Priority queues

Min Priority Queue

- Collection of elements.
- Each element has a key or priority.
- Operations:
  - insert an element into the priority queue
  - get the element with min priority
  - remove the element with min priority

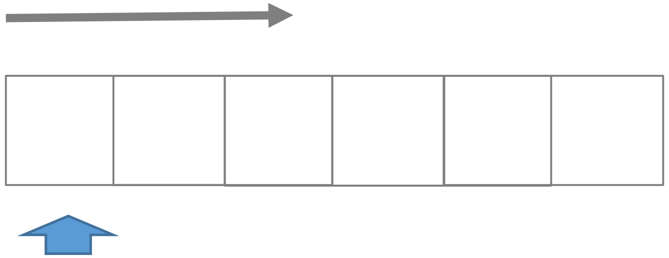
Max Priority Queue

- Collection of elements.
- Each element has a key or priority.
- Operations:
  - insert an element into the priority queue
  - get element with max priority
  - remove element with max priority

Sorting elements in ascending order

Use a min priority queue

- use element key as priority
- insert elements into a priority queue
- remove/pop elements in priority order

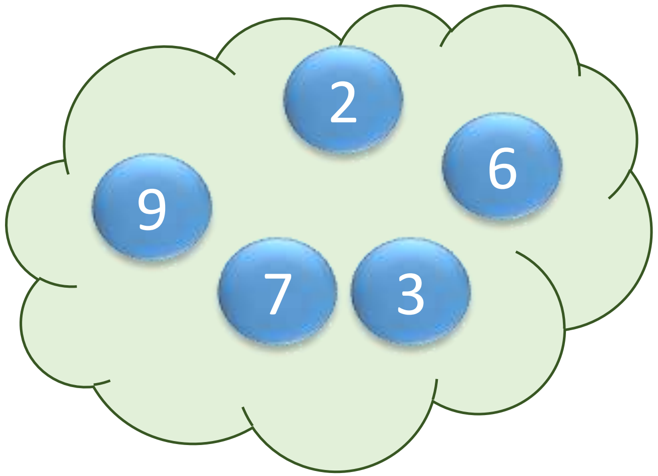


Sorting elements in ascending order

Use a min priority queue

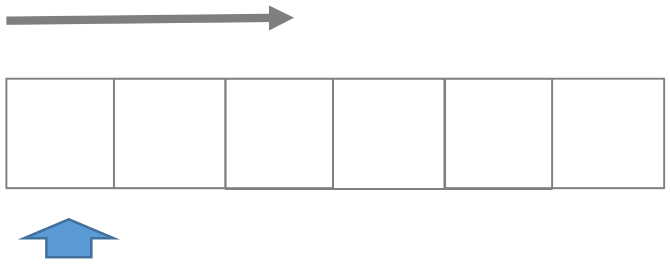
- use element key as priority
- insert elements into a priority queue
- remove/pop elements in priority order

min
key



Min priority queue

Sorted
array



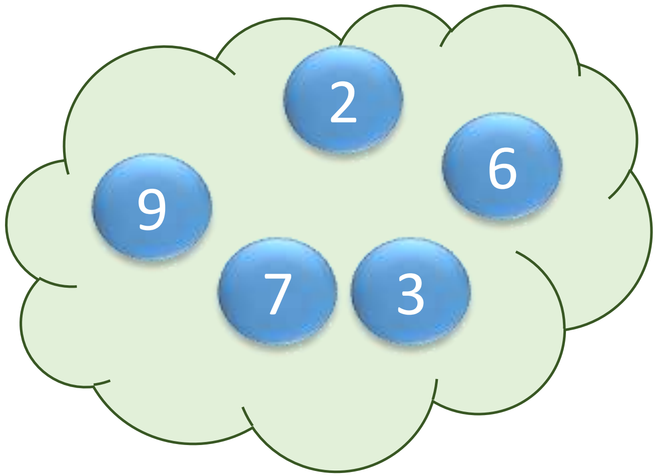
index

Sorting elements in ascending order

Use a min priority queue

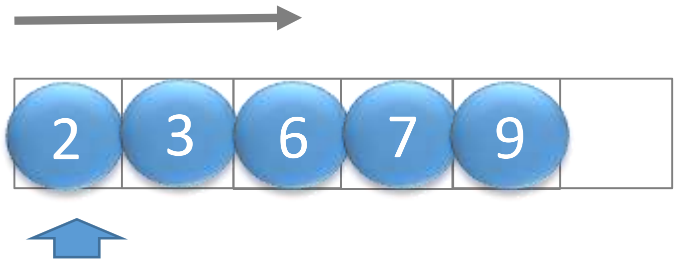
- use element key as priority
- insert elements into a priority queue
- remove/pop elements in priority order

min
key



Min priority queue

Sorted
array

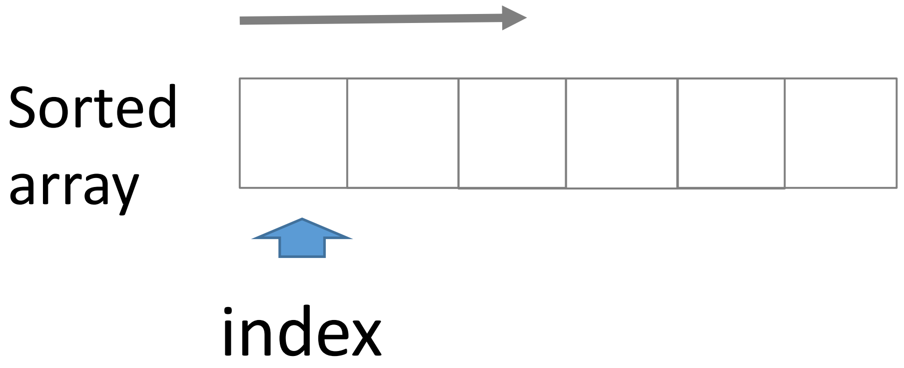


index

Sorting elements in descending order

Use a max priority queue

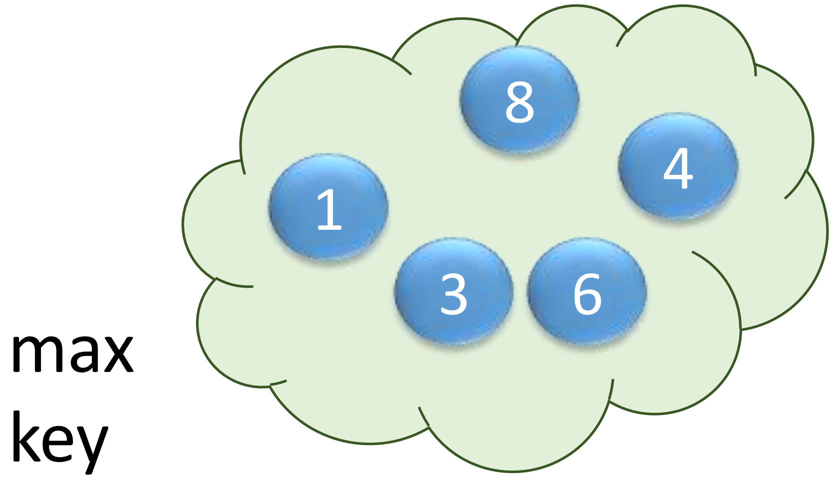
- use element key as priority
- insert elements into a priority queue
- remove/pop elements in priority order



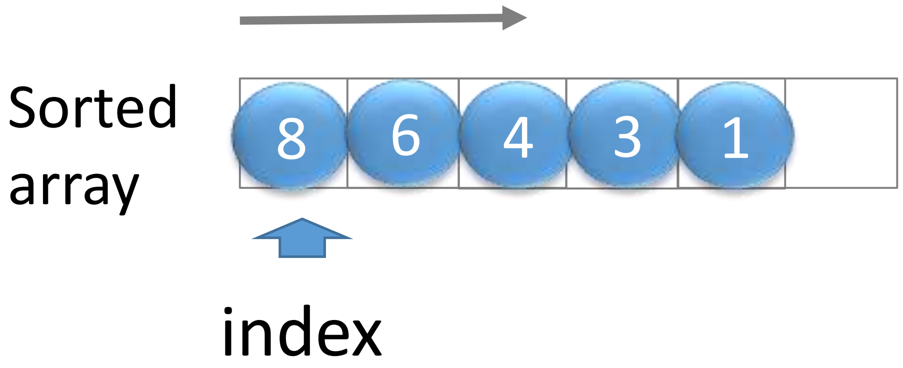
Sorting elements in descending order

Use a max priority queue

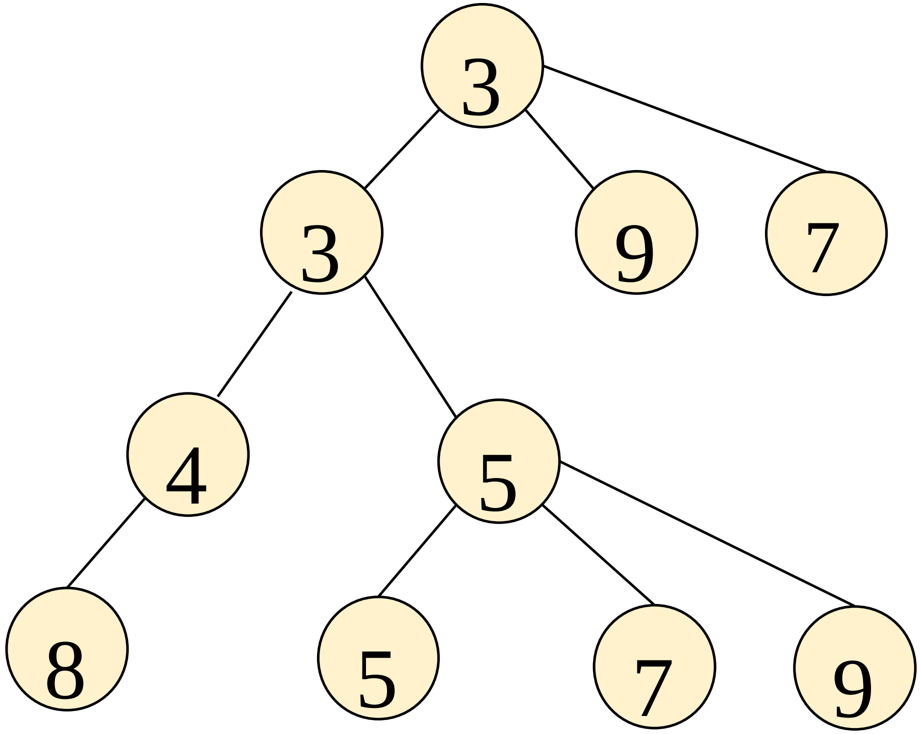
- use element key as priority
- insert elements into a priority queue
- remove/pop elements in priority order



Max priority queue

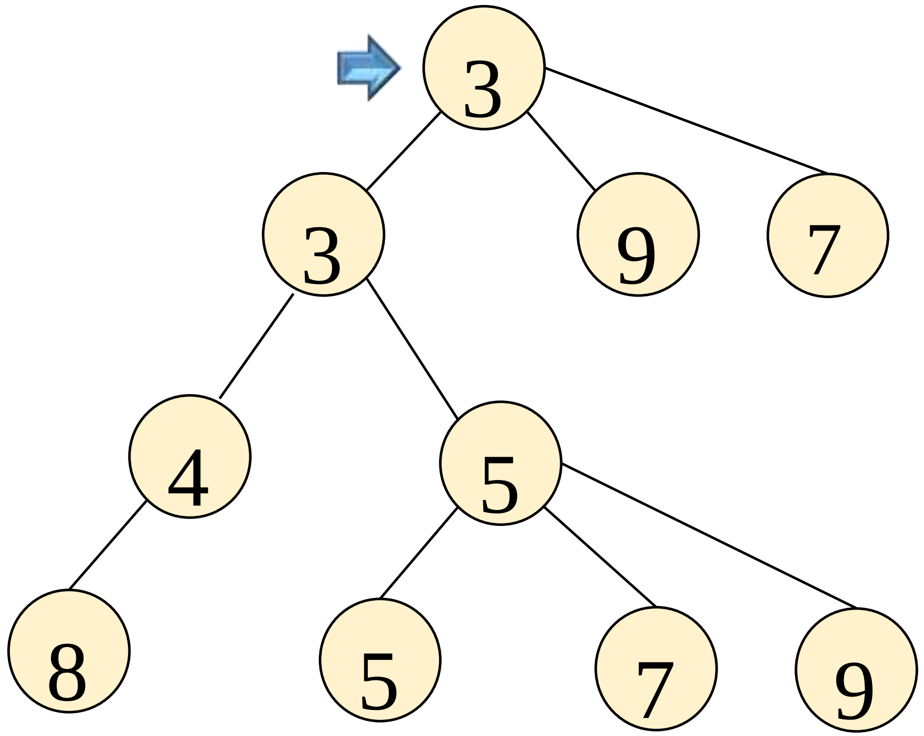


Min Tree Definition



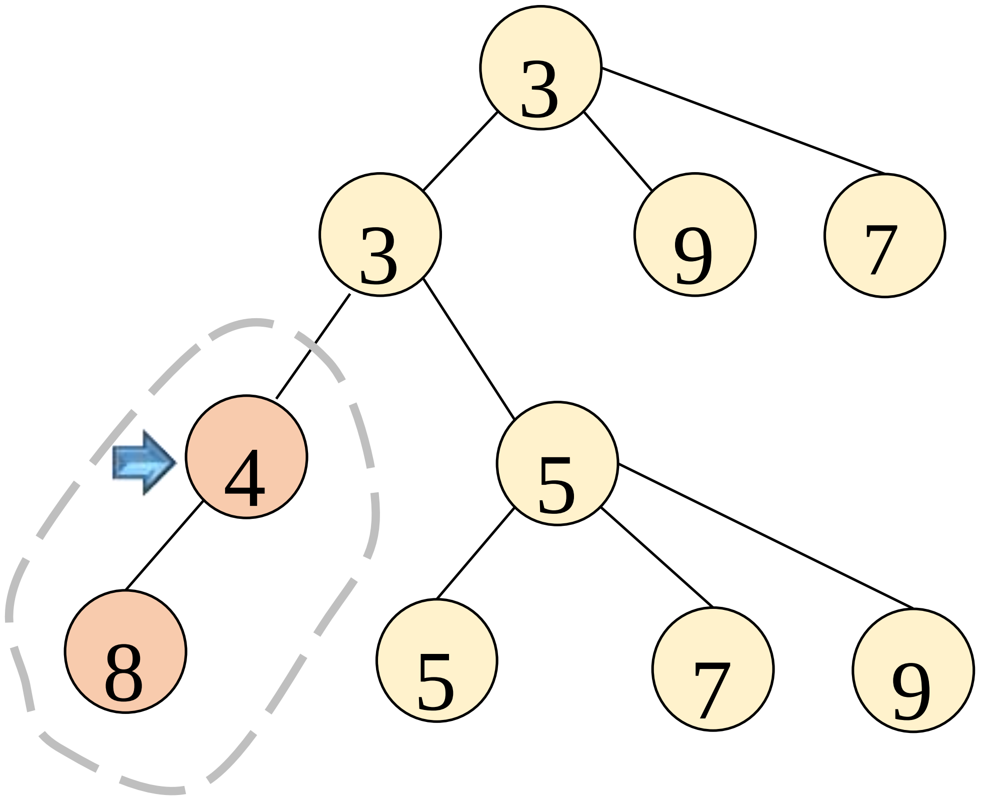
The root has the minimum value.

Min Tree Definition



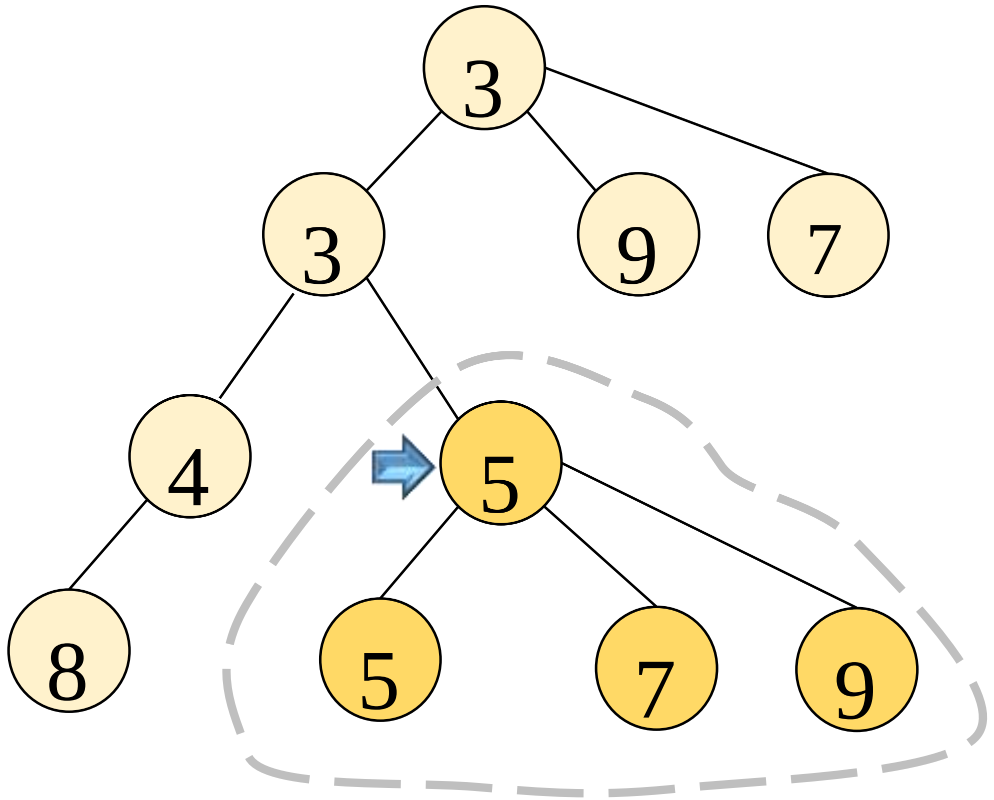
The root has the minimum value.

Min Tree Definition



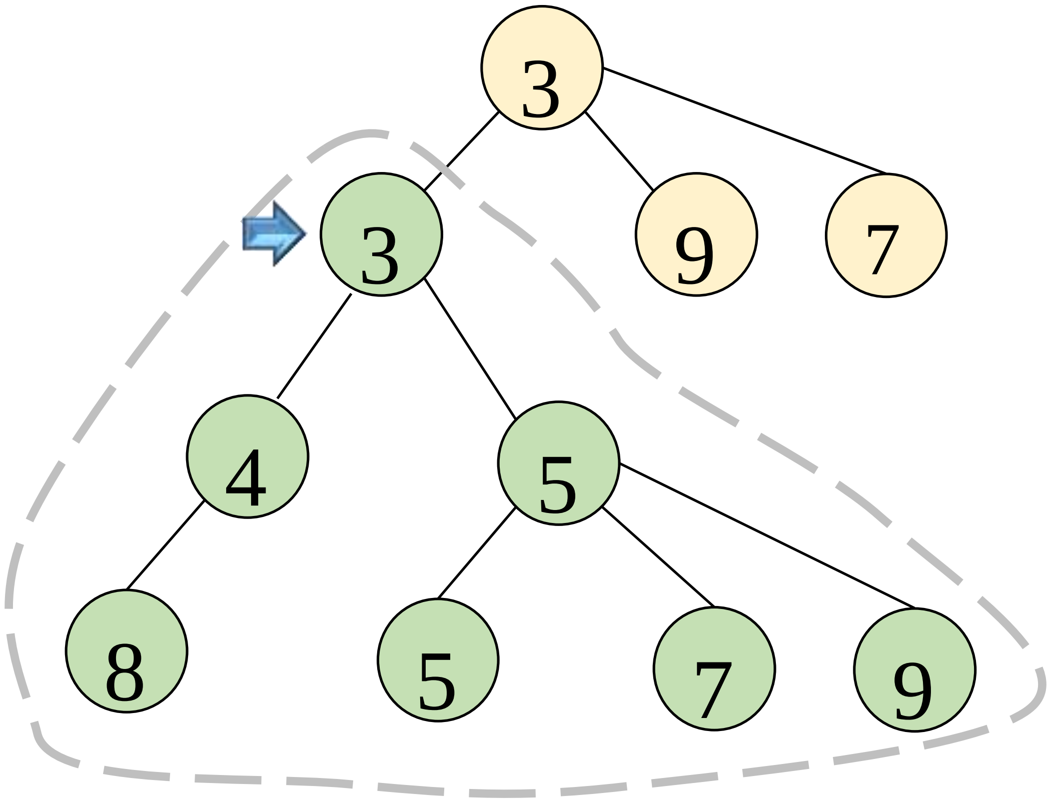
The root has the minimum value.

Min Tree Definition



The root has the minimum value.

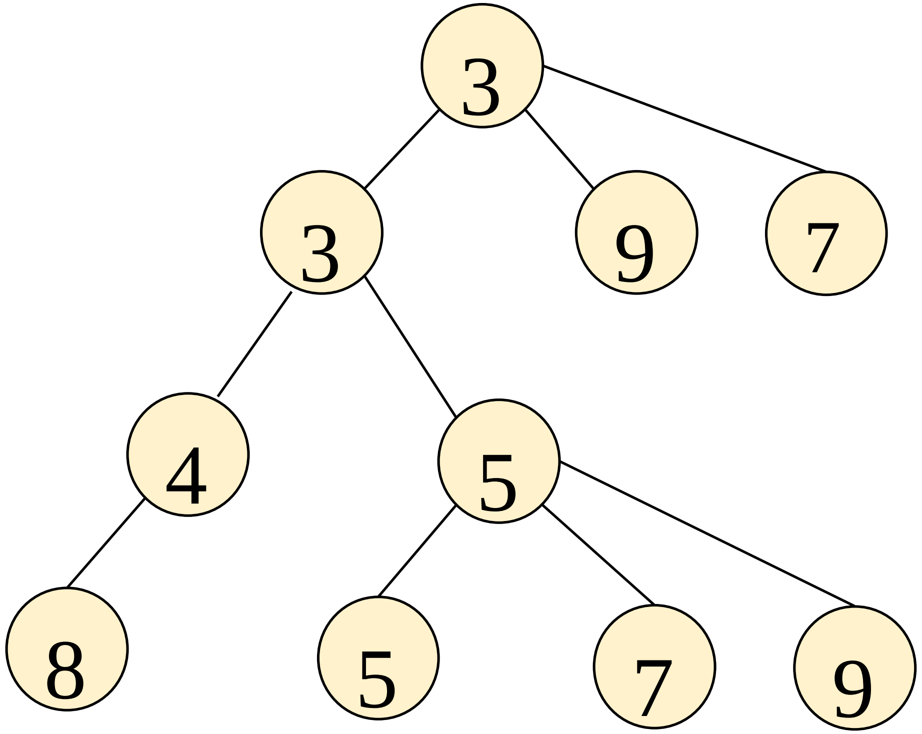
Min Tree Definition



The root has the minimum value.

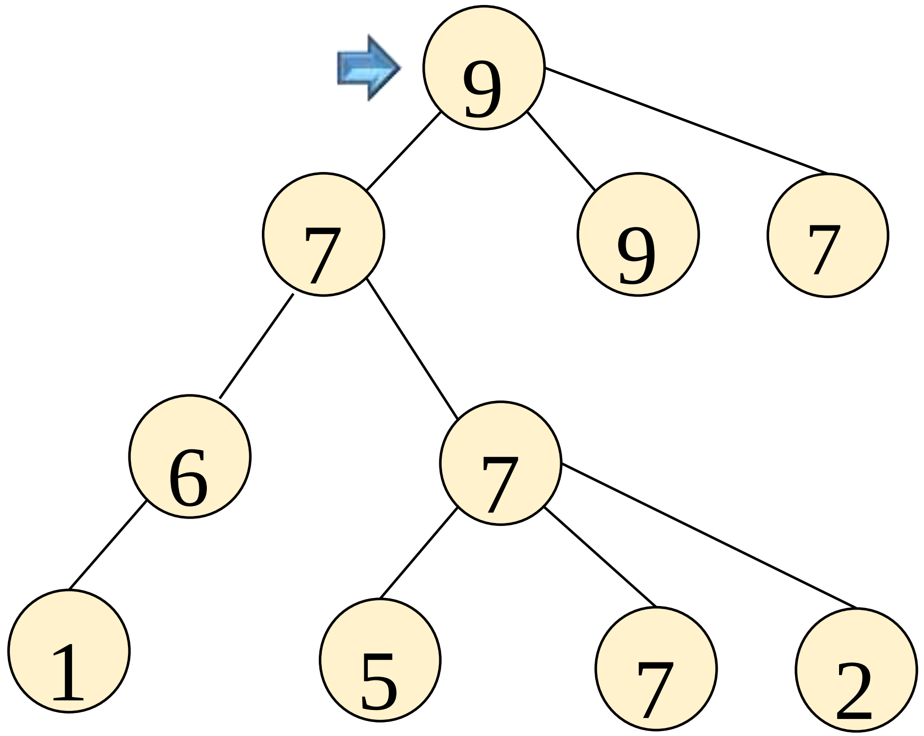
Min Tree Definition

- Each tree node has a value.
- The value of a node is the **minimum** value in the subtree rooted at that node.
- No descendent has a **smaller** value.



The root has the minimum value.

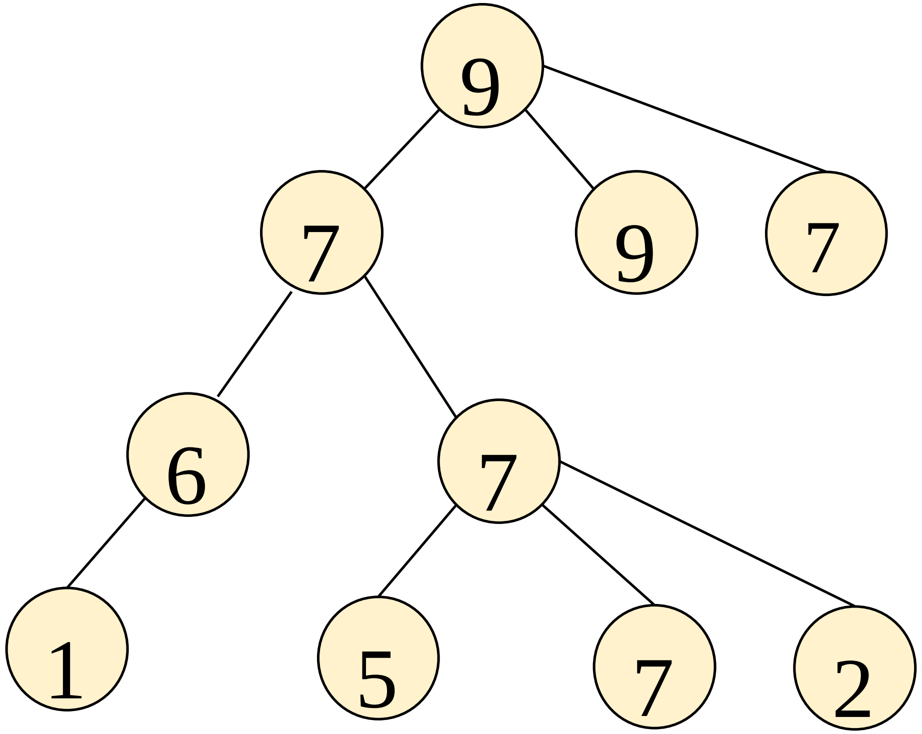
Max Tree Definition



The root has the maximum value.

Max Tree Definition

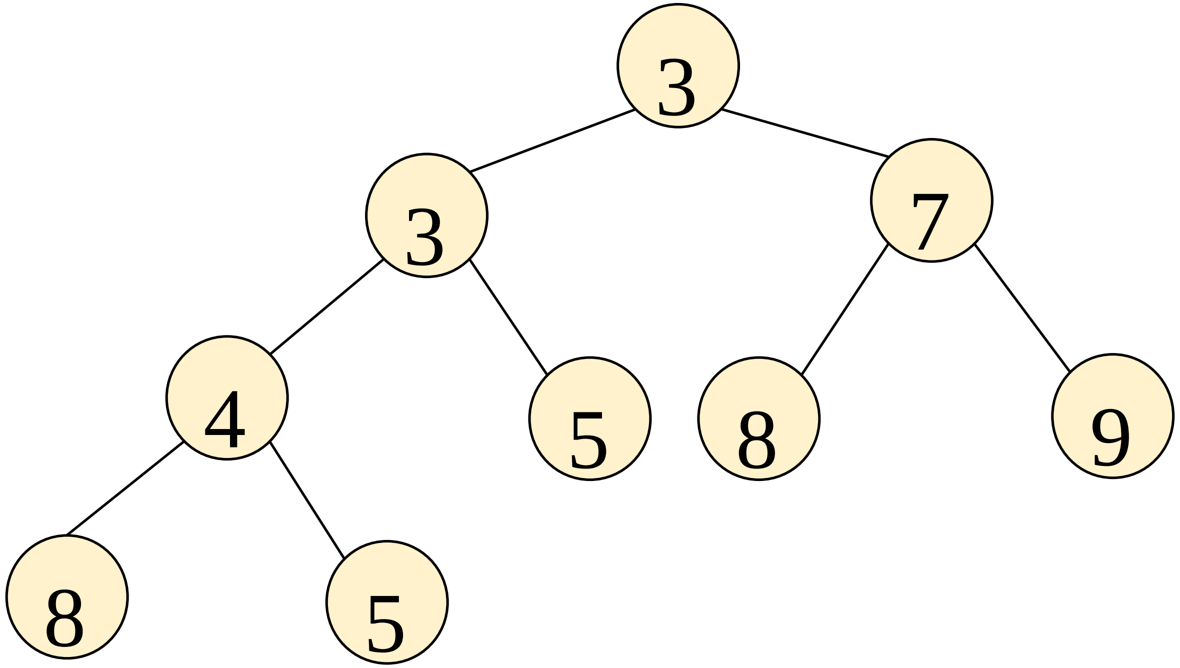
- Each tree node has a value.
- The value of a node is the **maximum** value in the subtree rooted at that node.
- No descendent has a **larger** value.



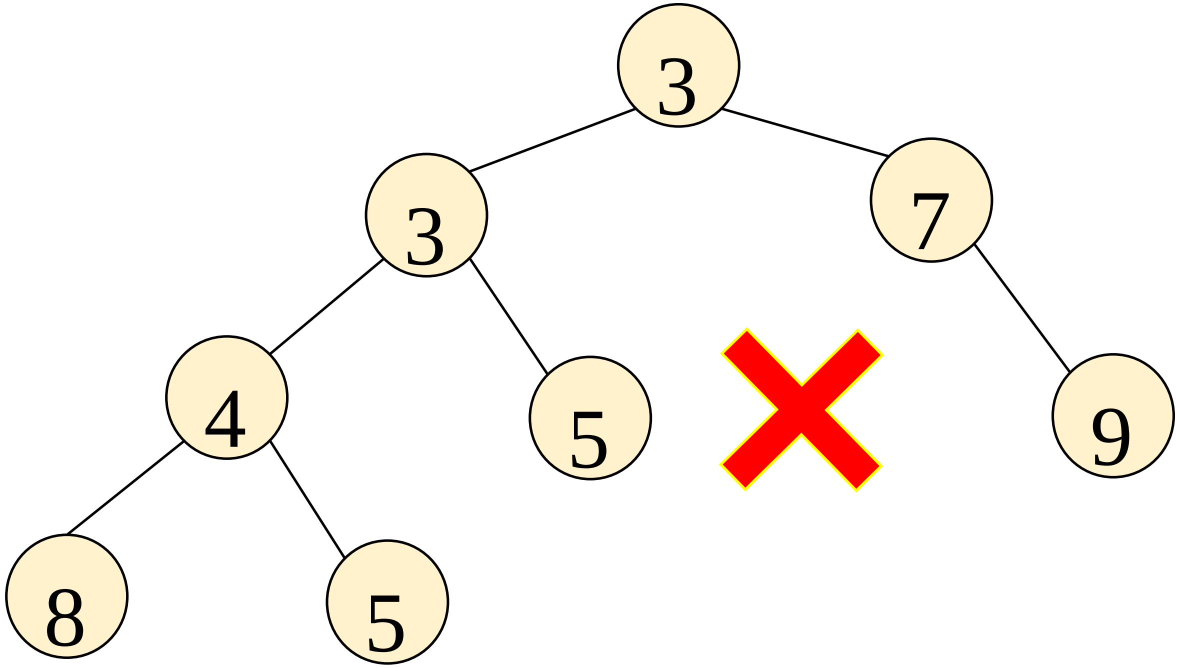
The root has the maximum value.

Min Heap Definition

- It is a complete binary tree
- It is a min tree

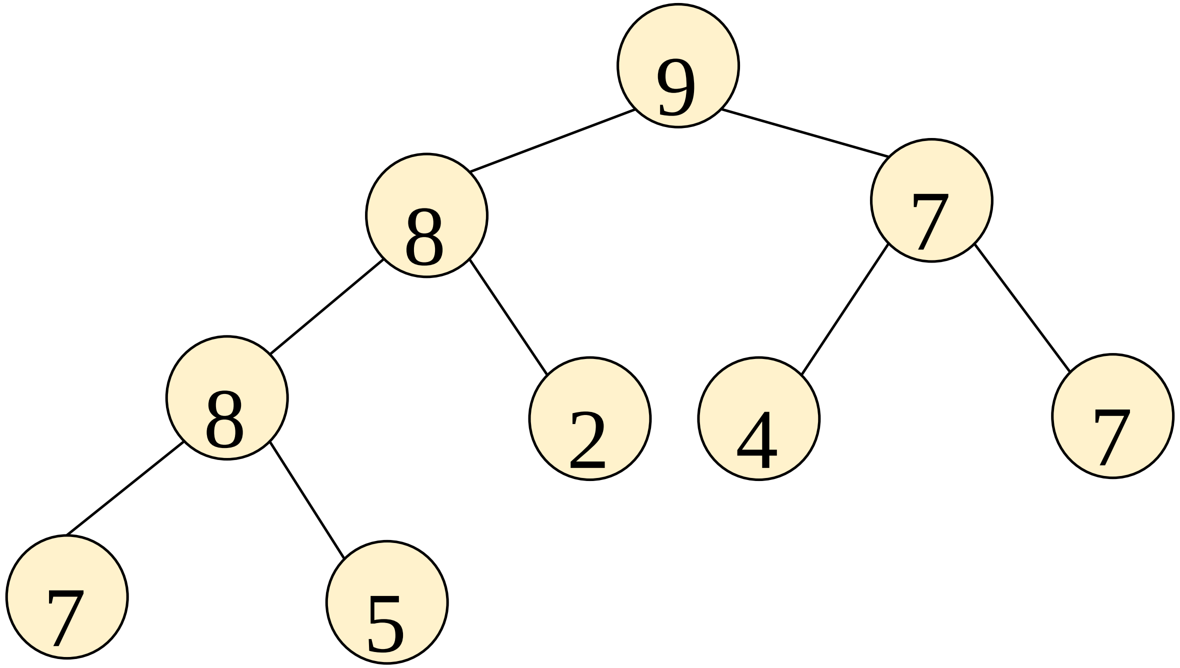


This one is not a heap.

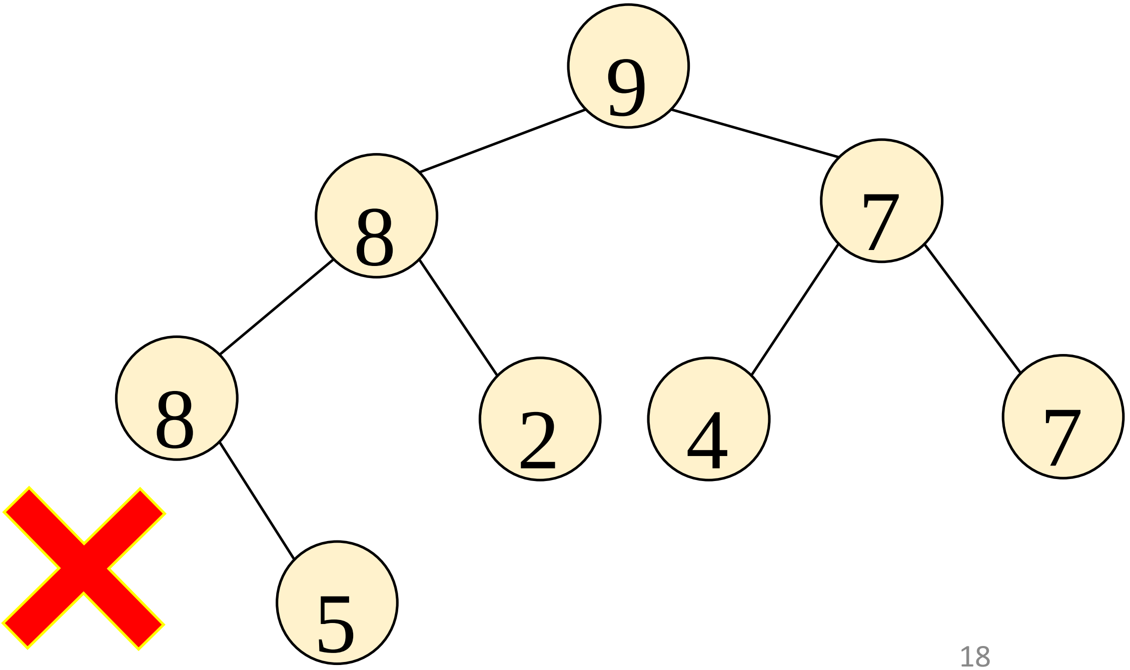


Max Heap Definition

- It is a complete binary tree
- It is a max tree



This one is not a heap.



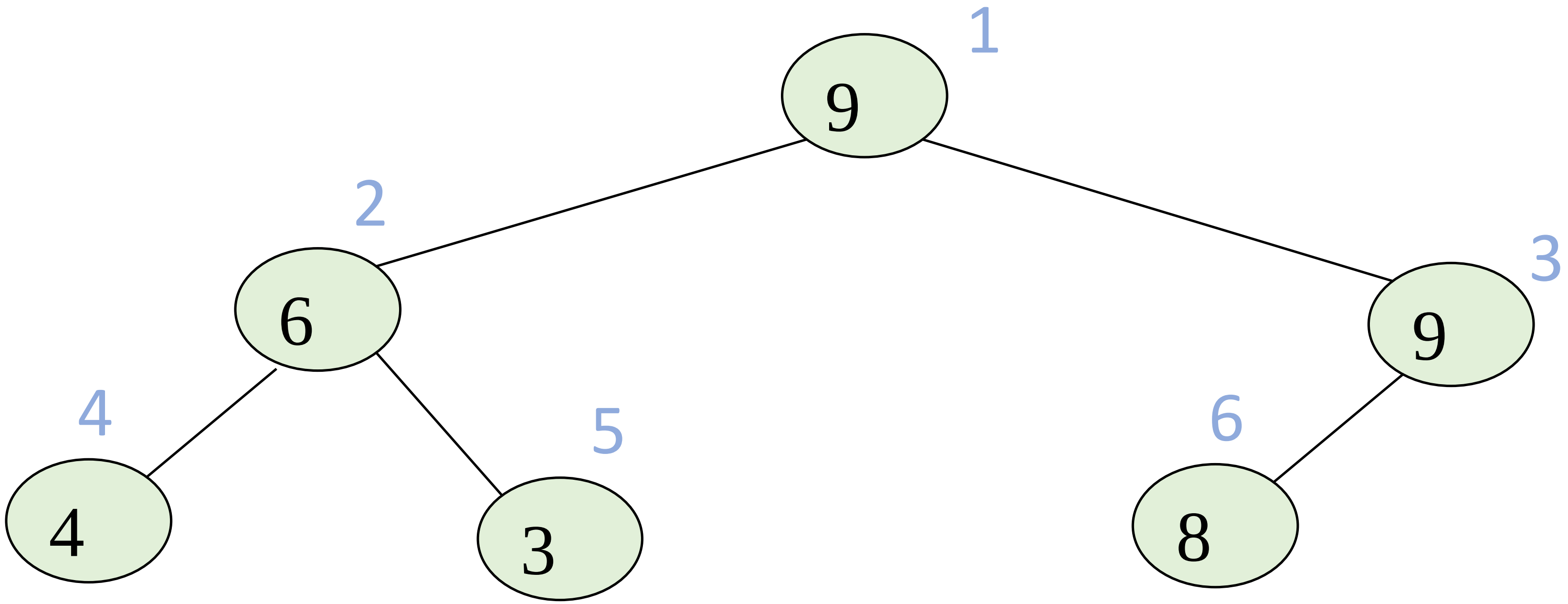
Heap Height

A heap is a complete binary tree.

The height of an n-node heap is ceil ($\log_2$ (n+1)).

Heap representation

- A heap is a complete binary tree.
- We can use array to store its elements.

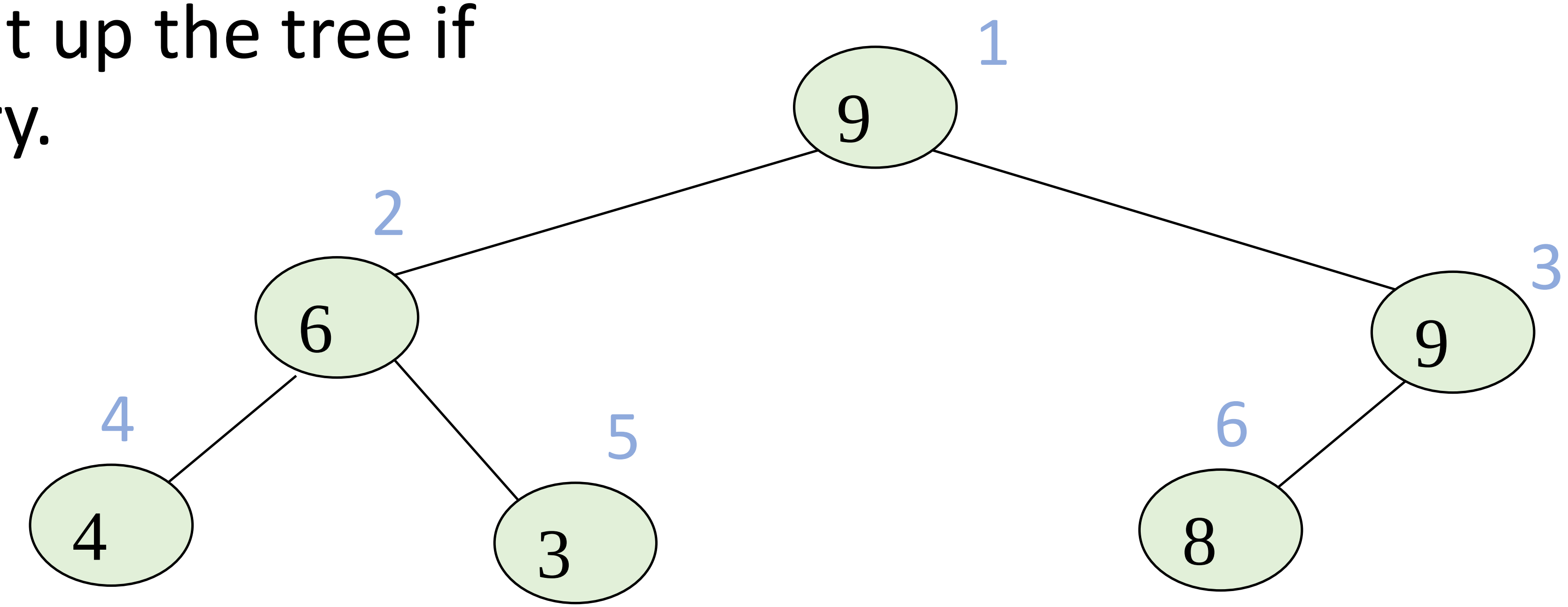


	0	1	2	3	4	5	6	7
heap[ ] =	-	9	6	9	4	3	8	-

-: a special symbol indicates that the node does not exist.

Element insertion to a max heap (Example 1)

- Place the element at the next available entry.
- Move it up the tree if necessary.

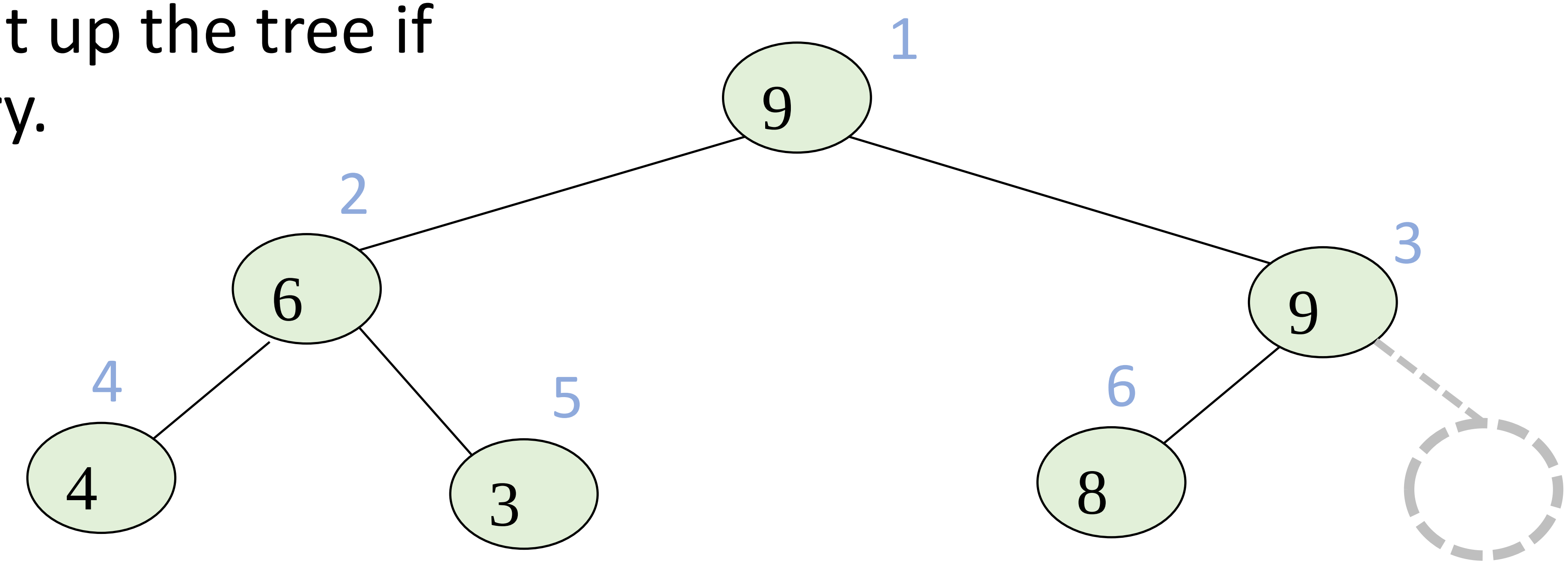


	0	1	2	3	4	5	6	7
heap[ ] =	-	9	6	9	4	3	8	-

-: a special symbol indicates that the node does not exist.

Element insertion to a max heap (Example 1)

- Place the element at the next available entry.
- Move it up the tree if necessary.

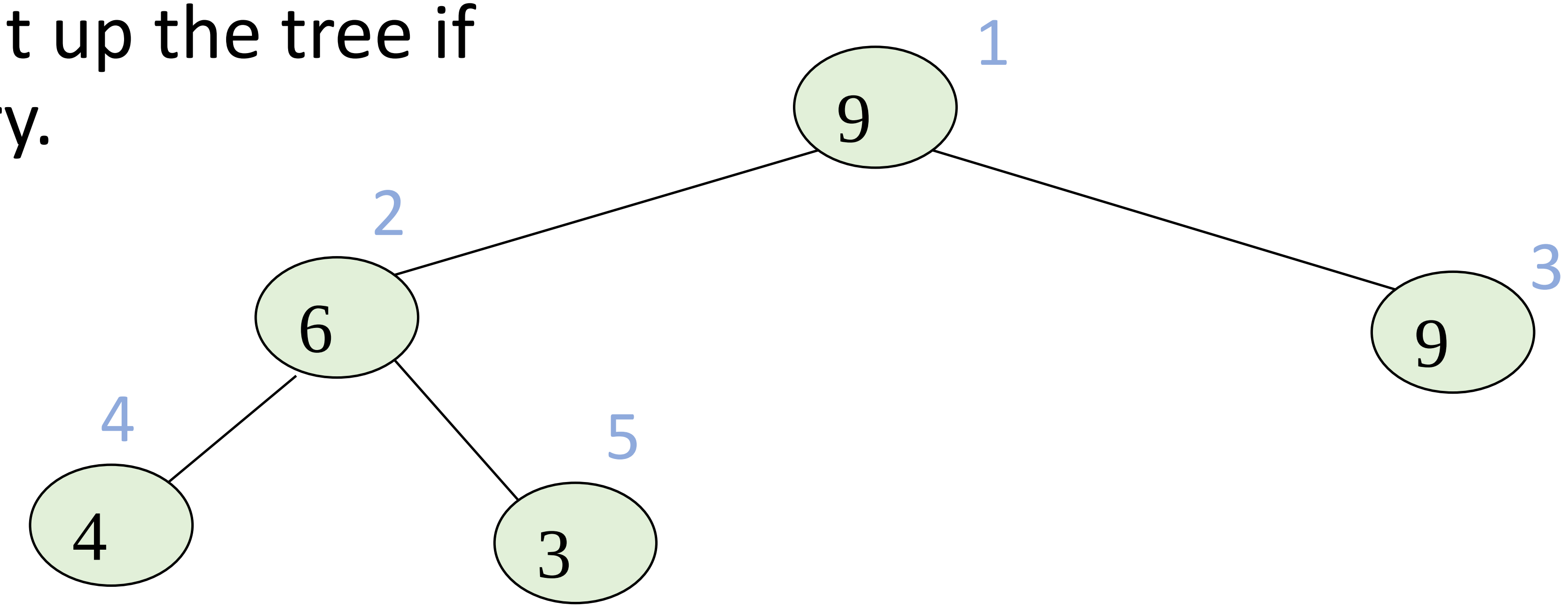


	0	1	2	3	4	5	6	7
heap[ ] =	-	9	6	9	4	3	8	-

-: a special symbol indicates that the node does not exist.

Element insertion to a max heap (Example 2)

- ➢Place the element at the next available entry.
- ➢Move it up the tree if necessary.



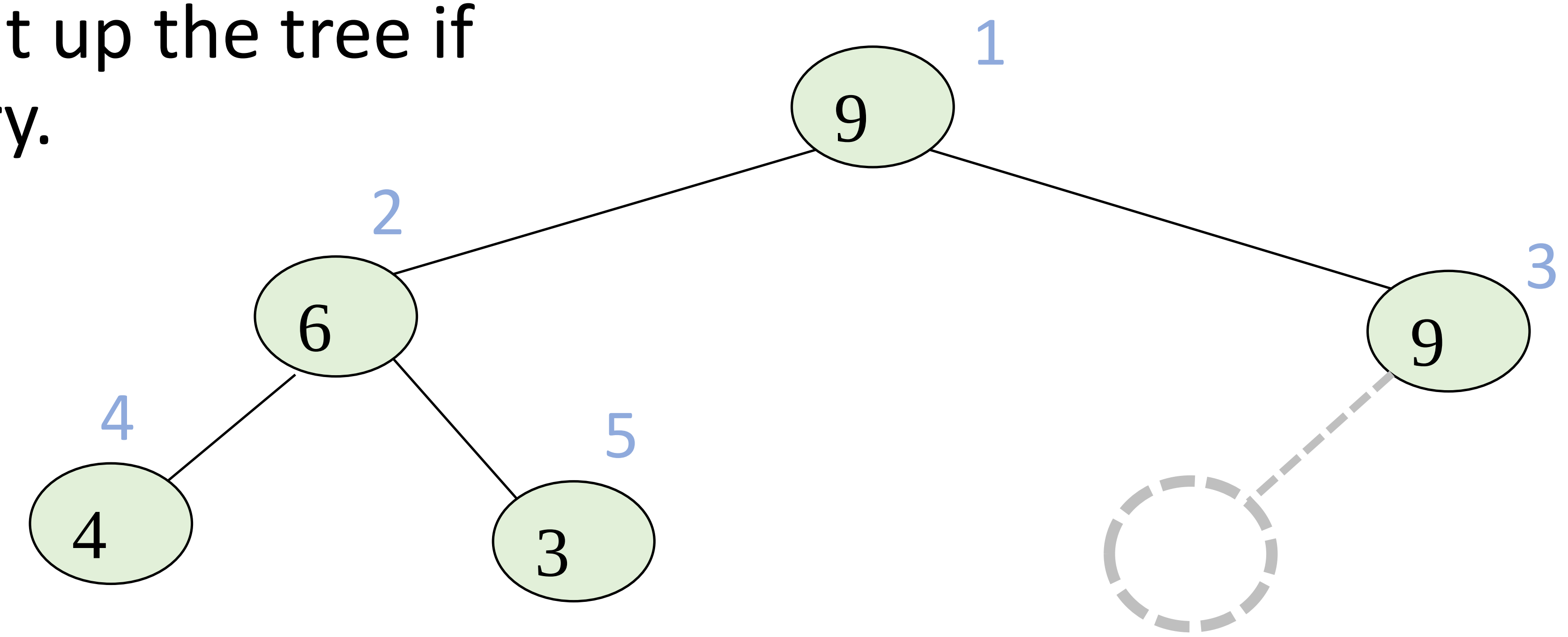
heap[ ] =

0	1	2	3	4	5	6	7
-	9	6	9	4	3	8	-

-: a special symbol indicates that the node does not exist.

Element insertion to a max heap (Example 2)

- Place the element at the next available entry.
- Move it up the tree if necessary.

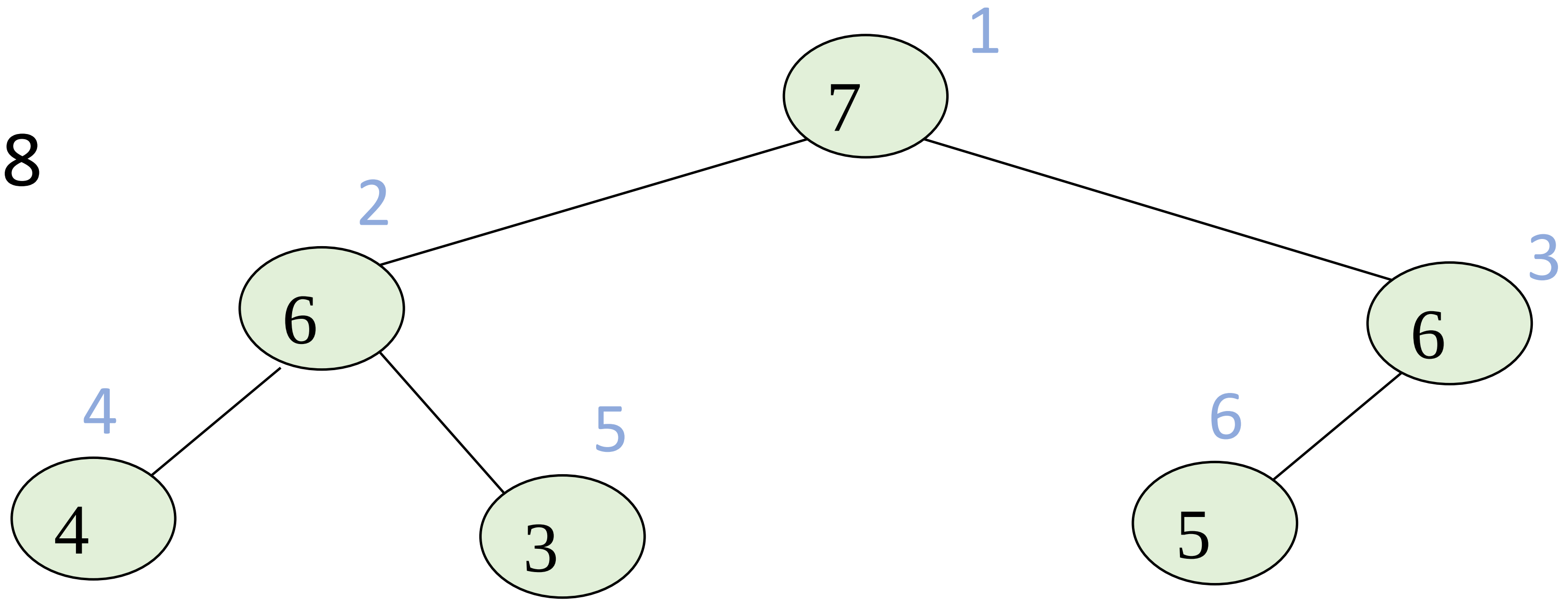


	0	1	2	3	4	5	6	7
heap[ ] =	-	9	6	9	4	3	8	-

-: a special symbol indicates that the node does not exist.

Element insertion to a max heap

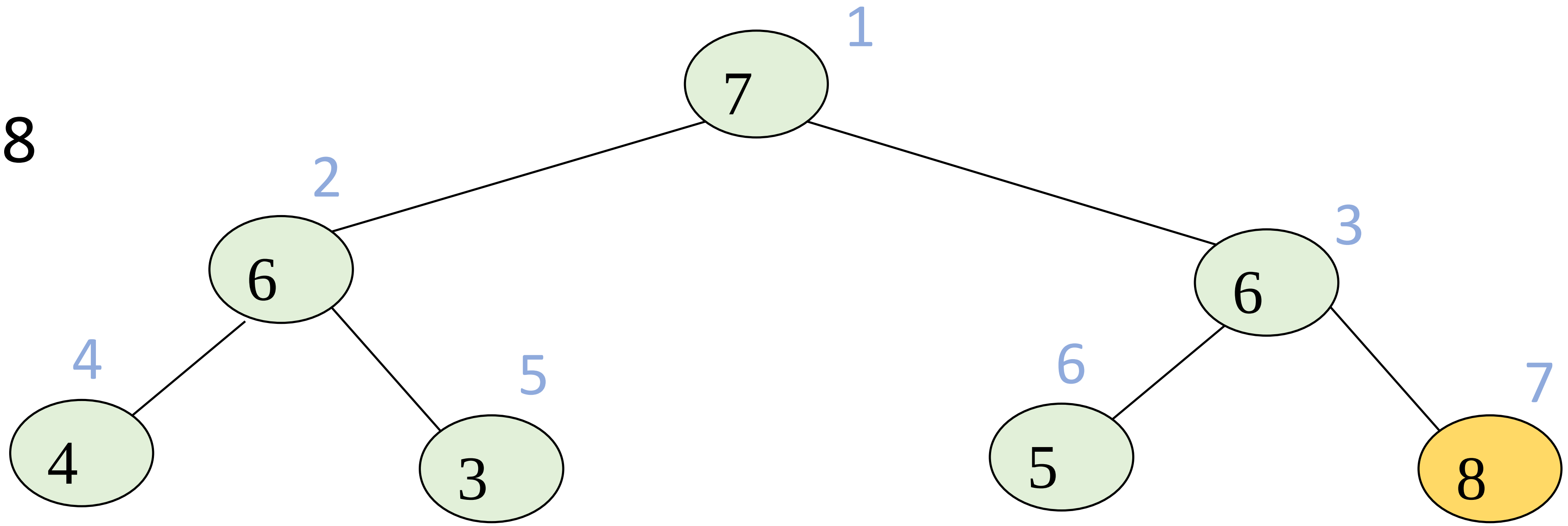
Insert 8



	0	1	2	3	4	5	6	7
heap[ ] =	-	7	6	6	4	3	5	-

Element insertion to a max heap

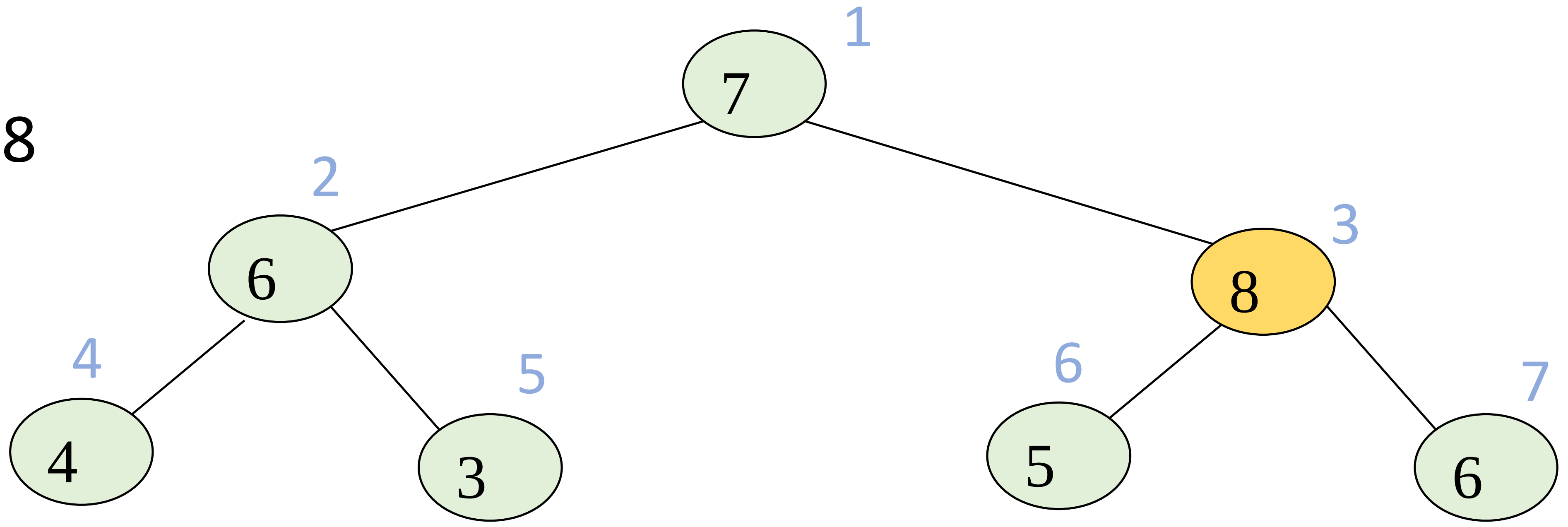
Insert 8



Move the new element up if it is larger than its parent.

Element insertion to a max heap

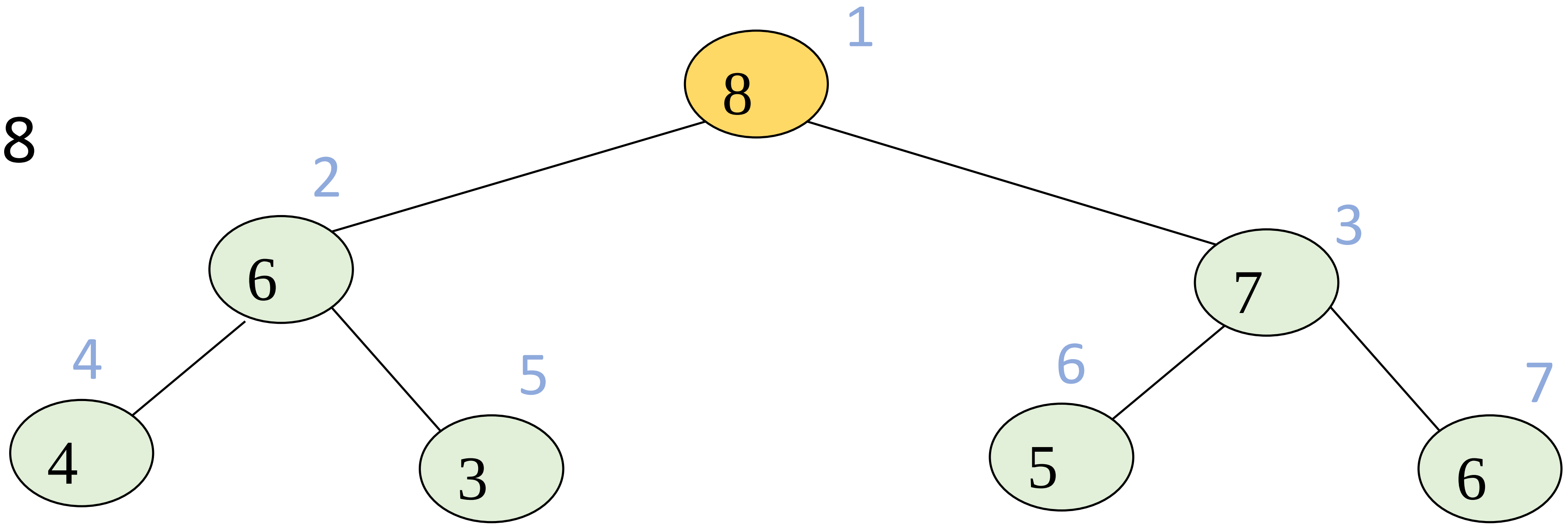
Insert 8



Move the new element up if it is larger than its parent.

Element insertion to a max heap

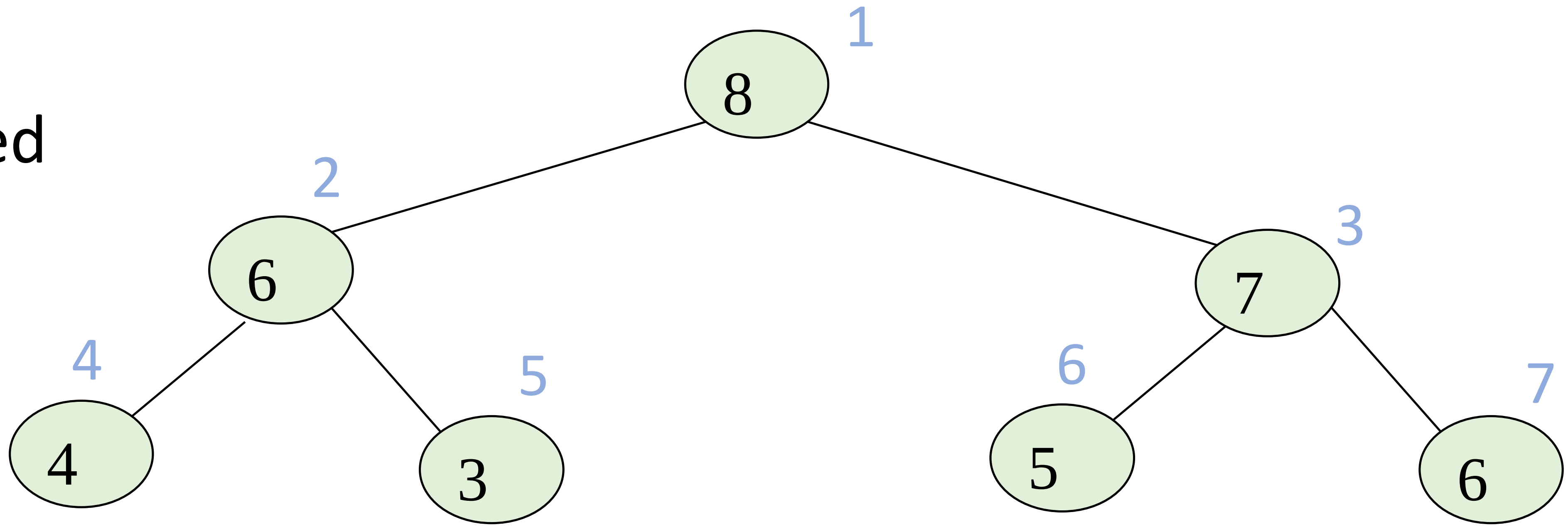
Insert 8



Move the new element up if it is larger than its parent.

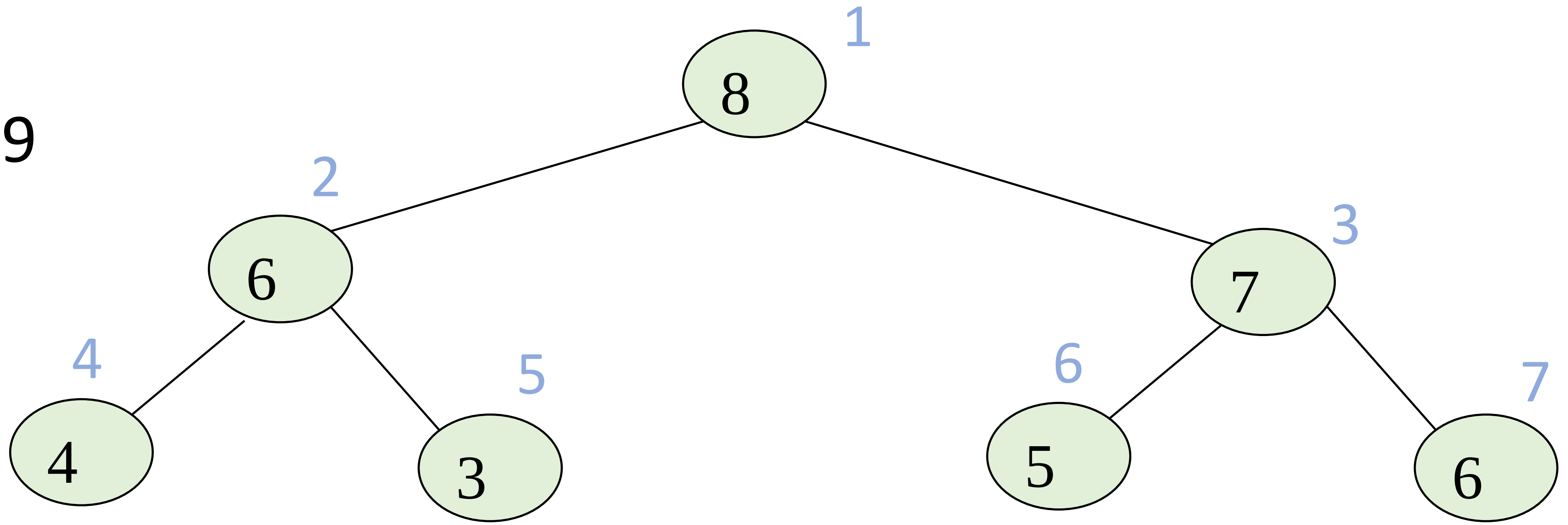
Element insertion to a max heap

finished



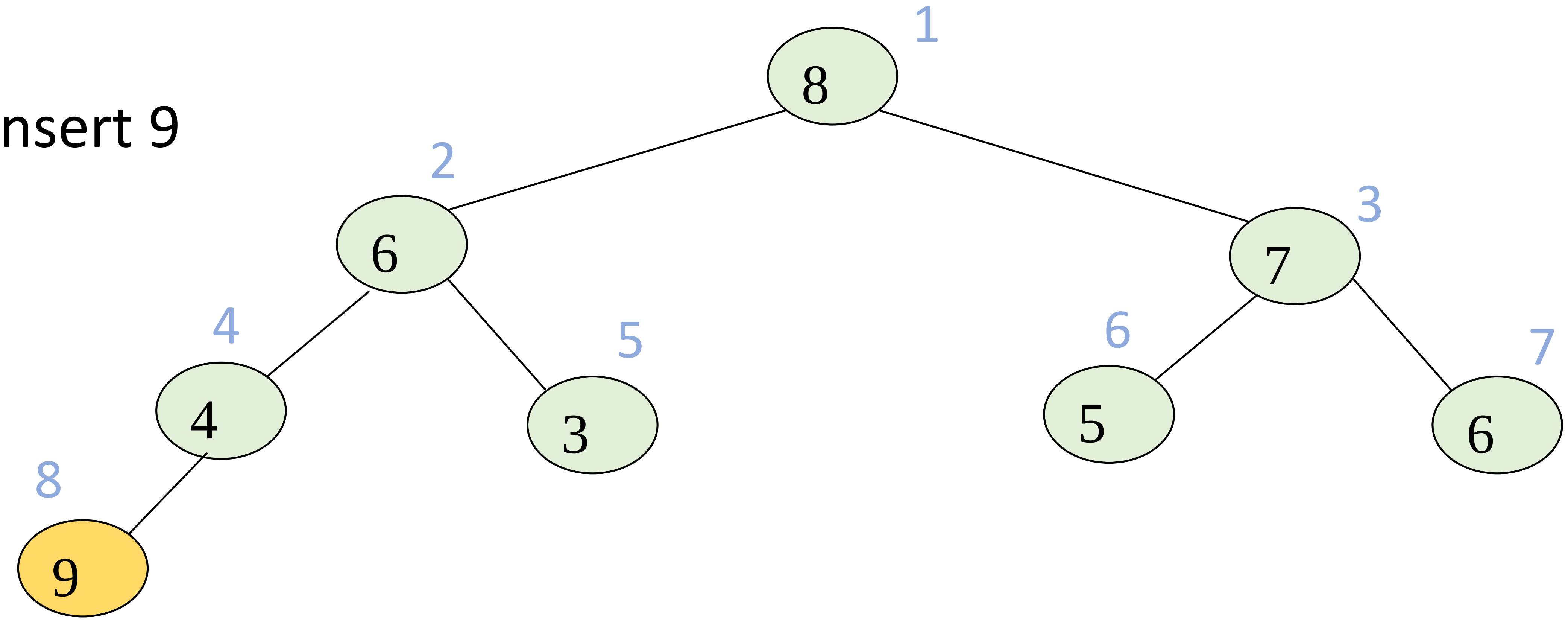
Element insertion to a max heap

Insert 9



Element insertion to a max heap

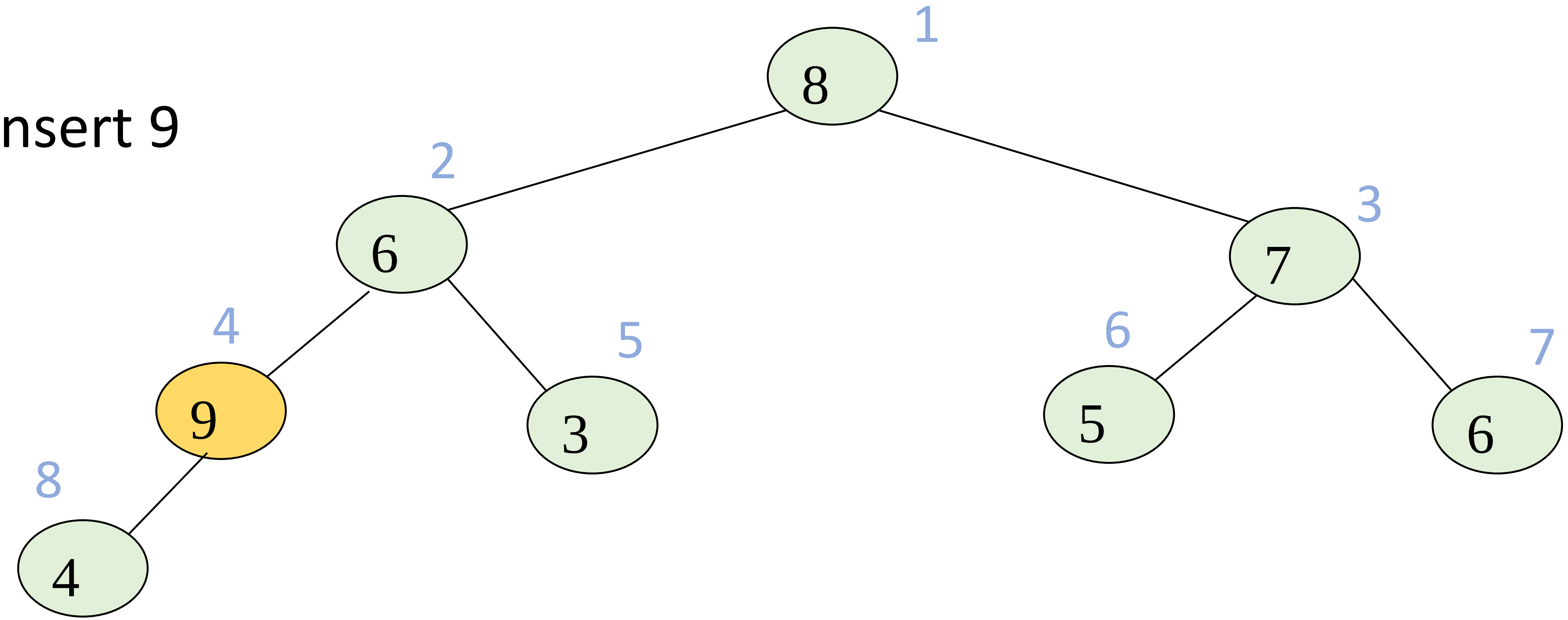
Insert 9



Move the new element up if it is larger than its parent.

Element insertion to a max heap

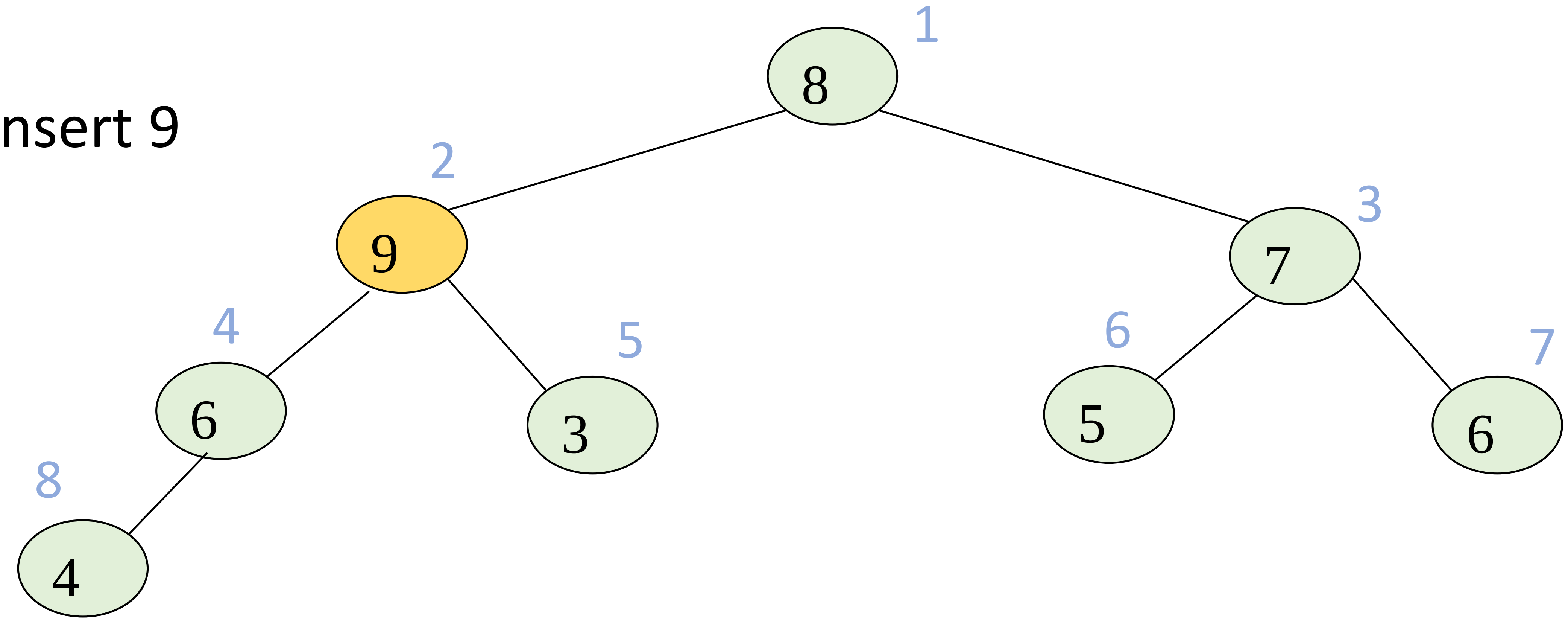
Insert 9



Move the new element up if it is larger than its parent.

Element insertion to a max heap

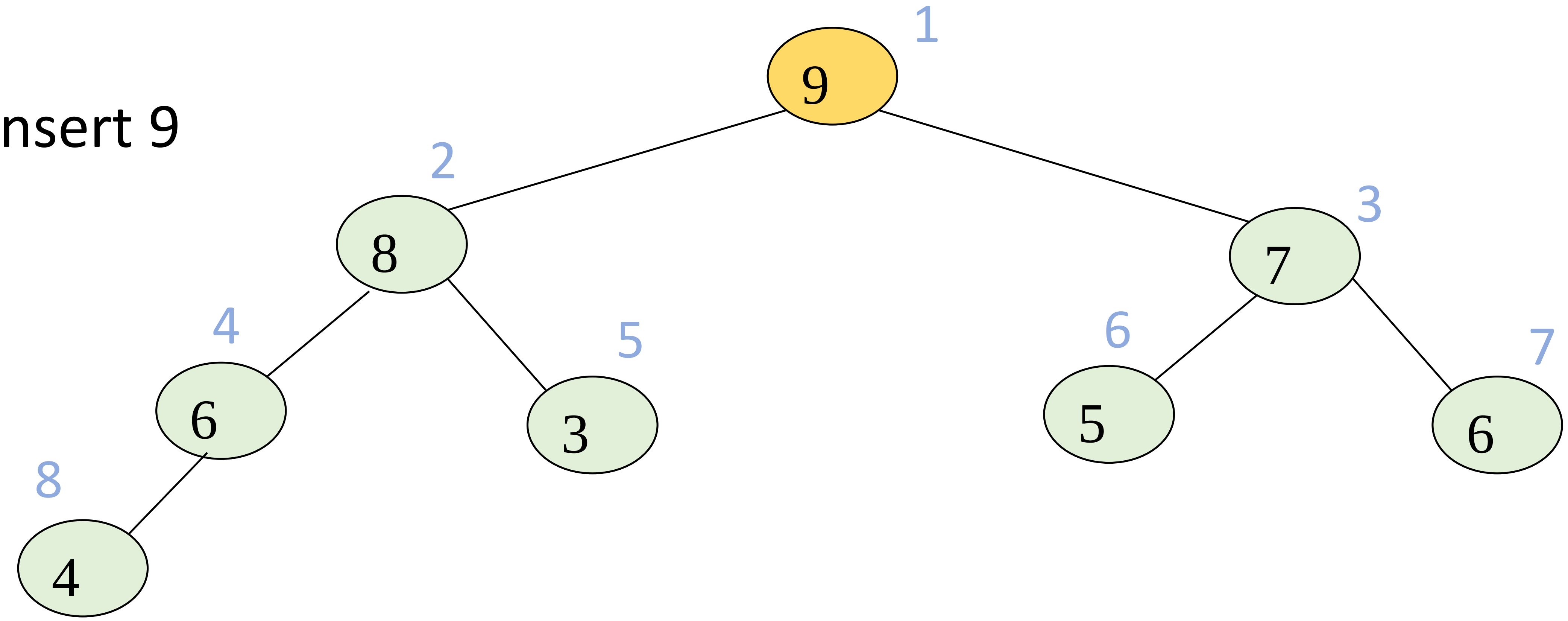
Insert 9



Move the new element up if it is larger than its parent.

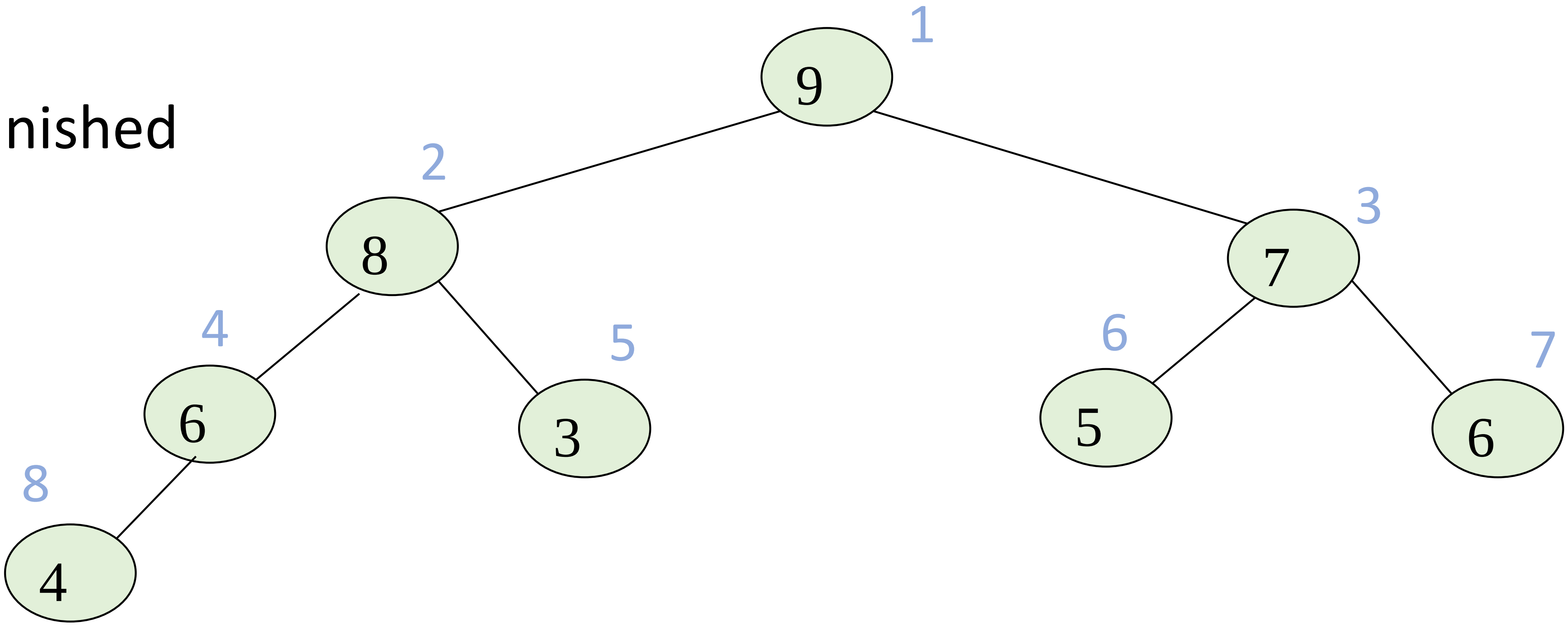
Element insertion to a max heap

Insert 9

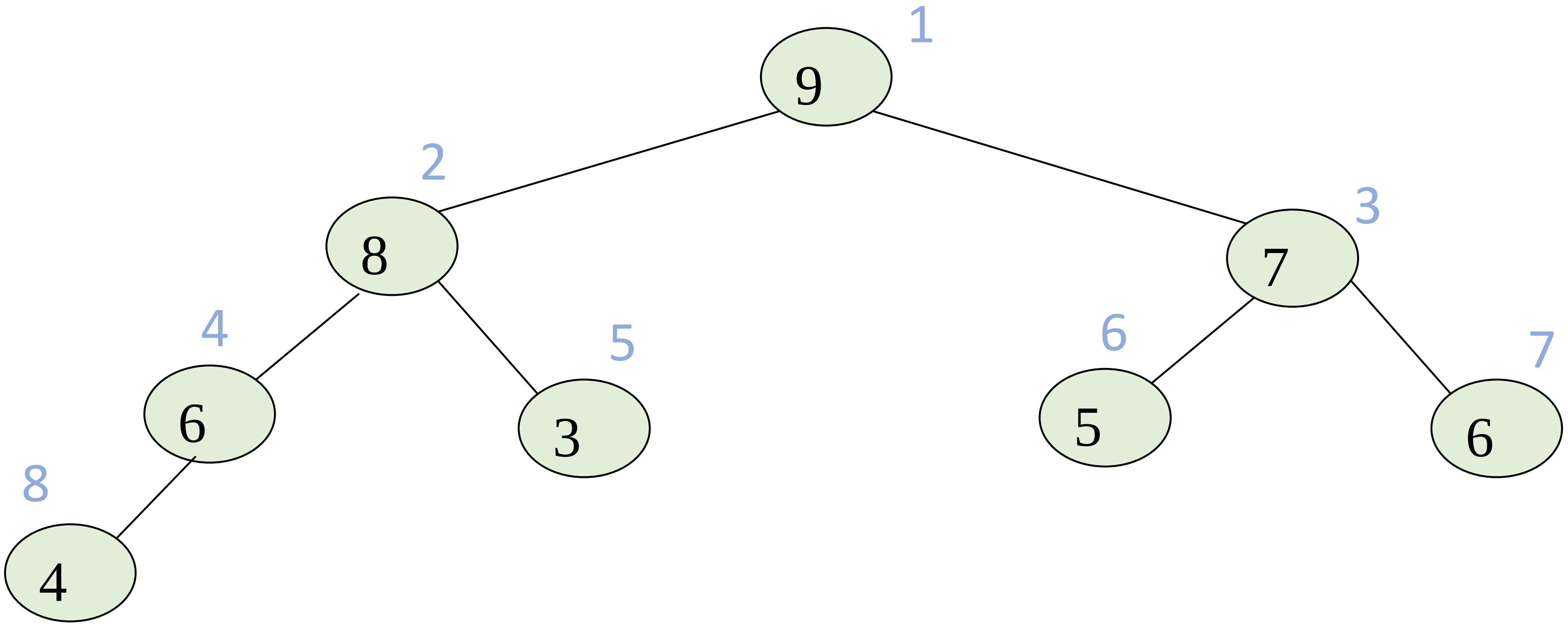


Element insertion to a max heap

finished



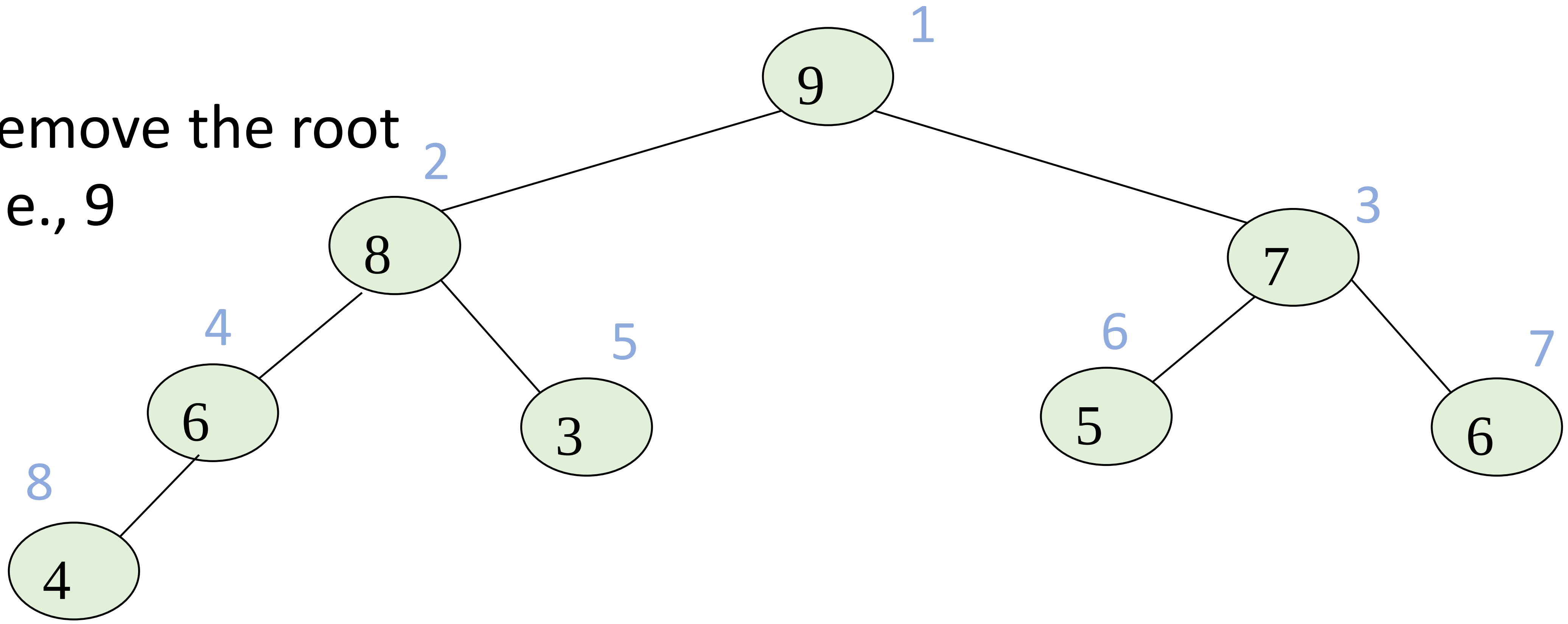
Element Removal from a max heap



Element Removal from a max heap

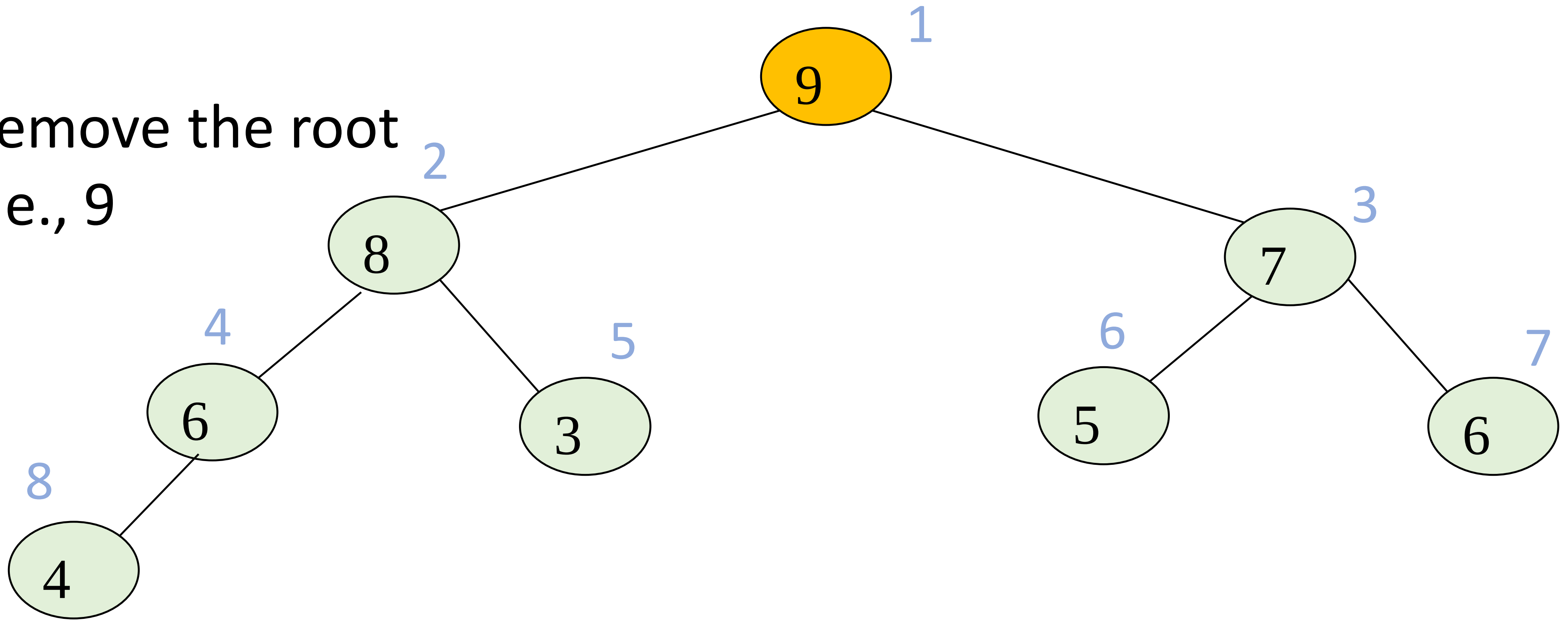
remove the root

i.e., 9



Element Removal from a max heap

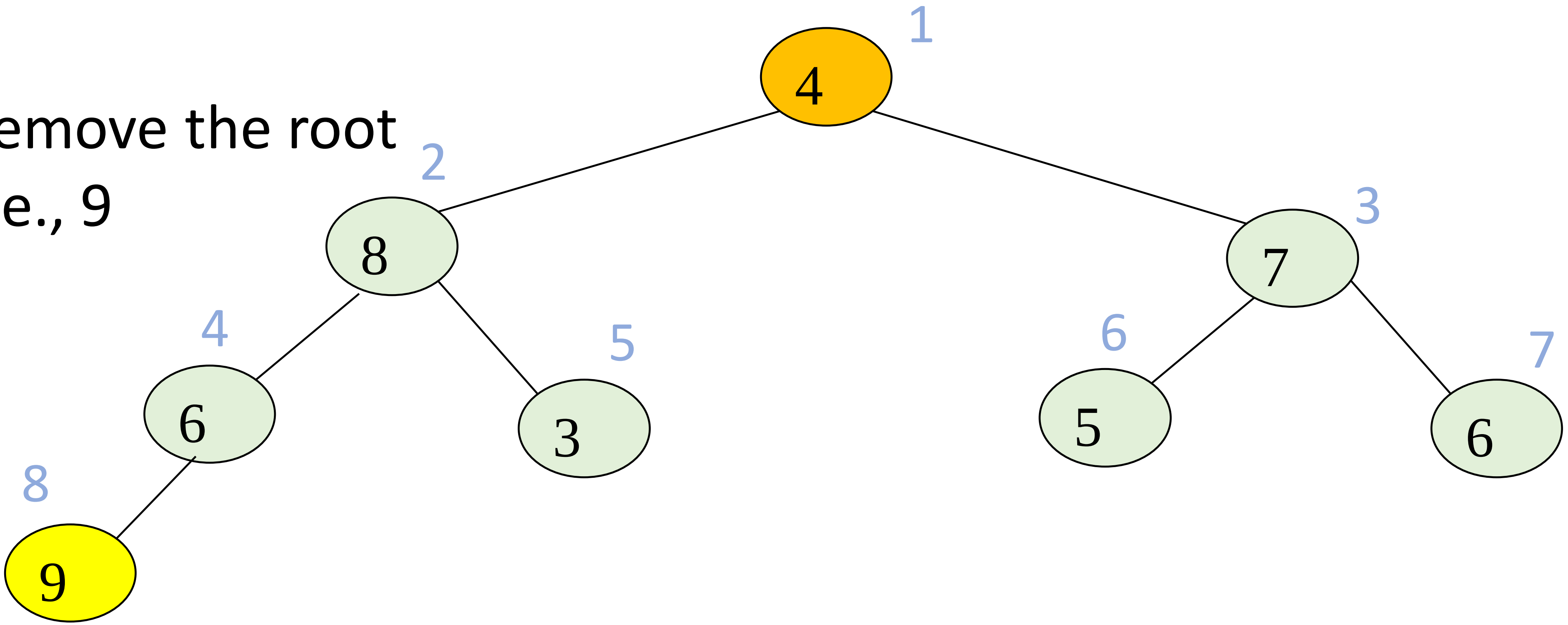
remove the root
i.e., 9



Swap the root and the last element.

Element Removal from a max heap

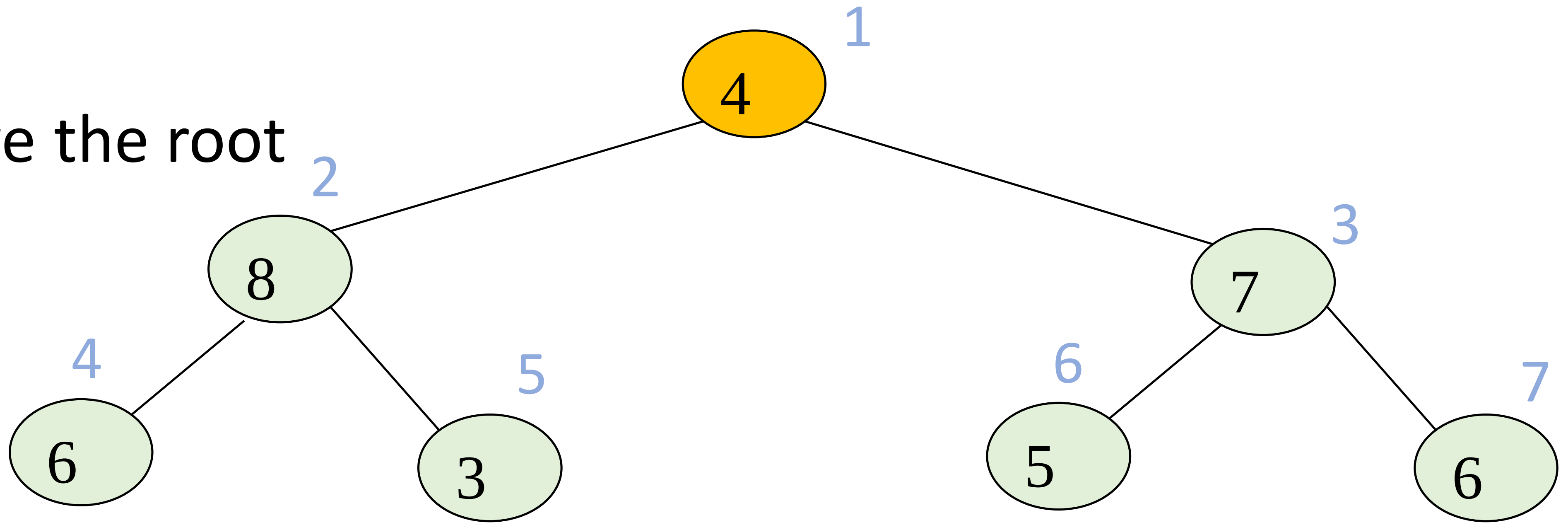
remove the root
i.e., 9



Remove the last element

Element Removal from a max heap

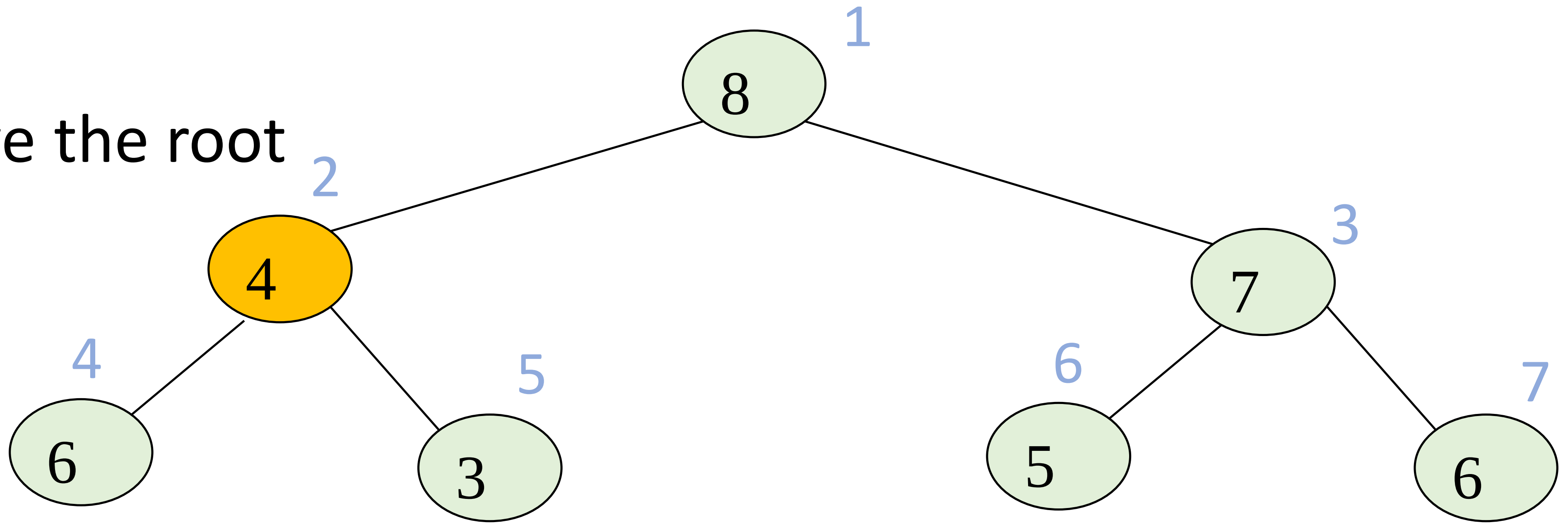
remove the root
i.e., 9



Move the element down if it is smaller than one of its child.
Swap with the maximum child.

Element Removal from a max heap

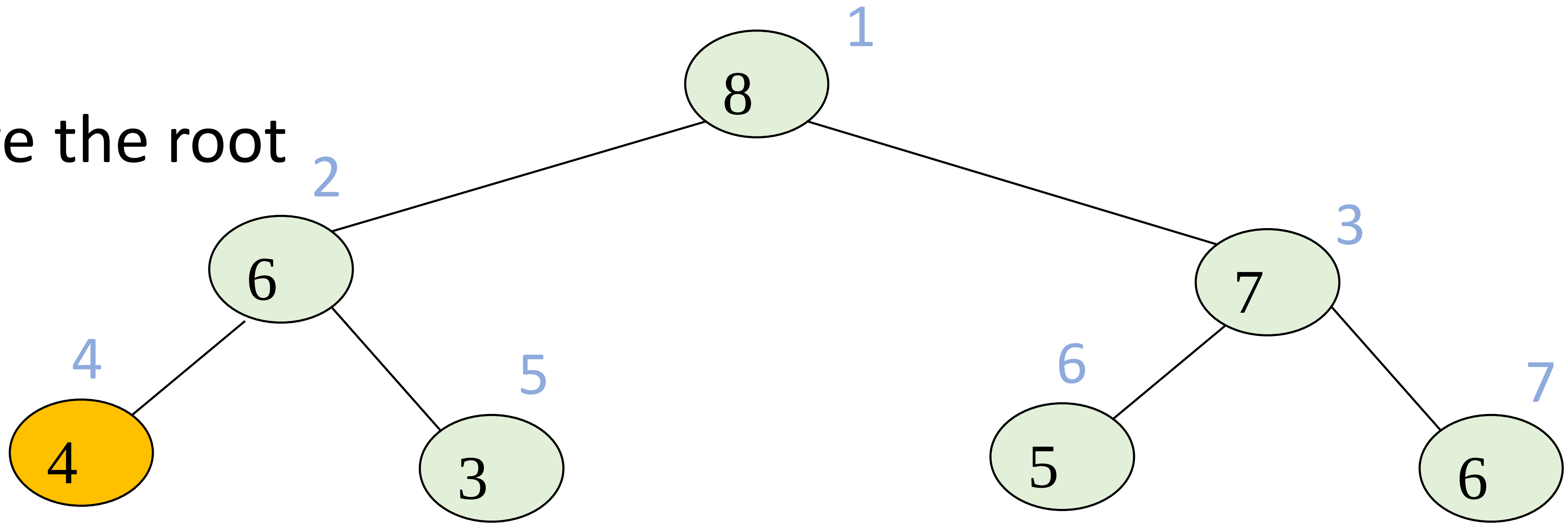
remove the root
i.e., 9



Move the element down if it is smaller than one of its child.
Swap with the maximum child.

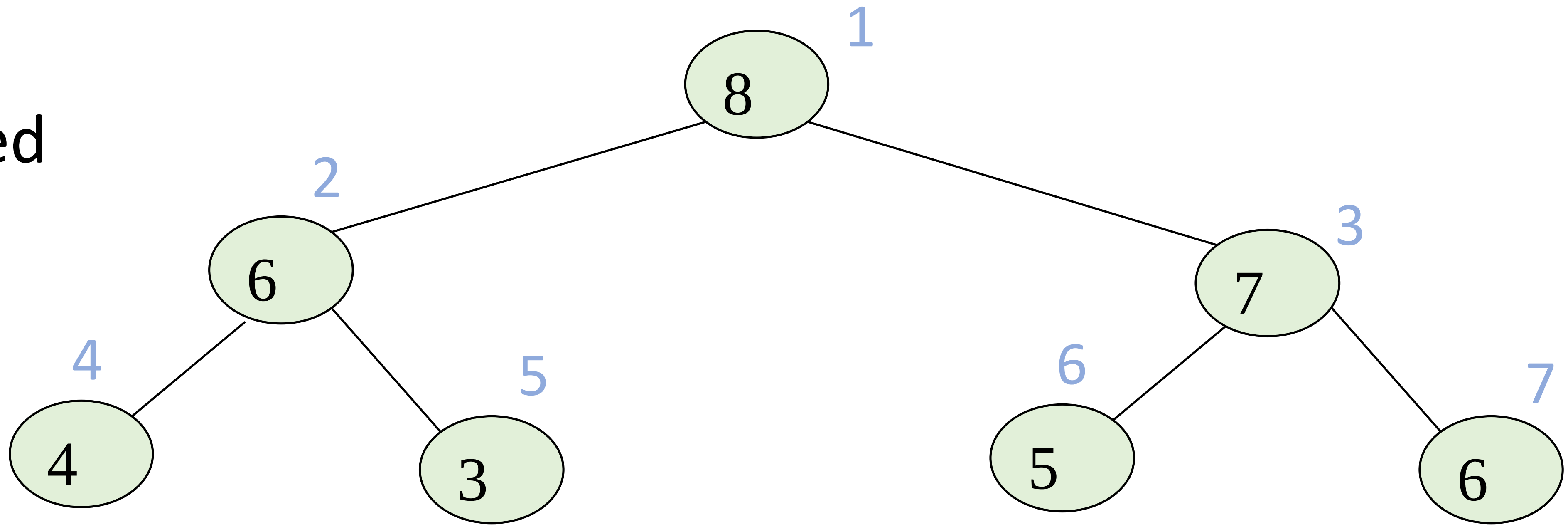
Element Removal from a max heap

remove the root
i.e., 9



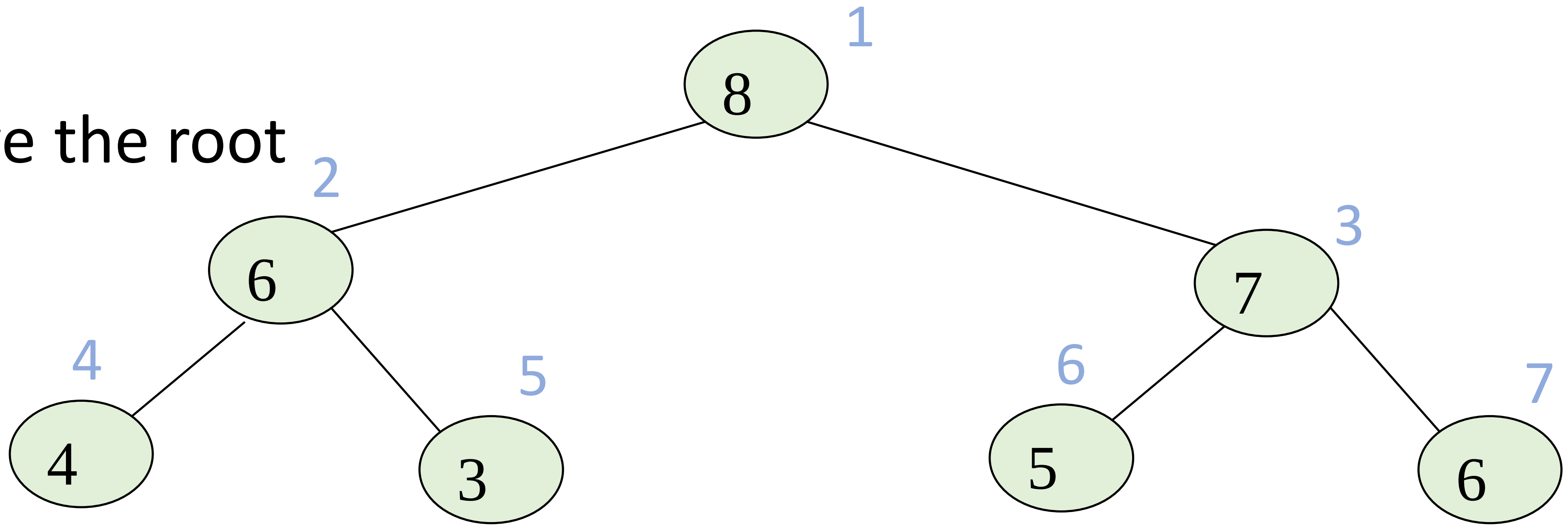
Element Removal from a max heap

finished



Element Removal from a max heap

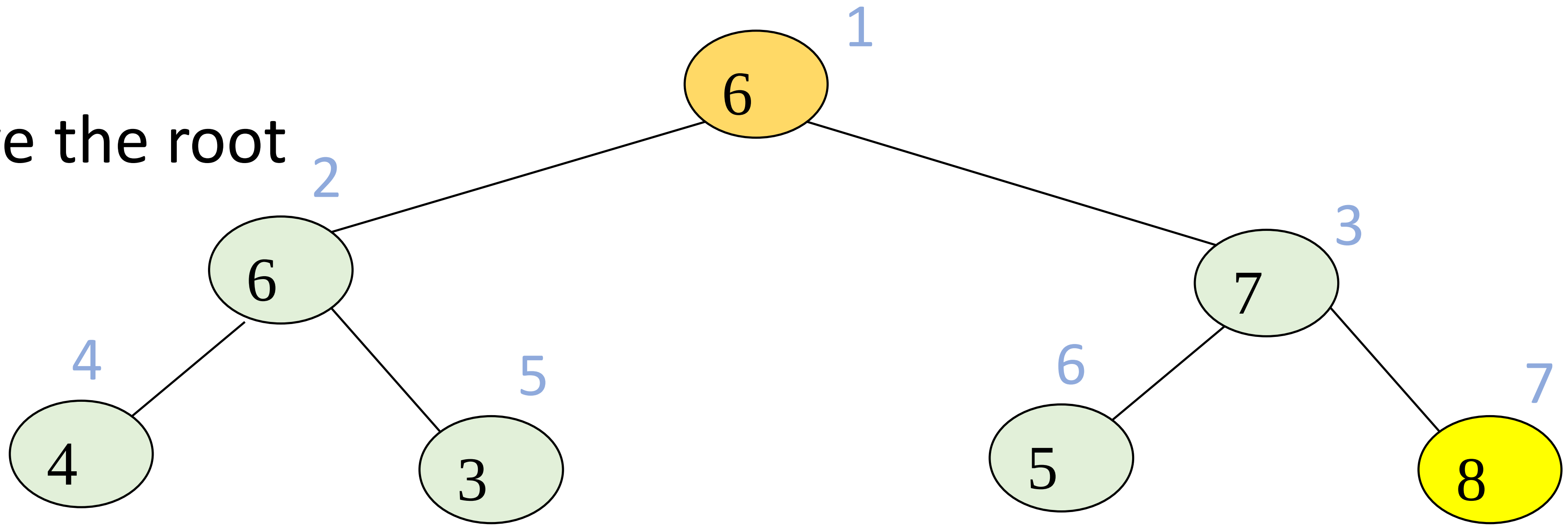
remove the root
i.e., 8



Element Removal from a max heap

remove the root

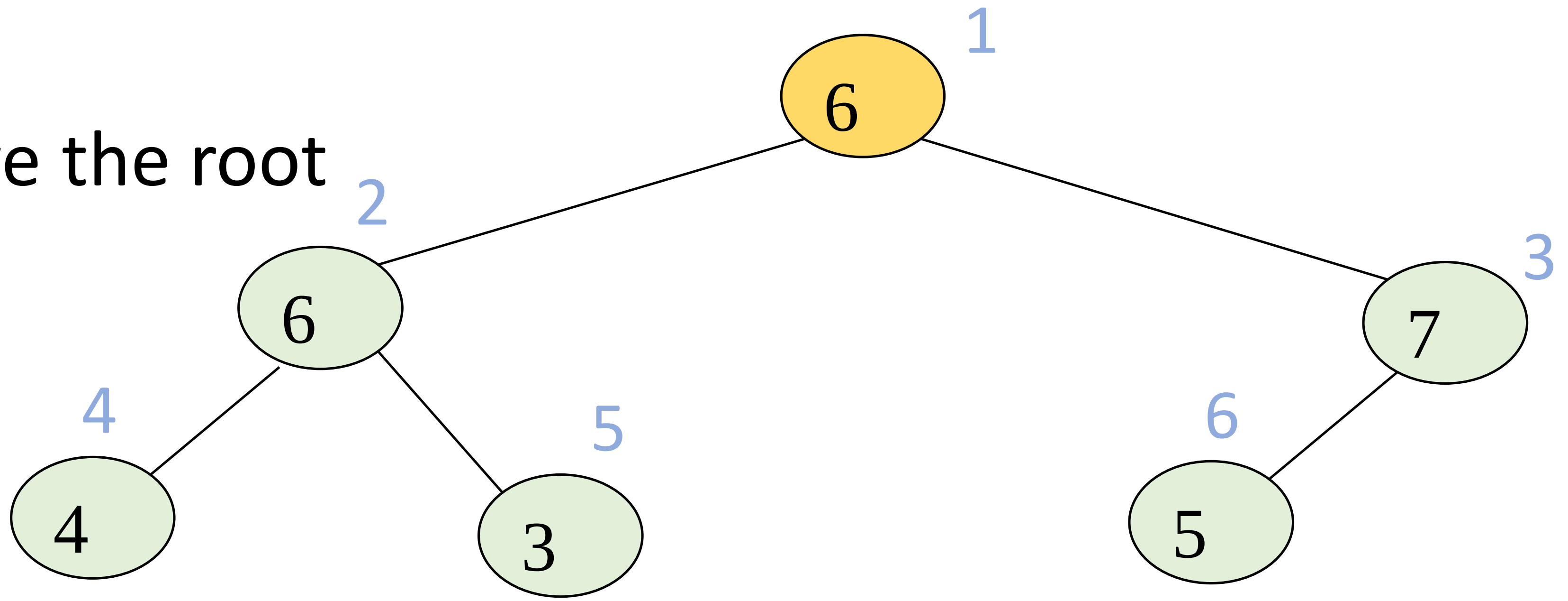
i.e., 8



Remove the last element

Element Removal from a max heap

remove the root
i.e., 8

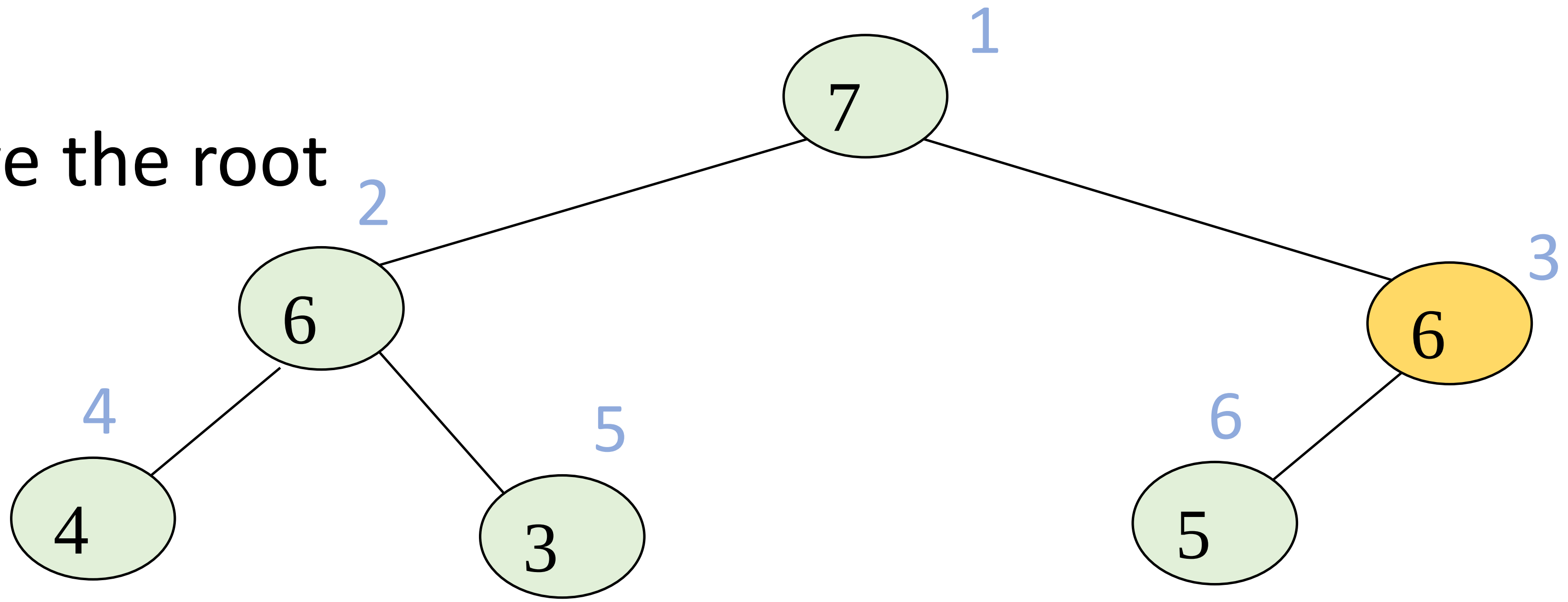


Move the element down if it is smaller than one of its child.
Swap with the maximum child.

Element Removal from a max heap

remove the root

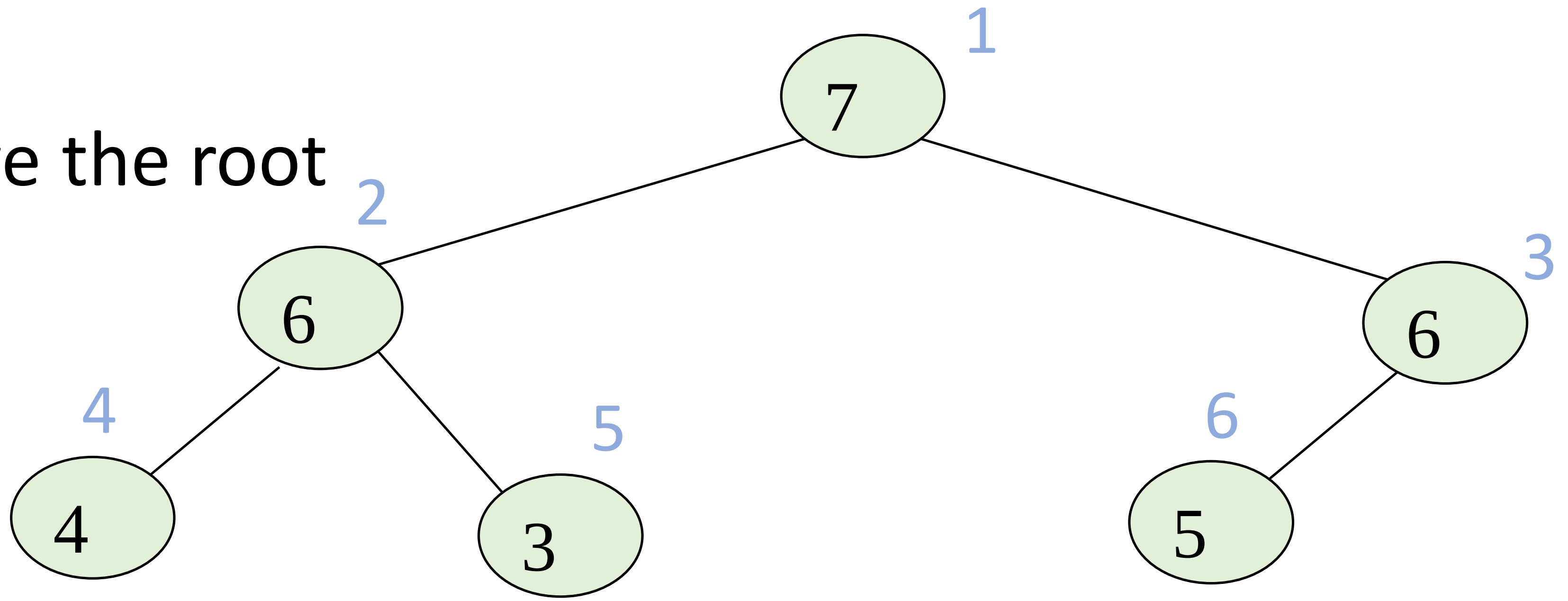
i.e., 8



Element Removal from a max heap

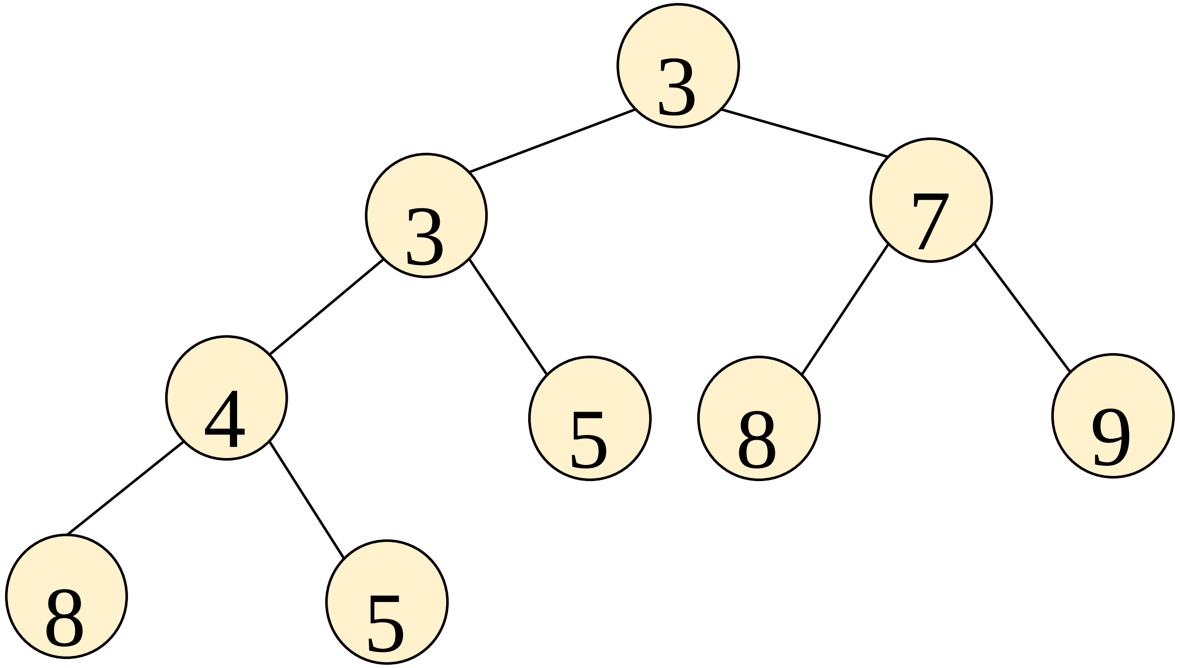
remove the root

i.e., 8



Complexity for element insertion and removal

- The tree height is $\log_2(n+1)$
- Insertion: $O(\log_2 n)$
- Removal: $O(\log_2 n)$



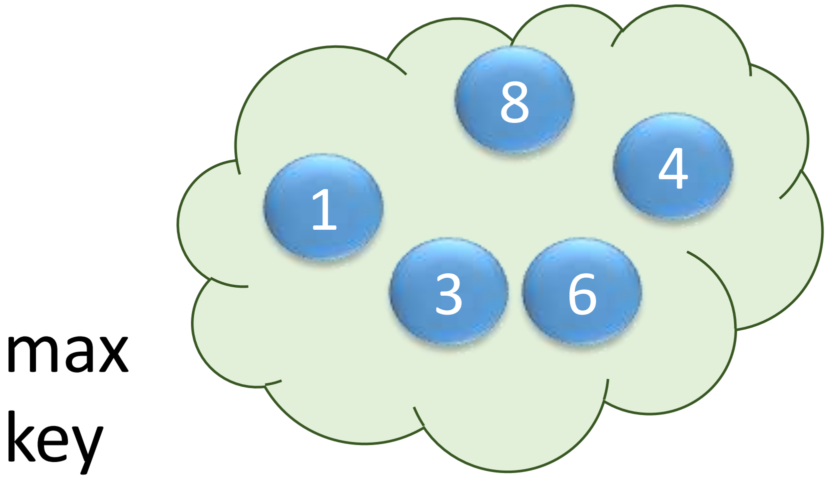
Heap Sort

- Use a max (or min) priority queue that is implemented as a heap.
- Time complexity O(n log n)
- Use the heap structure to sort elements
  - Store all the elements to the heap
  - Extract the elements from the heap one by one

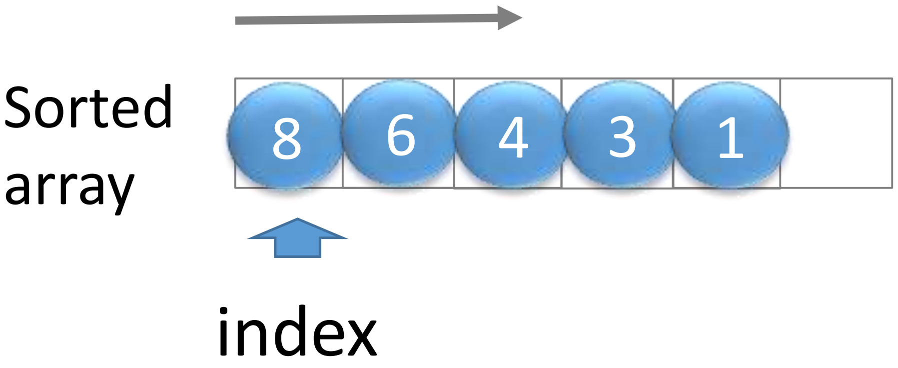
Sorting elements in descending order

Use a max priority queue

- use element key as priority
- insert elements into a priority queue
- remove/pop elements in priority order



Max priority queue



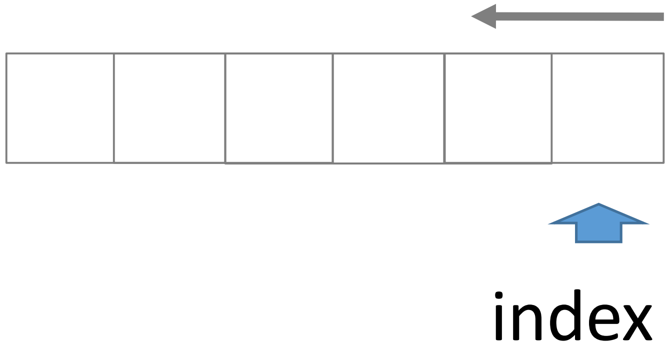
Supplemental Materials

Sorting elements in descending order

Use a max priority queue

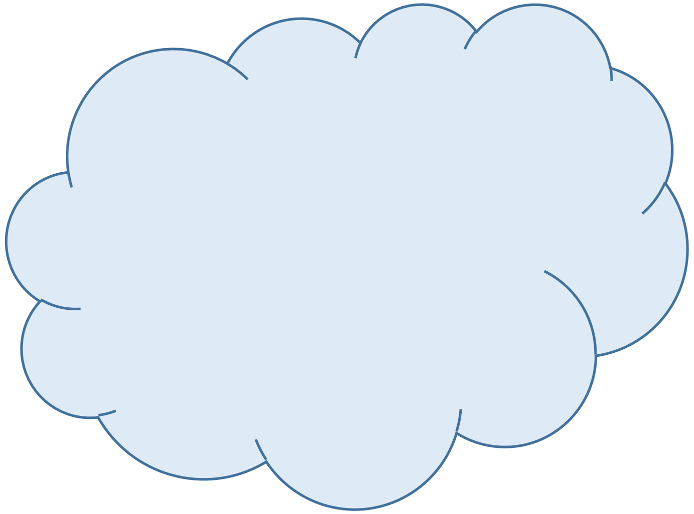
- use element key as priority
- insert elements into a priority queue
- remove/pop elements in priority order

A max priority queue can be also used for sorting elements in ascending order.



Example
Sorting in descending order

Sort five elements: 6, 8, 3, 2, 9



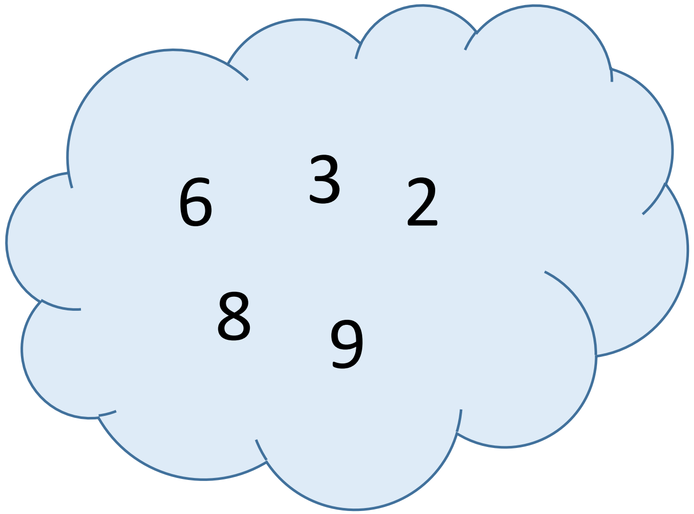
The approach:

1. Use a max priority queue. Use element key as priority
2. insert elements into a priority queue
3. remove/pop elements in priority order

Example
Sorting in ascending order

Sort five elements: 6, 8, 3, 2, 9

Elements are inserted into the max priority queue.



The approach:

1. Use a max priority queue. Use element key as priority
2. insert elements into a priority queue
3. remove/pop elements in priority order

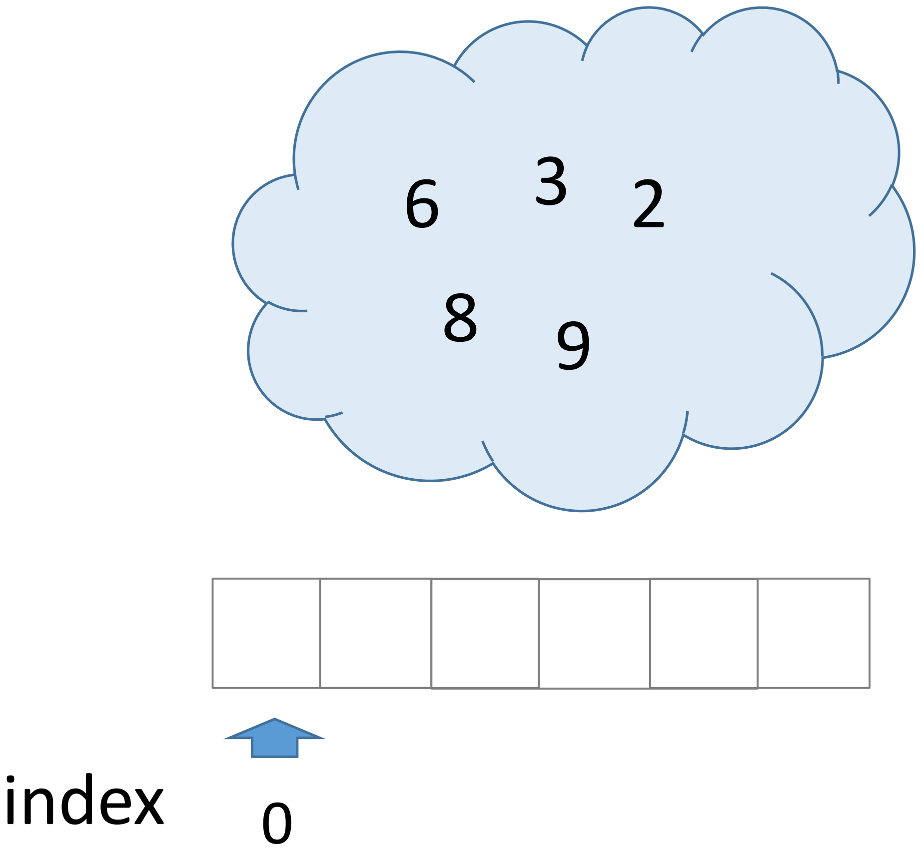
Example
Sorting in ascending order

Sort five elements: 6, 8, 3, 2, 9

Pop the elements one by one.

The approach:

1. Use a max priority queue. Use element key as priority
2. insert elements into a priority queue
3. remove/pop elements in priority order



Example
Sorting in ascending order

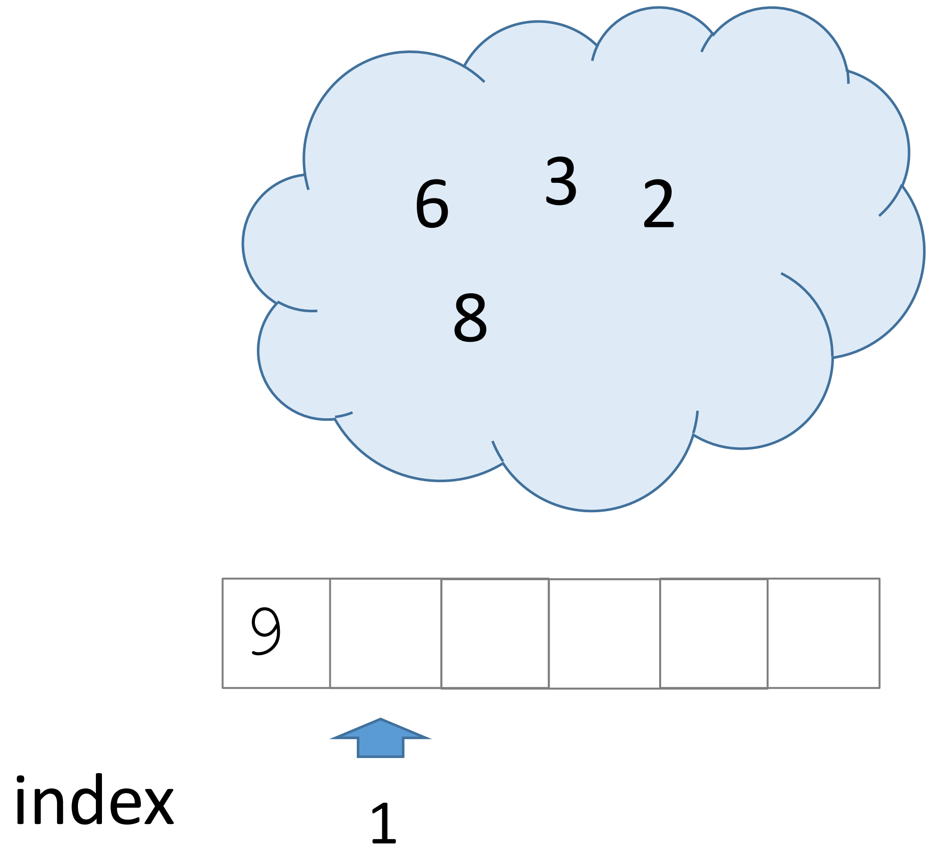
Sort five elements: 6, 8, 3, 2, 9

The approach:

1. Use a max priority queue.
   Use element key as priority
2. insert elements into a priority queue
3. remove/pop elements in priority order

Pop the elements one by one:

Step 3:1



Example
Sorting in ascending order

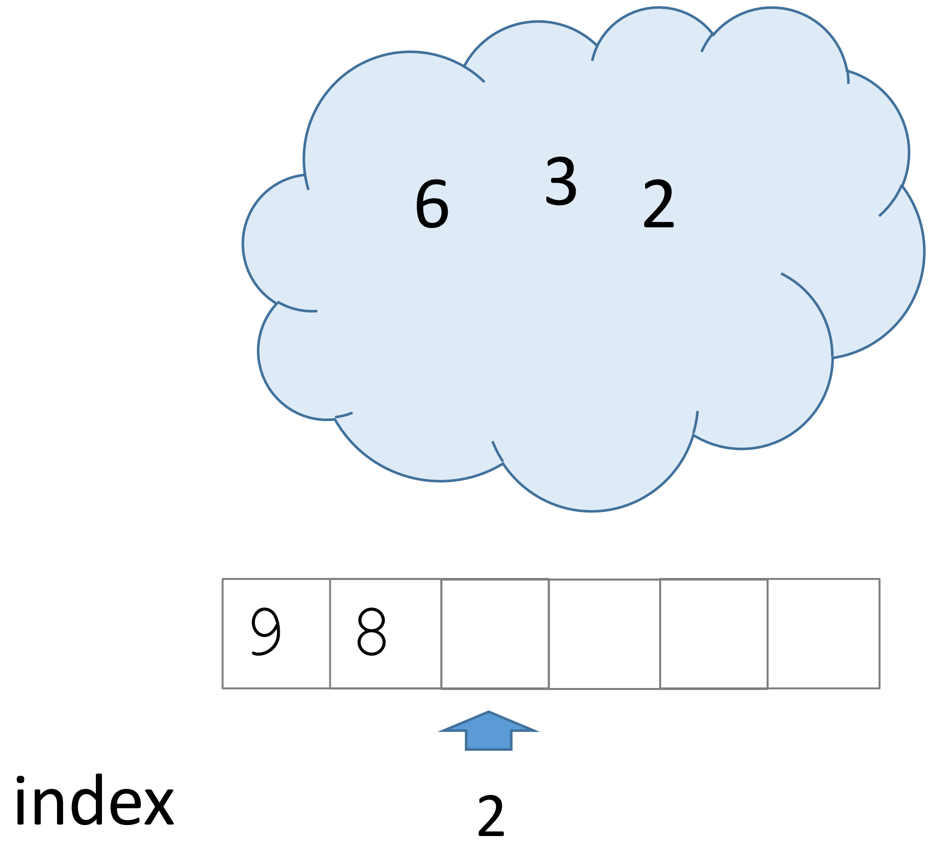
Sort five elements: 6, 8, 3, 2, 9

The approach:

1. Use a max priority queue.
   Use element key as priority
2. insert elements into a priority queue
3. remove/pop elements in priority order

Pop the elements one by one.

Step 3:2



Example
Sorting in ascending order

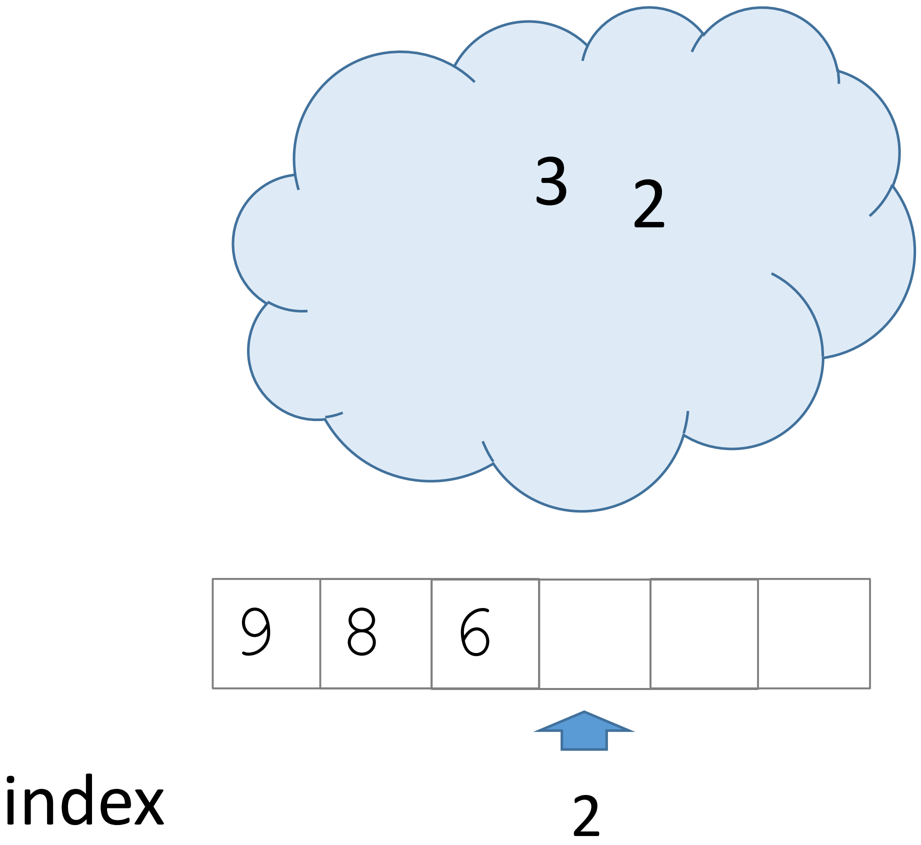
Sort five elements: 6, 8, 3, 2, 9

Pop the elements one by one.

Step 3:3

The approach:

1. Use a max priority queue.
   Use element key as priority
2. insert elements into a priority queue
3. remove/pop elements in priority order



Example
Sorting in ascending order

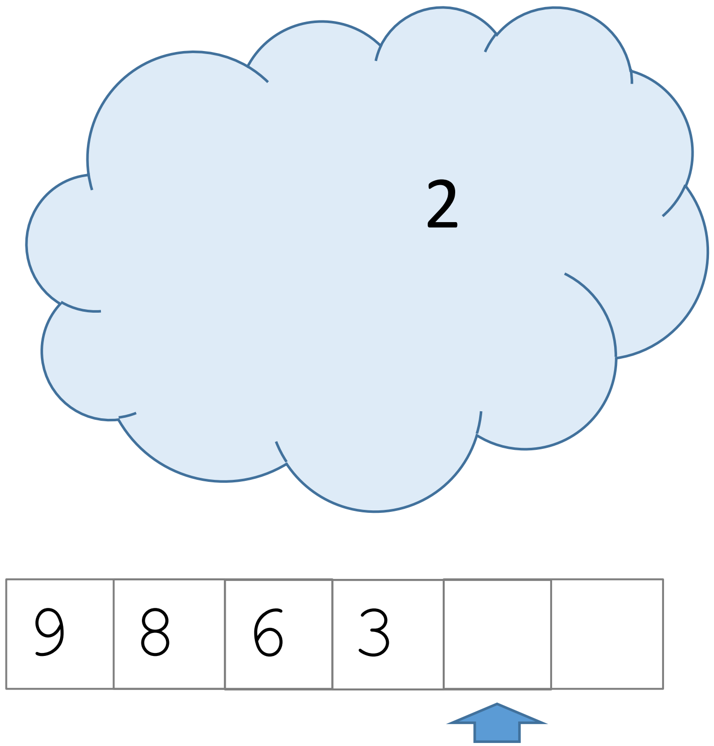
Sort five elements: 6, 8, 3, 2, 9

Pop the elements one by one.

Step 3:4

The approach:

1. Use a max priority queue. Use element key as priority
2. insert elements into a priority queue
3. remove/pop elements in priority order



index 2

Example
Sorting in ascending order

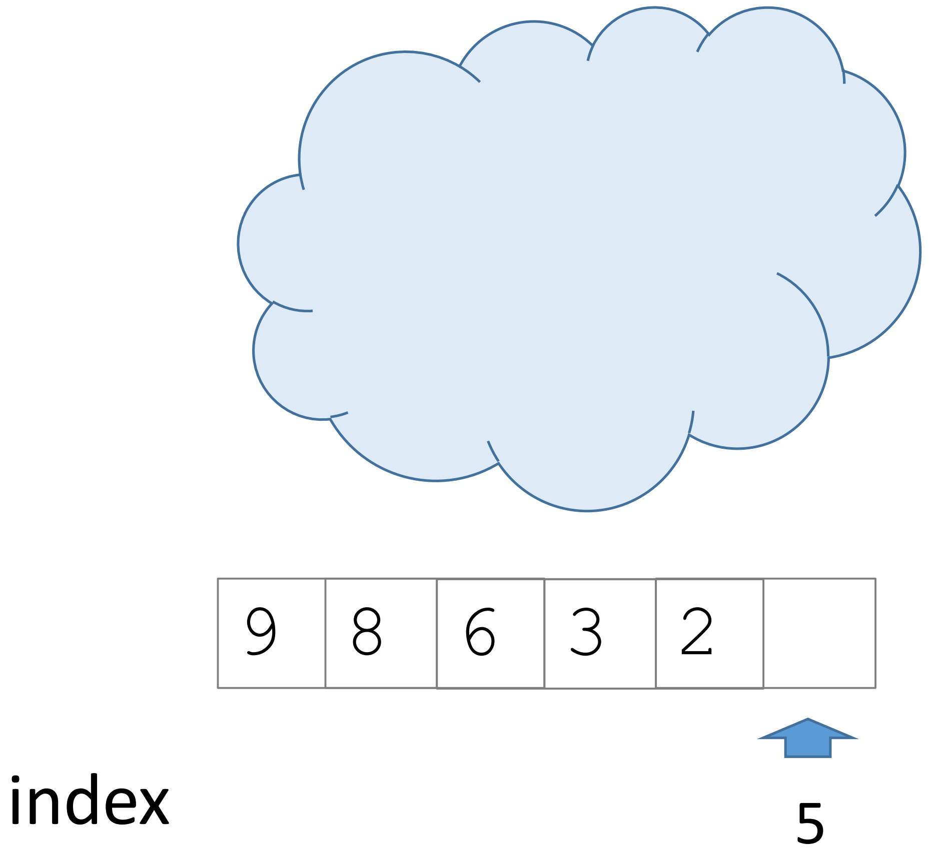
Sort five elements: 6, 8, 3, 2, 9

The approach:

1. Use a max priority queue.
   Use element key as priority
2. insert elements into a priority queue
3. remove/pop elements in priority order

Pop the elements one by one.

Step 3:5



Machine Scheduling

- m identical machines (drill press, cutter, sander, etc.)
- n jobs/tasks to be performed
- assign jobs to machines so that the time at which the last job completes is minimum

LPT Schedules

- Longest Processing Time first.
- Jobs are scheduled in the order, e.g., 24, 15, 11, 9, 7, 4, 2
- Each job is scheduled on the machine on which it finishes earliest.

LPT Schedule

- LPT rule does not guarantee minimum finish time schedules.
- Usually LPT finish time is closer to minimum finish time.
- Minimum finish time scheduling is NP-hard.

NP-hard Problems

- No algorithm whose complexity is $O(n^k)$ for any constant k is known for any NP-hard problem.
- It is unlikely that any NP-hard problem can be solved by a polynomial time algorithm.
- Adopt heuristic approaches to solve NP-hard problems.

Exercise: LPT Schedules

- Given a set of jobs, report the job assignment.
- Implement a program for performing the LPT schedule.
- Implement a heap structure.

- Input the number of jobs, n.
- Input the processing cost of each job and its name.
- Input the number of machines, m.
- Display the assignment result for each machine, e.g.,
    - total processing cost of each machine
    - the jobs assigned on each machine