Data Structures and Algorithms

Heaps

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Dr. Steven Halim for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

Recording of modifications

- Currently, there are no modification on these contents.

Outline

What are you going to learn in this lecture?

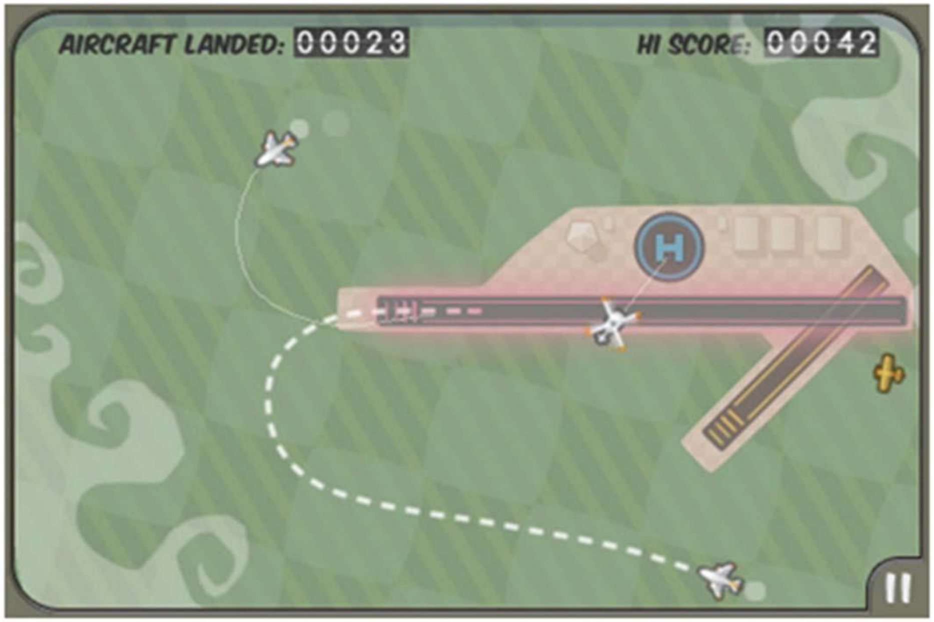
- Motivation: Abstract Data Type: **PriorityQueue**
- With major help from VisuAlgo Binary Heap Visualization
  - **Binary Heap** data structure and it's operations
  - Building Heap from a set of **n** numbers in **O(n)**
  - **Heap Sort** in **O(n log n)**
- CS2010 PS1 Overview: "Scheduling Deliveries, v2015"

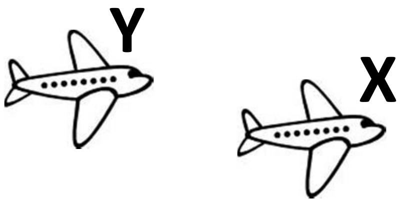
Reference in CP3 book: Page 43-47 + 148-150

Abstract Data Type: PriorityQueue (1)

Imagine that you are the Air Traffic Controller:

- You have scheduled the next **aircraft X** to land in the **next 3 minutes**, and **aircraft Y** to land in the **next 6 minutes**
- Both have enough fuel for at least the next **15 minutes** and both are just **2 minutes** away from your airport





The next two slides are hidden…

Attend the lecture to figure out

Abstract Data Type: PriorityQueue

Important Basic Operations:

- Enqueue(x)
  - Put a new item x in the priority queue PQ (in some order)
- y ← Dequeue()
  - Return an item y that has the **highest priority** (key) in the PQ
  - If there are more than one item with highest priority, return the one that is inserted first (FIFO)

Note: We can always define highest priority = higher number or it’s opposite: highest priority = lower number

A Few Points To Remember

Data Structure (DS) is…

- A way to **store** and **organize data** in order to support efficient insertions, searches, deletions, queries, and/or updates

Most data structures have propert(ies)

- Each operation on that data structure has to **maintain** that propert(ies)

PriorityQueue Implementation (2)

(Circular) **Array-Based** Implementation (Strategy 2)

- Property: dequeue() operation returns the correct item
- Enqueue(x)
  - Put the new item at the **back of the queue**, O(1)
- y □ Dequeue()
  - Scan the whole queue, return **first item with highest priority**, O(n)

Index	0 (front)	1 (back)
Key	Aircraft X*	Aircraft Y*
		Aircraft Z**

We may need to close the gap if this operation causes it, also O(n)

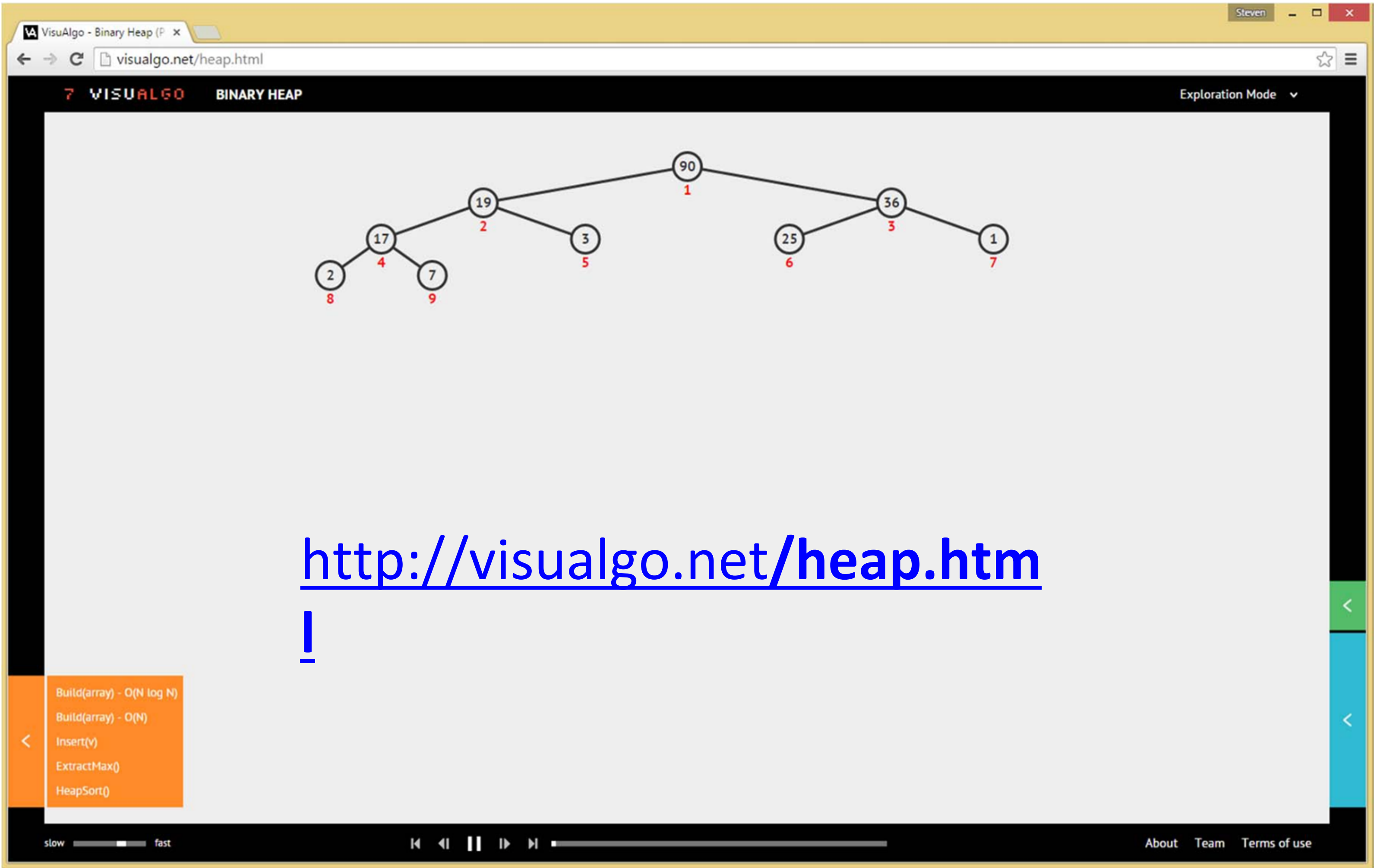
Index	0 (front)	1	2 (back)
Key	Aircraft X*	Aircraft Y*	Aircraft Z**

PriorityQueue Implementation (3)

If we just stop at CS1020 knowledge level:

Strategy	Enqueue	Dequeue
Circular-Array-Based PQ (1)	O(n)	O(1)
Circular-Array-Based PQ (2)	O(1)	O(n)
Can we do better?	O(?)	O(?)

If n is large, our queries are slow…



[http://visualgo.net/**heap.html**](http://visualgo.net/heap.html)

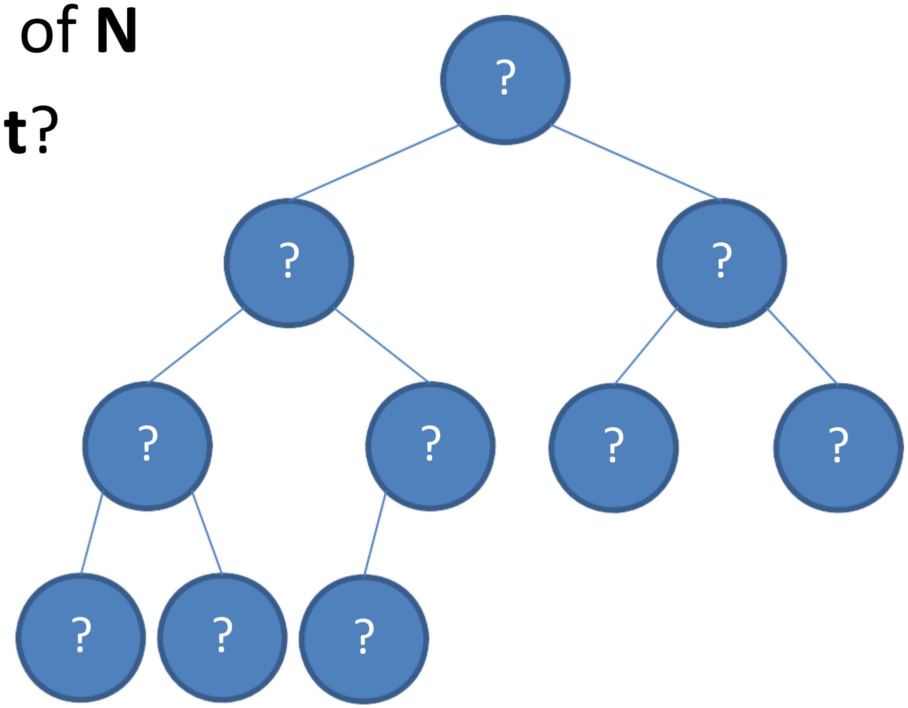
INTRODUCING BINARY HEAP DATA STRUCTURE

Complete Binary Tree

Introducing a few concepts:

- **Complete** Binary Tree
  - Binary tree in which every level, *except possibly the last*, is completely filled, and all nodes are as far left as possible
- Important Q:

  If you have a complete binary tree of **N** items, what will be **the height of it**?
  - Height = number of levels-1 = max edges from root to deepest leaf



The Height of a Complete Binary Tree of N Items is...

1. O(N)
2. O(sqrt(N))
3. O(log N)
4. O(1)

Memorize this answer!
We will need that for *nearly all* time complexity analysis of binary heap

Storing a Complete Binary Tree

Q: Why not 0-based?

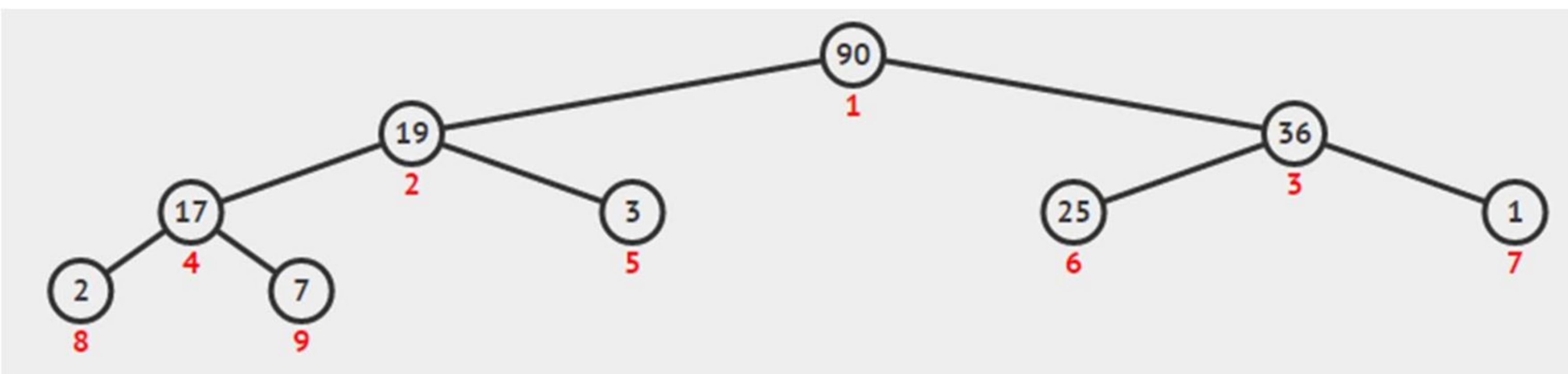
As a **1-based** compact array: `A[1..size(A)]`

`size(A)`

0	1	2	3	4	5	6	7	8	9	10	11
NIL	90	19	36	17	3	25	1	2	7	-	-

`heapsize` $\leq$ `size(A)`

Navigation operations:

- `parent(i) = floor(i/2)` except for `i = 1 (root)`
- `left(i) = 2*i`, No left child when: `left(i) > heapsize`
- `right(i) = 2*i+1`, No right child when: `right(i) > heapsize`
- 

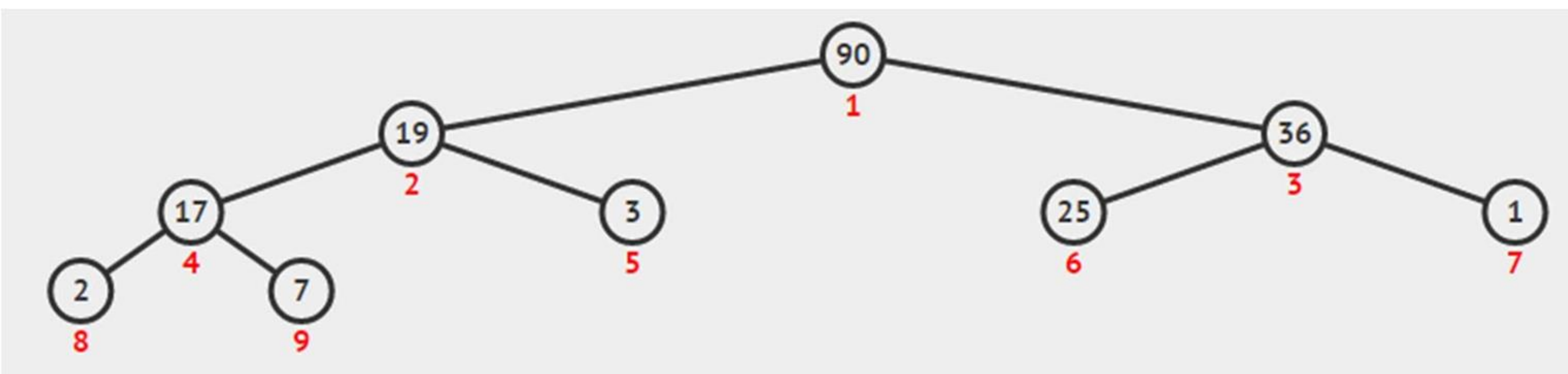
Binary Heap Property

Binary Heap property (except root)

- `A[parent(i)]` ≥ `A[i]` **(Max Heap)**
- `A[parent(i)]` ≤ `A[i]` **(Min Heap)**

Q: Can we write Binary Max Heap property as:
`A[i]` ≥ `A[left(i)]`
`&&`
`A[i]` ≥ `A[right(i)]`
?

Without loss of generality, we will use (**Binary Max) Heap** for all examples in this lecture and we ensure that the numbers are distinct



The largest element in a **Binary Max Heap** is stored at…

1. One of the leaves
2. One of the internal vertices
3. Can be anywhere in the heap
4. The root

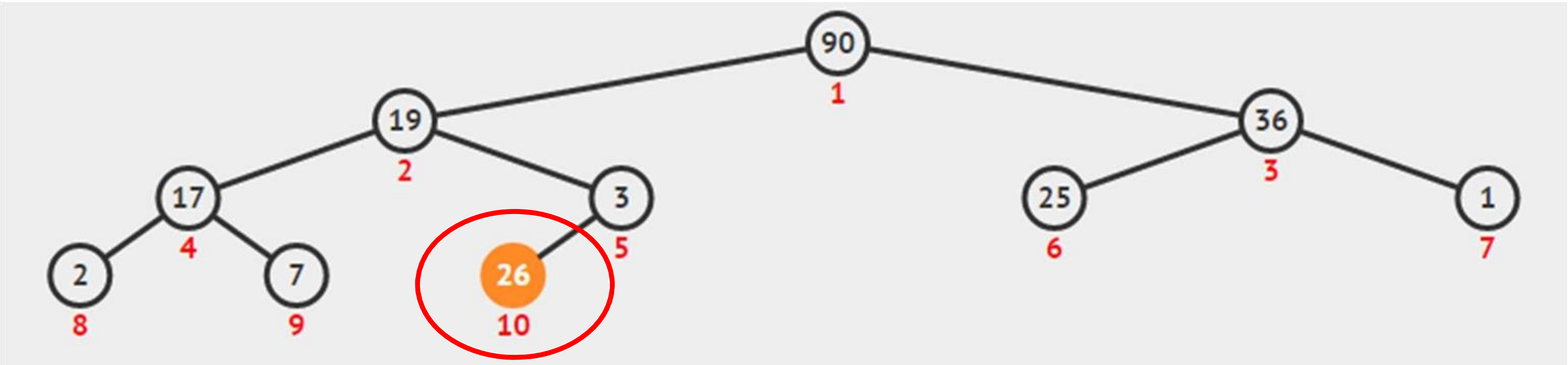
Insertion to an Existing B Max Heap

The most appropriate insertion point into an existing Binary Max Heap is at **A[heapsize]**

- Q: Why?

  A: ______

- But Binary Max Heap property can still be violated?
  - No problem, we use `ShiftUp(i)` to fix the heap property



Insert(v) – Pseudo Code

```
Insert(v)
  heapsize = heapsize+1; // extend, O(1)
  A[heapsize] = v     // insert at the back, O(1)
  ShiftUp(heapsize) // fix the heap property
                    // in O(?)
// Preliminary analysis:
// Insert(v) depends on ShiftUp(i)
```

ShiftUp – Pseudo Code

This name is not unique, the alternative names are:

ShiftUp/BubbleUp/IncreaseKey/etc

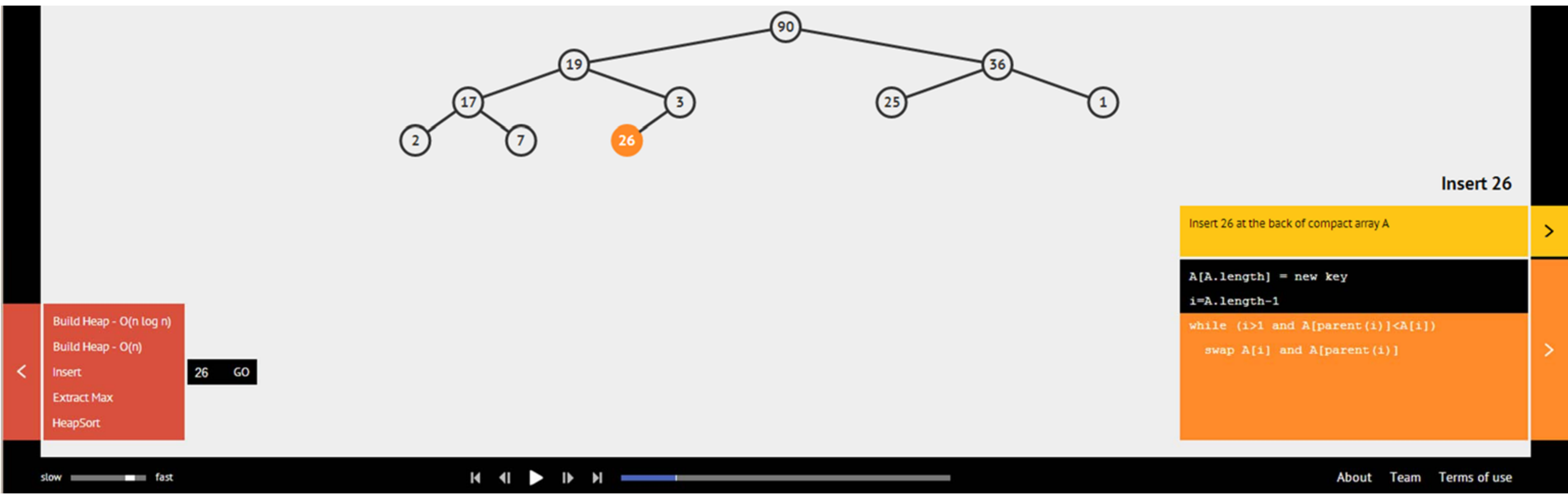
“not root” “violates max heap property”

```
ShiftUp(i)
  while i > 1 and A[parent(i)] < A[i] // don't swap
    swap(A[i],
     A[parent(i)])  i =
    parent(i)
// Analysis: ShiftUp() runs in __
```

Binary Heap: Insert(v)

Ask VisuAlgo to perform various insert operations on the sample Binary (Max) Heap

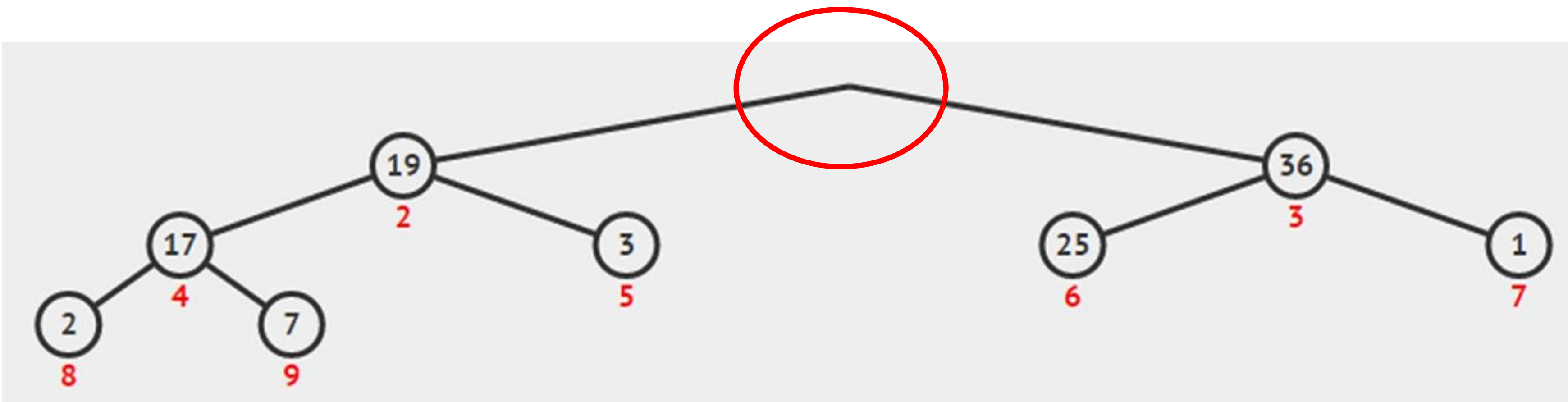
In the screen shot below, we show the first step of **Insert(26)**



Deleting Max Element (1)

The max element of a Binary Max Heap is at **the root**

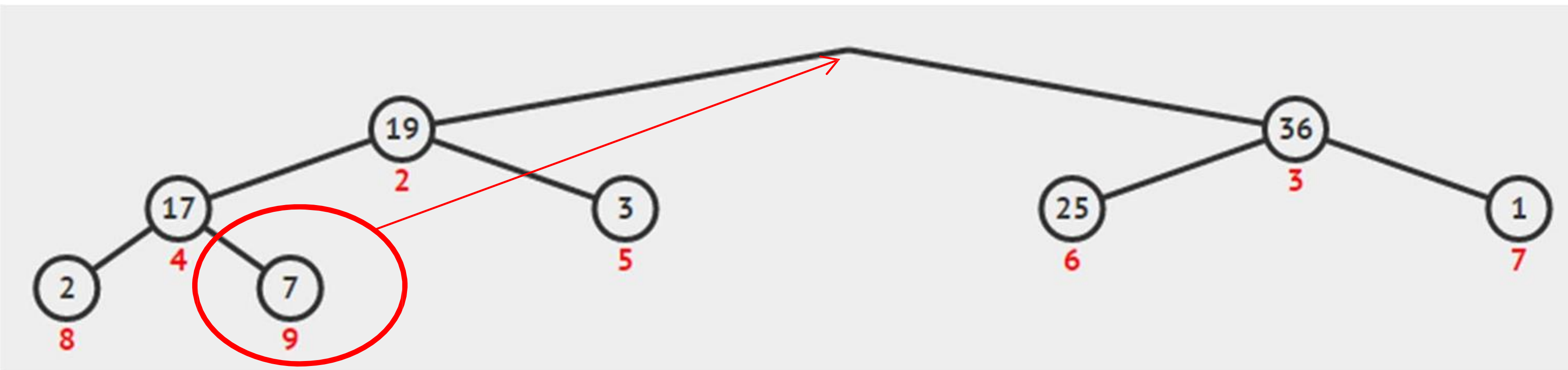
- But simply taking the root out from a Binary Max Heap will disconnect the complete binary tree ☹
  - We do not want that…

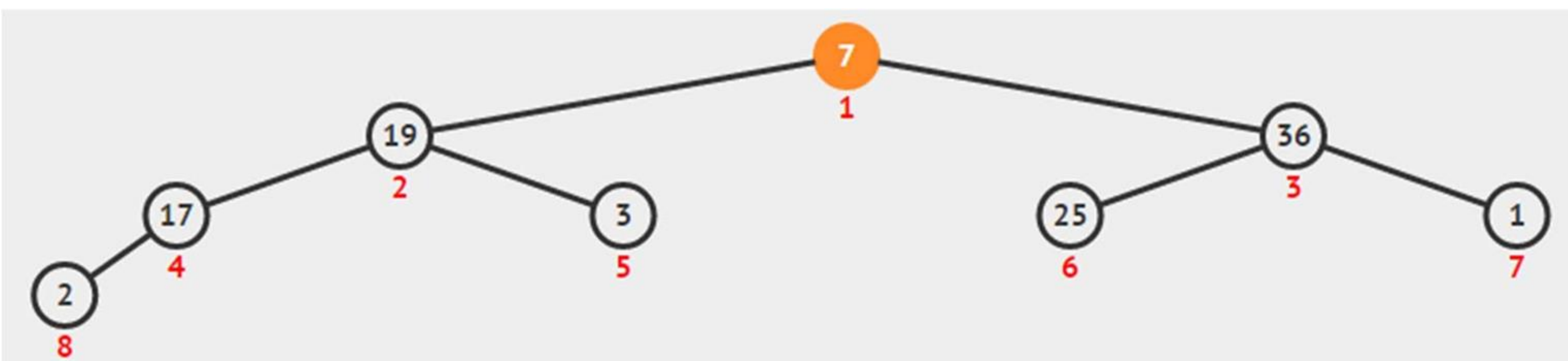


- Q: Which node is the best candidate to **replace** the root yet still maintain the complete binary tree property?

Deleting Max Element (2)

- A: The___**leaf**
  - Which is the last element in the compact array
- But the heap property can still be violated?
  - No problem, this time we call `ShiftDown(1)`





ExtractMax - Pseudocode

```
ExtractMax()
  maxV  A[1] // O(1)
  A[1]  A[heapsize] // O(1)
  heapsize = heapsize-  // O(1)
  1  ShiftDown(1) //
  O(?)  return maxV

// Preliminary analysis:
// ExtractMax() depends on ShiftDown()
```

ShiftDown – Pseudo Code

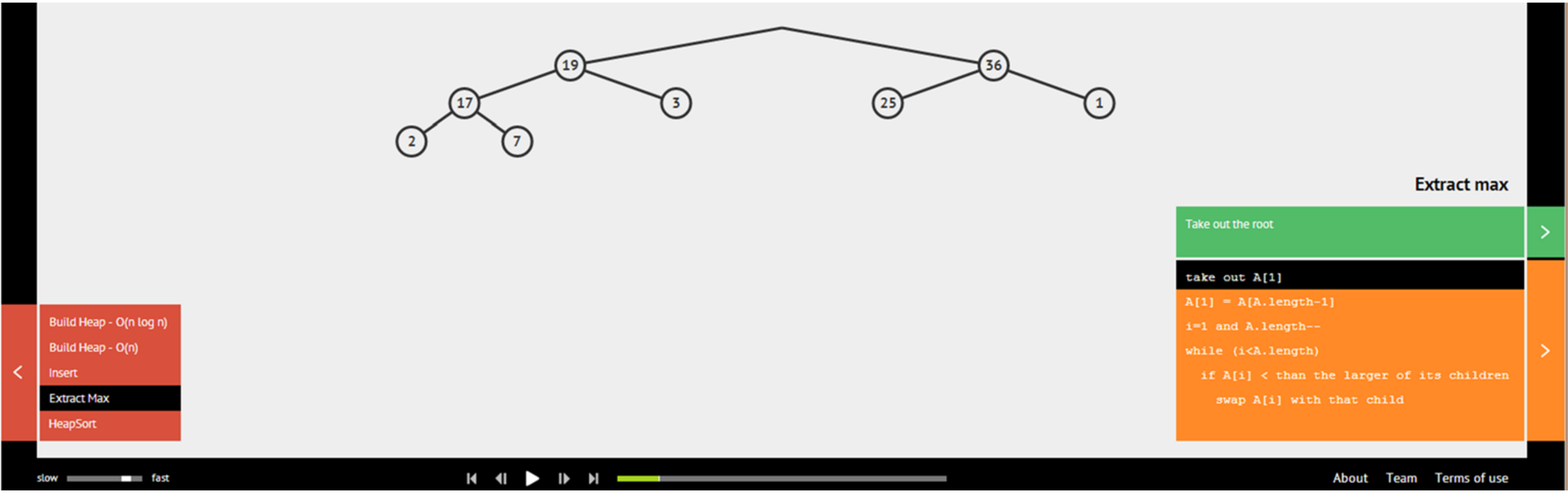
Again, the name is not unique:
ShiftDown/BubbleDown/Heapify/etc

```
ShiftDown(i)
  while i <= heapsize
    maxV ← A[i]; max_id ← i;
    if left(i) <= heapsize and maxV < A[left(i)]
      maxV ← A[left(i)]; max_id ← left(i)
    if right(i) <= heapsize and maxV < A[right(i)]
      maxV ← A[right(i)]; max_id ← right(i)
    // be careful with the implementation
    if (max_id != i)
      swap(A[i], A[max_id])
      i ← max_id;
    else
      break; // Analysis: ShiftDown() runs in ___
```

Binary Heap: ExtractMax()

Ask VisuAlgo to perform various ExtractMax() operations on the sample Binary (Max) Heap

In the screen shot below, we show the first step of **ExtractMax()** from the sample Binary (Max) Heap



PriorityQueue Implementation (4)

Now, with knowledge of *non linear* DS from CS2010:

Strategy	Enqueue	Dequeue
Array-Based PQ (1)	O(n)	O(1)
Array-Based PQ (2)	O(1)	O(n)
Binary-Heap (actually uses array too)	Insert(key) O(log n)	ExtractMax() O(log n)

Summary so far:

Heap data structure is an efficient data structure ***-- O(log n) enqueue/dequeue operations --*** to implement ADT priority queue where the ‘key’ represent the ‘priority’ of each item

Next Items:

- Building Binary Max Heap from an ordinary Array, the O(n log n) version
- And the faster O(n) version
- Heap Sort, O(n log n)
- Java Implementation of Binary Max Heap
- PS1 overview and introduction of one more Binary Max Heap operation: UpdateKey that has been purposely left out from this lecture

LECTURE BREAK

Review: We have seen MergeSort in CS1020. It can sort **n** items in…

1. O($\mathbf{n}^2$)
2. O(**n** log **n**)
3. O(**n**)
4. O(log **n**)

HeapSort Pseudo Code

With a max heap, we can do sorting too 😊

- Just call ExtractMax() **n** times
- If we do not have a max heap yet, simply build one!

```
HeapSort(array)
  BuildHeap(array)  // O(?)
  n ← size(array)
  for i from 1 to n  // O(n)
    A[n-i+1] ← ExtractMax()  // O(log n)
  return A

// Preliminary analysis:
// HeapSort runs in O(? + n log n)
```

BuildHeap, O(n log n) Version

```
BuildHeapSlow(array)    naïve version
  //  n ⍰  size(array)
  A[0]  ⍰    0 //    entry
  dummy  for i = 1  O(n)
  toInsert(array[i-1]) // O(log n)
  n //

// Analysis: This clearly runs in O(n log n)
// So HeapSort in previous slide is O(n log n)
```

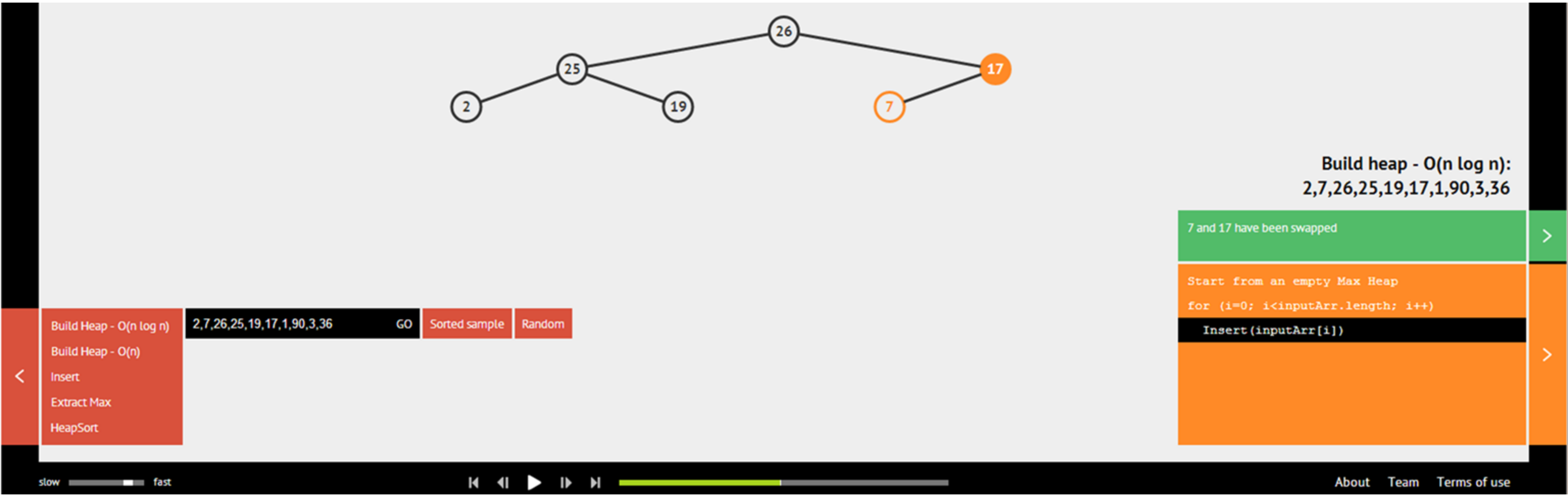
😊

Can we do better?

Build Binary Heap in O(n log n)

Ask VisuAlgo to build Binary (Max) Heap from an array in O(n log n) time by inserting each number one by one

In the screen shot below, the *partial state* of the O(n log n) Build Heap of the sample Binary (Max) Heap



BuildHeap, the Faster One

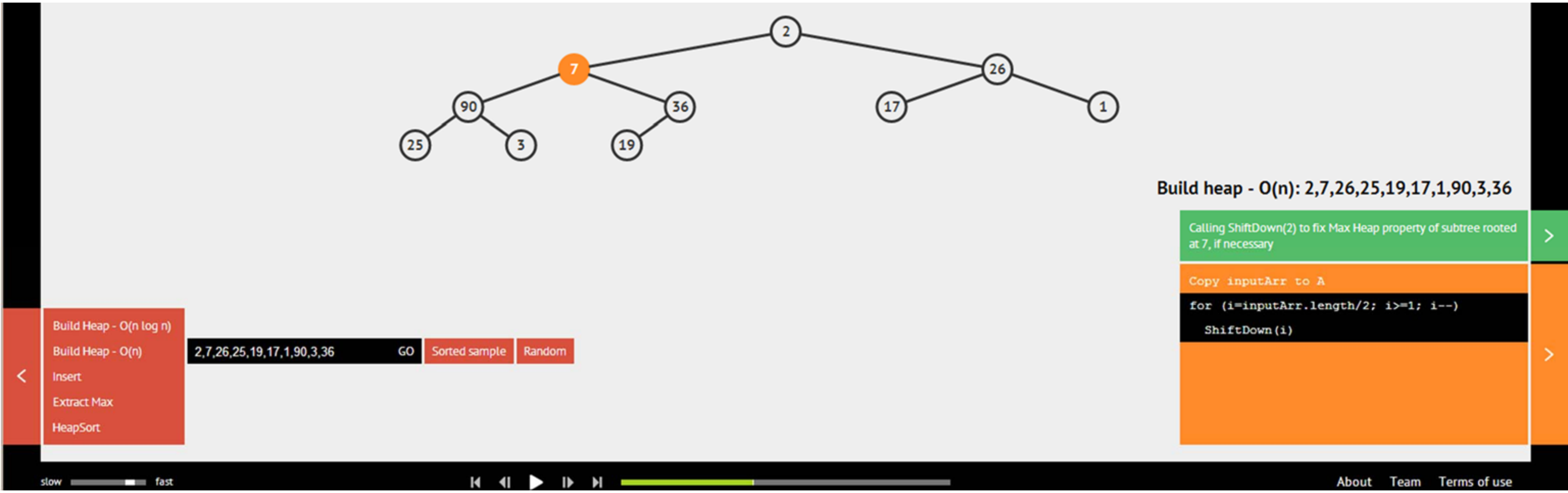
```
BuildHeap(array)
  heapsize    size(array)
  ? A[0]  ?   // dummy entry
       0    to heapsize // copy the content O(n)
  for i = 1 array[i-1]
  for i = parent(heapsize) down to 1 // O(n/2)
    A[i] ?
    ShiftDown(i) // O(log
    n)

// Analysis: Is this also O(n log n) ??
// No... soon, we will see that this is just O(n)
```

Build Binary Heap in O(n)

Ask VisuAlgo to build Binary (Max) Heap from an array in O(n) time by calling ShiftDown strategically

In the screen shot below, the *partial state* of the O(n) Build Heap of the sample Binary (Max) Heap

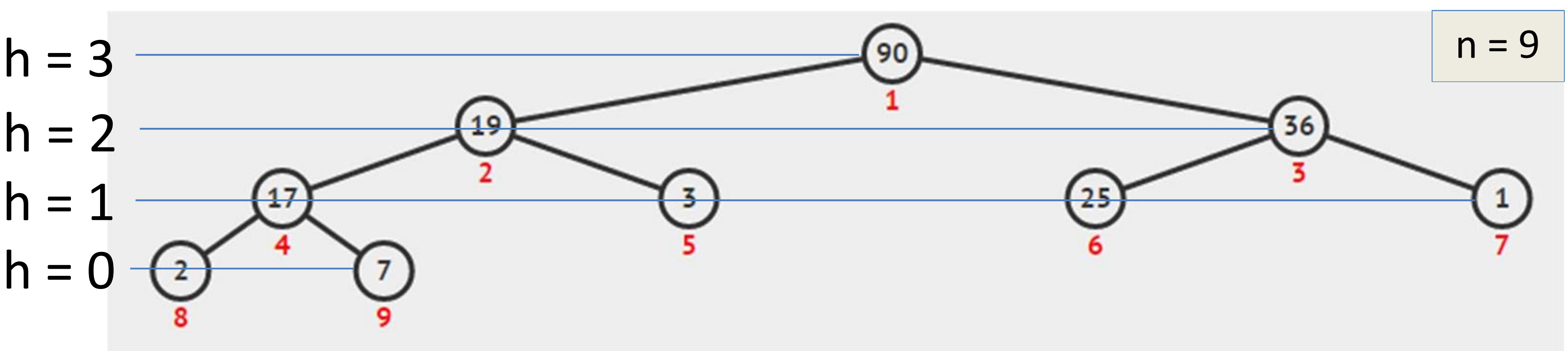


`BuildHeap()` Analysis... (1)

Recall: How many levels (height) are there in a complete binary tree (heap) of size **n**? __

Recall: What is the cost to run `shiftDown(i)`? ____

Q: How many nodes are there at height **h** of a full binary tree? ____



`BuildHeap()` Analysis... (2)

Cost of `BuildHeap()` is thus:

$$\underbrace{\sum_{h=0}^{\lfloor \lg(n) \rfloor}}_{\text{Sum over all levels}} \underbrace{\overbrace{\left\lceil \frac{n}{2^{h+1}} \right\rceil}^{\text{\# of nodes at height h}} \overbrace{O(h)}^{\text{Cost to Heapify a node at height h}}}_{\text{Cost for a level}} = \sum_{h=0}^{\lfloor \lg(n) \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil c*h = O\left(n \sum_{h=0}^{\lfloor \lg(n) \rfloor} \frac{h}{2^h}\right) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) = O(2n) = O(n)$$

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2} = 2$$

x = 1/2

$$\frac{0}{2^0} + \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \ldots = \frac{0}{1} + \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \ldots =$$

$$0 + 0.5 + 0.5 + 0.375 + 0.25 + 0.15625 + 0.09375 + \ldots < \mathbf{2}$$

HeapSort Analysis

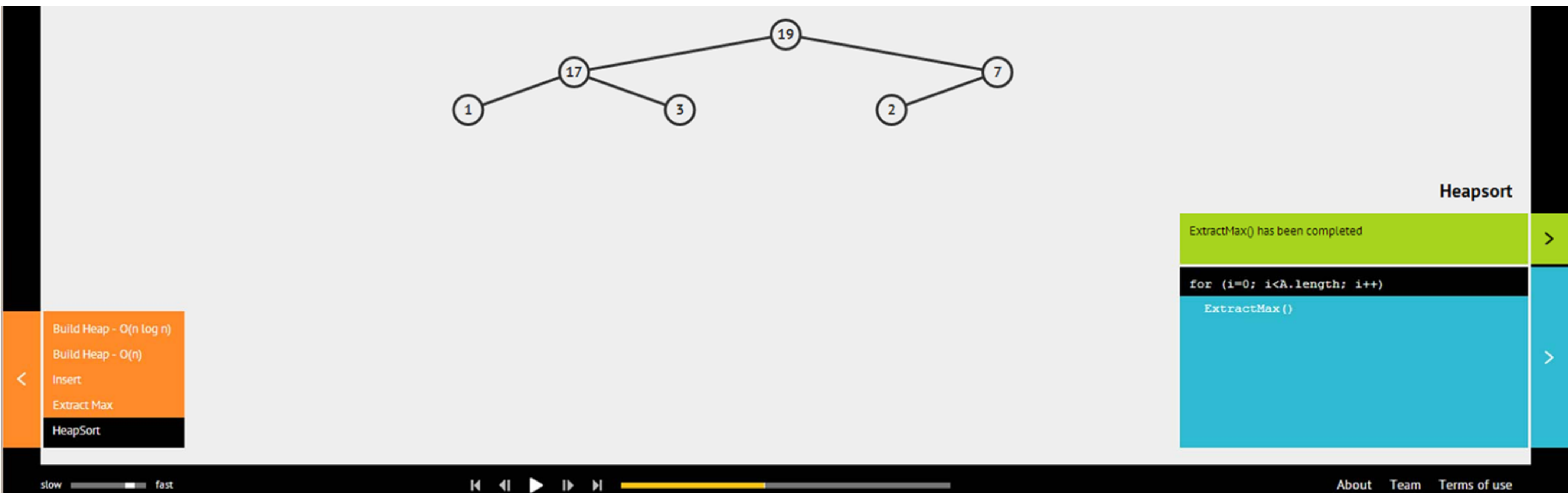
```
HeapSort(array)
  BuildHeap(array) // The best we can do is _____
  n ⍰  size(array)
  for i from 1 to n // O(n)
    A[n-i+1] ⍰ ExtractMax()      // O(log n)
  return A

// Analysis: Thus HeapSort runs in O(______)
// Do you notice that we do not need extra array
// like merge sort to perform sorting?
// Thus heap sort is more memory friendly.
// This is called "in-place sorting"
// But HeapSort is not "cache friendly"
//
```

Binary Heap: HeapSort()

Ask VisuAlgo to run HeapSort() on the sample Binary (Max) Heap

In the screen shot below, the *partial state* of the O(n log n) HeapSort() of the sample Binary (Max) Heap



Java Implementation

Priority Queue ADT

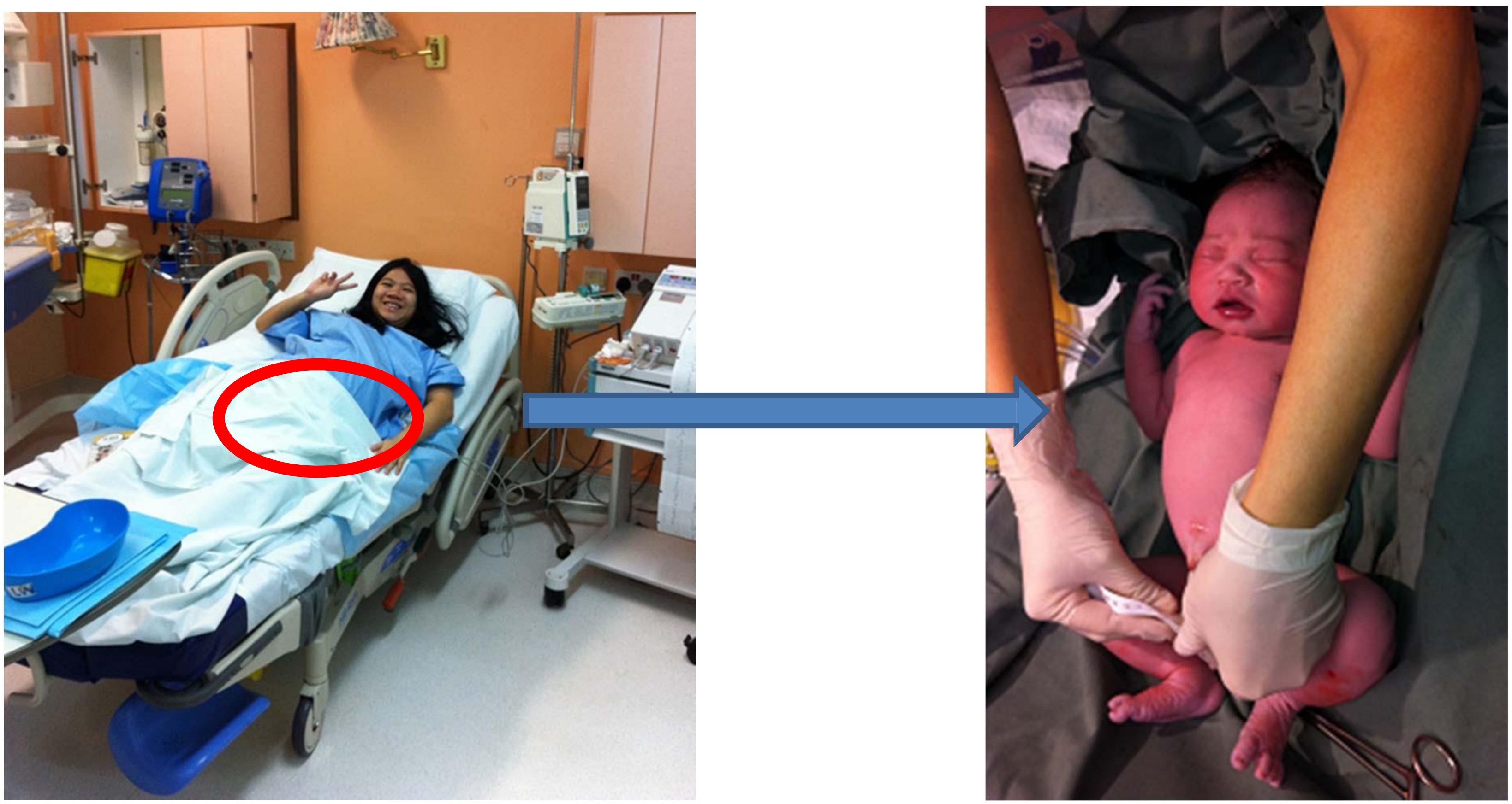
Heap Class (Java file given, you *can use* it for PS1)

- `ShiftUp(i)`
- `Insert(v)`
- `ShiftDown(i)`
- `ExtractMax()`
- `BuildHeapSlow(array)` and `BuildHeap(array)`
- `HeapSort()`

In OOP Style 😊

Scheduling Deliveries, v2015 (PS1)

This happens in the delivery suite (or surgery room for Caesarean section) of a hospital



PS1, the task

Given a list of ("insanely" many) pregnant women, prioritize the one who will give birth sooner over the one who will give birth later…

- Open on Wed, 19 Aug 2015, 11.45am, right after this lecture
- Clearly involving *some kind* of PriorityQueue 😊

PS1 Subtask A should be very easy

PS1 Subtask B may need Lab Demo 01 on Week 03

PS1 Subtask C is the challenge

- Introducing **UpdateKey** operation of a PriorityQueue

End of Lecture Quiz 😊

After Lecture 02, I will set a random test mode @ VisuAlgo to see if you understand Binary Heap

Go to:

http://visualgo.net**/test.html**

Use your CS2010 account to try the 5 Binary Heap questions (medium difficulty, 5 minutes)

Meanwhile, train first 😊

http://visualgo.net**/training.html**

Summary

In this lecture, we have looked at:

- Heap DS and its application as efficient PriorityQueue
- Storing heap as a compact array and its operations
  - Remember how we always try to maintain complete binary tree and heap property in all our operations!
- Building a heap from a set of numbers in O(**n**) time
- Simple application of Heap DS: O(**n** log **n**) HeapSort

We will use PriorityQueue in the 2^{nd} part of CS2010

- If some concepts are still unclear, ask your personal tutor: [http://visualgo.net**/heap.html**](http://visualgo.net/heap.html)