Today's utter madness

- Reviewing for the midterm
  - *This week: study list*
  - *Next week: Q&A session*
- Tree stuff (leftovers from last week)
- Transform-and-conquer algorithms

Study list, part 1

- Given a description (in English) of a problem:
  - *Be able to write pseudocode for an algorithm*
- Given an algorithm in code/pseudocode, be able to:
  - *Identify the basic operation*
  - *Write a summation formula that expresses the running time*
  - *Give the efficiency class of the algorithm (big-O notation)*
  - *Trace the code/pseudocode for a given input*
  - *Describe in English the purpose of the algorithm*

Study list, part 2

- For the categories of algorithms we've studied (bf, dec&c, div&c, xf&c):
  - *Describe the general characteristics of the category*
  - *Give an example of an algorithm*
- For all of the specific problems/algorithms we've examined:
  - *Be able to trace the code/pseudocode*
  - *Know the efficiency class*
  - *Identify as bf/div&c/dec&c/xf&c*
- For sorting algorithms:
  - *How to perform each one*

A LITTLE TREE STUFF FROM LAST WEEK

TRANSFORM AND CONQUER

(Chapter 6)

Transform and Conquer

- This technique solves a problem by a transformation to:
  - *a more convenient instance of the same problem (aka* ***instance simplification****)*
  - *a different representation of the same instance (aka* ***representation change****)*

Transform and Conquer examples

- **Instance simplification (pre-sorting)**
  - *Checking element uniqueness in an array*
  - *Computing the mode*

- **Representation change**
  - *Heap*
    - *Implementation*
    - *Insert and Delete*
    - *Construction*
  - *Heap sort*

ELEMENT UNIQUENESS IN AN ARRAY

Example: Element uniqueness in an array

- Problem: Determine if all elements in an array are distinct

- Brute force algorithm
  - *Compare all pairs of elements*
  - *Efficiency: O(n^2)*

- Instance simplification (presorting)
  - *Stage 1: sort by efficient sorting algorithm (e.g. mergesort)*
  - *Stage 2: scan array to check pairs of adjacent elements*
  - *Efficiency: O(nlogn) + O(n) = O(nlogn)*

Example: Element uniqueness in an array

ALGORITHM *PresortElementUniqueness*$(A[0..n-1])$

//Solves the element uniqueness problem by sorting the array first
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Returns "true" if A has no equal elements, "false" otherwise
sort the array A
for $i \leftarrow 0$ **to** $n-2$ **do**
    if $A[i] = A[i+1]$ **return false**
return true

COMPUTING A MODE

Computing a mode

- The *mode* is the value that occurs most often in a given list of numbers.

5	1	6	7	6	5	7	6

Mode: 6

Computing a mode

- Brute Force:
  - *Scan the list*
  - *Compute the frequencies of all distinct values*
  - *Find the value with the largest frequency*

5	1	6	7	6	5	7	6

Computing a mode

- Brute Force:

5	1	6	7	6	5	7	6

i

Data	5
Frequency	1

Computing a mode

- Brute Force:

5	1	6	7	6	5	7	6

i

Data	5	1
Frequency	1	1

Computing a mode

- Brute Force:

5	1	6	7	6	5	7	6

i

Data	5	1	6
Frequency	1	1	1

Computing a mode

- Brute Force:

5	1	6	7	6	5	7	6

i

Data	5	1	6	7
Frequency	1	1	1	1

Computing a mode

- Brute Force:

5	1	6	7	6	5	7	6

i

Data	5	1	6	7
Frequency	1	1	2	1

Computing a mode

- Brute Force:

5	1	6	7	6	5	7	6

i

Data	5	1	6	7
Frequency	2	1	2	1

Computing a mode

- Brute Force:

5	1	6	7	6	5	7	6

i

Data	5	1	6	7
Frequency	2	1	2	2

Computing a mode

- Brute Force:

5	1	6	7	6	5	7	6

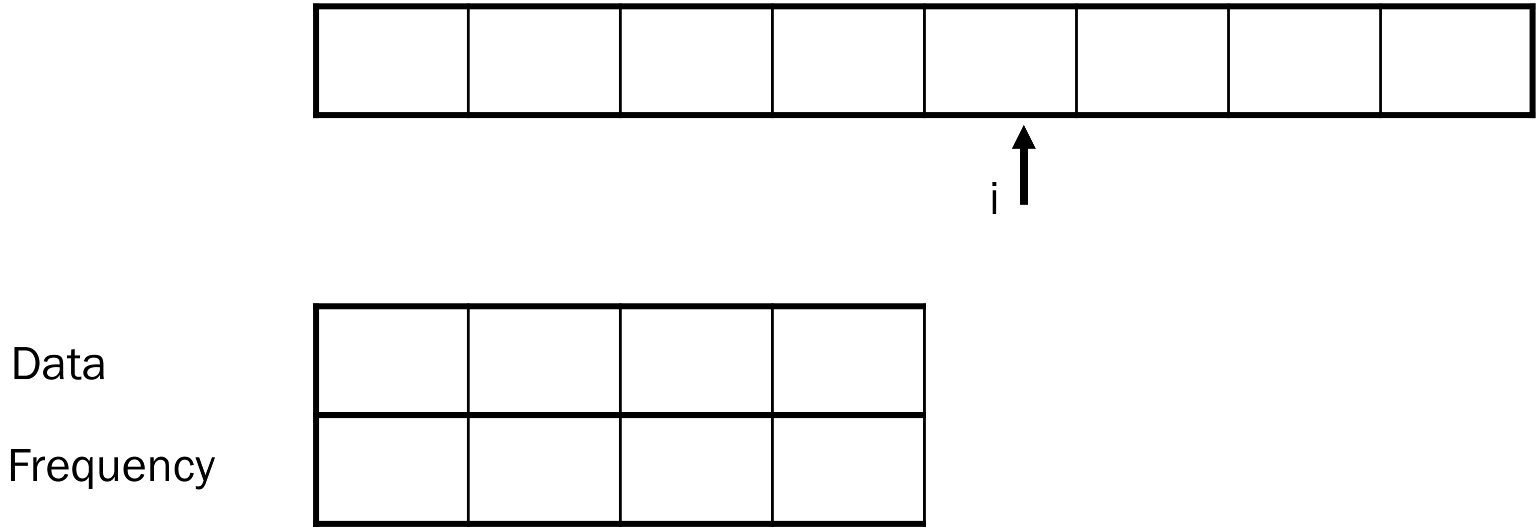
i

Data	5	1	6	7
Frequency	2	1	3	2

Max

Computing a mode

- Efficiency (worst-case):
  - *A list with no equal elements*
  - *i^{th} element is compared with $i - 1$ elements in the "Data" array*



Computing a mode

- Efficiency (worst-case):
  - *Creating auxiliary list ("Data" array):* $0 + 1 + 2 + \cdots + n - 1 = O(n^2)$
  - *Finding max: O(n)*
  - *Efficiency (worst-case):* $O(n^2)$

Computing a mode (pre-sorting)

- Sort the input
- All equal values will be adjacent to each other
- Find the longest run of adjacent equal values in the sorted array

Computing a mode (pre-sorting)

```
ALGORITHM PresortMode(A[0..n − 1])
    //Computes the mode of an array by sorting it first
    //Input: An array A[0..n − 1] of orderable elements
    //Output: The array's mode
    sort the array A
    i ← 0                      //current run begins at position i
    modefrequency ← 0          //highest frequency seen so far
    while i ≤ n − 1 do
        runlength ← 1;  runvalue ← A[i]
        while i + runlength ≤ n − 1 and A[i + runlength] = runvalue
            runlength ← runlength + 1
        if runlength > modefrequency
            modefrequency ← runlength;  modevalue ← runvalue
        i ← i + runlength
    return modevalue
```

Computing a mode (pre-sorting)

- Efficiency:

$$
\begin{aligned}
T(n) \quad &= T_{sort}(n) + T_{scan}(n) \\
&= (n \log n) + (n) \\
&= (n \log n)
\end{aligned}
$$

SEARCHING WITH PRESORTING

Searching with presorting

- Problem: Search for a given key K in an array A[0..n-1]

- Presorting-based algorithm:
  - *Stage 1 Sort the array by an efficient sorting algorithm*
  - *Stage 2 Apply binary search*

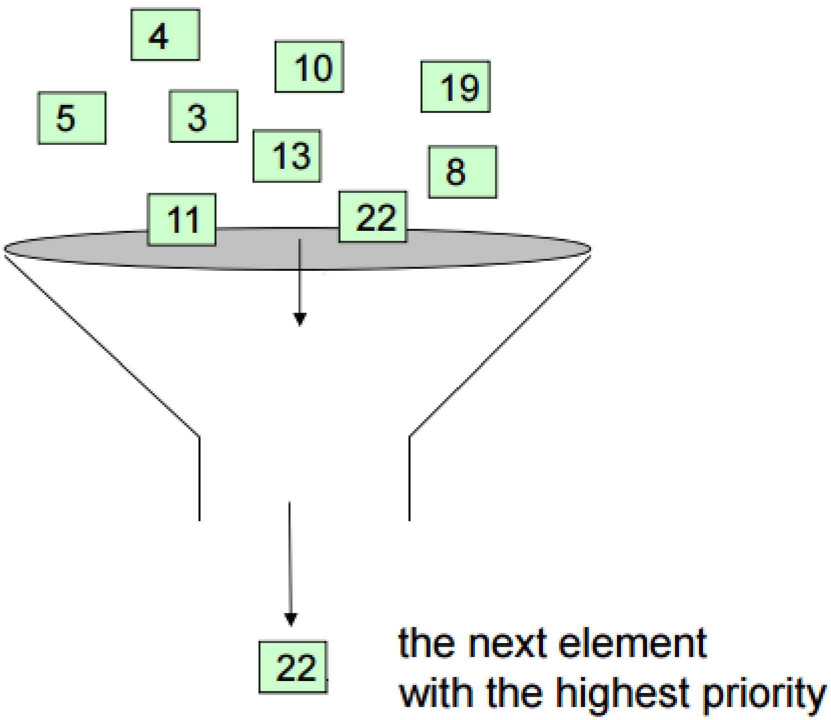
- Efficiency: O(n log n) + O(log n) = O(n log n)

- Good or bad? (Note that sequential search is O(n))
- Why do we have our dictionaries, telephone directories, etc. sorted?

REPRESENTATION CHANGE: HEAPS AND HEAPSORT

Sample problem

- You're running a hospital
- Patients are coming in with different priority



Simple implementations

- Arraylist
  - *Insert: O(1)*
  - *deleteMax: O(n)*

7	5	8	1	9

- SortedArraylist
  - *Insert: O(logn + n)*
  - *deleteMax: O(n)*

9	8	7	5	1

Representation change

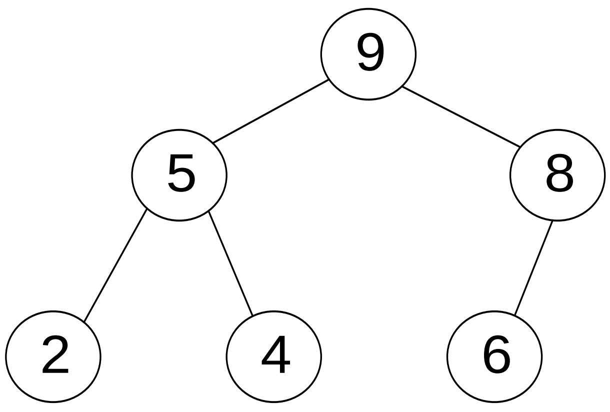
- Idea:
  - *Given an array*
  - *Transform to a new data structure*

    (Make a “**heap**” out of it)

- Efficiency of heap:
  - *Insert an item: O(logn)*
  - *Delete an item with max priority: O(logn)*

Heap definition

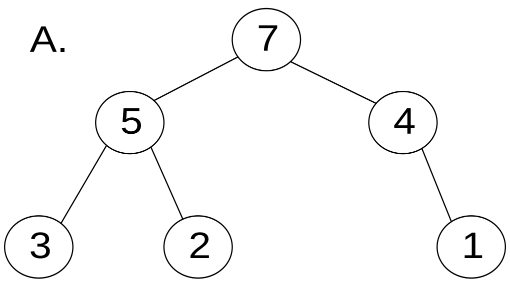
- Almost complete binary tree
  - *filled on all levels, except last, where filled from left to right*
- Every parent is greater than (or equal to) child



Heap or No Heap?

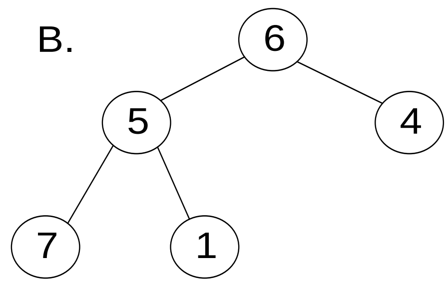
NO

A.



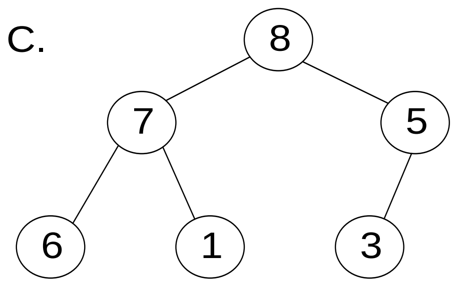
NO

B.



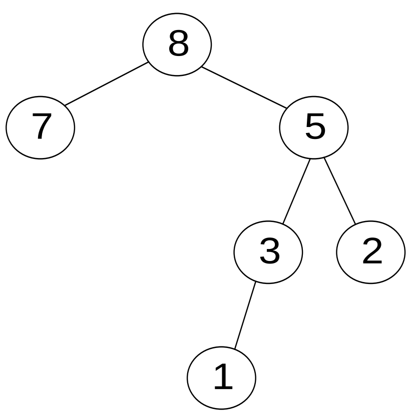
YES

C.



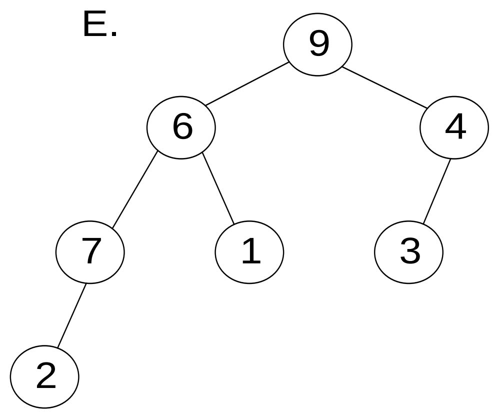
NO

D.



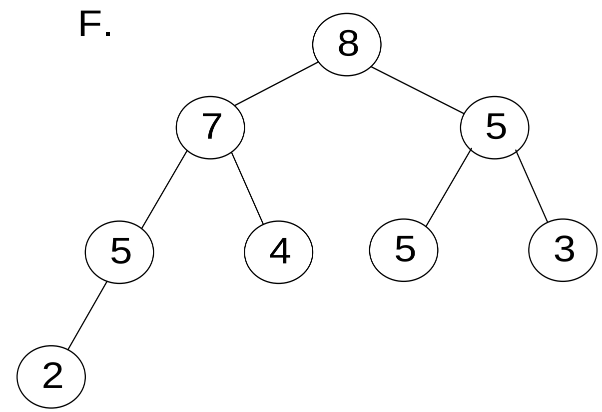
NO

E.



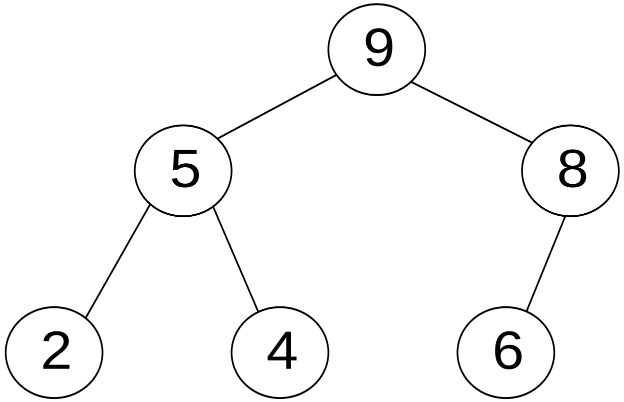
YES

F.



Heap properties

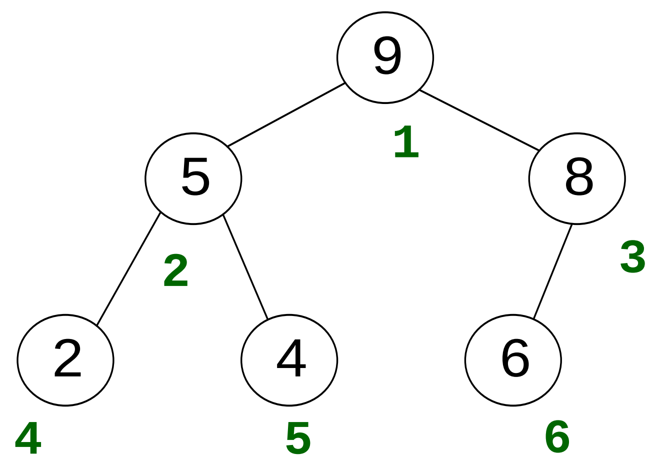
- Max element is in root
- Heap with N elements has height = $\lfloor \log_2 N \rfloor$



N = 6
Height = 2

Heap implementation

- Use an array: no need for explicit parent or child pointers.
  - `Parent(i) = ⌊i/2⌋`
  - `Left(i)   = 2i`
  - `Right(i)  = 2i + 1`

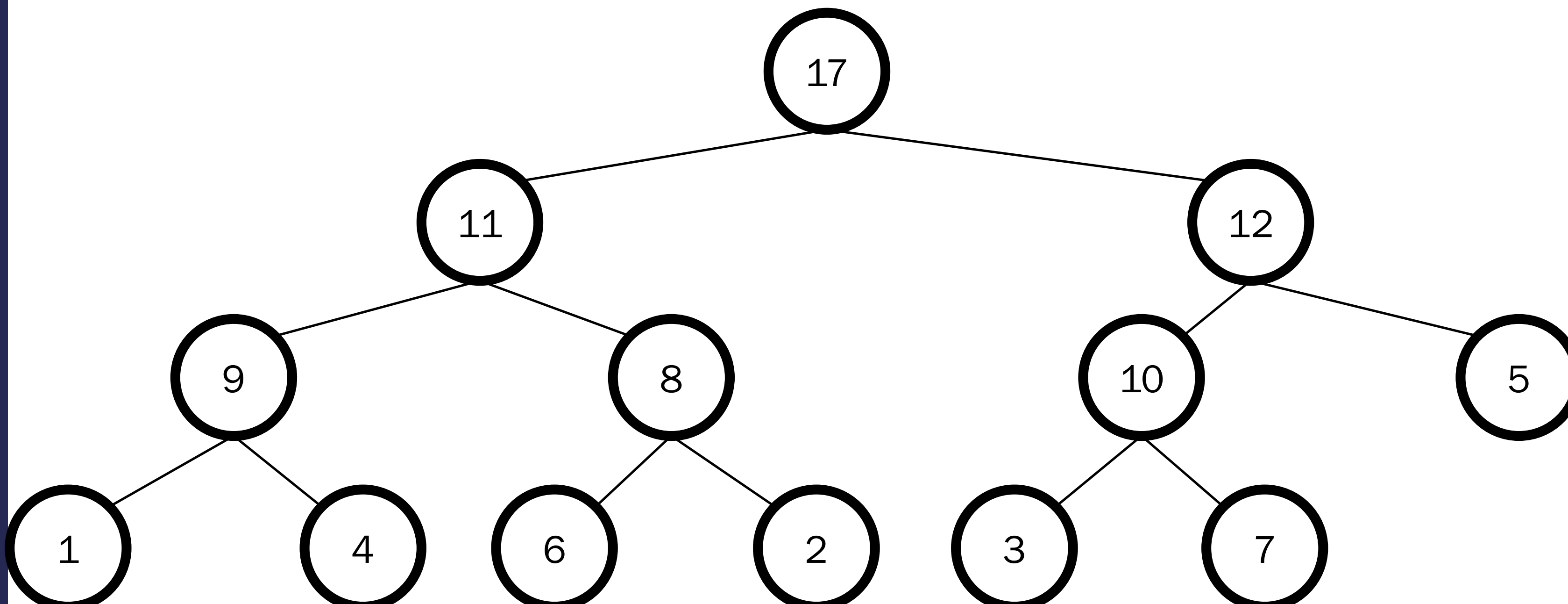


0	1	2	3	4	5	6
	9	5	8	2	4	6

Example 1

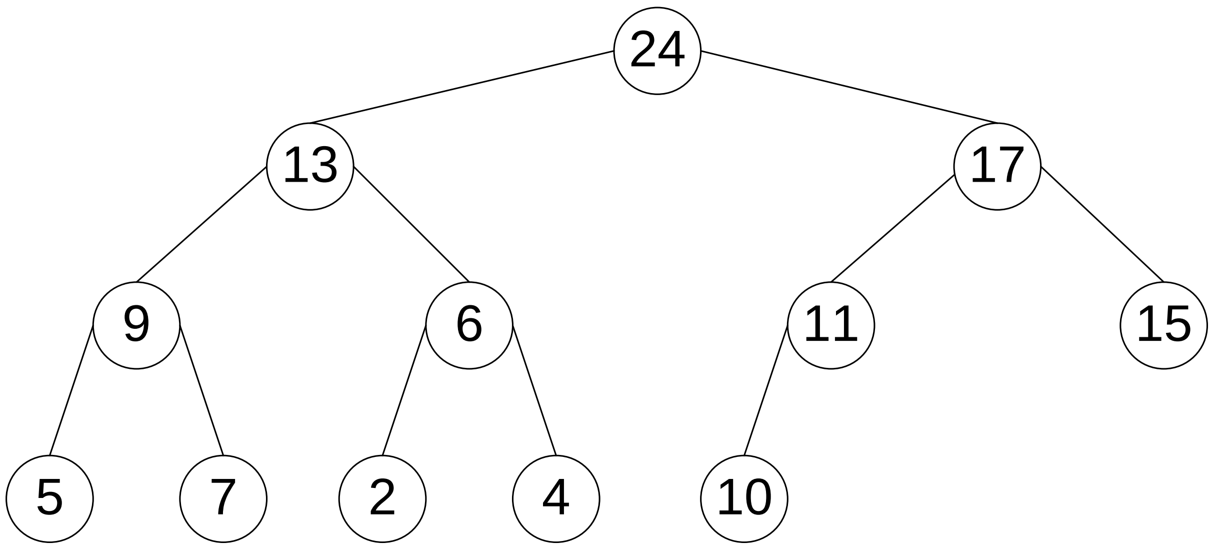
- Draw the tree representation of this heap

Index	1	2	3	4	5	6	7	8	9	10	11	12	13
value	17	11	12	9	8	10	5	1	4	6	2	3	7



Example 2

- Draw the array representation of this heap



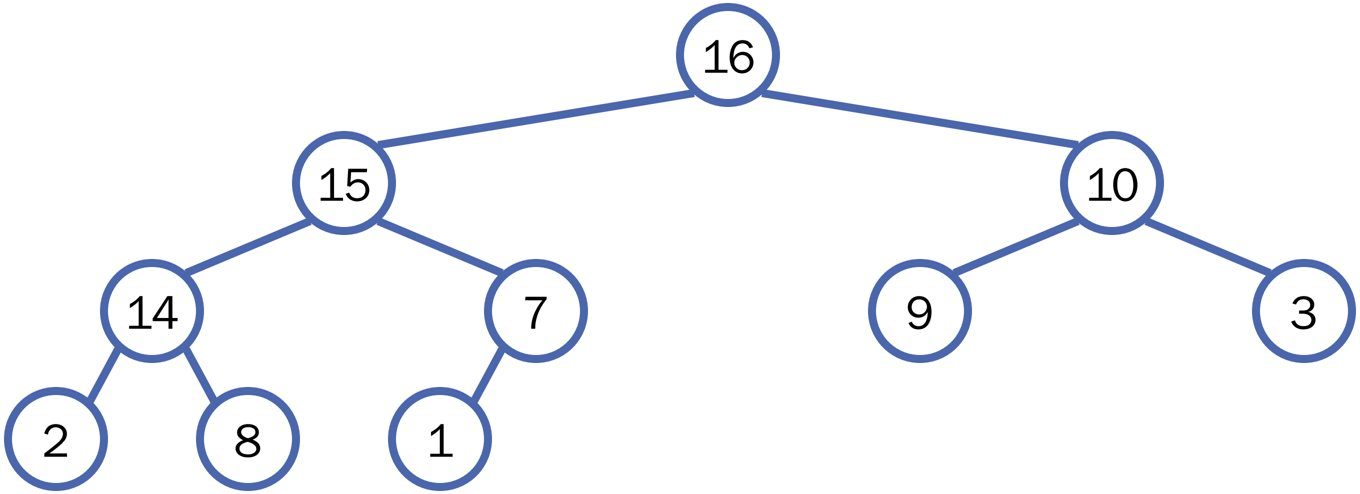
Index	1	2	3	4	5	6	7	8	9	10	11	12
value	24	13	17	9	6	11	15	5	7	2	4	10

Heap insertion

- Insert into next available slot
- Bubble up until it's heap ordered (heapify)

Insert to heap example

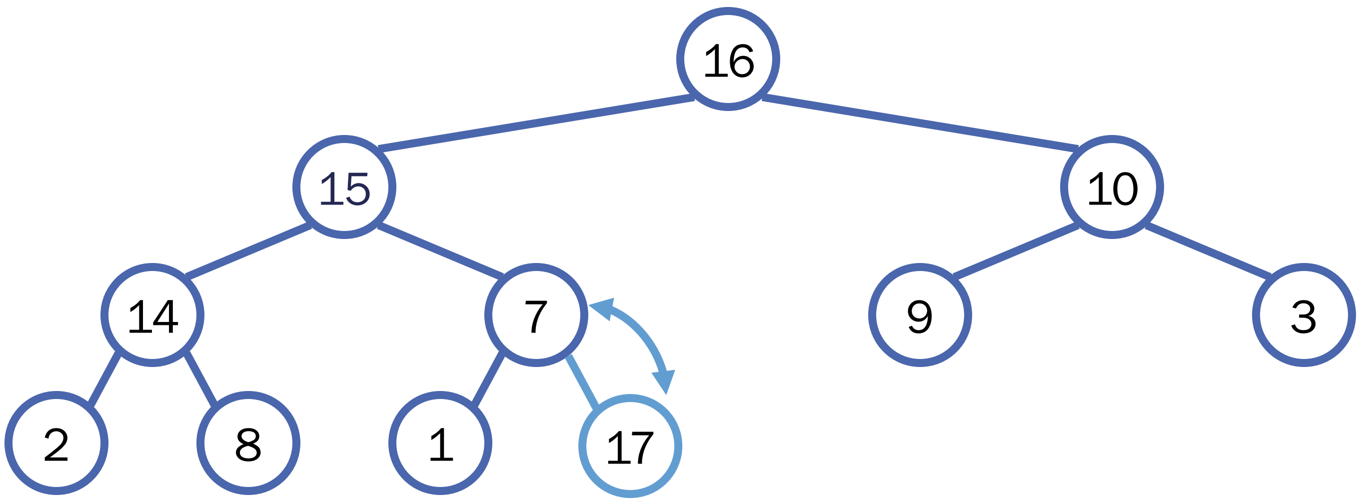
- Insert 17



A =

16	15	10	14	7	9	3	2	8	1

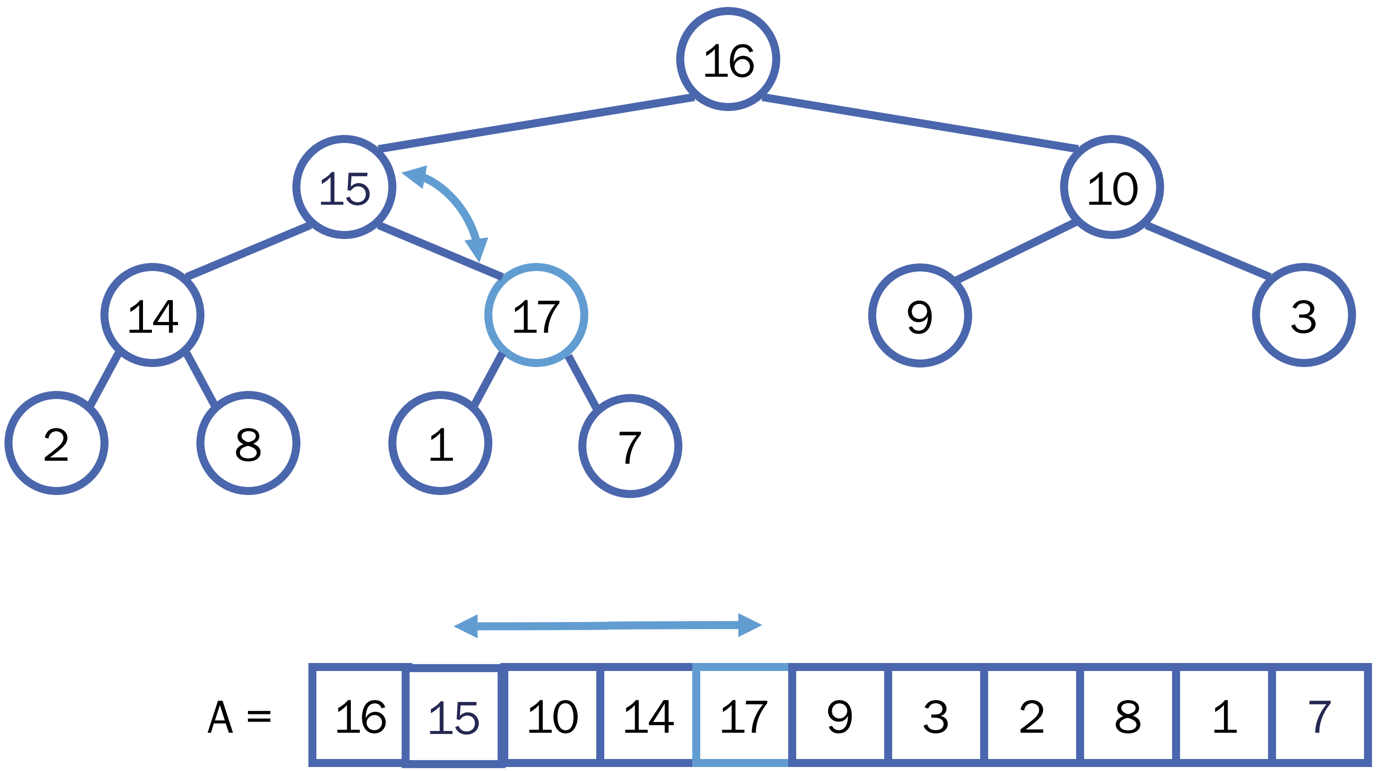
Insert to heap example



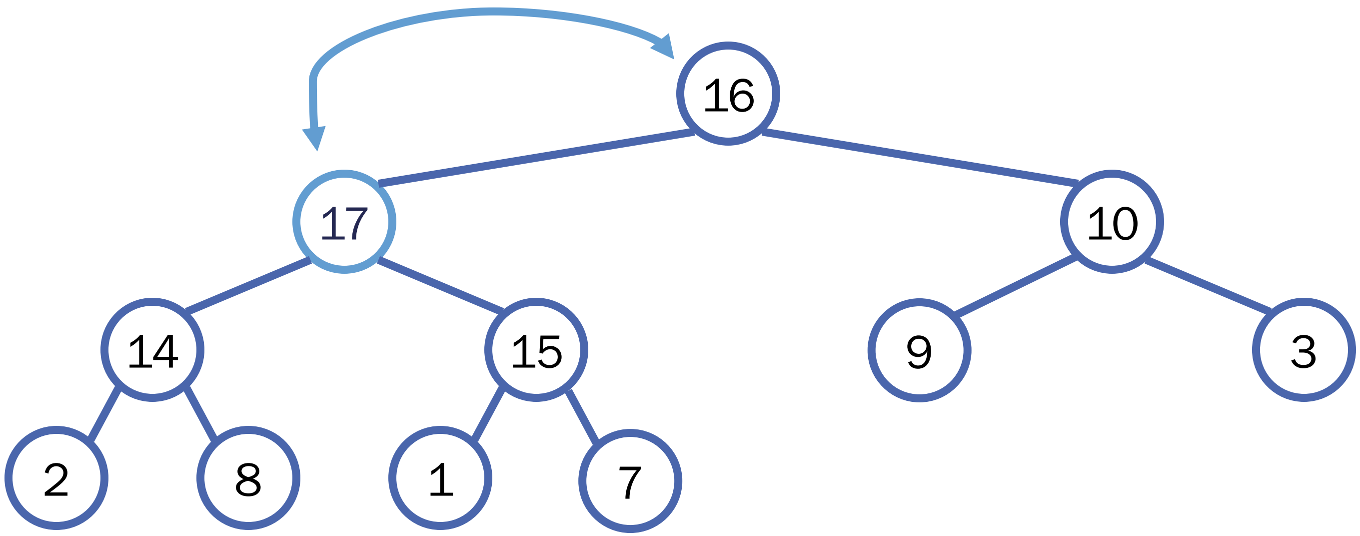
A =

16	15	10	14	7	9	3	2	8	1	17

Insert to heap example

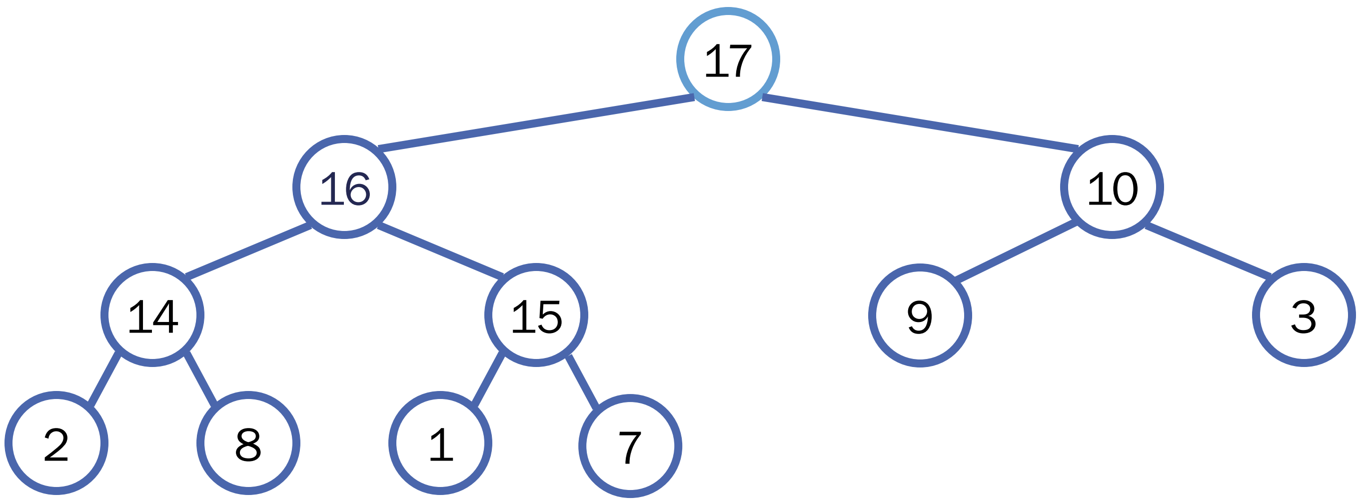


Insert to heap example



A =	16	17	10	14	15	9	3	2	8	1	7

Insert to heap example

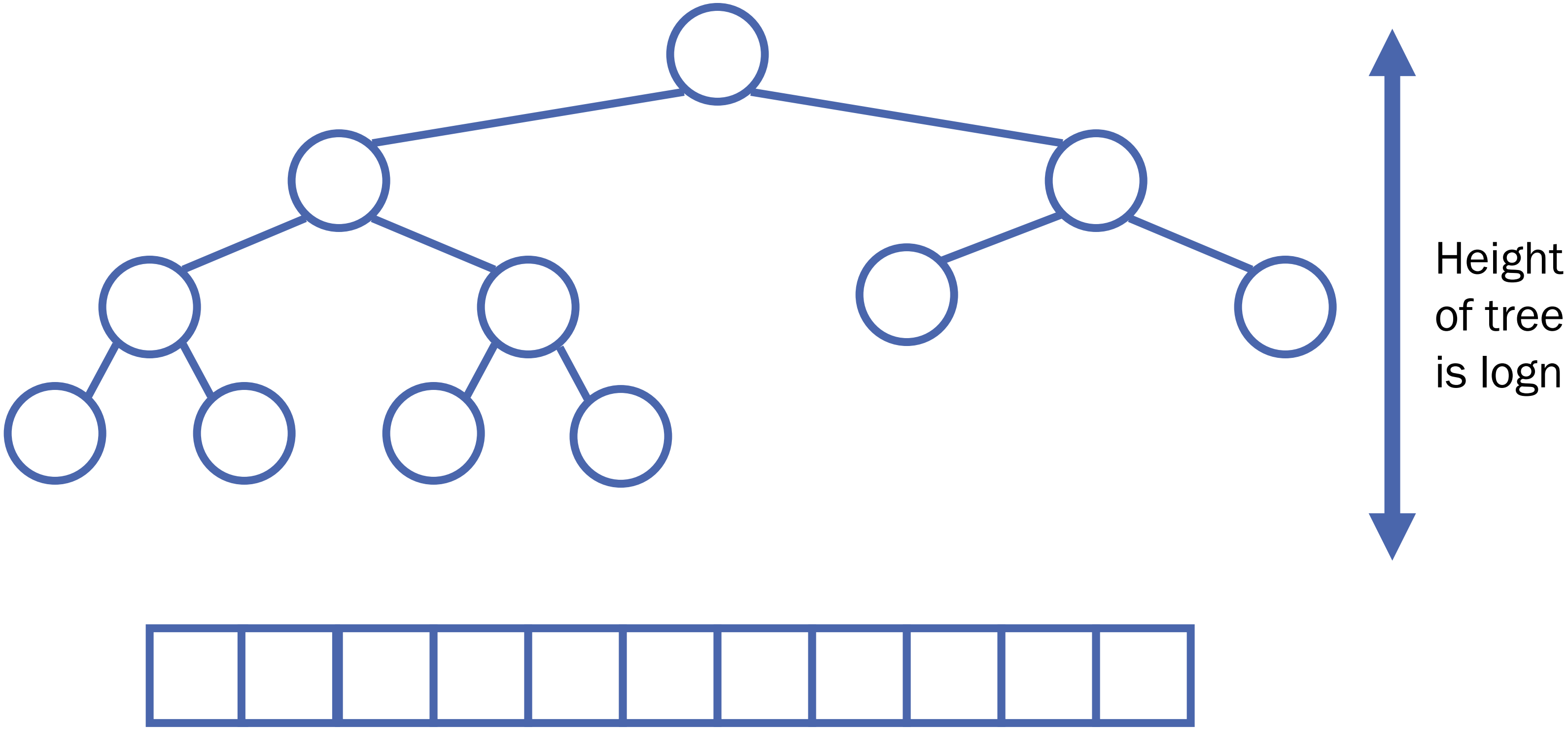


A =

17	16	10	14	15	9	3	2	8	1	7

Insert to heap example

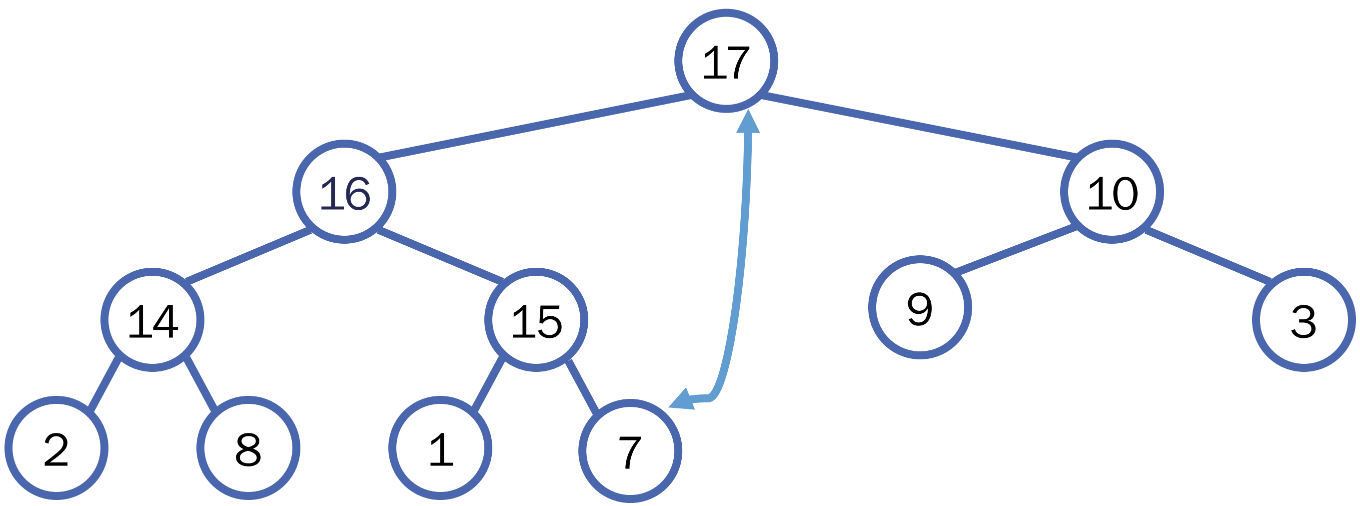
- Efficiency is O(log n)



Delete max from heap

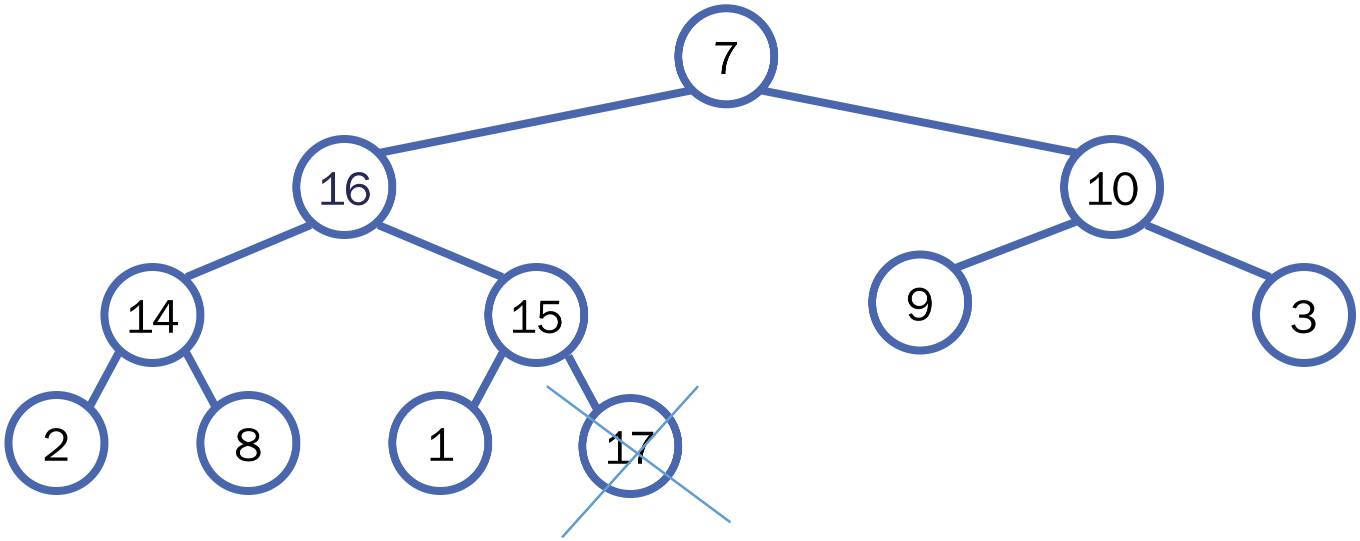
- Exchange root with rightmost leaf
- Delete element
- Bubble root down until it's heap ordered

Delete max from heap
Example



A =										
17	16	10	14	15	9	3	2	8	1	7

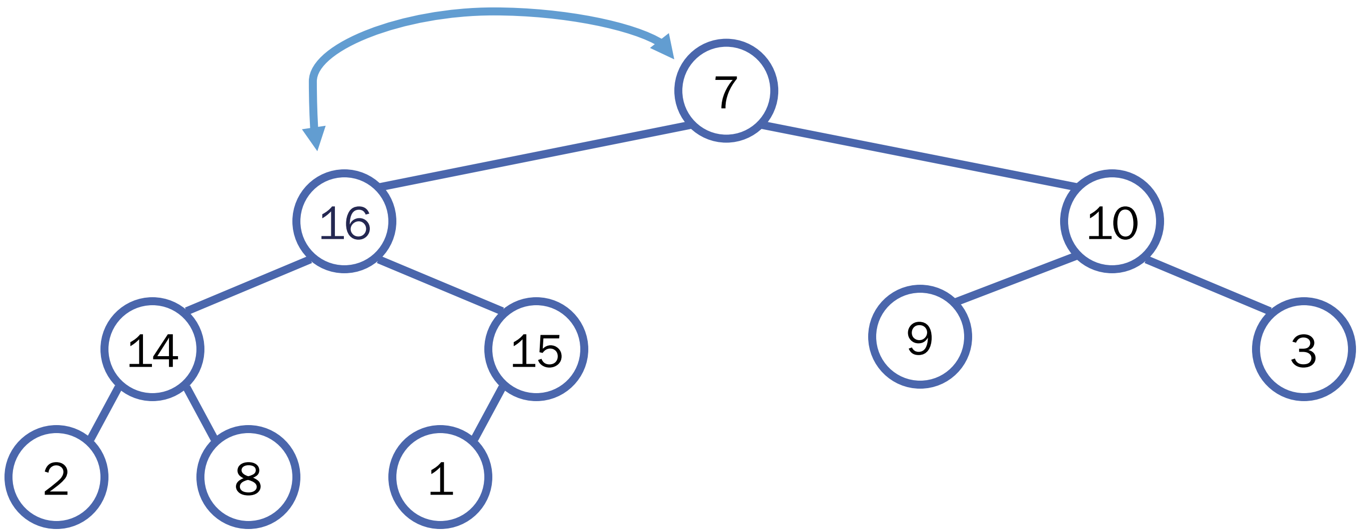
Delete max from heap Example



A =

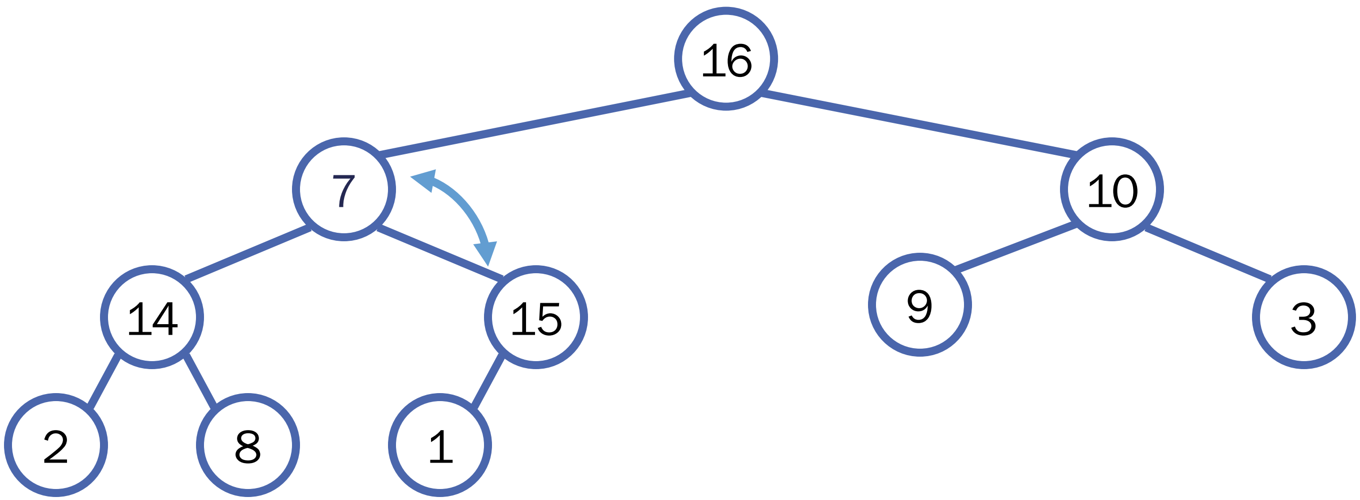
7	16	10	14	15	9	3	2	8	1	~~17~~

Delete max from heap Example



A =									
7	16	10	14	15	9	3	2	8	1

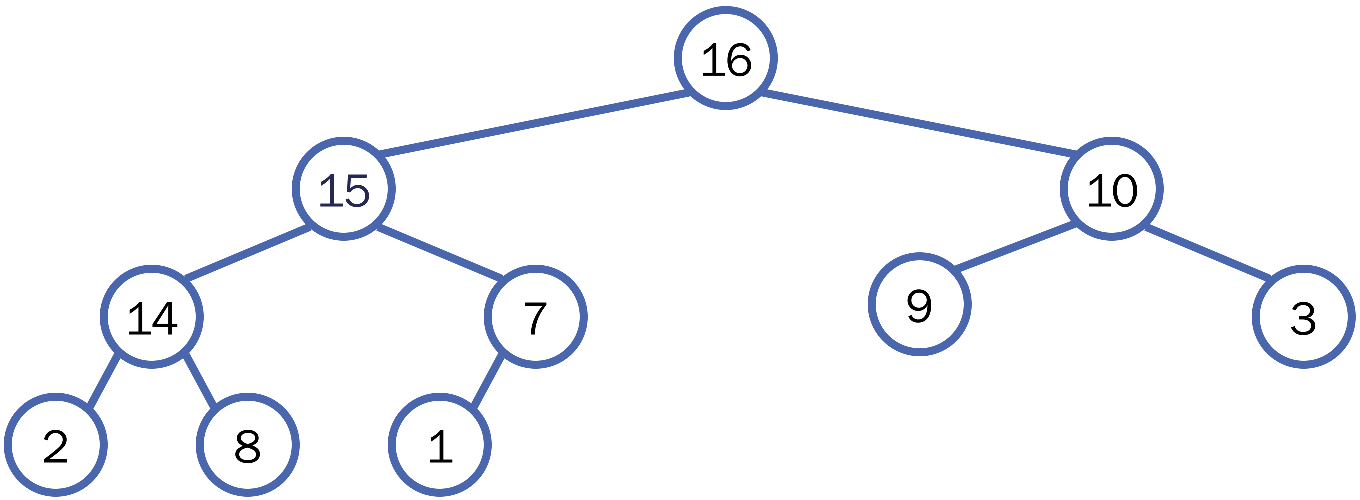
Delete max from heap Example



A =

16	7	10	14	15	9	3	2	8	1

Delete max from heap Example

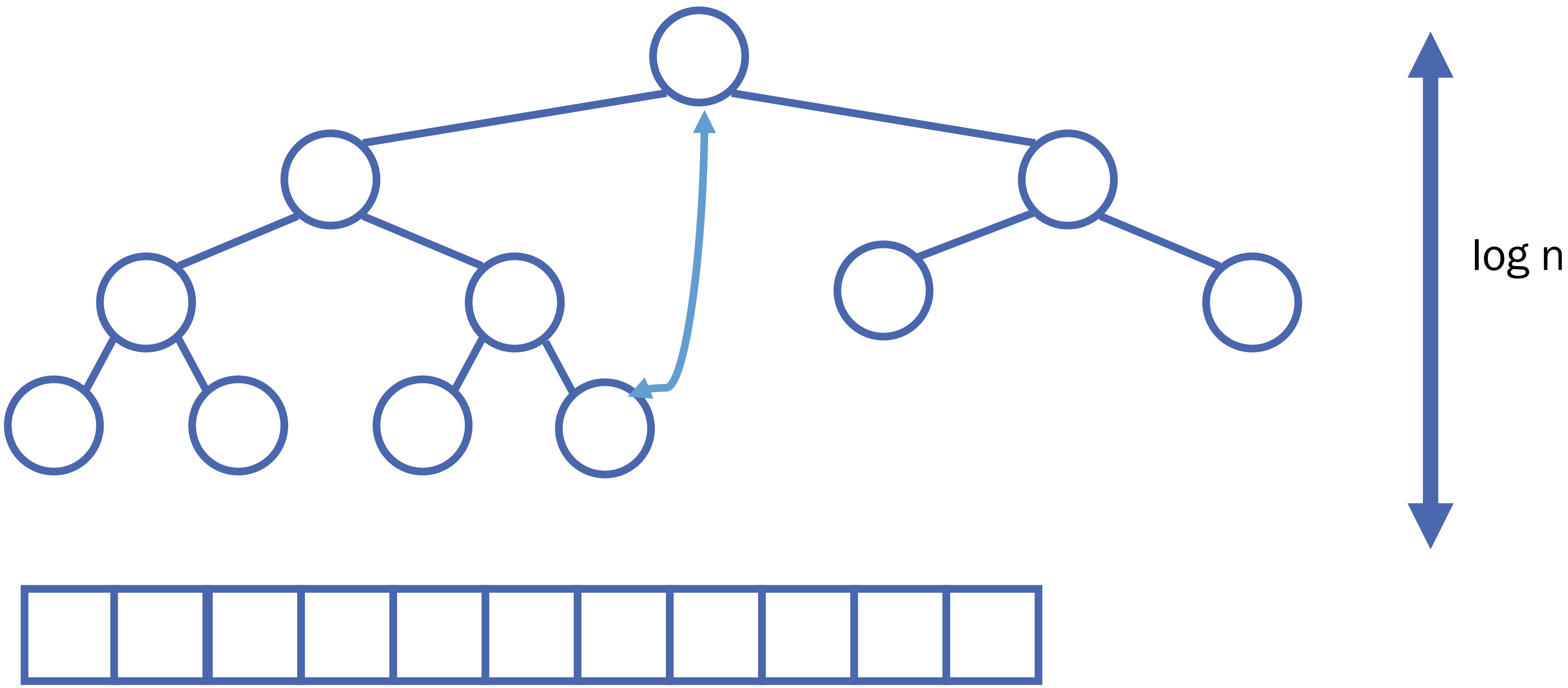


A =

16	15	10	14	7	9	3	2	8	1

Delete from heap Example

- Efficiency is O(log n)

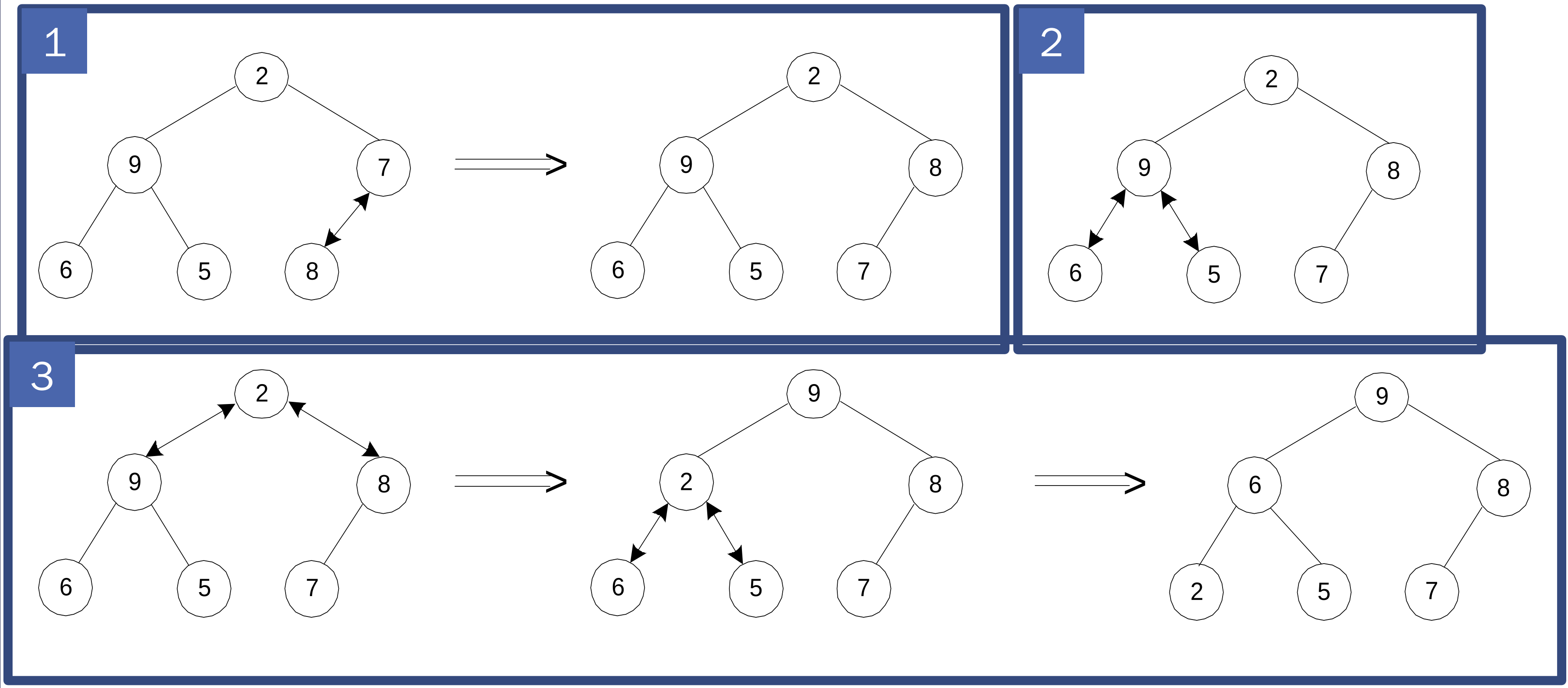


Heap Construction

- Step 0: Initialize the structure with keys in the order given
- Step 1: Starting with the last (rightmost) parental node, fix the heap rooted at it, if it doesn't satisfy the heap condition: keep exchanging it with its largest child until the heap condition holds
- Step 2: Repeat Step 1 for the preceding parental node

Example of heap construction

Construct a heap for the list 2, 9, 7, 6, 5, 8



HEAPSORT

HeapSort

- How can we use a Heap to sort an arbitrary array?
  - *Stage 1: Transform the array into a heap (Construct a heap)*
  - *Stage 2: Call RemoveMax to get all array elements in sorted order*

Analysis of Heapsort

- Stage 1: Build heap for a given list of n keys
  - *O(nlogn)*

- Stage 2: Repeat operation of root removal n-1 times (fix heap)
  - *O(nlogn)*

Try it/ homework

- Chapter 6.1, page 205, questions 2, 3, 7
- Chapter 6.4, page 233, question 1,2,7