Priority Queue: Heap

Fatemeh Panahi
Department of Computer Science
University of Toronto
CSC263-Fall 2017
Lecture 2

Announcements

- Assignment 1: is out and the deadline is Sep 29. Do not wait till the last minute.

- Small quiz game next week: You would need to have a smartphone, a tablet or a laptop to participate.

- Tutorial on Friday: Out TA will give you an example of average case running time.
  - Insertion sort and the running time.
  - Few exercises from the textbook, chapter 6.

Average-Case running time

- Average Case: Expected value over the sample space by considering the probability distribution over inputs.

  t_n be a random variable that denotes the number of comparisons executed (Line #3)

$$E(t_n) = \sum_{(A,v)\in S_n} t_n(A, v) \times P[(A, v)]$$

$\Pr[(A, n)] = p,$ v not in A as special

$\Pr[(A, v)] = (1 - p)/n$ other cases equally likely

Today

- Priority Queue
- Heap
  - Insert
  - Delete
  - Max, Extract Max
  - Increase Priority
- Heap Sort
- Build max Heap

Reading Assignments

Chapter 6 

ADT we already know

First in first serve!

- Queue:

First In First Out (LIFO) data structure
Objects: A set of elements.
Operations: Enqueue (x, Q), Dequeue(Q)

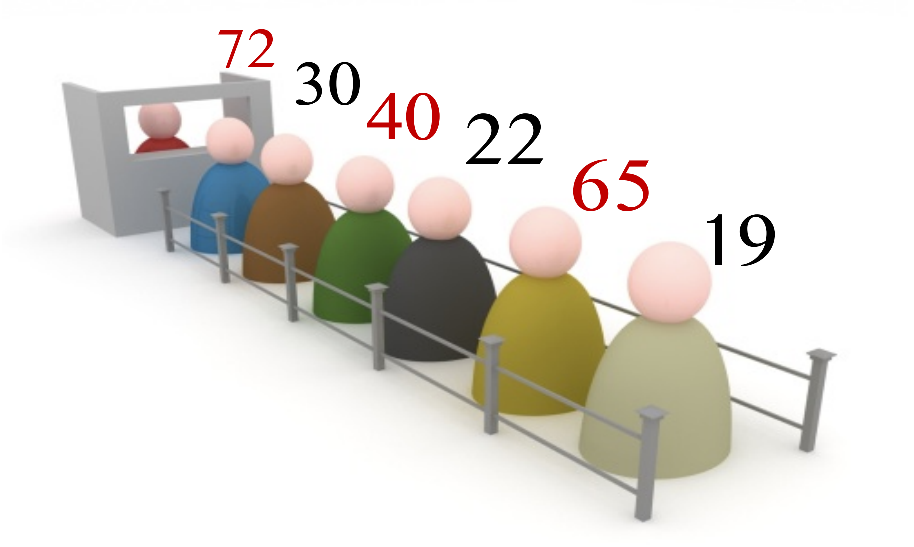
- Stack:

Last In First Out (LIFO) data structure
Objects: A set of elements.
Operations: Push(x, Q), Pop(Q)

Last in first read!

Priority Queue

Oldest person first!



ADT: Priority Queue:

- Abstract data type: like a regular queue or stack data structure,
  but each element has also a "**priority**" associated with it.
  An element with high **priority** is served first.

- Objects:
  - a collection of elements with priorities,
    i.e., each element x has x.priority.
- operations
  - Insert(Q, x)
  - ExtractMax(Q)
  - Max(Q)
  - IncreasePriority(Q, x, k)

Priority Queue:

- Applications:
  - **Job scheduling:** in an operating system

    Processes have different priorities (Normal, high…)
  - **Operating System:** It is also use in Operating System for
    - load balancing on server
    - interrupt handling.
  - etc.

Priority queue:

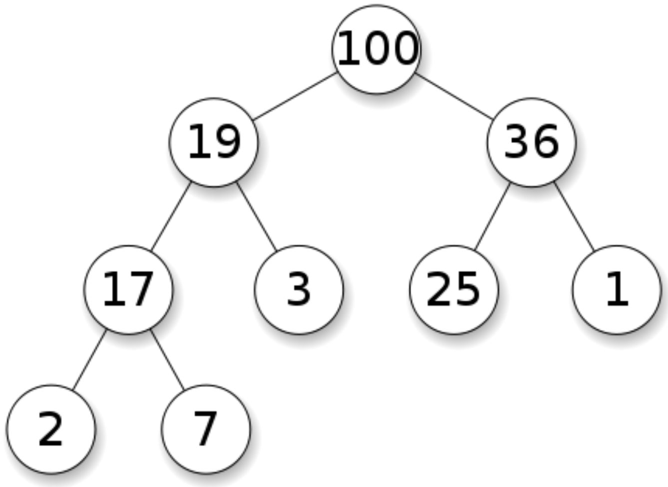
Data structures:

- **Unsorted list?**
  - $\Theta(n)$ for EXTRACT-MAX.
- **Sorted list in an array?**
  - $\Theta(n)$ for INSERT.

A new data structure: Heap

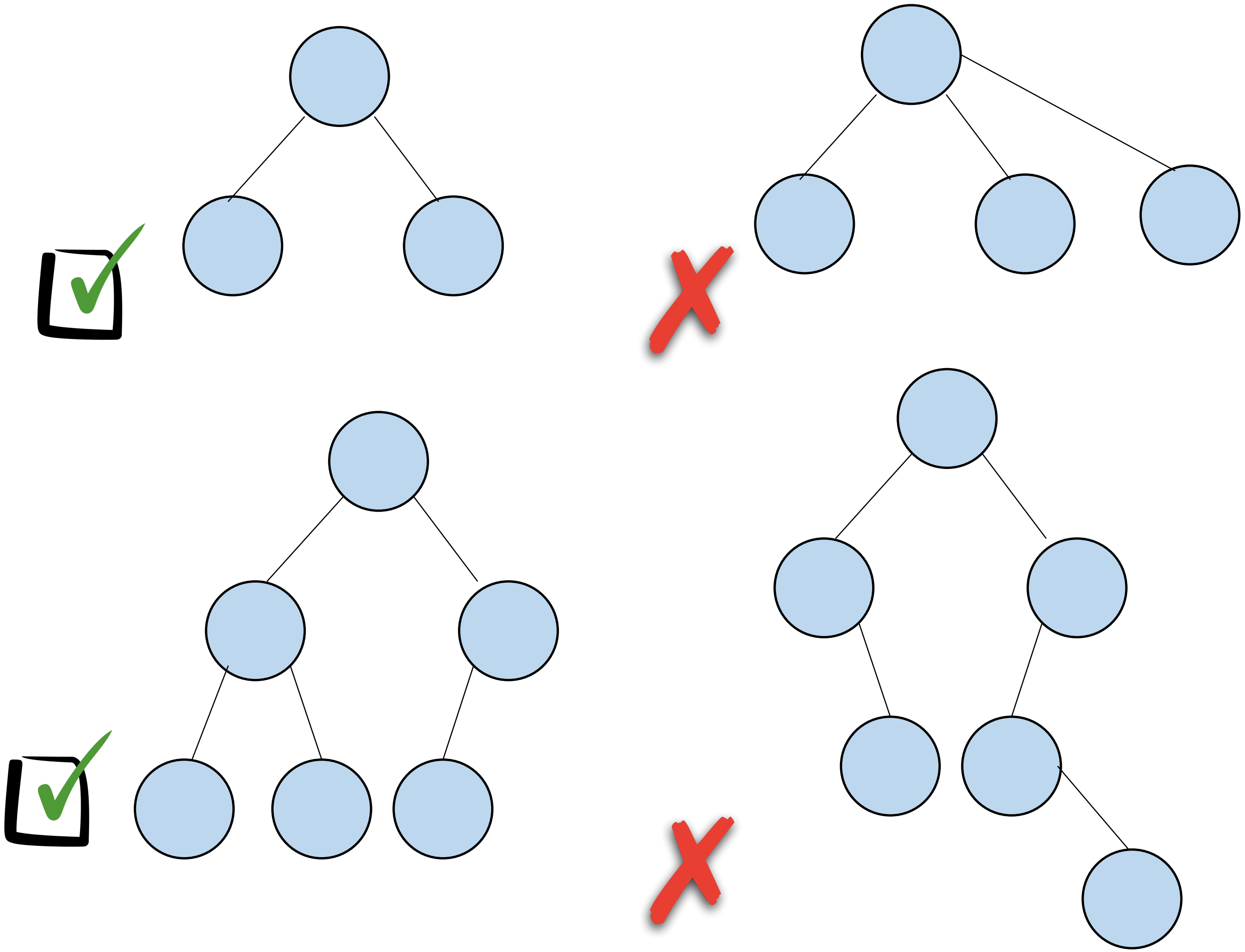
Data Structure: Heap

- Always keep the thing we are most interested in the top in order to a fast access.
- Like a binary search tree, but less structured.
- No relationship between keys at the same level.



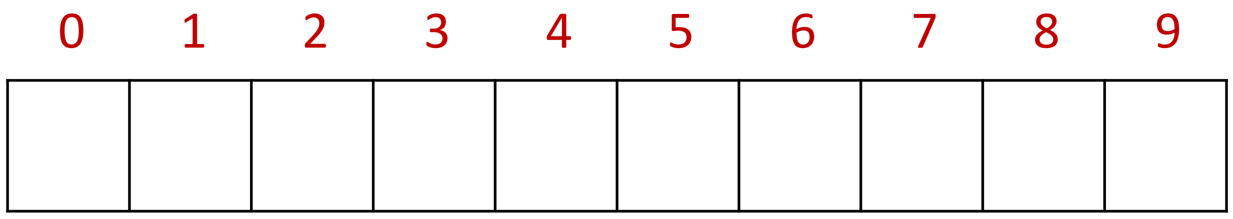
Tree

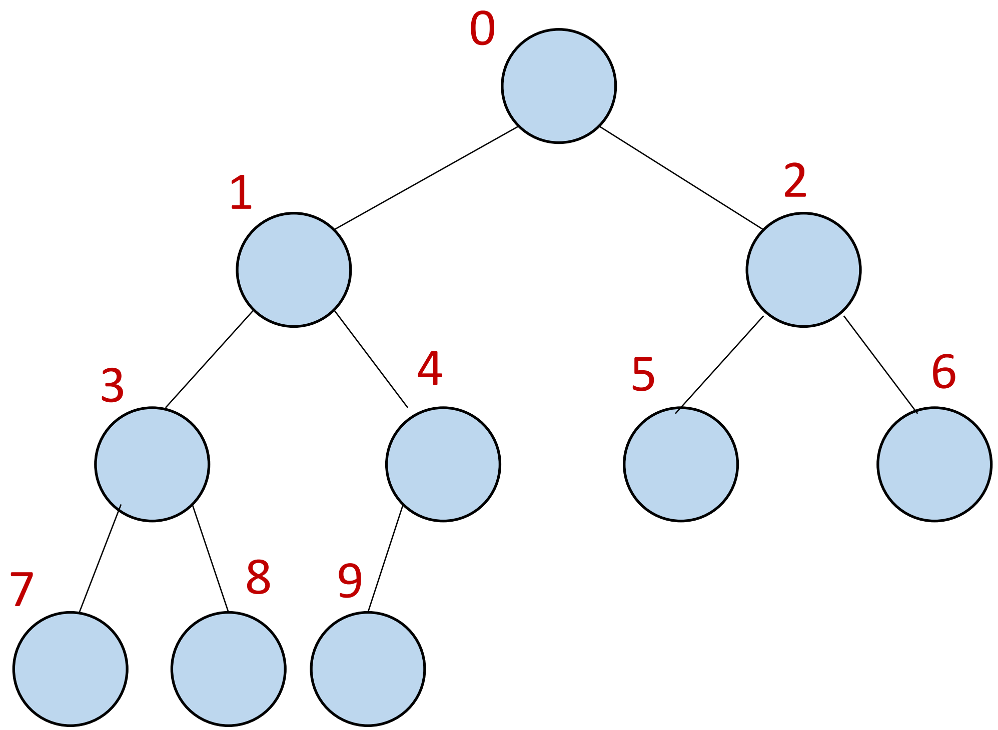
- Binary tree
- Complete: every level, except possibly the last, is completely filled, and all nodes are as far left as possible.
- Full: every node other than the leaves has two children.



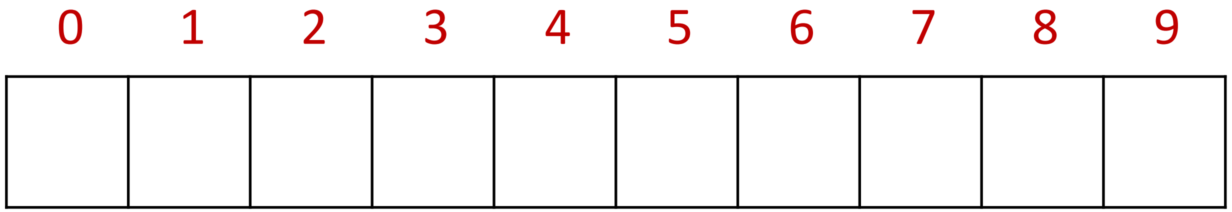
Heap

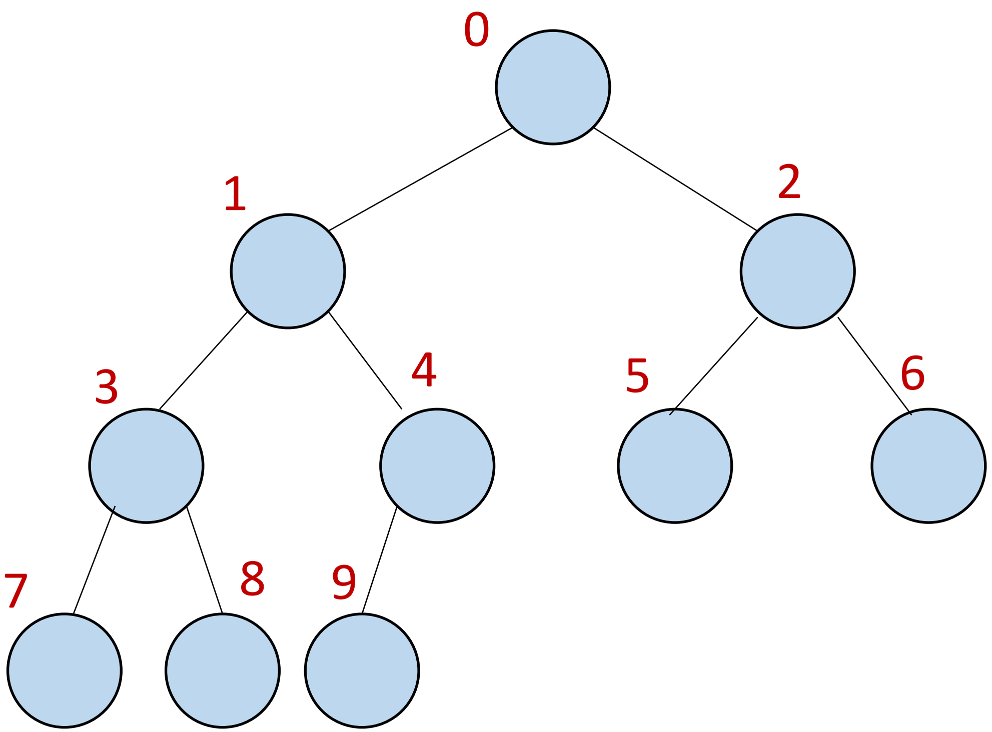
- The (binary) heap data structure is an array object that we can view as a nearly complete binary tree.
- Heap elements stored in array together with an integer "heapsize".





Heap





Conventions:

?

root at index 0

$left[i] = 2i + 1$ for $i < n/2$

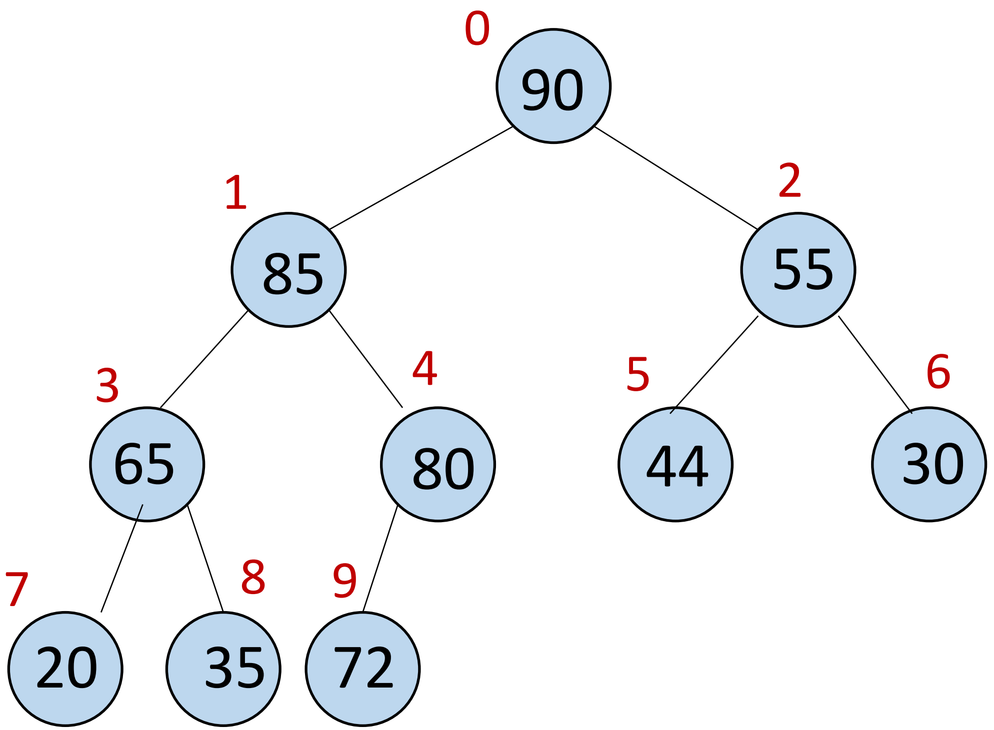
$right[i] = 2i + 2$ for $i < n/2$

$parent[i] = \lfloor i/2 \rfloor - 1$ for $i > 0$

Heap property

- Max Heap: every node has priority $\geq$ to priorities of its immediate children.
- Min Heap: every node has priority $\leq$ to priorities of its immediate children.

0	1	2	3	4	5	6	7	8	9
90	85	55	65	80	44	30	20	35	72

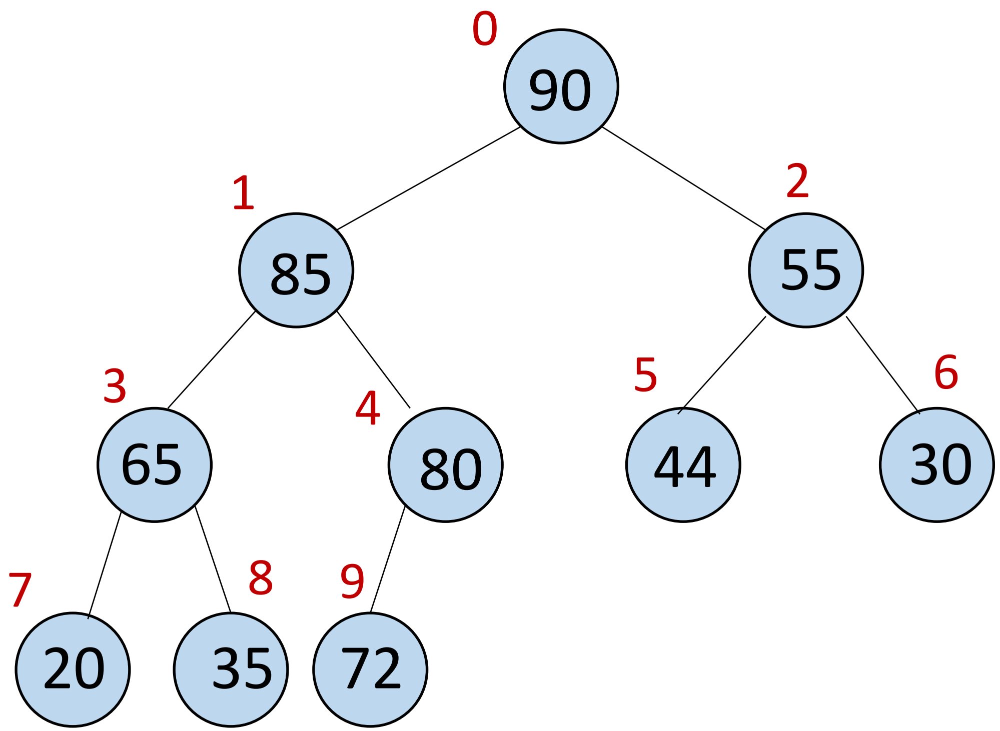


Remark: every subtree of a heap is also a heap.

Operations - Insert

INSERT:

Example: Insert 87

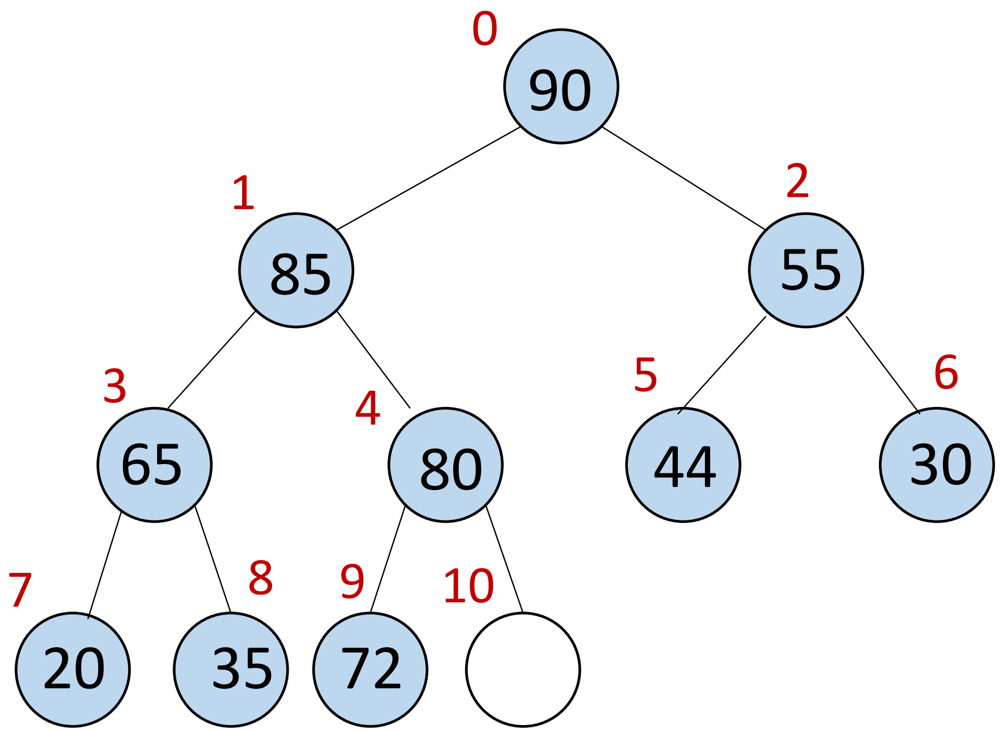


Operations - Insert

INSERT:

- Increment 'heapsize',

Example: Insert 87

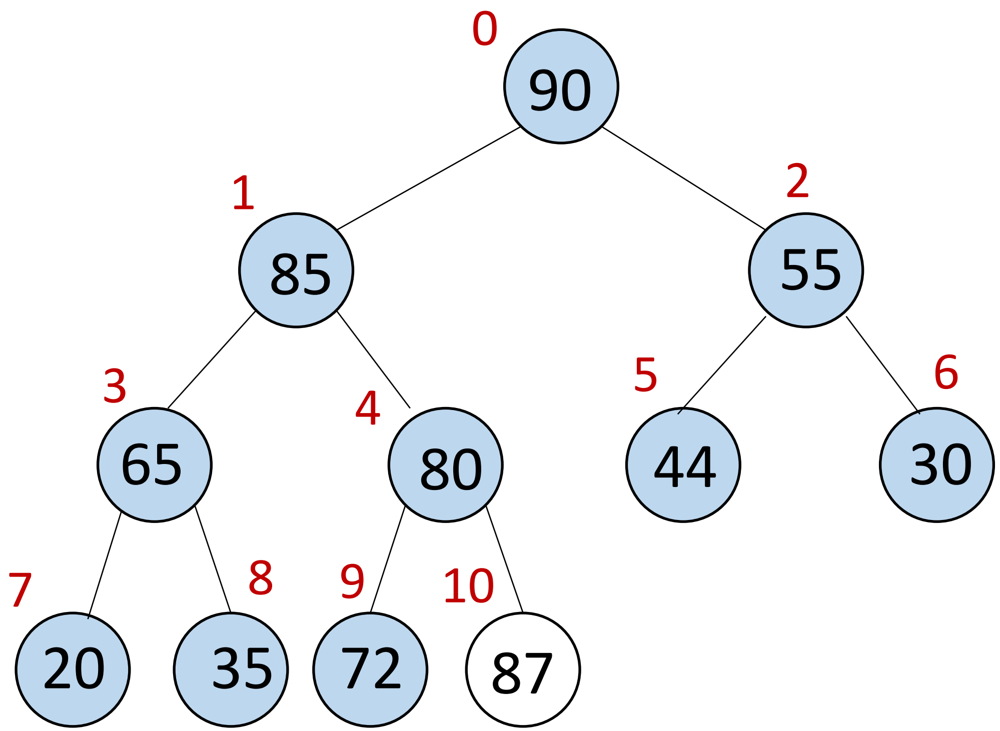


Operations - Insert

INSERT:

- Increment 'heapsize',
- add element at new index 'heapsize'.
- Result might violate heap property.

Example: Insert 87



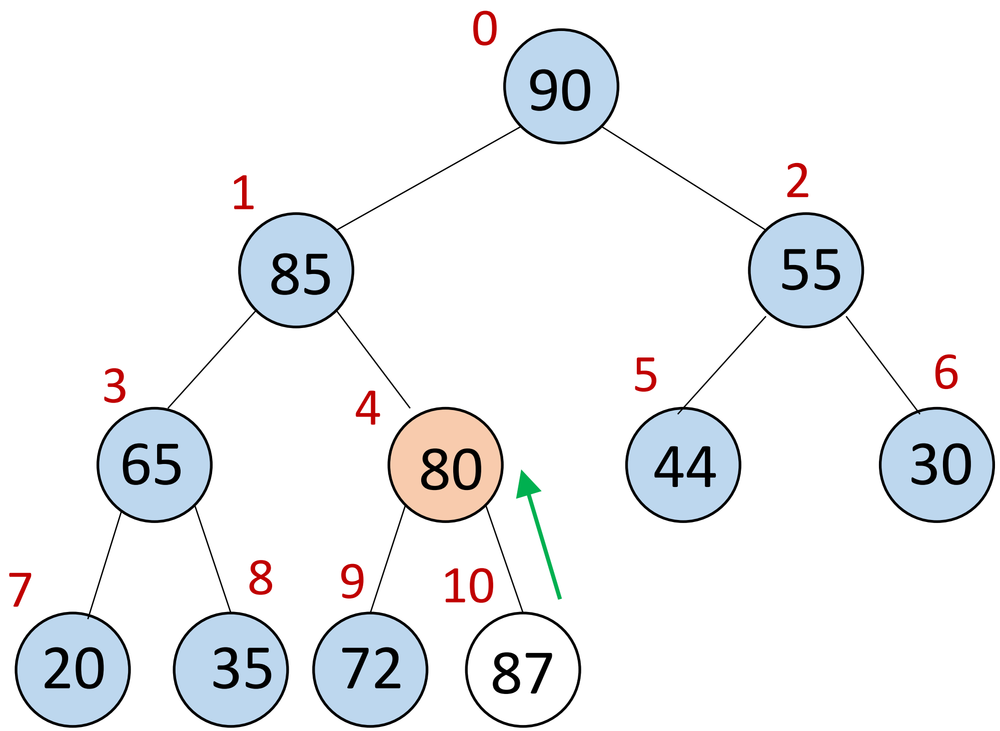
Operations - Insert

INSERT:

- Increment 'heapsize',
- add element at new index 'heapsize'.
- Result might violate heap property.

  "bubble-up" element (exchange it with its parent until priority no greater than priority of parent.)

Example: Insert 87



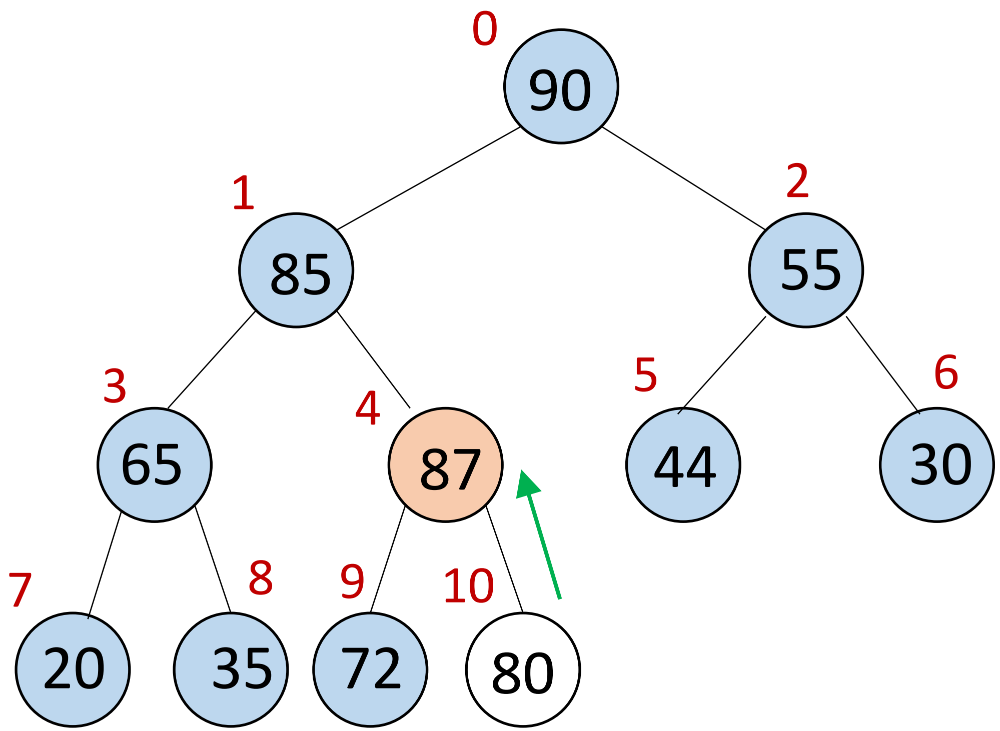
Operations - Insert

INSERT:

- Increment 'heapsize',
- add element at new index 'heapsize'.
- Result might violate heap property.

  "bubble-up" element (exchange it with its parent until priority no greater than priority of parent.)

Example: Insert 87



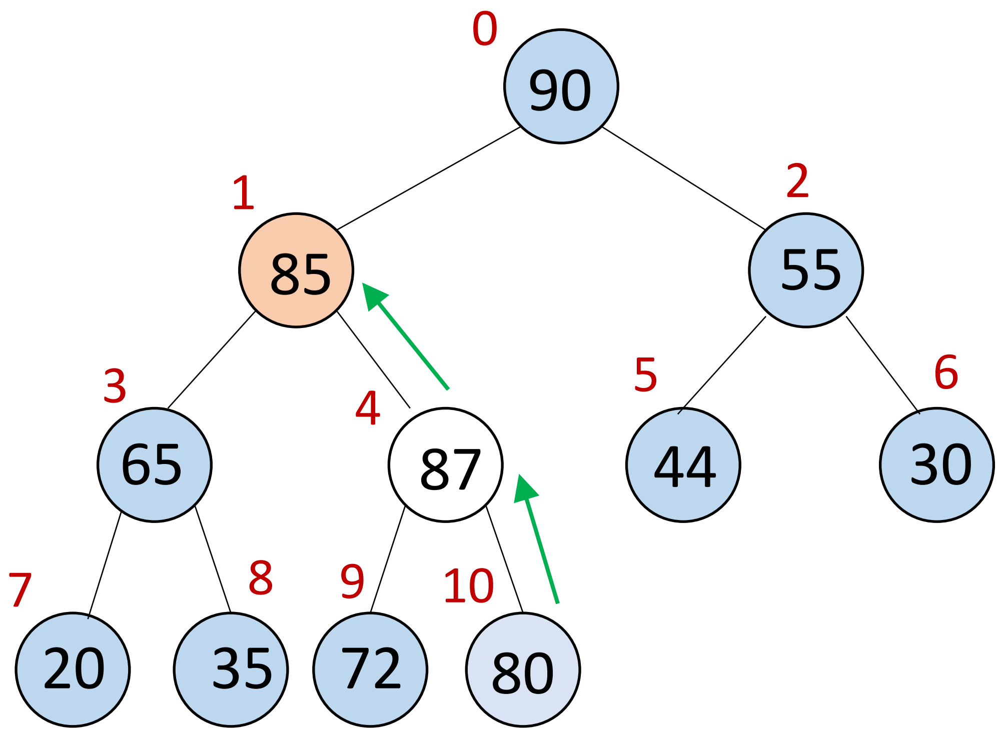
Operations - Insert

INSERT:

- Increment 'heapsize',
- add element at new index 'heapsize'.
- Result might violate heap property.

  "bubble-up" element (exchange it with its parent until priority no greater than priority of parent.)

Example: Insert 87



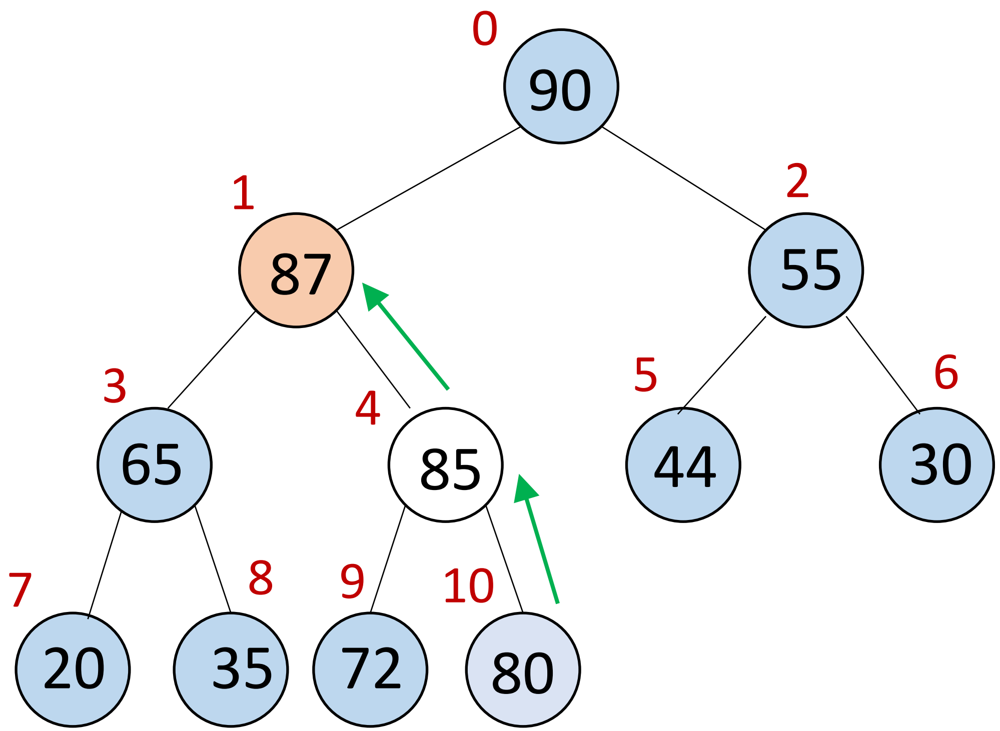
Operations - Insert

INSERT:

- Increment 'heapsize',
- add element at new index 'heapsize'.
- Result might violate heap property.

  “bubble-up” element (exchange it with its parent until priority no greater than priority of parent.)

Example: Insert 87



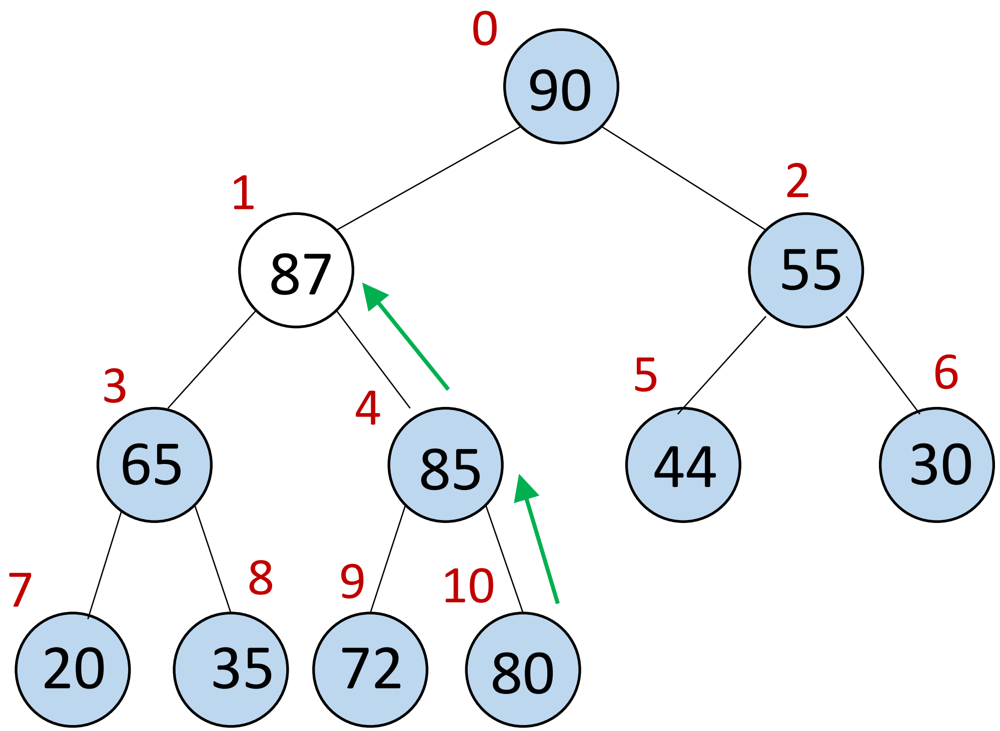
Operations - Insert

INSERT:

- Increment 'heapsize',
- add element at new index 'heapsize'.
- Result might violate heap property.

  "bubble-up" element (exchange it with its parent until priority no greater than priority of parent.)

Example: Insert 87



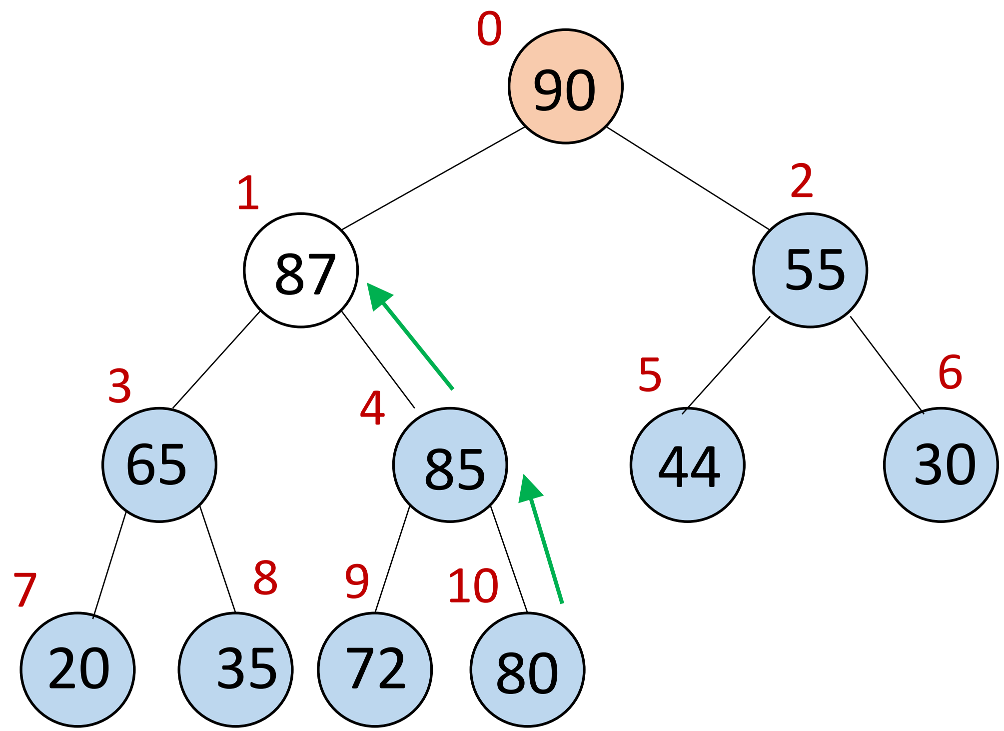
Operations - Insert

INSERT:

- Increment 'heapsize',
- add element at new index 'heapsize'.
- Result might violate heap property.

  “bubble-up” element (exchange it with its parent until priority no greater than priority of parent.)

Example: Insert 87



Operations - Insert

INSERT:

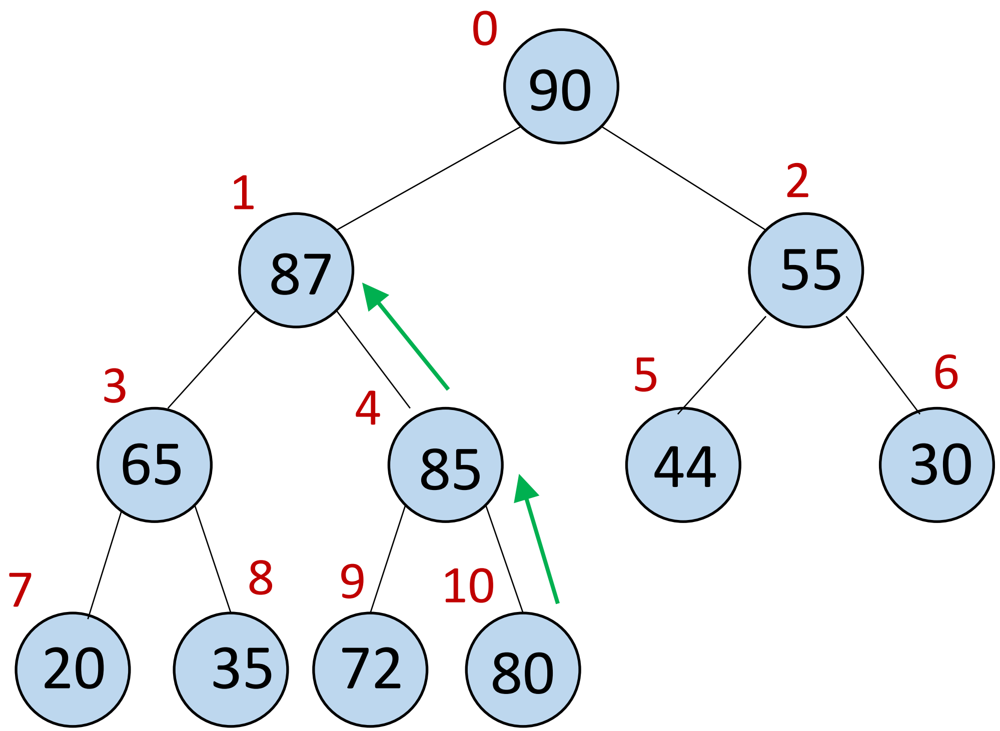
- Increment 'heapsize',
- add element at new index 'heapsize'.
- Result might violate heap property.

  "bubble-up" element (exchange it with its parent until priority no greater than priority of parent.)

Running time?

? $\Theta(height) = \Theta(\log n)$.

Example: Insert 87



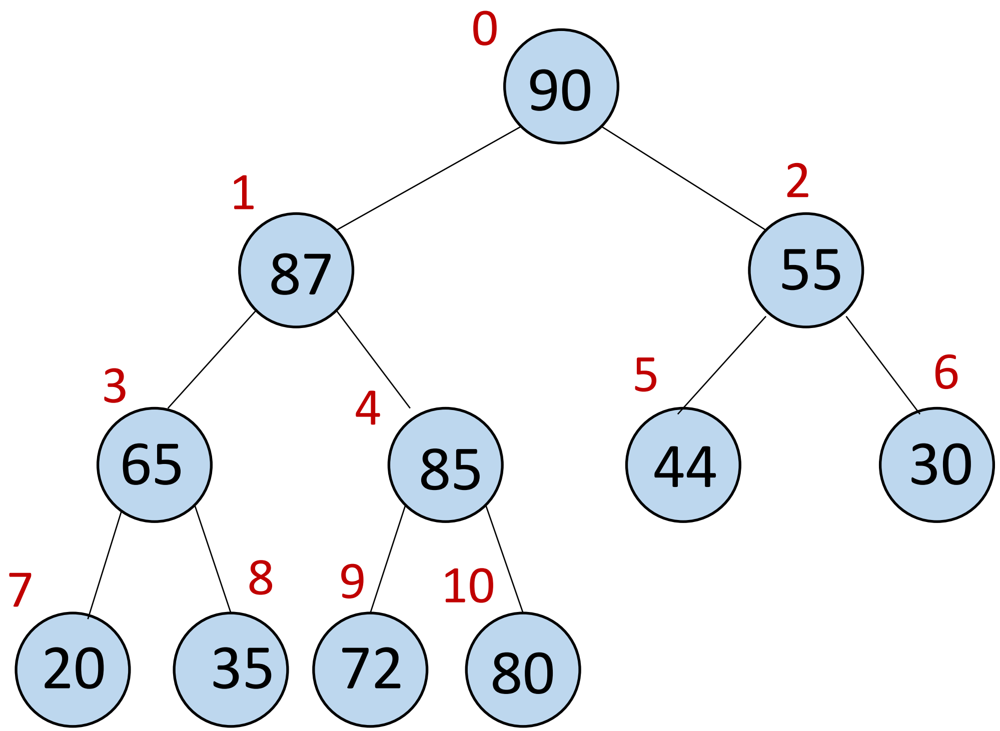
Operations - Maximum

Maximum:

- Return element at index 0

  Running time? $\Theta(1)$

Example:



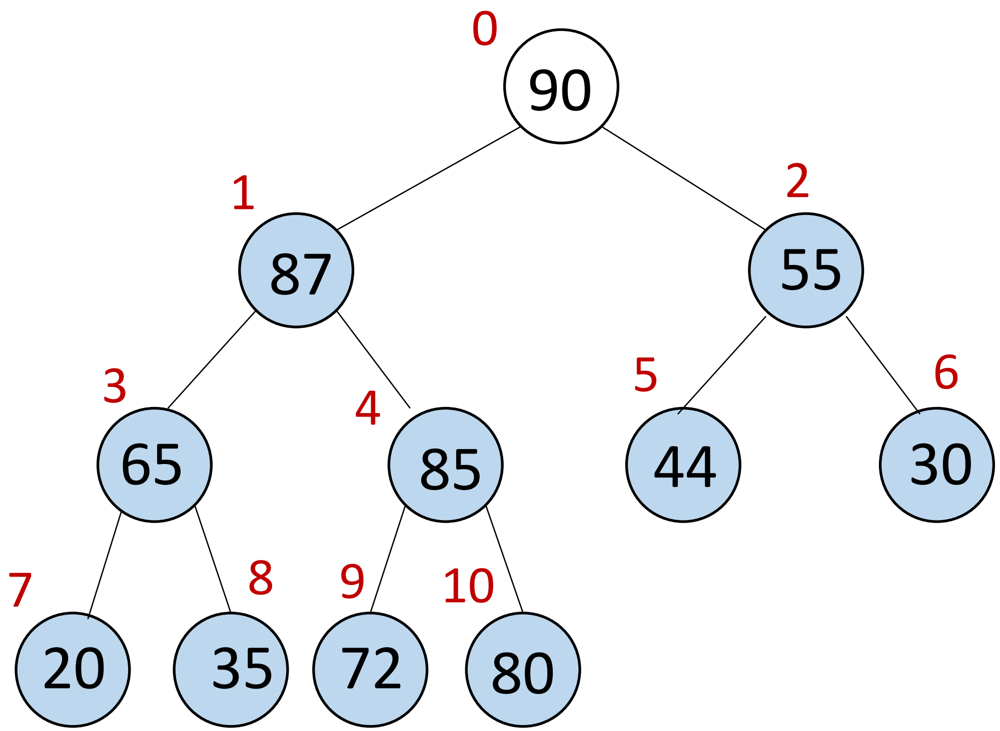
Operations - Maximum

Maximum:

- Return element at index 0

Running time? $\Theta(1)$

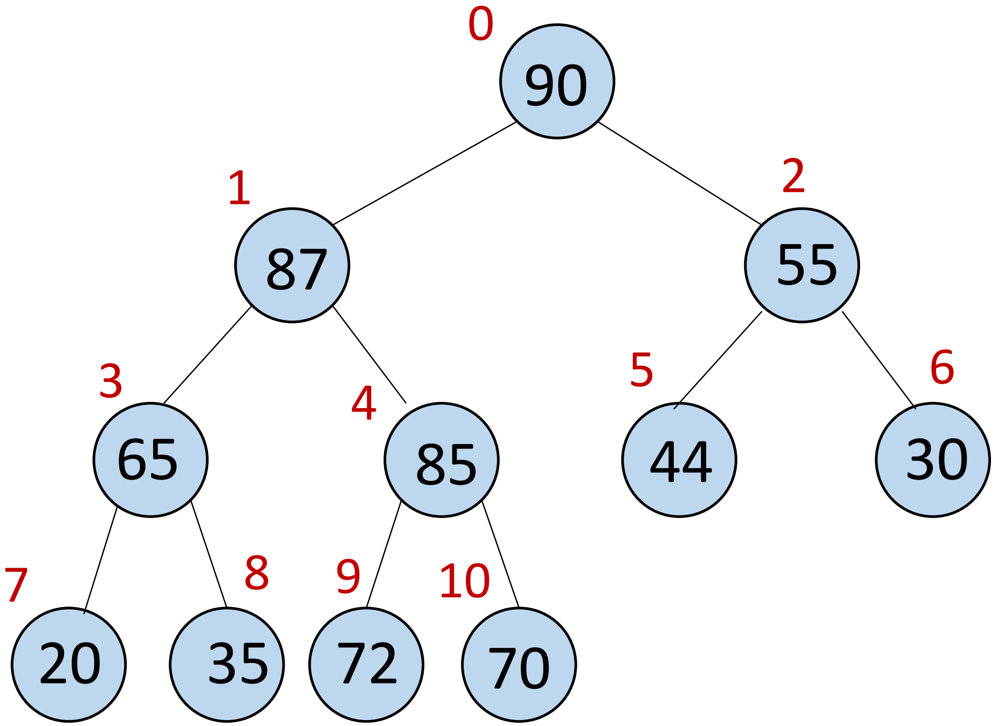
Example: 90



Operations – Extract Max

- Return element at index 0 and remove it from the Max-Heap

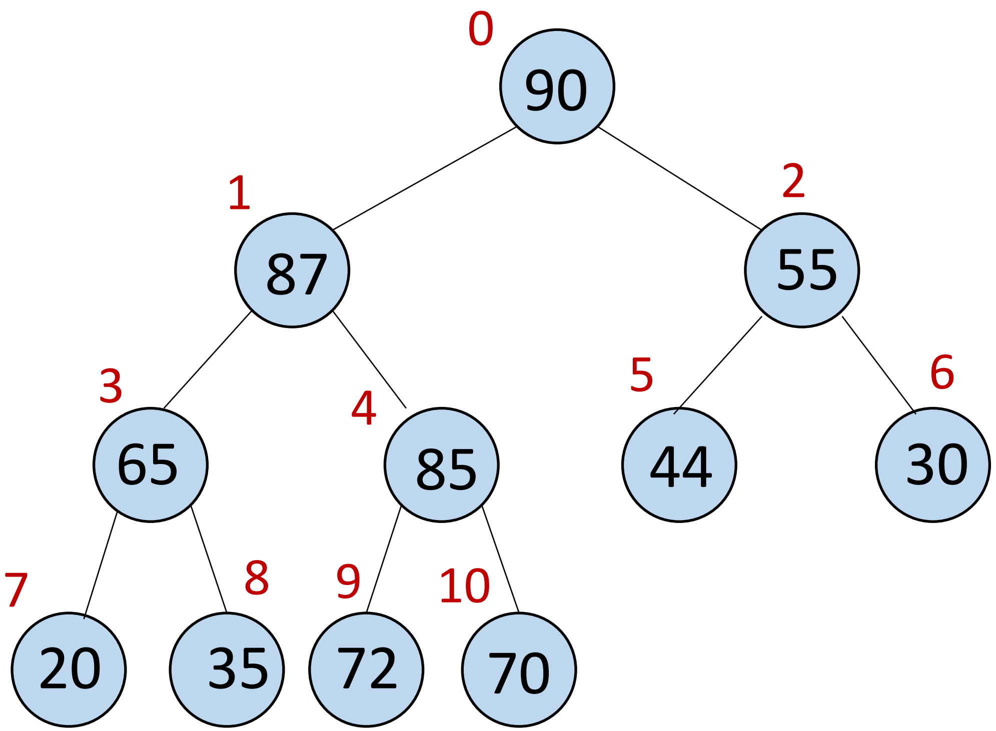
Extract Max:



Operations – Extract Max

- Decrement 'heapsize', remove element at index 0.

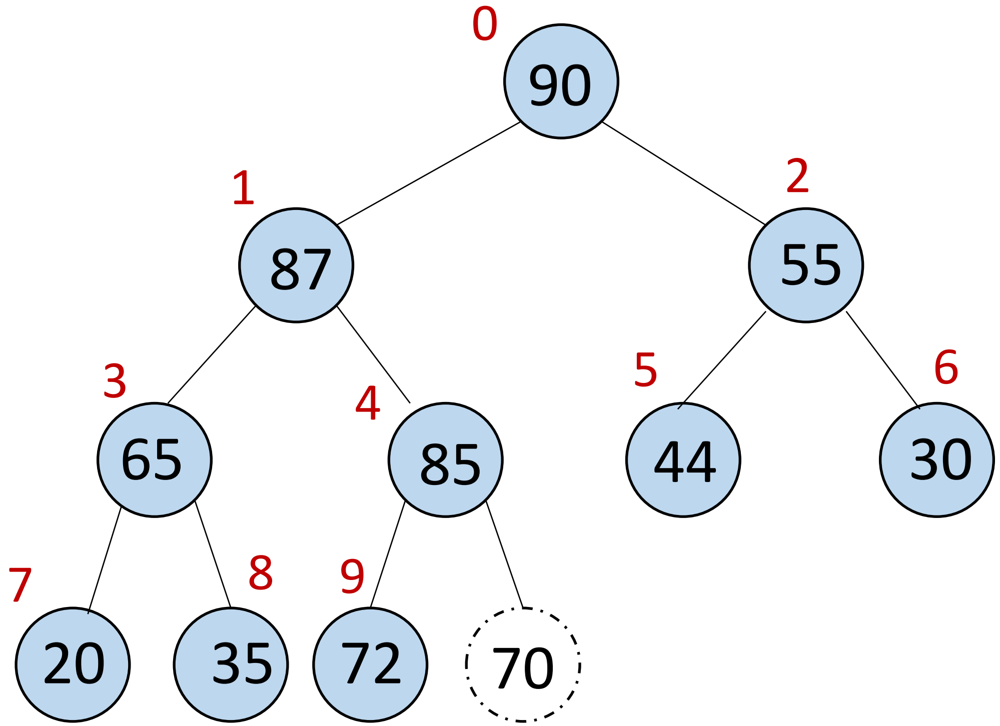
Extract Max:



Operations – Extract Max

- Decrement 'heapsize', remove element at index 0.

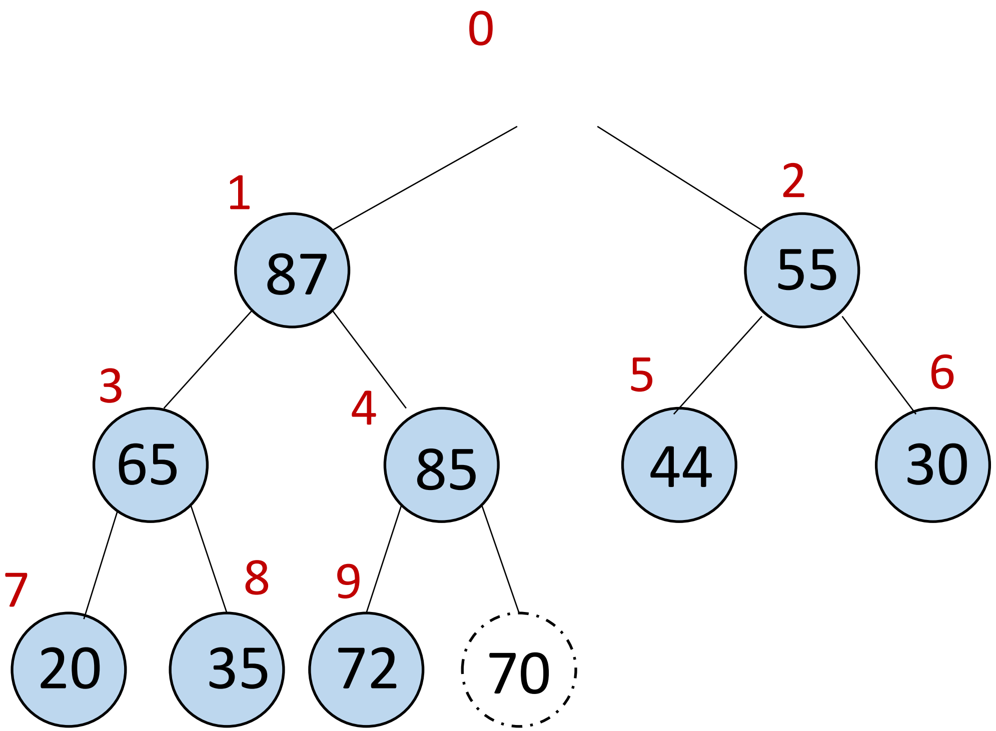
Extract Max:



Operations – Extract Max

- Decrement 'heapsize', remove element at index 0.

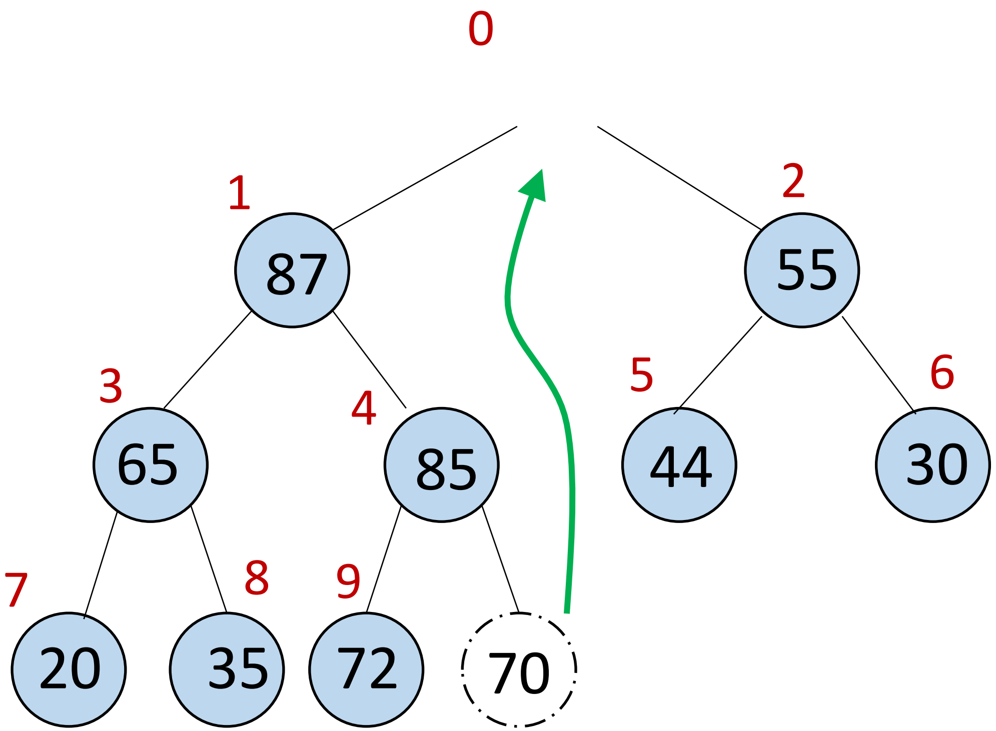
Extract Max:



Operations – Extract Max

- Decrement 'heapsize', remove element at index 0.
- Move element at index 'heapsize+1' into index 0.

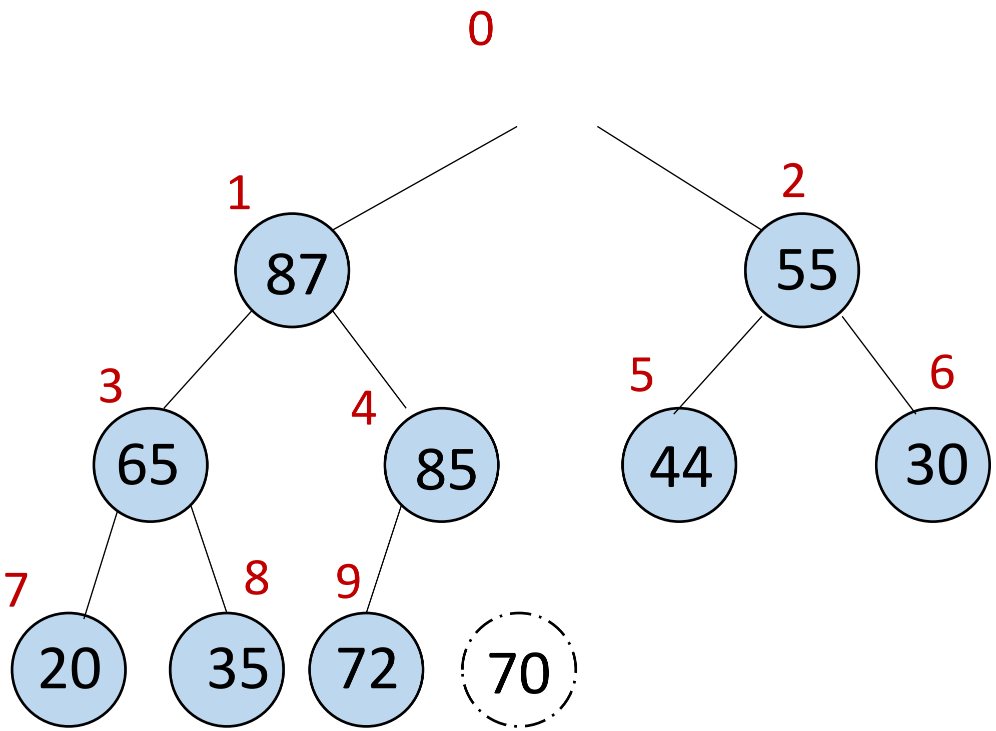
Extract Max:



Operations – Extract Max

- Decrement 'heapsize', remove element at index 0.
- Move element at index 'heapsize+1' into index 0.

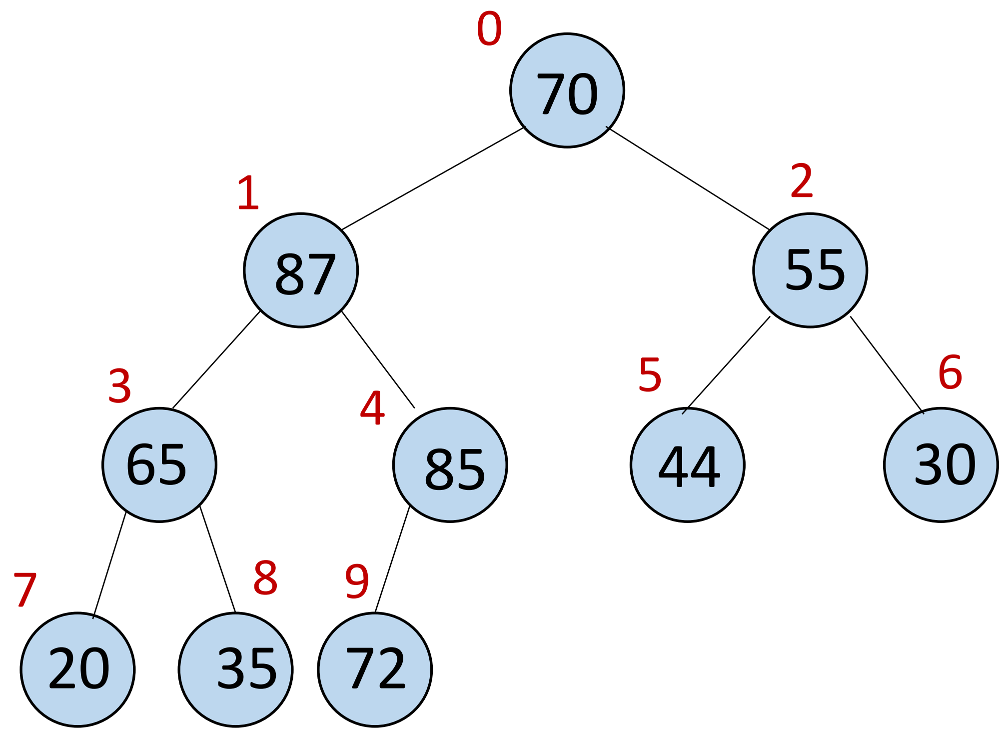
Extract Max:



Operations – Extract Max

- Decrement 'heapsize', remove element at index 0.
- Move element at index 'heapsize+1' into index 0.

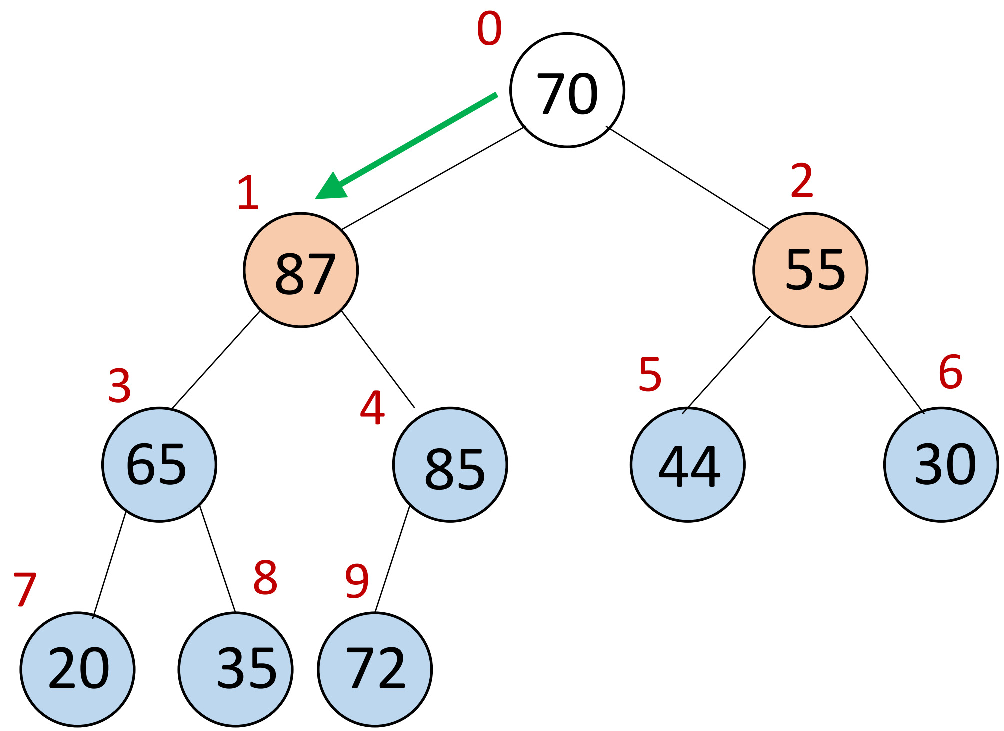
Extract Max:



Operations – Extract Max

- Decrement 'heapsize', remove element at index 0.
- Move element at index 'heapsize+1' into index 0.
- Restore heap order by "bubbling down" (exchange with highest priority child until priority is greater than or equal to both children, or leaf is reached): called MAX-HEAPIFY in textbook.

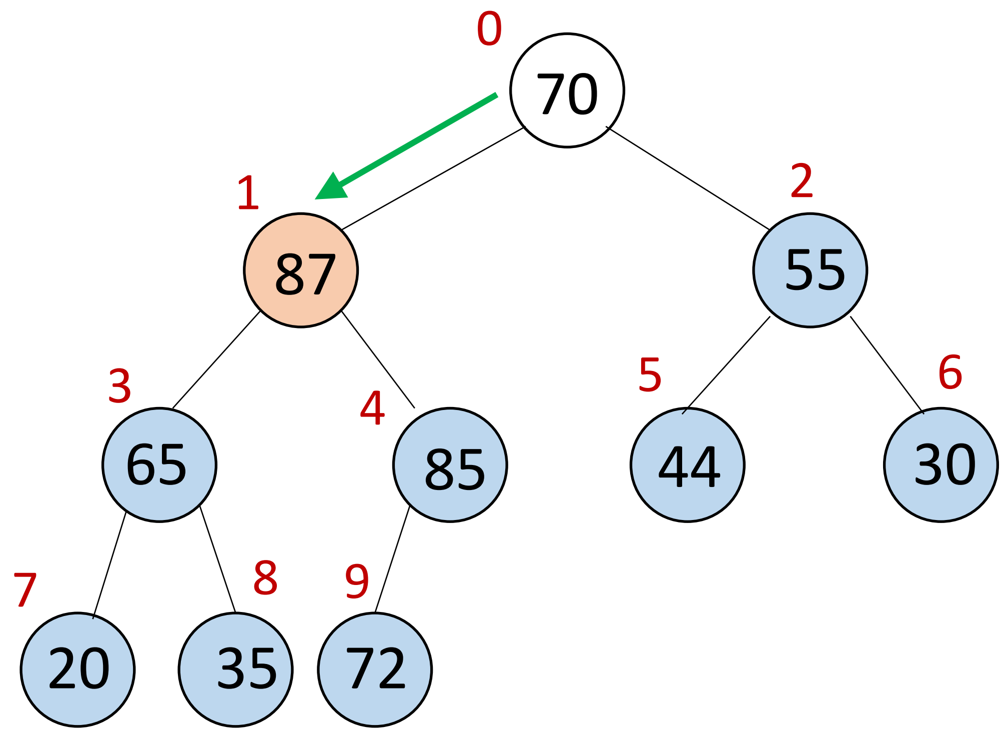
Extract Max:



Operations – Extract Max

- Decrement 'heapsize', remove element at index 0.
- Move element at index 'heapsize+1' into index 0.
- Restore heap order by "bubbling down" (exchange with highest priority child until priority is greater than or equal to both children, or leaf is reached): called MAX-HEAPIFY in textbook.

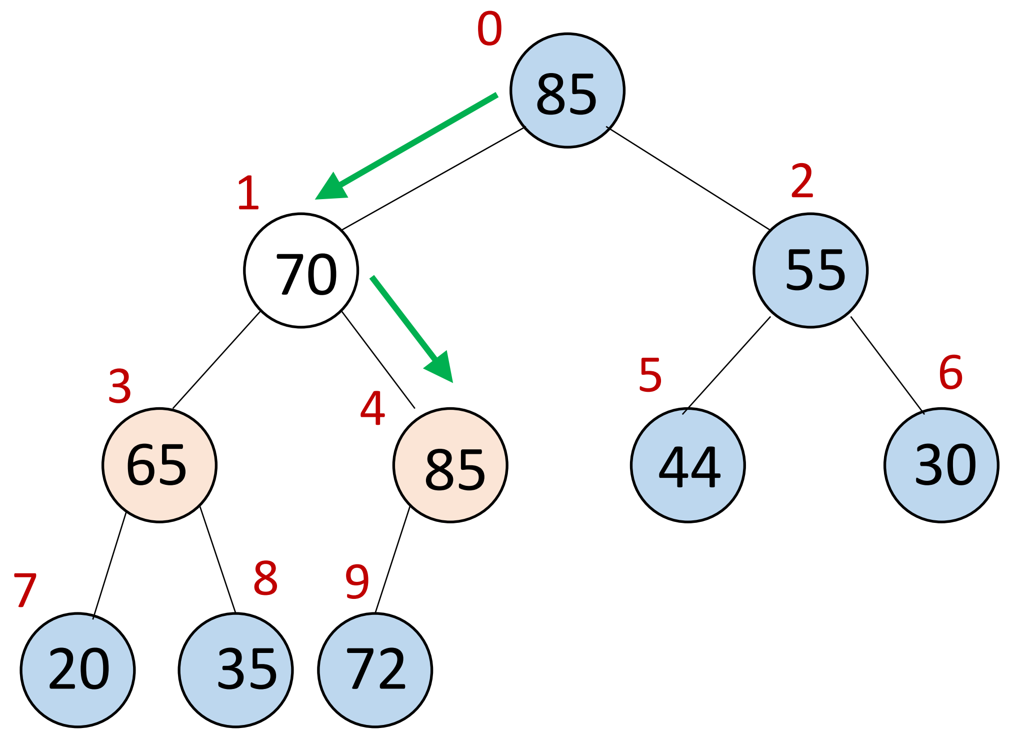
Extract Max:



Operations – Extract Max

- Decrement 'heapsize', remove element at index 0.
- Move element at index 'heapsize+1' into index 0.
- Restore heap order by "bubbling down" (exchange with highest priority child until priority is greater than or equal to both children, or leaf is reached): called MAX-HEAPIFY in textbook.

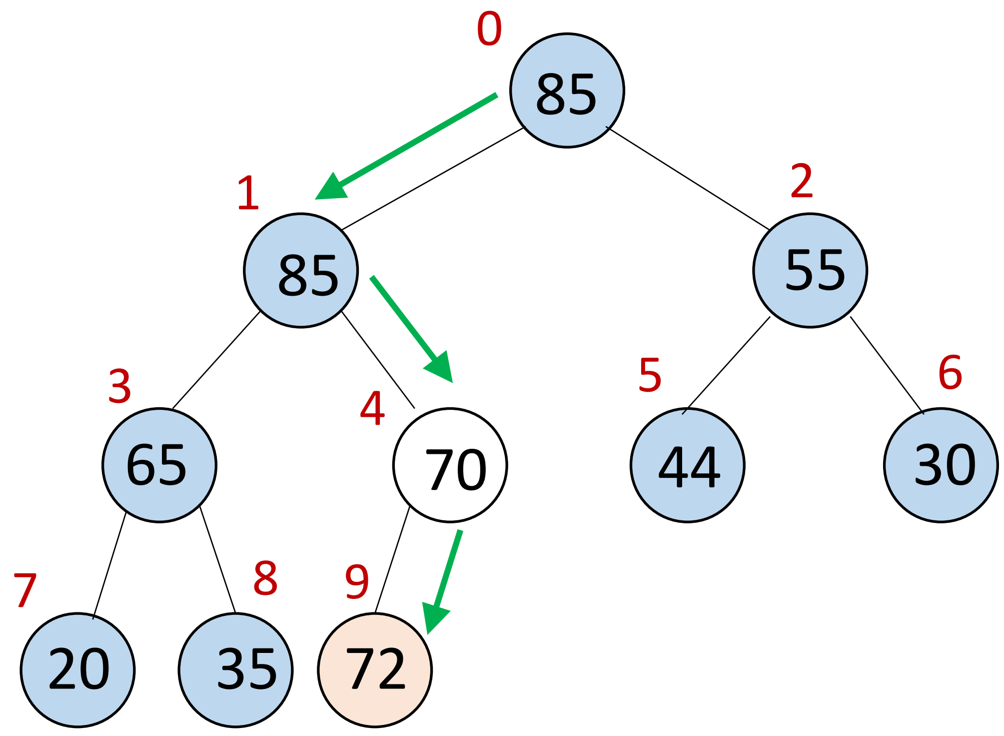
Extract Max:



Operations – Extract Max

- Decrement 'heapsize', remove element at index 0.
- Move element at index 'heapsize+1' into index 0.
- Restore heap order by "bubbling down" (exchange with highest priority child until priority is greater than or equal to both children, or leaf is reached): called MAX-HEAPIFY in textbook.

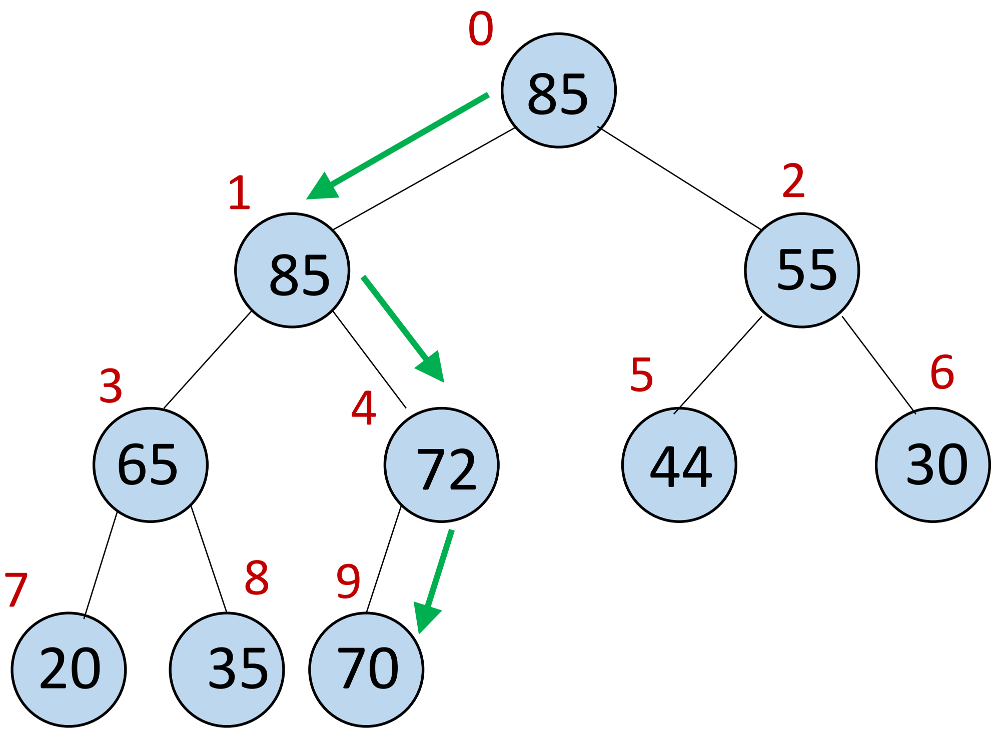
Extract Max:



Operations – Extract Max

- Decrement 'heapsize', remove element at index 0.
- Move element at index 'heapsize+1' into index 0.
- Restore heap order by "bubbling down" (exchange with highest priority child until priority is greater than or equal to both children, or leaf is reached): called MAX-HEAPIFY in textbook.

Extract Max:



Heap Sort

Sorts an array, in $O(n\ log n)$ time

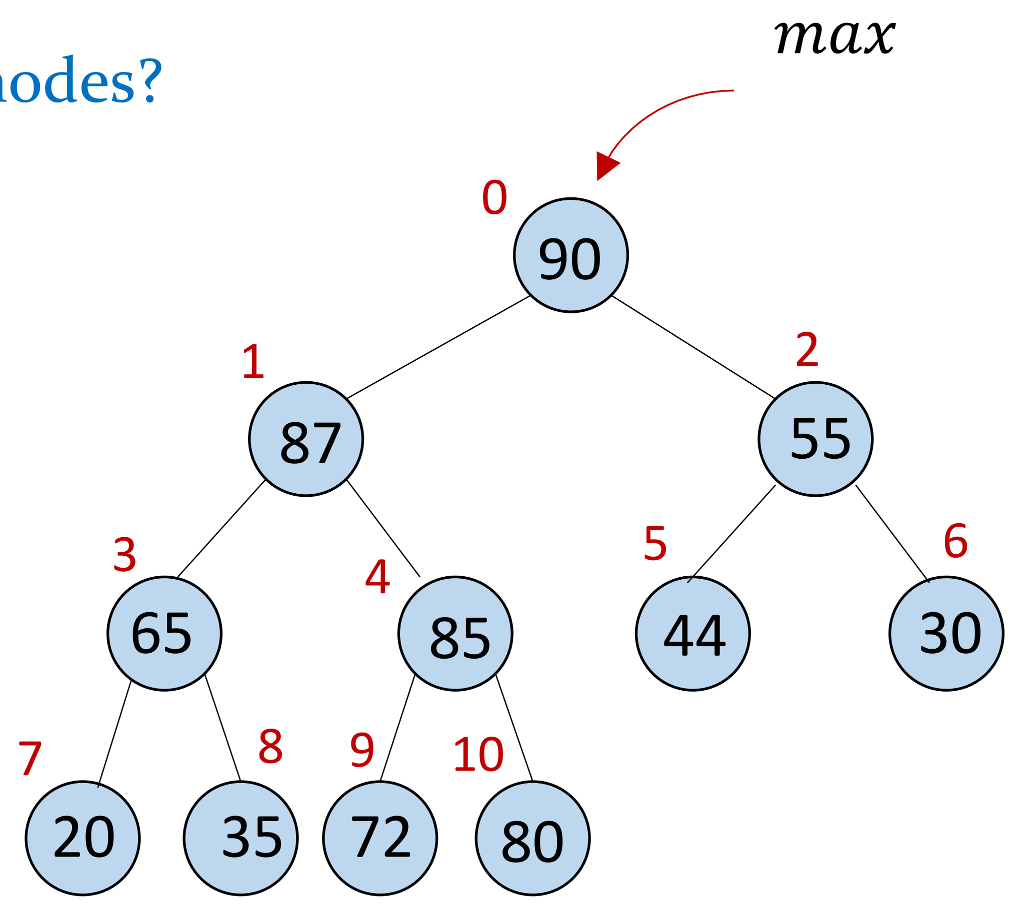
Heap Sort – the idea

How to get a sorted list out of a heap with n nodes?

- Keep extracting max for n times.

Worst case running time?

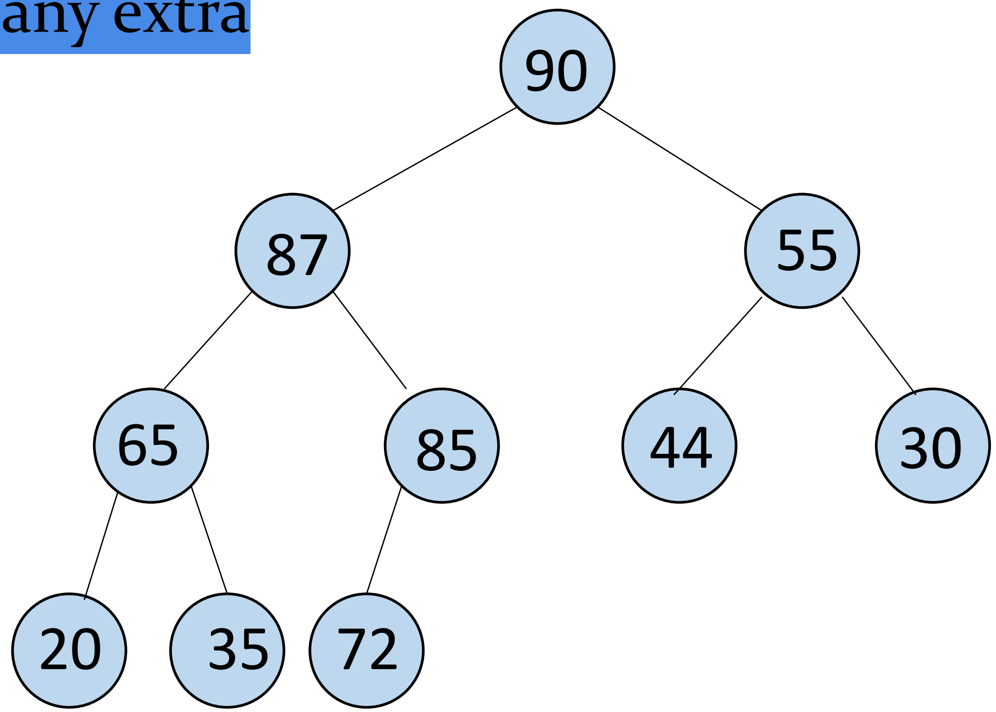
Each Extract-Max is $O(\log n)$, we perform it n times, so overall it's $\Theta(n \, log \, n)$



Heap Sort – Let’s be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this “in-place” without using any extra array space.

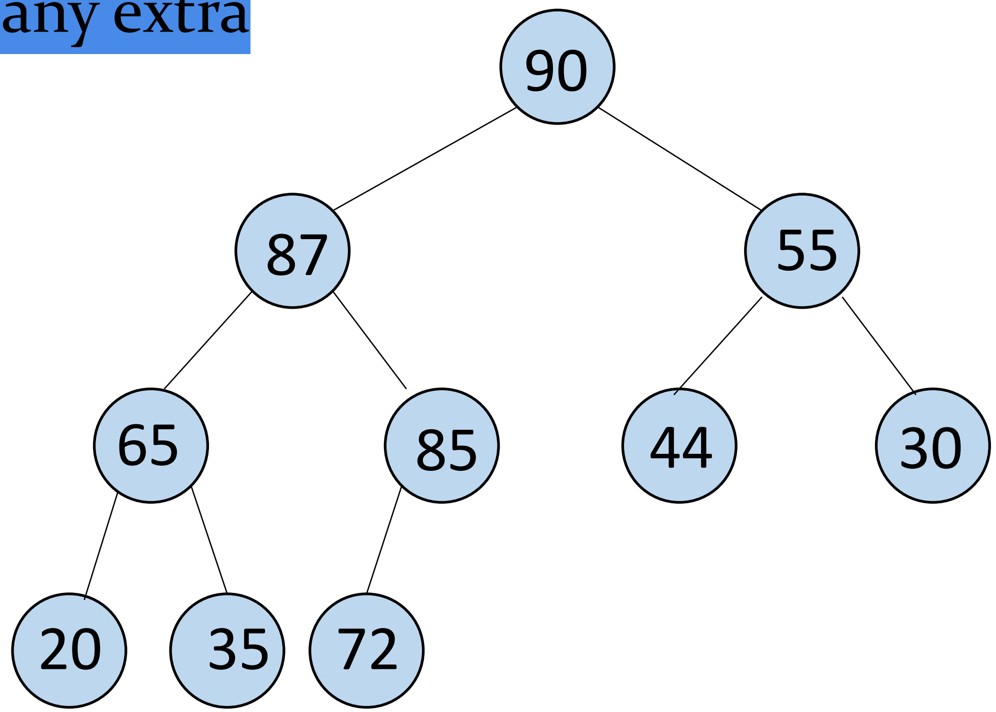
90	~~85~~ 87	55	65	~~80~~ 85	44	30	20	35	72



Heap Sort – Let’s be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this “in-place” without using any extra array space.

90	85	55	65	80	44	30	20	35	72

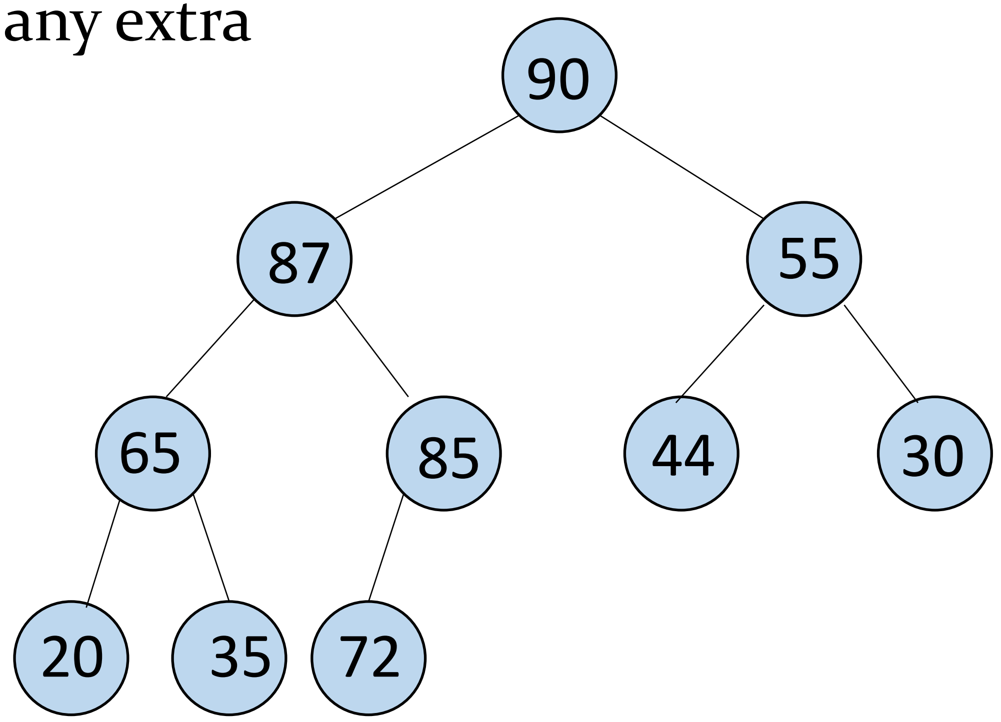


Heap Sort – Let's be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this "in-place" without using any extra array space.

90	85	55	65	80	44	30	20	35	72

72	85	55	65	80	44	30	20	35	90

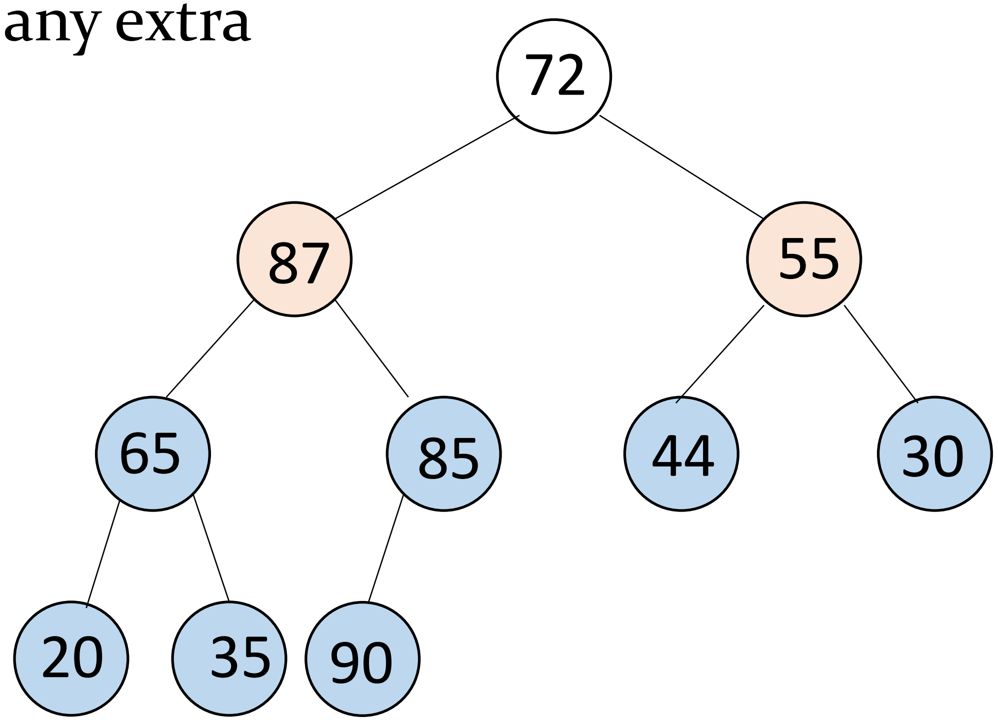


Heap Sort – Let’s be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this “in-place” without using any extra array space.

90	85	55	65	80	44	30	20	35	72

72	85	55	65	80	44	30	20	35	90

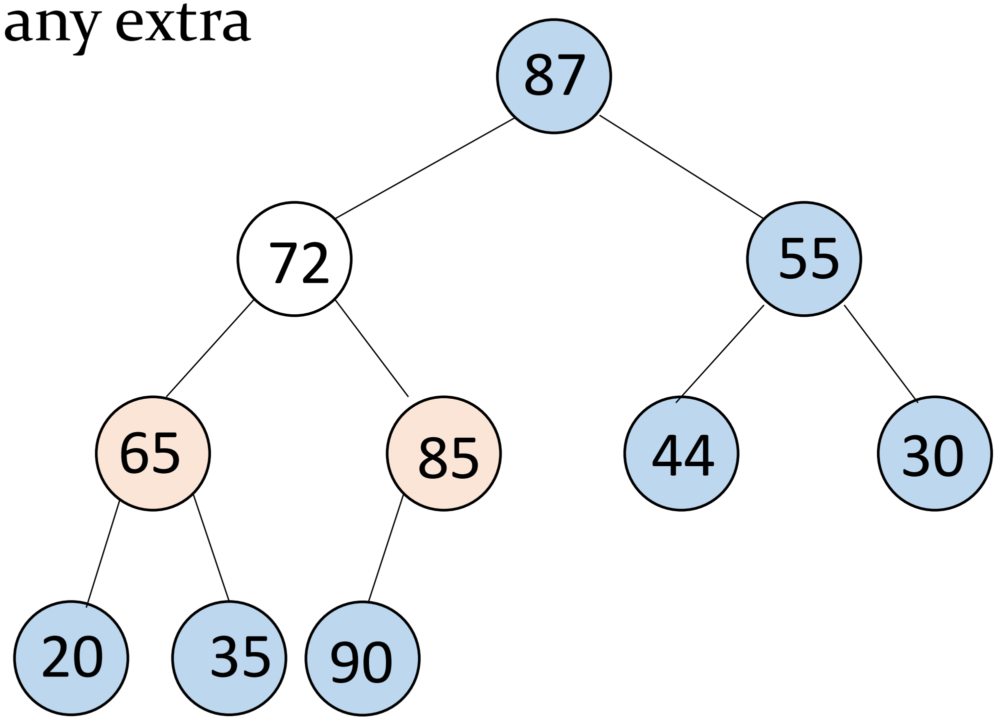


Heap Sort – Let's be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this "in-place" without using any extra array space.

90	85	55	65	80	44	30	20	35	72

72	85	55	65	80	44	30	20	35	90

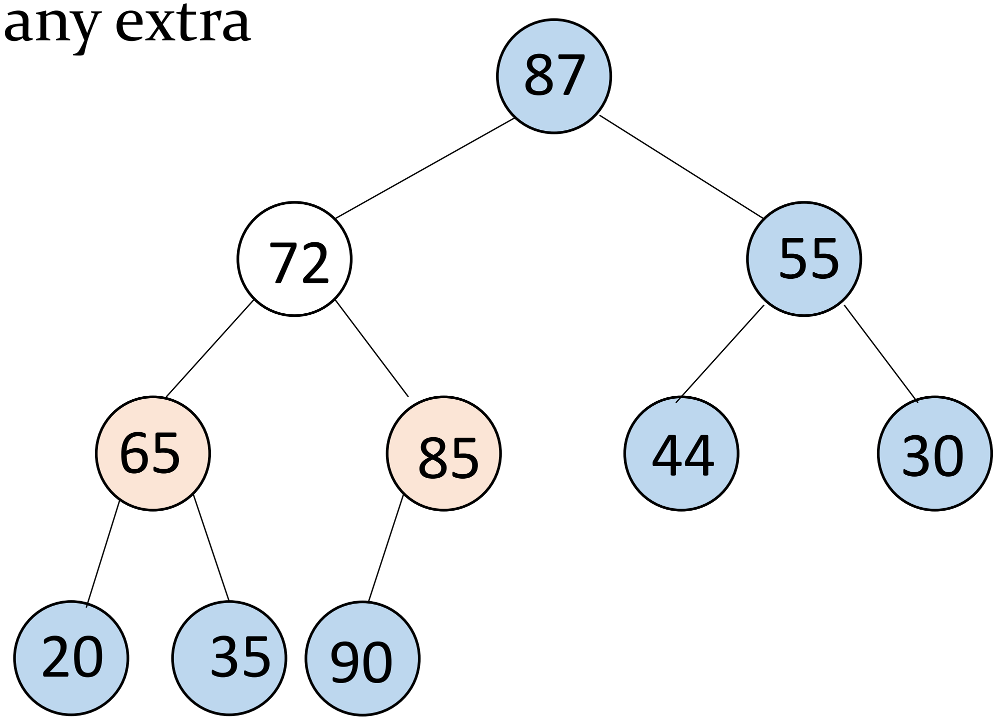


Heap Sort – Let's be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this "in-place" without using any extra array space.

90	85	55	65	80	44	30	20	35	72

72	85	55	65	80	44	30	20	35	90

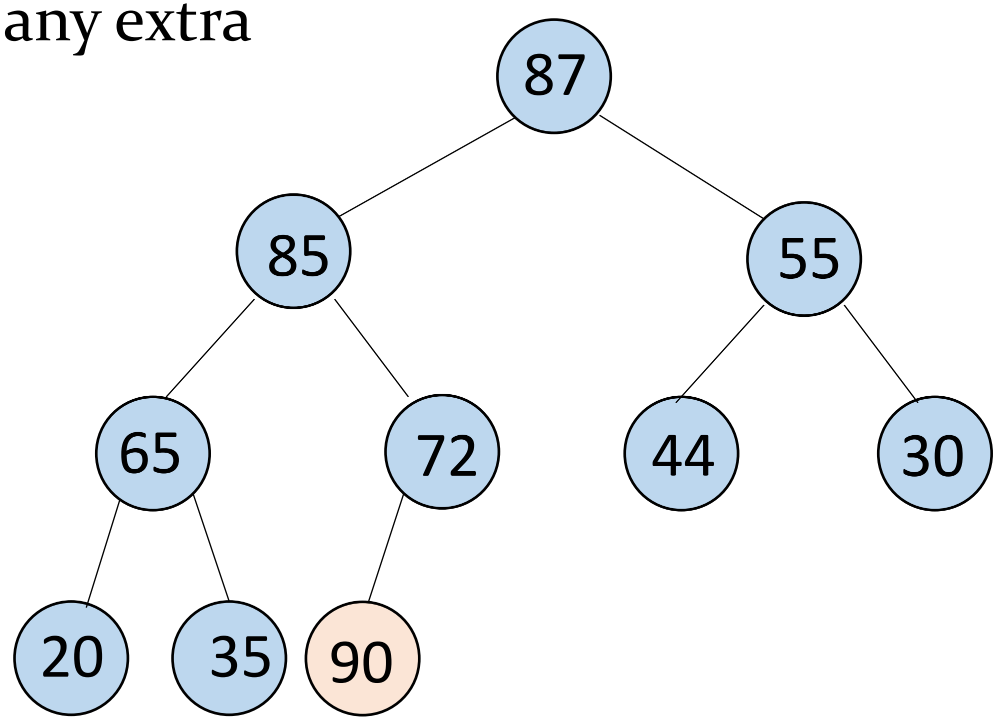


Heap Sort – Let’s be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this “in-place” without using any extra array space.

90	85	55	65	80	44	30	20	35	72

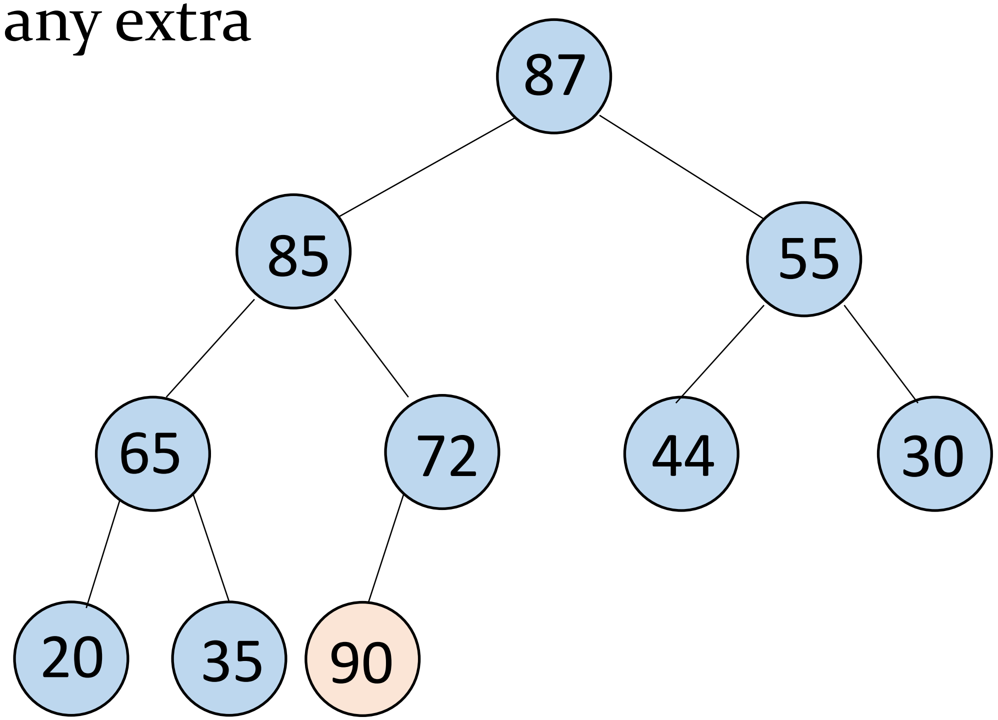
87	85	55	65	72	44	30	20	35	90



Heap Sort – Let's be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this "in-place" without using any extra array space.

90	85	55	65	80	44	30	20	35	72
87	85	55	65	72	44	30	20	35	90
35	85	55	65	72	44	30	20	87	90



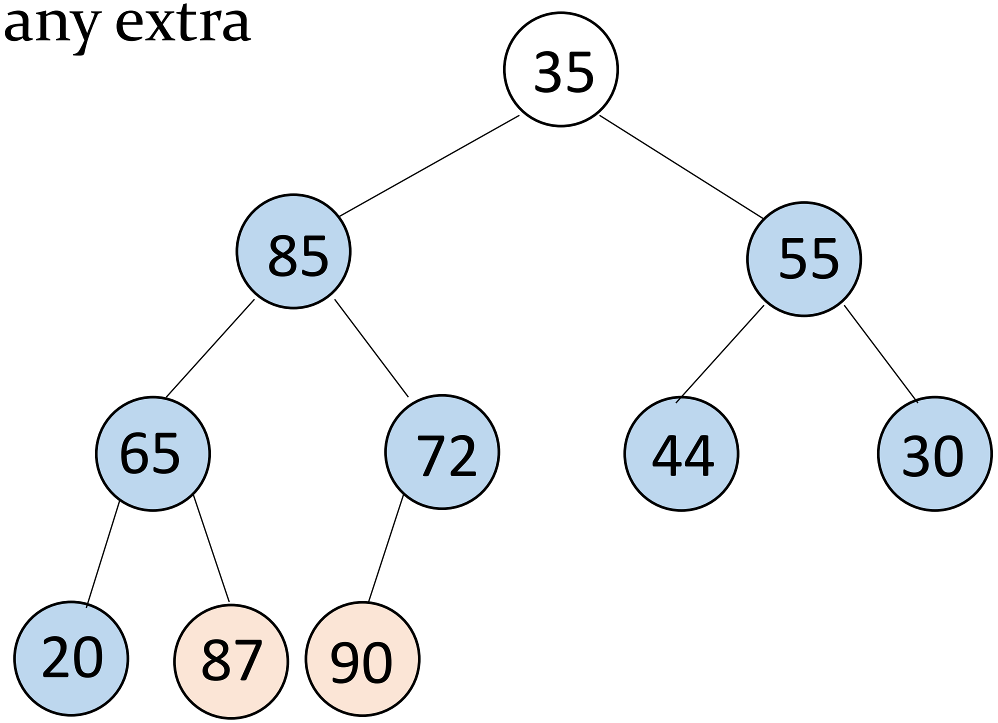
Heap Sort – Let’s be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this “in-place” without using any extra array space.

90	85	55	65	80	44	30	20	35	72

87	85	55	65	72	44	30	20	35	90

35	85	55	65	72	44	30	20	87	90



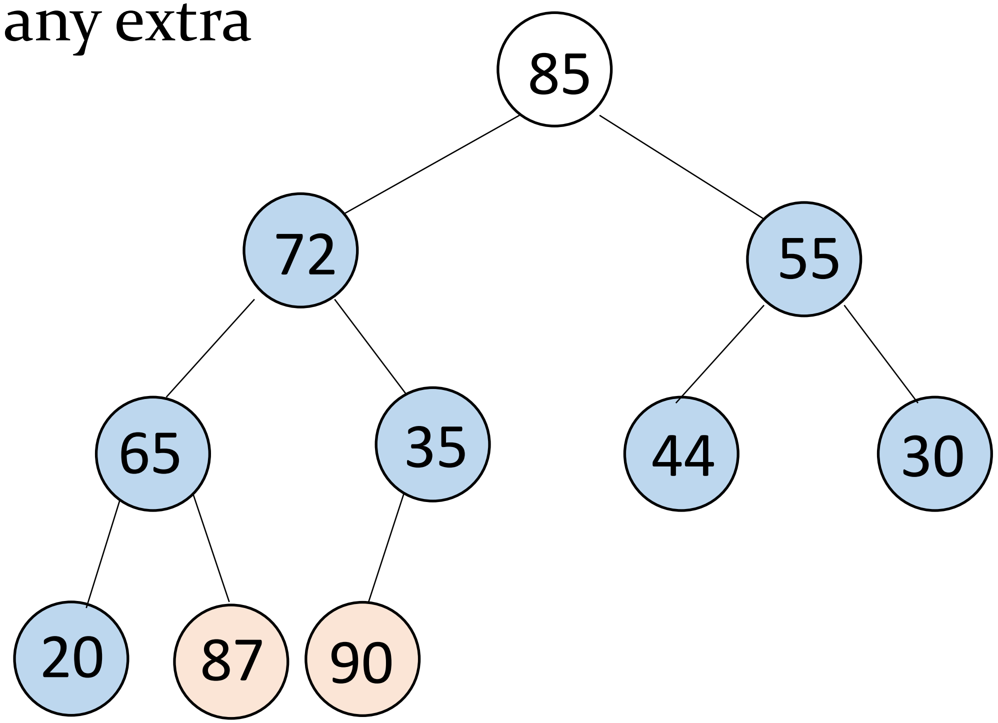
Heap Sort – Let's be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this "in-place" without using any extra array space.

90	85	55	65	80	44	30	20	35	72

87	85	55	65	72	44	30	20	35	90

85	72	55	65	35	44	30	20	87	90



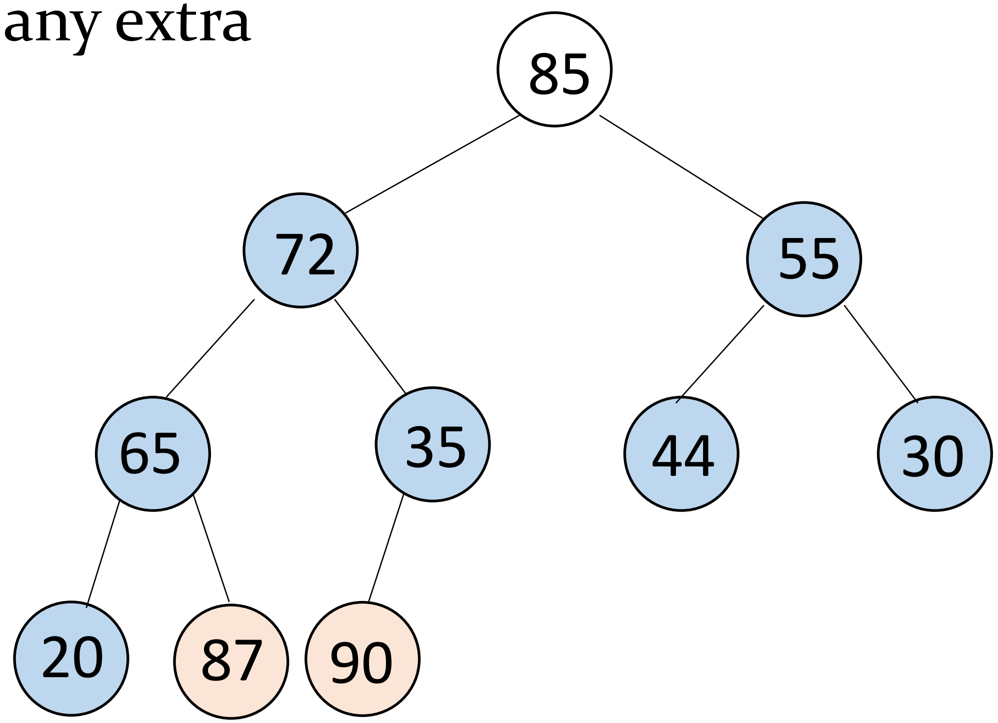
Heap Sort – Let's be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this "in-place" without using any extra array space.

90	85	55	65	80	44	30	20	35	72

87	85	55	65	72	44	30	20	35	90

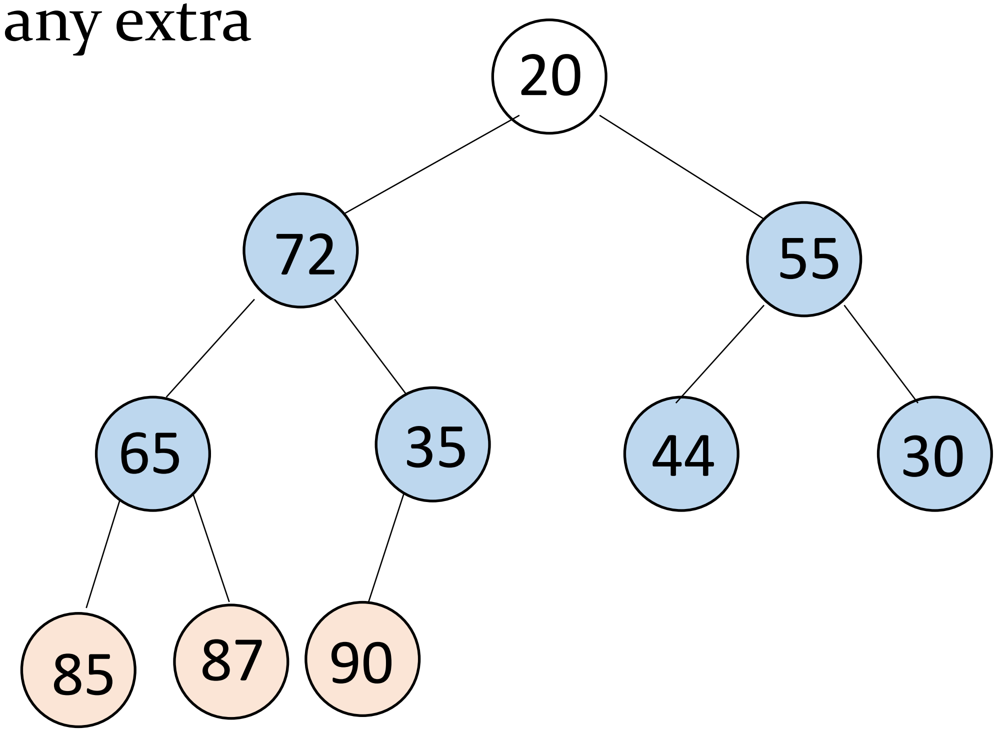
20	72	55	65	35	44	30	85	87	90



Heap Sort – Let's be more precise

- modify a max-heap-ordered array into a sorted array.
- We want to do this "in-place" without using any extra array space.

90	85	55	65	80	44	30	20	35	72
87	85	55	65	72	44	30	20	35	90
20	72	55	65	35	44	30	85	87	90



Heap Sort – Let's be more precise

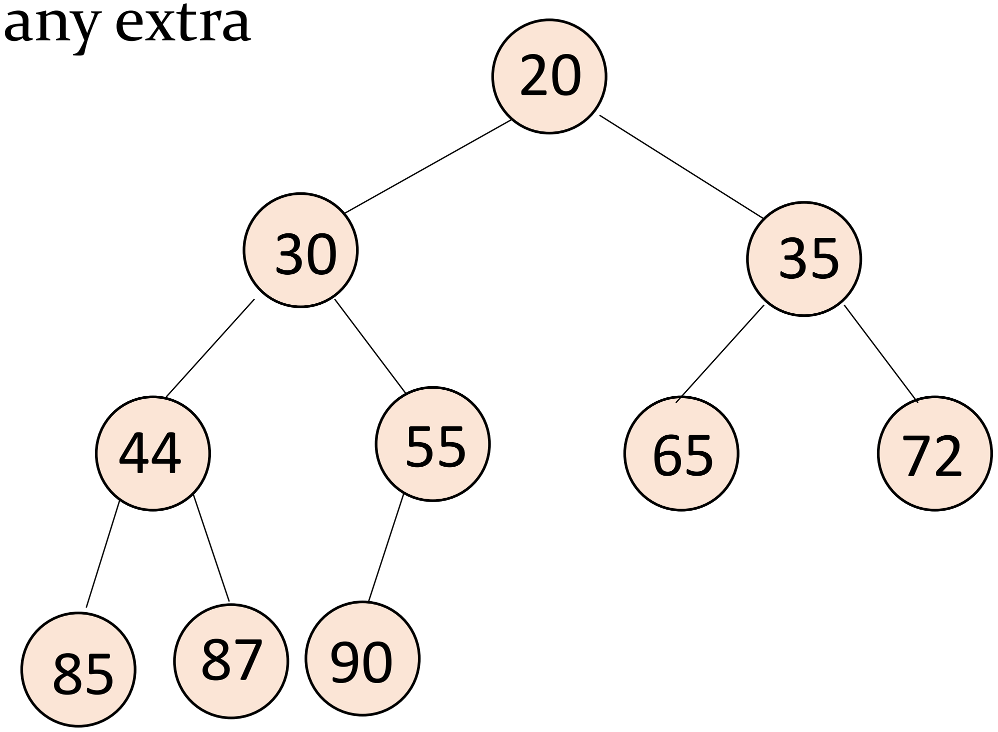
- modify a max-heap-ordered array into a sorted array.
- We want to do this "in-place" without using any extra array space.

90	85	55	65	80	44	30	20	35	72

87	85	55	65	72	44	30	20	35	90

20	72	55	65	35	44	30	85	87	90

20	30	35	44	55	65	72	85	87	90



Heap Sort – Psudocode

HeapSort(A) #sort any array A into non-descending order '

BuildMaxHeap(A) # convert any array A into a heap-ordered one

for $i \leftarrow A.size$ downto

swap $A[1]$ and $A[i]$ # Step 1: swap the first and the last

$A.size \leftarrow A.size - 1$ # Step 2: decrement size of heap

BubbleDown$(A, 1)$ # Step 3: bubble down the 1st element in A

Does it work?

It works for an array A that is initially heap ordered,
it does work NOT for any array!

BuildMaxHeap

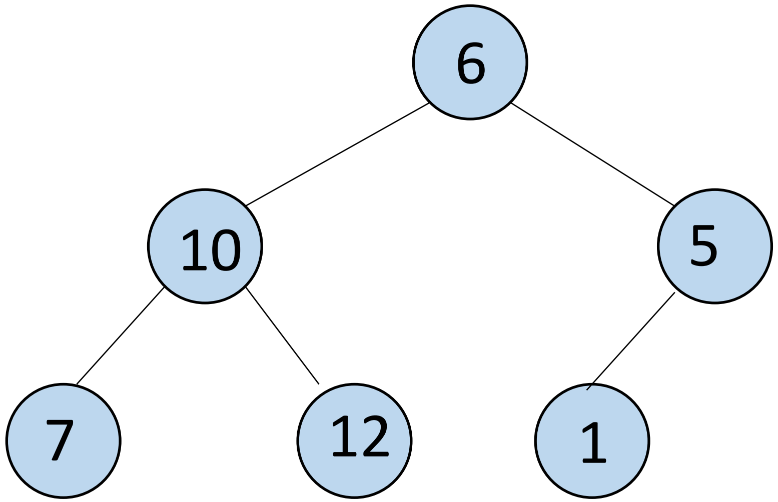
Build Max Heap

Converts an array into a max-heap ordered array, in $O(n)$ time

Build Max Heap

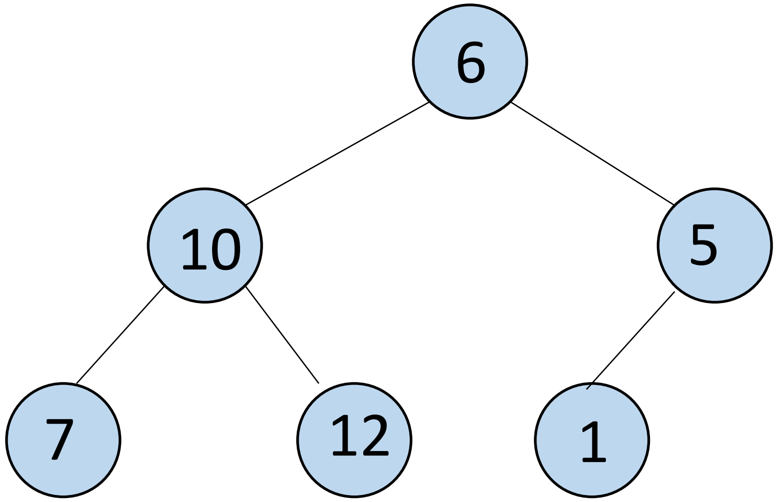
A given arbitrary array

6	10	3	7	12	5	1



A Max-Heap

6	10	3	7	12	5	1



Build Max Heap – First idea

BuildMaxHeap(A):

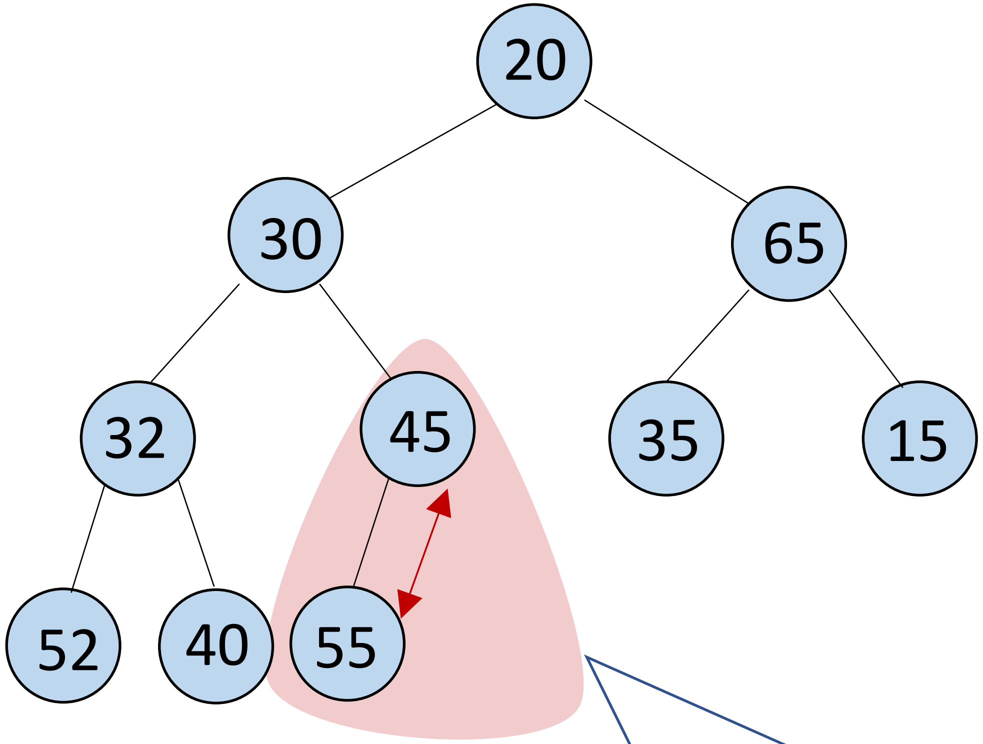
```
B ← empty array          # empty heap
for x ∈ A:
    Insert(B, x)         # heap insert
A ← B                    # overwrite A with B
```

Running time:

Each Insert takes $O(\log n)$, there are n inserts. So it's $O(n \log n)$, not very exciting. Not in-place, needs a second array.

Build Max Heap – A better Algorithm

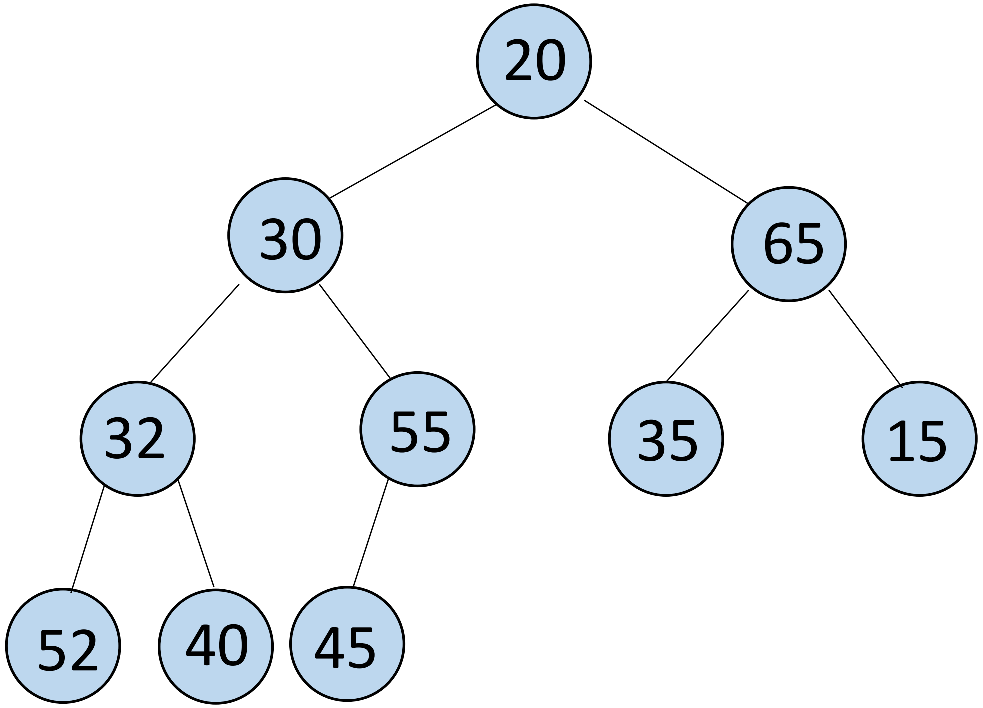
Fix heap order, from bottom up.



not a heap: root is out of order, bubble-down the root

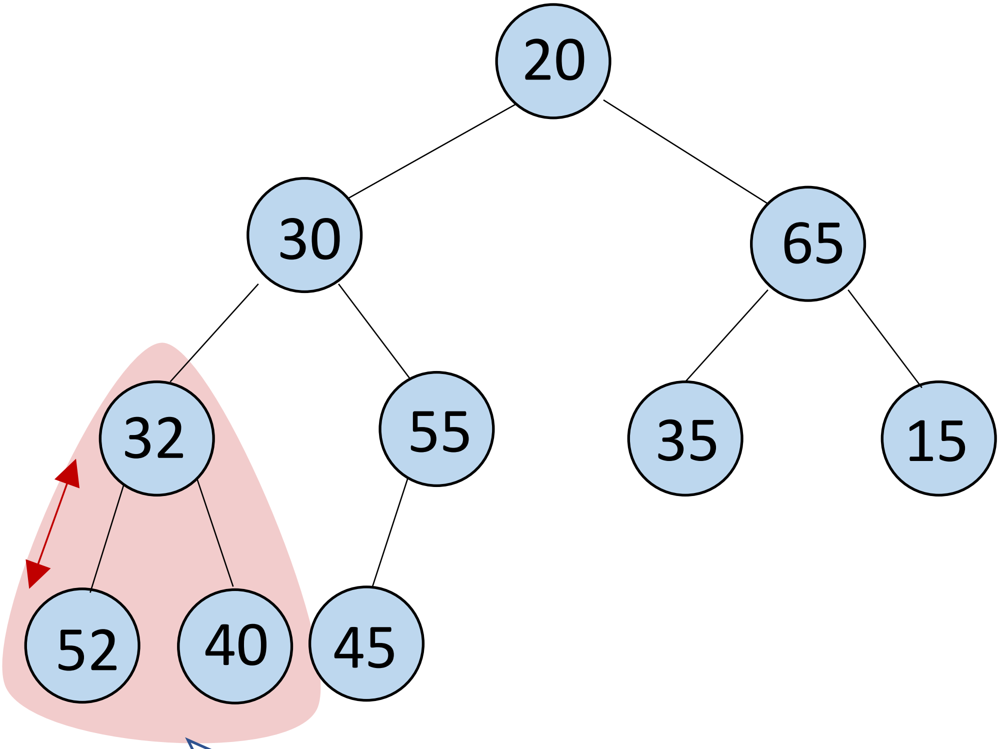
Build Max Heap – A better Algorithm

Fix heap order, from bottom up.



Build Max Heap – A better Algorithm

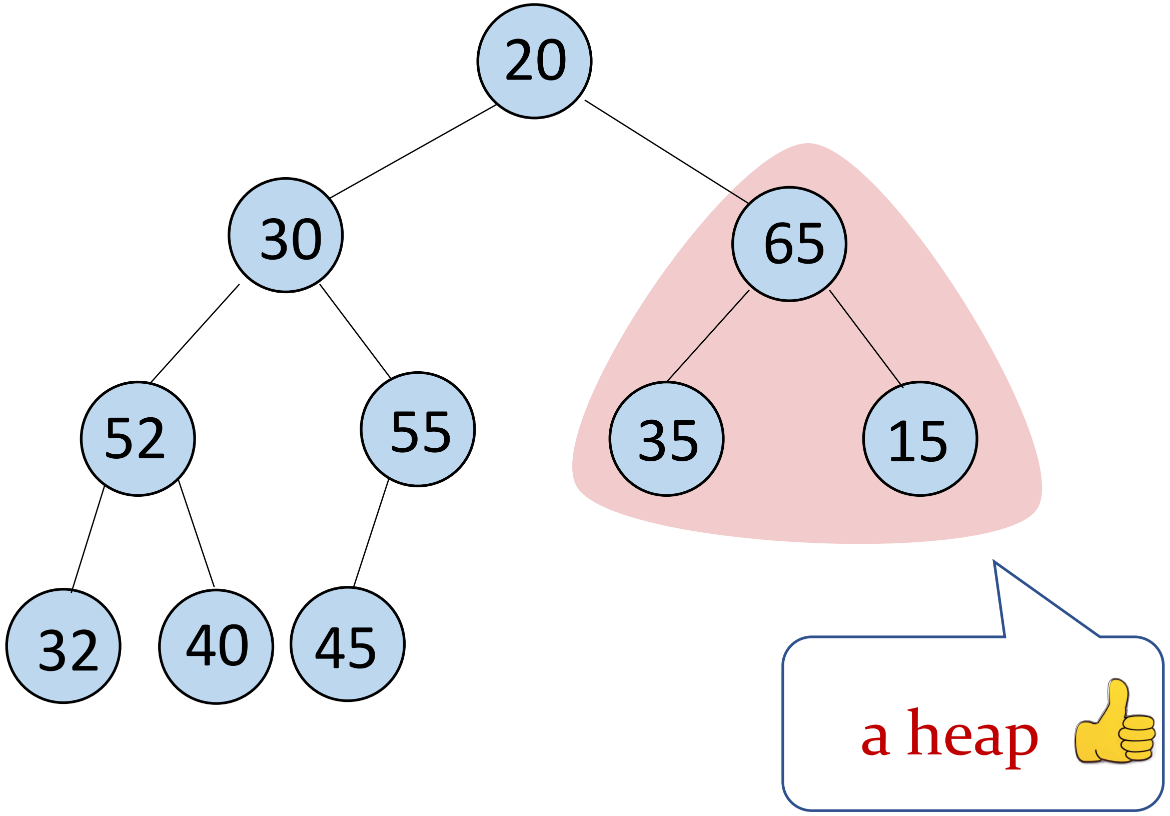
Fix heap order, from bottom up.



not a heap: root is out of order, bubble-down the root

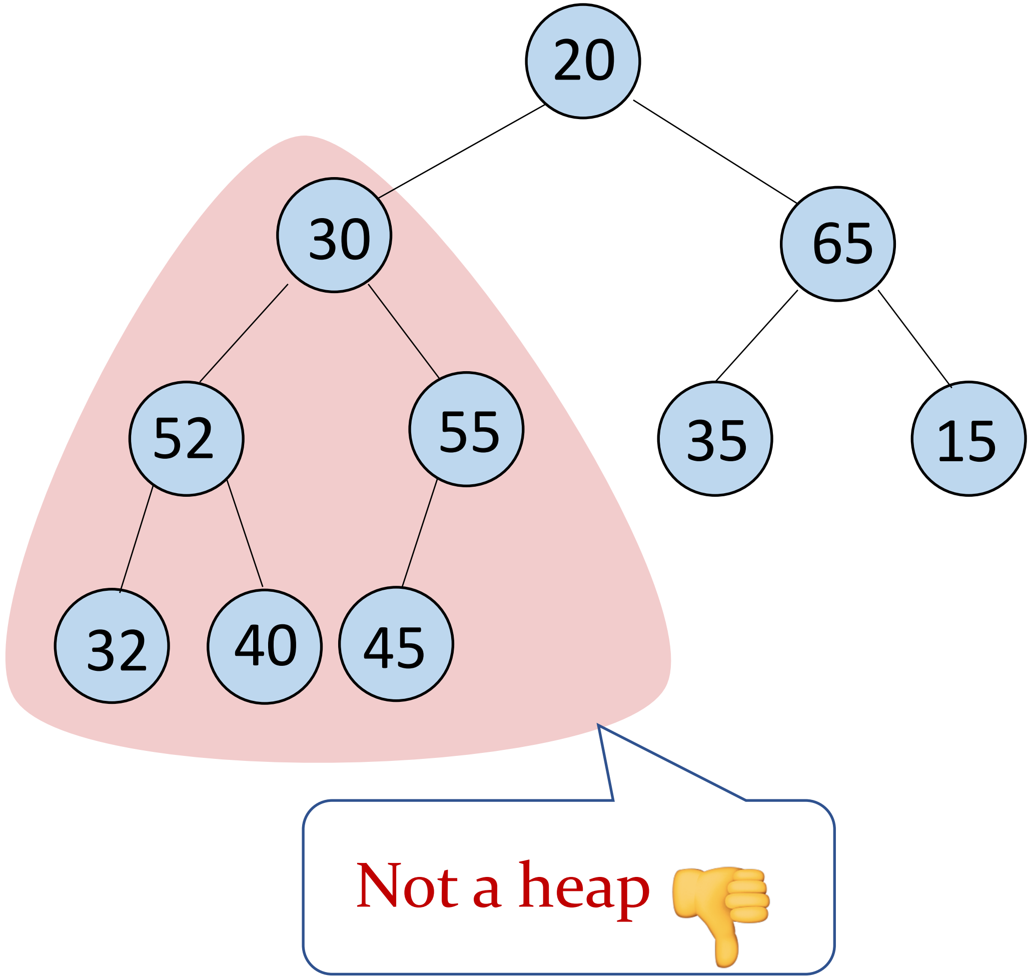
Build Max Heap – A better Algorithm

Fix heap order, from bottom up.



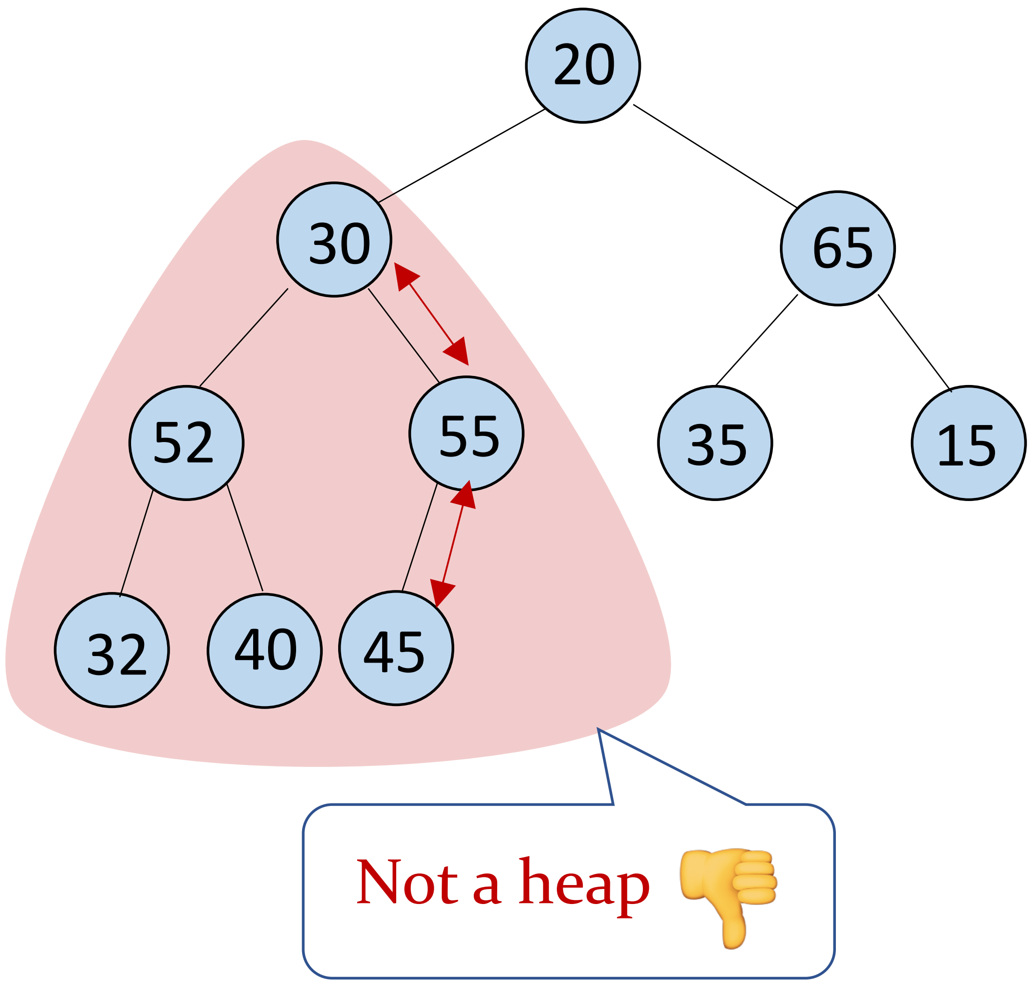
Build Max Heap – A better Algorithm

Fix heap order, from bottom up.



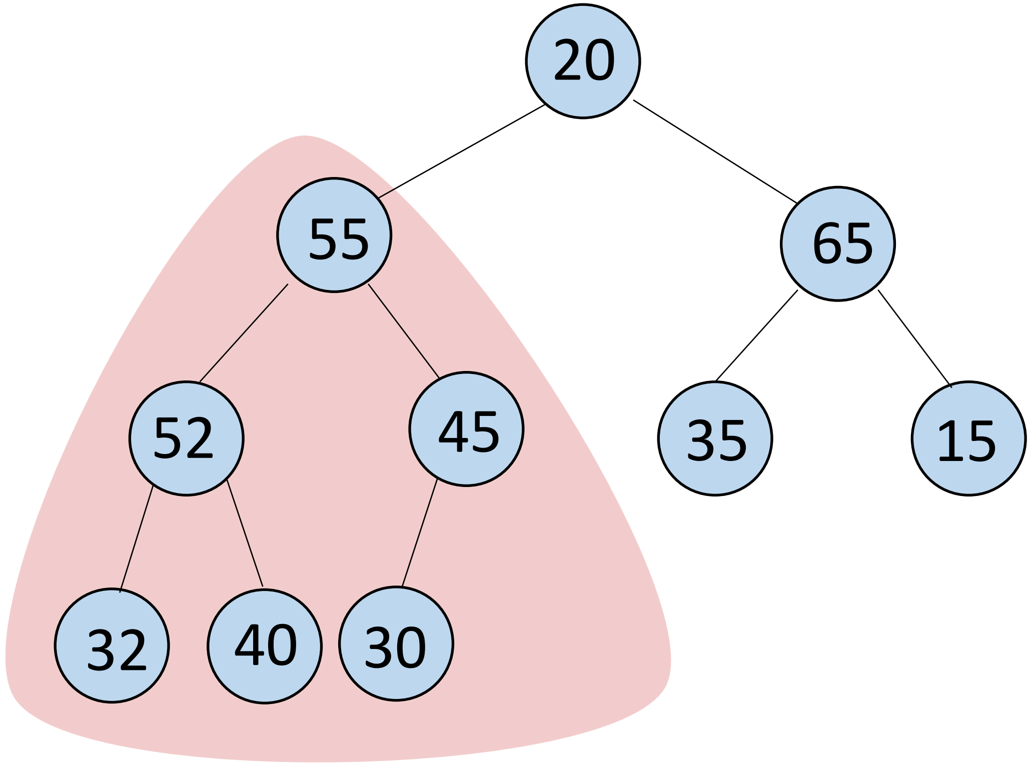
Build Max Heap – A better Algorithm

Fix heap order, from bottom up.



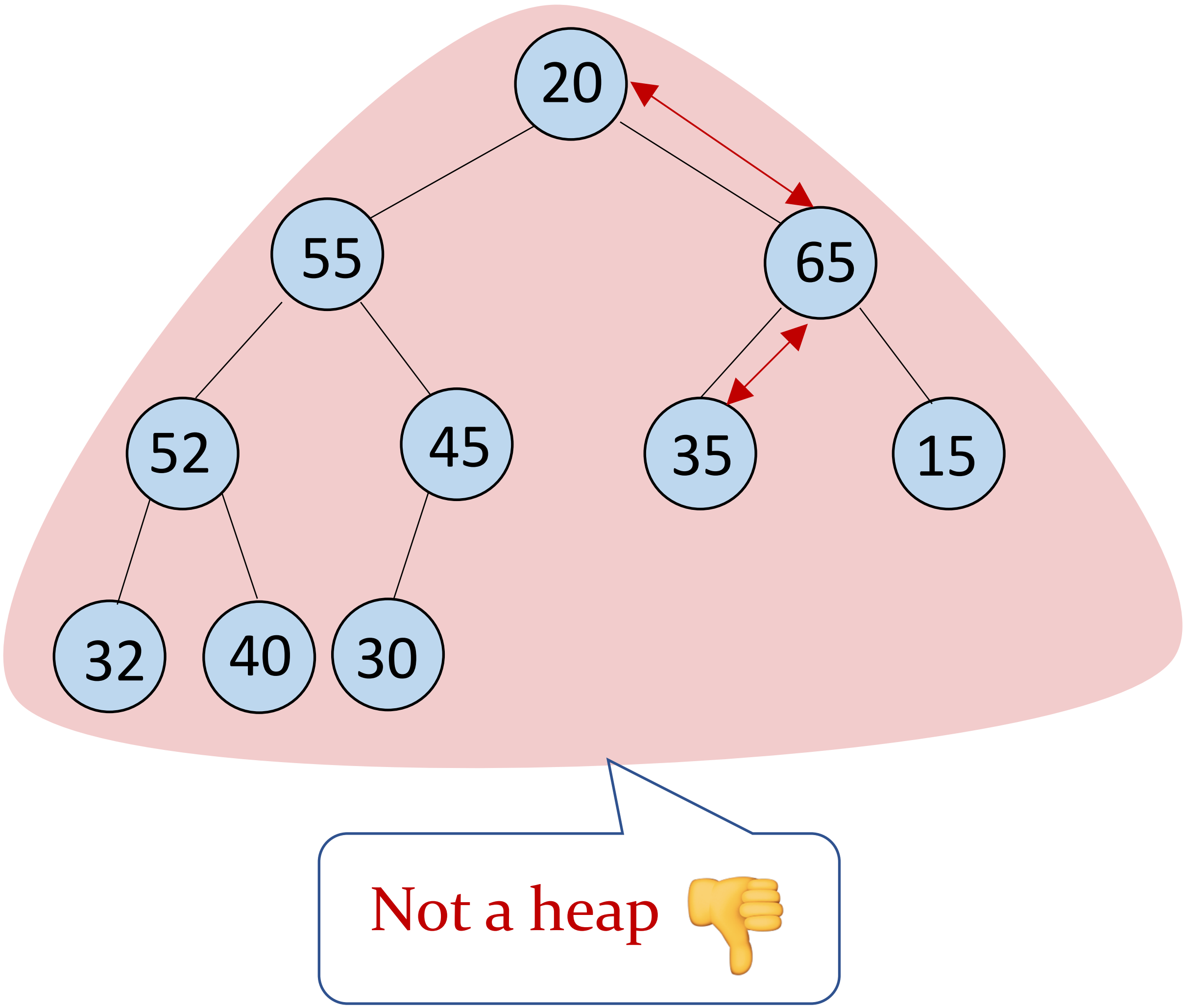
Build Max Heap – A better Algorithm

Fix heap order, from bottom up.



Build Max Heap – A better Algorithm

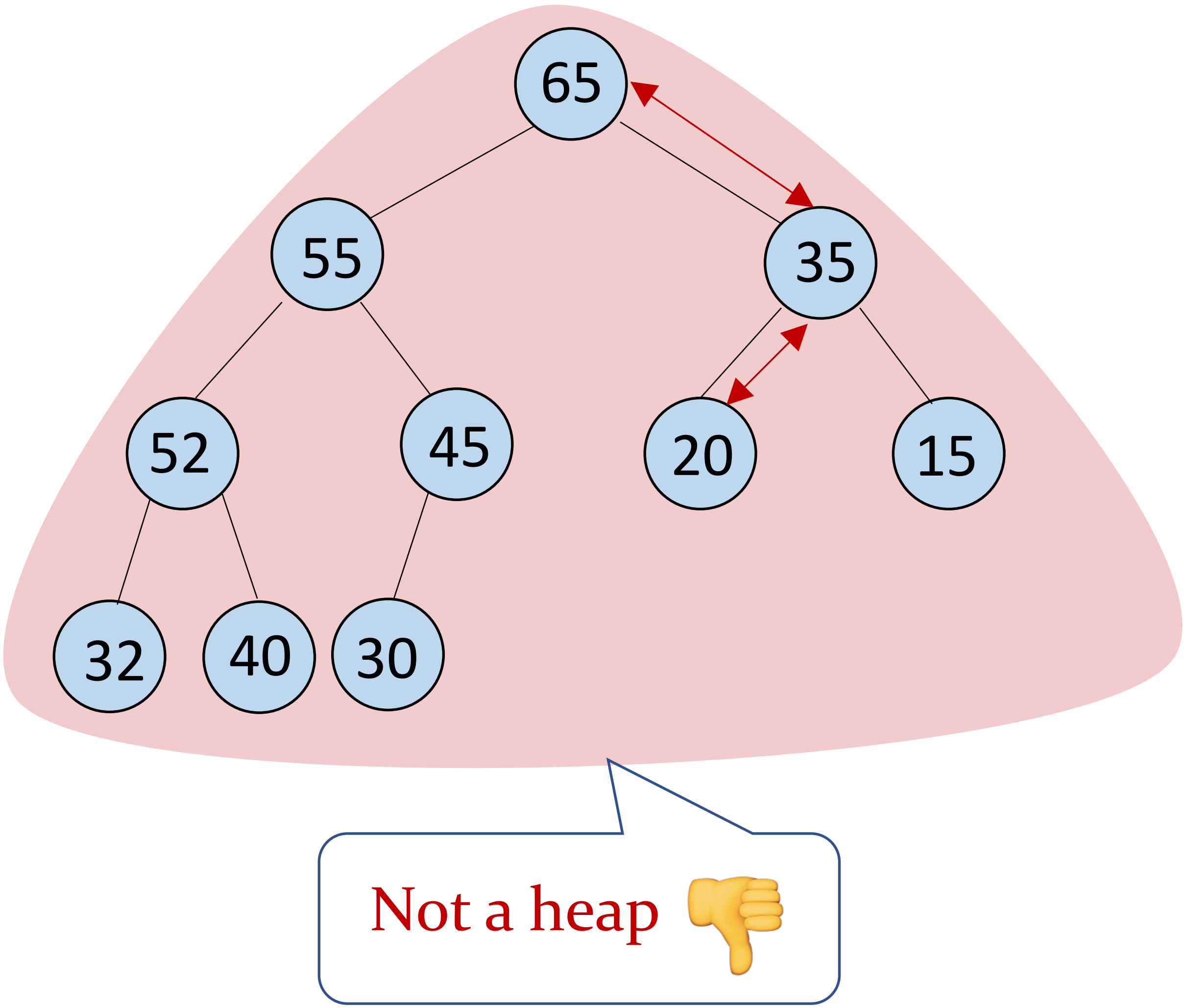
Max-Heap built!



Build Max Heap – A better Algorithm

Max-Heap built!

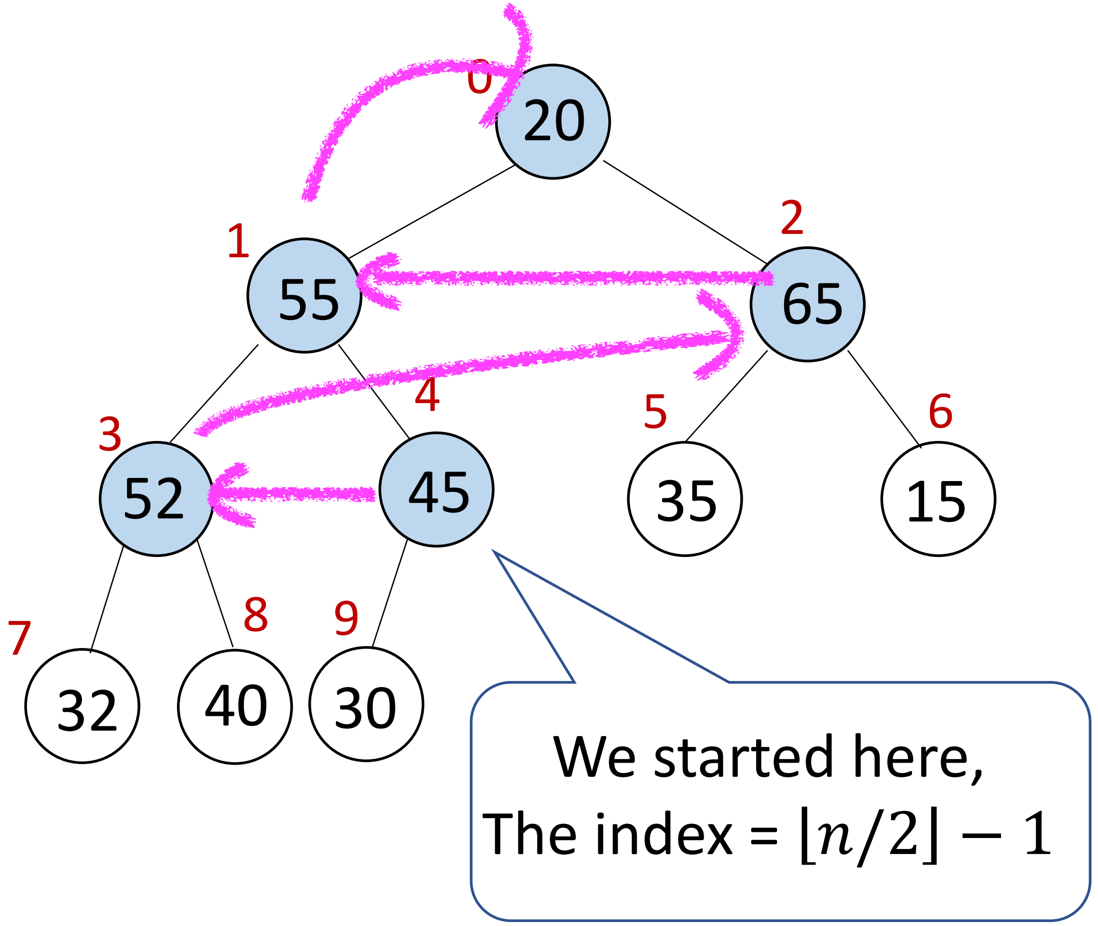
We did nothing
but
bubbling-down!



Build Max Heap – A better Algorithm

What is the starting index?

We always start from $\lfloor n/2 \rfloor - 1$ and go down to 0.

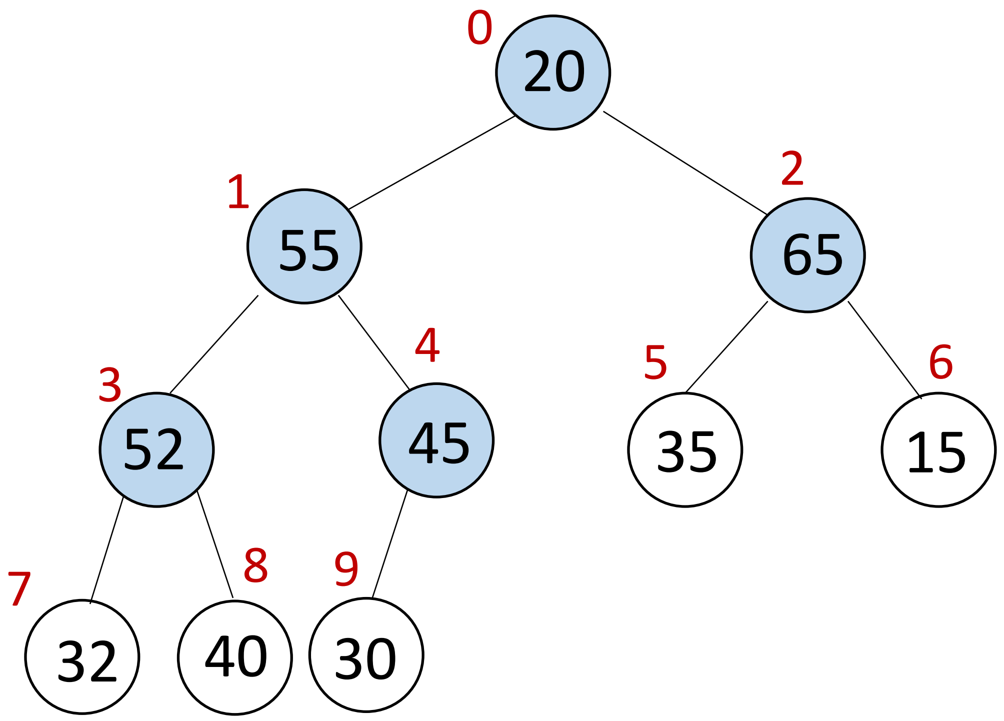


Build Max Heap – A better Algorithm

Build-Max-Heap(A):

for i ← $\lfloor n/2 \rfloor - 1$ downto 1:

BubbleDown(A, i)



- It's in-place, no need for extra array (we did nothing but swappings).
- It's worst-case running time is $O(n)$, instead of $O(n \log n)$.

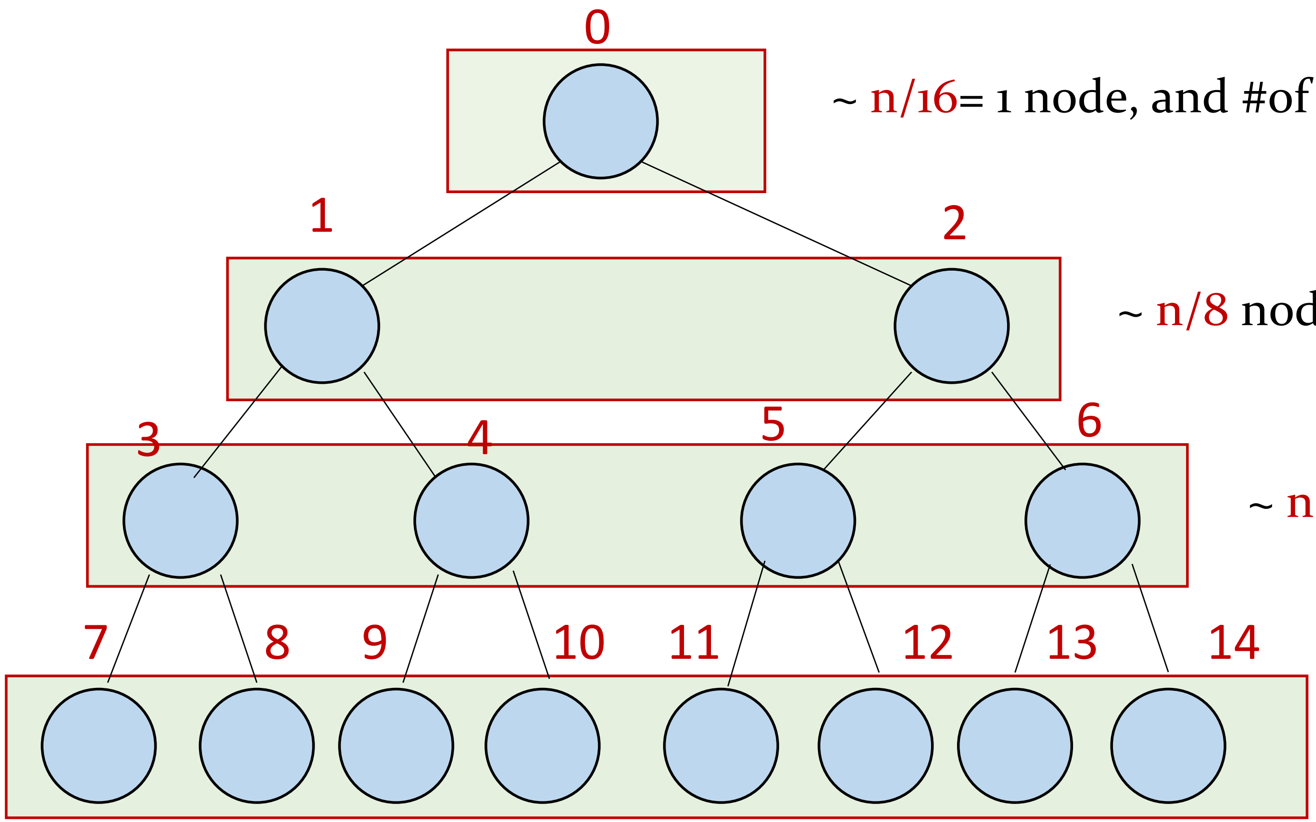
$O(n)$?
Really!?

Analysis:

Worst-case running time of BuildMaxHeap(A)

Intuition

A complete binary tree:



~ n/16= 1 node, and #of swaps per bubble-down: ≤3

~ n/8 nodes, and # of swaps per bubble-down: ≤2

~ n/4 nodes # of swaps per bubble-down: ≤1

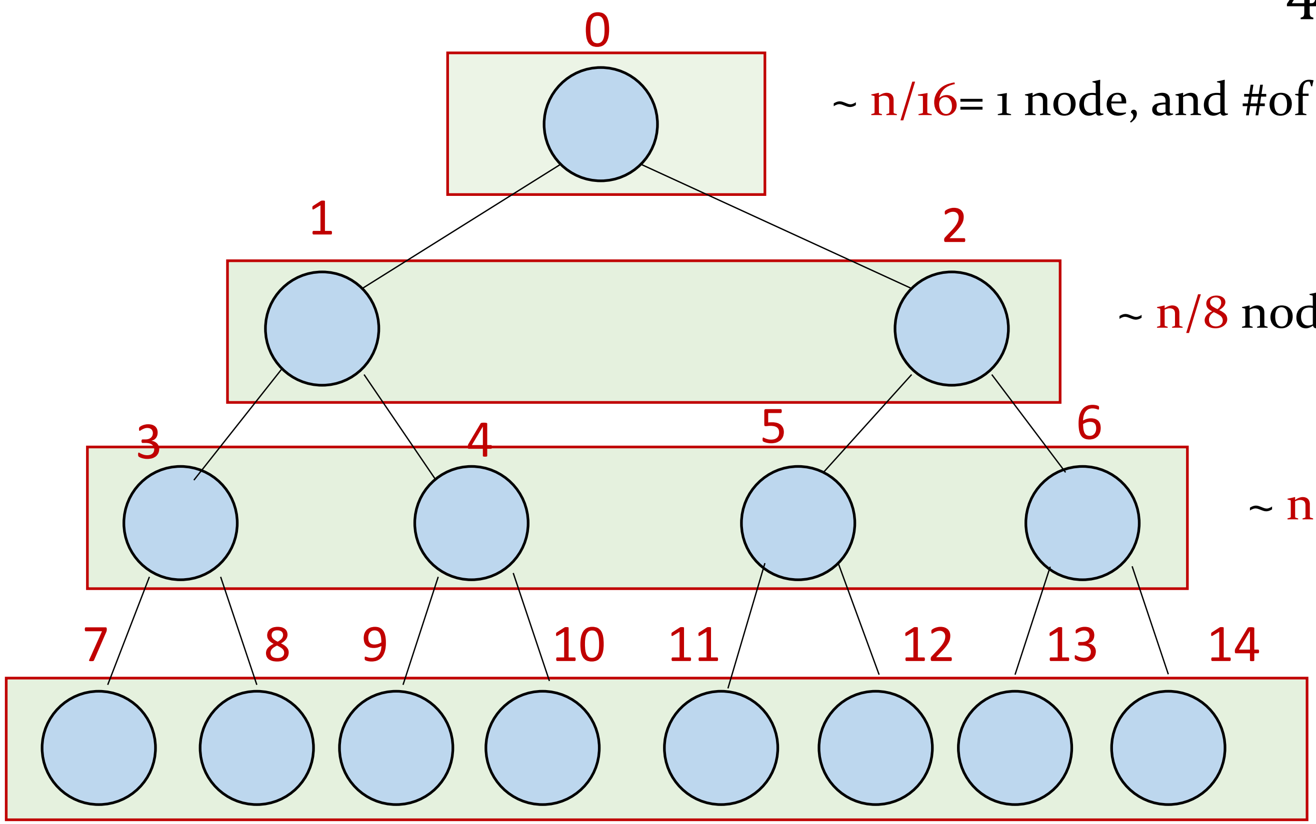
~ n/2 nodes, and no work done at this level.

How many levels? ~ log n

Total number of swaps:

A complete binary tree:

$$T(n) \leq 1.\frac{n}{4} + 2.\frac{n}{8} + \ \ldots + \log n.\frac{n}{2^{logn}}$$



~ n/16= 1 node, and #of swaps per bubble-down: ≤3

~ n/8 nodes, and # of swaps per bubble-down: ≤2

~ n/4 nodes # of swaps per bubble-down: ≤1

~ n/2 nodes, and no work done at this level.

How many levels? ~ log n

BuildMaxHeap-Running time

$$T(n) \leq 1.\frac{n}{4} + 2.\frac{n}{8} + \ \ldots + \log n.\frac{n}{2^{logn}} \ = \sum_{i=1}^{\log n} i.\frac{n}{2^{i+1}} \leq n \sum_{i=1}^{\infty} \frac{i}{2^{i+1}} = \text{n}$$

$$\sum_{i=1}^{\infty} \frac{i}{2^{i+1}} = 1$$

Building a Max-Heap has a linear time complexity.

Summary

	Time Complexity
BuildMaxHeap	$\Theta(n)$
Maximum	$\Theta(1)$
Extract Max	$\Theta(\log n)$
Delete	$\Theta(\log n)$
IncreaseKey	$\Theta(\log n)$
Heap Sort	$\Theta(\text{n}\log n)$

Questions?