# Computational Complexity

Problem Solving and Programming Techniques 2014: Lectured by Dr. Min-Zheng Shieh @ I²CRC, NCTU.

# Computational complexity

- Complexity of a program P: the resources required to finish the execution of P.
  - Time complexity:
    the running time of P
  - Space complexity:
    the extra memory used by P
  - Kolmogorov complexity:
    the code length of P

# Asymptotic notation

- To precisely measure complexity is hard.
- For large enough n:
  - $T(n)=O(f(n))$: $T(n)\leq c_1 f(n)$
  - $T(n)=\Omega(f(n))$: $T(n)\geq c_2 f(n)$
  - $T(n)=\Theta(f(n))$: $c_2 f(n)\leq T(n) \leq c_1 f(n)$
- $T(n)=o(f(n))$: $\lim_{n\to\infty} T(n)/f(n)=0$
- $T(n)=\omega(f(n))$: $\lim_{n\to\infty} f(n)/T(n)=0$

$c_1$ and $c_2$ are constatnts

# Asymptotic notation

| small-ω | big-Ω | Θ | big-O | small-o |
|---|---|---|---|---|
| > | ≥ | = | ≤ | < |

# Sorting problem

- Given a sequence, rearrange it into a particular order.
- Example: ascending order
  - Input: 0,3,5,7,1,2,1,2,1
  - Output: 0,1,1,1,2,2,3,5,7
- Different programs/algorithms/DSs may have different complexity.

# Selection sort

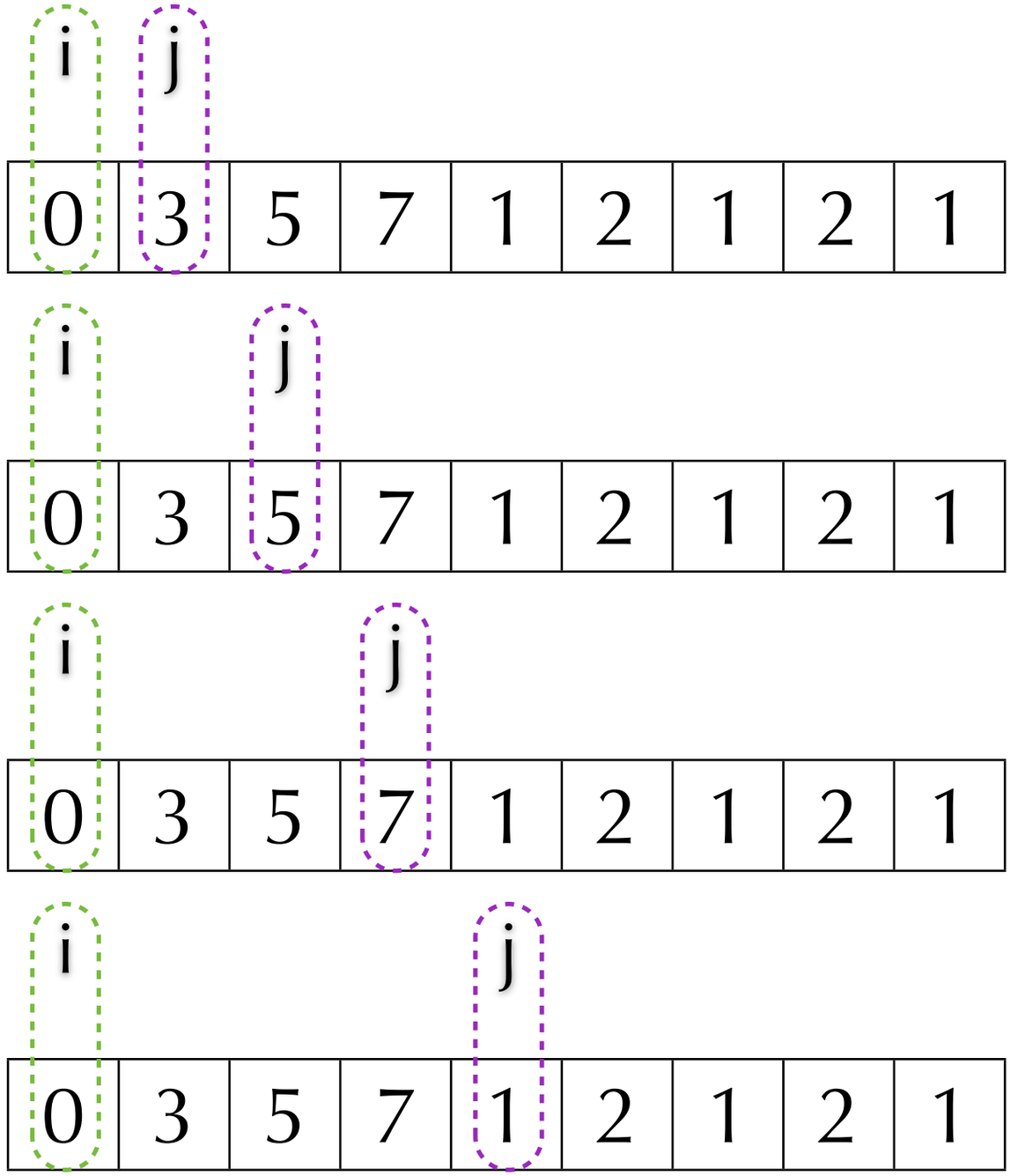
- Maintain a list L
- Repeat appending the smallest number in S to L until S is empty.
- makeEmptyList(L)

```
S=makeSet(a)
while S is not empty do
    L.append(extractMin(S))
loop
output L
```

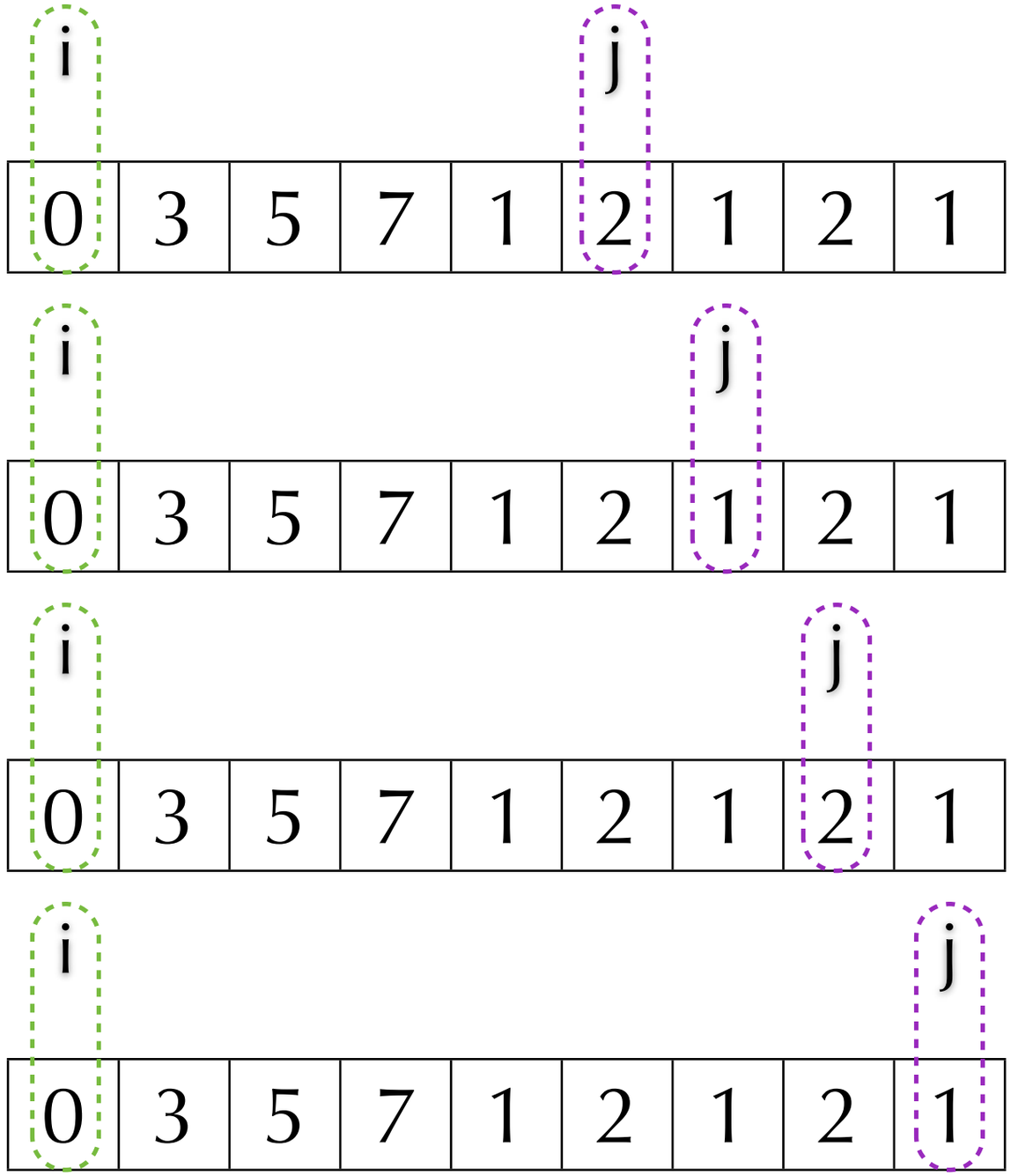
# Implementation by array

```
for i=1 to n-1 do
    for j = i+1 to n do
        if a[j]<a[i] then
            swap(a[i],a[j])
    loop
loop
output a
```

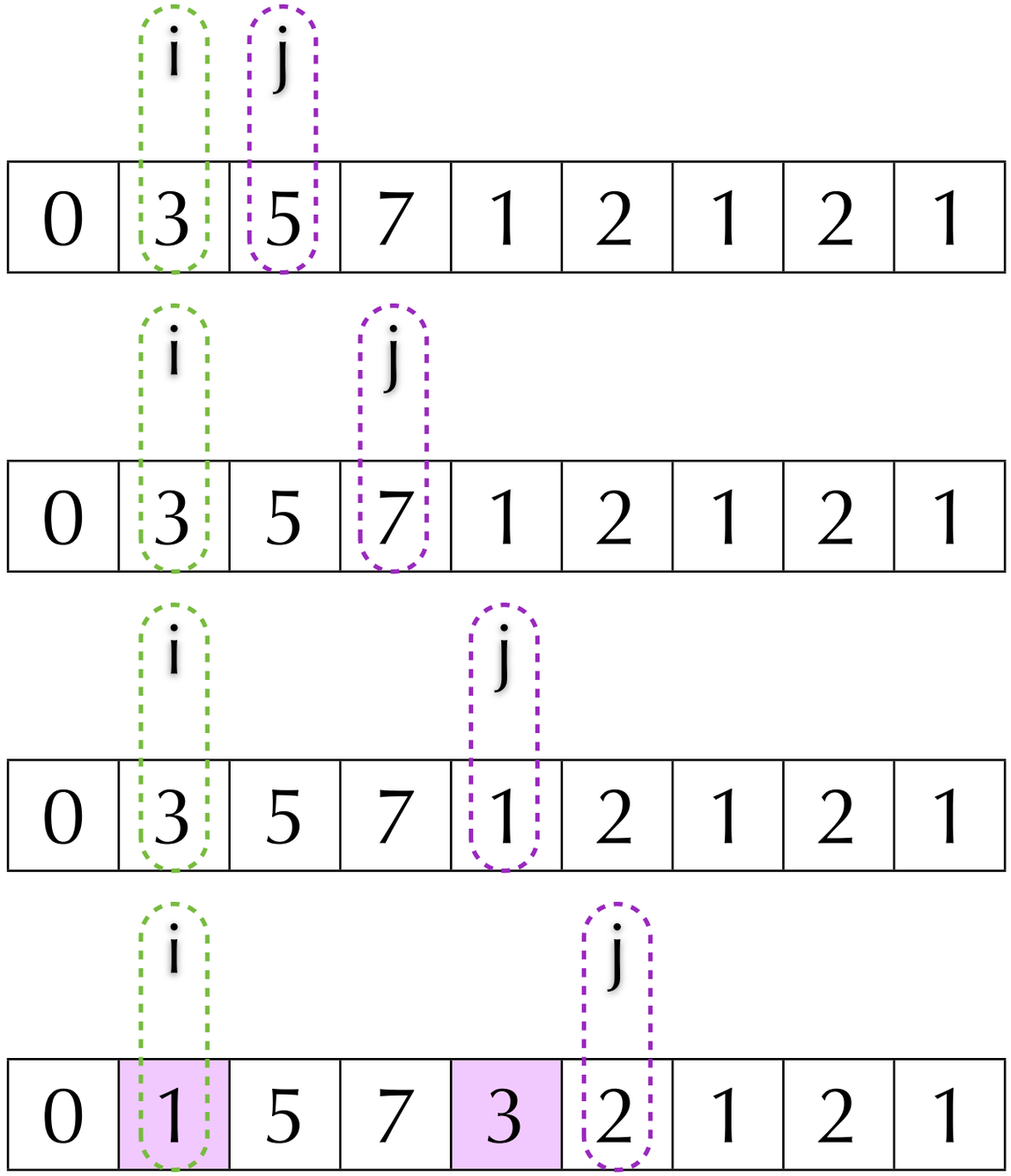
# Selection Sort



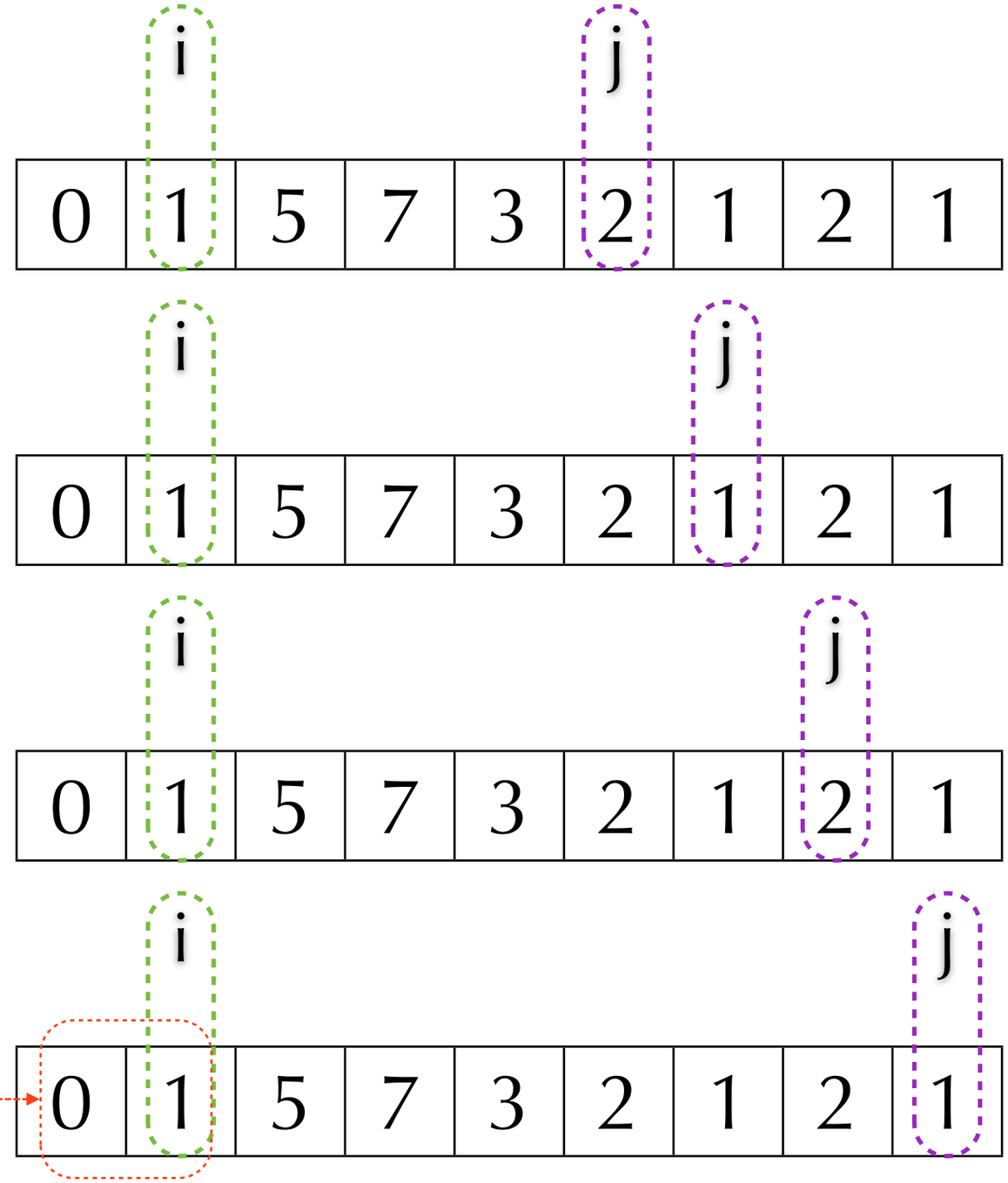
# Selection Sort



# Selection Sort



# Selection Sort



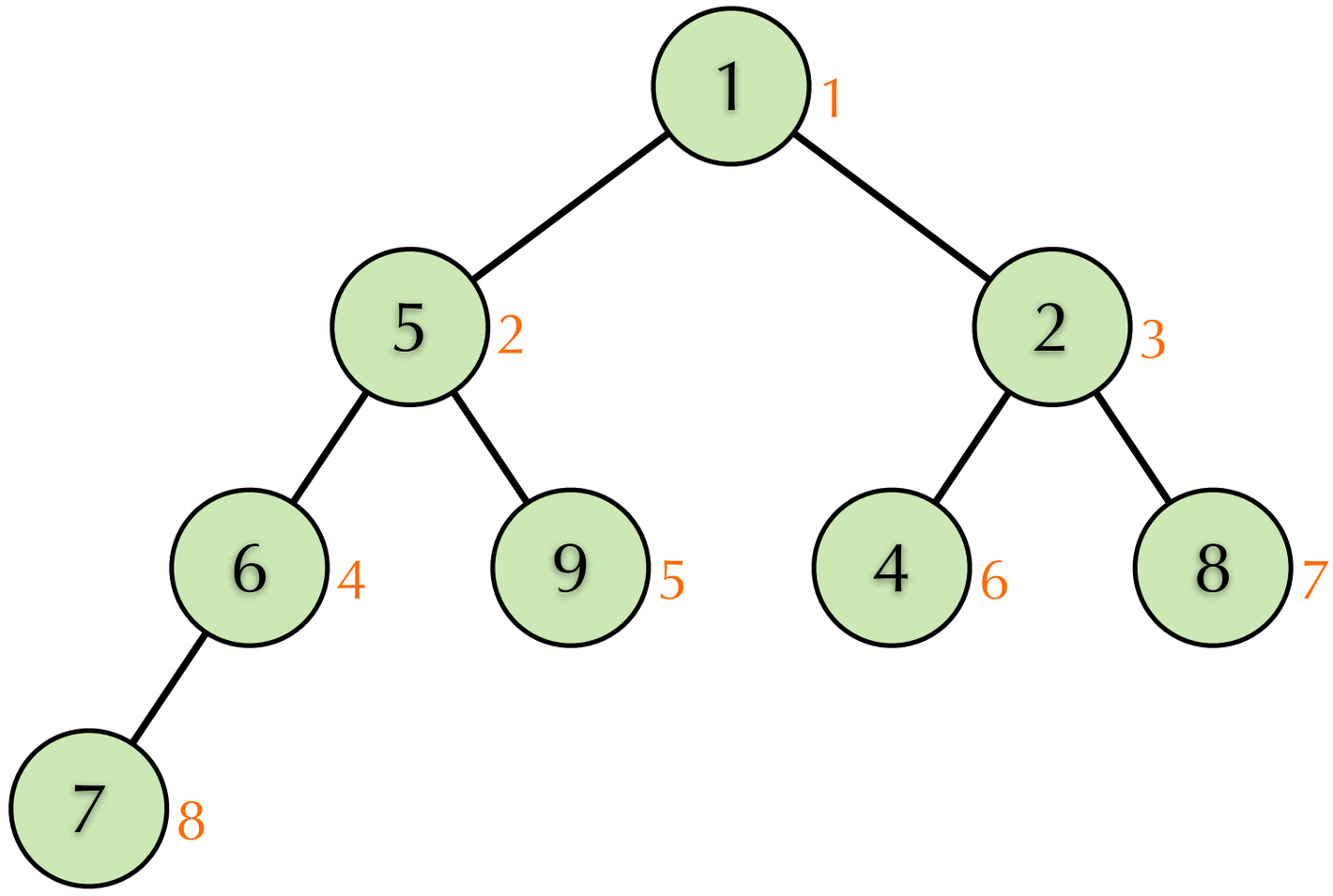
# Time complexity

- #Comparisons: $(n-1)+\ldots+1=(n^2-n)/2$
- #Swaps:
  - no more than #comparisons
    - possible 0
- Time complexity: $O(n^2)$

# Heap sort

- Use a binary heap to implement the set S
- Binary (min) heap: a rooted tree structure
  - Every node v has a key $k_v$.
    - If v is u's parent, then $k_v \geq k_u$.
- The minimum key in the heap is $k_r$ where r is the root of the heap.
- A binary heap of height h has at least $2^{h-1}$ nodes and at most $2^h-1$ nodes.

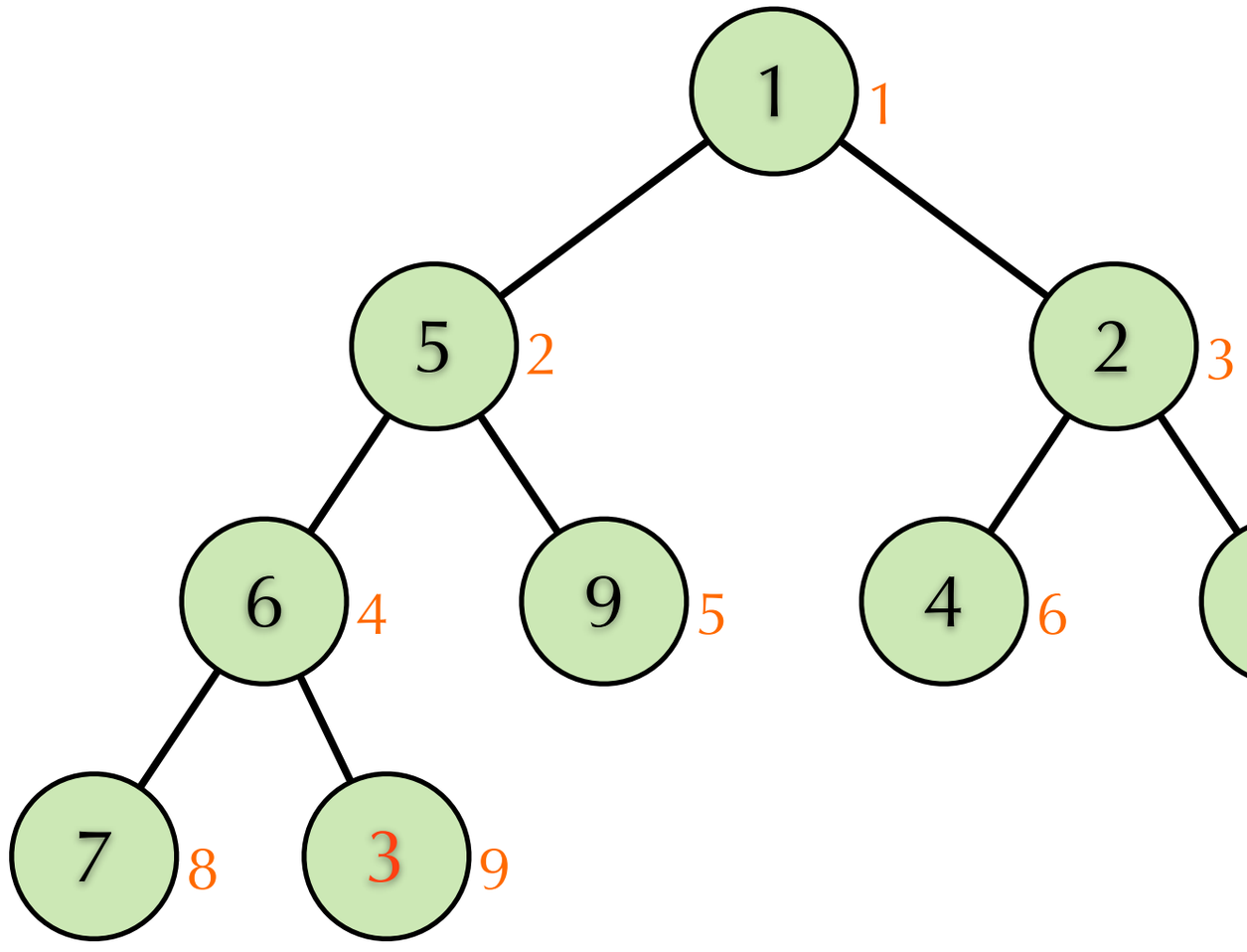
# Example: binary heap




| 1 | 5 | 2 | 6 | 9 | 4 | 8 | 7 | | |
|---|---|---|---|---|---|---|---|---|---|

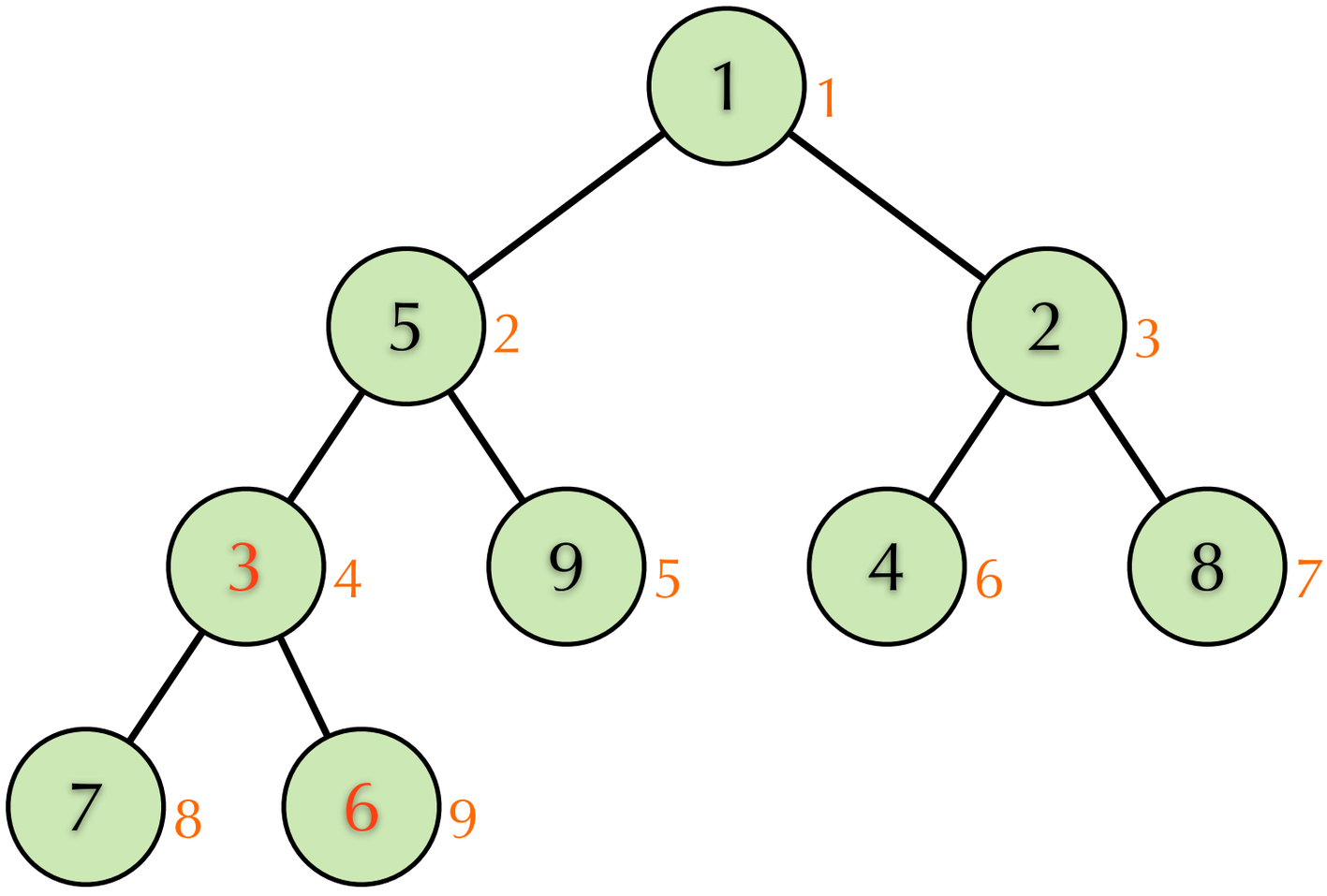
$2^{h-1} \le n \le 2^h - 1$ implies $h = \Theta(\log n)$.

# Insertion




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

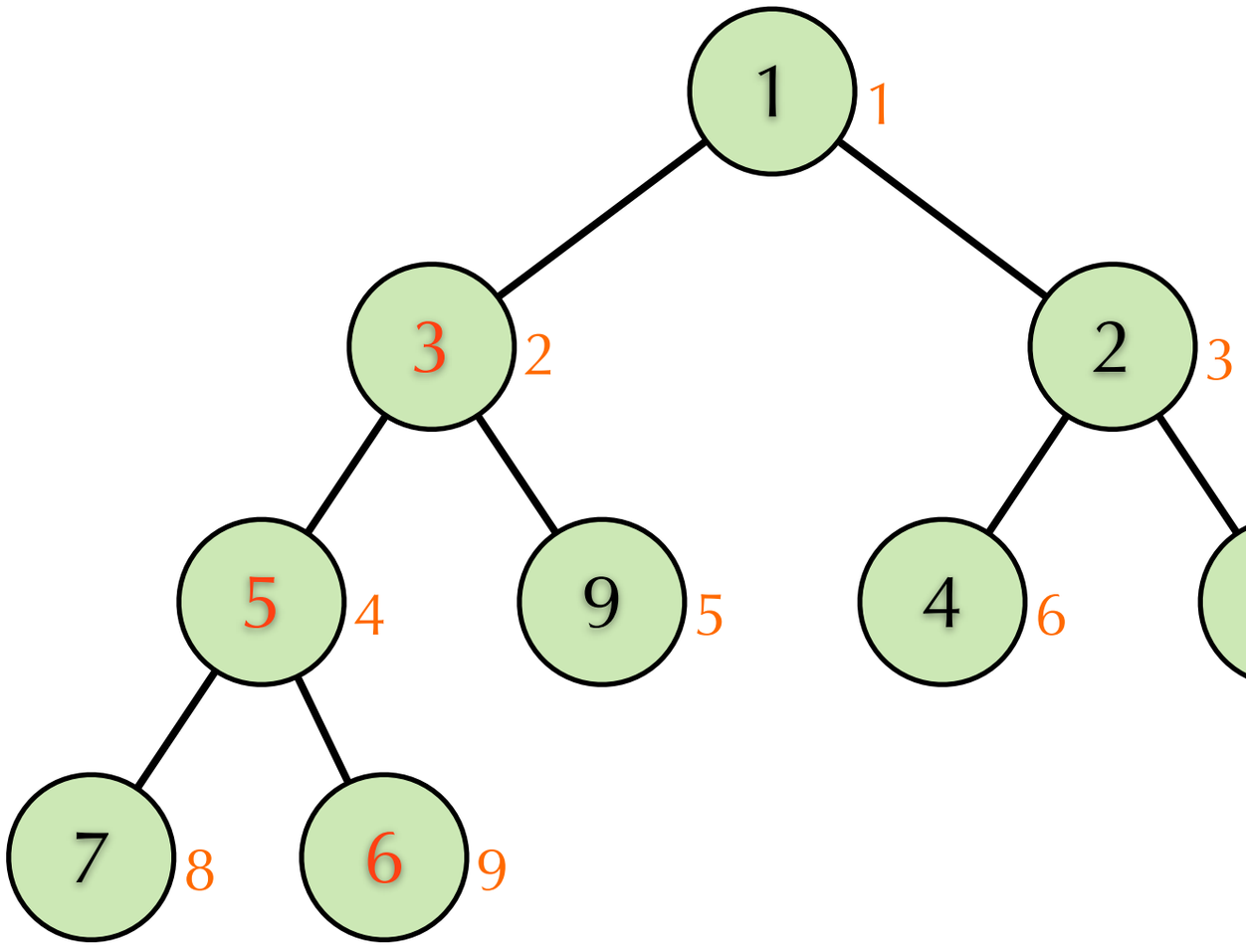
# Insertion




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

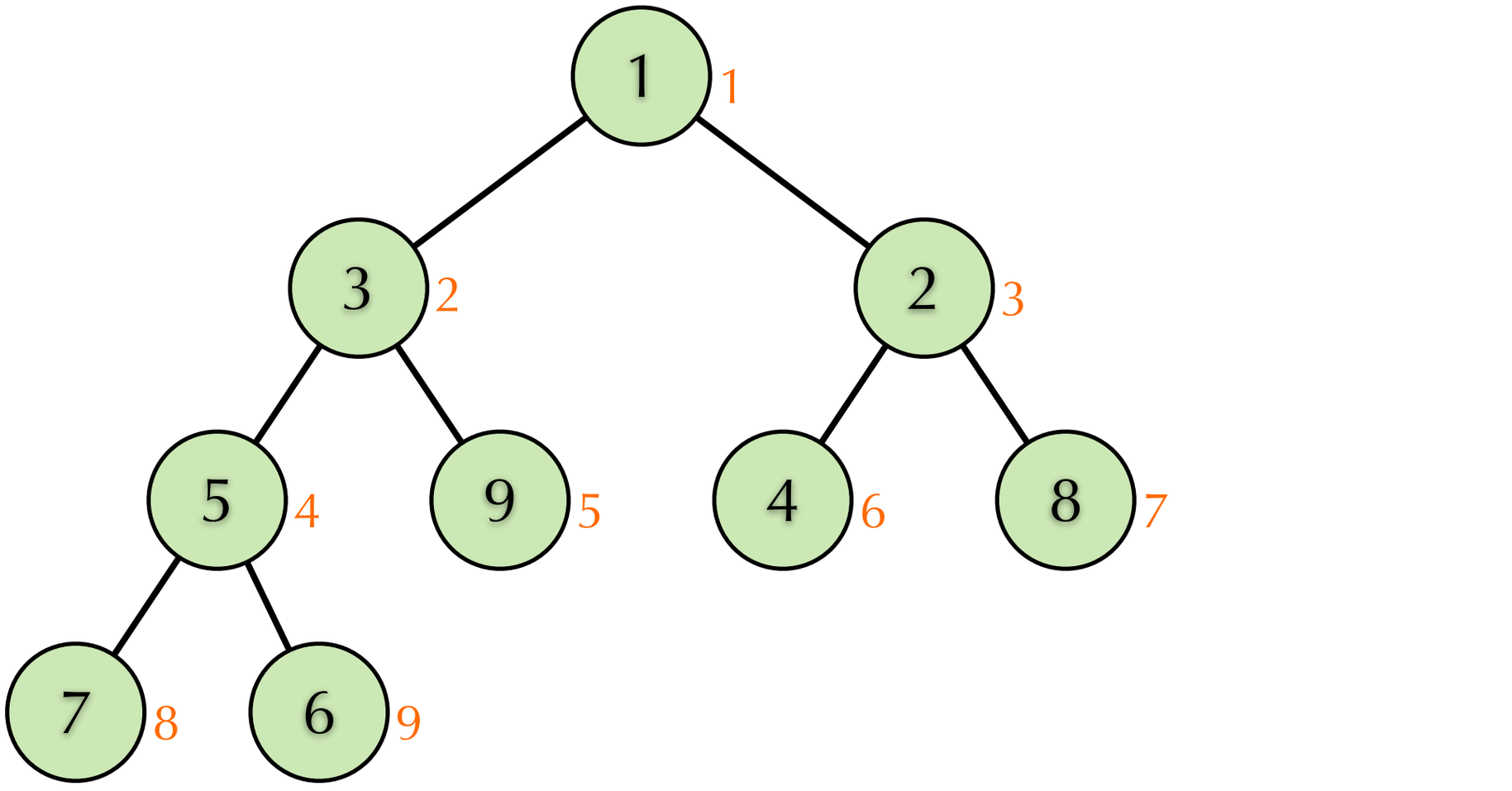
# Insertion

O(h)=O(logn)



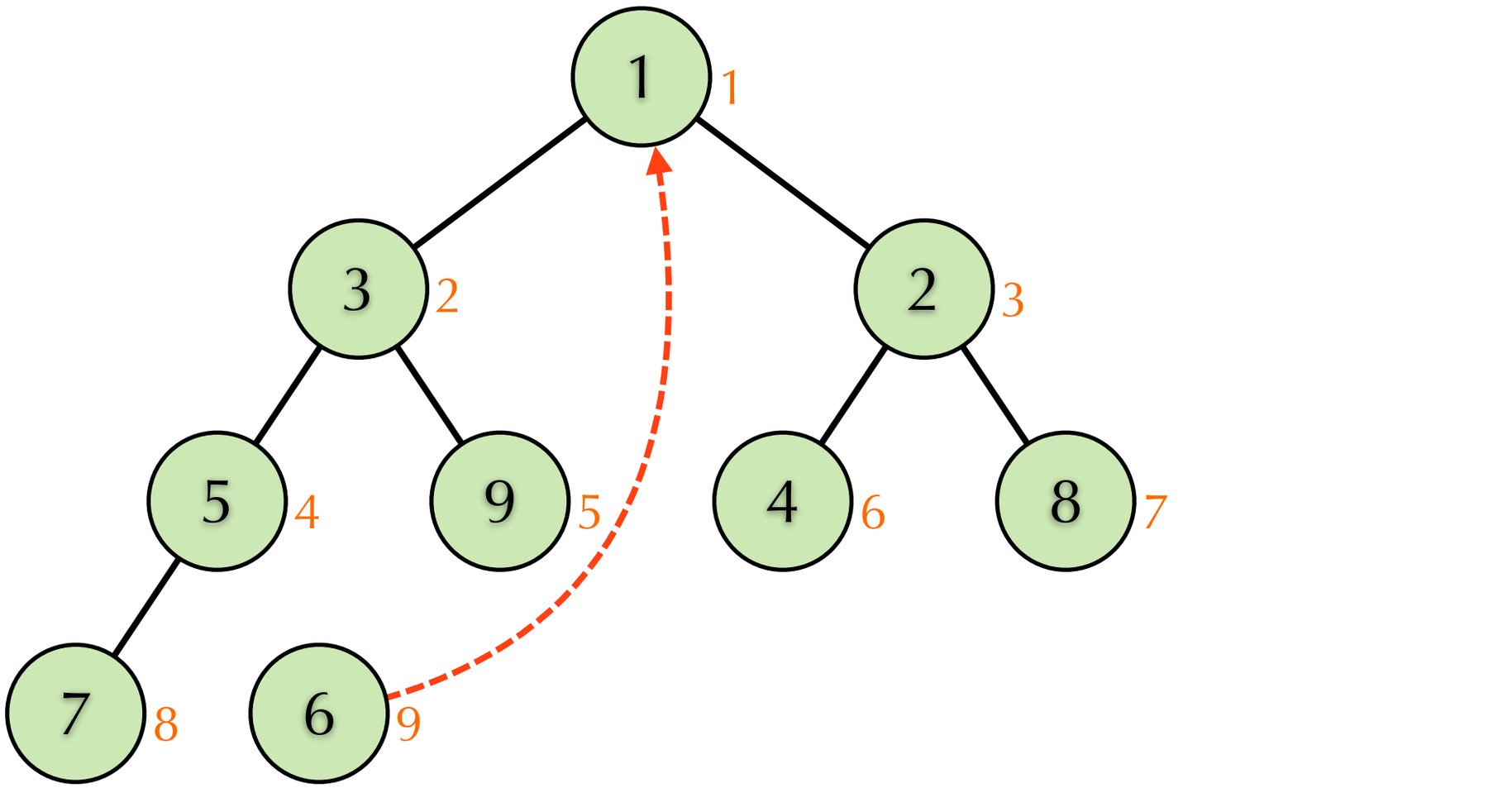

| 1 | 3 | 2 | 5 | 9 | 4 | 8 | 7 | 6 | |
|---|---|---|---|---|---|---|---|---|---|

# Extract minimum



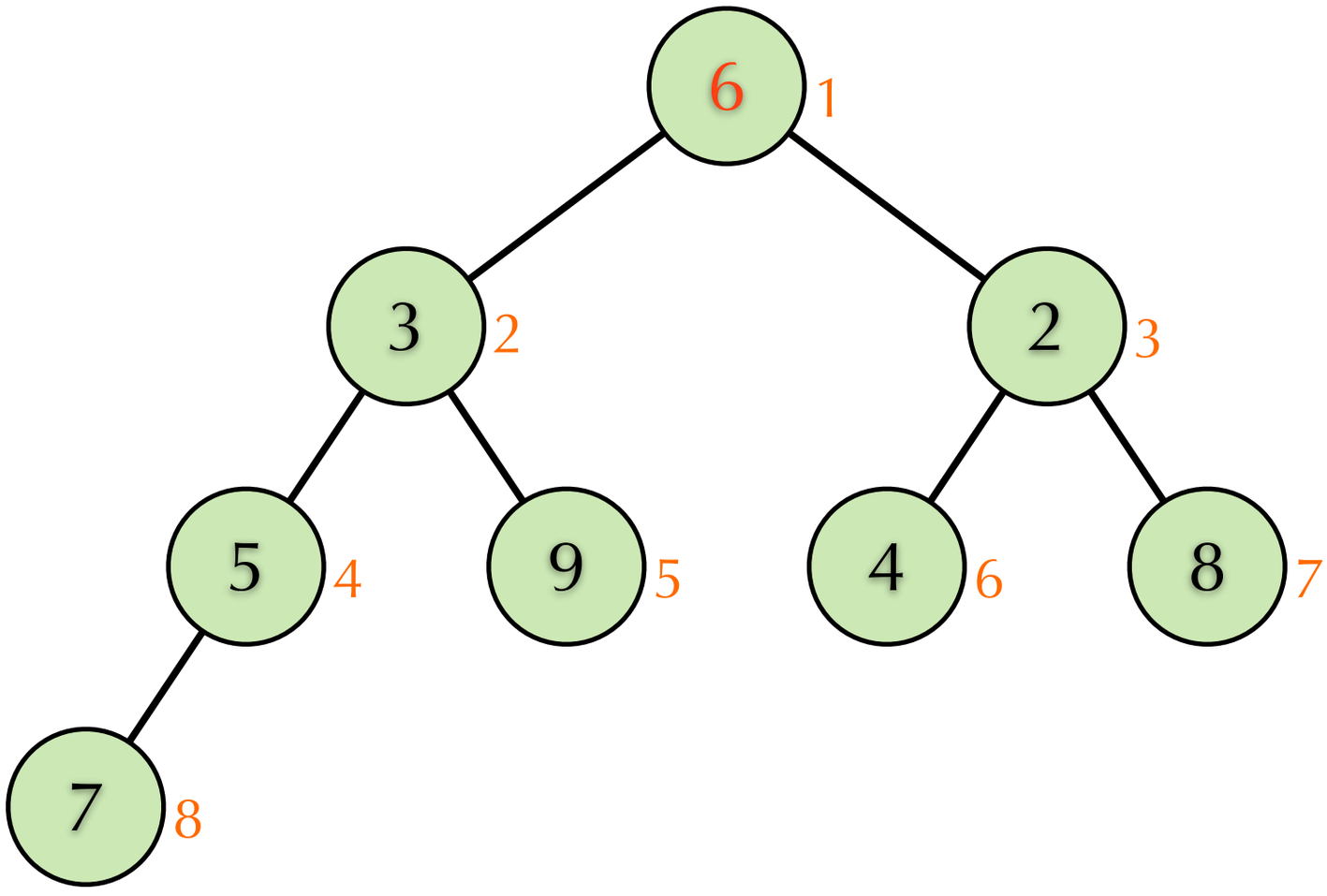

| 1 | 3 | 2 | 5 | 9 | 4 | 8 | 7 | 6 | |
|---|---|---|---|---|---|---|---|---|---|

# Extract minimum



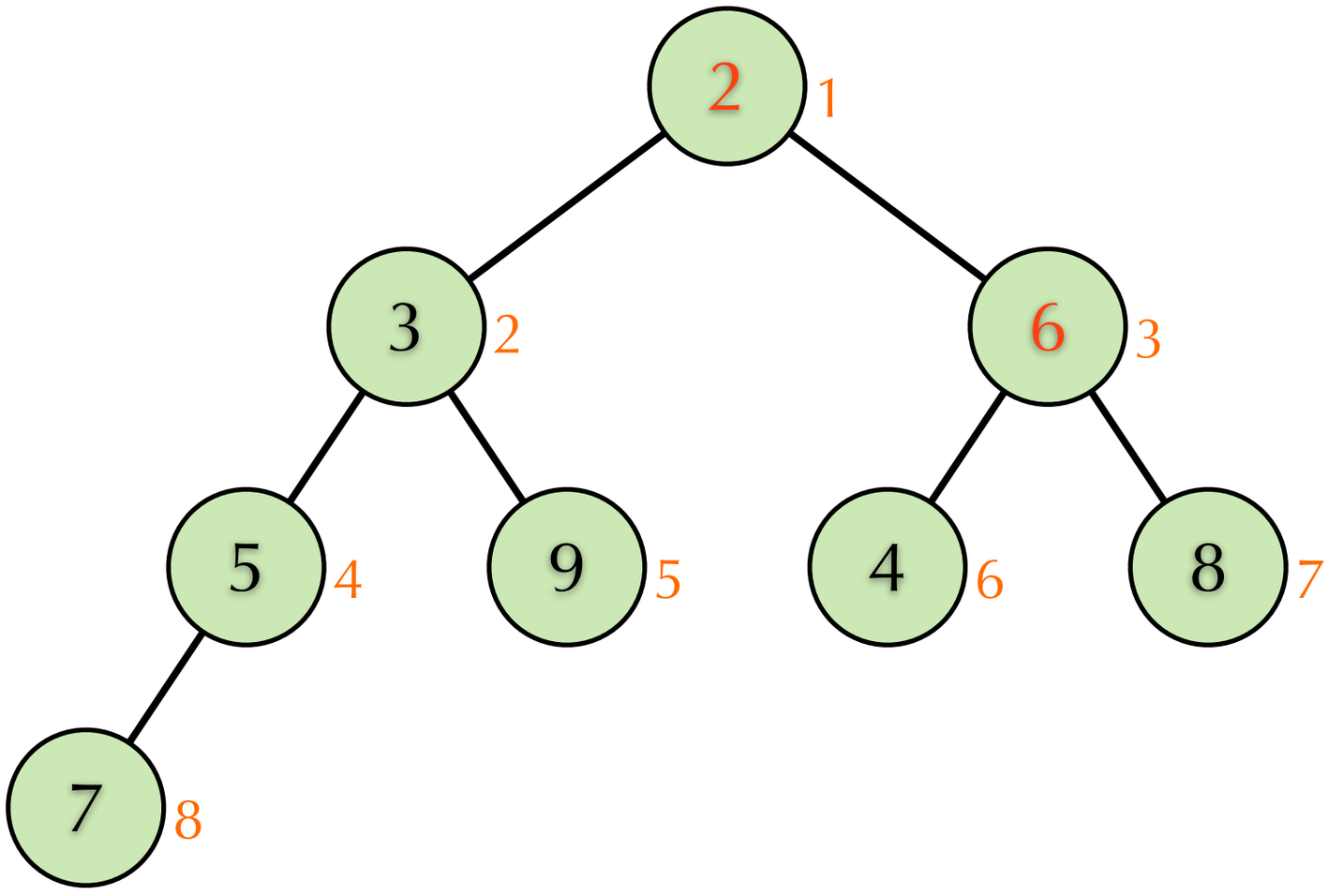

| 1 | 3 | 2 | 5 | 9 | 4 | 8 | 7 | 6 | |
|---|---|---|---|---|---|---|---|---|---|

# Extract minimum




| 6 | 3 | 2 | 5 | 9 | 4 | 8 | 7 | | |
|---|---|---|---|---|---|---|---|---|---|

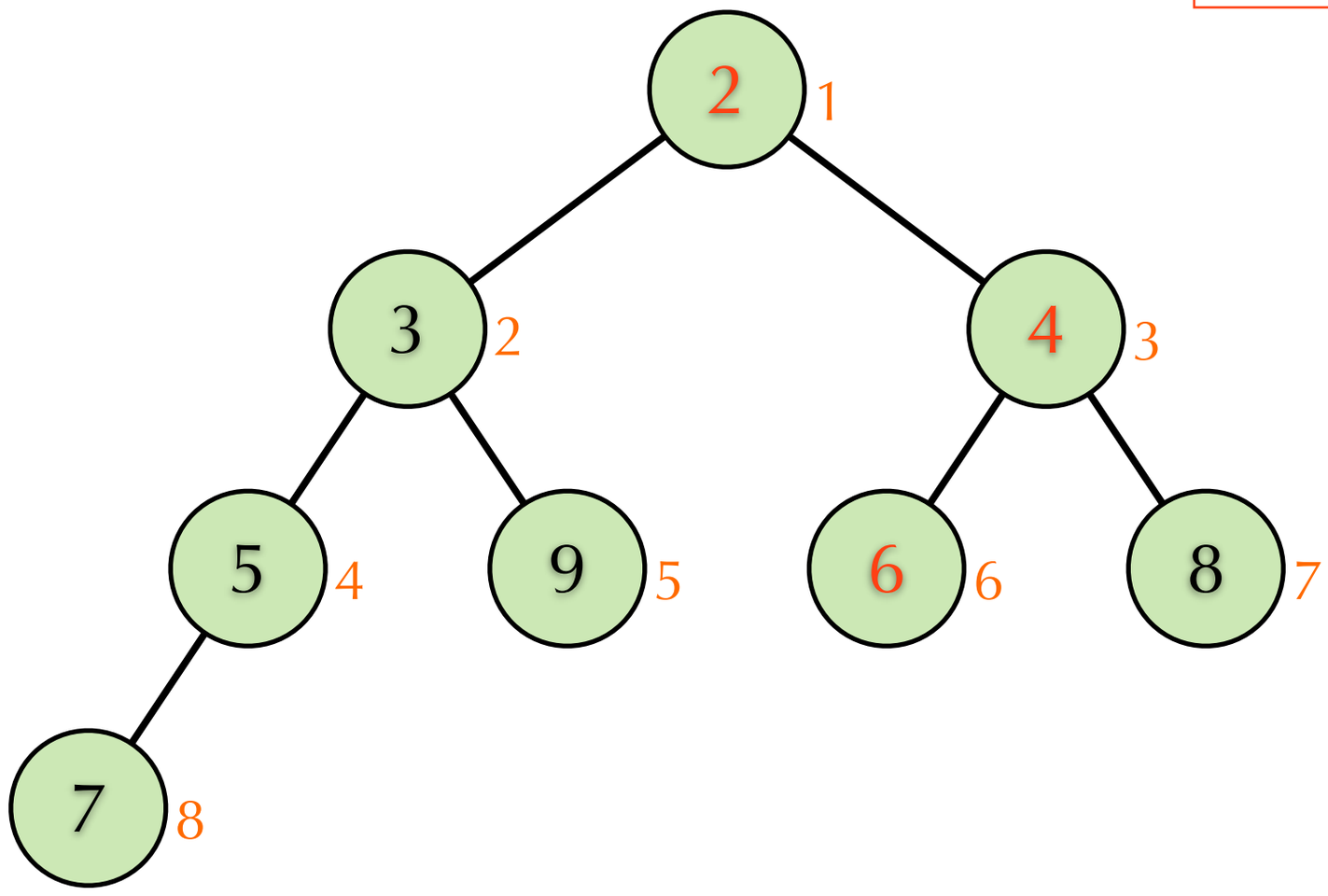
# Extract minimum




| 2 | 3 | 6 | 5 | 9 | 4 | 8 | 7 | | |
|---|---|---|---|---|---|---|---|---|---|

# Extract minimum

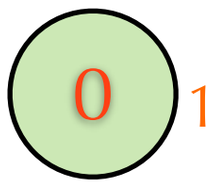
O(h)=O(logn)

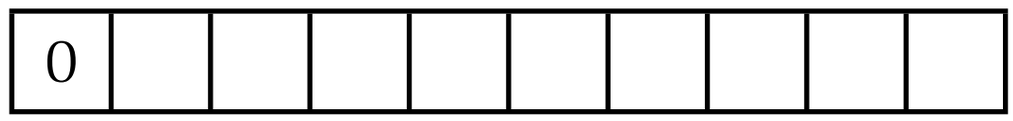



| 2 | 3 | 4 | 5 | 9 | 6 | 8 | 7 | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: makeSet

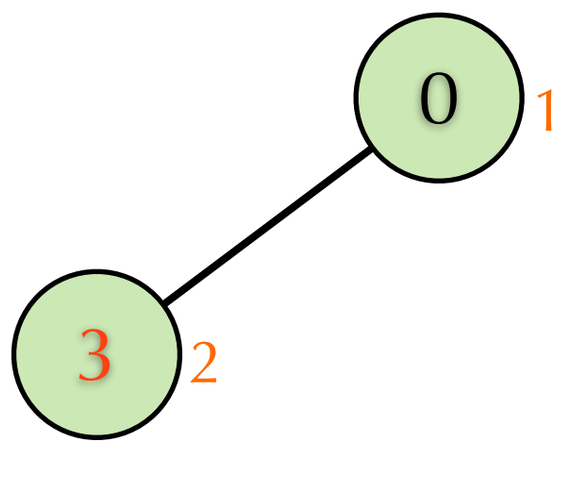
0,3,5,7,1,2,1,2,1



# Heap sort: makeSet

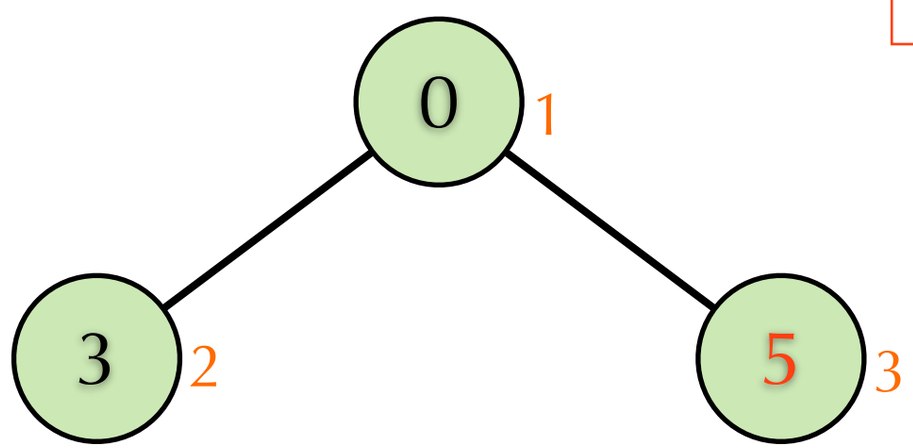
0,3,5,7,1,2,1,2,1




| 0 | 3 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: makeSet

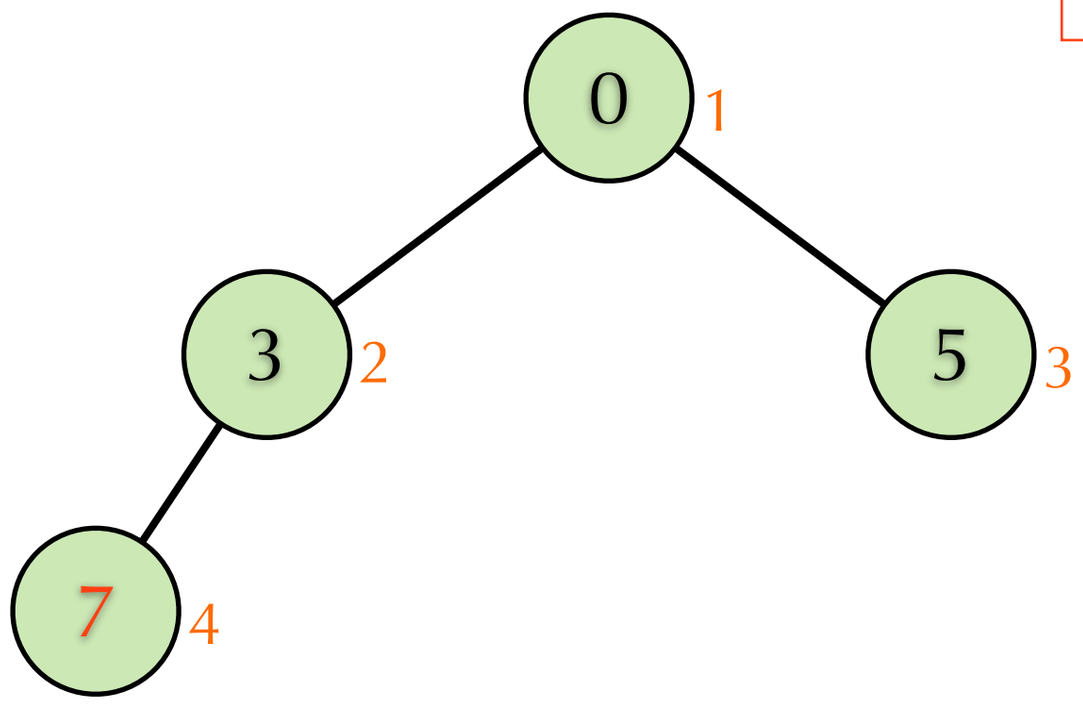
0,3,5,7,1,2,1,2,1




| 0 | 3 | 5 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: makeSet

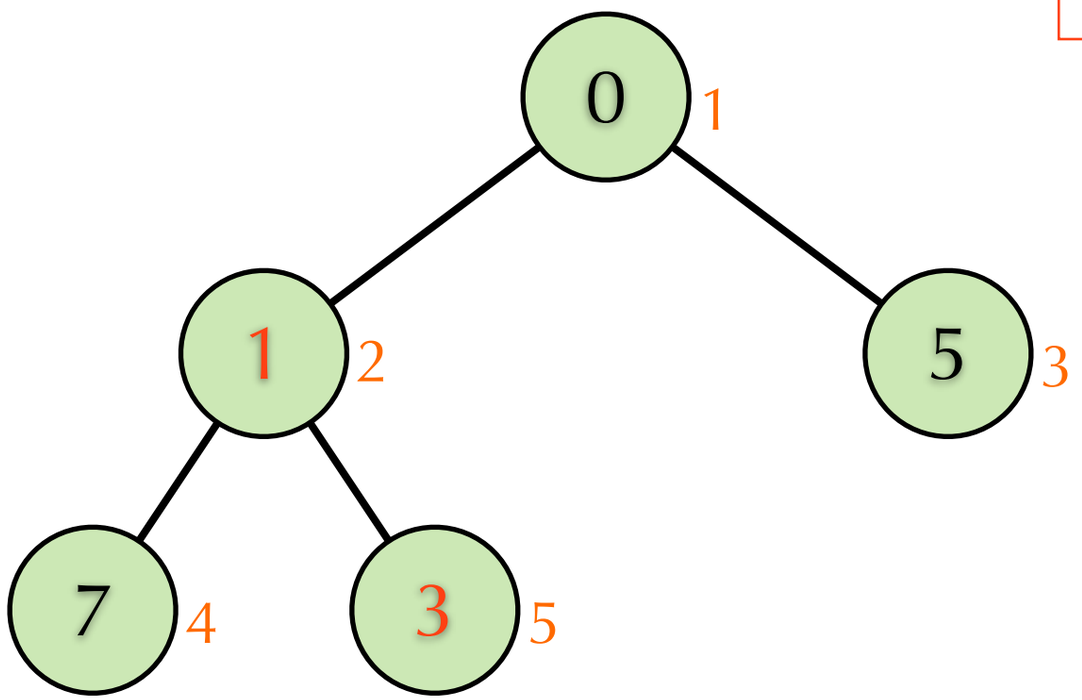
0,3,5,7,1,2,1,2,1




| 0 | 3 | 5 | 7 | | | | | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: makeSet

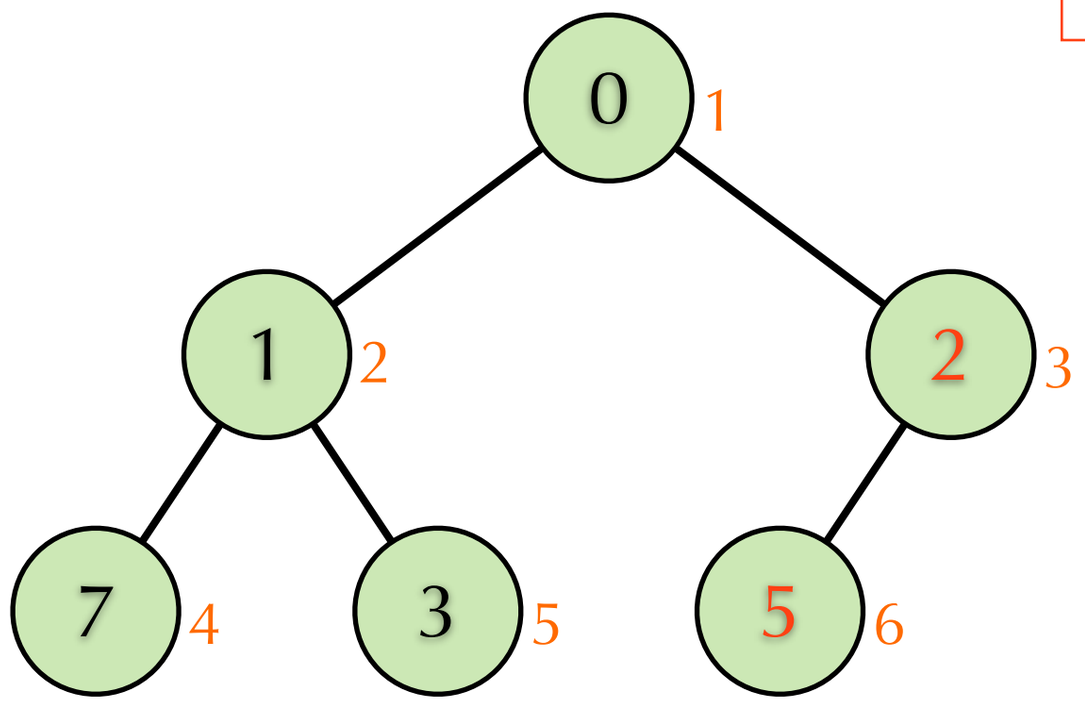
0,3,5,7,1,2,1,2,1




| 0 | 1 | 5 | 7 | 3 | | | | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: makeSet

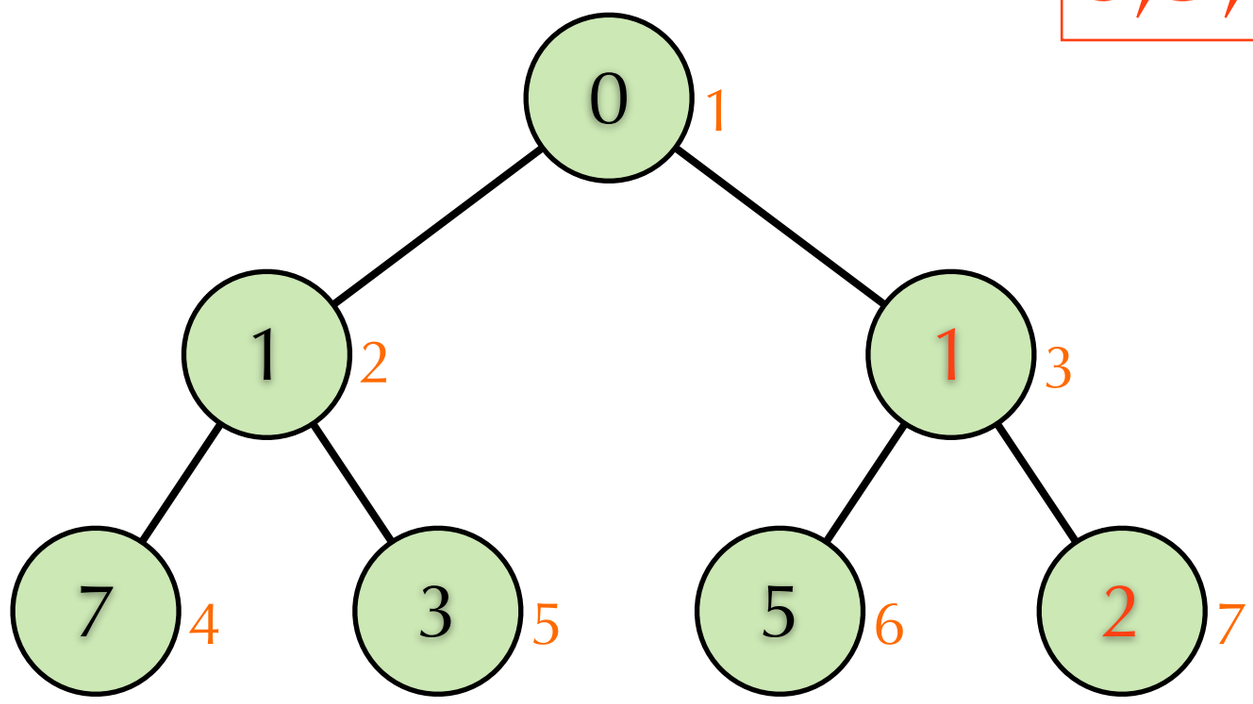
0,3,5,7,1,2,1,2,1




| 0 | 1 | 2 | 7 | 3 | 5 | | | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: makeSet

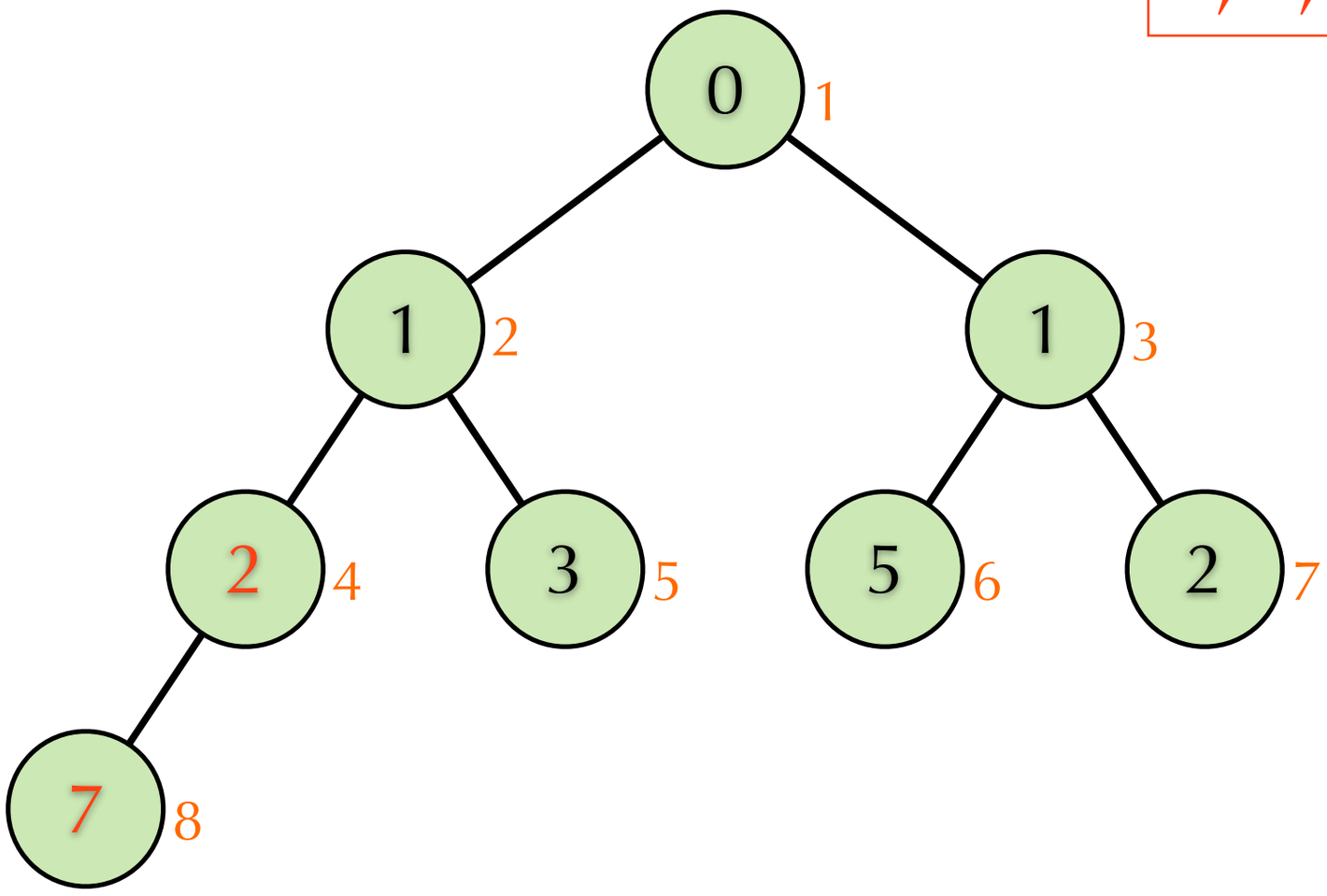
0,3,5,7,1,2,1,2,1




| 0 | 1 | 1 | 7 | 3 | 5 | 2 | | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: makeSet

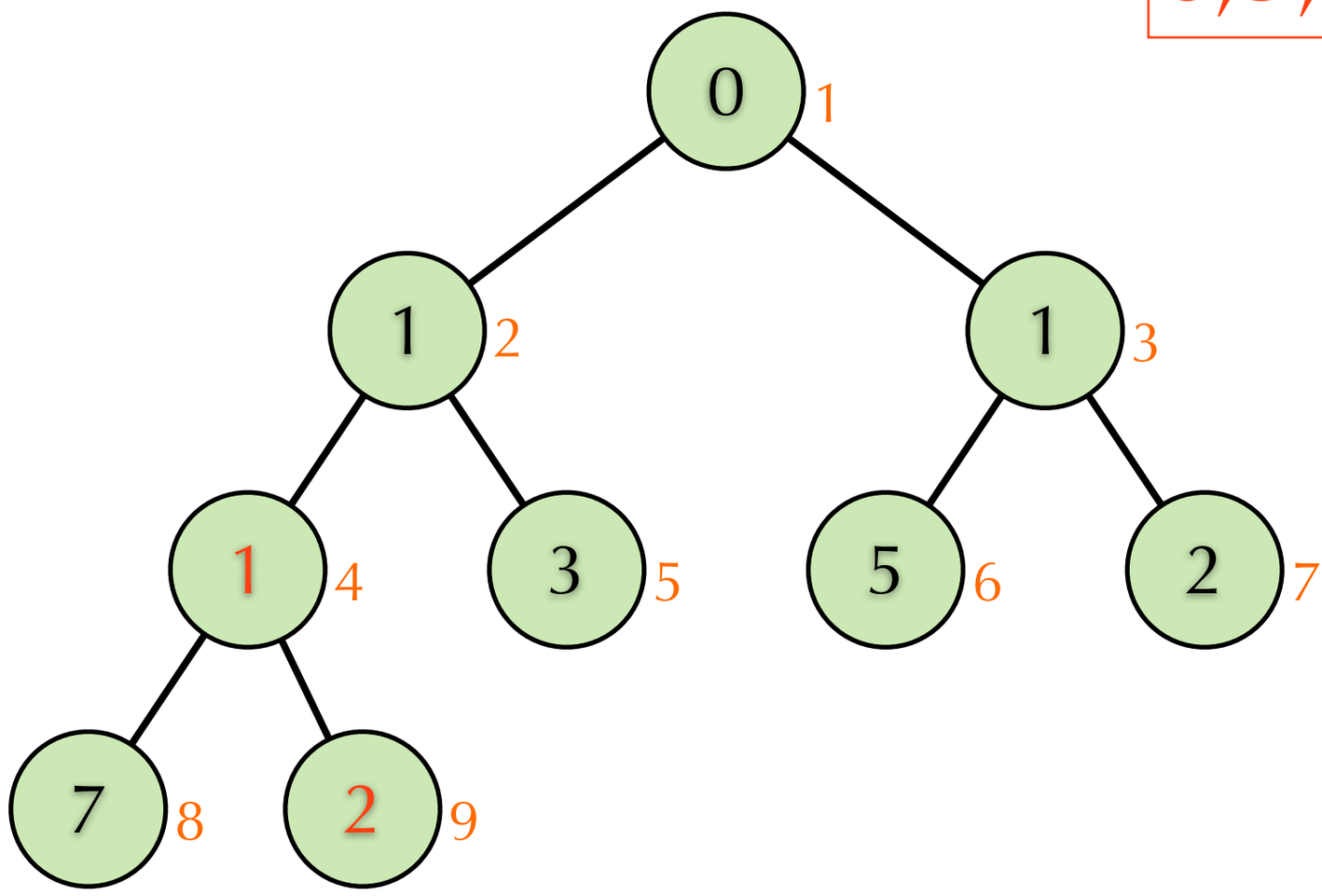
0,3,5,7,1,2,1,2,1




| 0 | 1 | 1 | 2 | 3 | 5 | 2 | 7 | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: makeSet

0,3,5,7,1,2,1,2,1

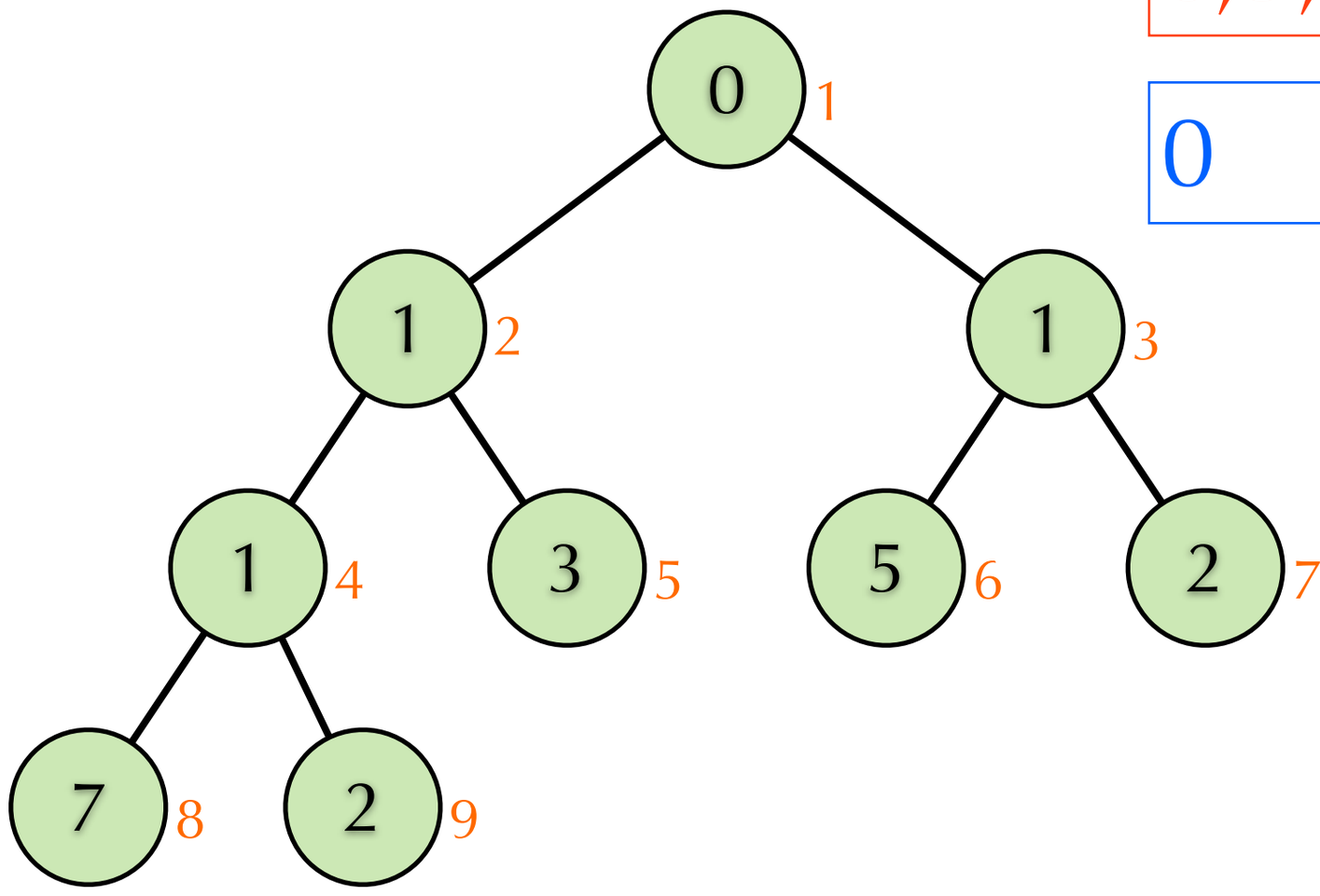



| 0 | 1 | 1 | 1 | 3 | 5 | 2 | 7 | 2 | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: ExtractMin

0,3,5,7,1,2,1,2,1

0

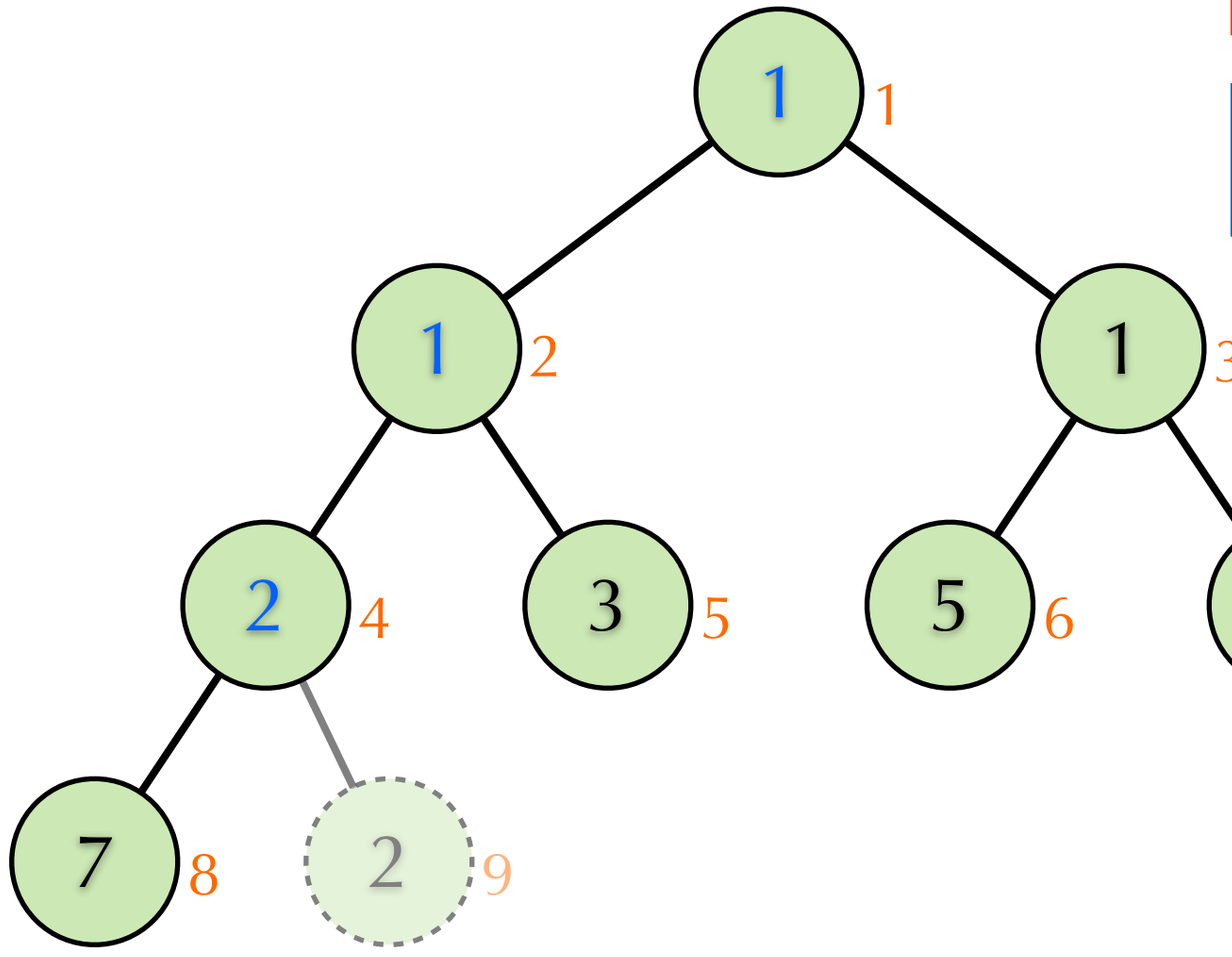



| 0 | 1 | 1 | 1 | 3 | 5 | 2 | 7 | 2 | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: ExtractMin

0,3,5,7,1,2,1,2,1

0,1

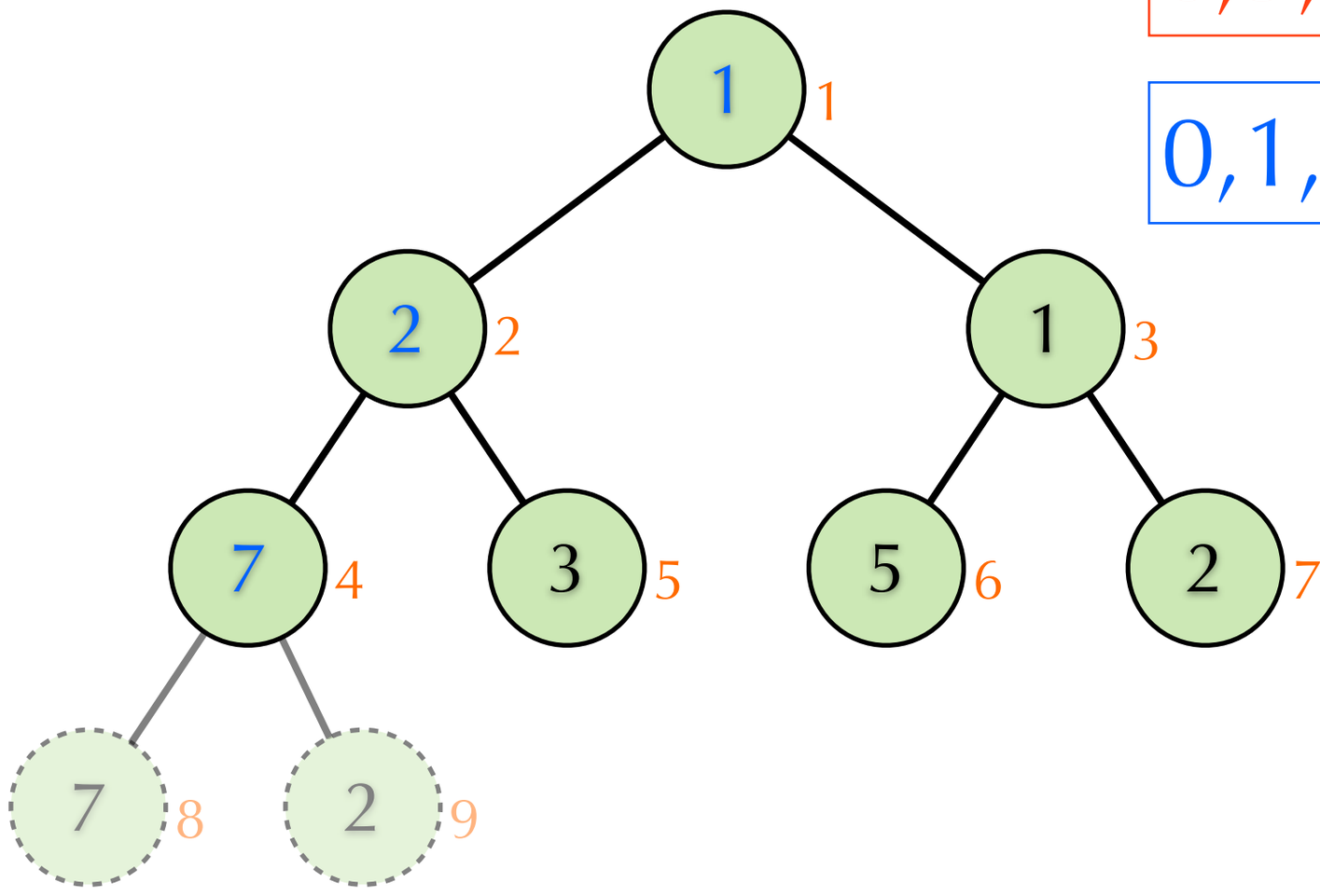



| 1 | 1 | 1 | 2 | 3 | 5 | 2 | 7 | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: ExtractMin

0,3,5,7,1,2,1,2,1

0,1,1

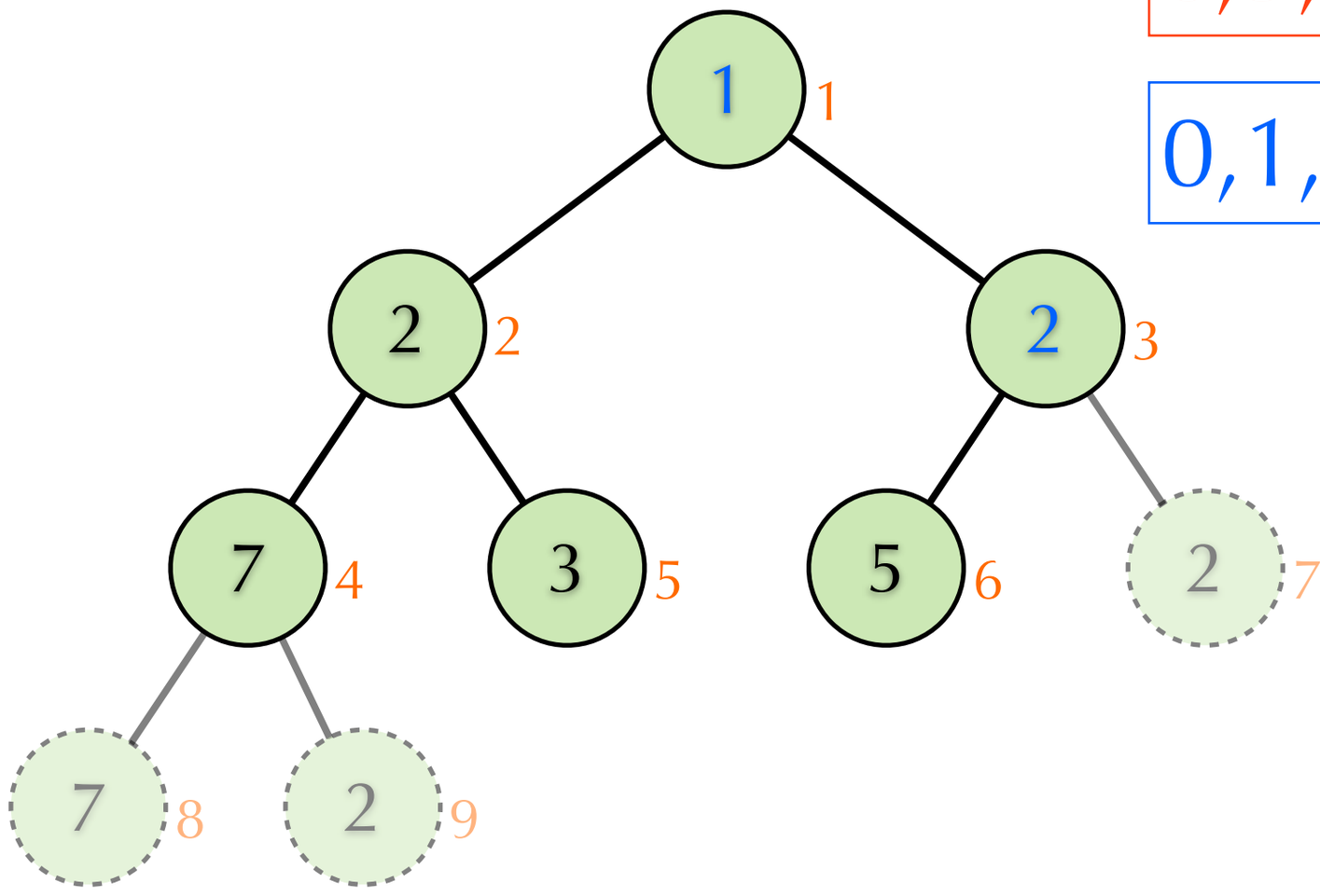



| 1 | 2 | 1 | 7 | 3 | 5 | 2 | | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: ExtractMin

0,3,5,7,1,2,1,2,1

0,1,1,1

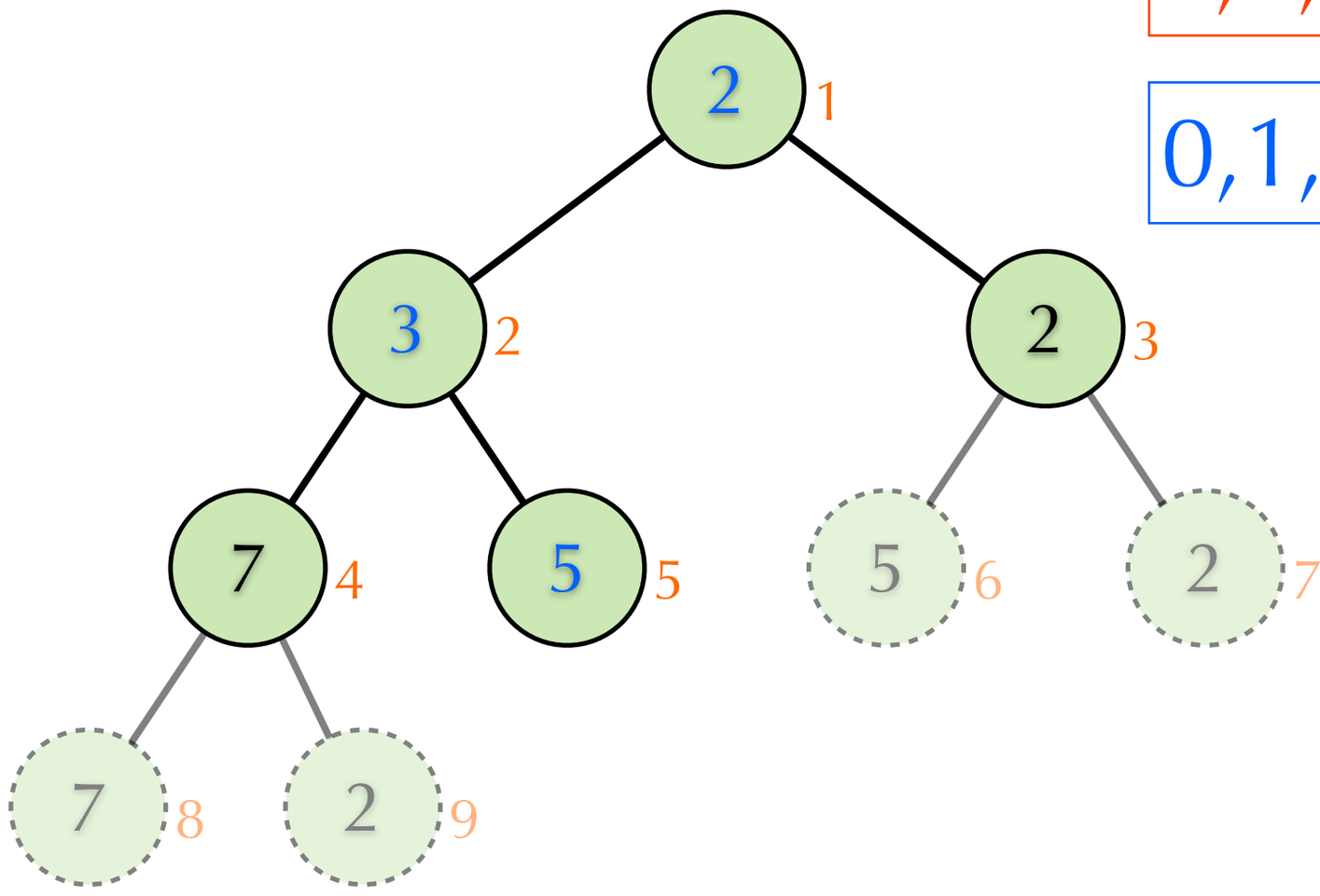



| 1 | 2 | 2 | 7 | 3 | 5 | | | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: ExtractMin

0,3,5,7,1,2,1,2,1

0,1,1,1,2

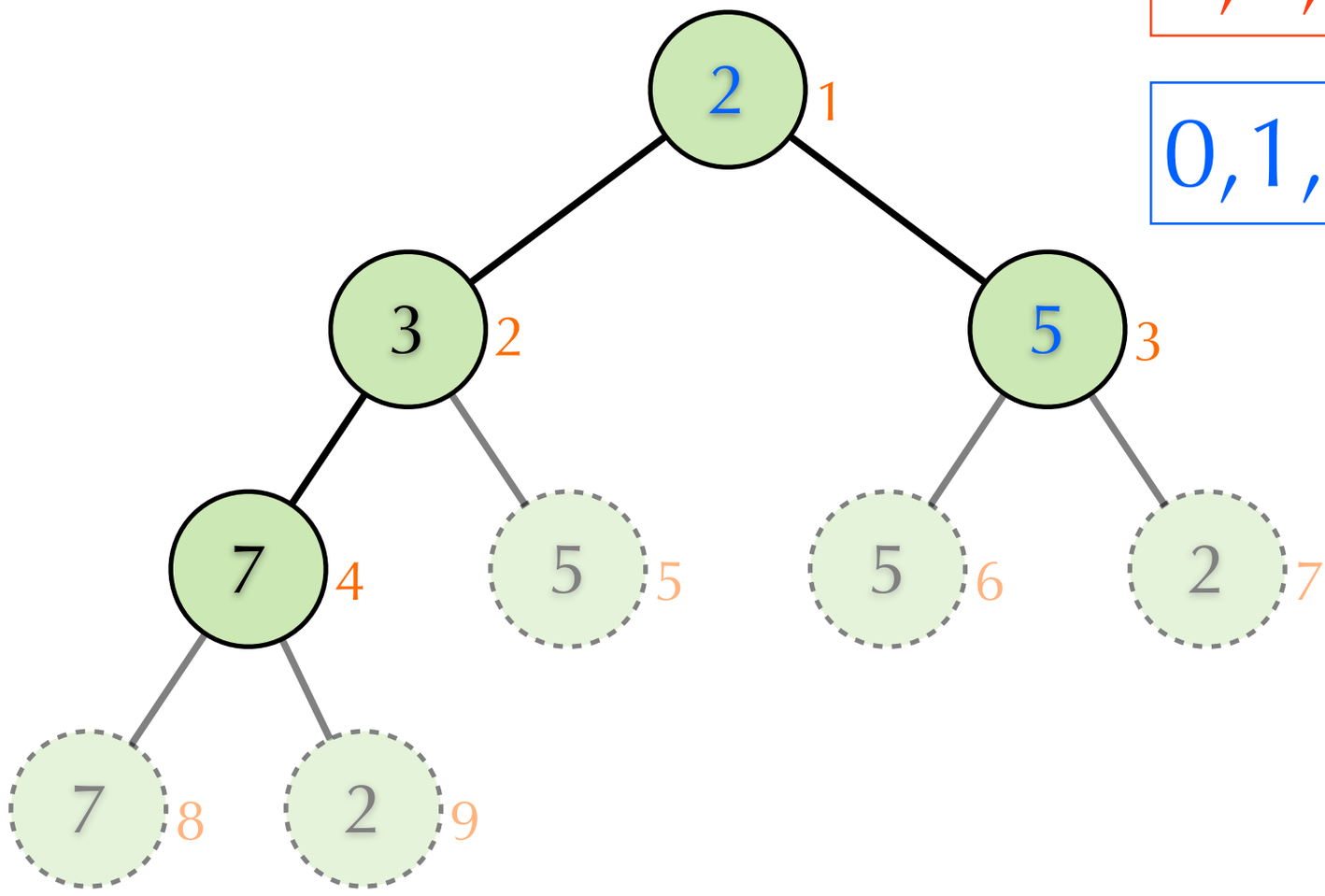



| 2 | 3 | 2 | 7 | 5 | | | | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: ExtractMin

0,3,5,7,1,2,1,2,1

0,1,1,1,2,2

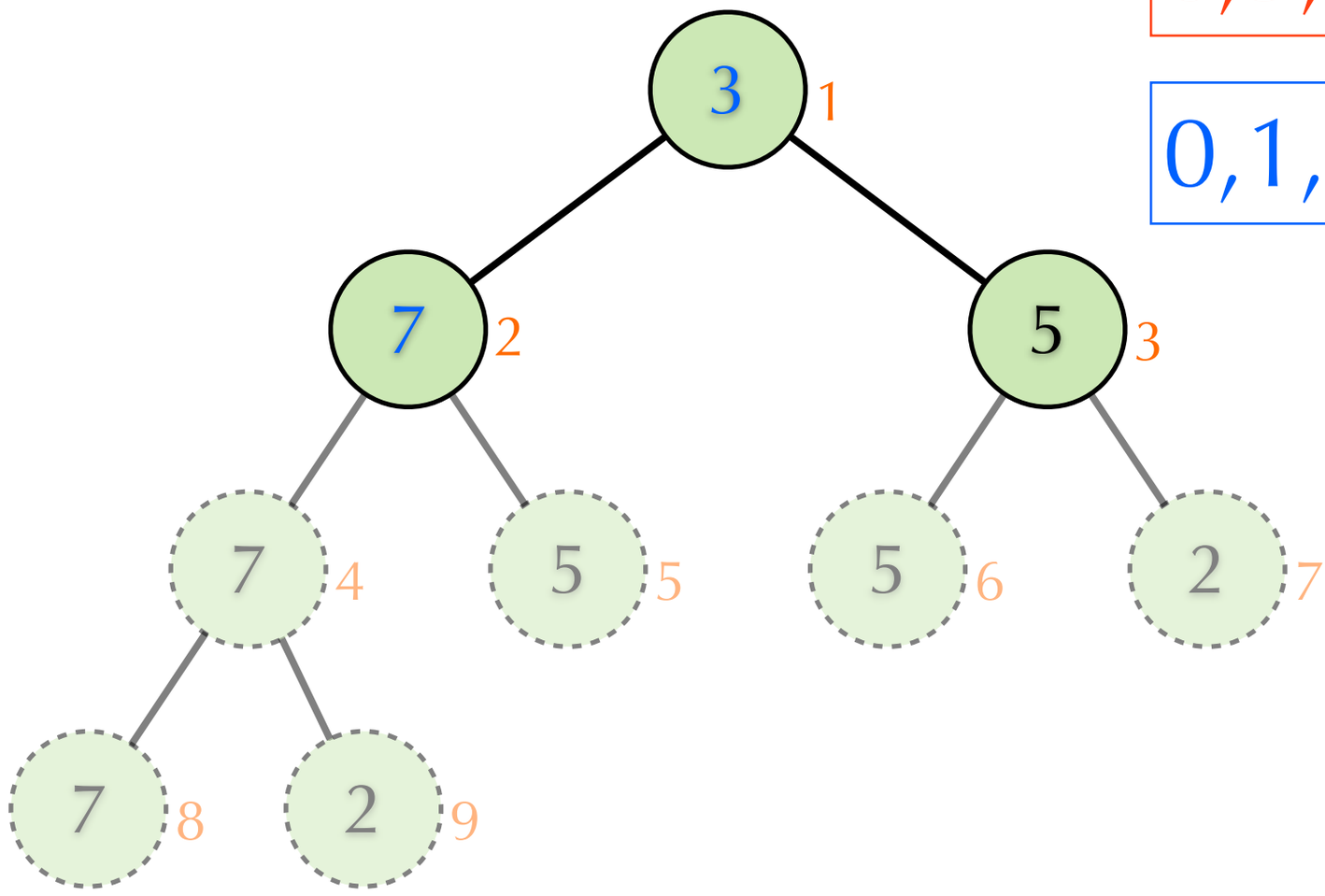



| 2 | 3 | 5 | 7 | | | | | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: ExtractMin

0,3,5,7,1,2,1,2,1

0,1,1,1,2,2,3

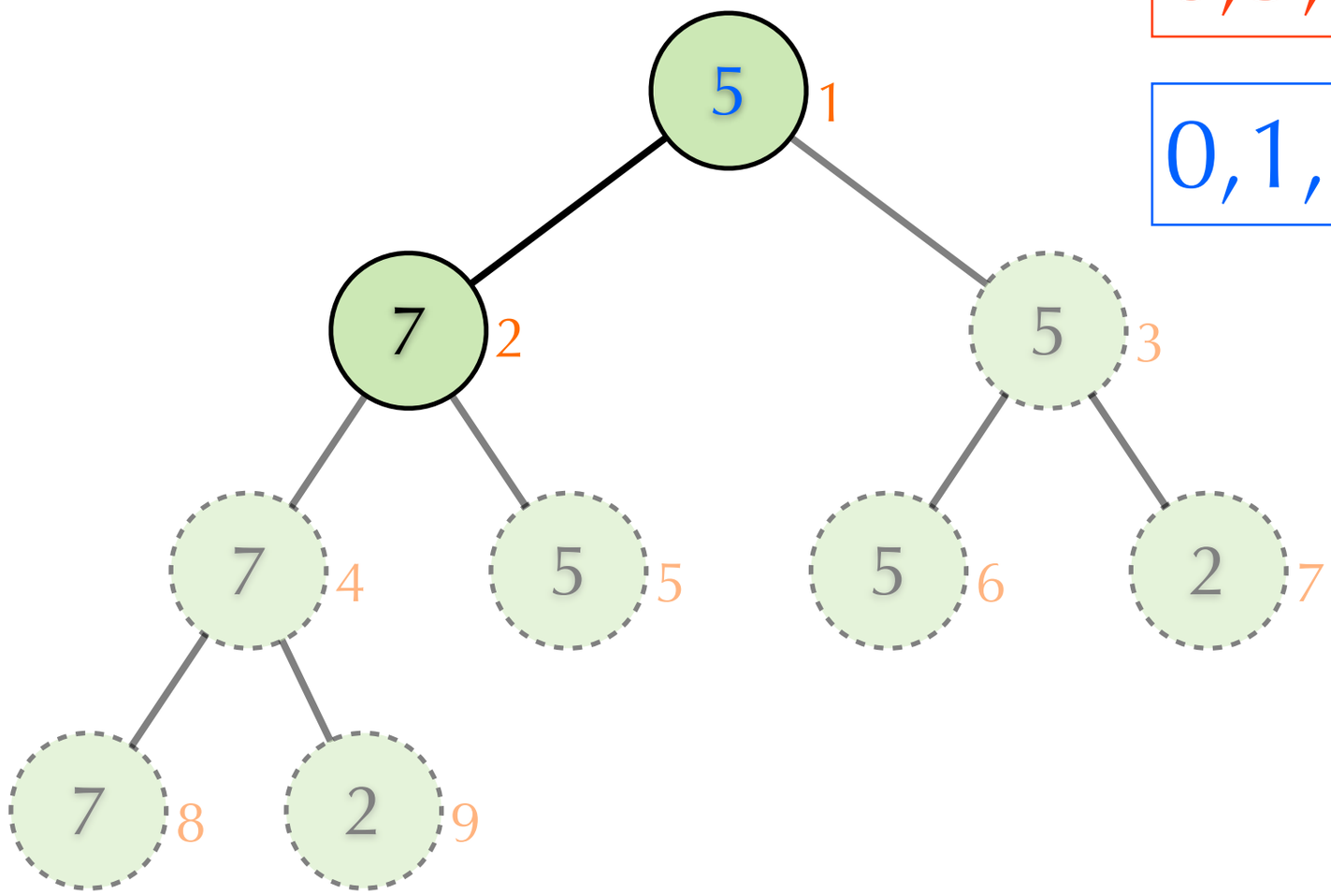



| 3 | 7 | 5 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|

# Heap sort: ExtractMin

0,3,5,7,1,2,1,2,1

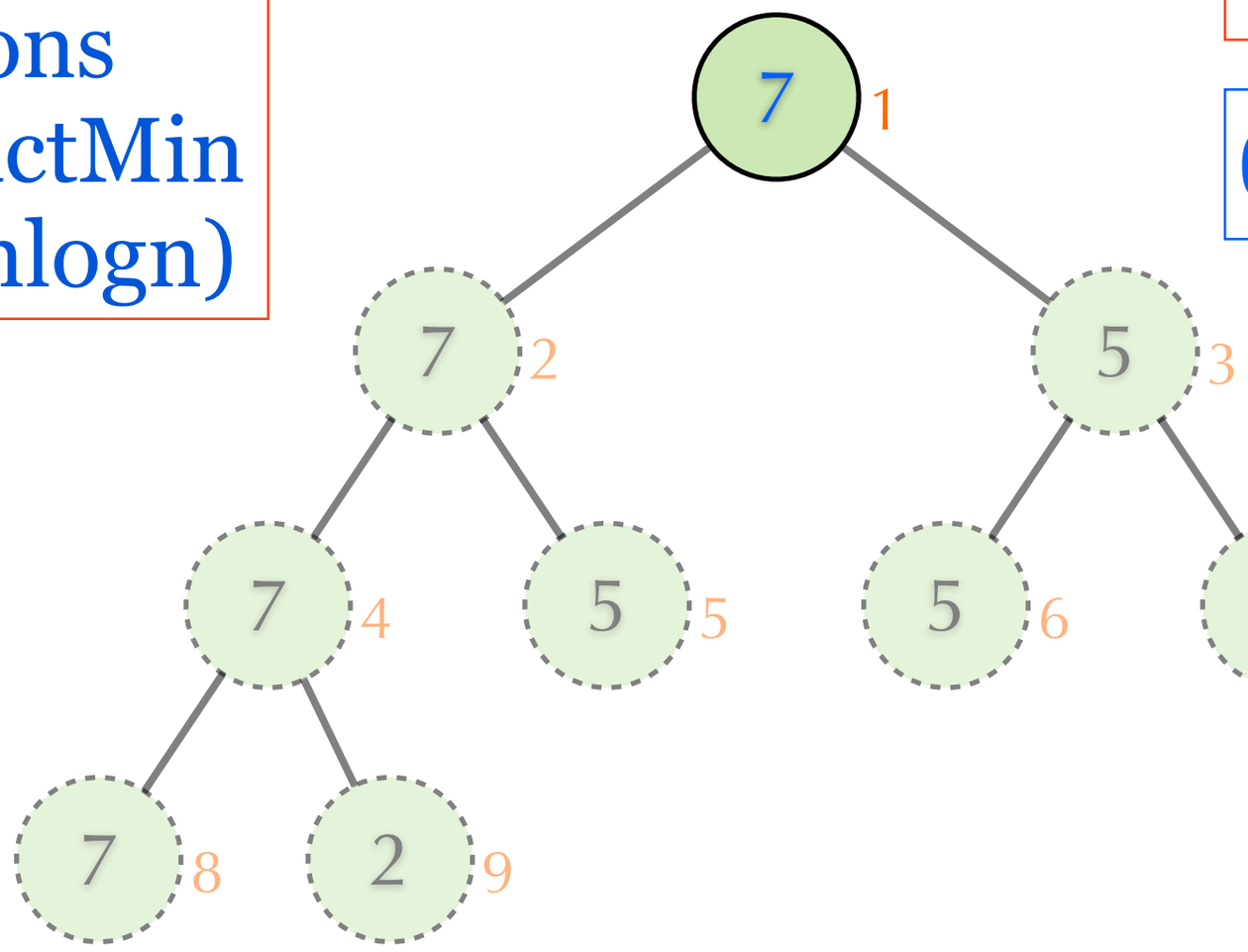
0,1,1,1,2,2,3,5




| 5 | 7 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|

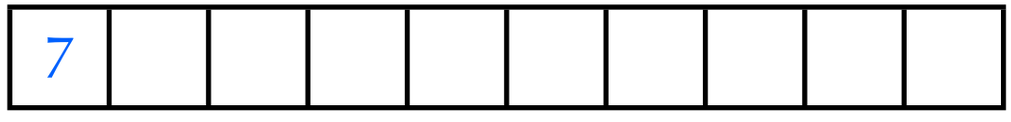
# Heap sort: ExtractMin

n insertions
& n extractMin
takes O(nlogn)




0,3,5,7,1,2,1,2,1

0,1,1,1,2,2,3,5,7



# $O(n^2)$ versus $O(n\log n)$

- Suppose we can execute $10^8$ instructions per second.
- If $n^2=10^8$ then $n=10^4$.
- If $n\log n=10^8$, then $n\approx 10^7$.
- We can sort $10^7$ numbers in seconds by heap sort, but it might take weeks to sort $10^7$ numbers by the array based selection sort.

1 week = 7 days = 168 hours = 10080 minutes = 604800 seconds