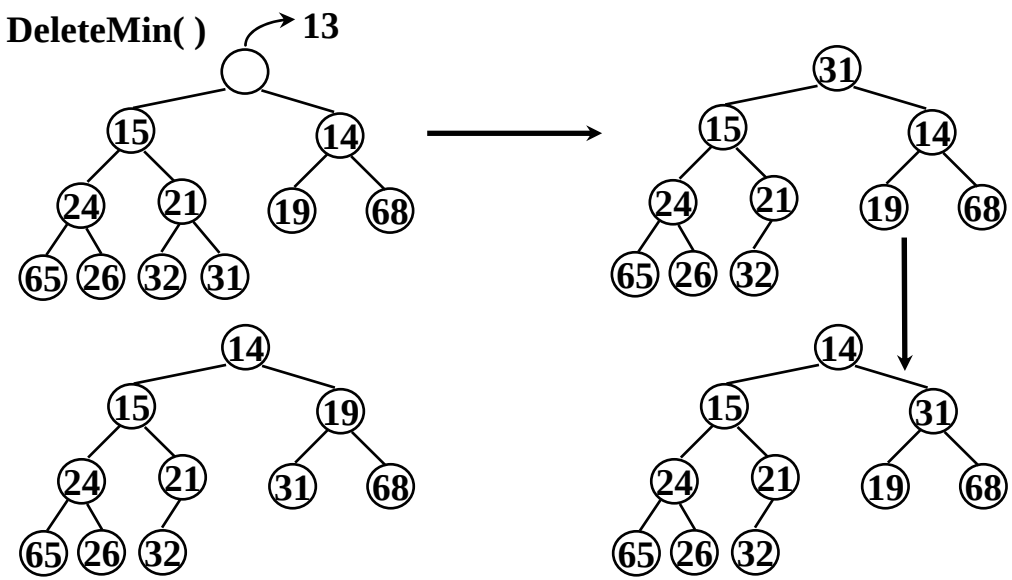




DeleteMin

- Remove the root from the tree and return its value
- Remove their rightmost leaf and place it at the root
- Perform *sifting down* operations (if the key is larger than the smaller of its two children, then swap these two. Repeat this)

DeleteMin



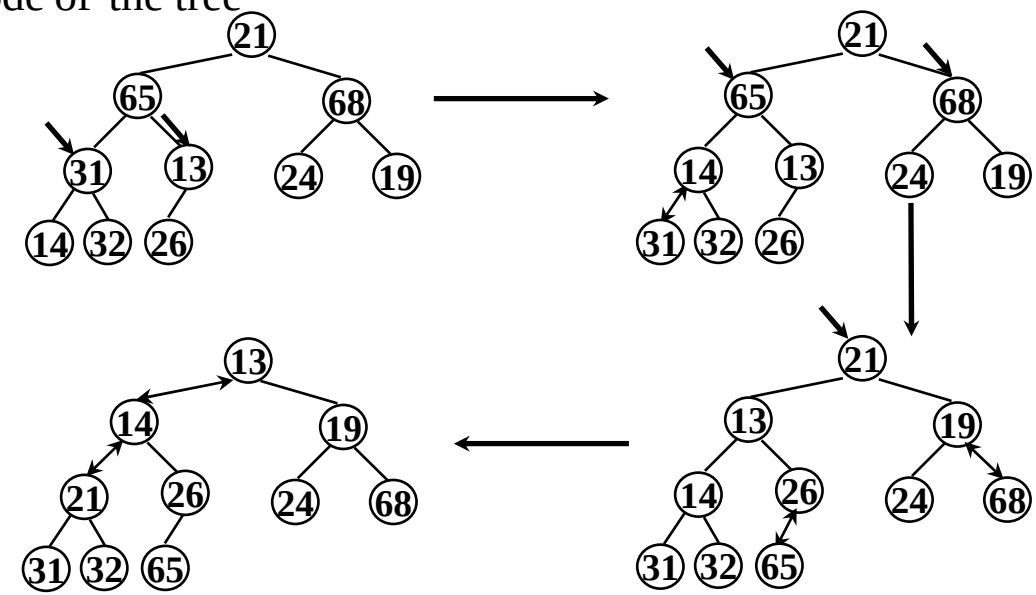
Analysis of Insertion and Deletion : $O(\log n)$

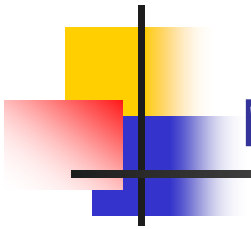
Building a heap

Build a Heap containing *n* keys

takes $O(n \log n)$ with consecutive insertions

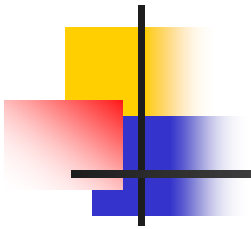
But, it can take $O(n)$ if they are already in an array. Starting with the lowest non-leaf node, working back towards root, perform shift-down on each node of the tree





MAX-HEAP

```
MAX-HEAPIFY(A, i)
1       L <- Left(i)
2       R <- Right(i)
3       If l <= heap-size[A] and A[l] > A[i]
4           then largest <- l
5           else largest <- i
6       If r <= heap-size[A] and A[r] > A[largest]
7           then largest <- r
8       If largest != i
9           then exchange A[i] <-> A[largest]
10              MAX-HEAPIFY(A, largest)
```



Build-Max-Heap(A)

1. Heap-size[A] <- length[A]
2. For i <- floor(length[A]/2) downto 1
3. Do Max-Heapify(A, i)

- Loop Invariant:
  - At the start of each iteration of the for-loop of lines 2-3, each node i+1, I+i,…,n is the root of a max-heap
  - Proof: Look at the book!

Analysis of Building a Heap

Let' s assume the tree is complete : $n = 2^{h+1} - 1$

There is one key at the level 0, which might shift down h levels

There is two key at the level 1, which might shift down $h-1$ levels

There is four key at the level 2, which might shift down $h-2$ levels

.....

$$S = h + 2(h-1) + 4(h-2) + \ldots + 2^{h-1}(1)$$

$$2S = 2h + 4(h-1) + 8(h-2) + \ldots + 2^{h-1}(2) + 2^h(1)$$

$$2S - S = S$$

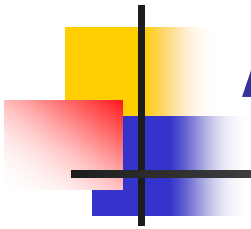
$$S = -h + (2 + 4 + \ldots + 2^{h-1}) + 2^h$$

$$= -h - 1 + (1 + 2 + 4 + \ldots + 2^{h-1}) + 2^h$$

$$= 2^h + 2^h - (h+1) = 2 \cdot 2^h - h - 1$$

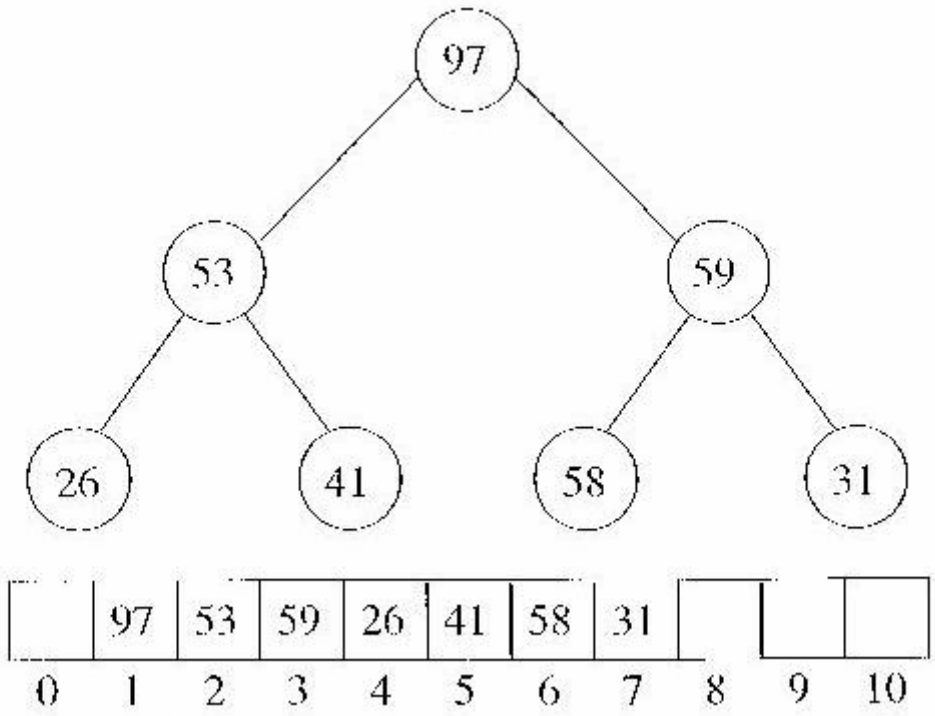
$$= 2 \cdot 2^{\log n} - \log n - 1 \leq 2n$$

Application of Heaps

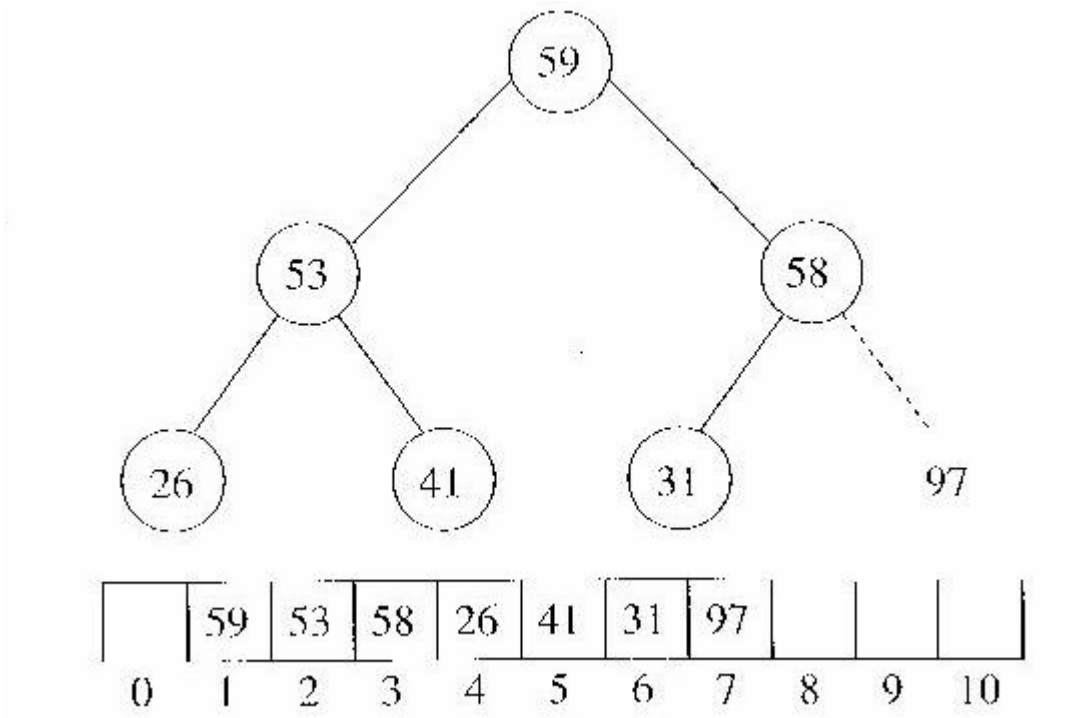


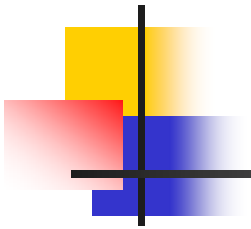
- Heap Sorting: Storing n keys in O(n), and then repetitively applying DeleteMin() operations in O(nlogn). The most basic algorithm.
- Graph algorithms
  - Shortest path finding algorithm
  - Minimum spanning tree finding algorithm
- Scheduling algorithms

Heap Sort (Step 1)



Heap Sort (Step 2)





Heapsort(A)

```
Build-Max-Heap(A)
For I <- length[A] downto 2
    Do exchange A[1] <-> A[I]
        Heapsize[A] <- heapsize[A]-1
        MAX-HEAPIFY(A,1)
```