Priority Queues

Outline

- Single-Ended Priority Queue
  - Binary Heap
  - Leftist Tree
  - Binomial Heap
- Double-Ended Priority Queue
  - Interval Heap

ADT: SEPQ

- Objects: a set of <key,object>
- Operations:
  - Insert(PQ,k,obj): insert <k,obj> into PQ
  - ExtractMin(PQ): remove the object of min key
  - Min(PQ): return the object of min key
  - DecreaseKey(PQ,k,obj): decrease the key value of obj to k
  - Union(PQ_1,PQ_2): union PQ_1 and PQ_2 into one priority queue.

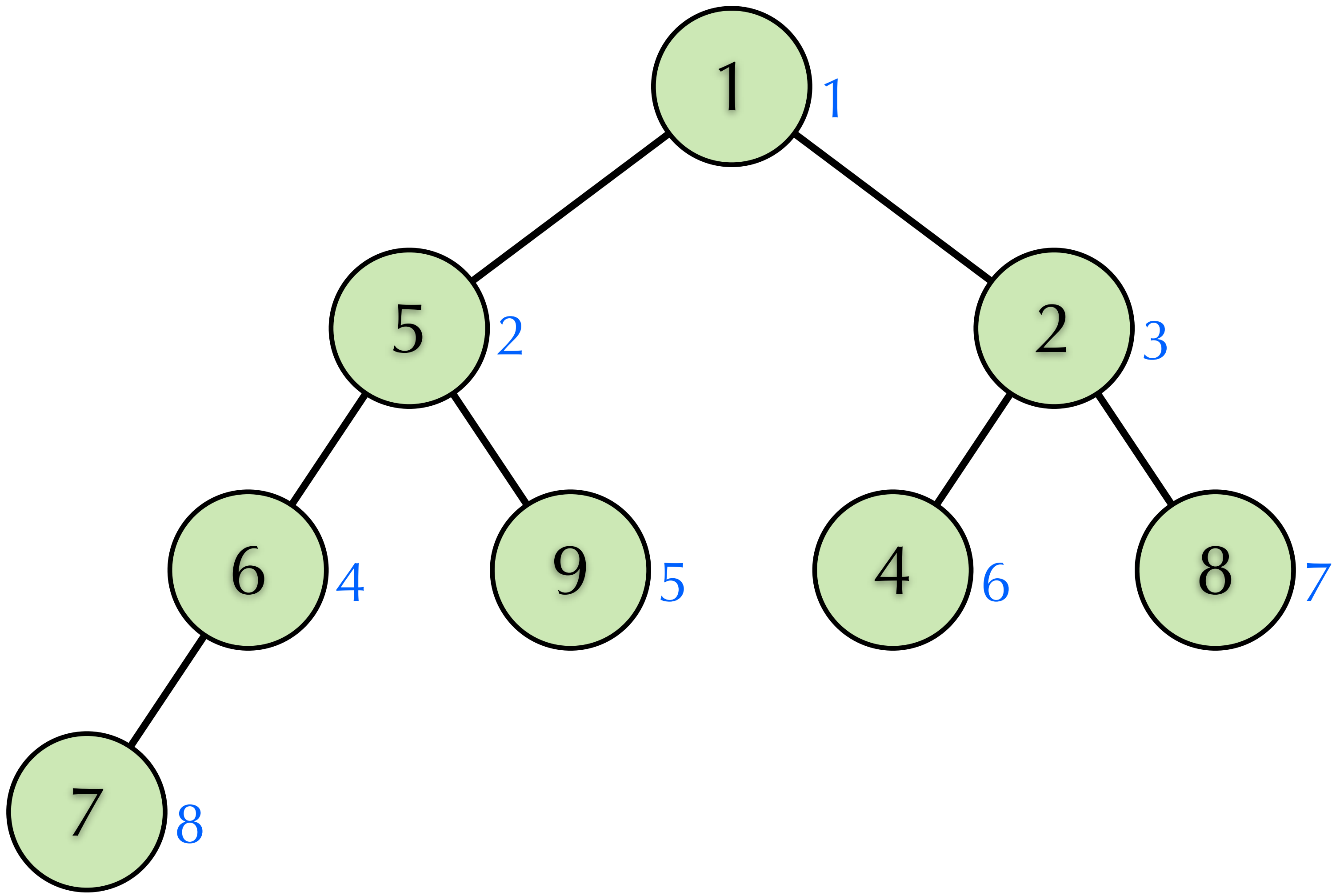
ADT: DEPQ

- Objects: a set of <key,object>
- Operations:
  - Insert(PQ,k,obj): insert <k,obj> into PQ
  - ExtractMin(PQ): remove the object of min key
  - Min(PQ): return the object of min key
  - ExtractMax(PQ): remove the object of max key
  - Max(PQ): return the object of max key
  - ChangeKey(PQ,k,obj): Change the key of obj to k

Binary Heap (Min)

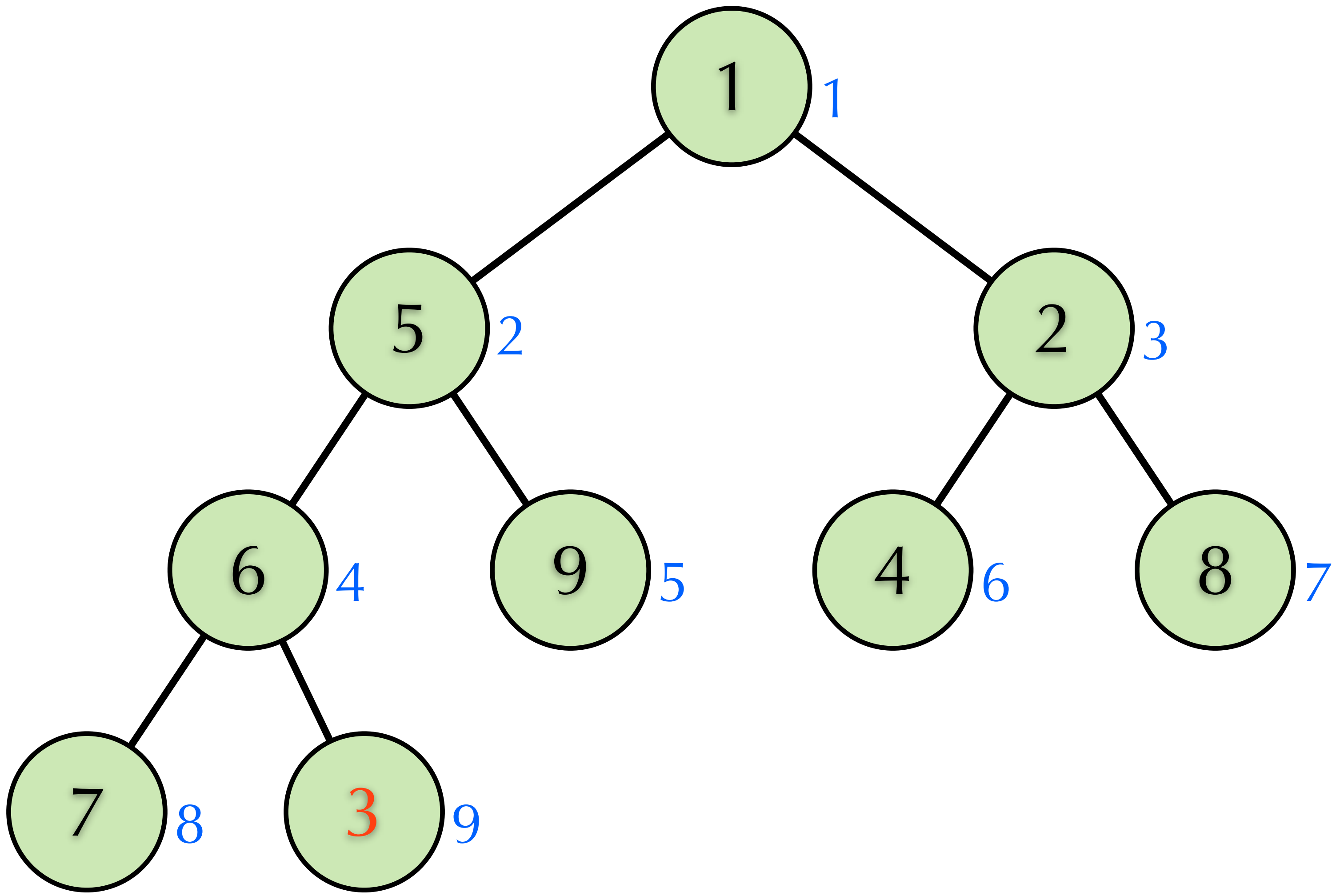
- A binary tree H of n nodes
  - The nodes are in n consecutive positions in H’s array representation
- Every node stores a key-object pair.
  - A’s key≤B’s key if A is B’s parent.
- This is a nice structure for finding minimum!
  - The root has the minimum key.

Example: binary heap



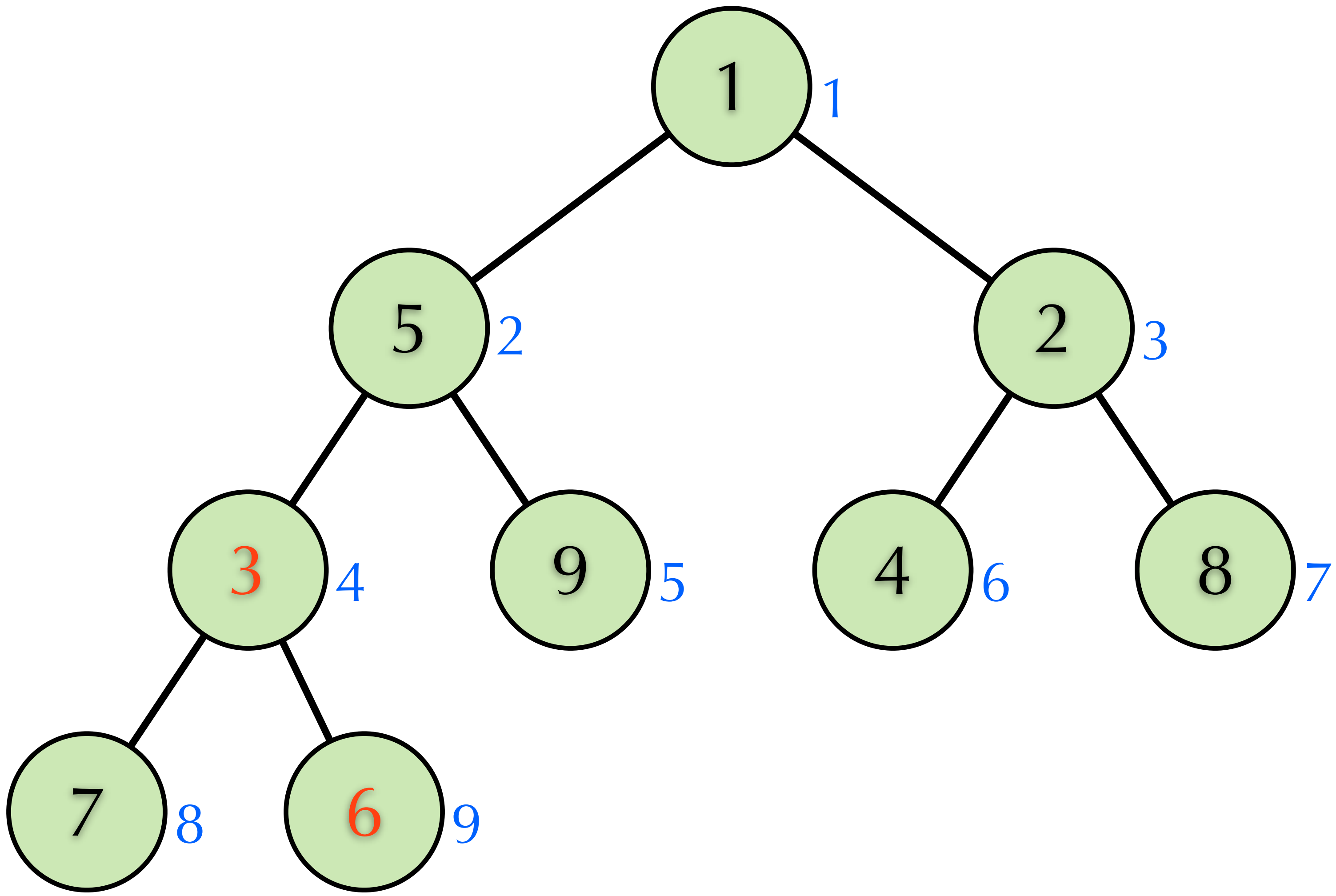
1	2	3	4	5	6	7	8	9	10
1	5	2	6	9	4	8	7		

Insert 3



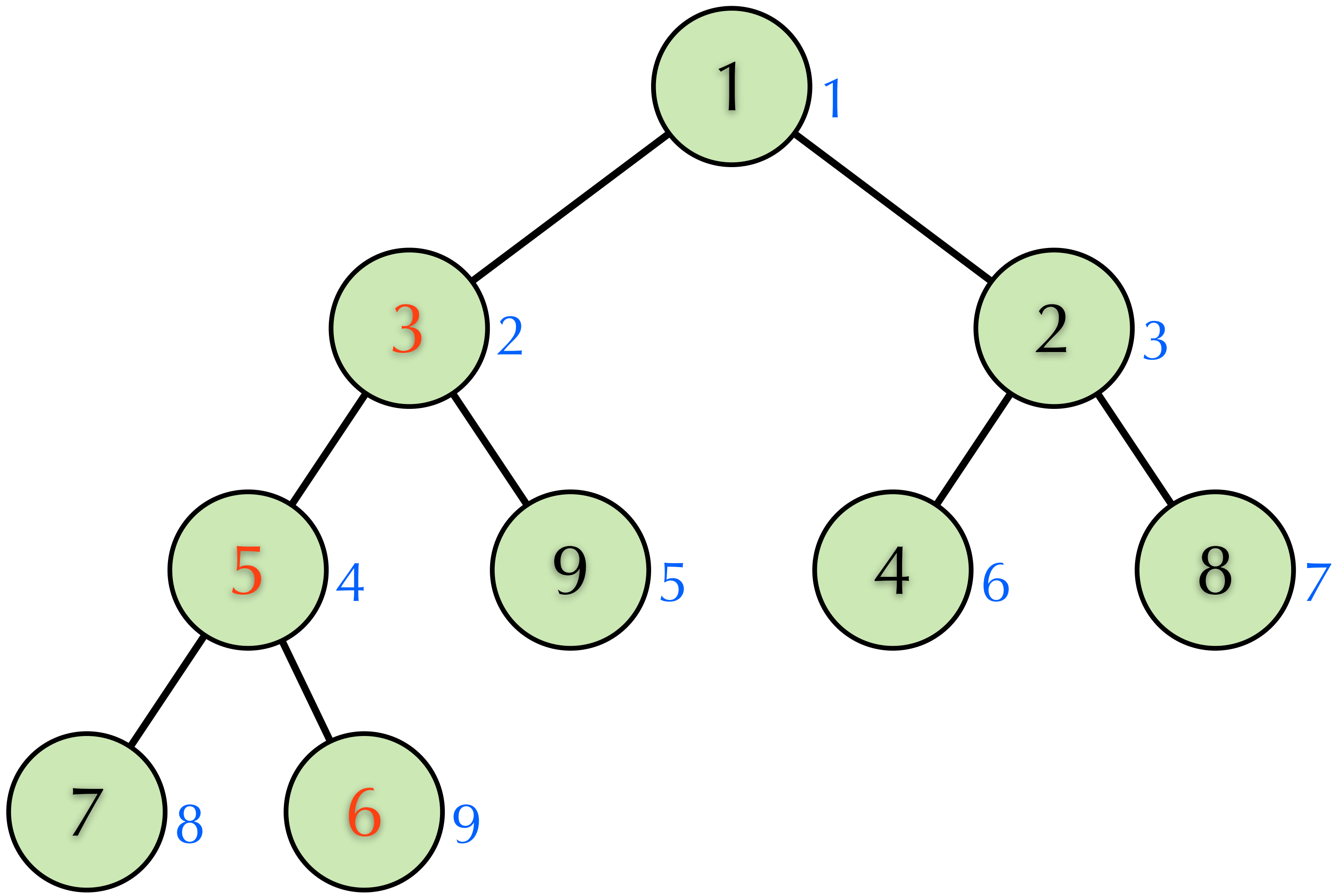
1	2	3	4	5	6	7	8	9	10
1	5	2	6	9	4	8	7	3	

Insert 3



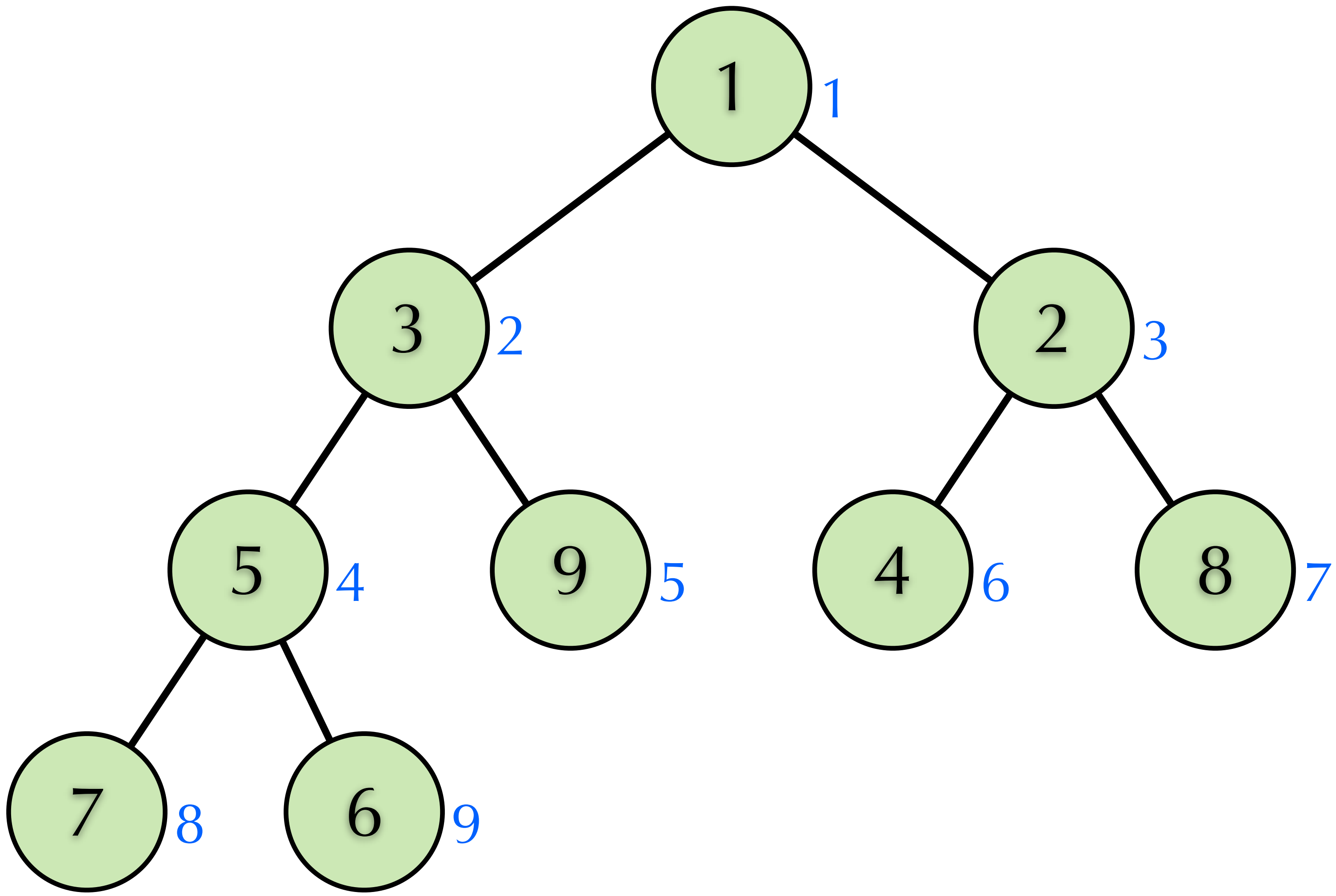
1	2	3	4	5	6	7	8	9	10
1	5	2	3	9	4	8	7	6	

Insert 3



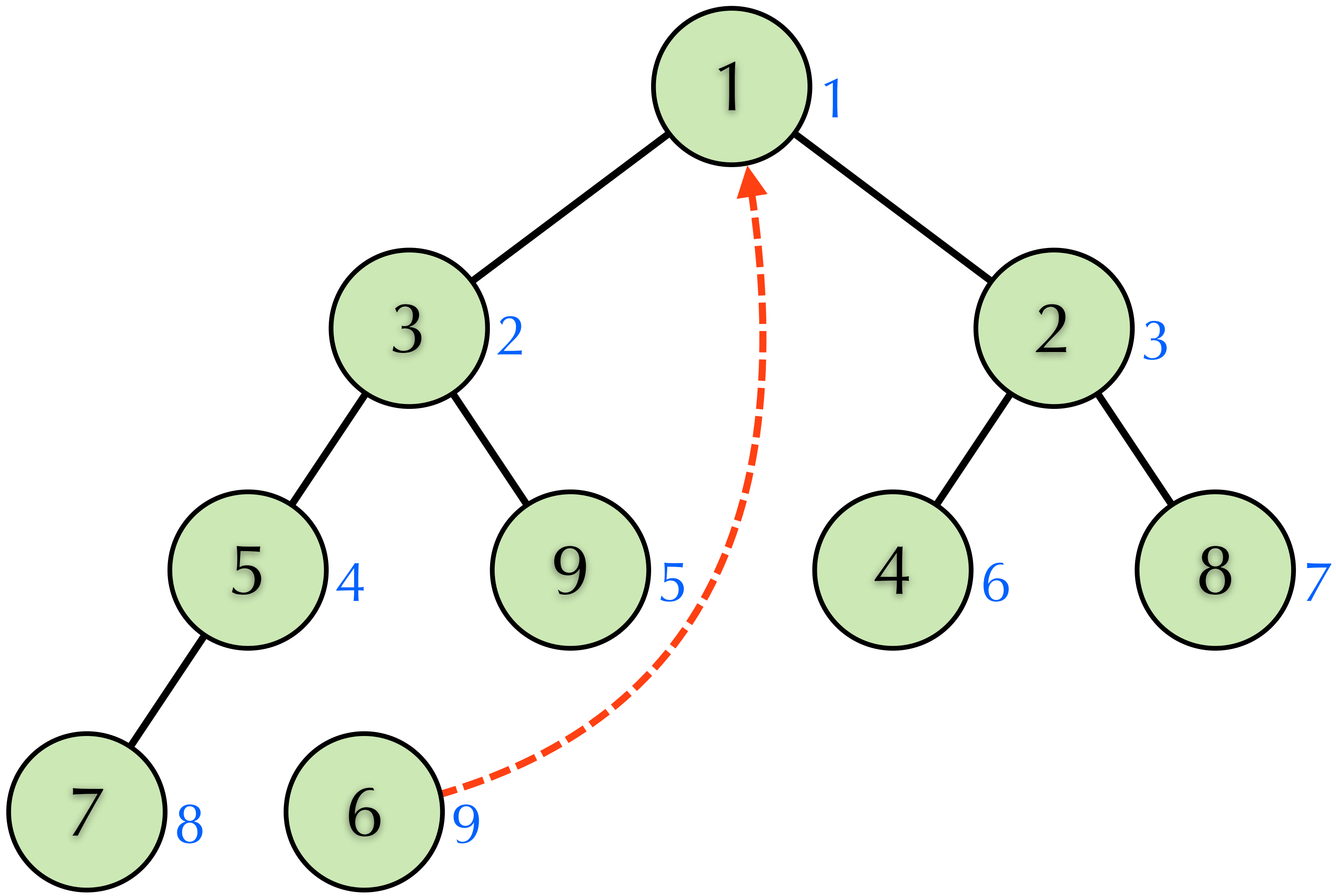
1	2	3	4	5	6	7	8	9	10
1	3	2	5	9	4	8	7	6	

Extract Minimum



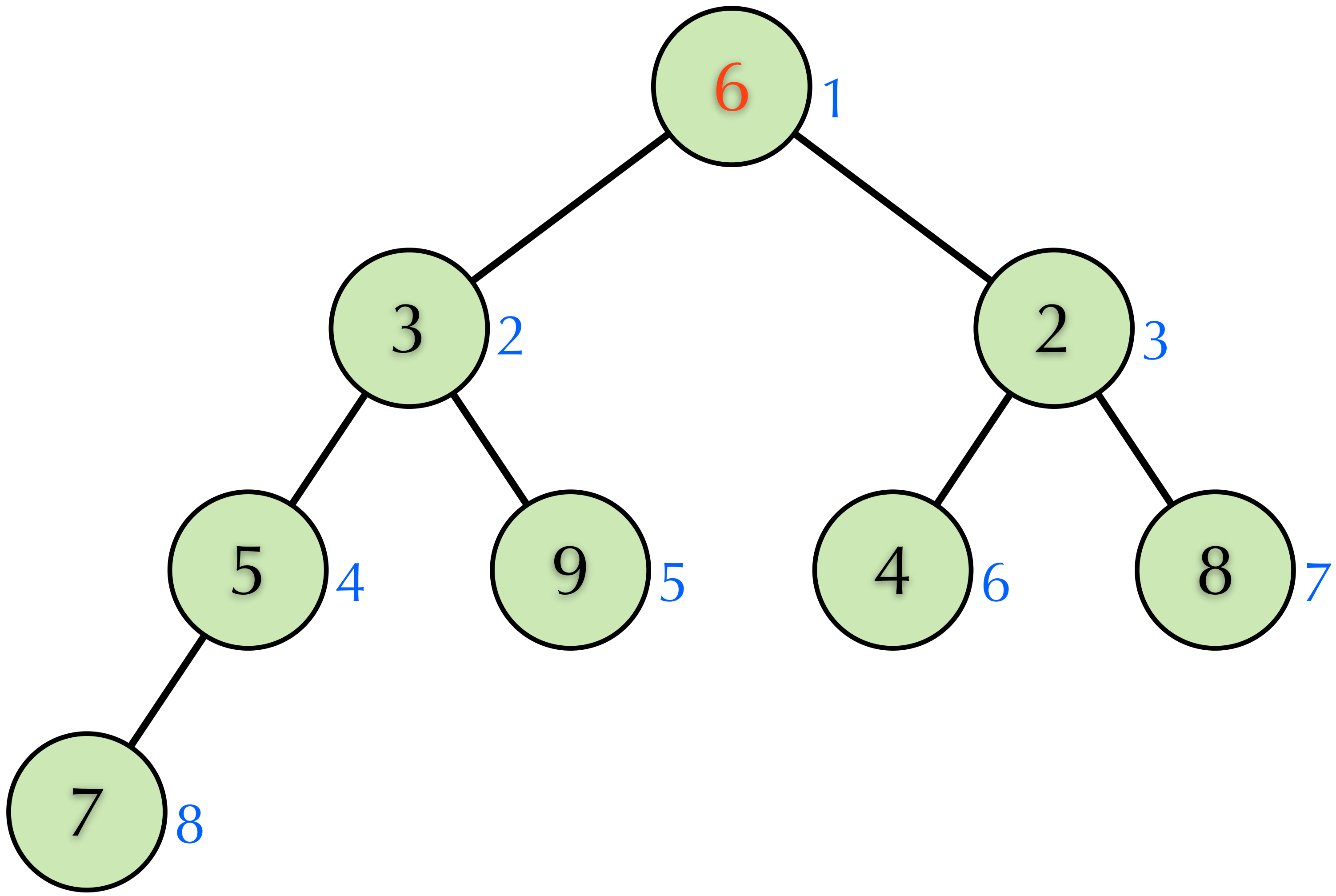
1	2	3	4	5	6	7	8	9	10
1	3	2	5	9	4	8	7	6	

Extract Minimum



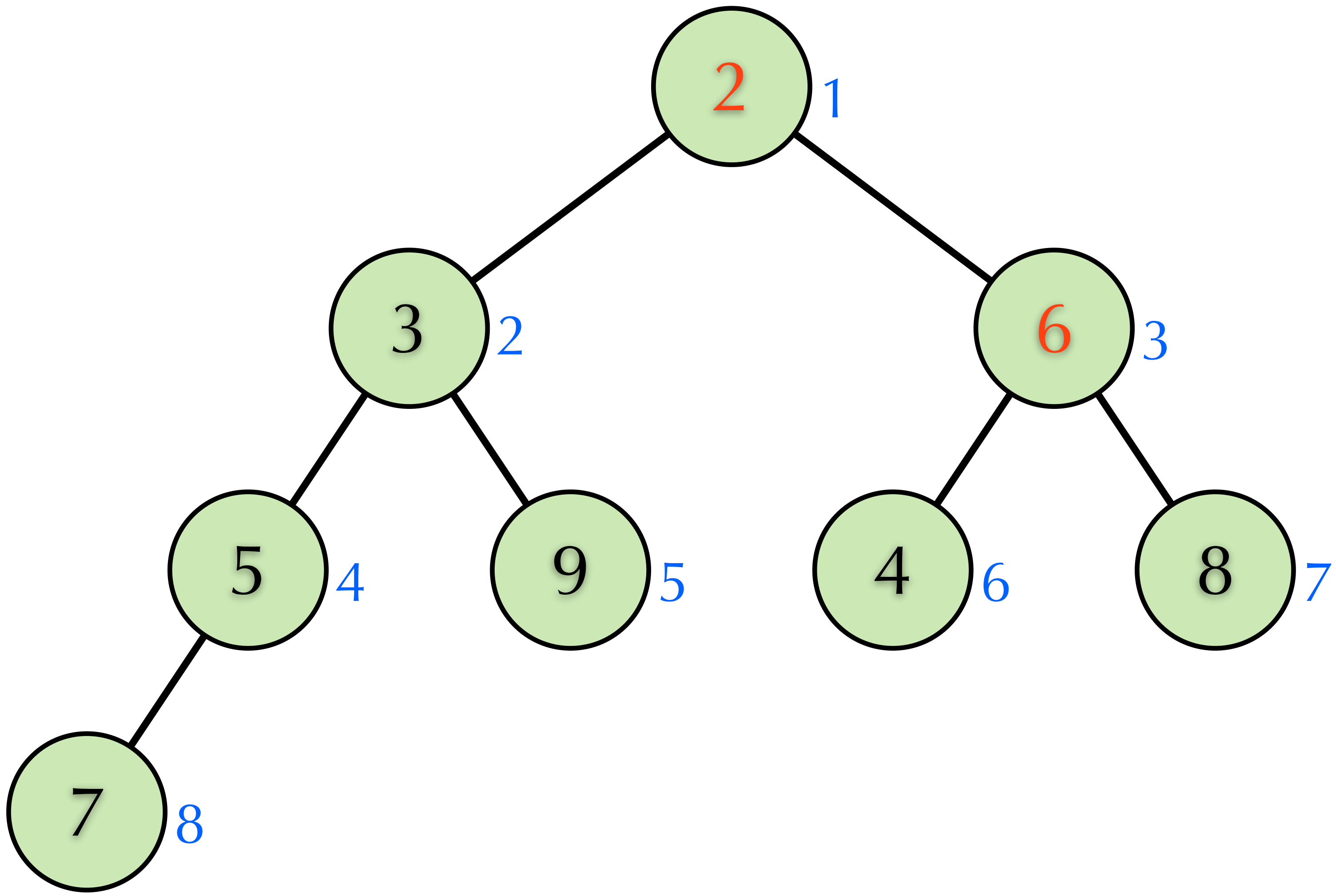
1	2	3	4	5	6	7	8	9	10
1	3	2	5	9	4	8	7	6	

Extract Minimum



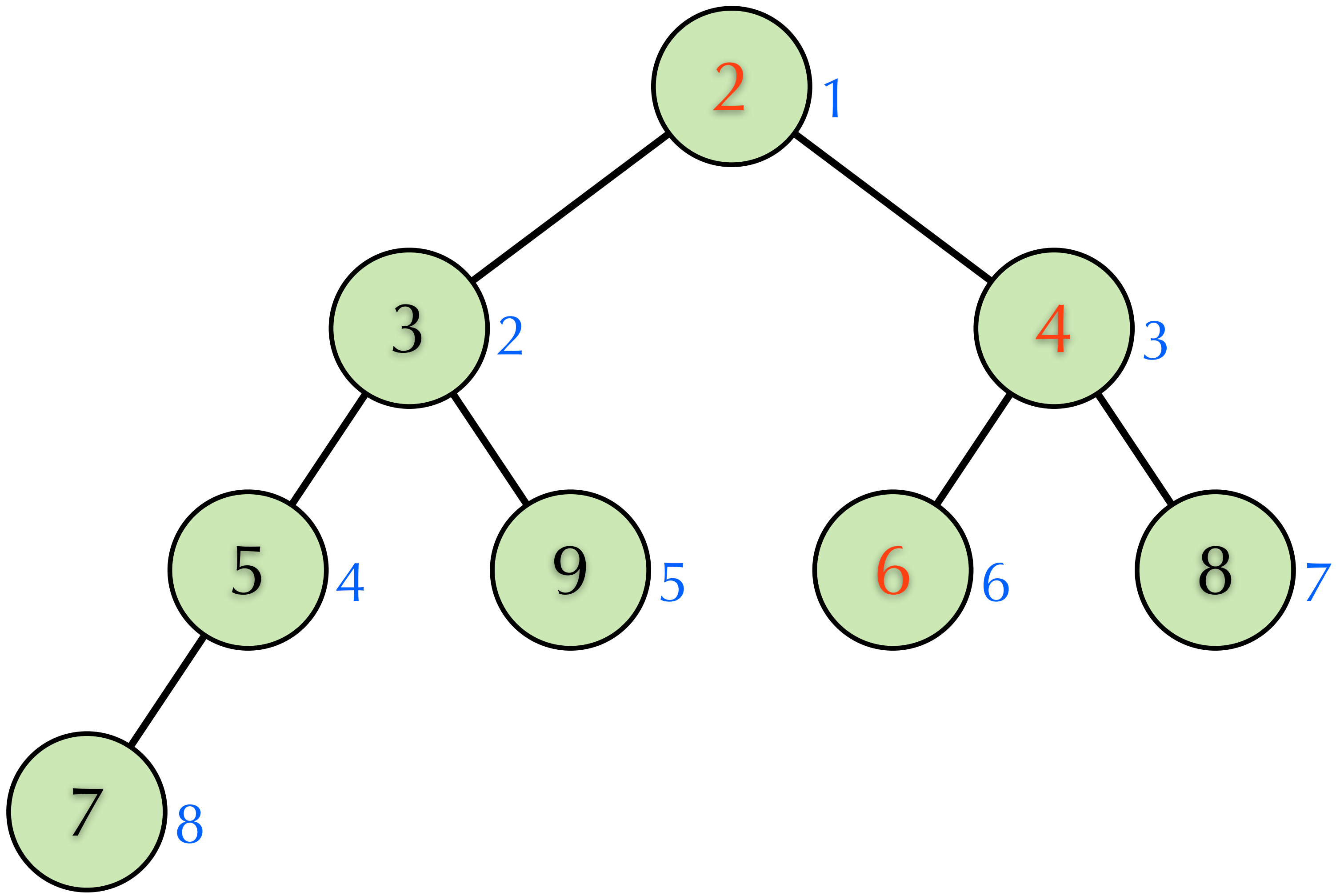
1	2	3	4	5	6	7	8	9	10
6	3	2	5	9	4	8	7		

Extract Minimum



1	2	3	4	5	6	7	8	9	10
2	3	6	5	9	4	8	7		

Extract Minimum



1	2	3	4	5	6	7	8	9	10
2	3	4	5	9	6	8	7		

Binary Heap

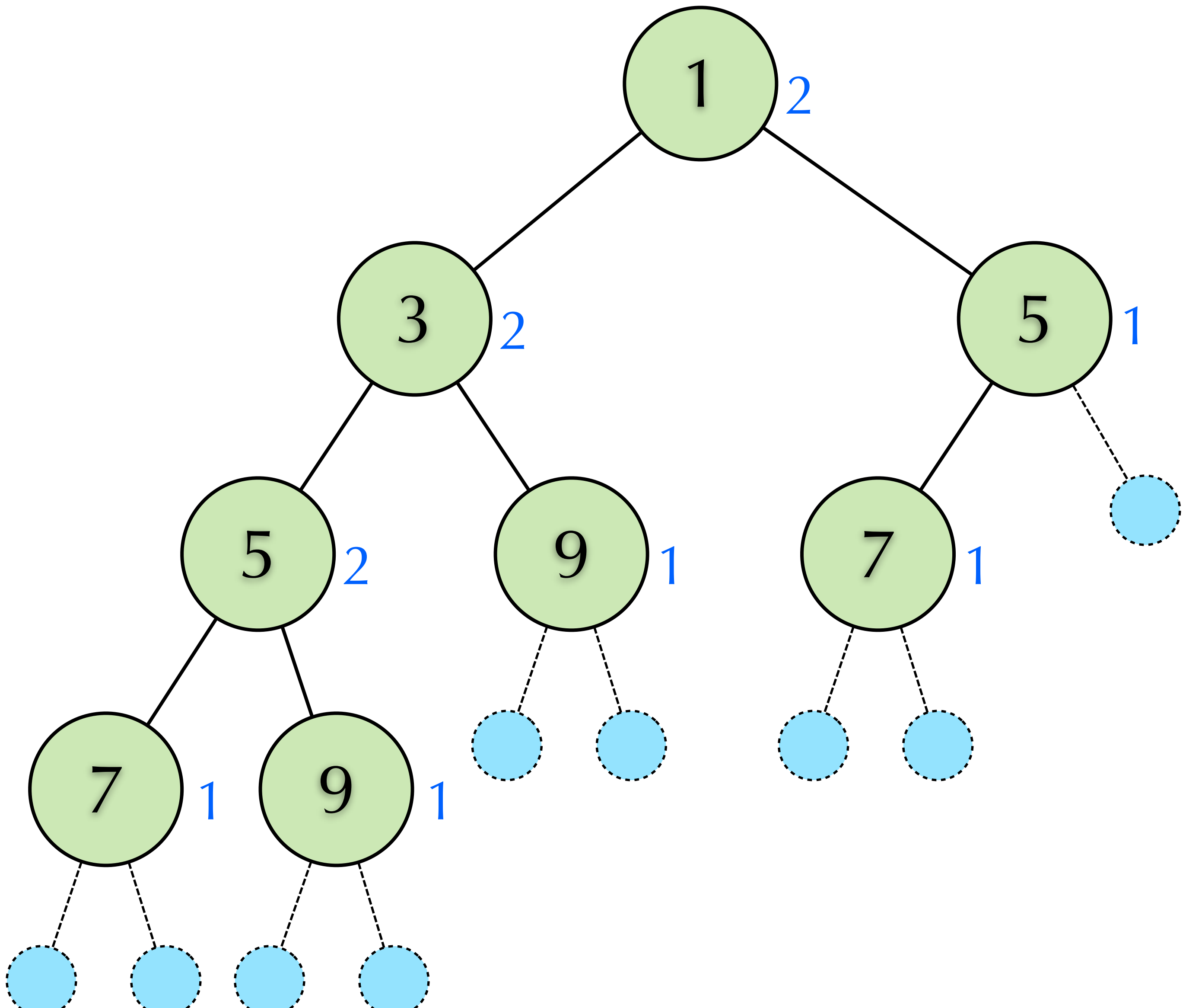
- Pros
  - Array-based
  - Iterative implementation is easy
  - Almost in place ($O(1)$-space overhead)
- Cons
  - Slow union: takes $O(n)$-time

Leftist Tree

- A binary tree with external leaves
  - Root: r
  - S-value of a node v: the shortest length from v to any leaf in subtree rooted v.
    - $S(v)=0$ if v is an external leaf
    - $S(v)=\min(S(v.L),S(v.R))+1$
  - For any node v, $S(v.L) \geq S(v.R)$.
  - The shortest path from r to leaf is the rightmost path.

Leftist Tree

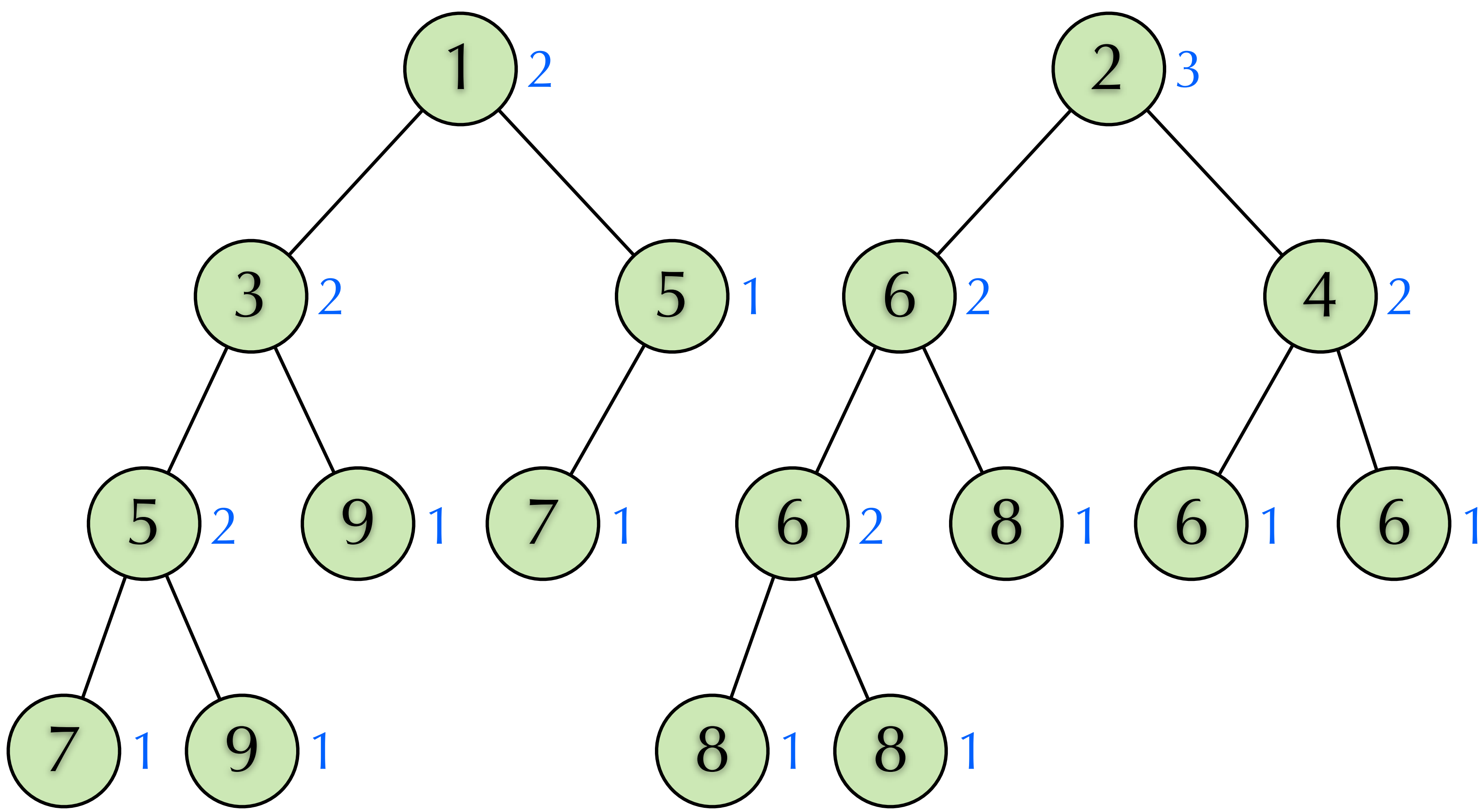
Min Heap Property: Parent's key is no greater than child's key.



Meld Operation

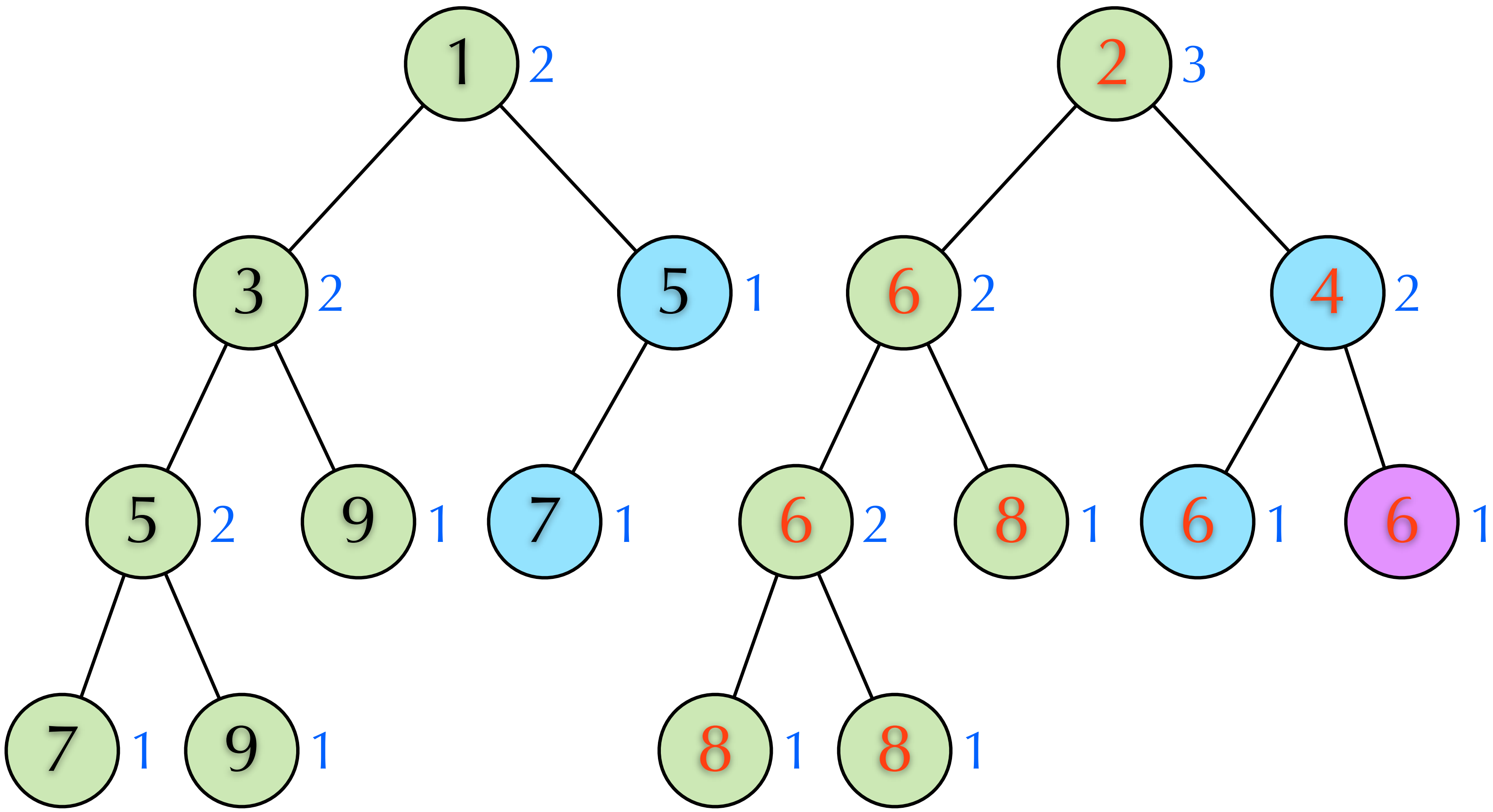
- The basic operation on leftist trees.
- Insert(PQ,k,obj)
  - Meld PQ and a one-node leftist tree whose root is <k,obj>
- ExtractMin(PQ)
  - Meld the left subtree and right subtree, then return the original root.
- DecreaseKey(PQ,k,obj)
  - Homework

Meld

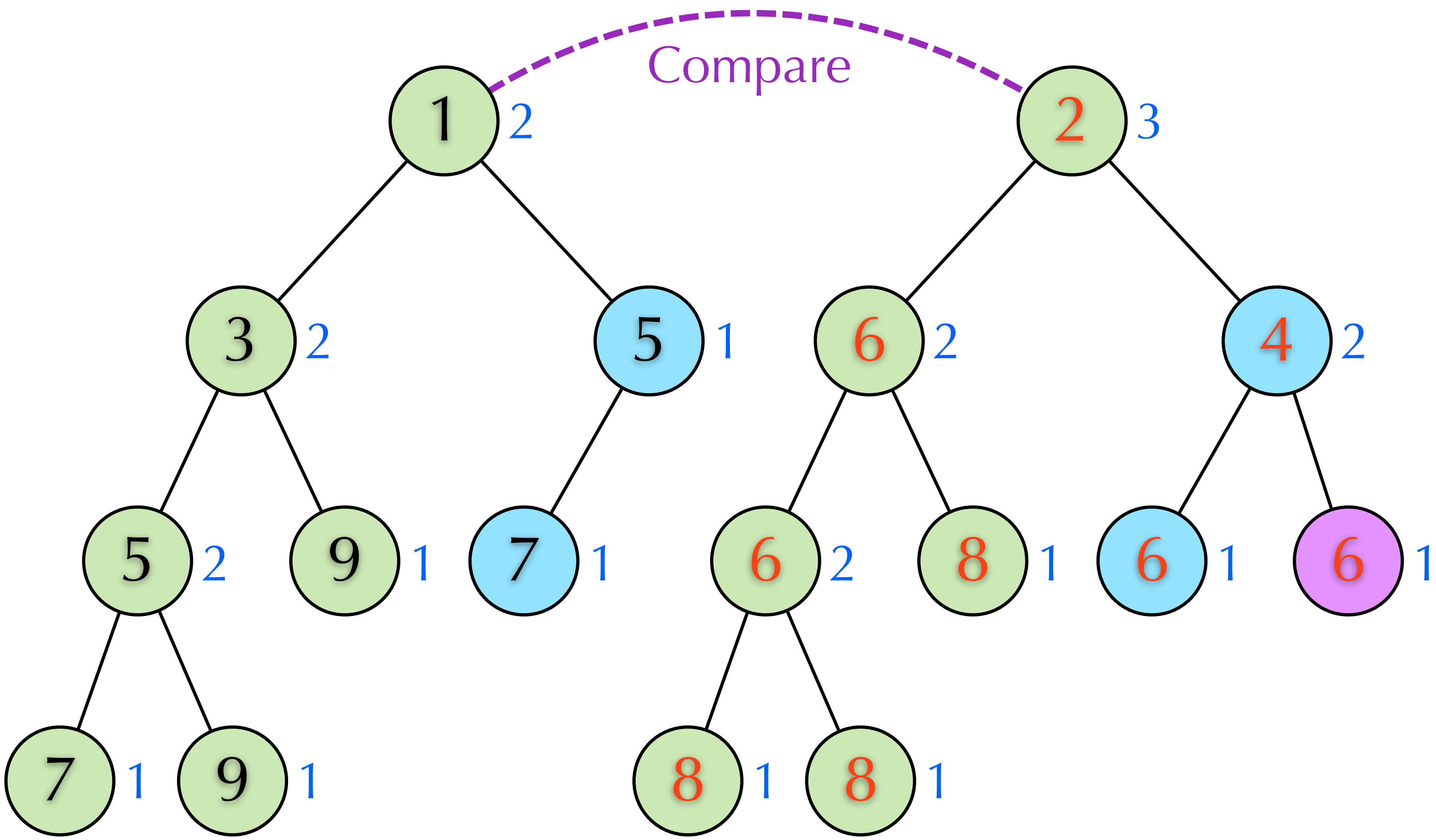


- Two phases:
- 1. Merge the rightmost paths into one. O(logn)
- 2. Rebuild the S-values and swap v's subtrees if S(v.L)<S(v.R). O(logn)

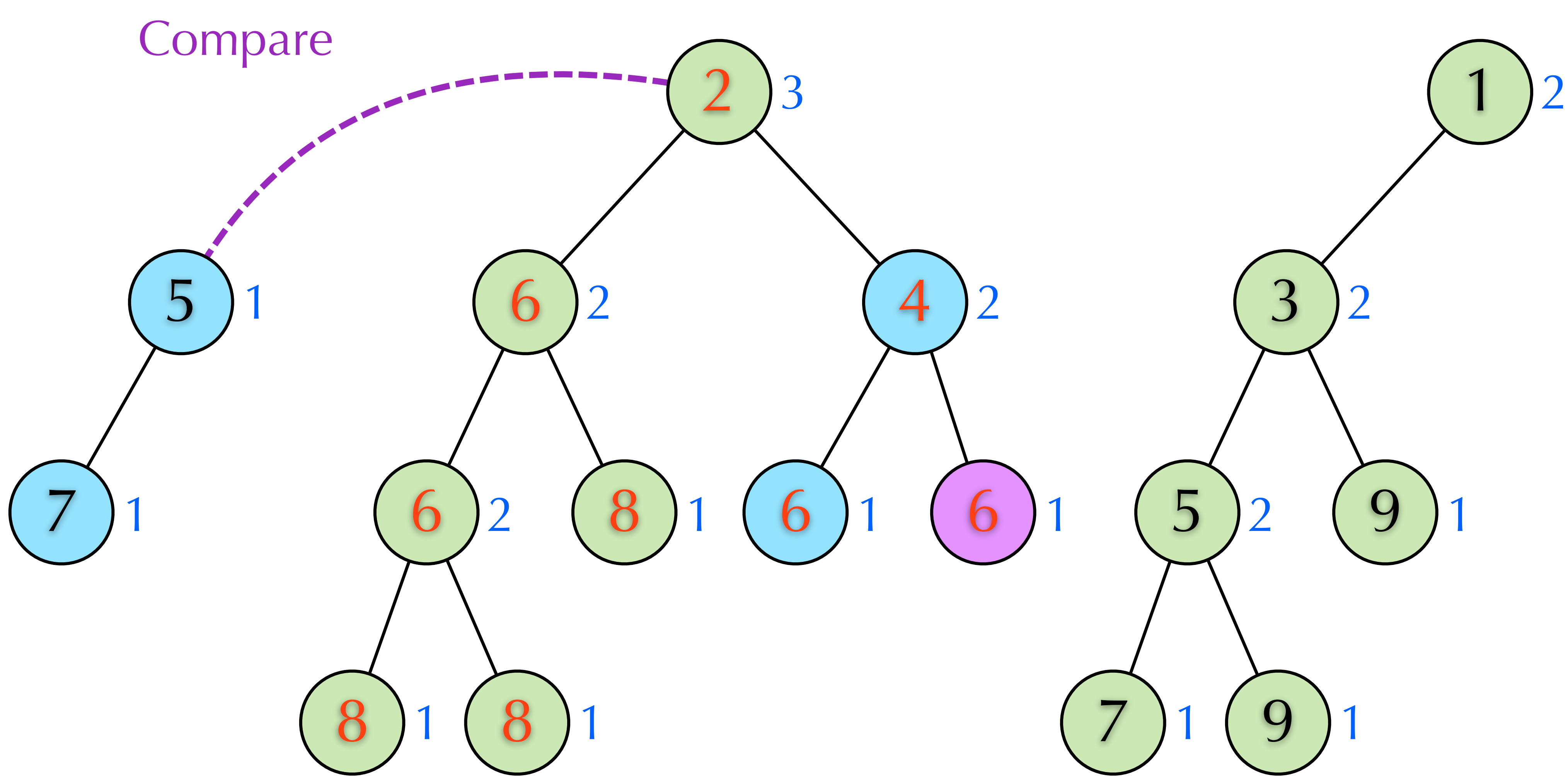
Meld: Phase 1



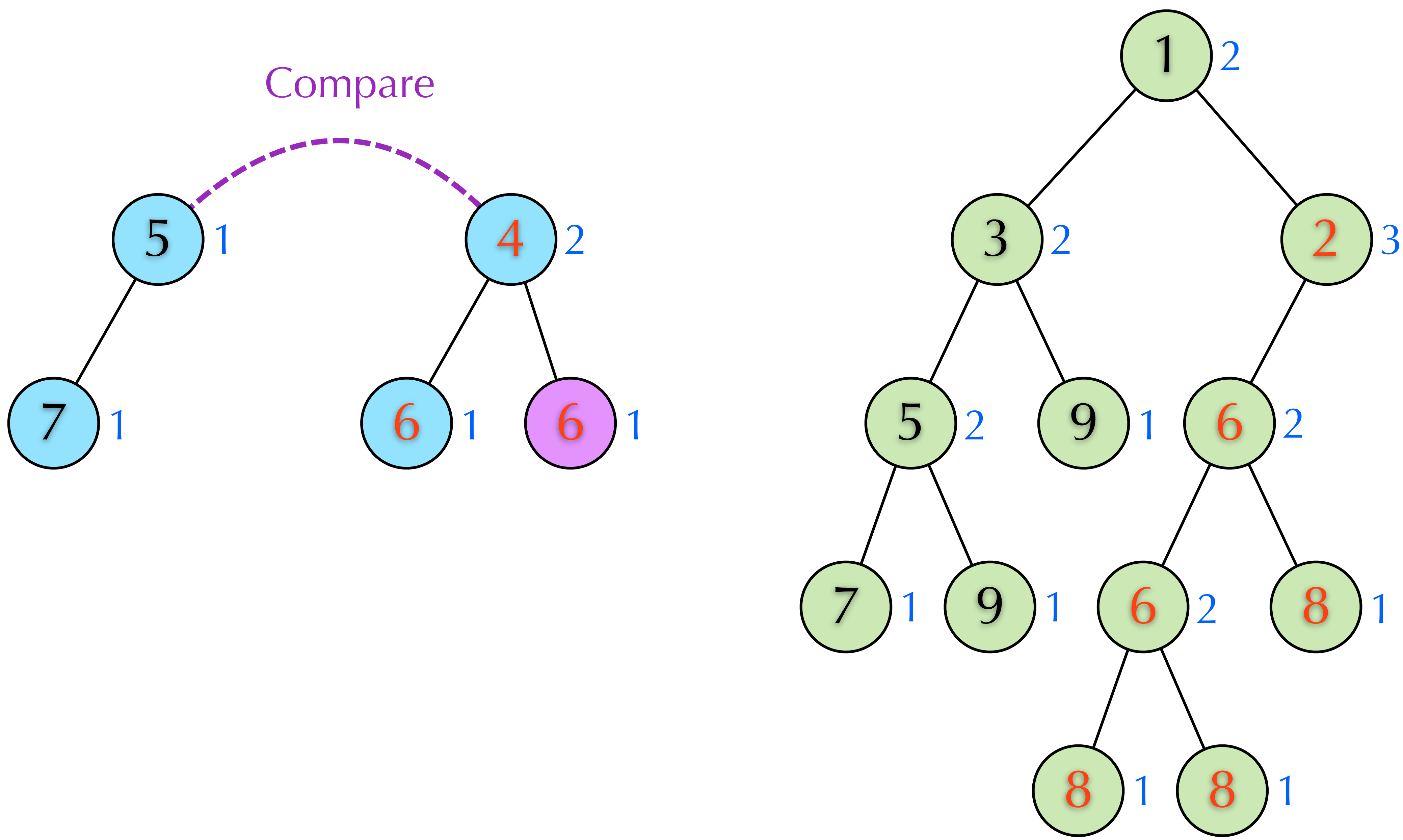
Meld: Phase 1



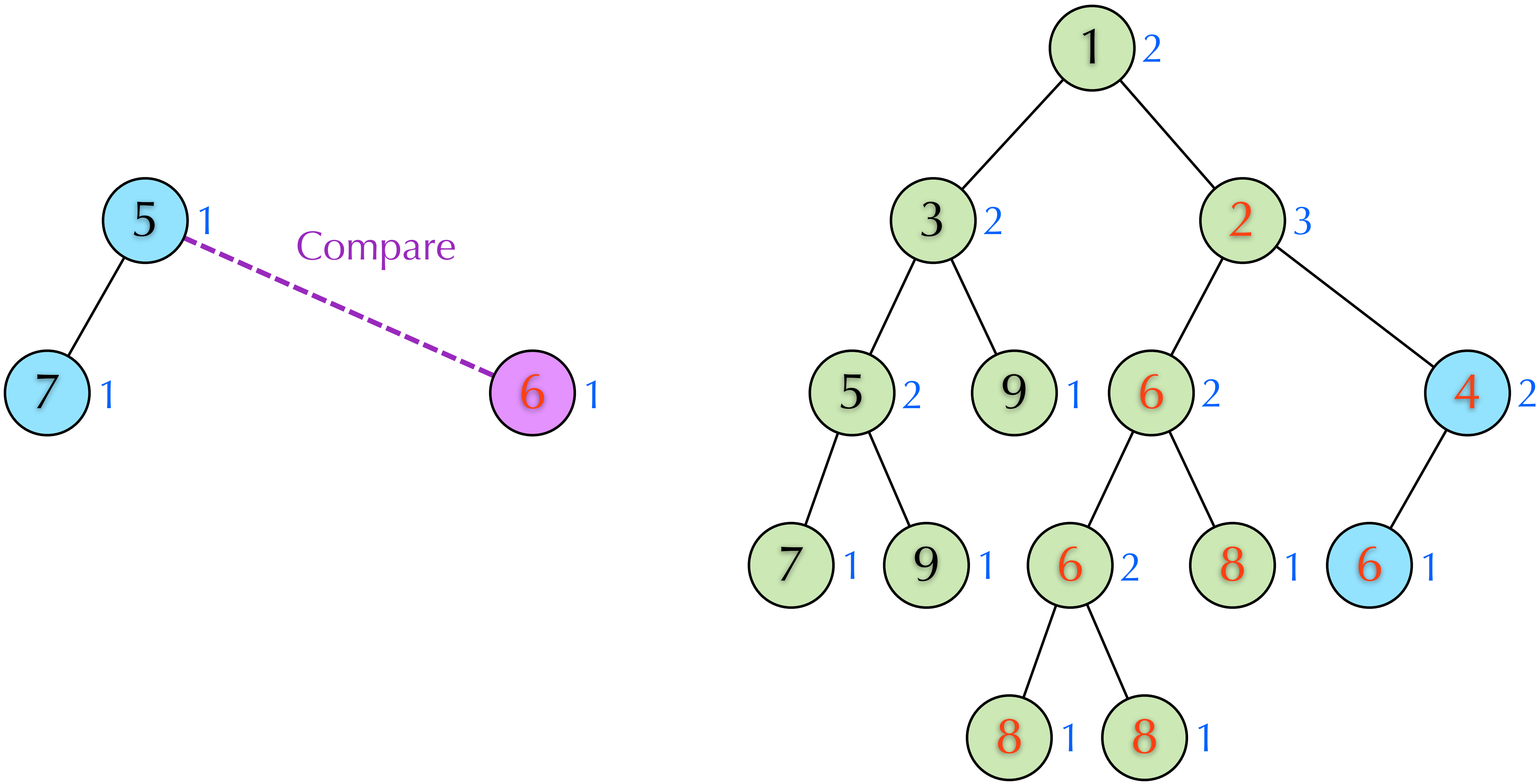
Meld: Phase 1



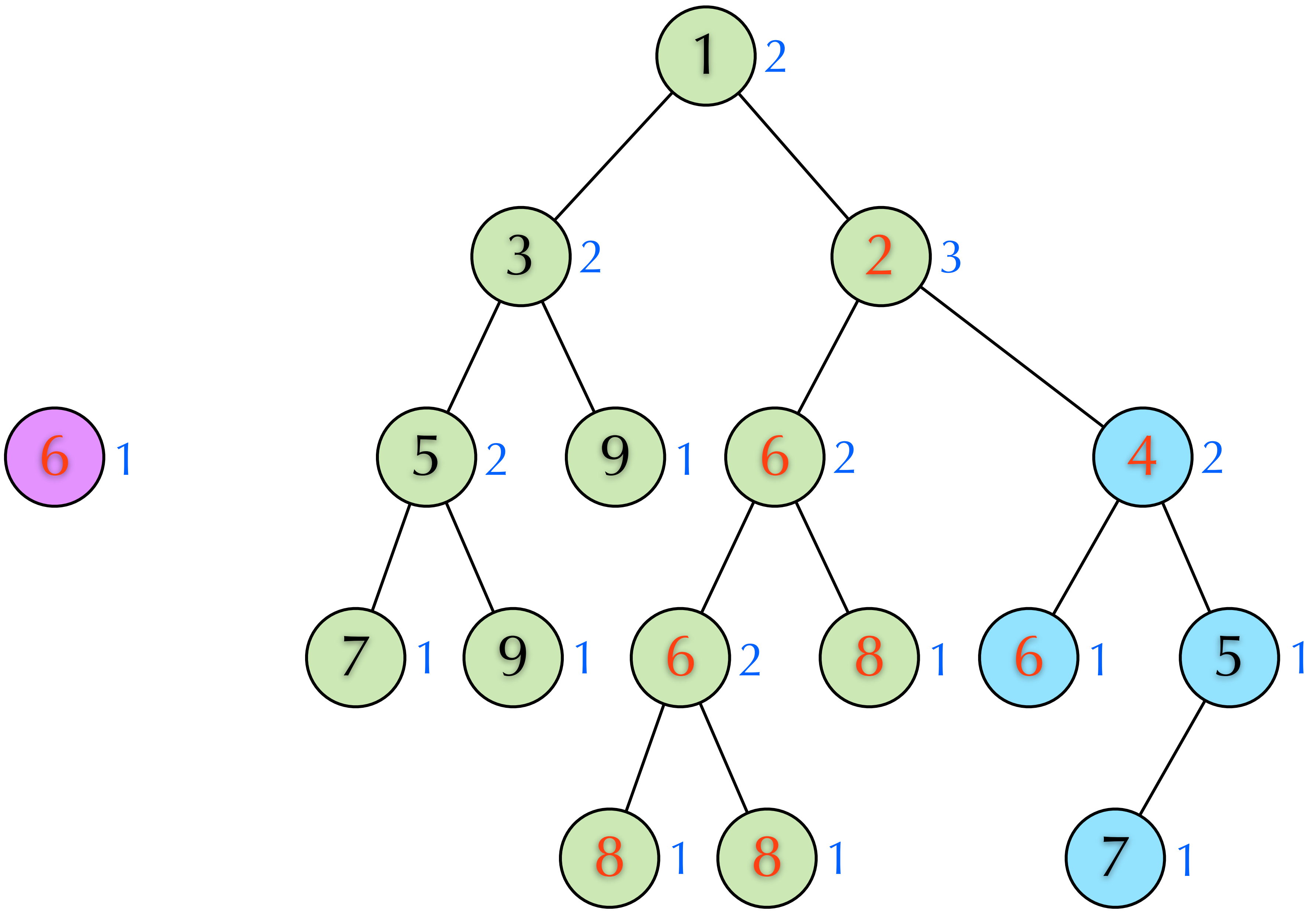
Meld: Phase 1



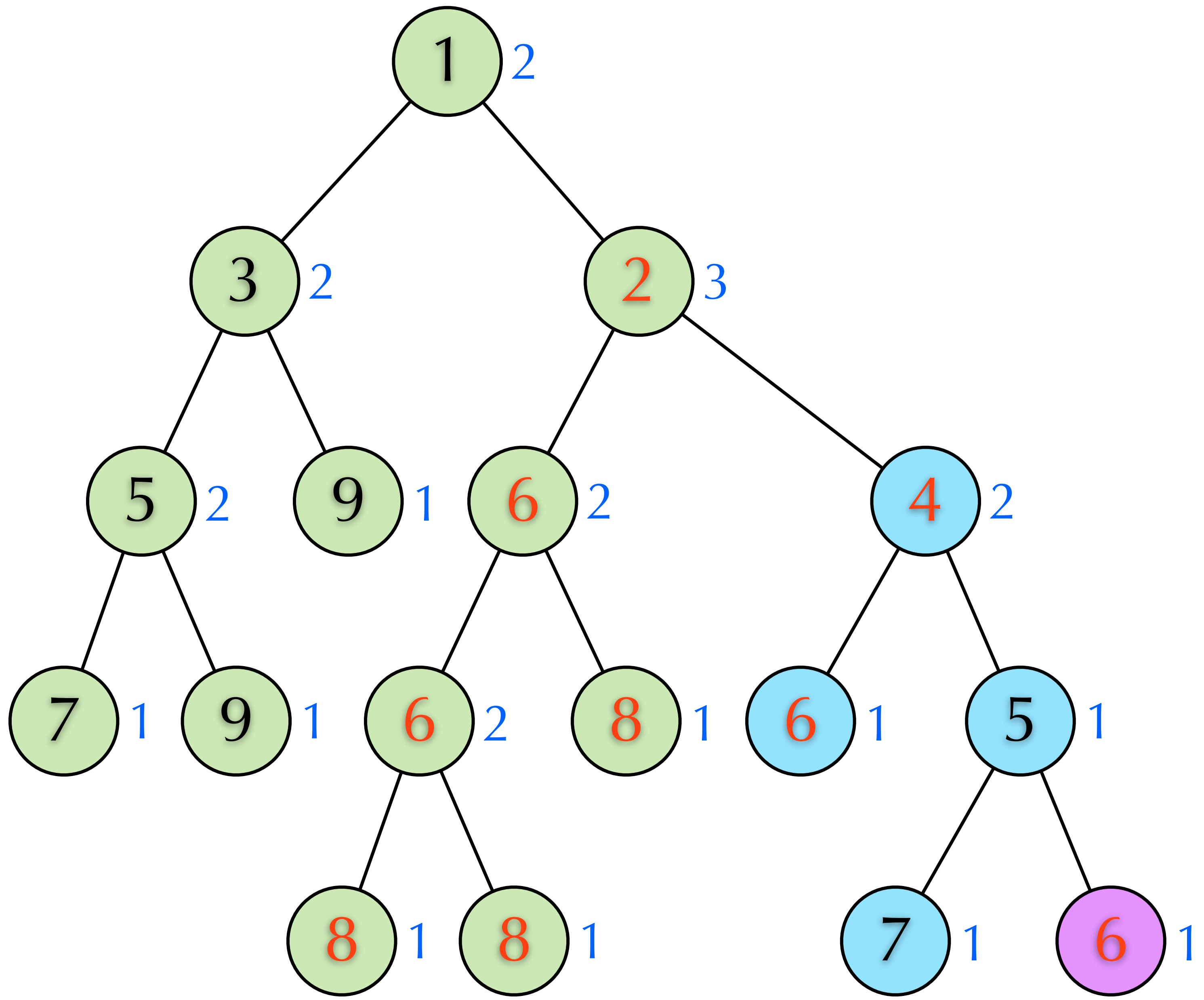
Meld: Phase 1



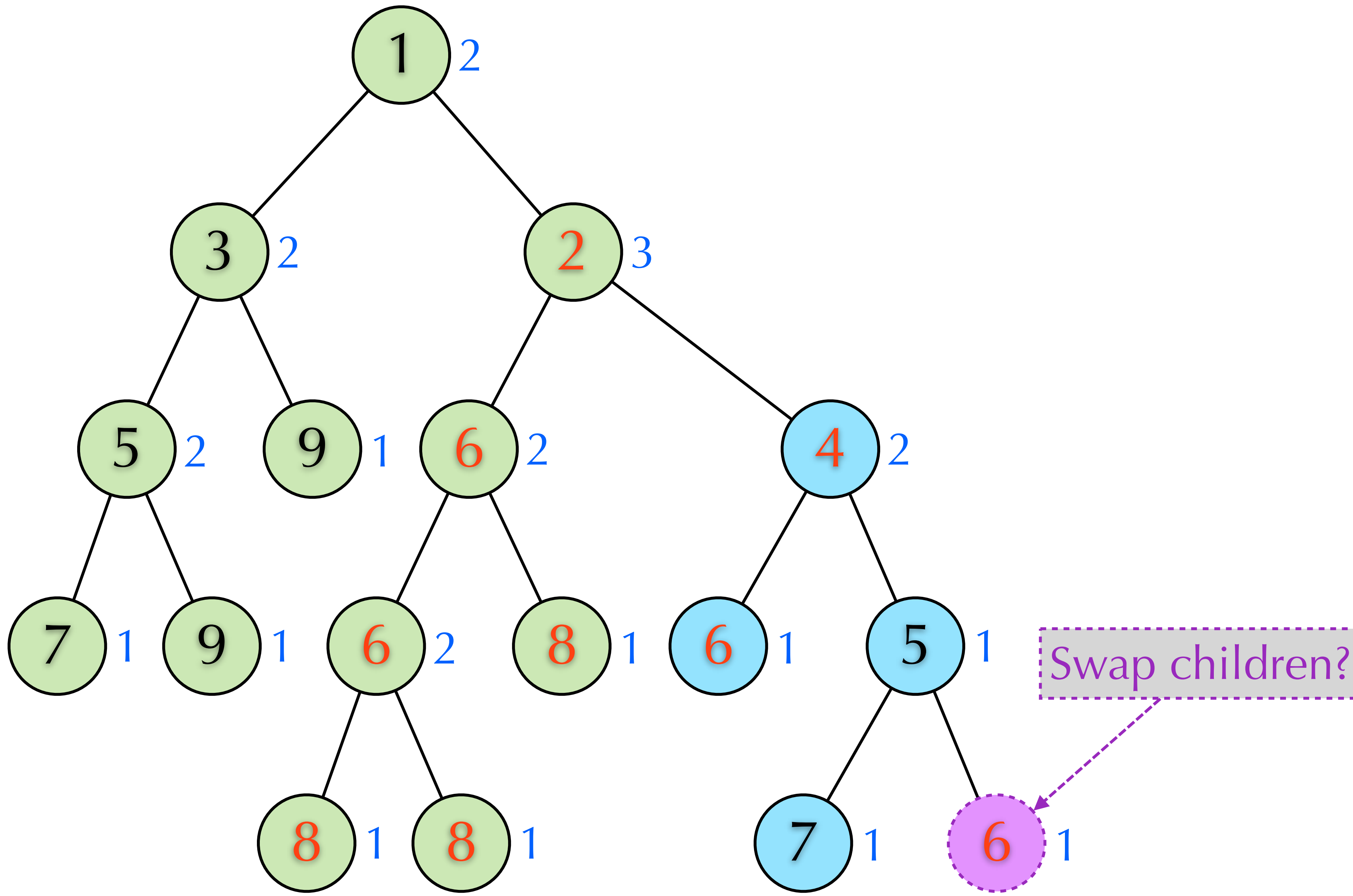
Meld: Phase 1



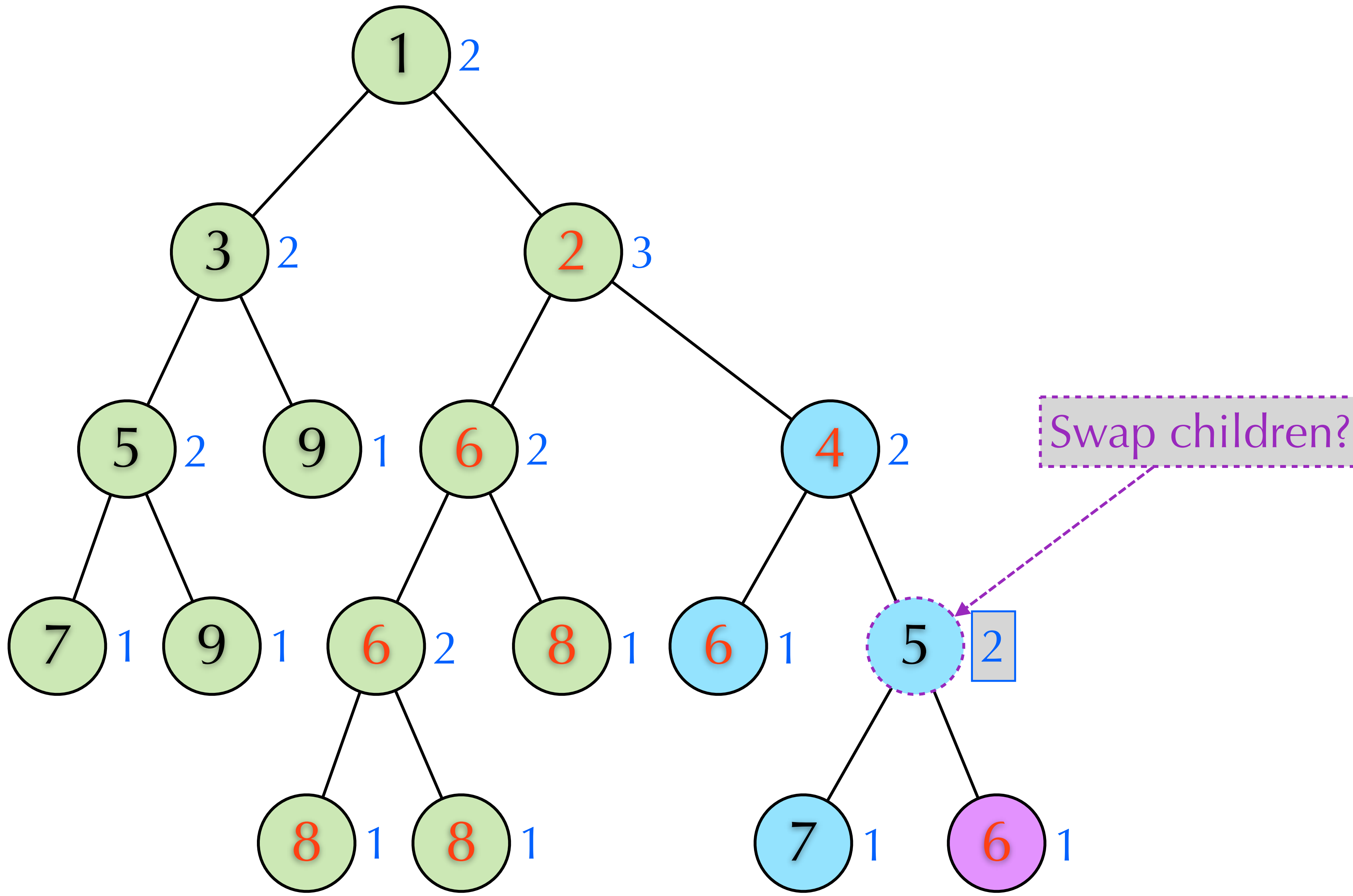
Meld: Phase 1



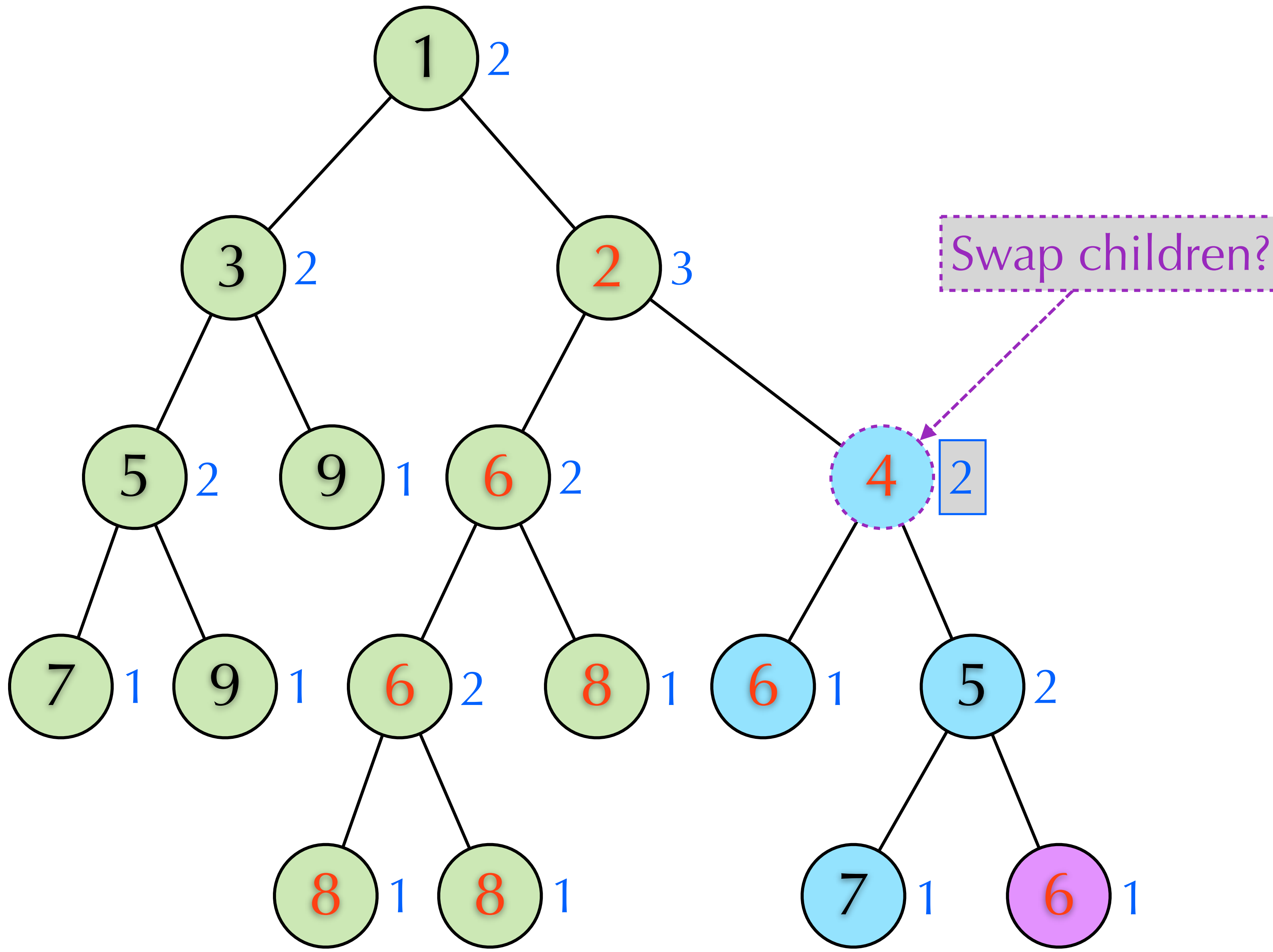
Meld: Phase 2



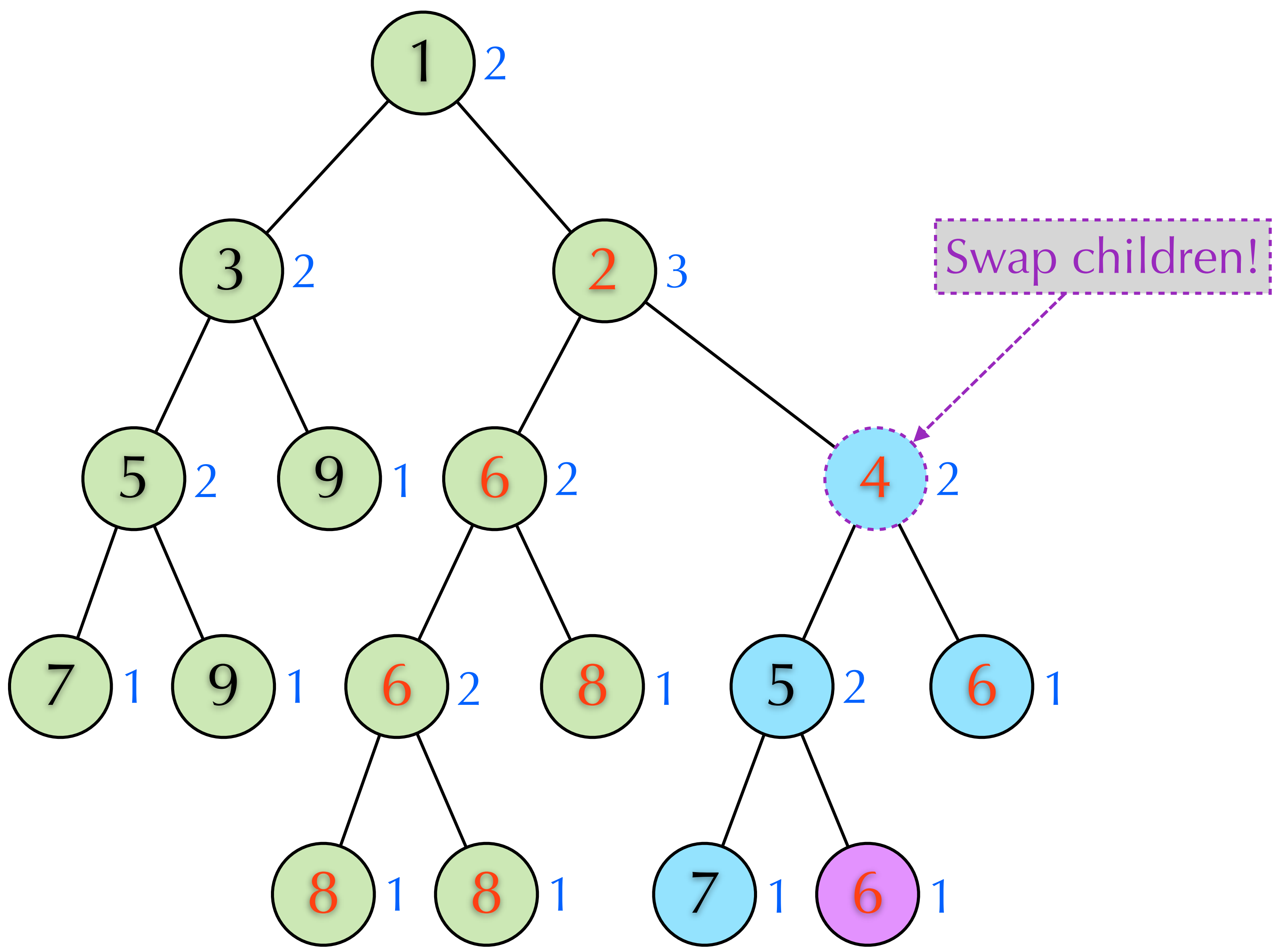
Meld: Phase 2



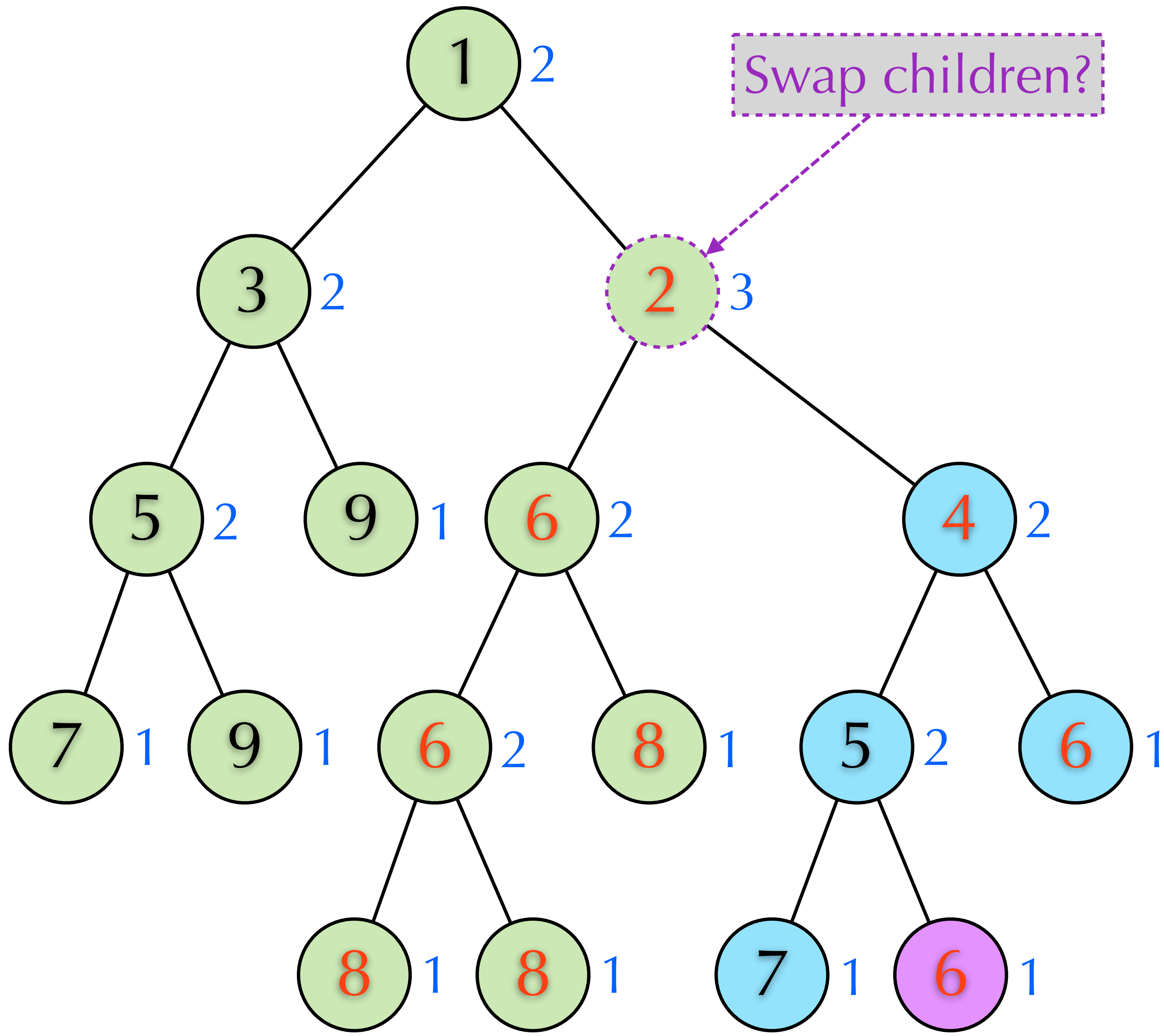
Meld: Phase 2



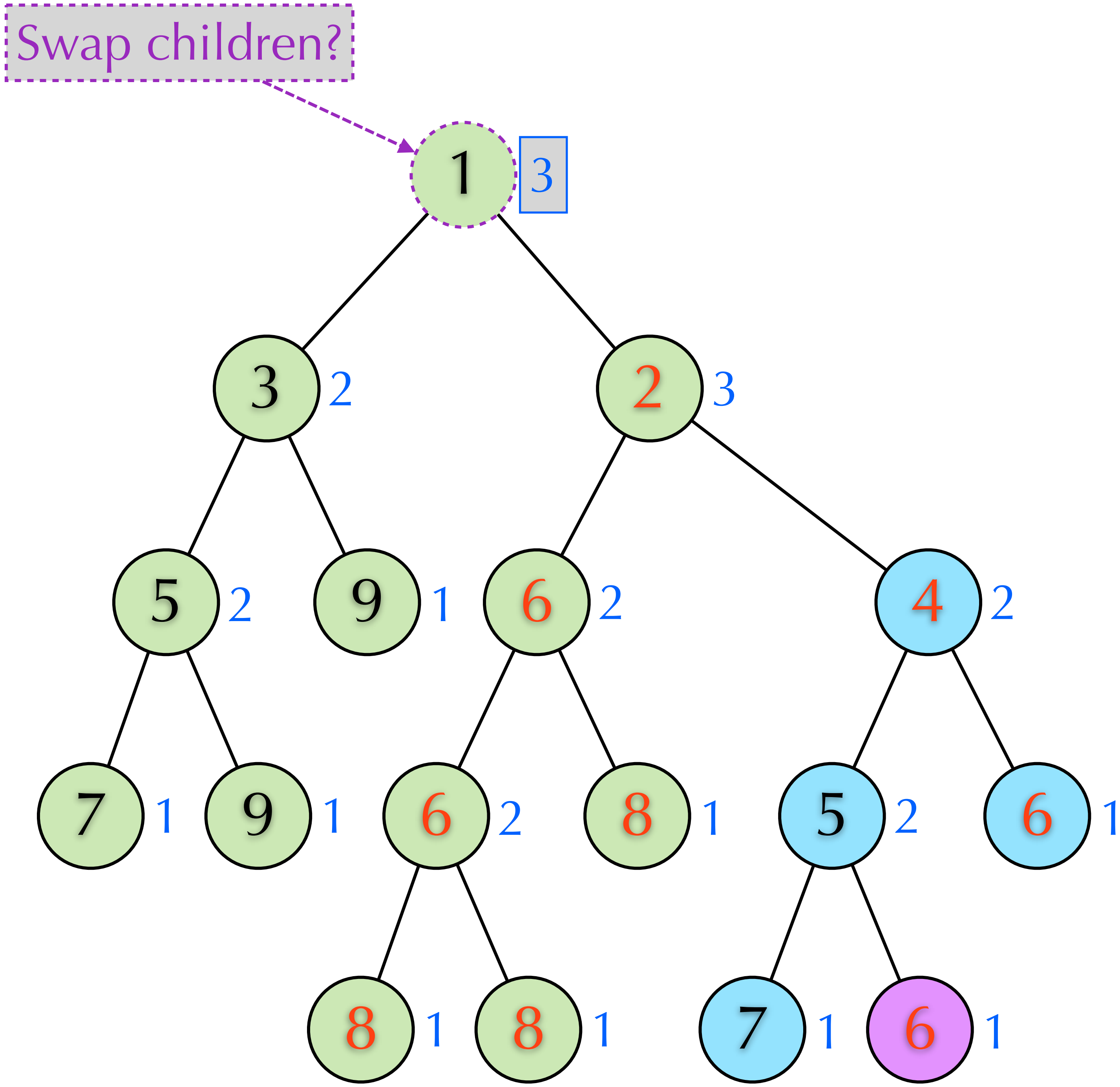
Meld: Phase 2



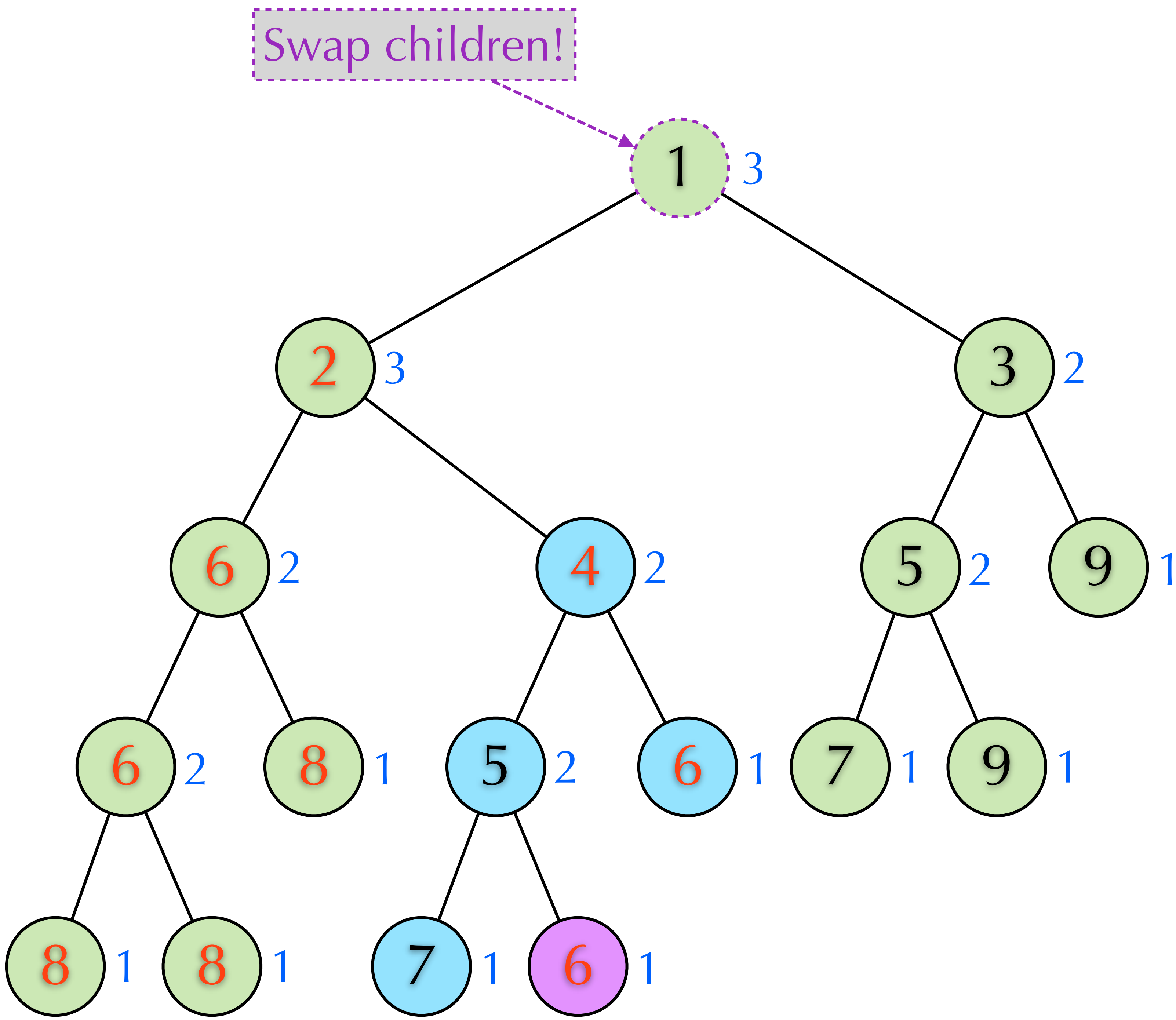
Meld: Phase 2



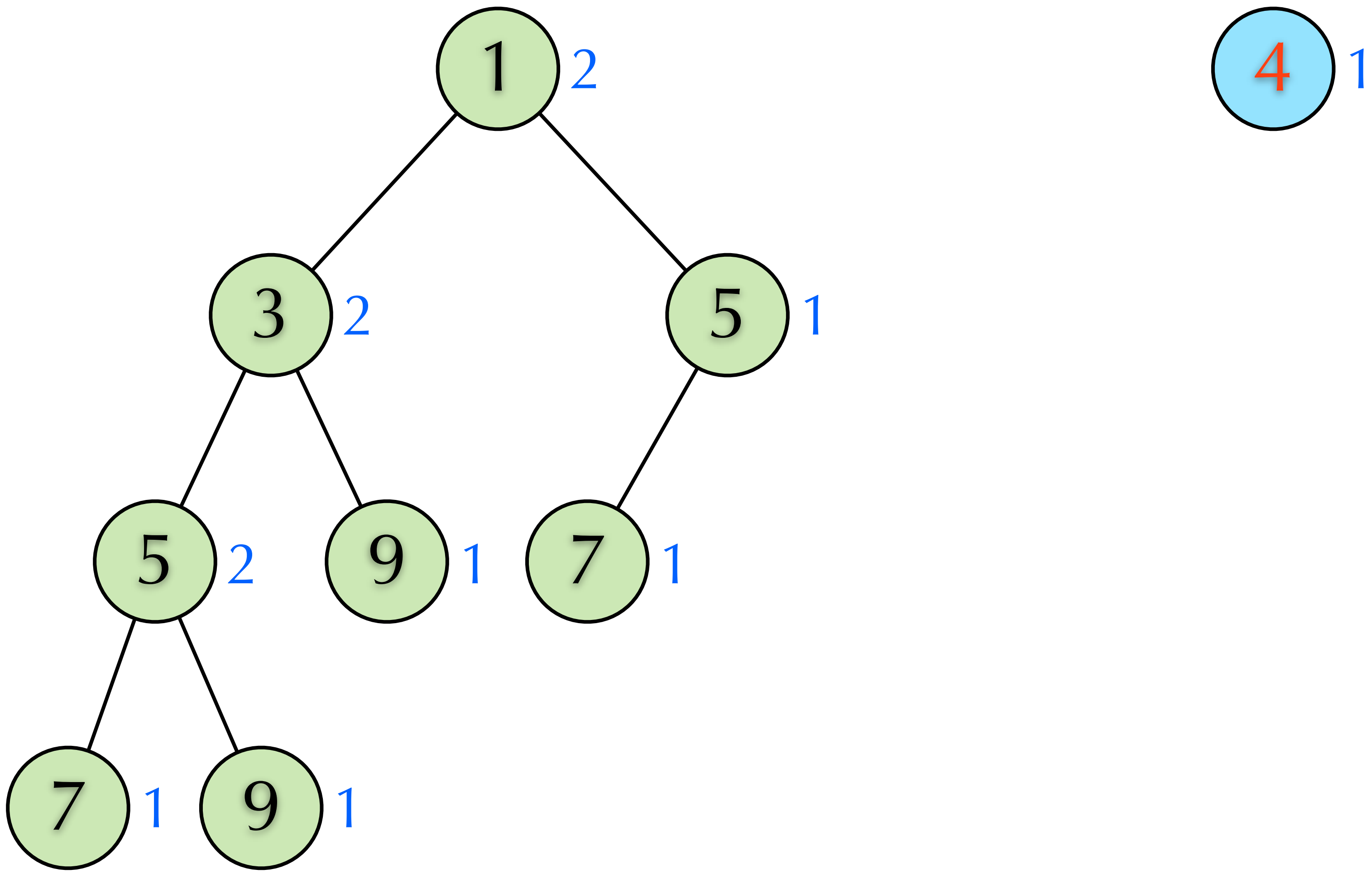
Meld: Phase 2



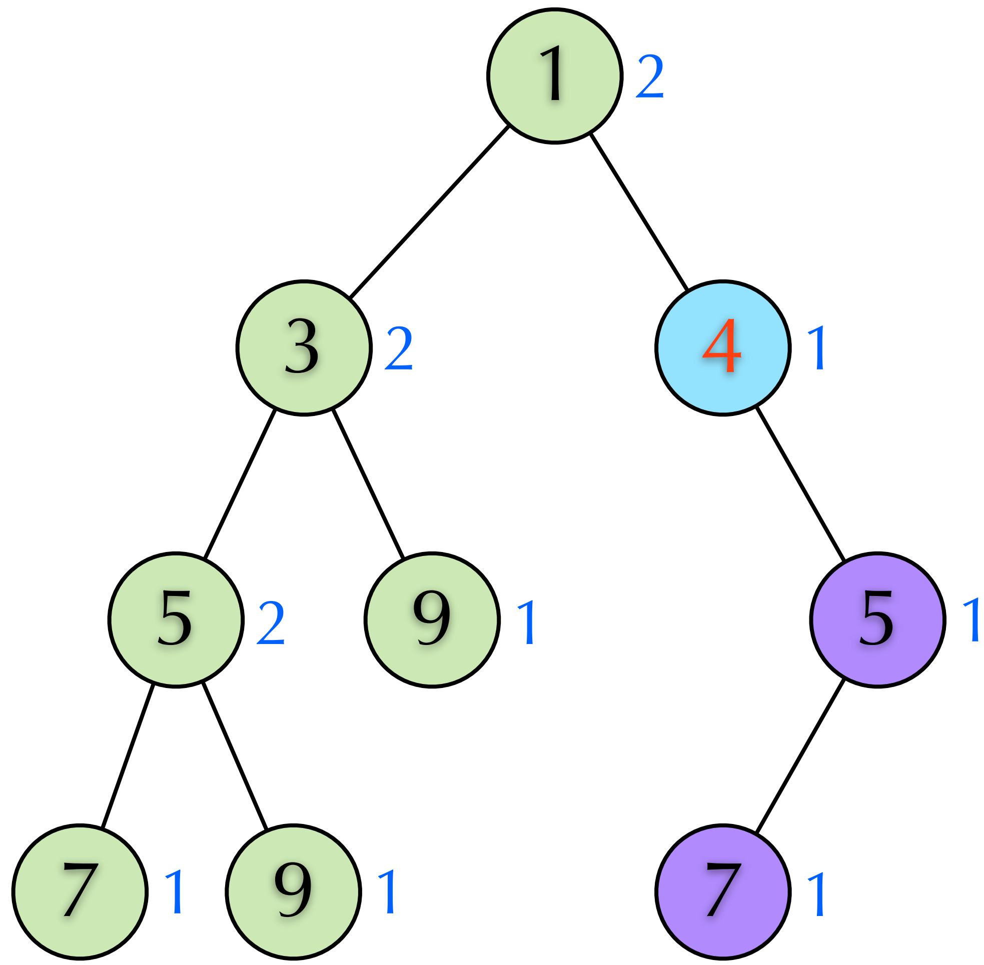
Meld: Phase 2



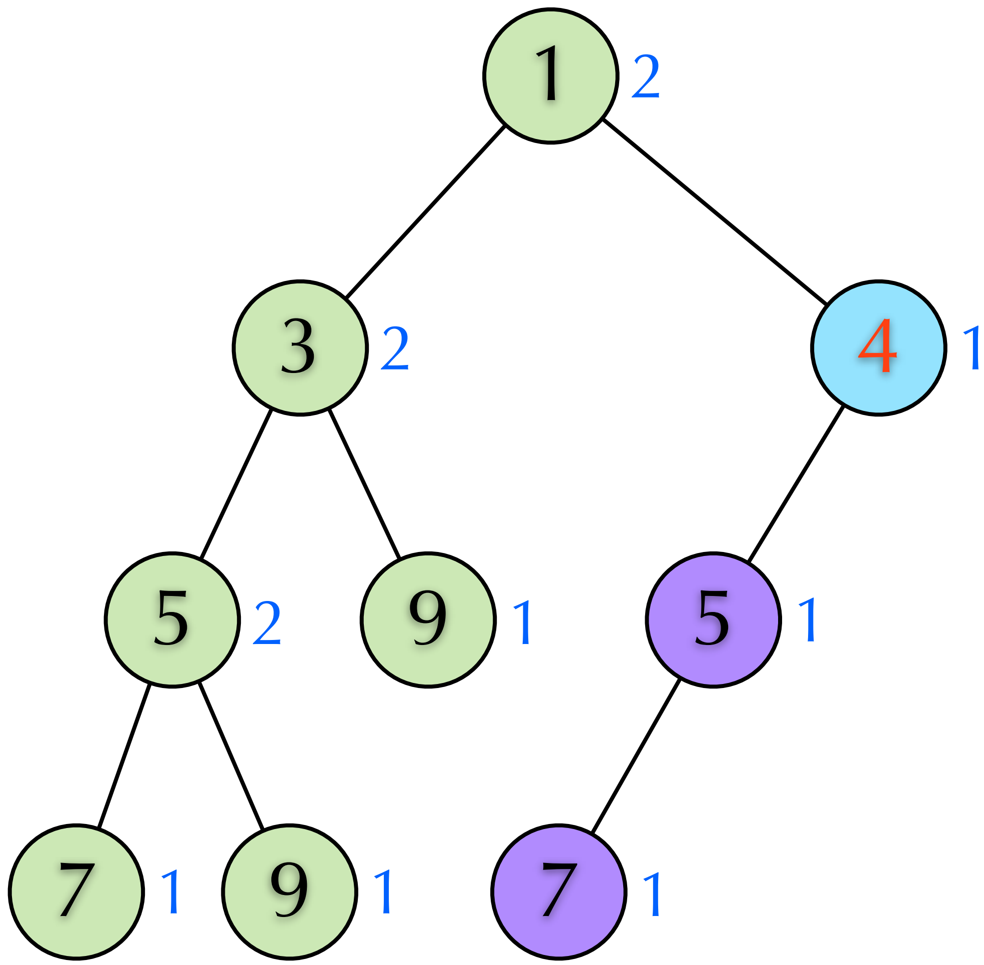
Insert 4



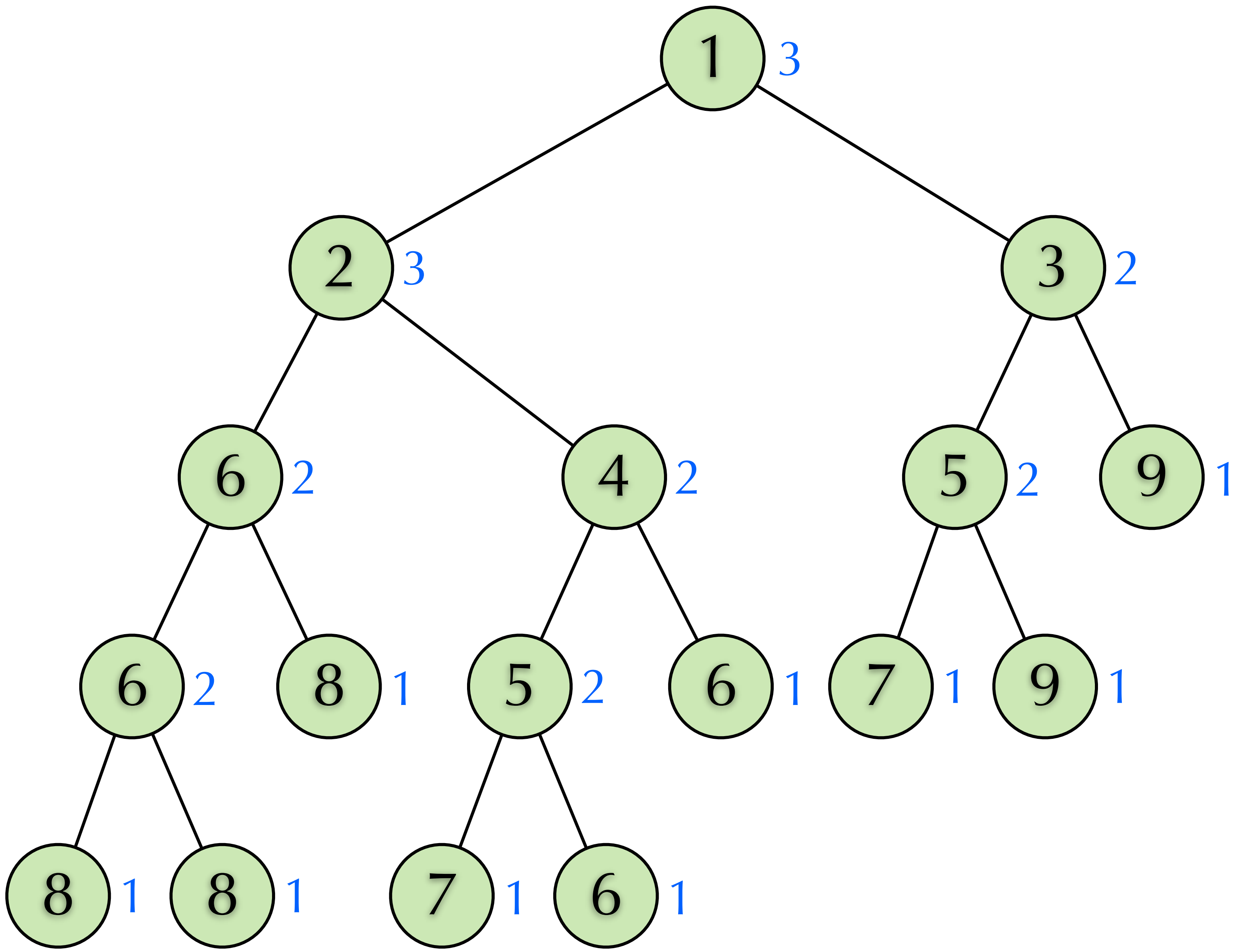
Insert 4



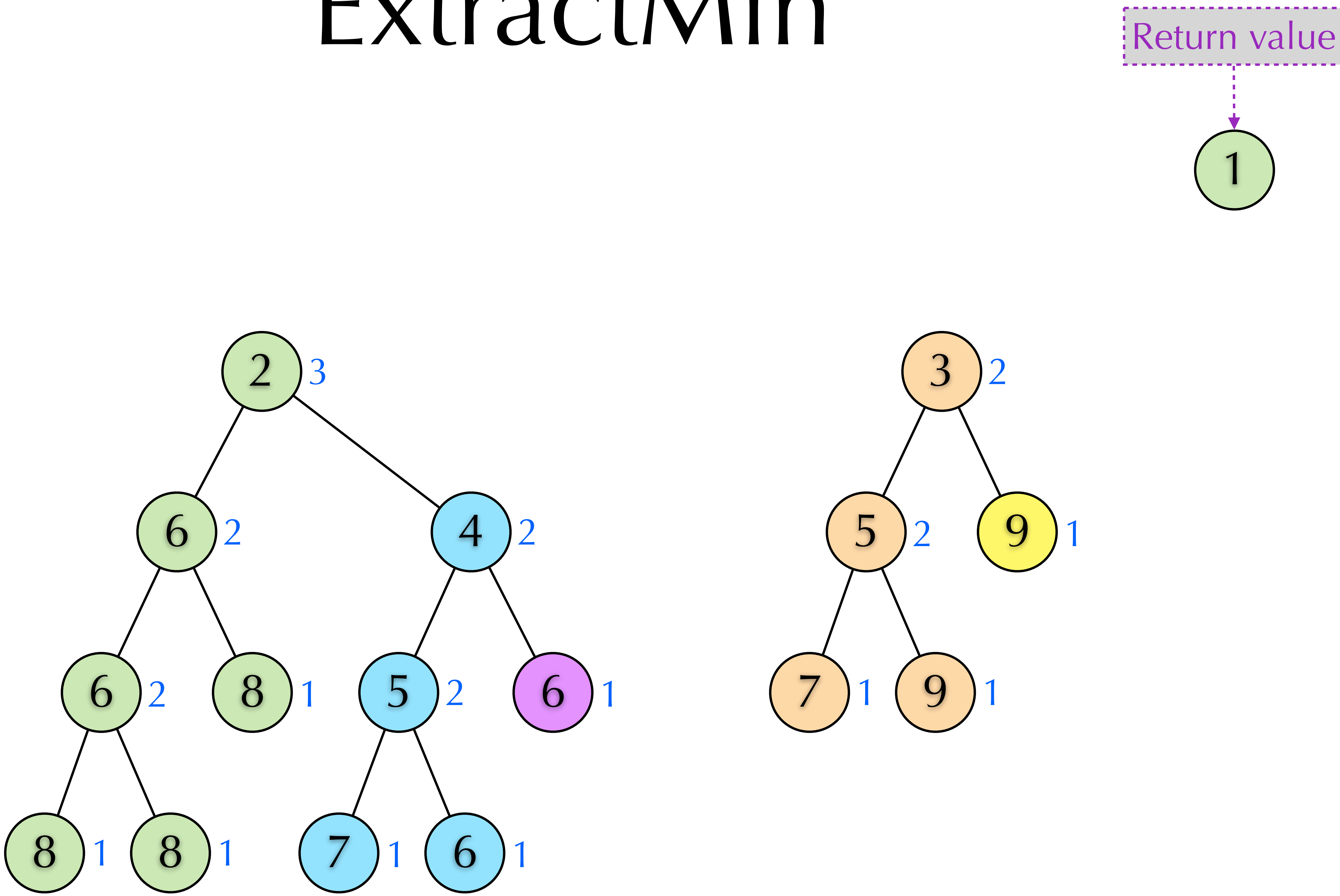
Insert 4



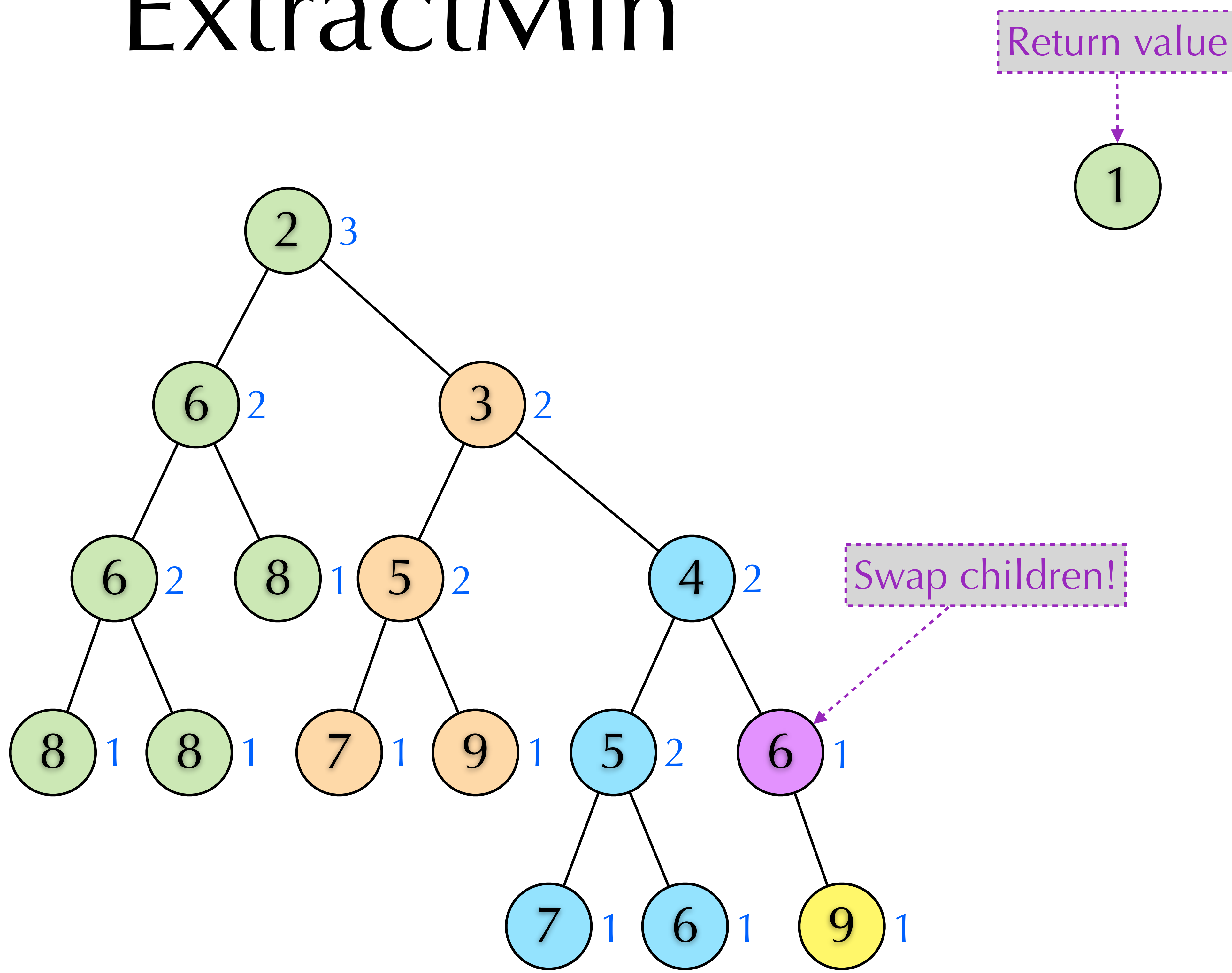
ExtractMin



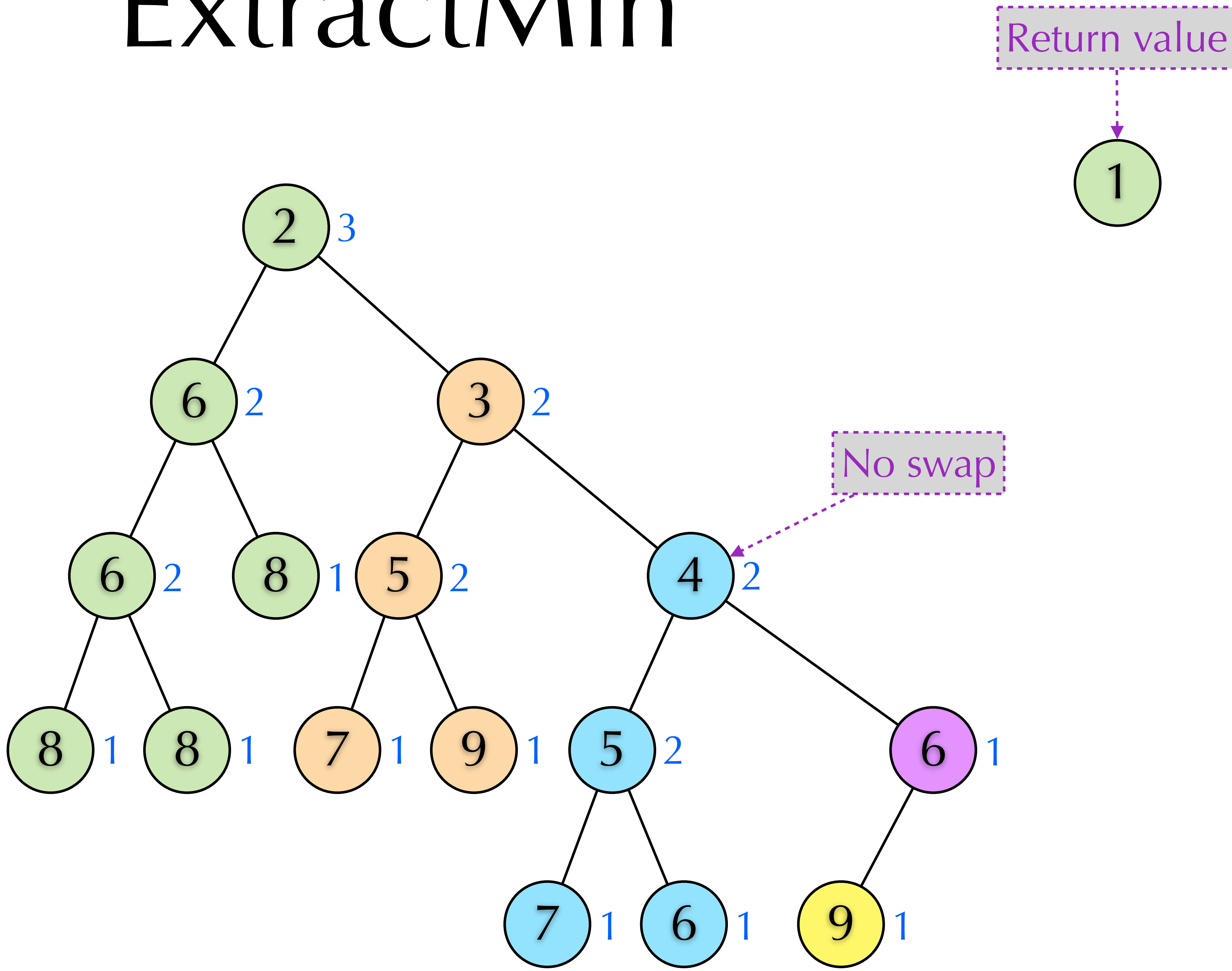
ExtractMin



ExtractMin

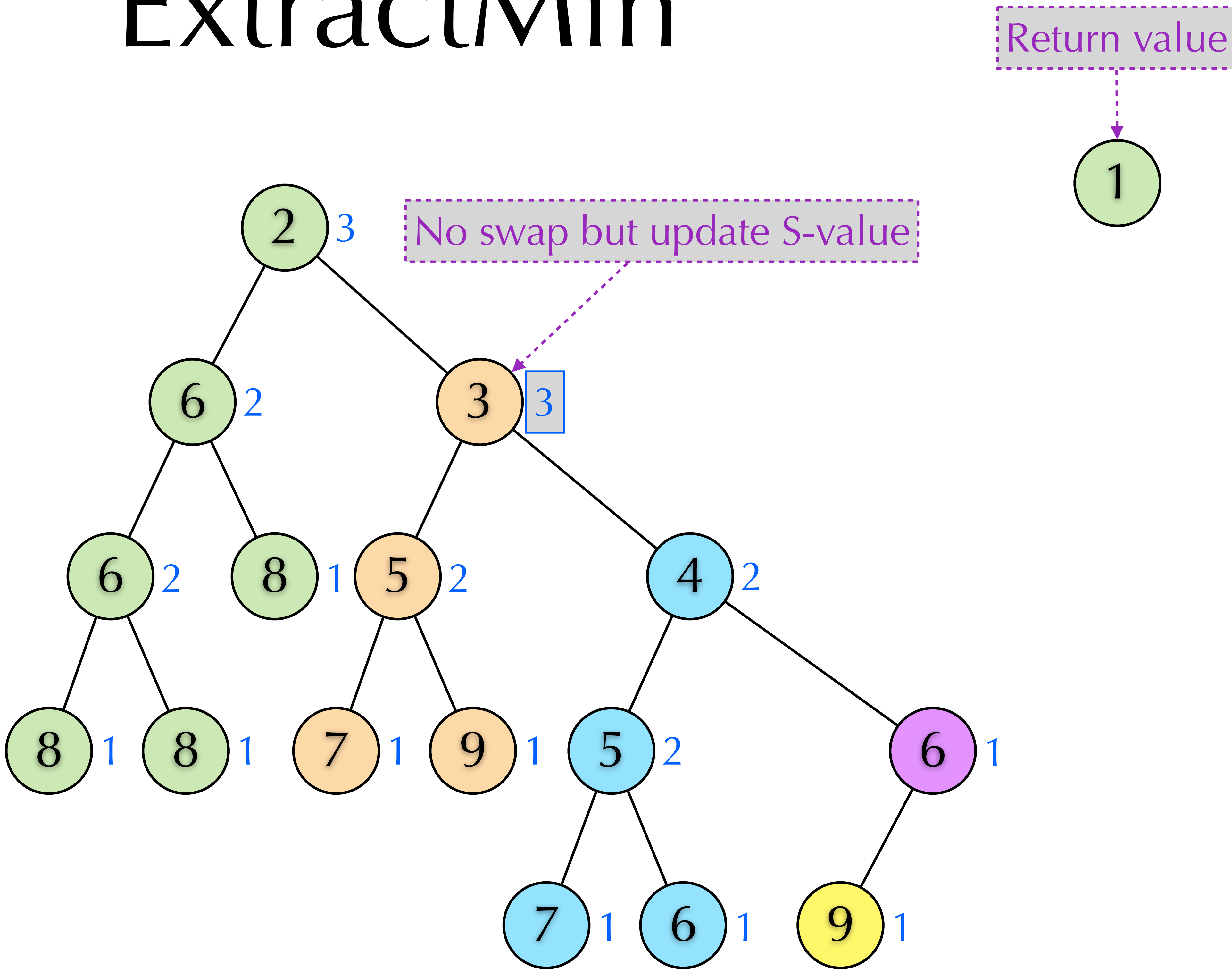

Return value
1
2
3
6
2
3
2
6
2
8
1
5
2
4
2
Swap children!
8
1
8
1
7
1
9
1
5
2
6
1
7
1
6
1
9
1

ExtractMin

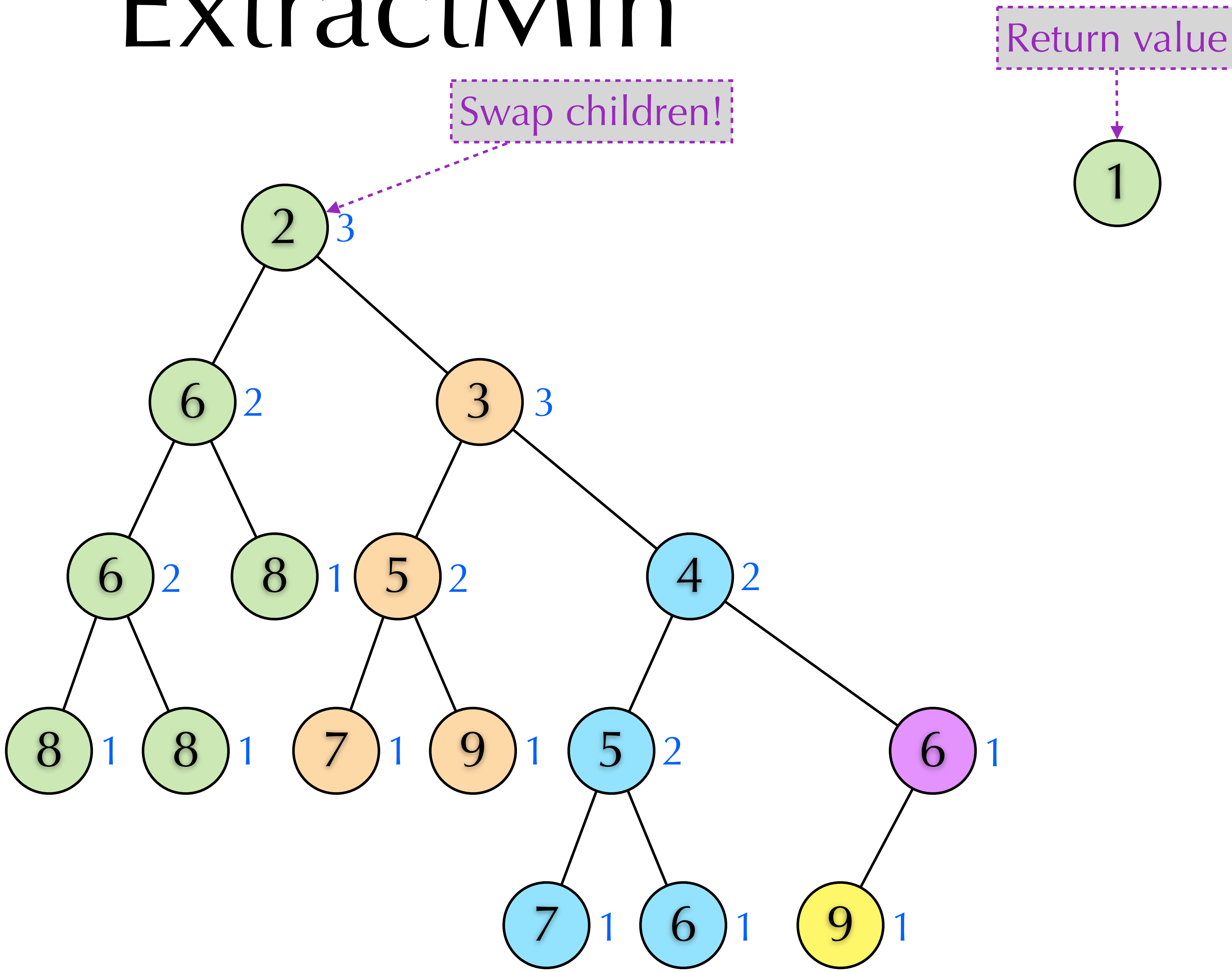


Return value
1
No swap
2
3
6
2
3
2
6
2
8
1
5
2
4
2
8
1
8
1
7
1
9
1
5
2
6
1
7
1
6
1
9
1

ExtractMin

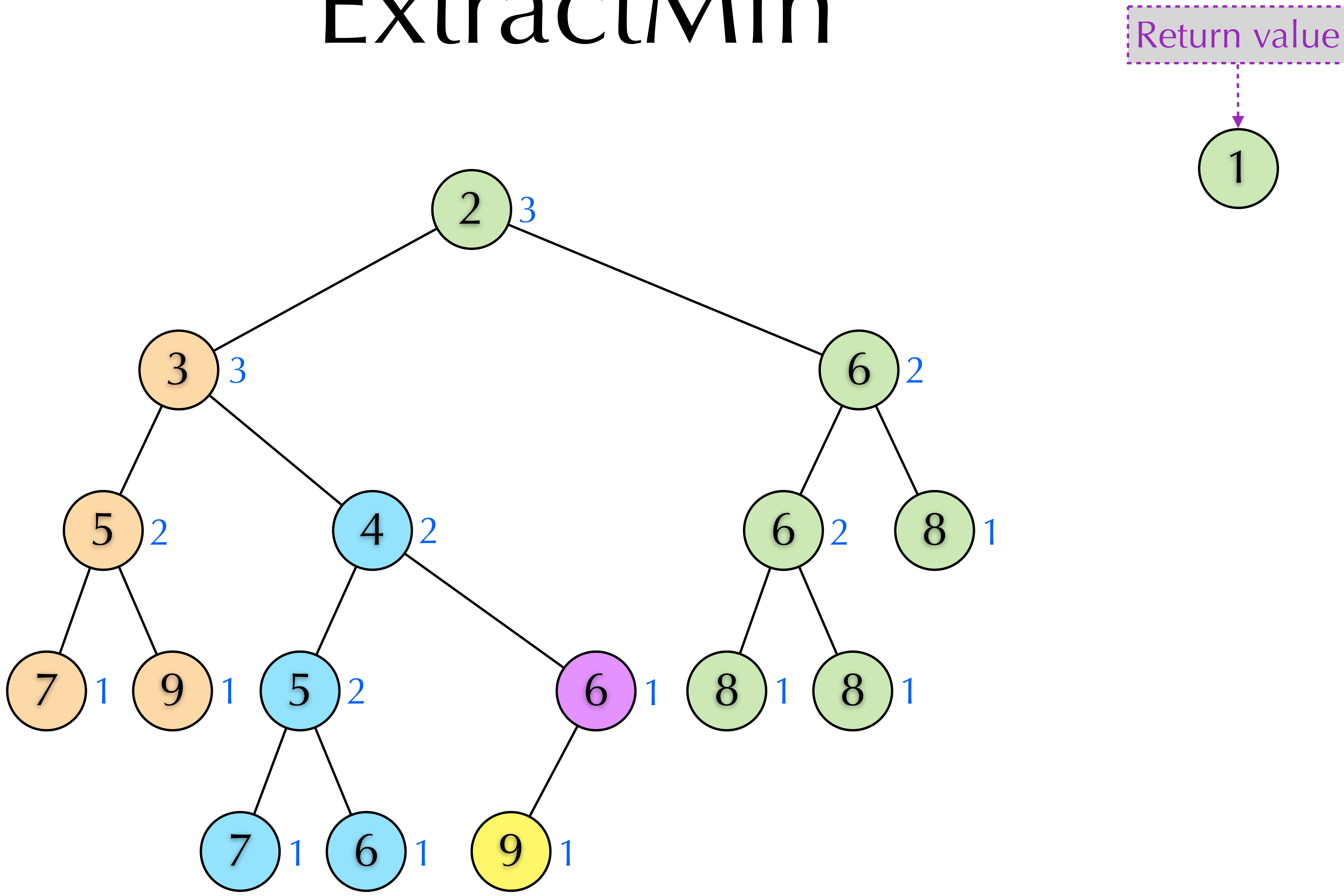

Return value
1
No swap but update S-value
2
3
6
2
3
3
6
2
8
1
5
2
4
2
8
1
8
1
7
1
9
1
5
2
6
1
7
1
6
1
9
1

ExtractMin



Swap children!
Return value
1
2 3
6 2
3 3
6 2
8 1
5 2
4 2
8 1
8 1
7 1
9 1
5 2
6 1
7 1
6 1
9 1

ExtractMin



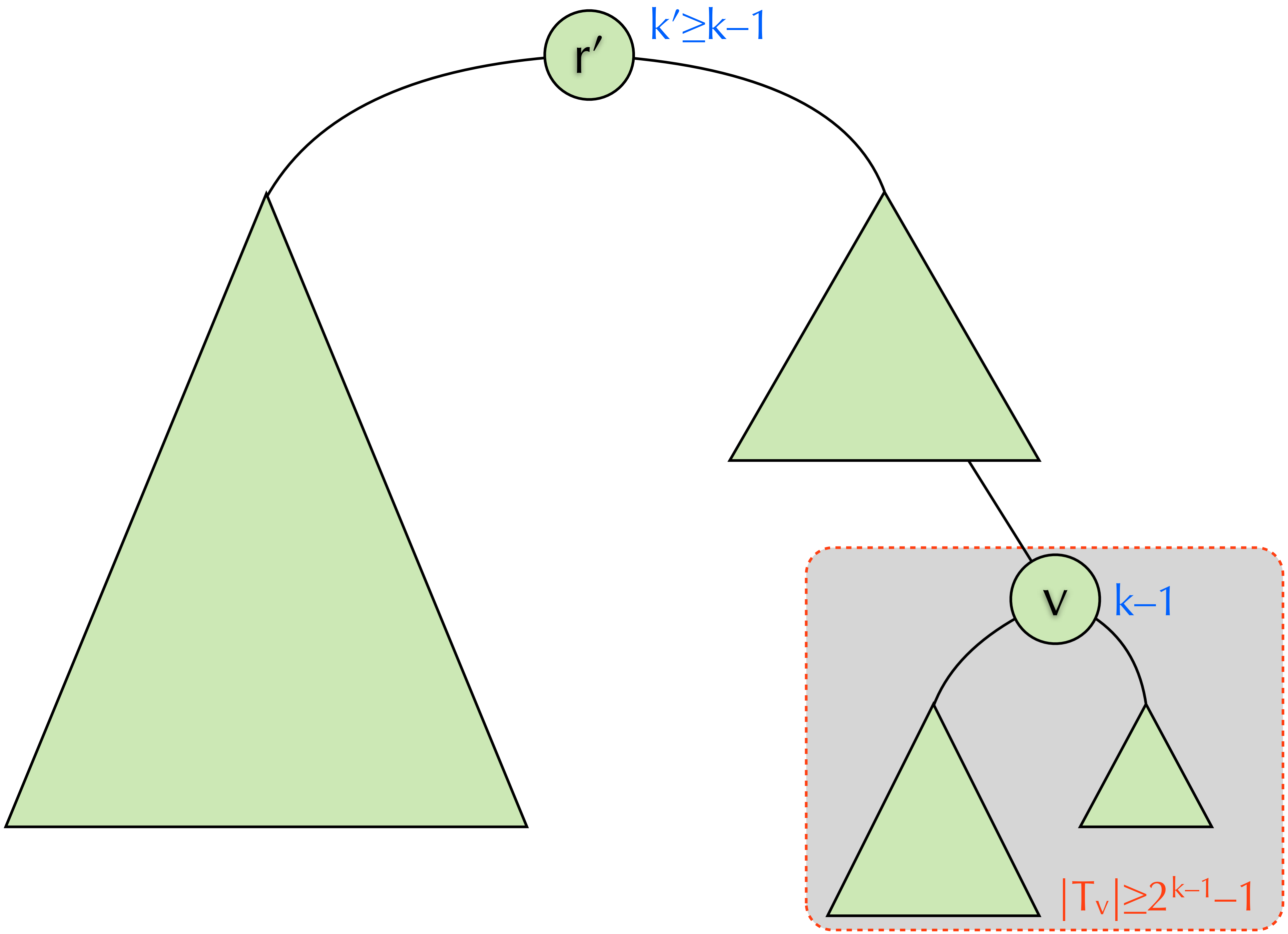
Complexity of Meld

- It is obvious that melding two leftist tree takes $O(S(r_1)+S(r_2))$-time where r_1 and r_2 are roots.
- Target: O(logn)
- The rest part is to prove the size of a leftist tree is at least $2^{S(r)}-1$, where r is its root.

Complexity of Meld

- Prove $|T| \geq 2^{S(r)} - 1$ by induction
- Basis: $S(r)=1$
  - Since the tree has a root r and $2^{S(r)} - 1 = 1$, the statement is true.
- Induction hypothesis:
  - The statement is true for $S(r) < k$.
- An important observation:
  - If $S(r') \geq k-1$ where r' is the root of T', then we have $|T'| \geq 2^{k-1} - 1$. (By induction hypothesis)

Important Observation



Complexity of Meld

- Inductive step: $S(r)=k$
  - Recall: For $v \in T$, we have $S(v.L) \geq S(v.R)$ & $S(r)=\min(S(r.L),S(r.R))+1=S(r.R)+1$
  - Note that $S(r.R)=k-1<k$. By induction hypothesis, the right subtree has size at least $2^{k-1}-1$.
  - Note that $S(r.L) \geq S(r.R)$. By the important observation, the left subtree has size at least $2^{k-1}-1$, too.
  - $|T| \geq 1+2^{k-1}-1+2^{k-1}-1=2^k-1$. We are done.

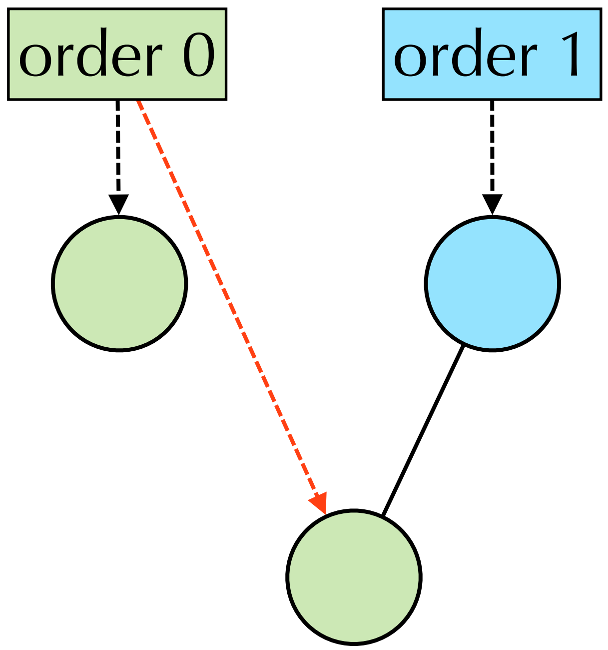
Homework 11.1

- a) How to initialize an n-element leftist tree?
- b) Suppose we redefine the S-value to the size of the subtree. Will the meld operation still work? If yes, what is the time complexity of meld operation?
- c) How to implement decrease key? What is the time complexity of your implementation?
- d) What is a skew heap? Compare skew heaps and leftist trees.

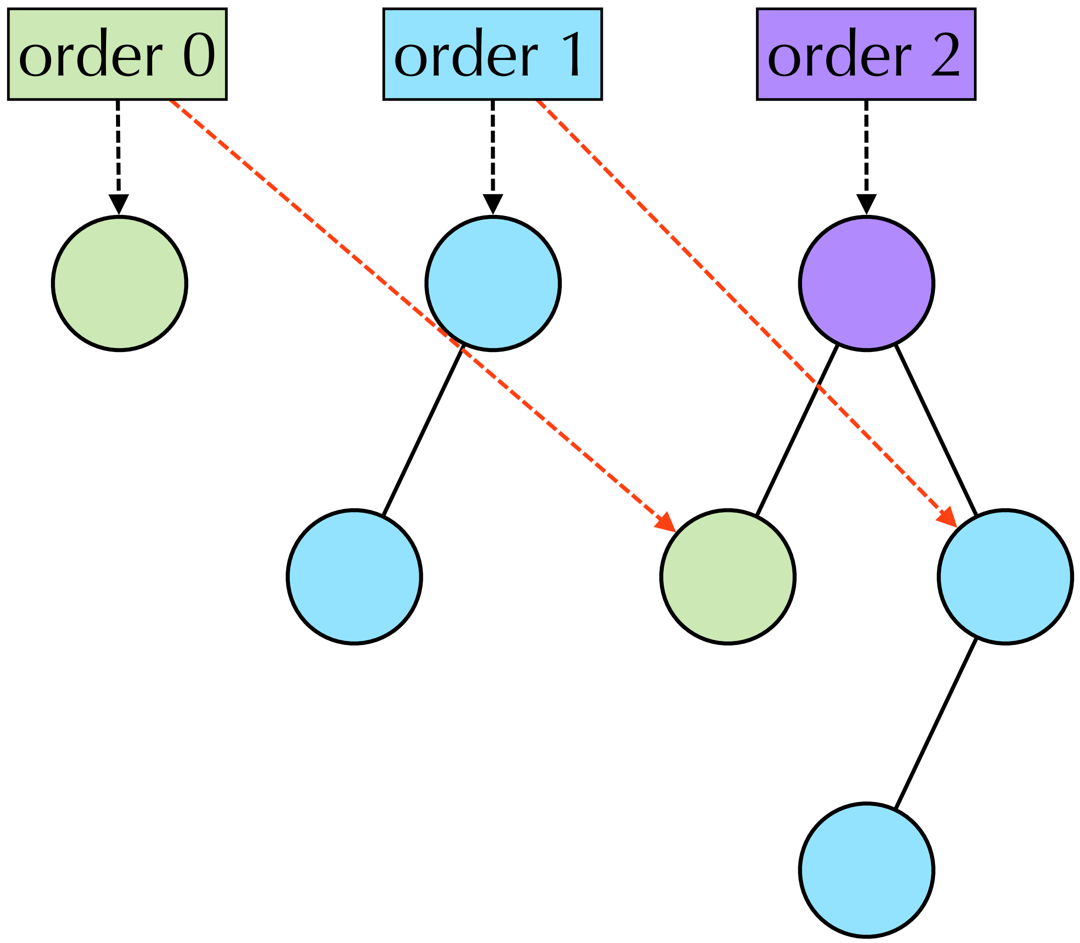
Binomial Tree

- A recursive tree structure
- Binomial tree of order 0: A single root r
- Binomial tree of order k:
  - Has a root r
  - r has k children $c_0, \ldots, c_{k-1}$ such that c_i is the root of a binomial tree of order i.
  - Has exactly 2^k nodes.

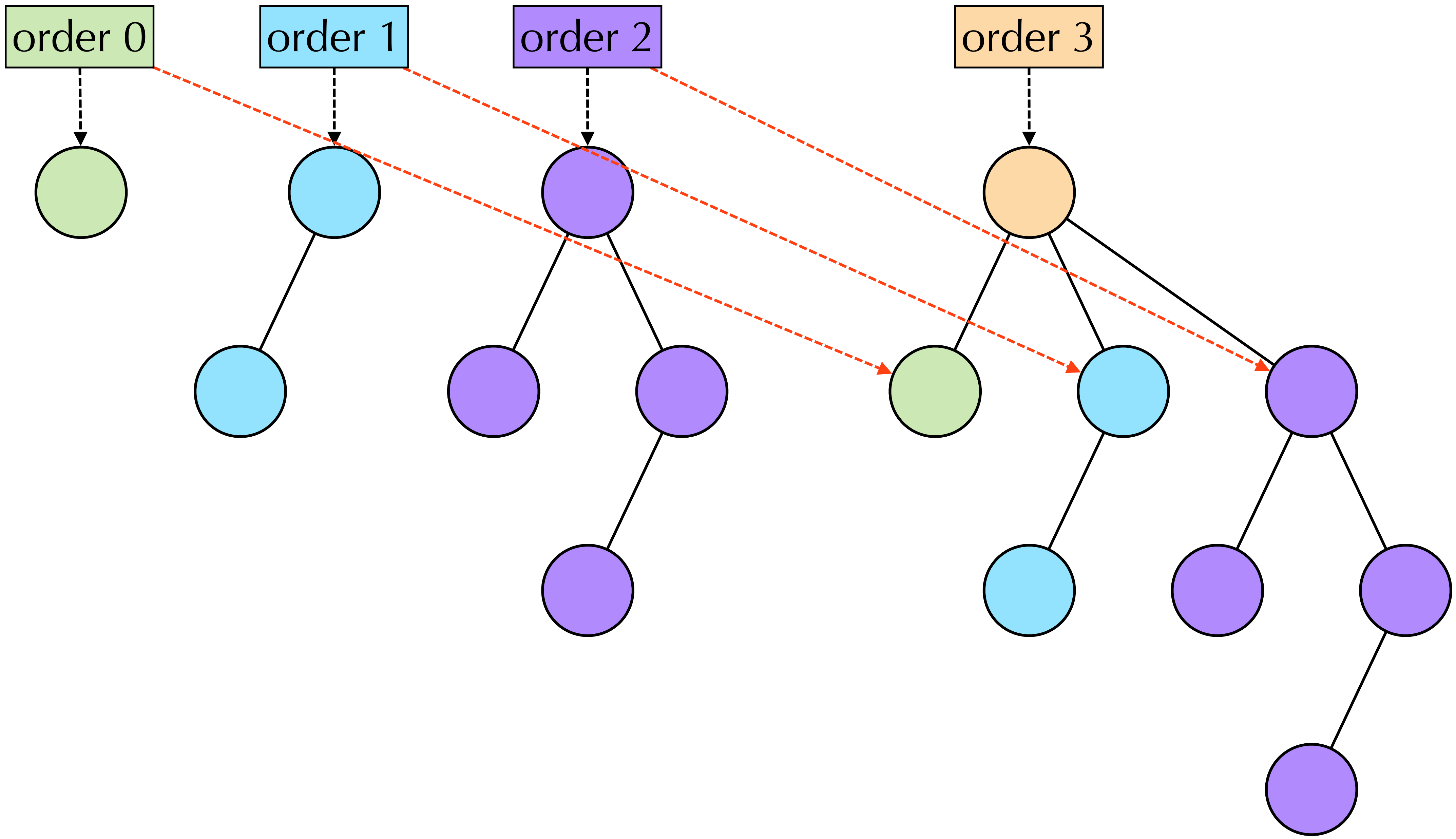
Binomial Tree: Definition



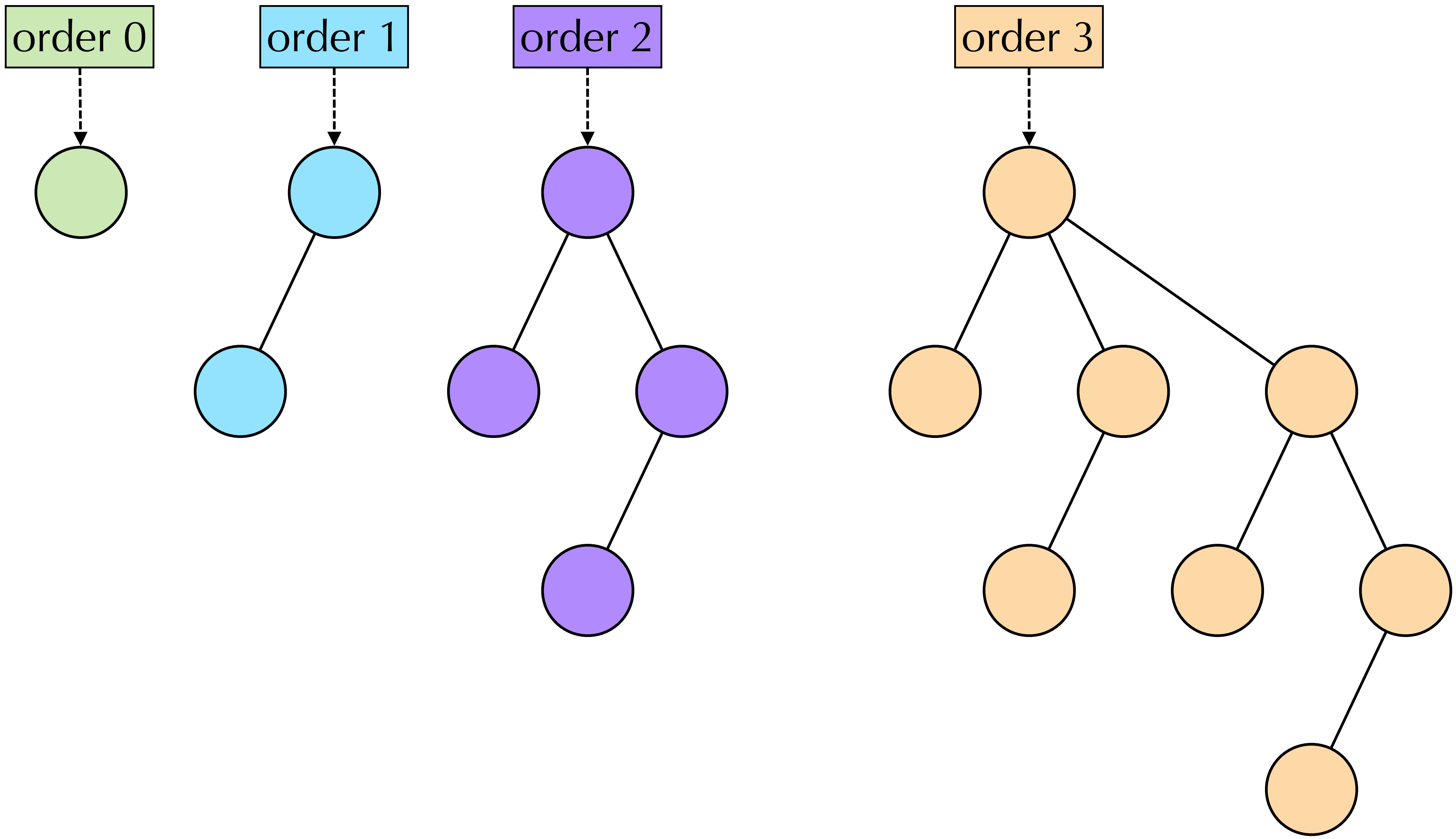
Binomial Tree: Definition



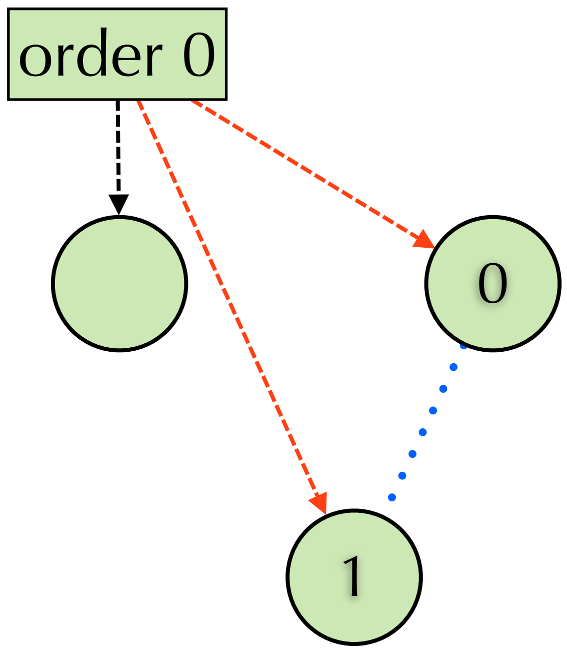
Binomial Tree: Definition



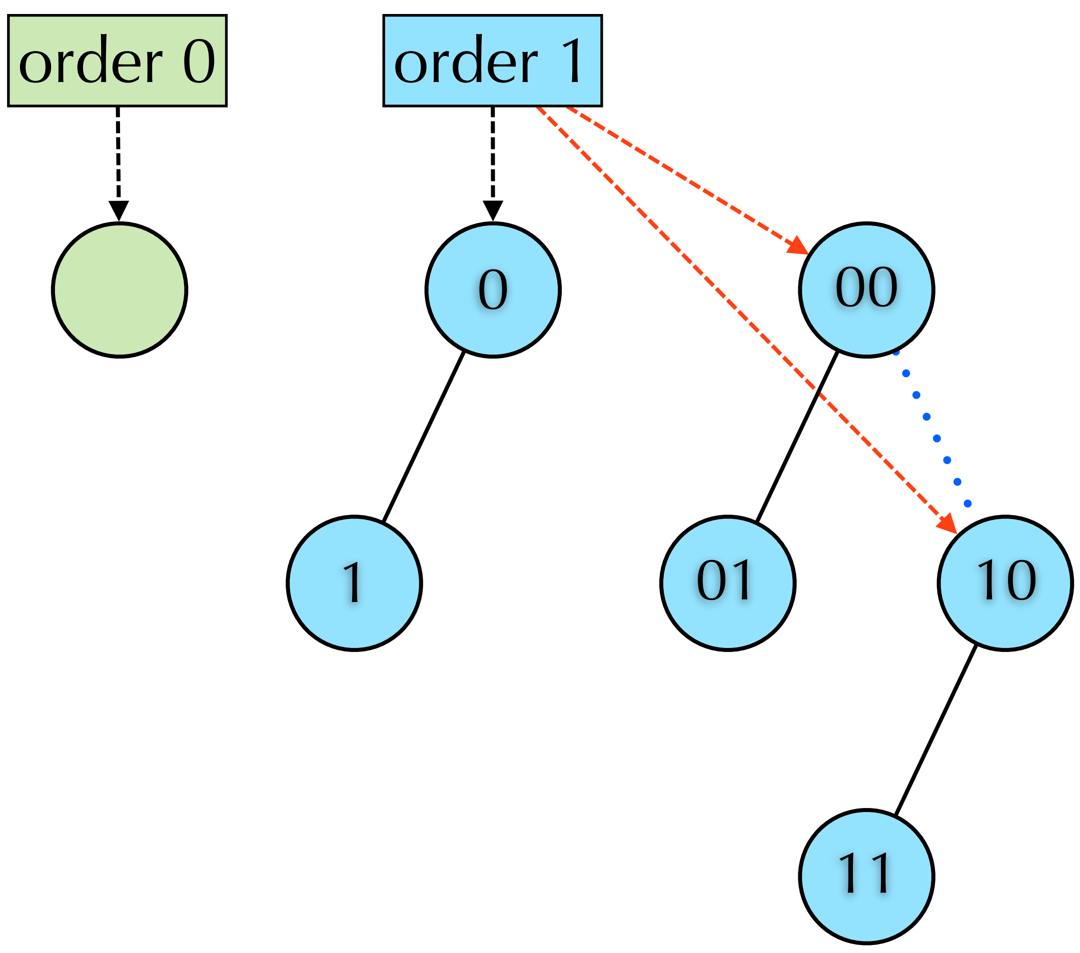
Binomial Tree: Definition



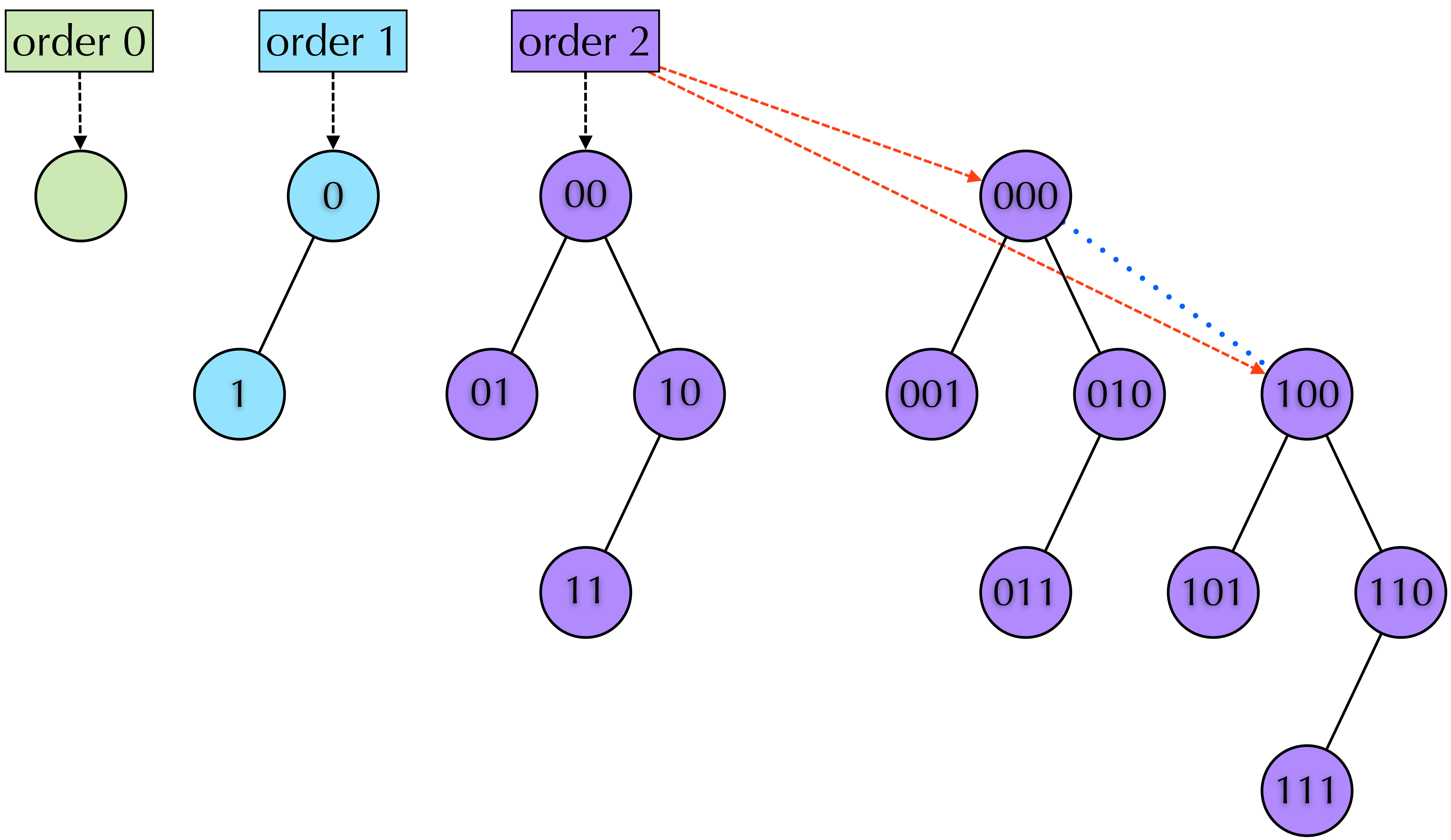
Alternative Definition



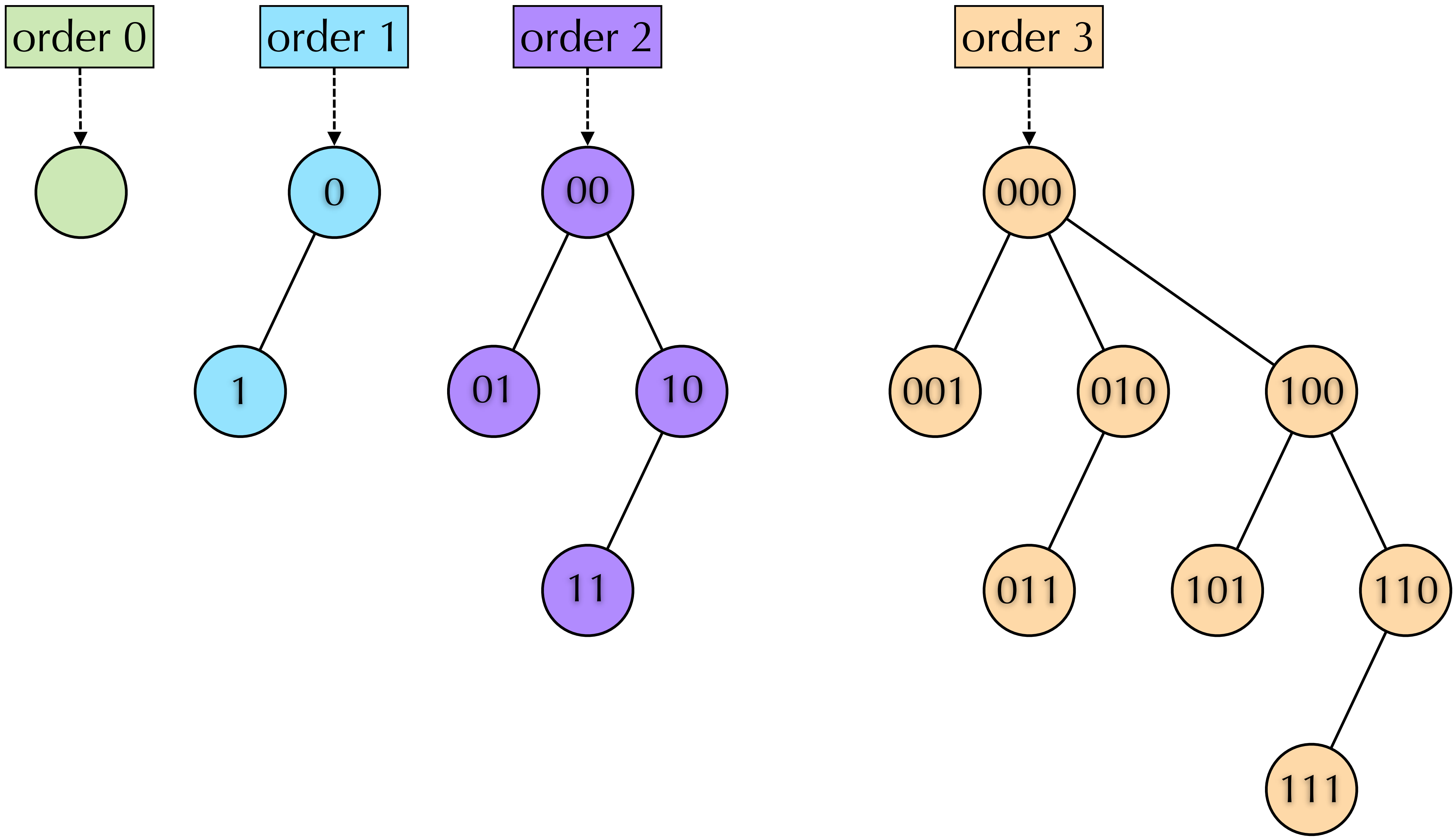
Alternative Definition



Alternative Definition



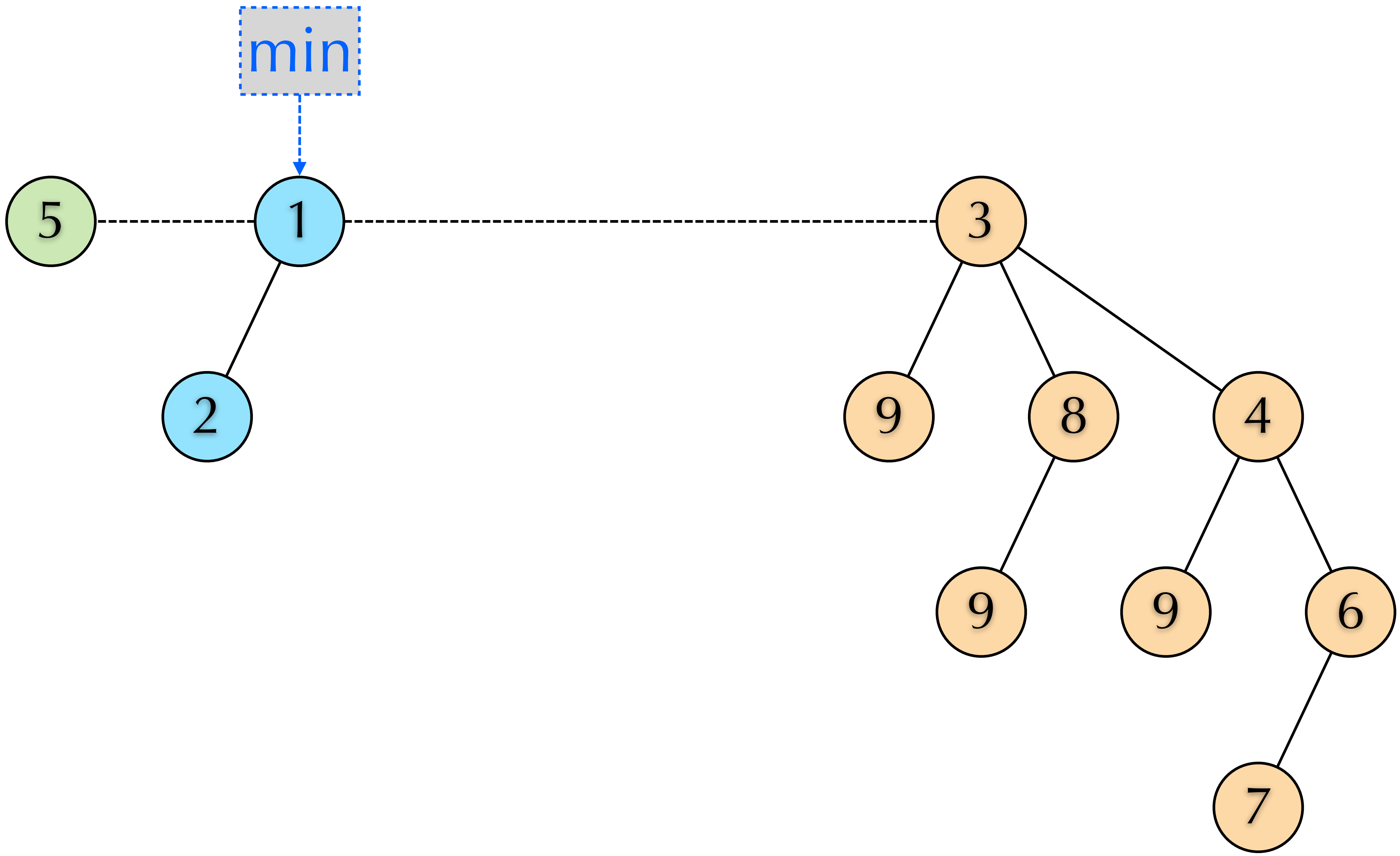
Alternative Definition



Binomial Heap

- An n-element binomial heap consists of several binomial trees
  - Each tree has the min heap property
  - Let $b_d b_{d-1} \ldots b_0$ be the binary representation of n. I.e., $n=\Sigma_{i \in [d]} b_i 2^i$. Then the binomial heap has b_i binomial tree of order i.
  - Stores a pointer to the binomial tree containing minimum element.

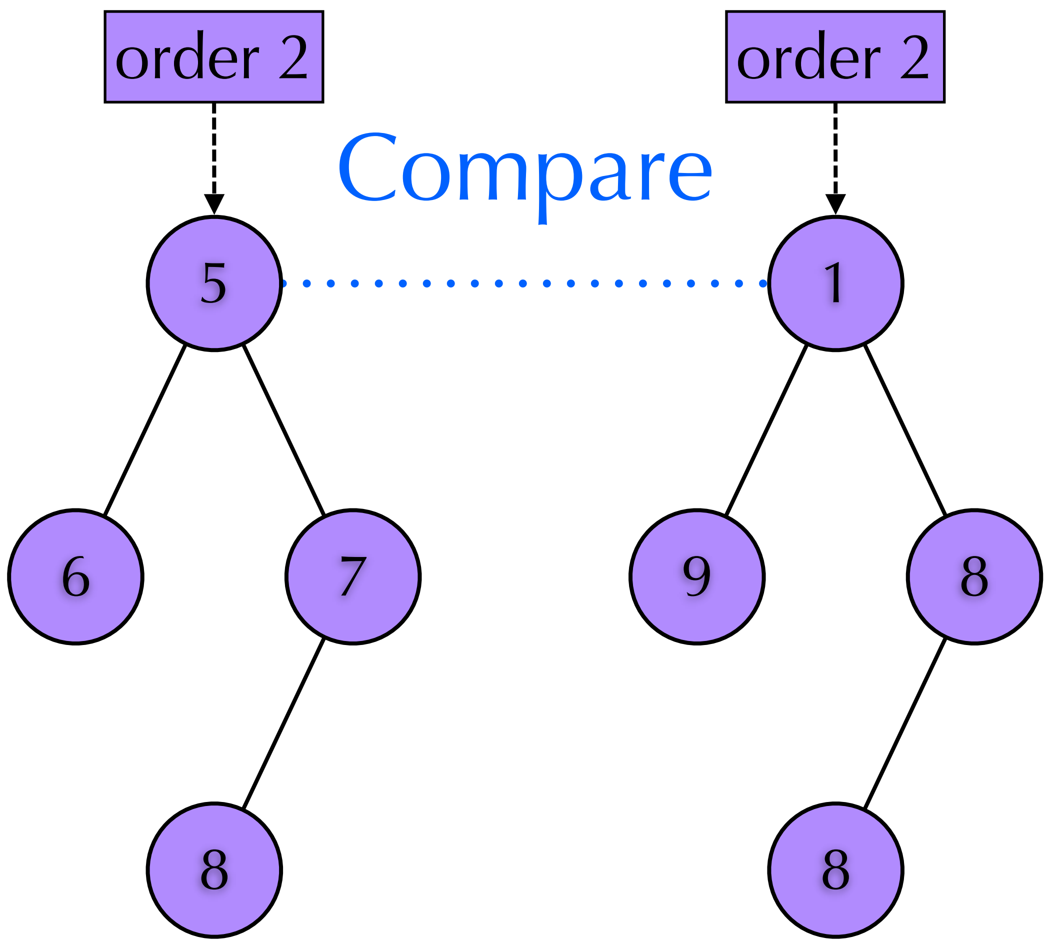
Example: 11 elements



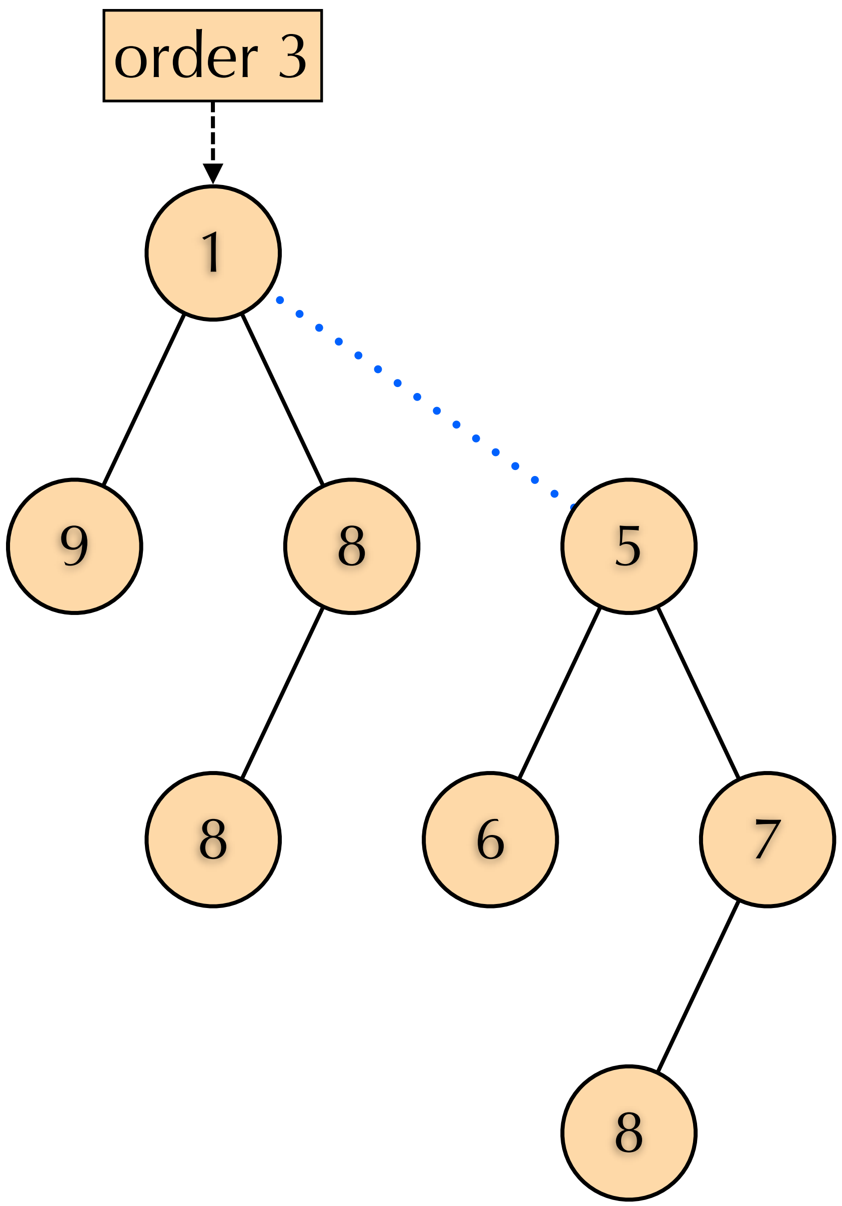
Merge Trees

- Merge binomial trees: (NOT Heaps!!)
  - Apply on two trees T_0 and T_1 of same order!
  - Output a binomial tree of order d+1 if the inputs are trees of order d.
  - Assume K_0 and K_1 is the keys of the roots of T_0 and T_1 respectively. If $K_i < K_{1-i}$, then make T_{1-i} as a subtree of T_i.
  - Time complexity: O(1)

Merge Trees: Example



Merge: Example



Merge Heaps

- D: the max order of binomial trees in $H_0=T_{0,0}\cup\ldots\cup T_{0,D}$ and $H_1=T_{1,0}\cup\ldots\cup T_{1,D}$.
  - If H_i has a binomial tree of order d, then T_i,d is the tree, otherwise $T_{i,d}=\emptyset$.
- Output a binomial heap $H=T_0\cup\ldots\cup T_{D+1}$ and the pointer to minimum element.
- Concept: Adding two binary numbers
  - Order: from low to high

Merge Heaps

Concept: adding two binary numbers

- $T_0 = \emptyset$

  For d = 0 to D do

  if $T_{0,d} \neq \emptyset \neq T_{1,d}$ then $T_{d+1} = \mathrm{merge}(T_{0,d}, T_{1,d})$;

  else if $T_{0,d} = \emptyset$ and $T_{1,d} \neq \emptyset \neq T_d$ then

  $T_{d+1} = \mathrm{merge}(T_{1,d}, T_d)$, $T_d = \emptyset$;

  else if $T_{1,d} = \emptyset$ and $T_{0,d} \neq \emptyset \neq T_d$ then

  $T_{d+1} = \mathrm{merge}(T_{0,d}, T_d)$, $T_d = \emptyset$;

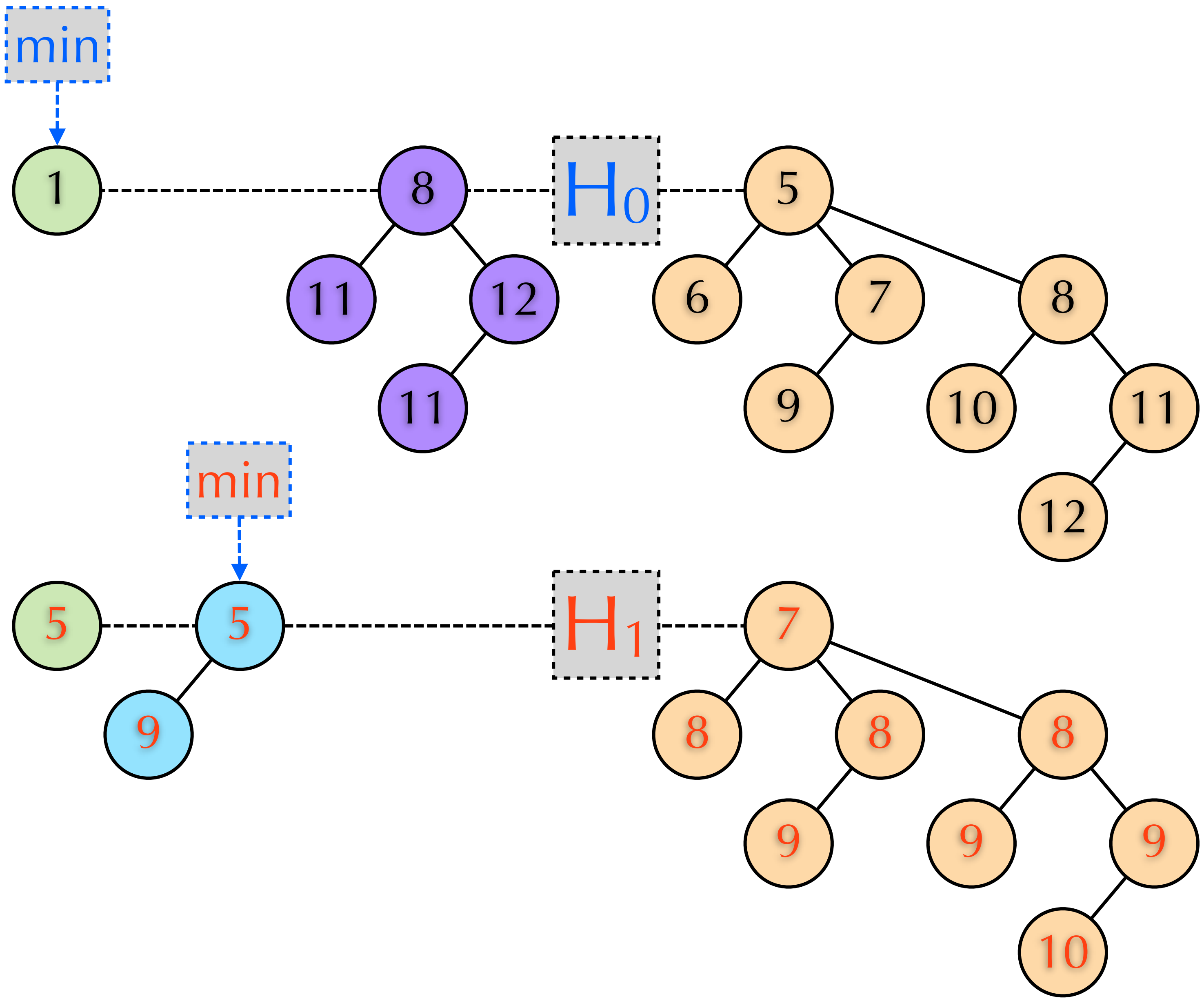
  else $T_{d+1} = \emptyset$, $T_d = T_{0,d} \cup T_{1,d}$;

  Output $\{T_0, \ldots, T_{D+1}\}$

- Minimum: easy in O(D)

T_d: carry & sum

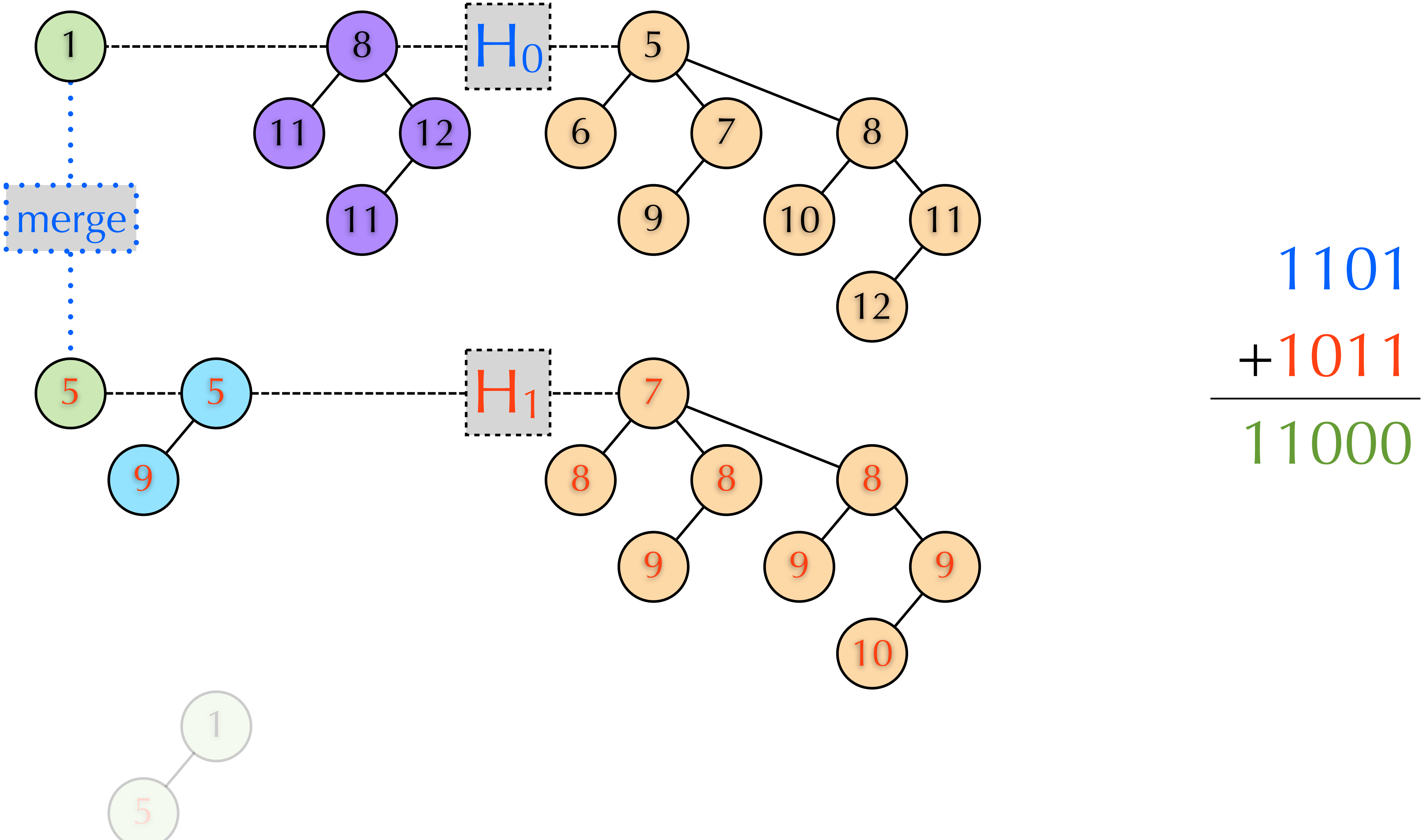
For convenience, let $T \cup \emptyset = T$.



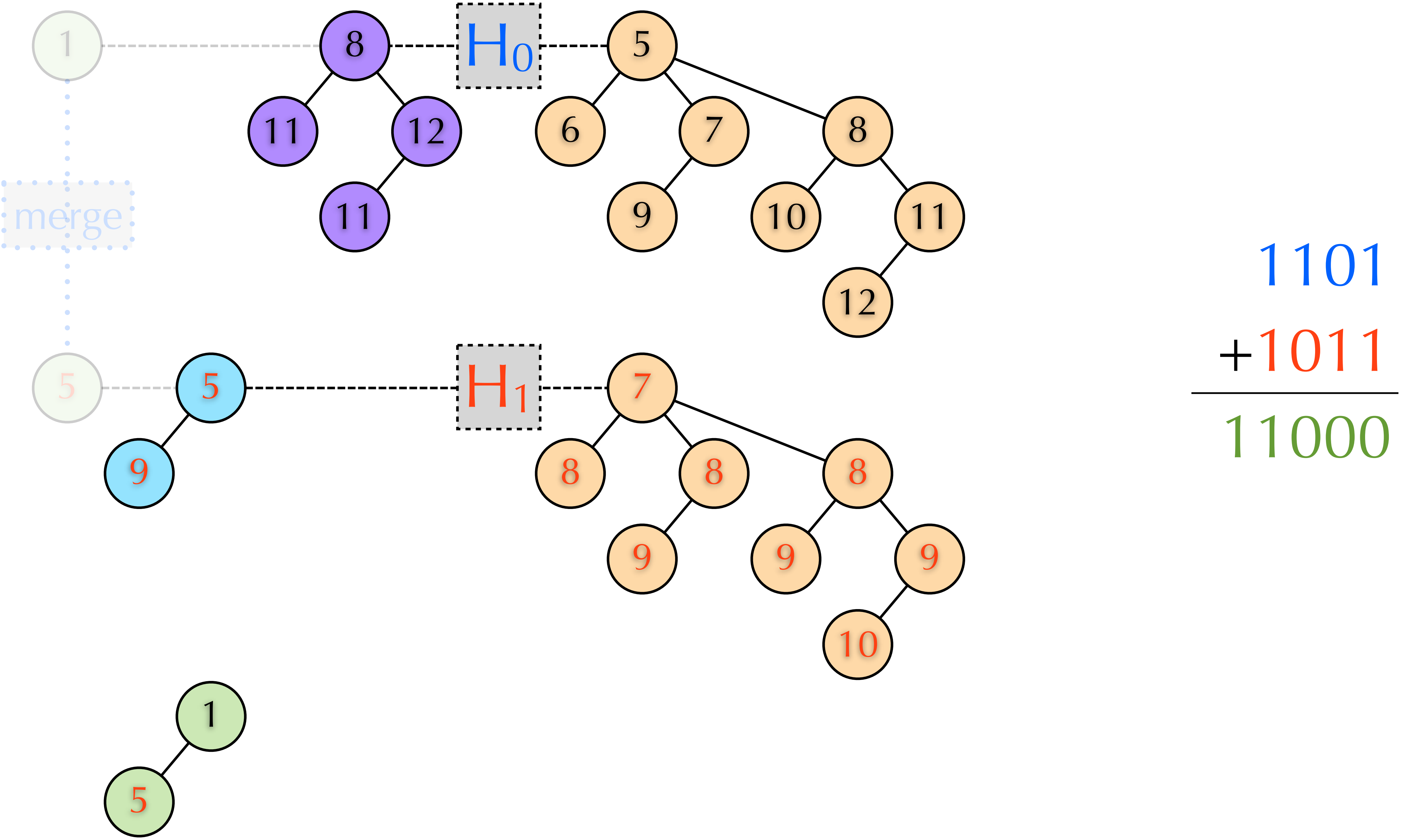
$$13+11=24$$

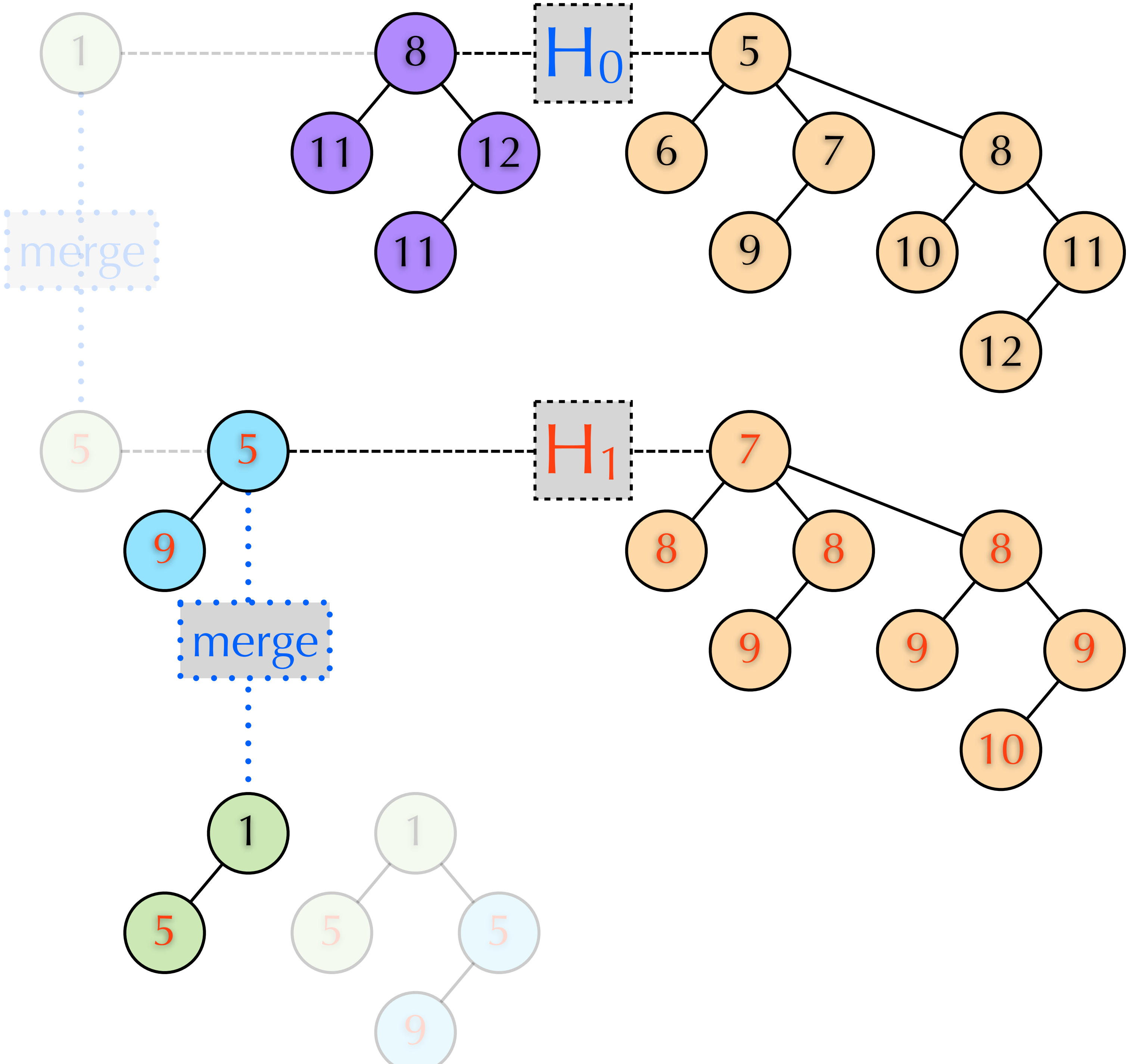
$$1101_2+1011_2=11000_2$$

$$\begin{array}{r} 1101 \\ +1011 \\ \hline 11000 \end{array}$$

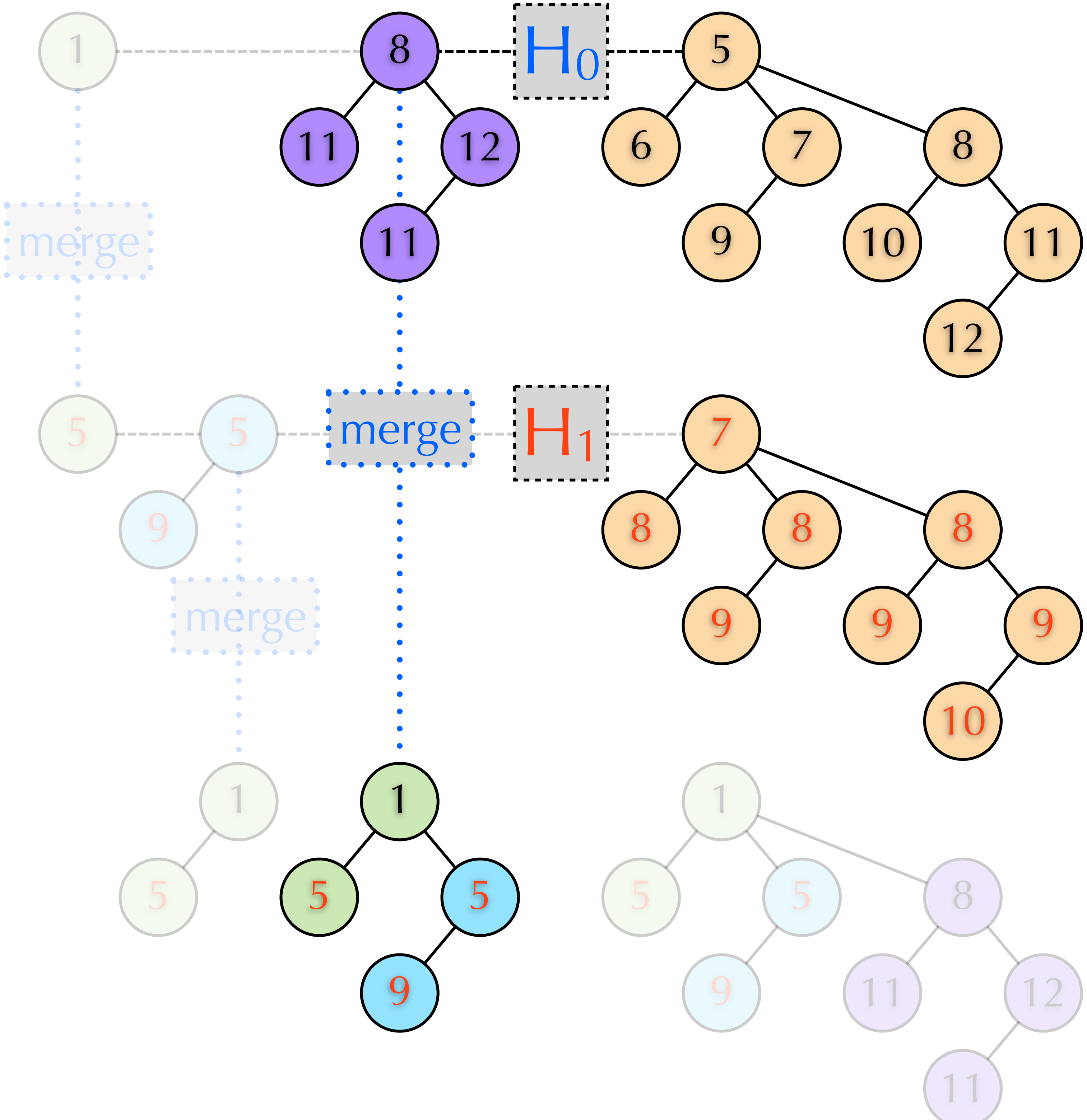


1
8
11
12
11
H0
5
6
7
8
9
10
11
12
merge
5
5
9
H1
7
8
8
8
9
9
9
10
1
5
1101
+1011
11000

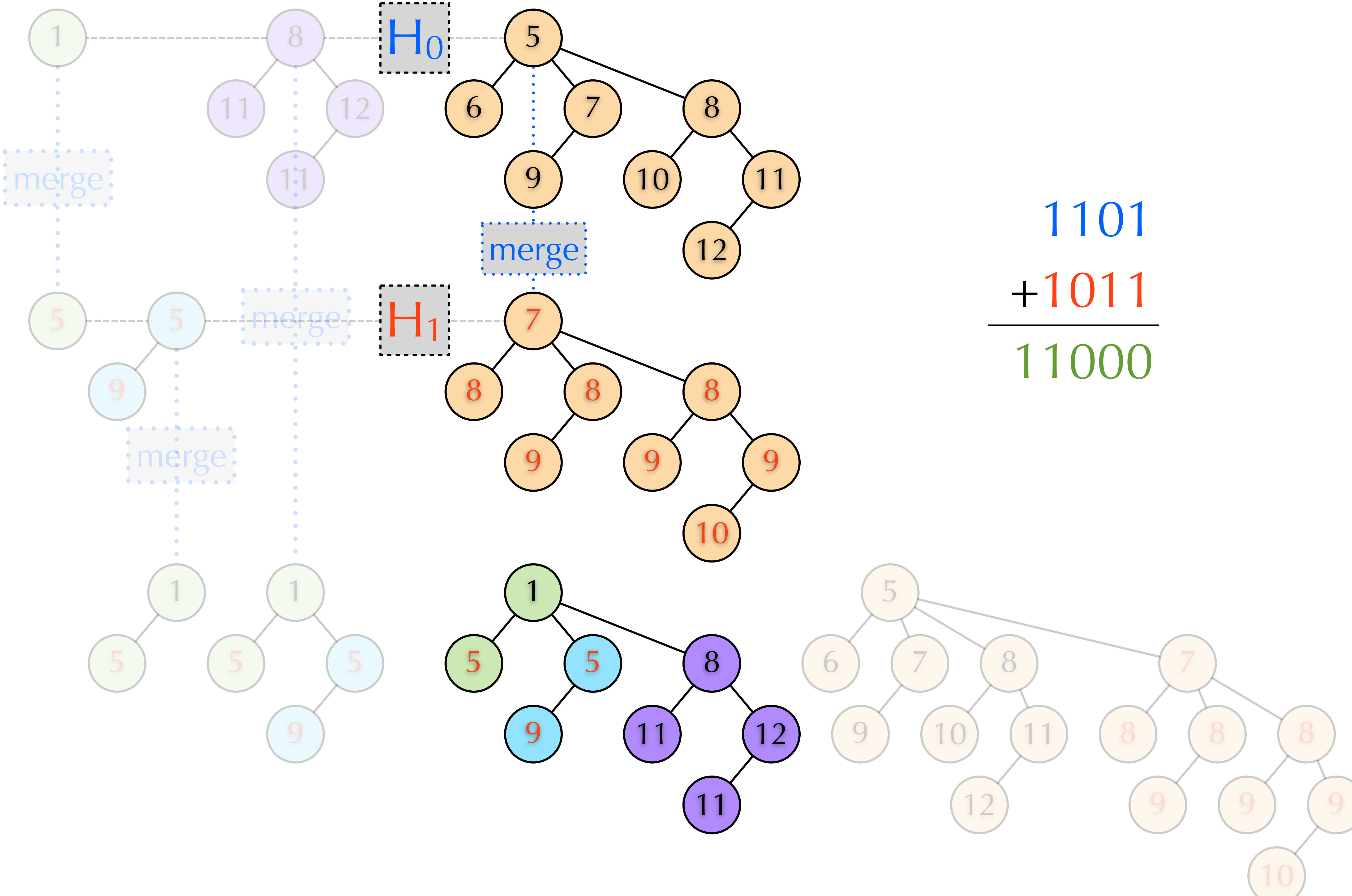

merge
H0
H1
1101
+1011
11000



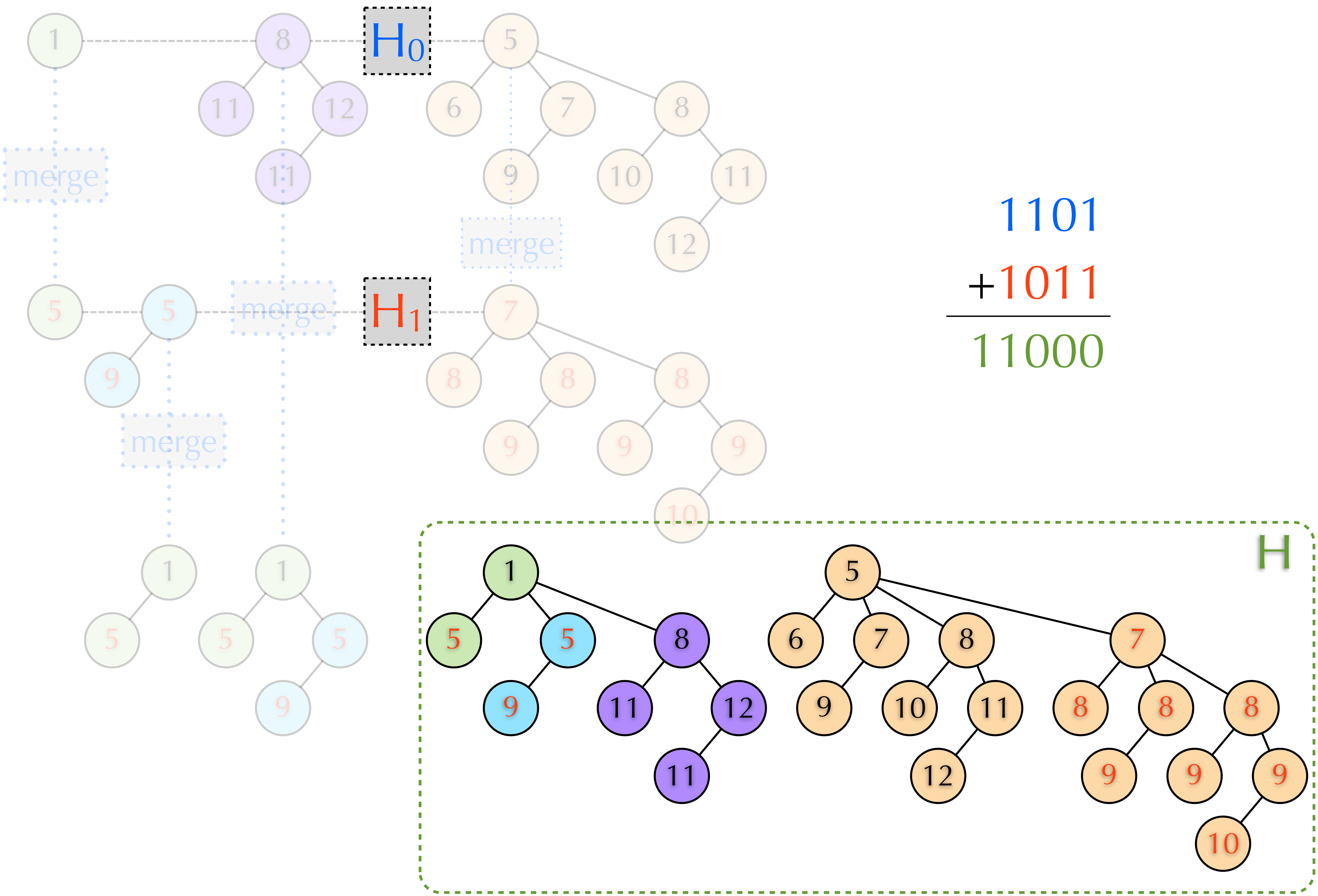
$$
\begin{array}{r}
1101 \\
+1011 \\
\hline
11000
\end{array}
$$

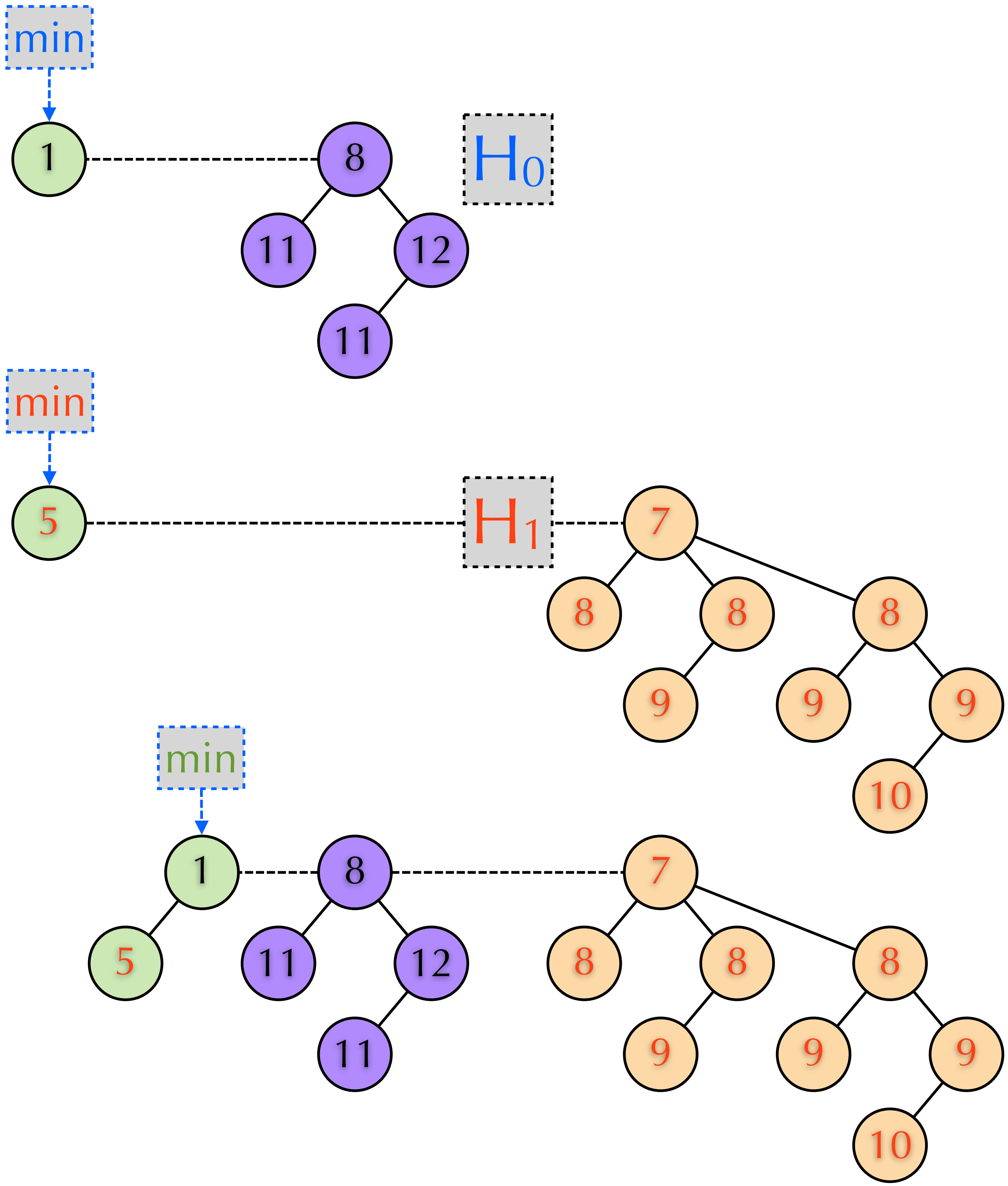


$$
\begin{array}{r}
1101 \\
+1011 \\
\hline
11000
\end{array}
$$



$$
\begin{array}{r}
1101 \\
+1011 \\
\hline
11000
\end{array}
$$


H0
H1
merge
merge
merge
merge
1101
+1011
11000
H

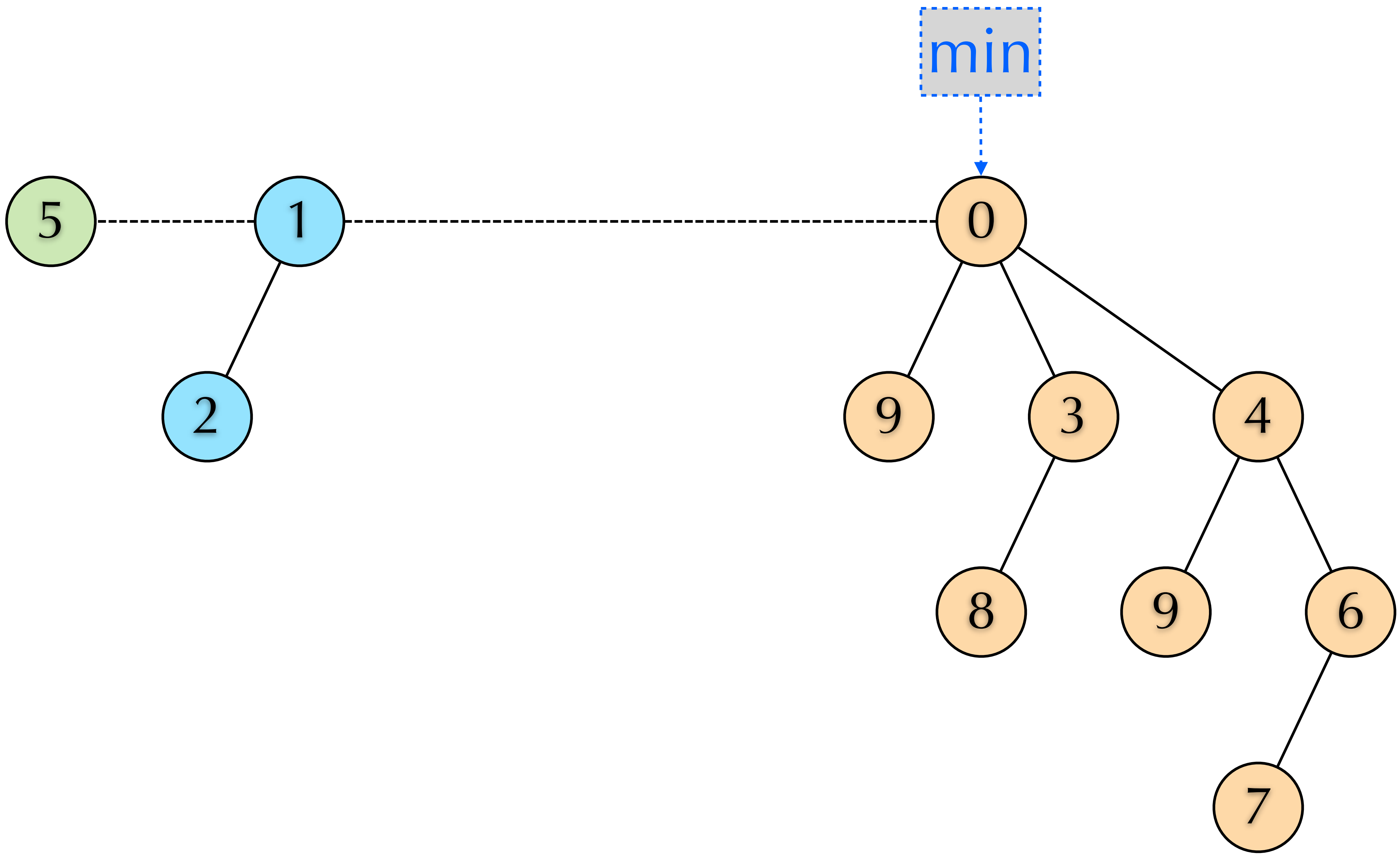


$$5+9=14$$

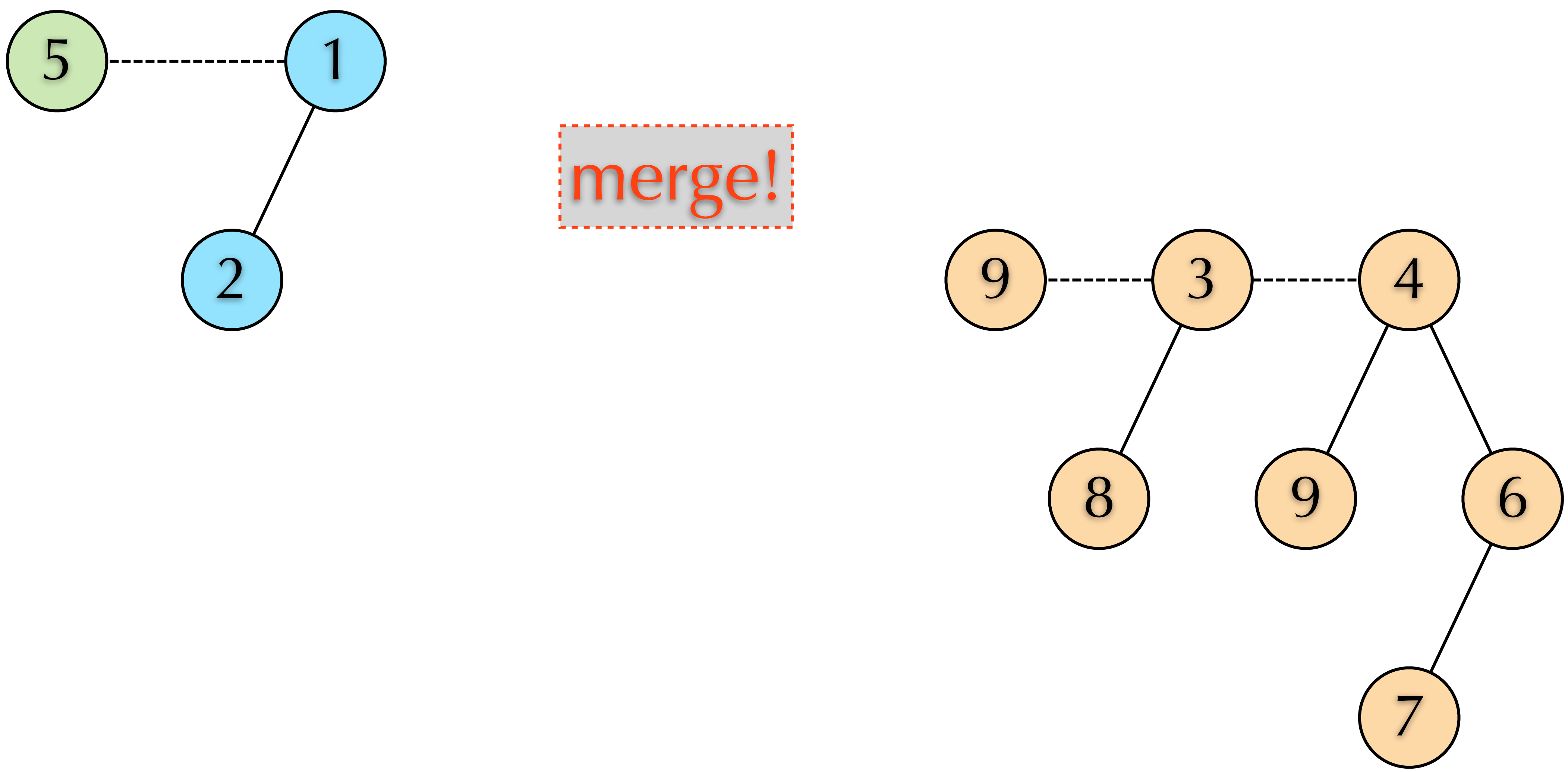
$$101_2+1001_2=1110_2$$

$$\begin{array}{r} 101 \\ +1001 \\ \hline 1110 \end{array}$$

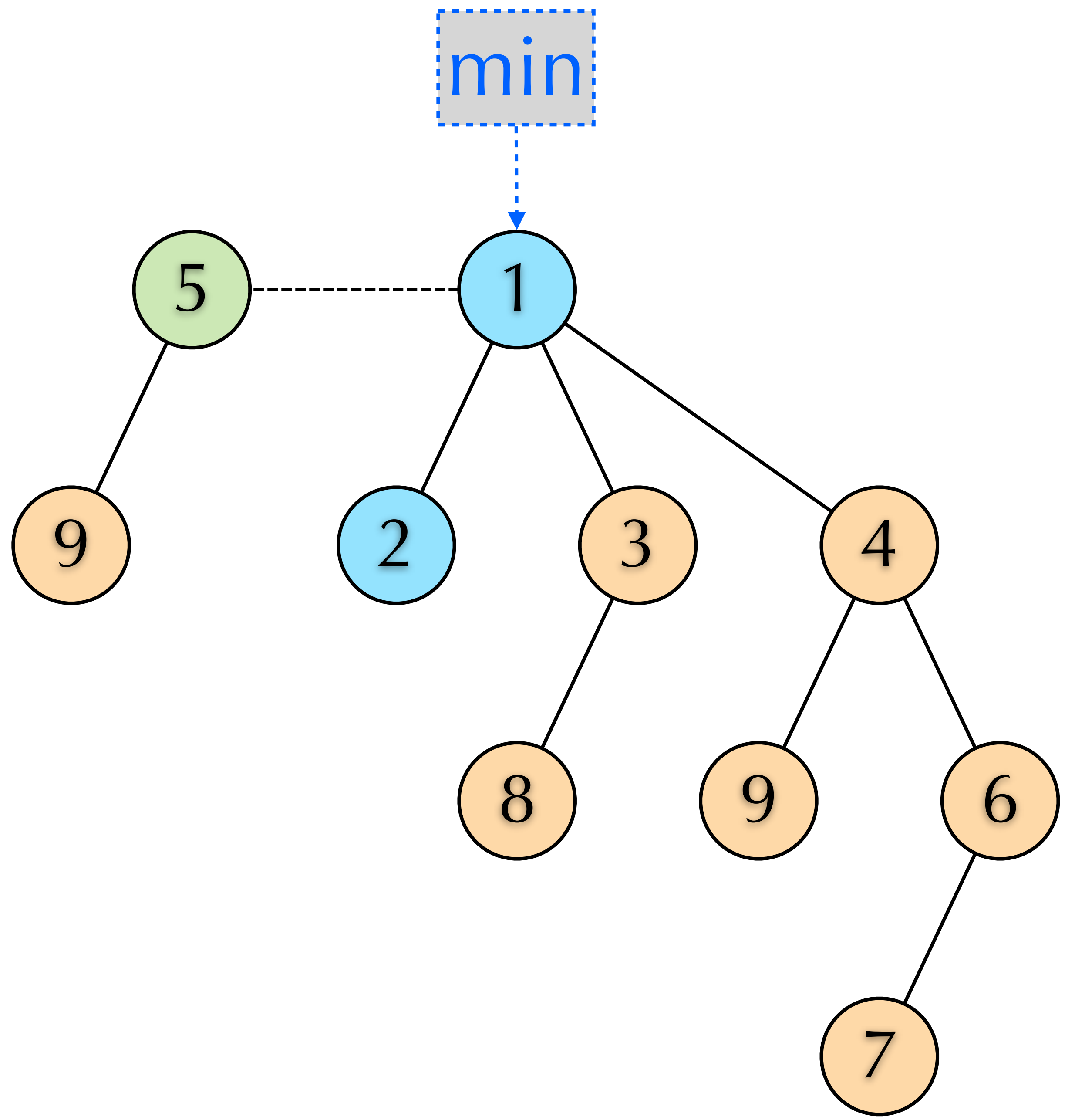
Extract Minimum



Extract Minimum



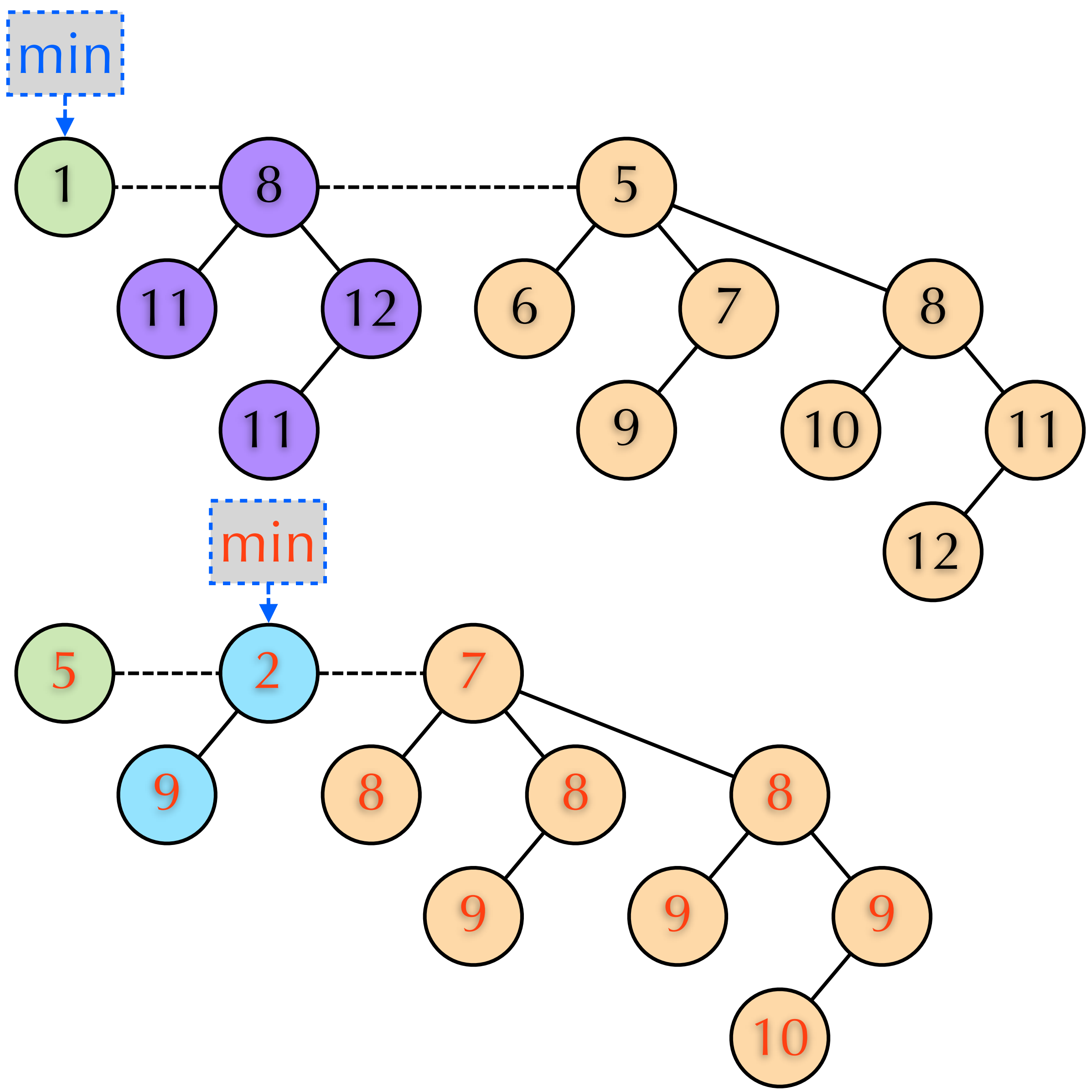
Extract Minimum



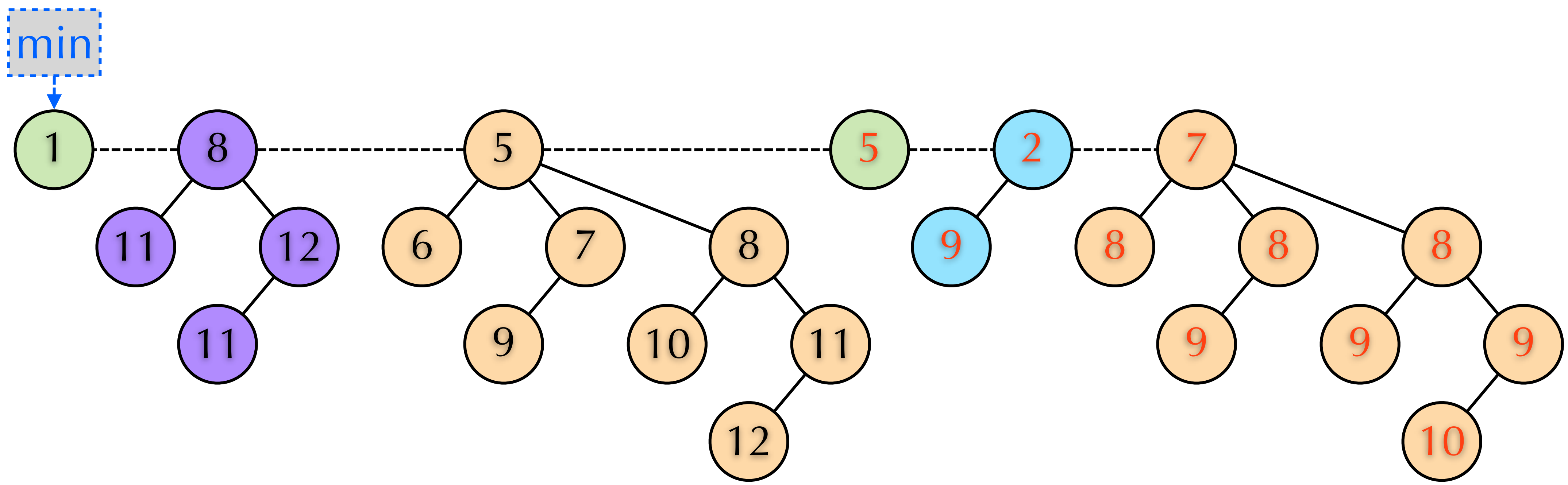
Lazy Merge & Insertion

- We don't have to perform O(logn)-time REAL merge for every merge & insertion.
- Just chain two lists of binomial trees into one. O(1)-time
- Update the minimum if necessary. O(1)
- Delay the REAL merges until extract minimum. O(logn) per operation

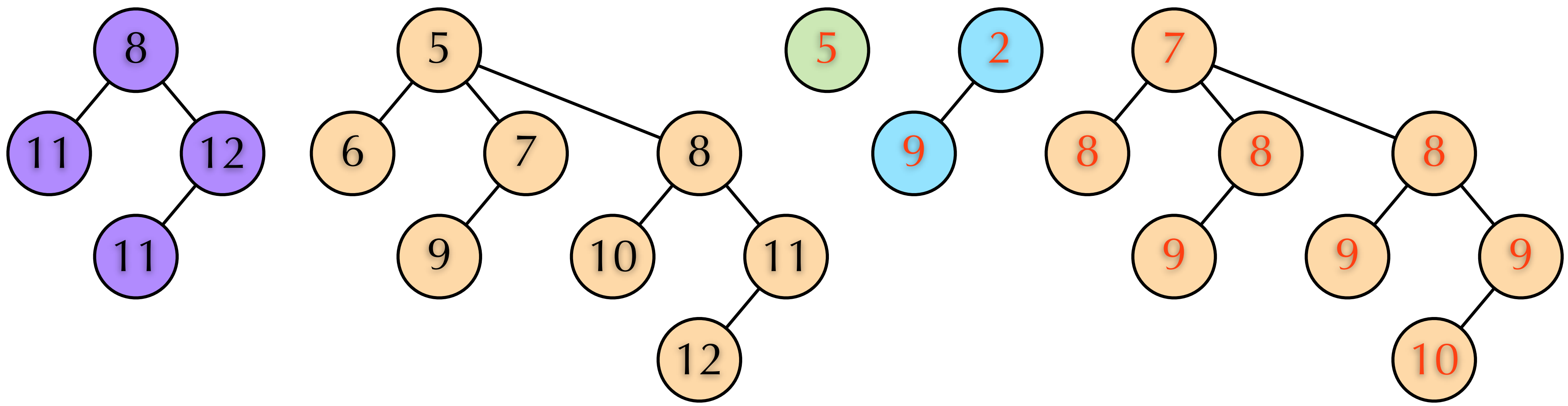
Lazy Merge



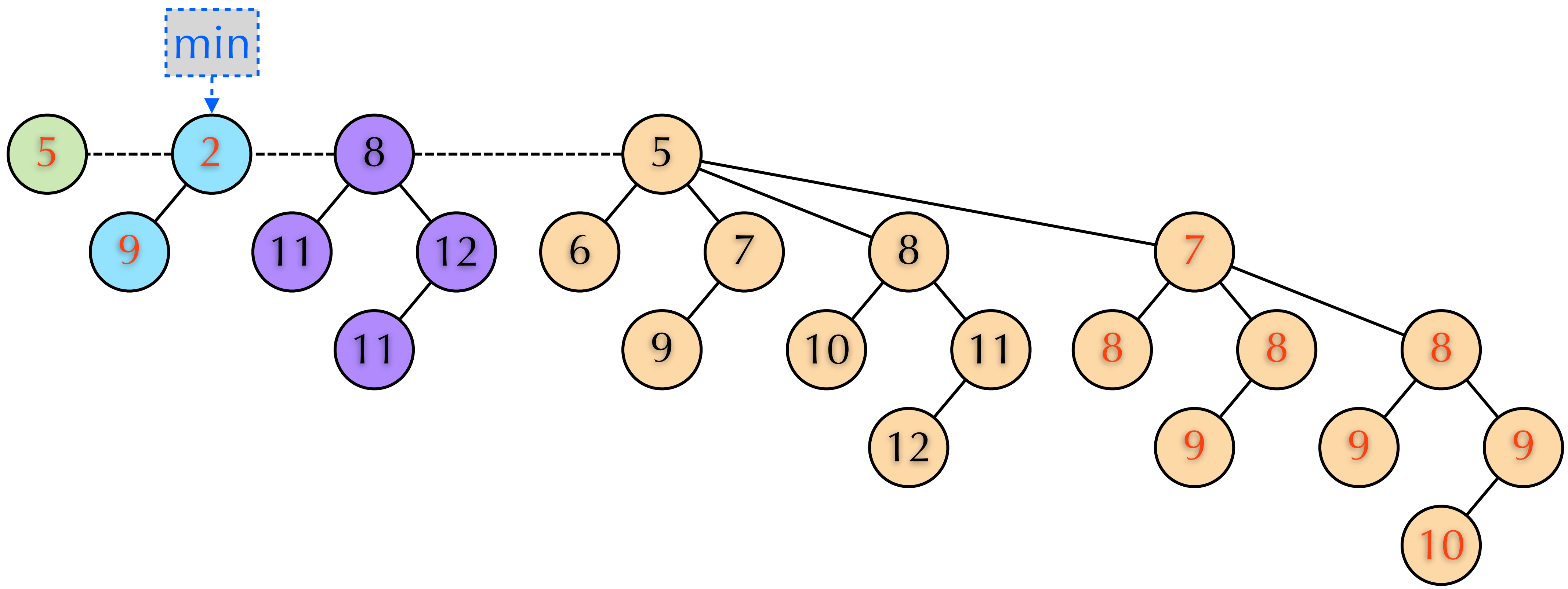
Lazy Merge



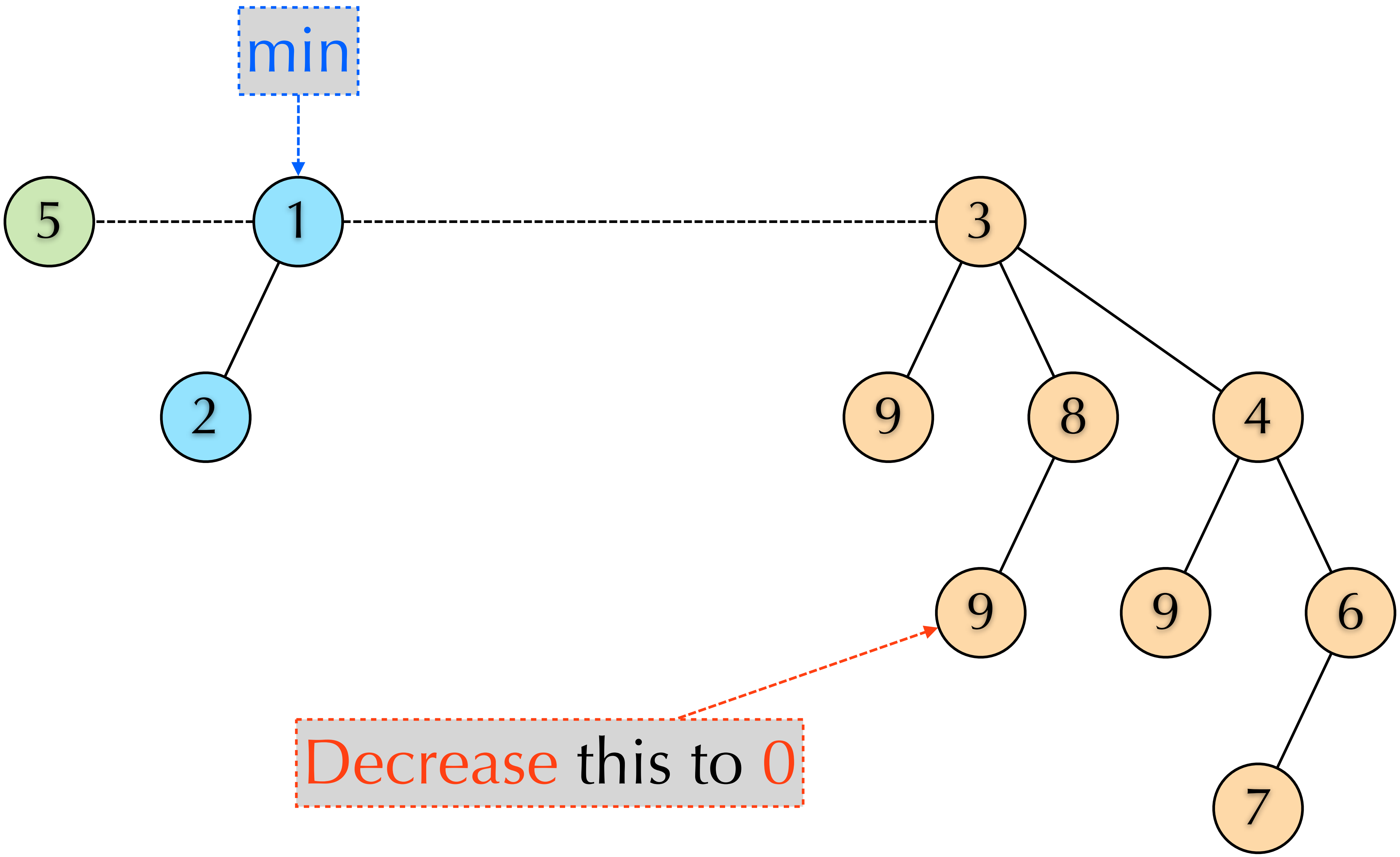
Lazy Merge: Extract Min



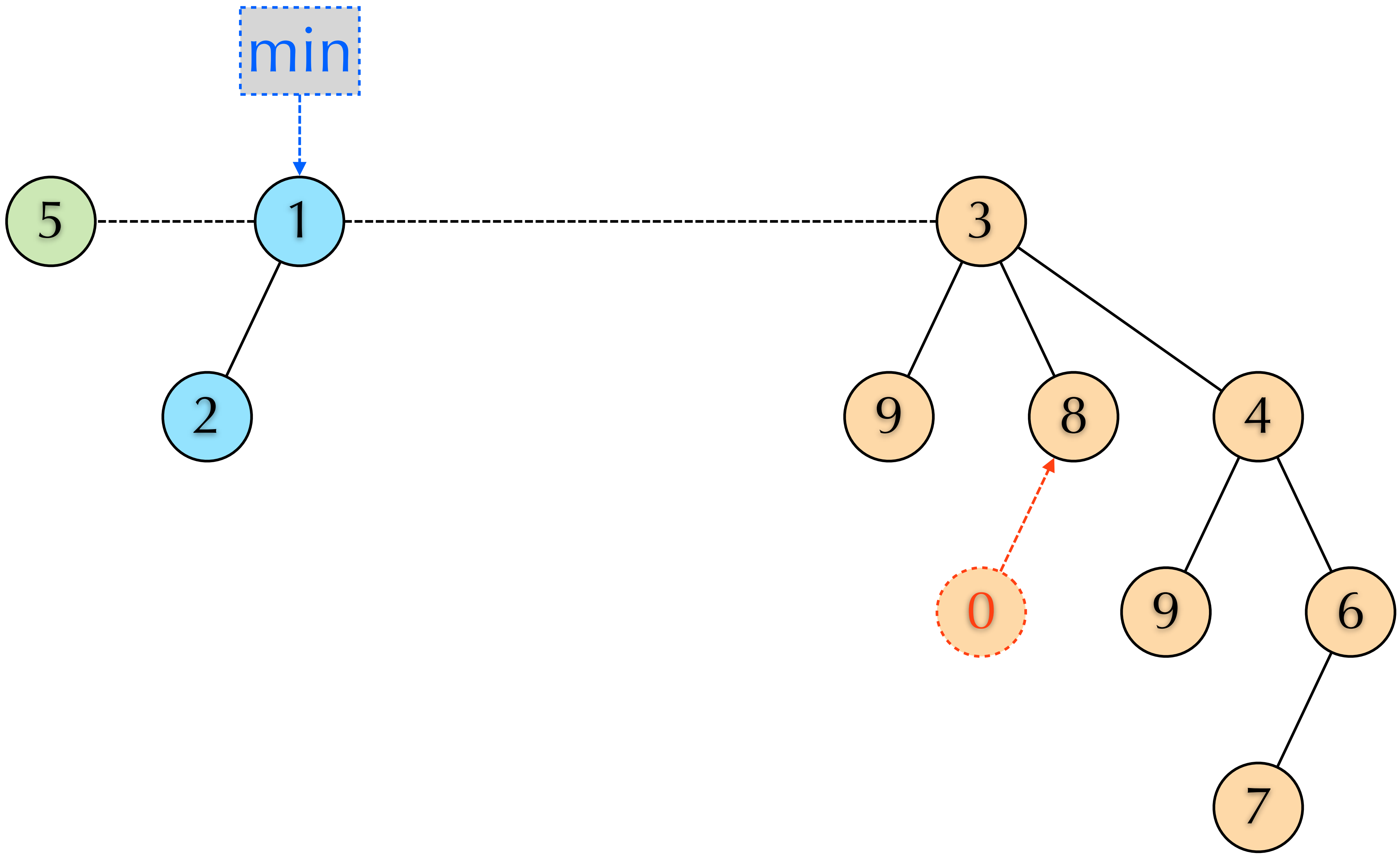
Lazy Merge: Extract Min



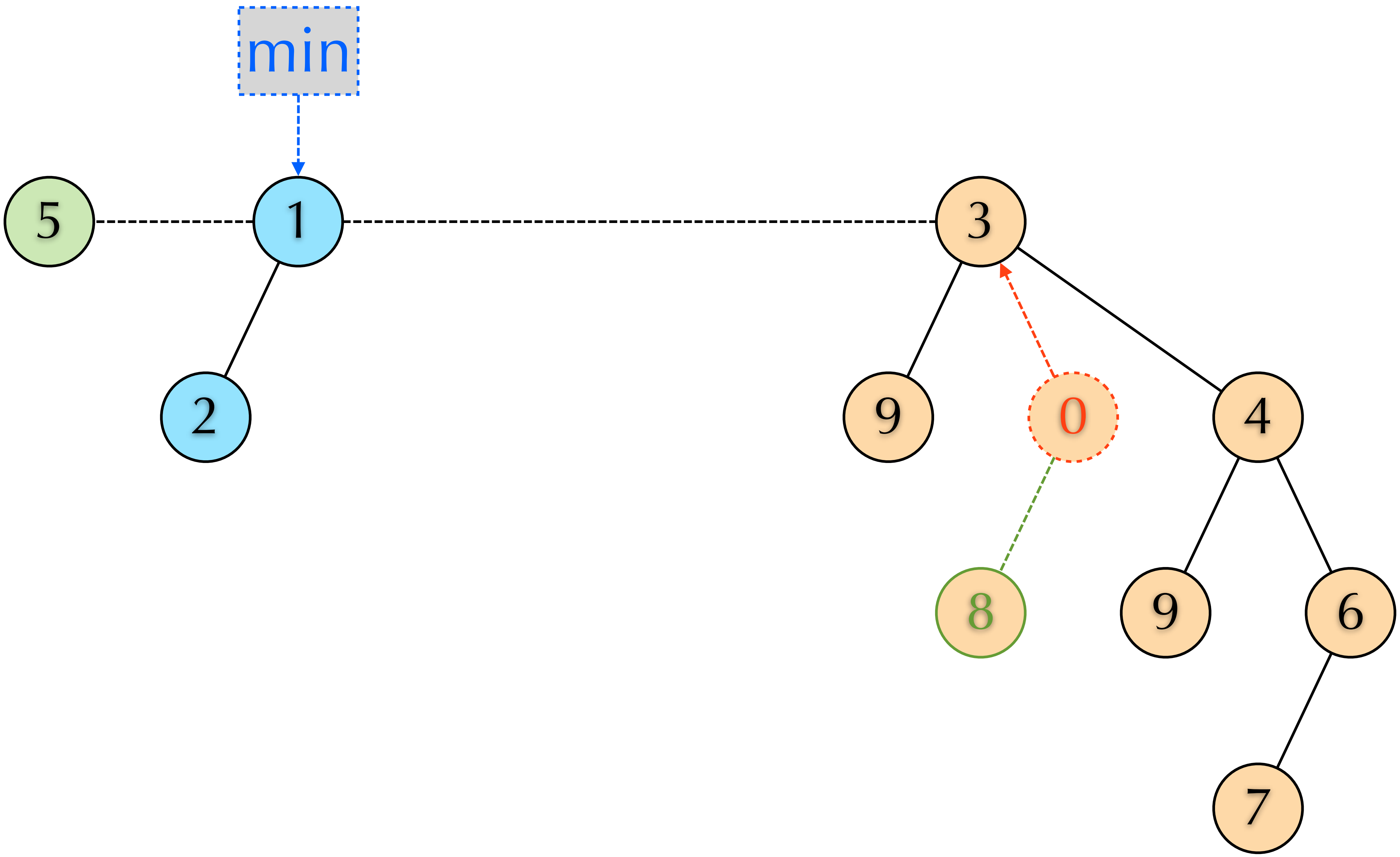
Decrease Key



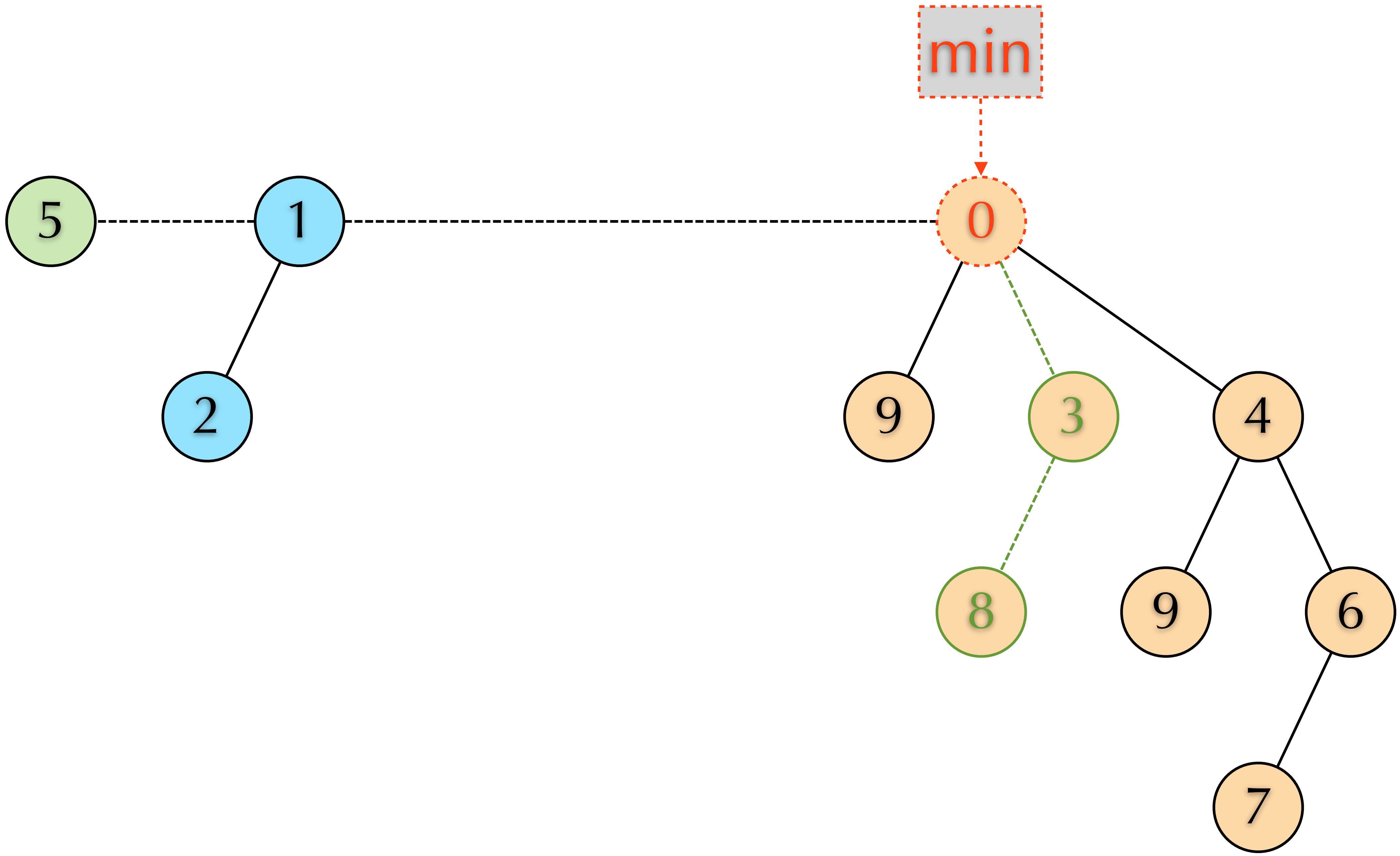
Decrease Key



Decrease Key



Decrease Key

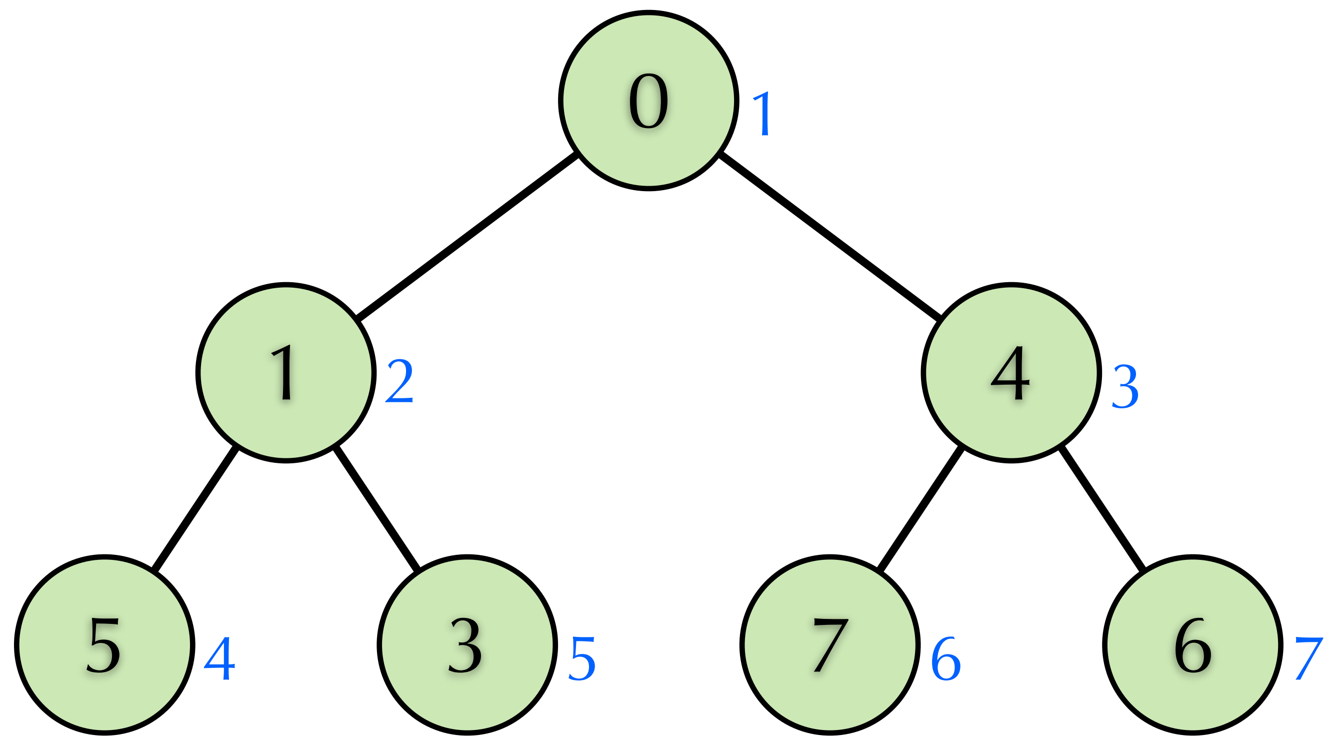


Decrease Key Needs Map

- DecreaseKey(PQ,k,obj):
  Decrease the key value of obj to k.
- We need to find out where is the node storing <k',obj> where k' is the key value before the invocation of DecreaseKey.

Decrease Key Needs Map

Index	Key	Value
1	0	紅
2	4	橙
3	1	黃
4	5	綠
5	3	藍
6	7	靛
7	6	紫

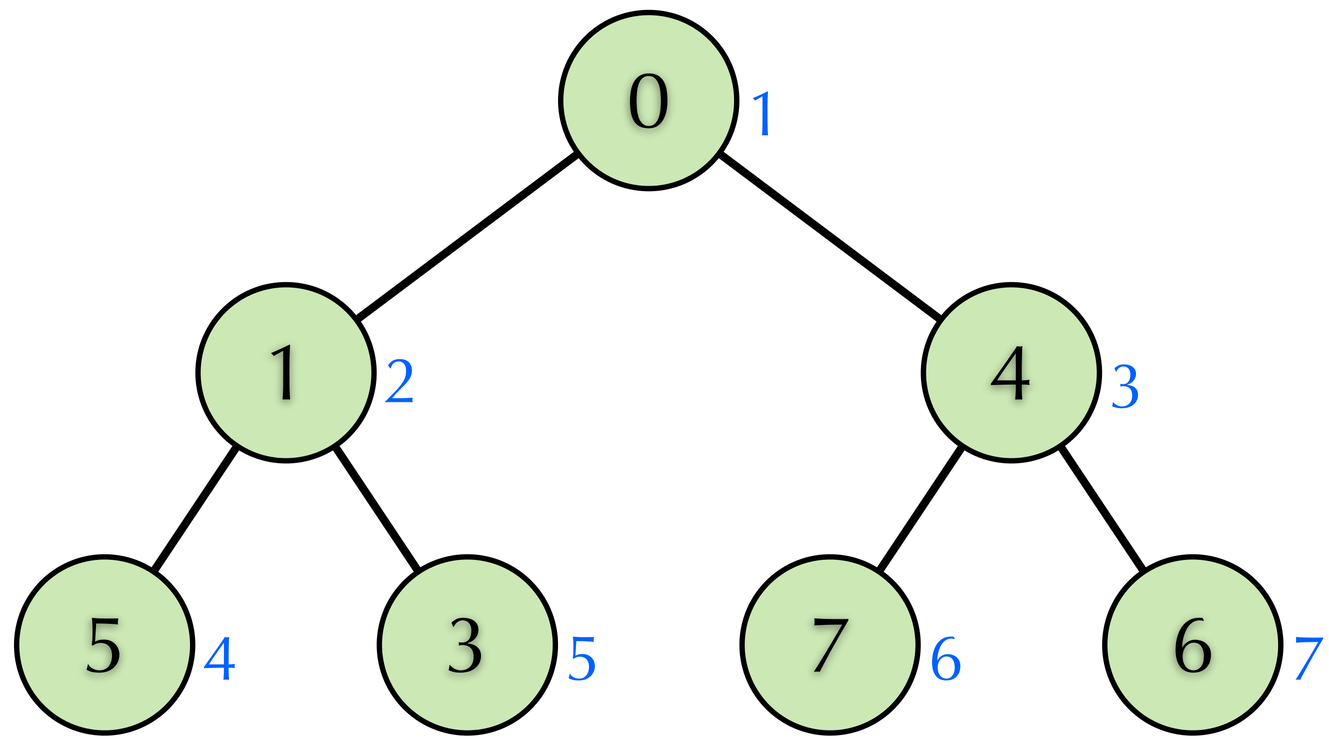


Decrease 紫's key to 2

We need to know 紫's index!

Decrease Key Needs Map

Index	Key	Value
1	0	紅
2	4	橙
3	1	黃
4	5	綠
5	3	藍
6	7	靛
7	6	紫

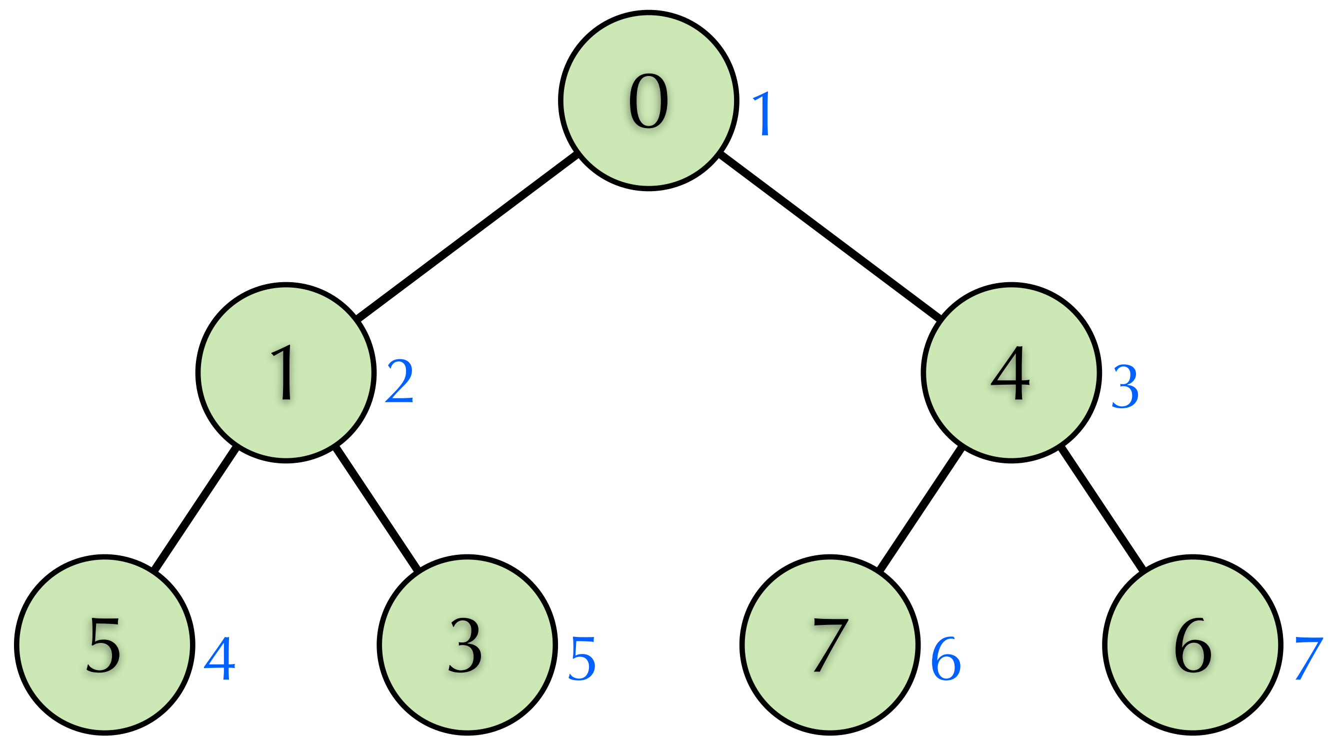


Sequential search: O(n)-time

Use Value as the key of map!

Decrease Key Needs Map

Map Key	Map Value	
Value	Index	Key
紅	1	0
橙	2	4
黃	3	1
綠	4	5
藍	5	3
靛	6	7
紫	7	6



Search by BST Map: O(logn)-time

Search by hash map: expected O(1)-time

Array Based vs Link Based

- Array based implementations need faster map since they might need to modify O(logn) key-value pairs/entries.
  - Example: Binary heap
- Some link based implementations do not need to any key-value pair/entry.
  - Change the links only. The key-value pair stays at the same place.
  - Example: Binomial heap

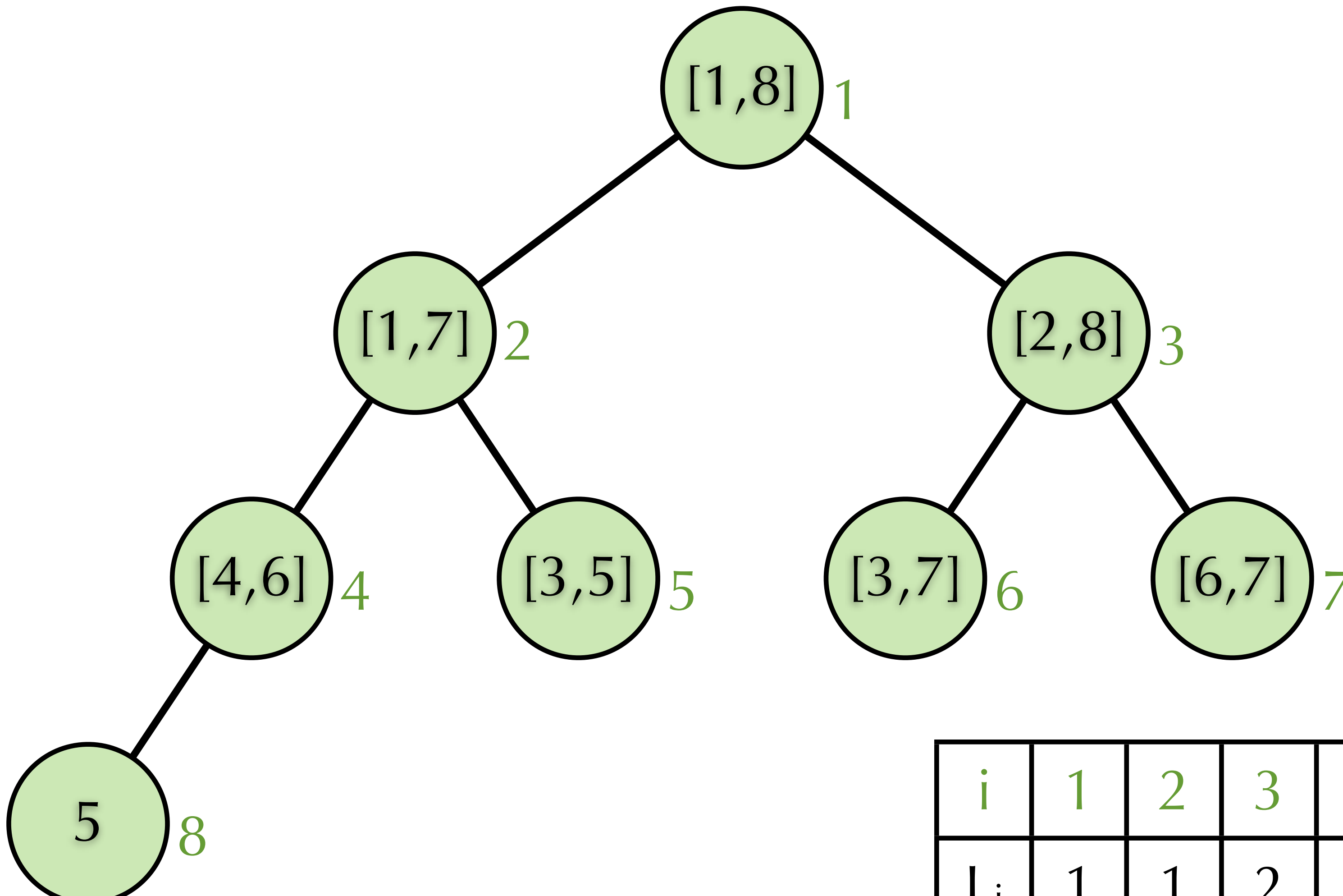
Homework 11.2

- a) How to initialize an n-element binomial heap?
- b) Will the lazy merge policy increase the performance? Explain your answer.
- c) What is a Fibonacci heap? Compare Fibonacci heaps and binomial heaps.

Interval Heap

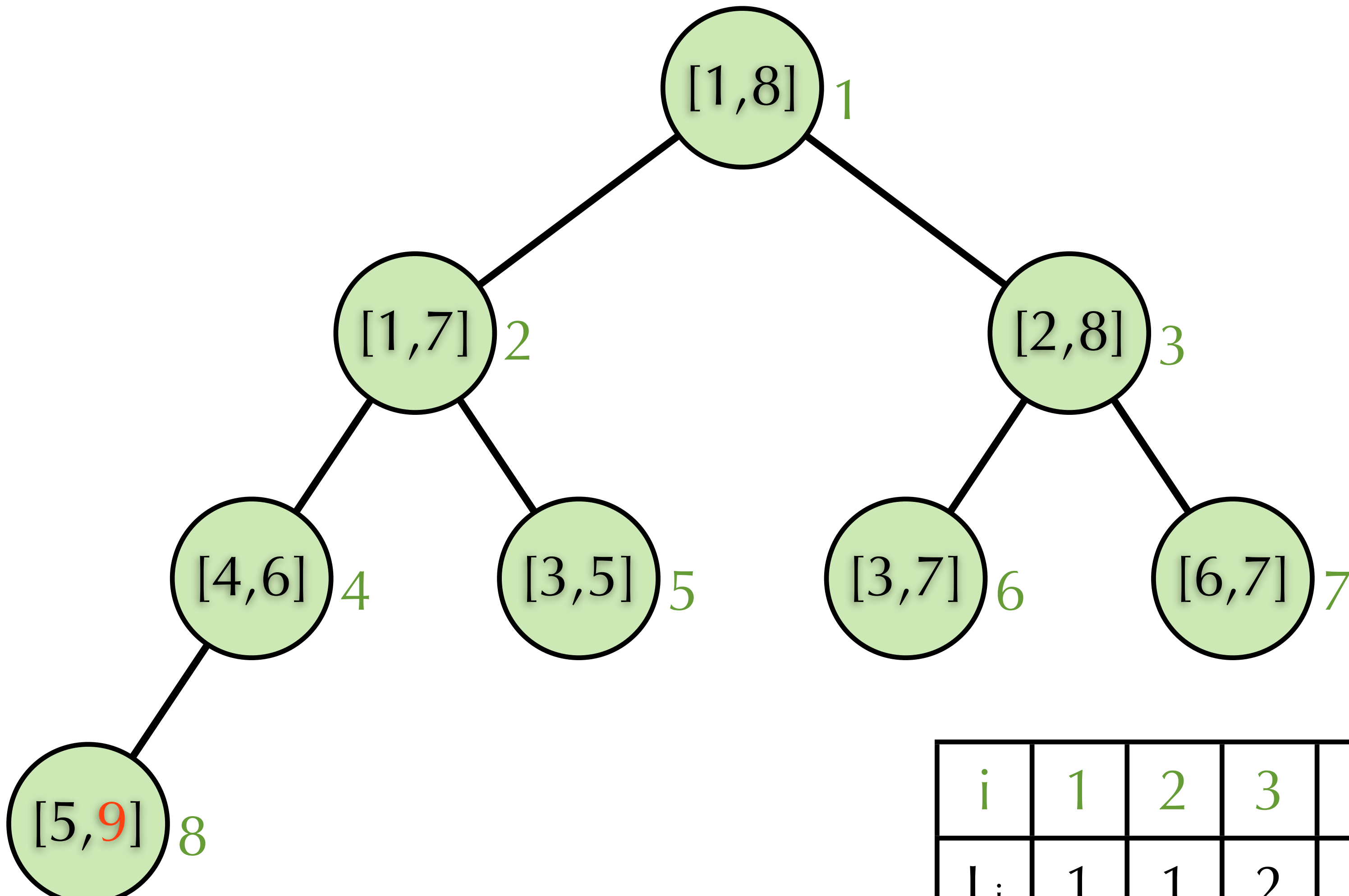
- [L,R] denotes the interval $\{x: L \le x \le R\}$.
- $[L,R] \subseteq [L',R']$ if and only if $L' \le L \le R \le R'$.
- $\subseteq$ is a partial ordered relation.
  - If $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$.
- Interval heap: a variant of binary heap
  - Each node stores an interval [L,R]. Only EXCEPTION: the last node can store only one value x.
  - Interval heap property: If A is B'parent, then interval $[L_A,R_A] \subseteq [L_B,R_B]$.

Example: Interval Heap



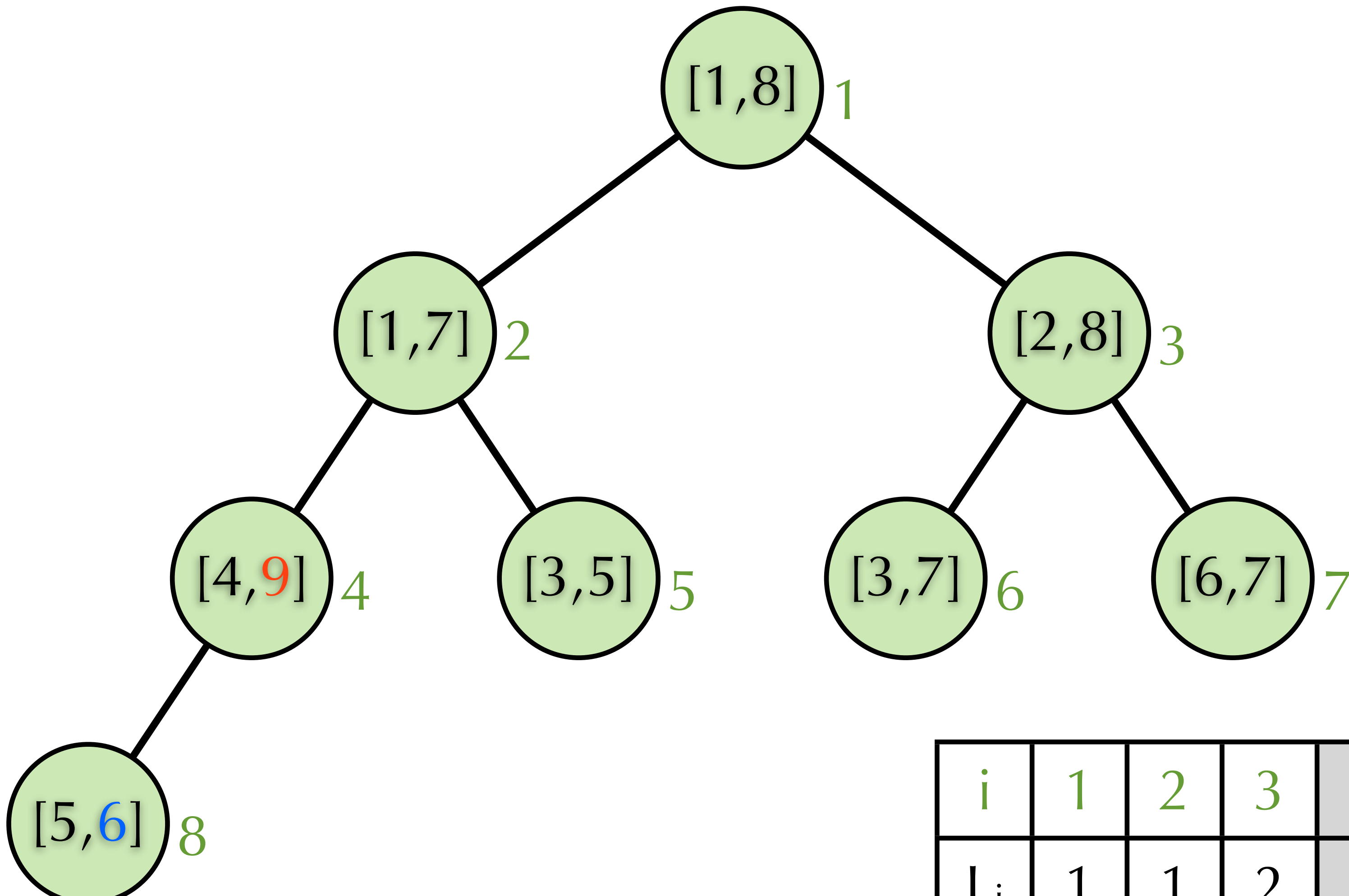
i	1	2	3	4	5	6	7	8	9
L_i	1	1	2	4	3	3	6	5	
R_i	8	7	8	6	5	7	7	5	
k_i	2	2	2	2	2	2	2	1	

Insert 9



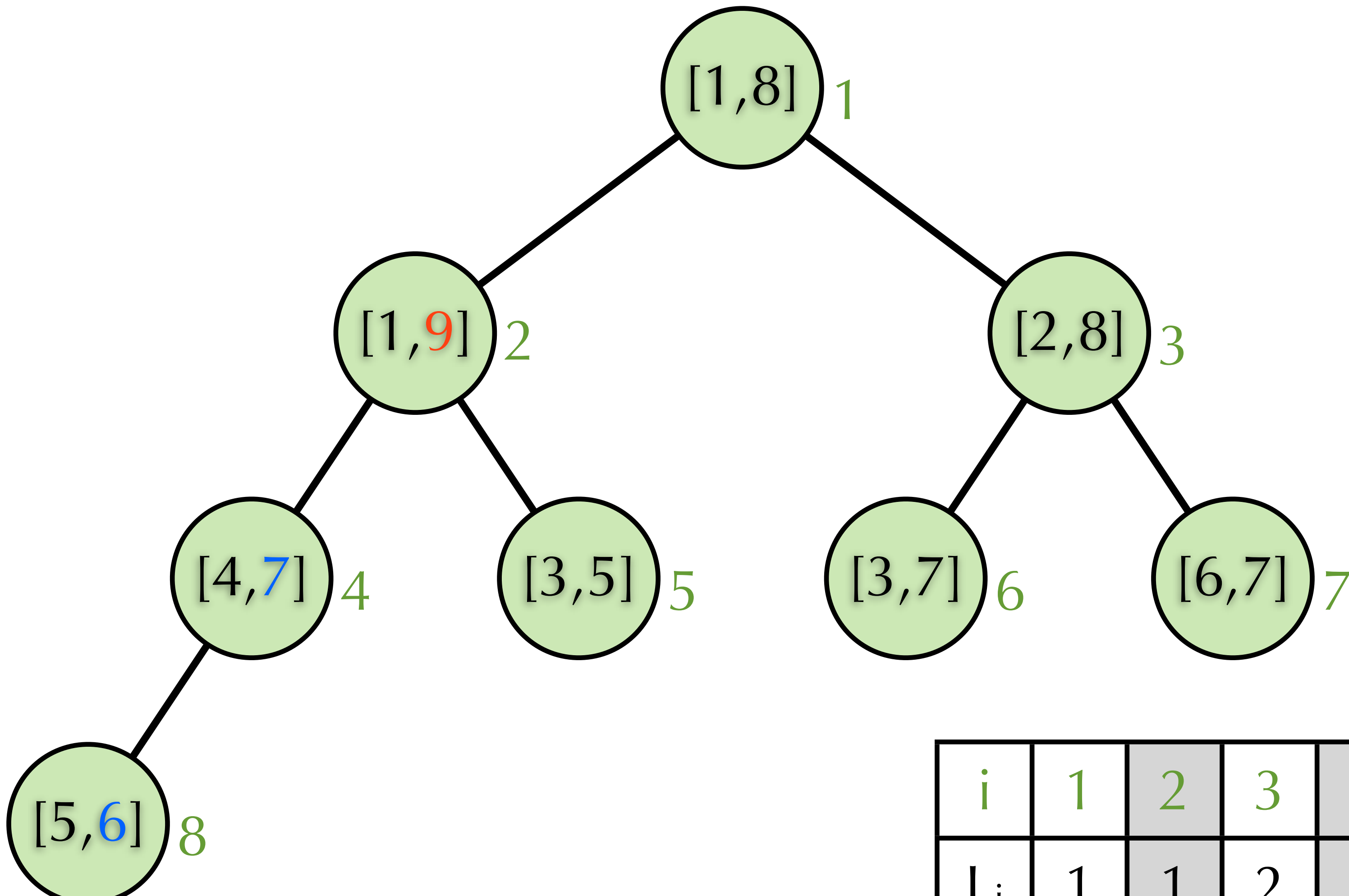
i	1	2	3	4	5	6	7	8	9
L_i	1	1	2	4	3	3	6	5	
R_i	8	7	8	6	5	7	7	9	
k_i	2	2	2	2	2	2	2	2	

Insert 9



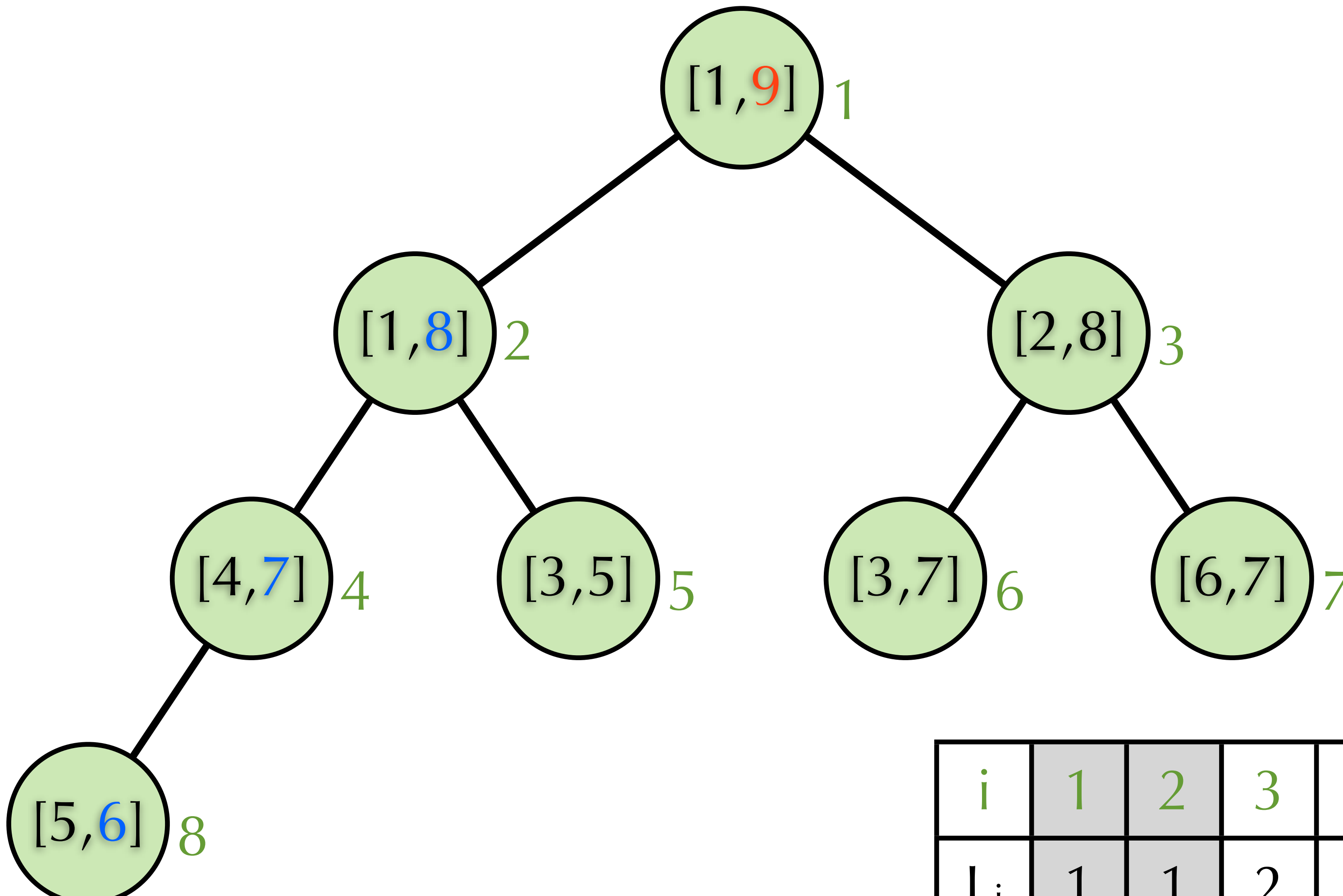
i	1	2	3	4	5	6	7	8	9
L_i	1	1	2	4	3	3	6	5	
R_i	8	7	8	9	5	7	7	6	
k_i	2	2	2	2	2	2	2	2	

Insert 9



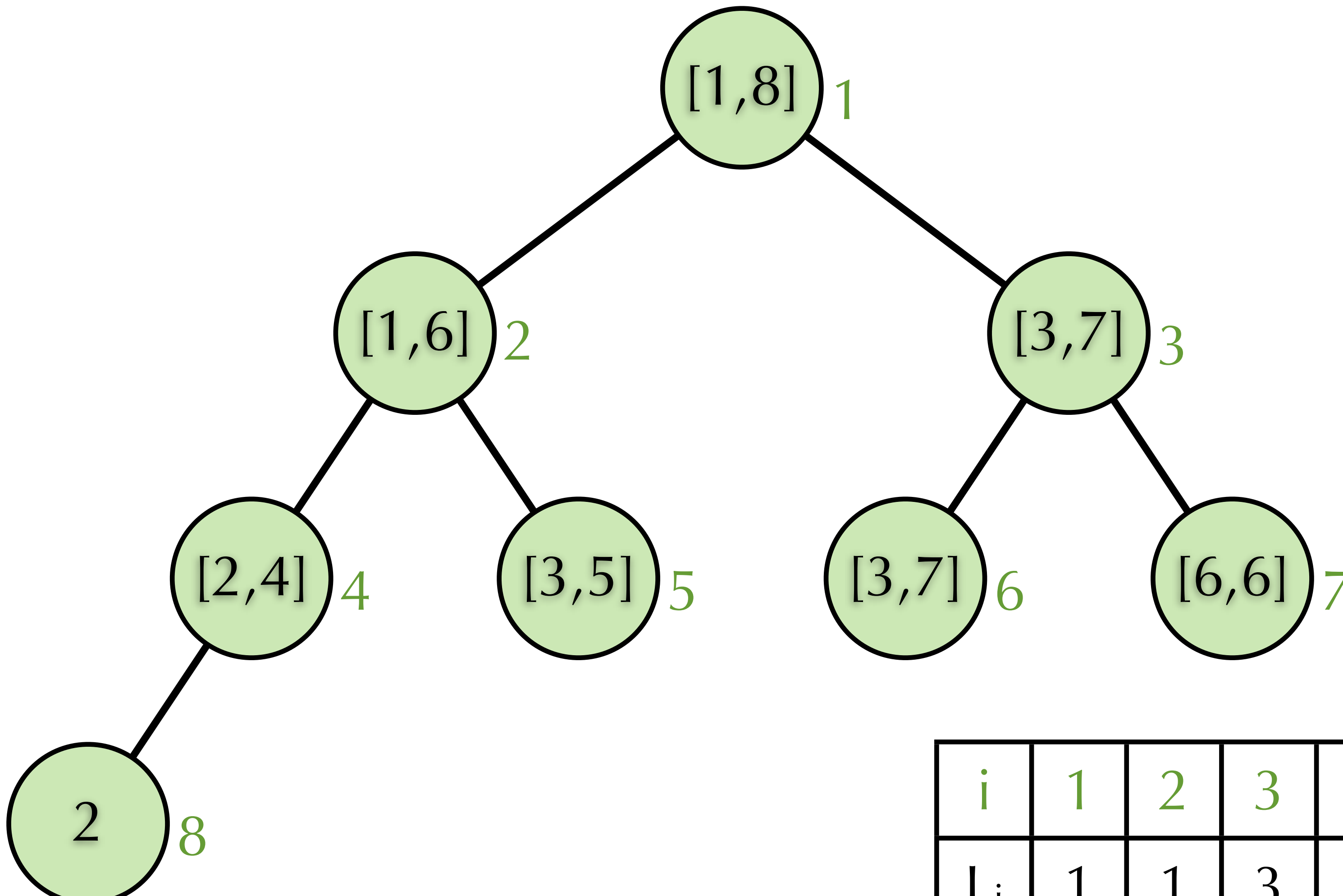
i	1	2	3	4	5	6	7	8	9
L_i	1	1	2	4	3	3	6	5	
R_i	8	9	8	7	5	7	7	6	
k_i	2	2	2	2	2	2	2	2	

Insert 9



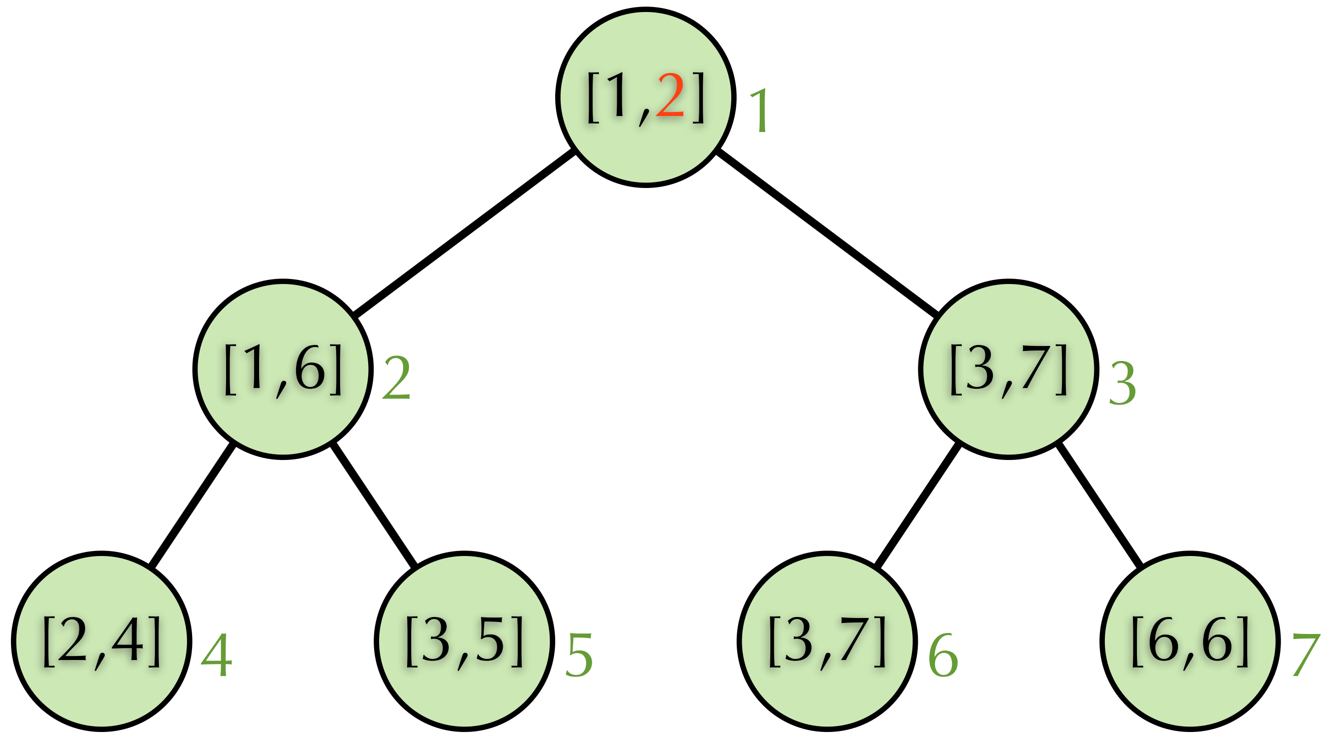
i	1	2	3	4	5	6	7	8	9
L_i	1	1	2	4	3	3	6	5	
R_i	9	8	8	7	5	7	7	6	
k_i	2	2	2	2	2	2	2	2	

Extract Max



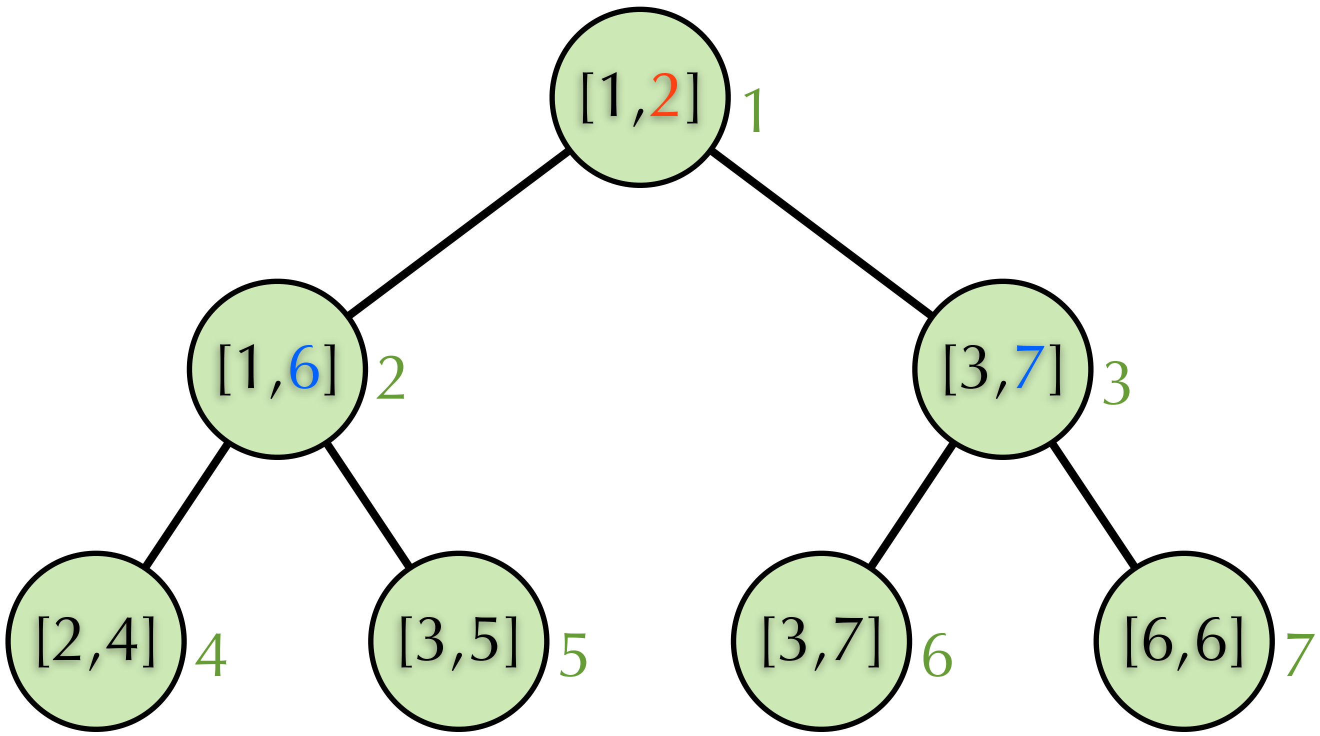
i	1	2	3	4	5	6	7	8	9
L_i	1	1	3	2	3	3	6	2	
R_i	8	6	7	4	5	7	6	2	
k_i	2	2	2	2	2	2	2	1	

Extract Max



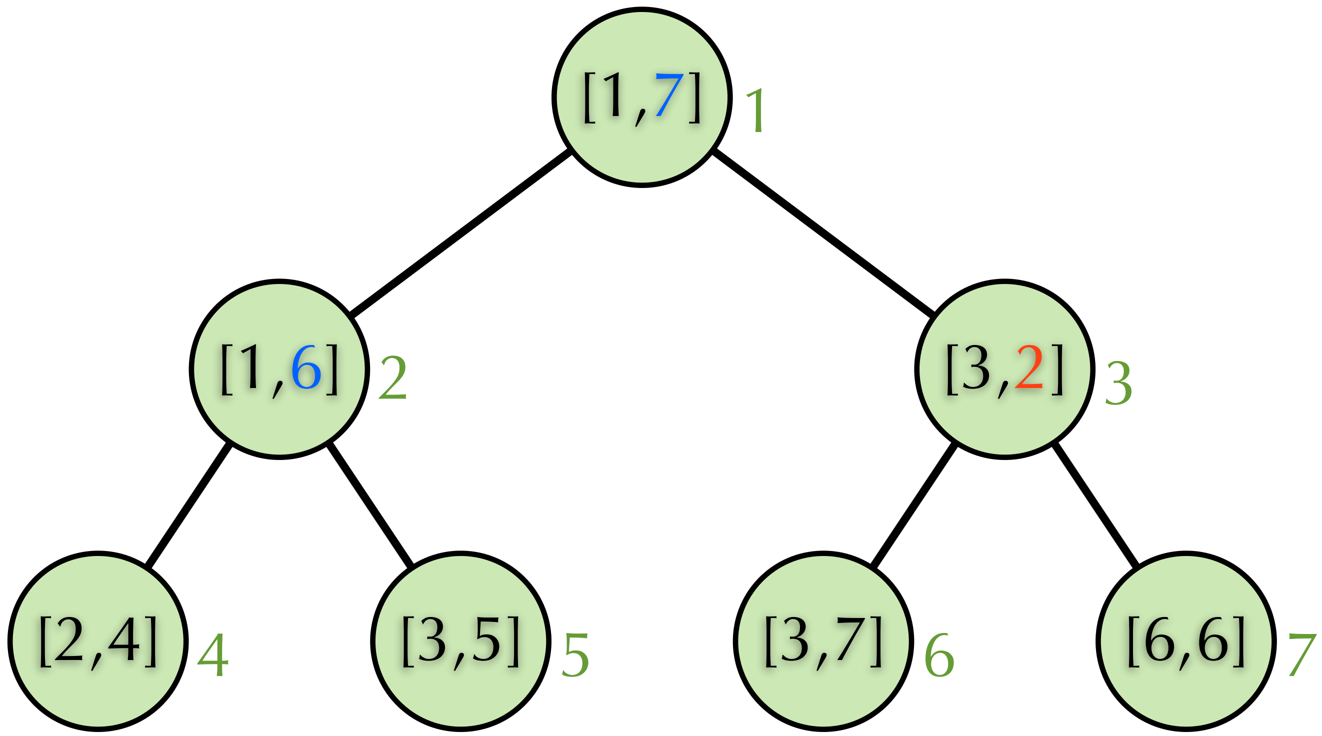
i	1	2	3	4	5	6	7	8	9
L_i	1	1	3	2	3	3	6	2	
R_i	2	6	7	4	5	7	6	2	
k_i	2	2	2	2	2	2	2	0	

Extract Max



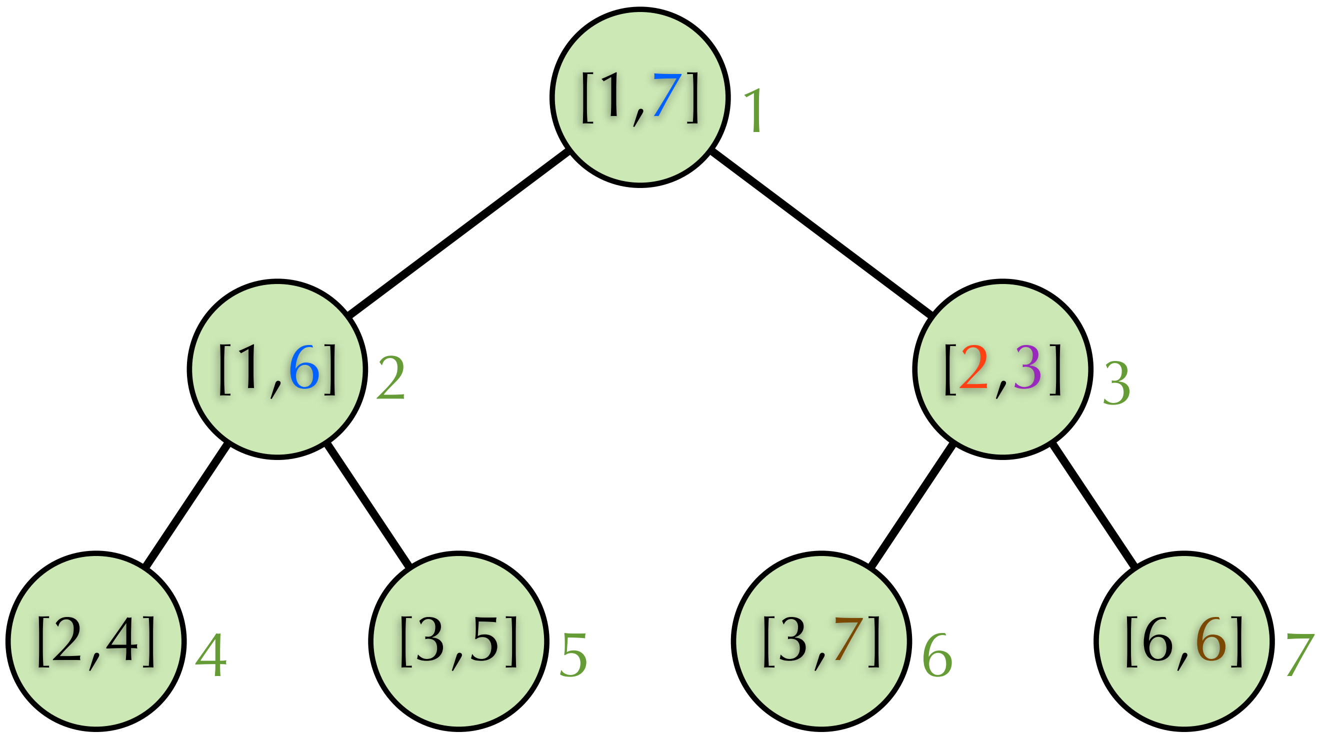
i	1	2	3	4	5	6	7	8	9
L_i	1	1	3	2	3	3	6	2	
R_i	2	6	7	4	5	7	6	2	
k_i	2	2	2	2	2	2	2	0	

Extract Max



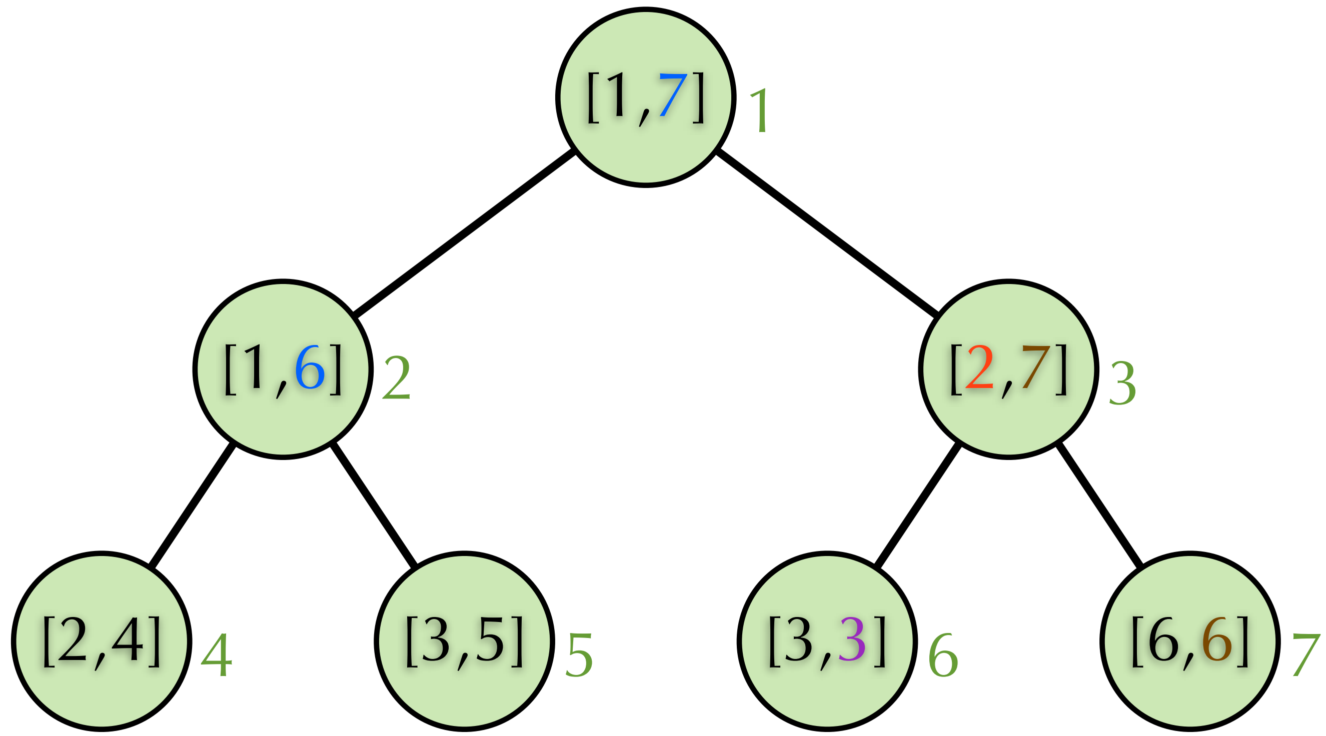
i	1	2	3	4	5	6	7	8	9
L_i	1	1	3	2	3	3	6	2	
R_i	7	6	2	4	5	7	6	2	
k_i	2	2	2	2	2	2	2	0	

Extract Max



i	1	2	3	4	5	6	7	8	9
L_i	1	1	2	2	3	3	6	2	
R_i	7	6	3	4	5	7	6	2	
k_i	2	2	2	2	2	2	2	0	

Extract Max



i	1	2	3	4	5	6	7	8	9
L_i	1	1	2	2	3	3	6	2	
R_i	7	6	7	4	5	3	6	2	
k_i	2	2	2	2	2	2	2	0	

Homework 11.3

- a) How should we perform change key operation on an interval heap?
- b) Can we apply the idea of interval heap property to priority queue supporting meld/merge operation such as leftist trees or binomial heaps? If yes, what modification must to be done? If no, explain the reason.