UCSC Silicon Valley Extension

Advanced C Programming

Binomial Heaps

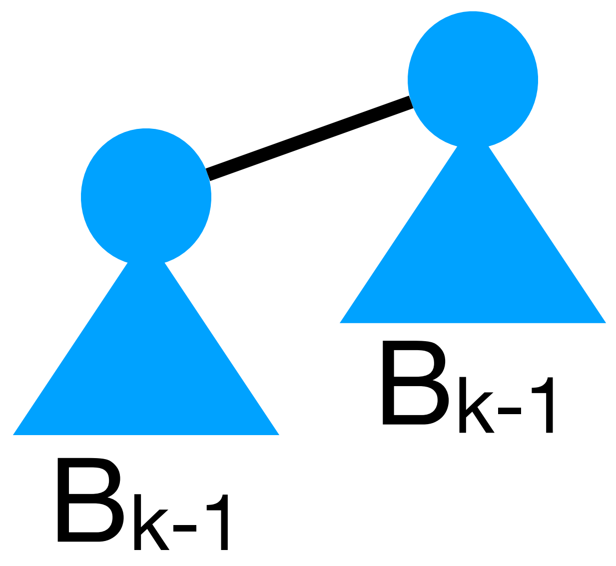
Radhika Grover

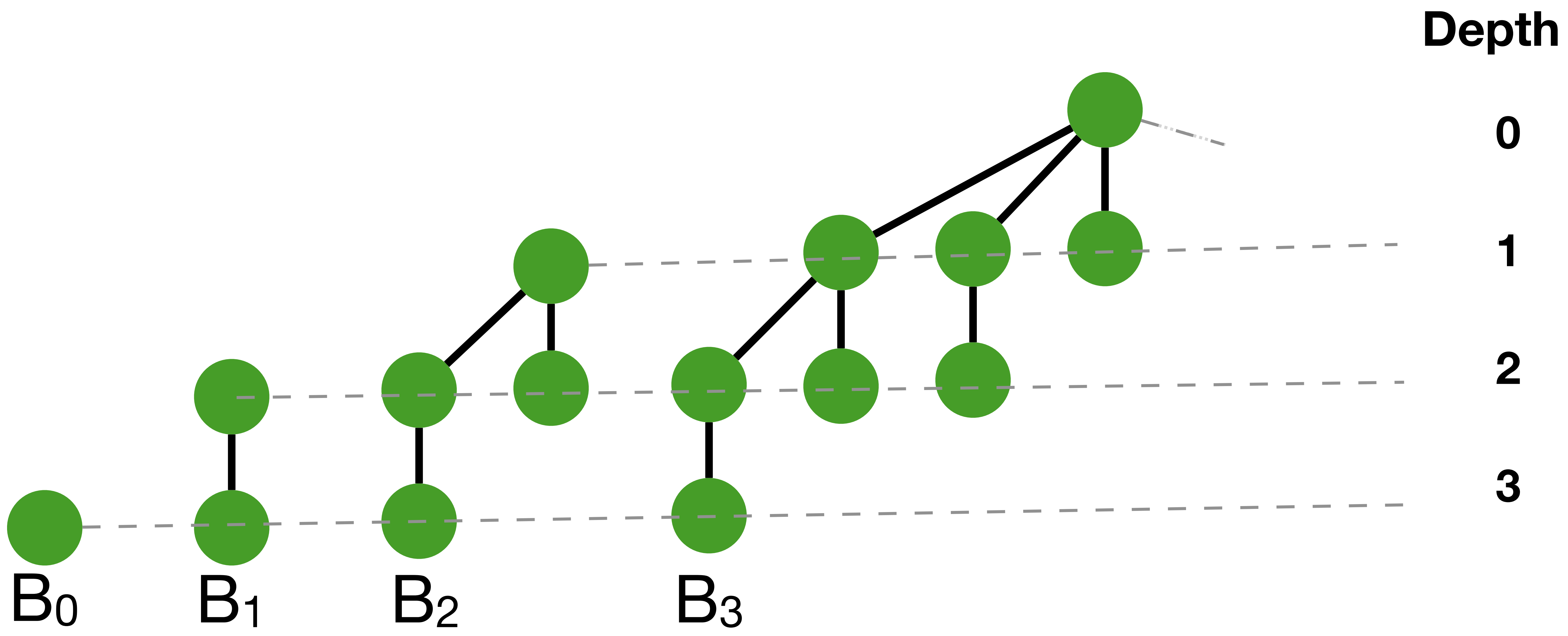
Binomial heap

- Collection of heap-ordered trees
- Minimum element is found by searching the roots of all trees
- Supports union operation efficiently
- Used to build other data structures such as Fibonacci heap

Binomial heap structure

- Collection of binomial heap trees : B_0, B_1, B_2, .., B_n
- B_0 has a single node
- B_k has two trees B_{k-1}





Properties of binomial tree

1. B_k has 2^k nodes
2. B_k has height k
3. At depth i, B_k has $k! / i! (k - i)!$ nodes
4. Root of B_k has degree k

Binomial heap property

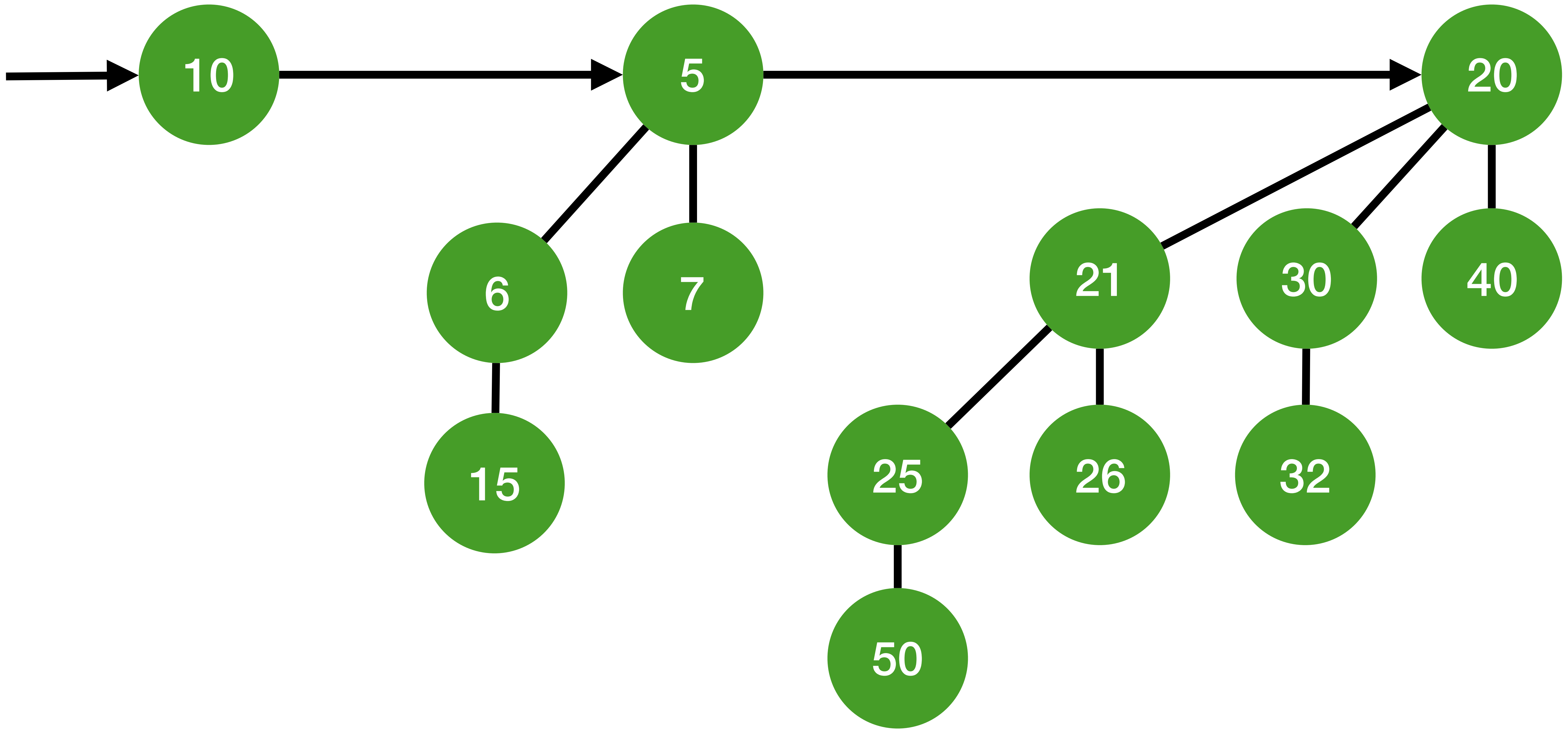
1. Min-heap property : key of any node is greater than or equal to its parent

2. For any non-negative integer k, the root of at most one binomial tree has degree k

Binomial heap implementation

N node binomial heap has at most $\log N + 1$ binomial trees

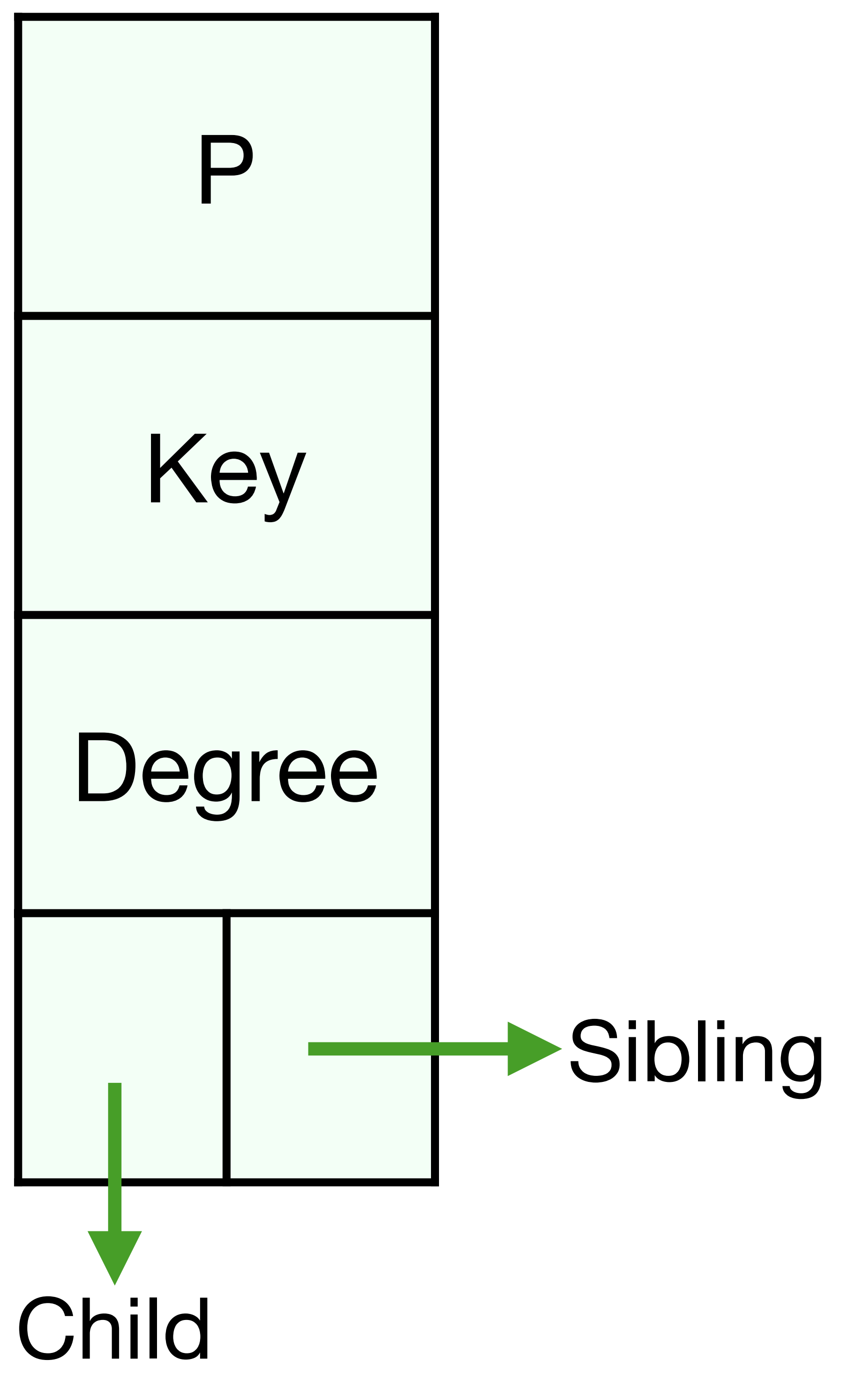
13 nodes <1 1 0 1> => <B_3 B_2 B_1 B_0>



Node representation

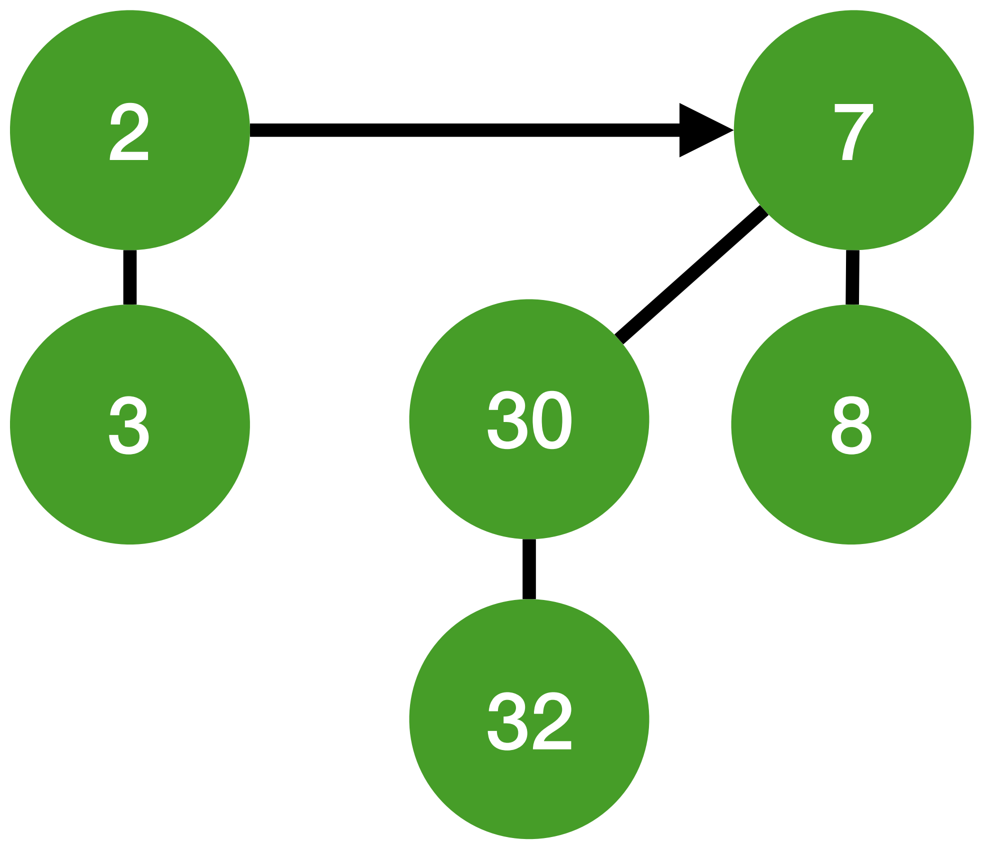
Each node stores :

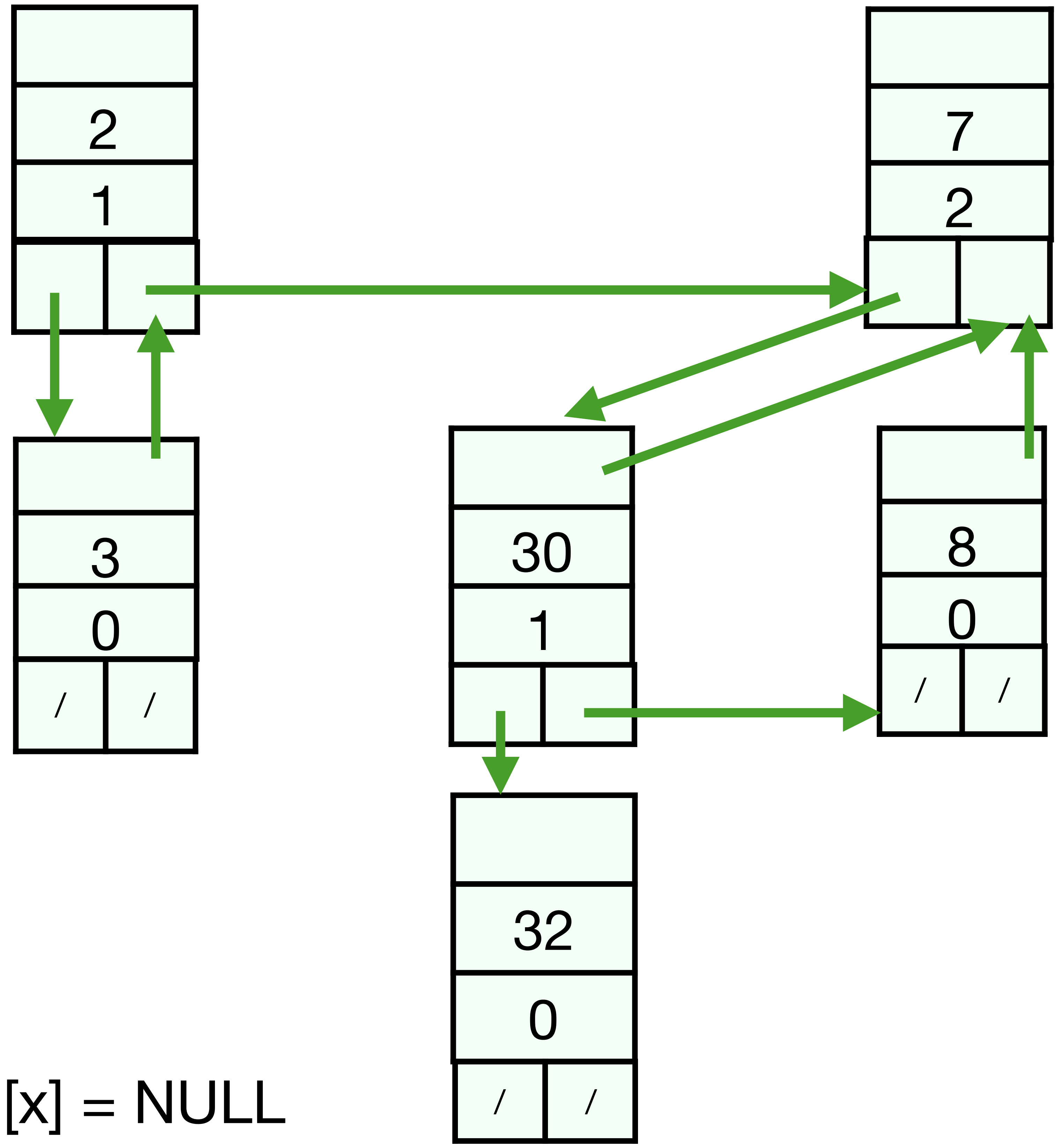
1. Pointer to its parent (if any)
2. Key
3. Degree
4. Pointer to left-most child (if any)
5. Pointer to right sibling (if any)



Binomial heap structure

Sibling and next node pointers in binomial heap





if x is rightmost child of its parent, sibling[x] = NULL

Binomial heap : create new

```
MakeBinomialHeap(){
   create H;
   head[H] = NULL;
   return H;
   }
Time Complexity : θ(1)
```

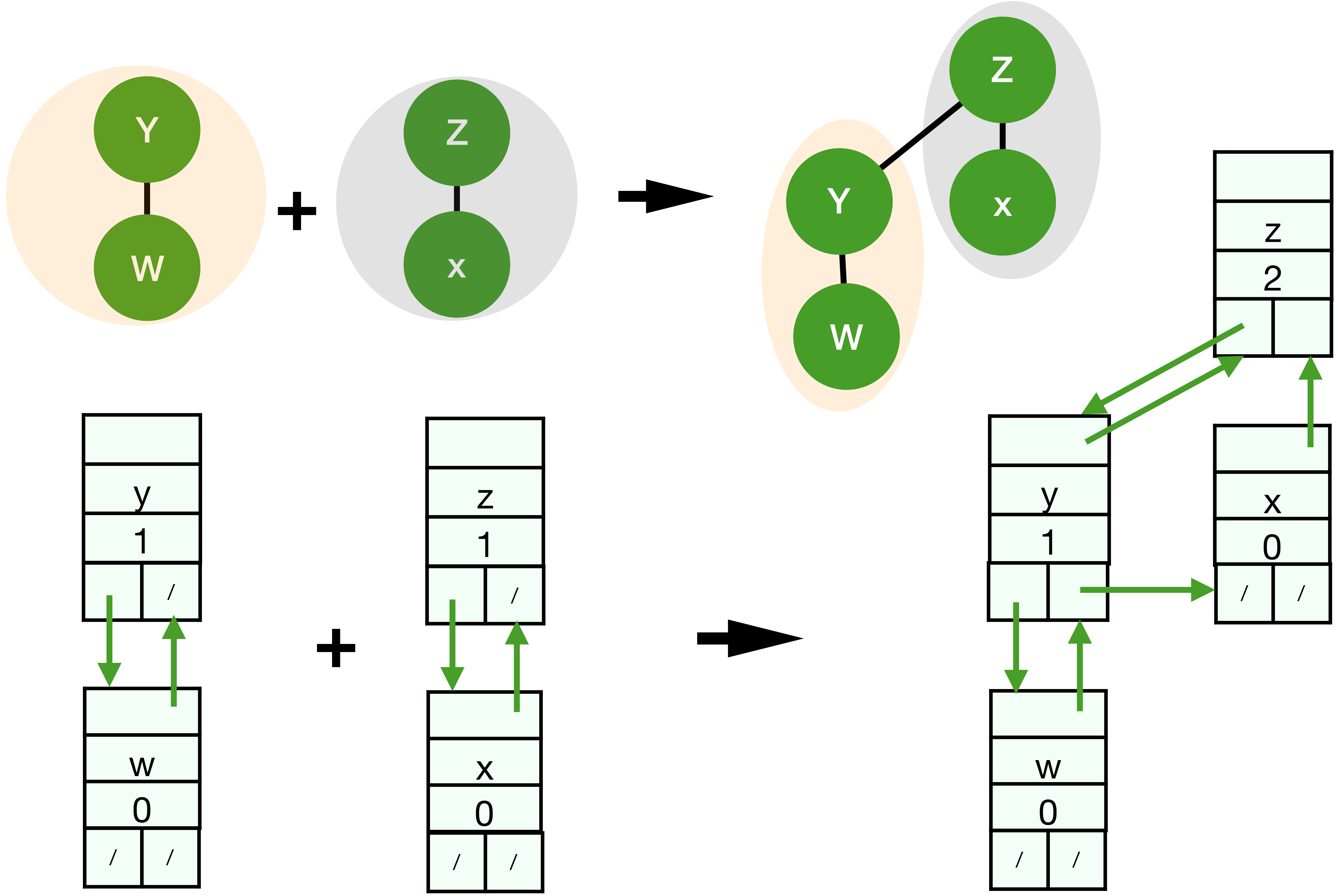
Binomial heap : find minimum key

Search through at most log N + 1 root nodes

```
Binomial_heap_minimum(H){
    y = null;
    x = head[H];
    min = ∞ ;
    while(x != NULL){
        if(key[x] < min){
          min = key[x];
          y = x ;
        }
       x = sibling[x];
    }
   return y;
}
```

Time Complexity : O(log n)

Binomial heap link diagram



Binomial heap link

Links trees whose trees have same degree so that z is parent of y

```
BINOMIAL_LINK(y,z){
  p[y] = z ;
  sibling[y] = child[z];
  child[z] = y;
  degree[z] = degree[z]+1;
}
```

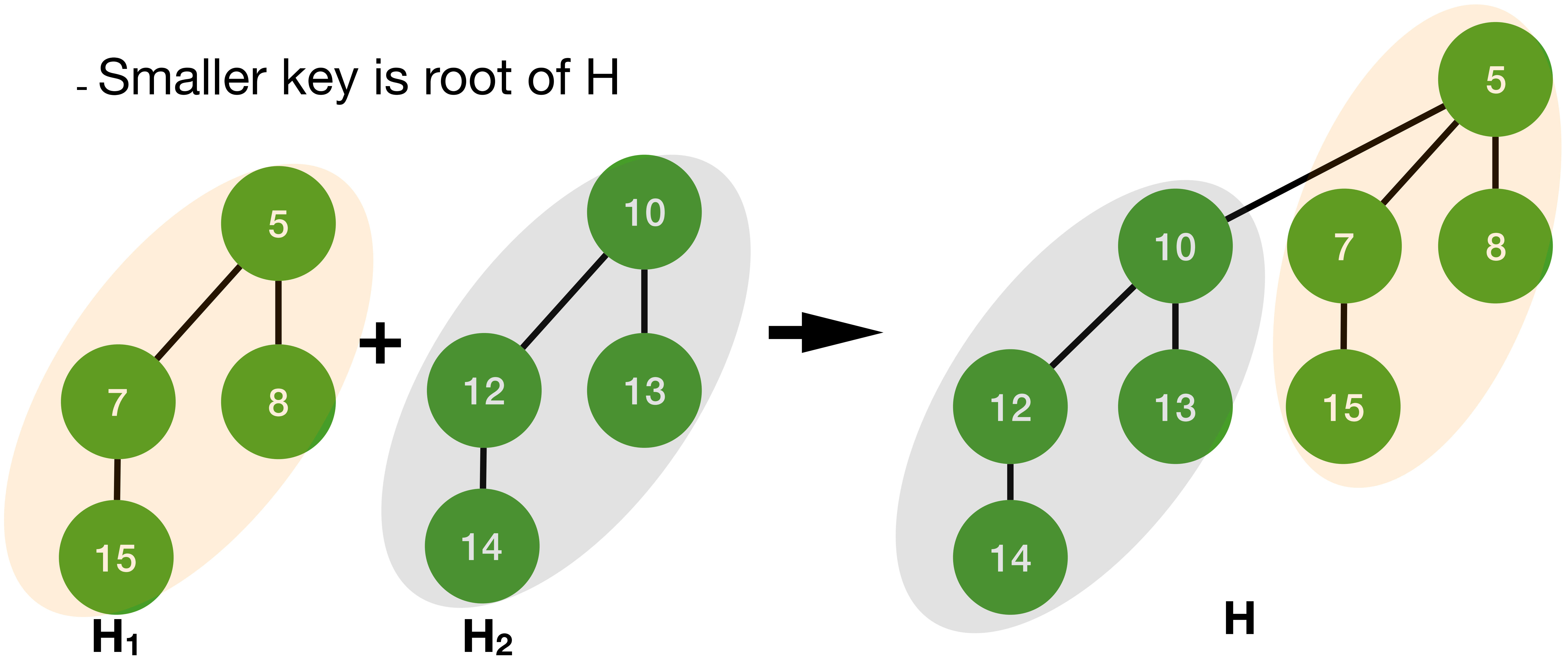
Binomial heap union

H_1 and H_2 have same order :

- H is union of H_1 and H_2

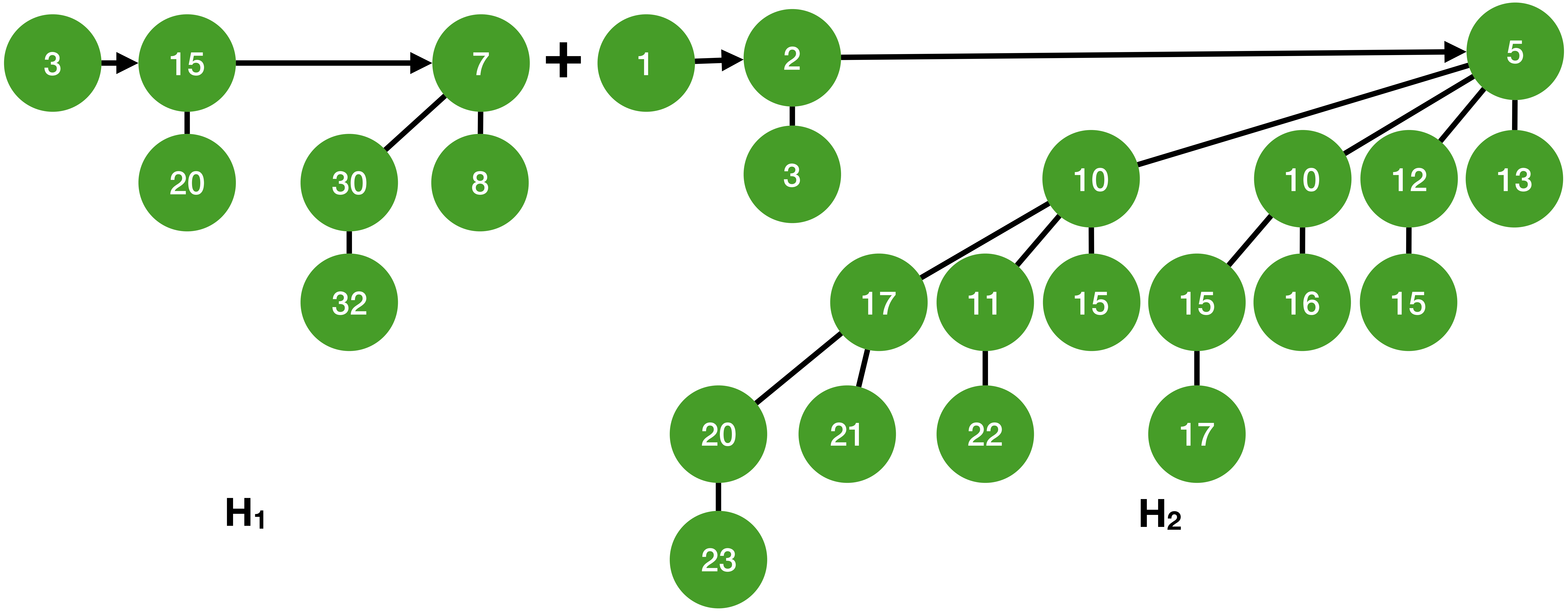
- Connect root of H_1 and H_2

- Smaller key is root of H



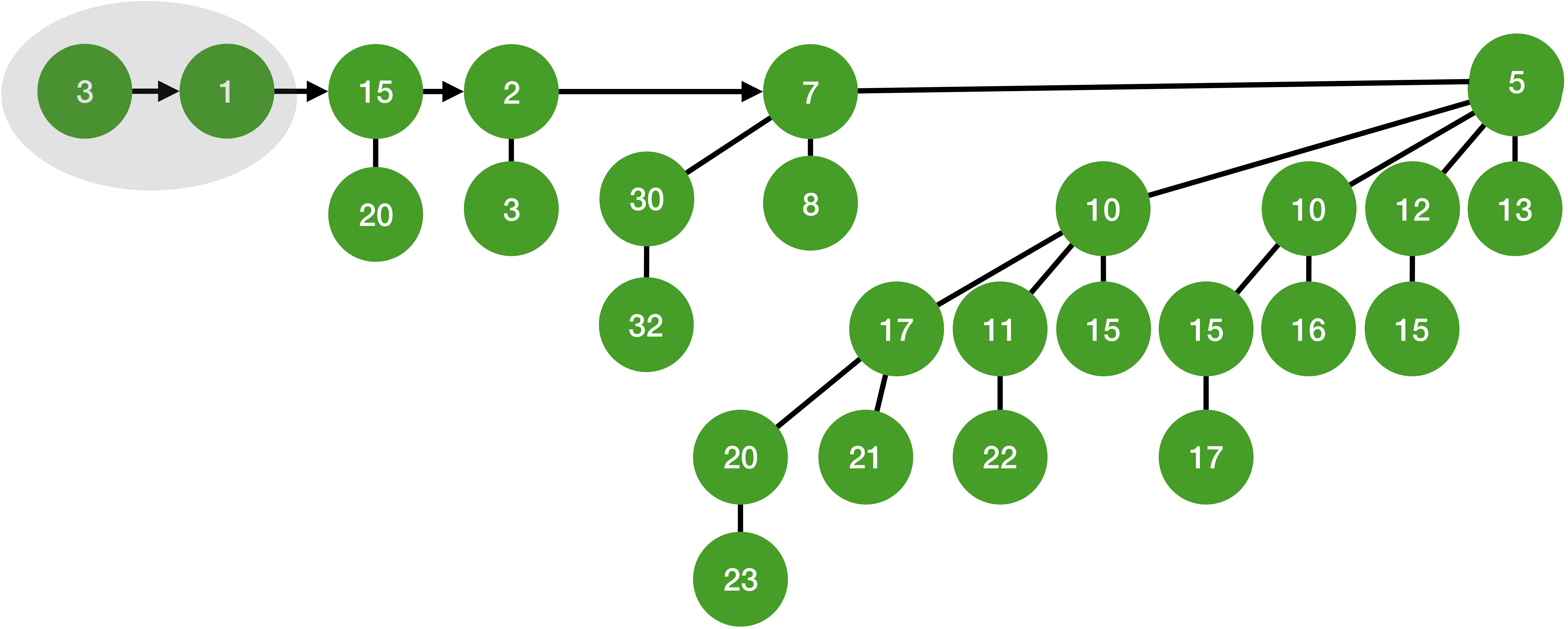
Binomial heap union example

H_1 and H_2 have different orders

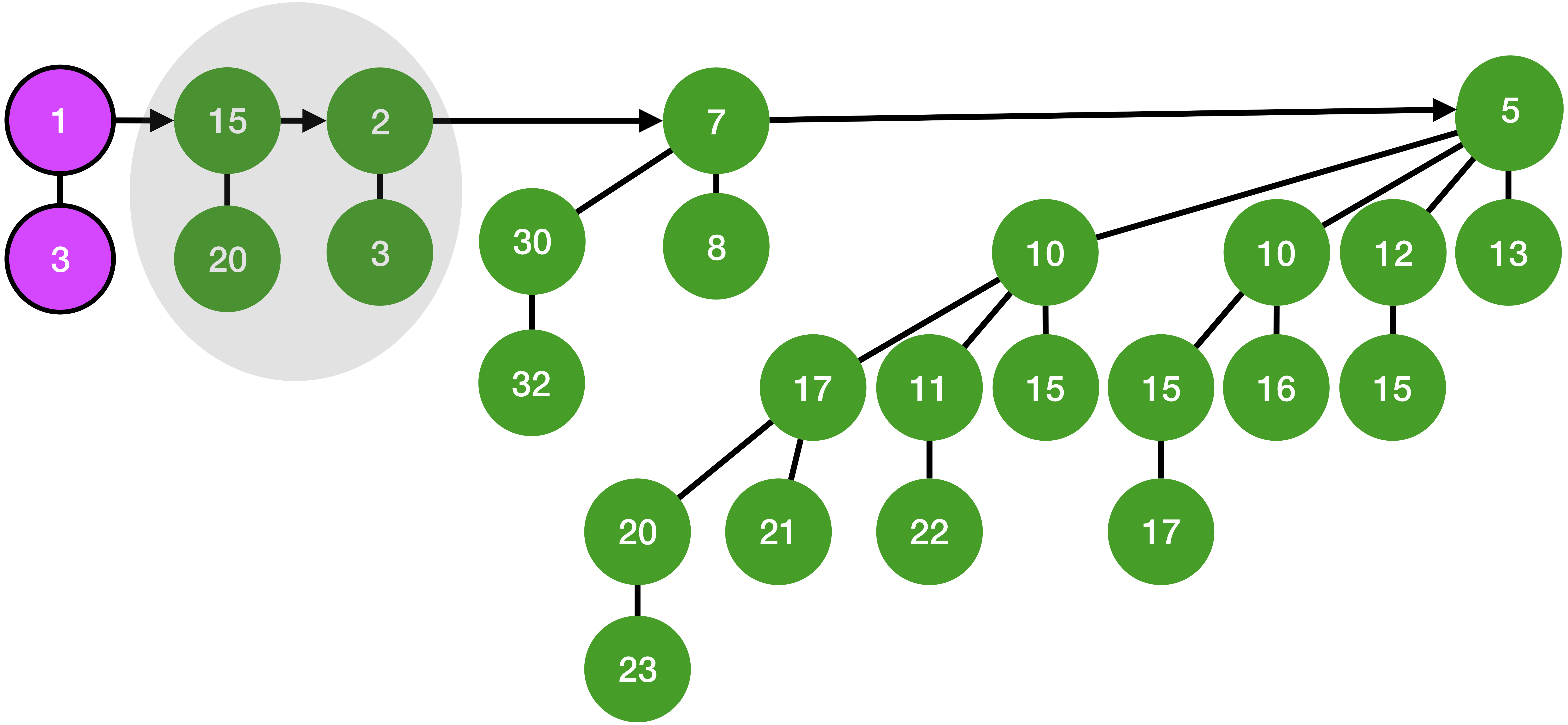


Binomial heap union example

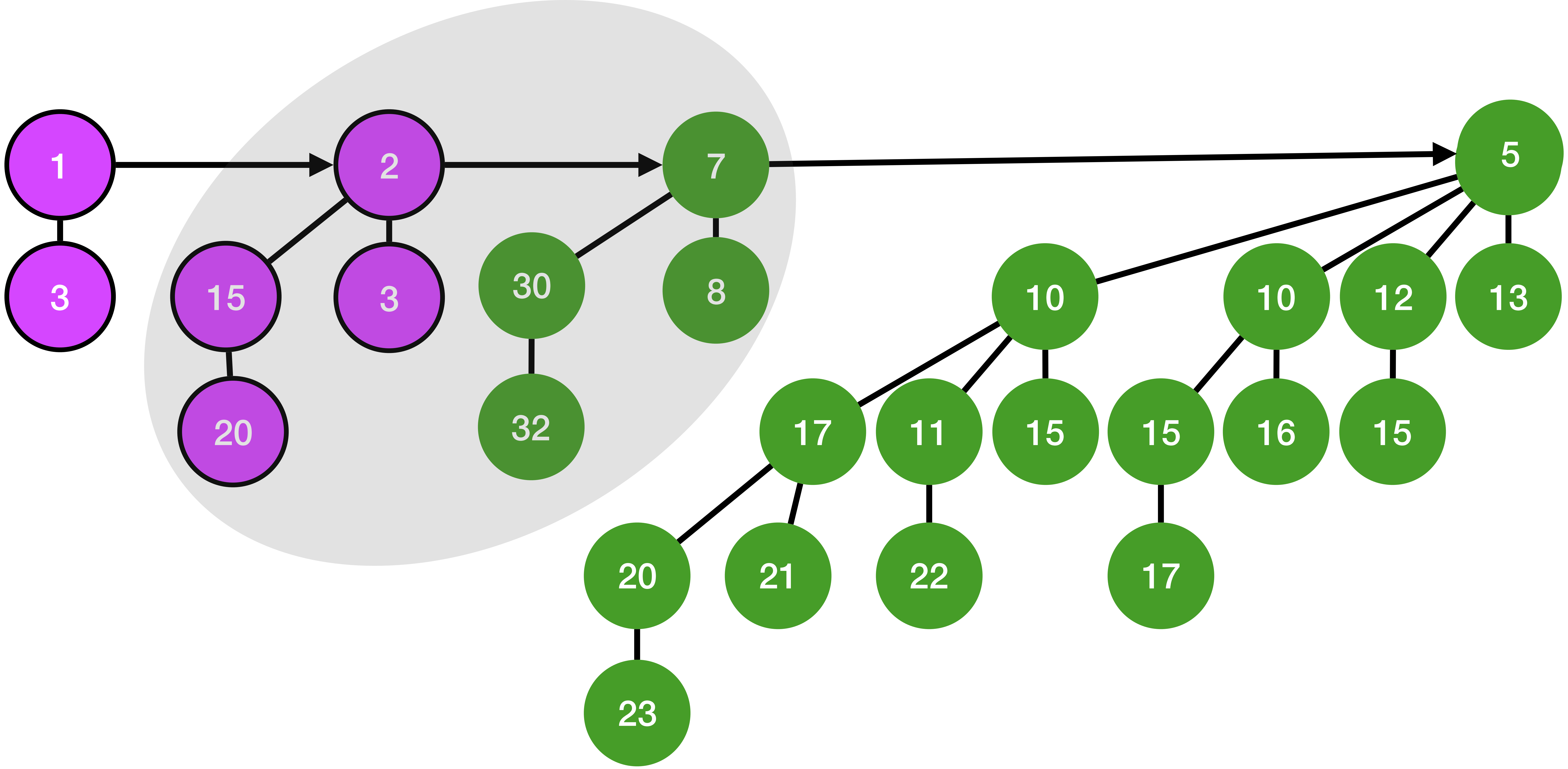
Step 2 : Link all trees whose roots have the same degree and new root has smaller key



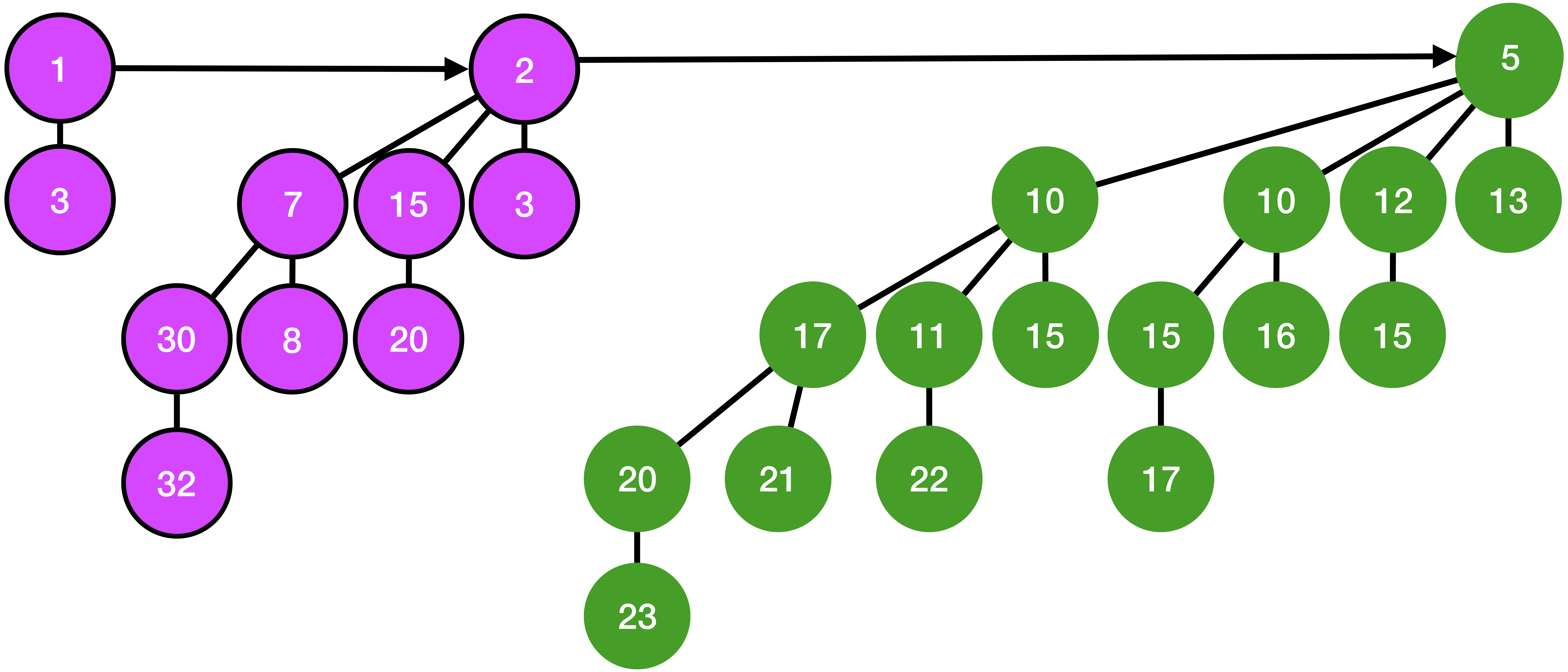
Binomial heap union example



Binomial heap union example



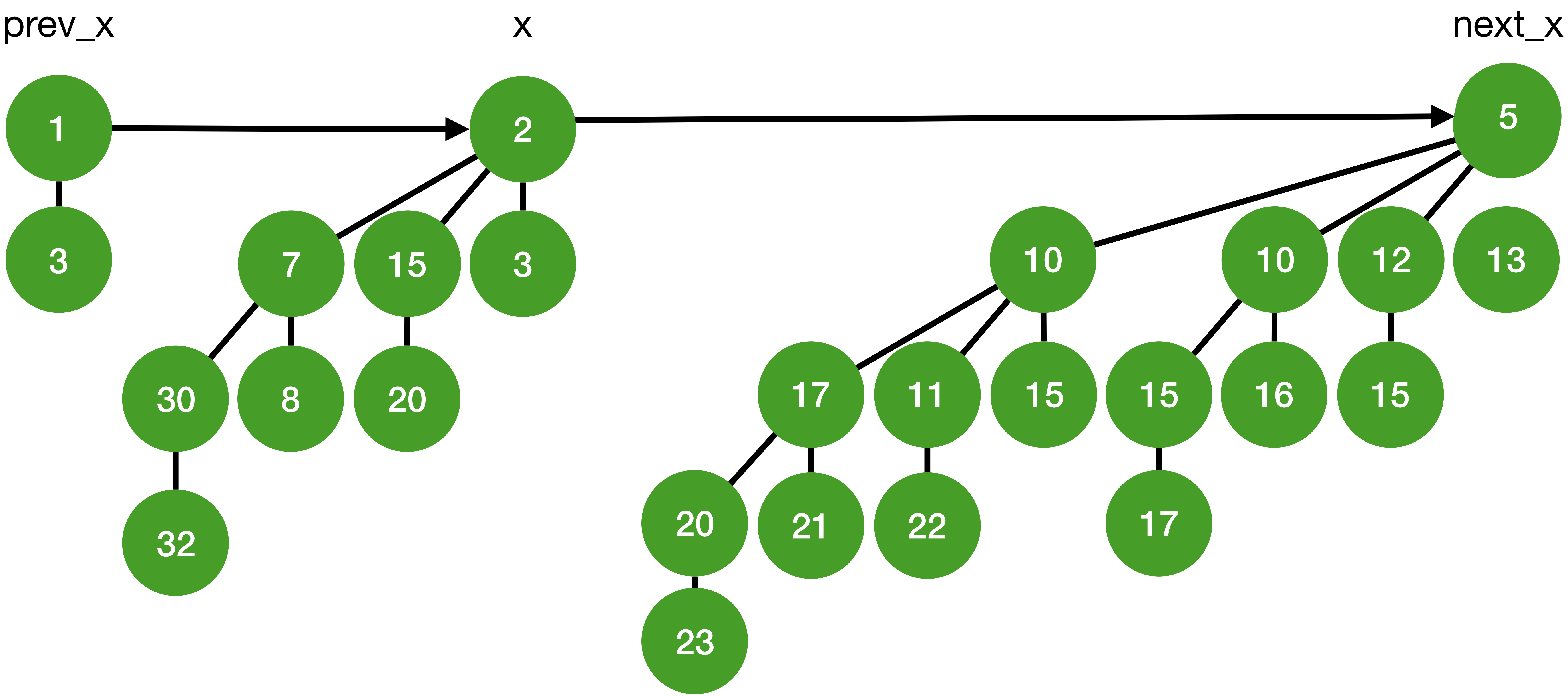
Binomial heap union example



Binomial heap union

- Four cases:
  - Case 1 : two adjacent roots with different degrees, move pointers down
  - Case 2 : three adjacent roots with same degree, move pointers down
  - Case 3 : two adjacent roots with same degree, key of first root is smaller, link together
  - Case 4 : two adjacent roots with same degree, key of second root is smaller, link together

Binomial heap union case 1



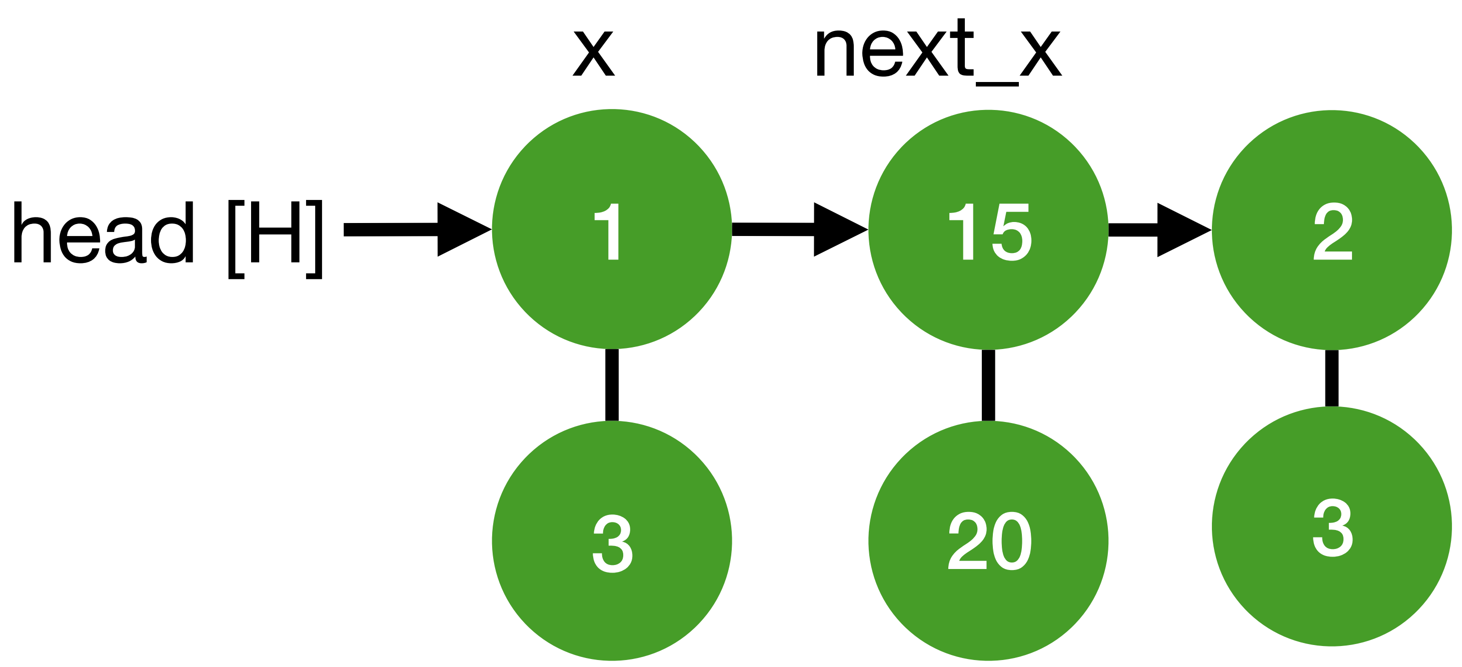
degree[x] != degree[next_x]

Move pointers one position further in list

Binomial heap union case 1 pseudocode

```
if(degree[x] != degree[next_x]){
  prev_x = x;
  x = next_x;
  next_x = sibling[x];
}
```

Binomial heap union case 2



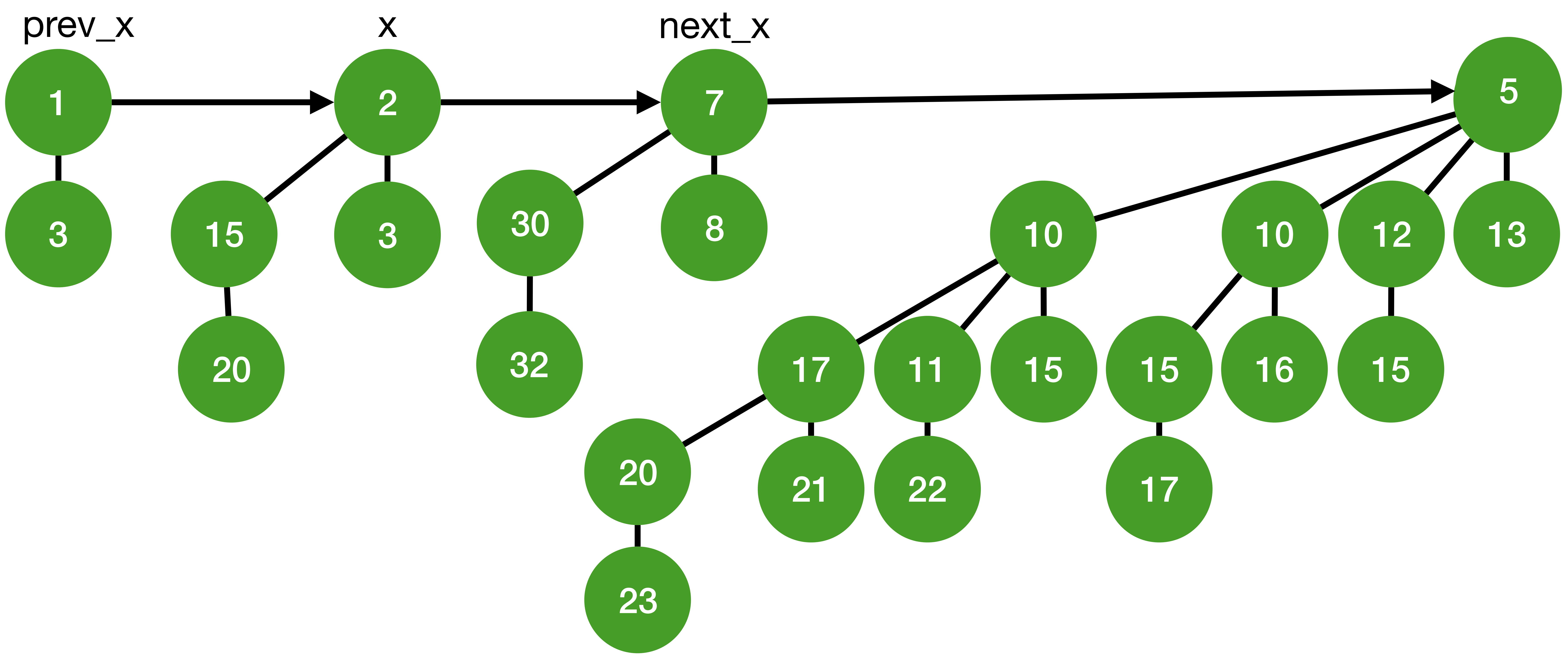
degree[x] == degree[next_x] == degree[sibling[next_x]]

Move pointers one position down

Binomial heap union case 2 pseudocode

```
if((degree[x] == degree[next_x]) && (degree[x] ==
degree[sibling[next_x])){
  prev_x = x;
  x = next_x;
  next_x = sibling[x];
}
```


Binomial heap union case 3



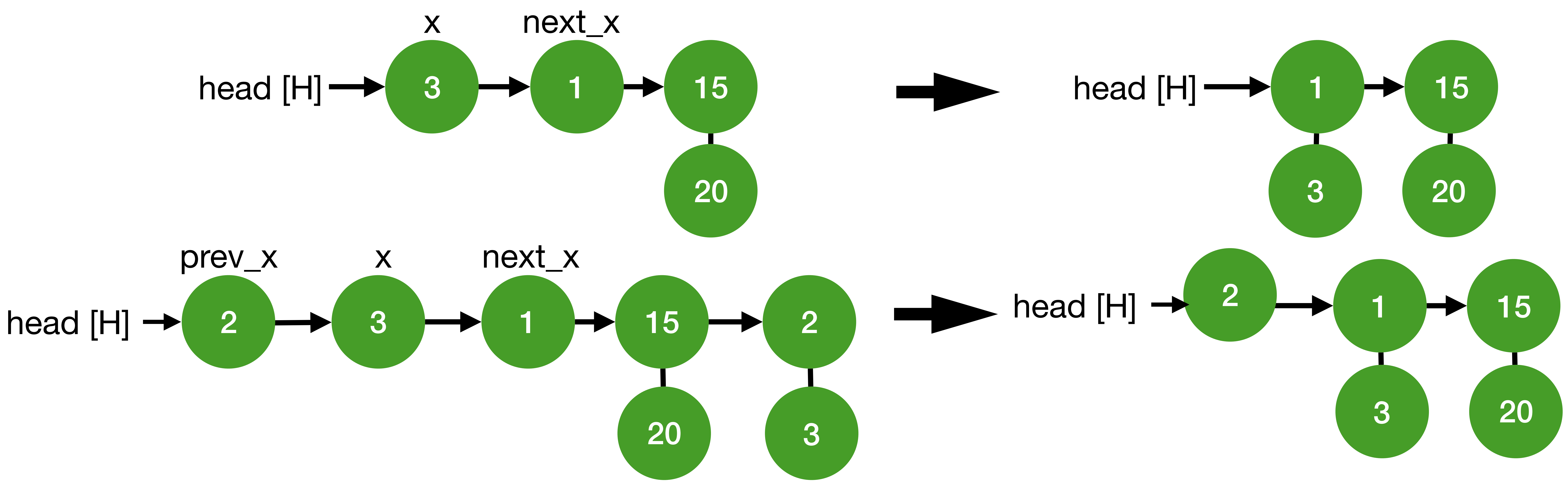
degree[x] == degree[next_x] != degree[sibling[next_x]]

AND key[x] ≤ key [next_x]

Binomial heap union case 3 pseudocode

```
if( (degree[x] == degree[next_x]) && (degree[x] !=
degree[sibling[next_x]]) ){
  if (key[x] ≤ key[next_x] ) {
     sibling[x] = sibling[next_x];
     BINOMIAL_LINK(next_x, x);
     next_x = sibling[x];
  }
}
```


Binomial heap union case 4



degree[x] == degree[next_x] != degree[sibling[next_x]]

key[x] > key [next_x]

degree[3] = degree[1] != degree[15]

Binomial heap union case 4 pseudocode

```
if( (degree[x] == degree[next_x]) && (degree[x] !=
degree[sibling[next_x]]) ){
   if(key[x] > key[next_x]){
      if(prev_x == NULL)
         head[H] = next_x;
      else
         sibling[prev_x] = next_x;
      BINOMIAL_LINK(x, next_x);
      x = next_x;
      next_x = sibling[x];
   }
 }
```


Binomial heap union complete pseudocode

```
BINOMIAL-HEAP-UNION(H1,H2)
    H = MAKE-BINOMIAL-HEAP()
    head[H] = BINOMIAL-HEAP-MERGE(H1,H2)
    free the objects H1 and H2 but not the lists they point to
    if head[H] == NULL  then return H
    prev_x = NULL
    x = head[H]
    next_x = sibling[x]
    while next_x != NULL
        if (degree[x] != degree[next_x]) ||(sibling[next_x] != NULL &&
        degree[sibling[next_x]]== degree[x])
             then prev_x <- x
                  x <- next_x
             else if key[x] <= key[next_x]
                  then sibling[x] = sibling[next_x]
                      BINOMIAL-LINK(next_x, x)
                  else if prev_x == NULL
                       then head[H] = next_x
                       else sibling[prev_x] = next_x
                       BINOMIAL-LINK(x, next_x)
                       x = next_x
        next_x = sibling[x]
    return H
```

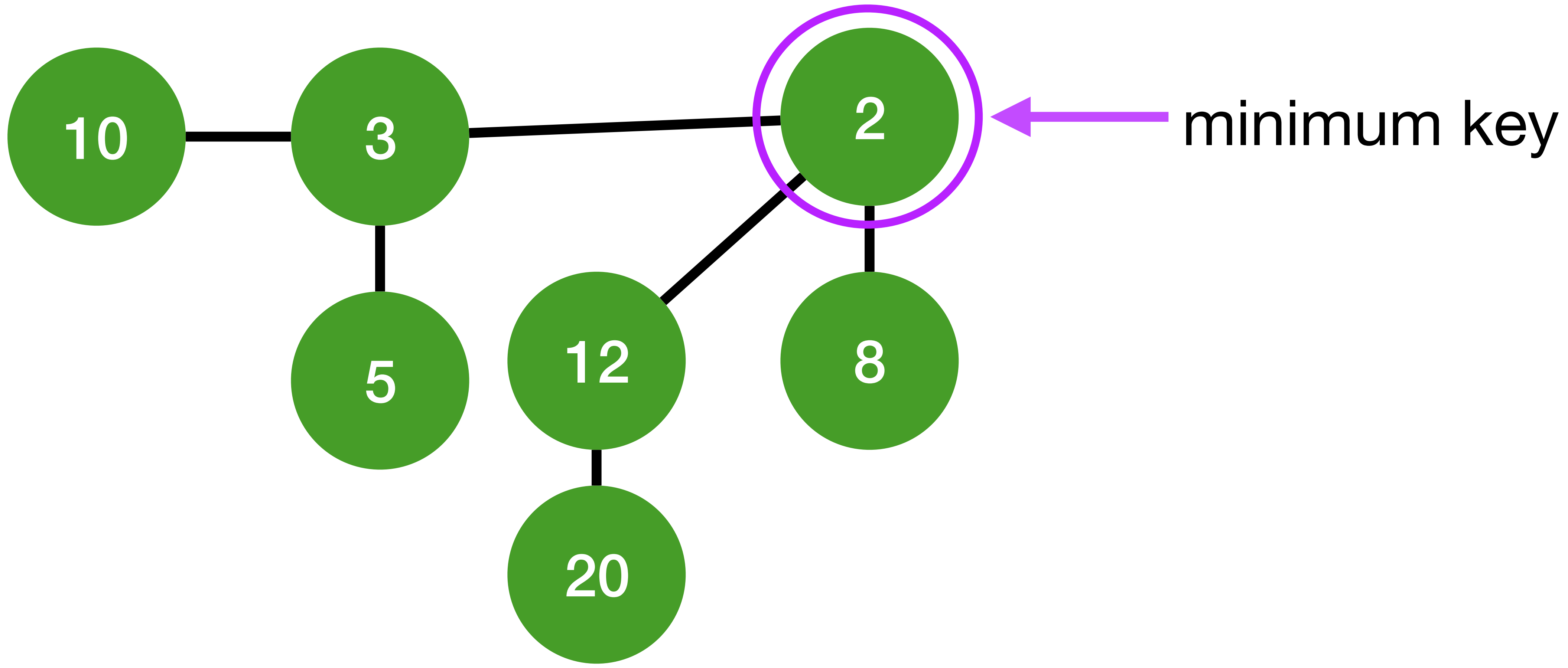
Binomial heap inserting a node

1. Create a new heap with new node n and initialize it
2. Merge the two heaps

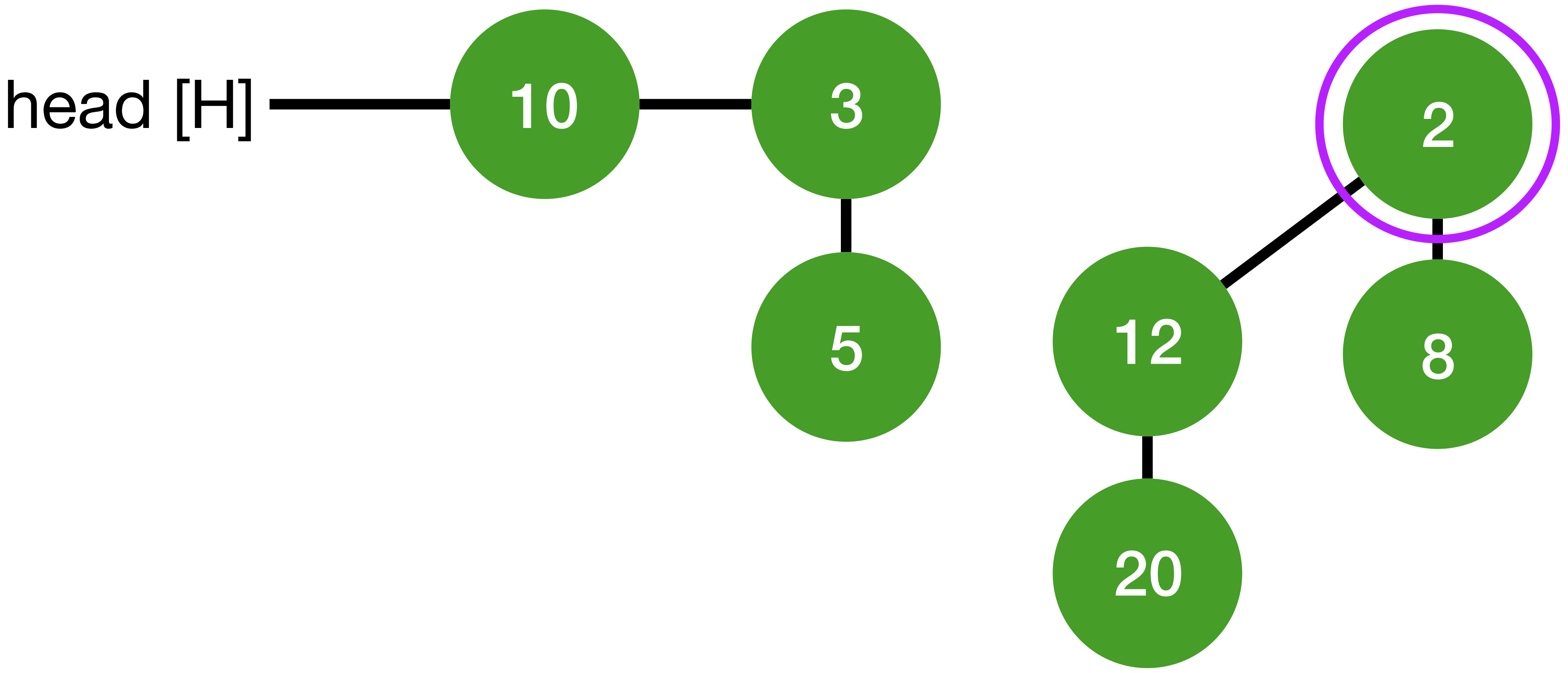
Binomial heap inserting a node

```
BinomialHeapInsert(H, n){
  H' = MakeBinomialHeap();
  head[H'] = n;
  H = BinomialHeapUnion(H, H');
}
//Node n is initialized as follows
P[n] = child[n] = sibling[n] = NULL;
degree[n] = 0;
Key[n] = value;

Time Complexity : O(log n)
```


Binomial heap extract node with minimum key



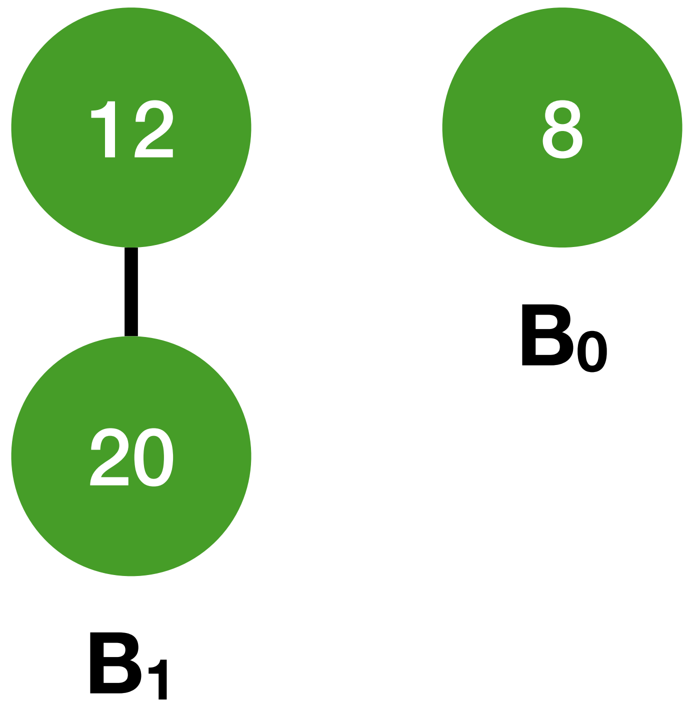
1. Remove tree vertex whose root x has minimum key

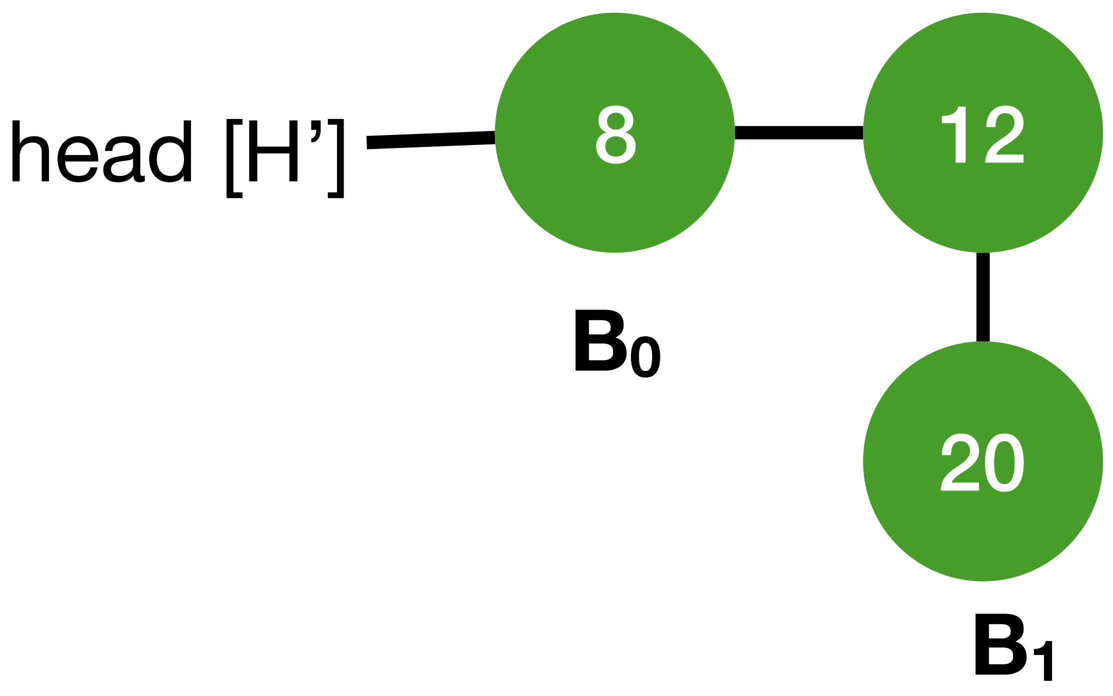


Binomial heap extract node with minimum key

2. Create empty binomial heap H’ with head [H’]

3. When root x of B_k tree is removed, its children are trees B_{k-1} , B_{k-2} ; B_0

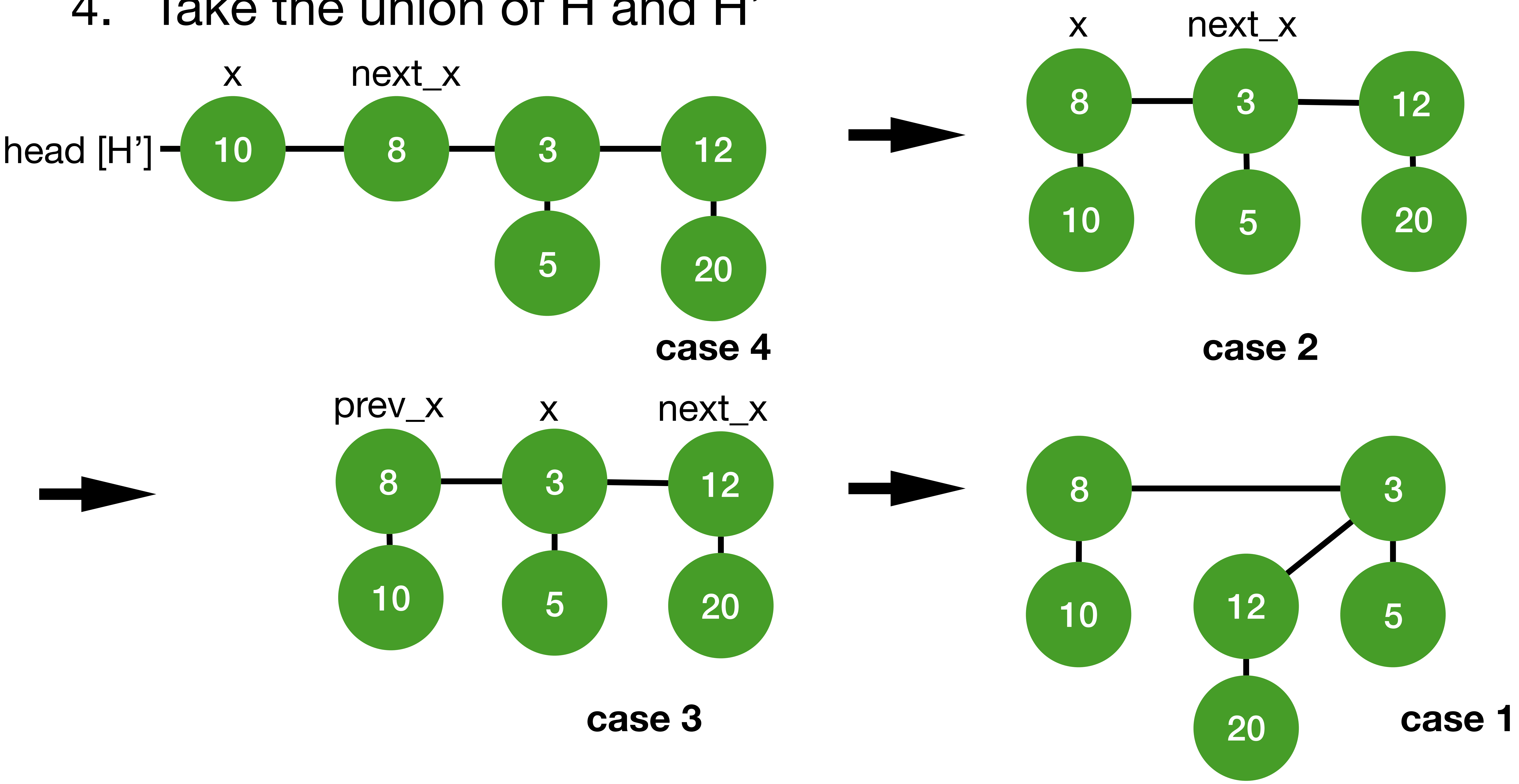




Reverse the order of linked-list and set H’ to point to it

Binomial heap extract node with minimum key

4. Take the union of H and H’



Binomial heap extract node with minimum key (pseudocode)

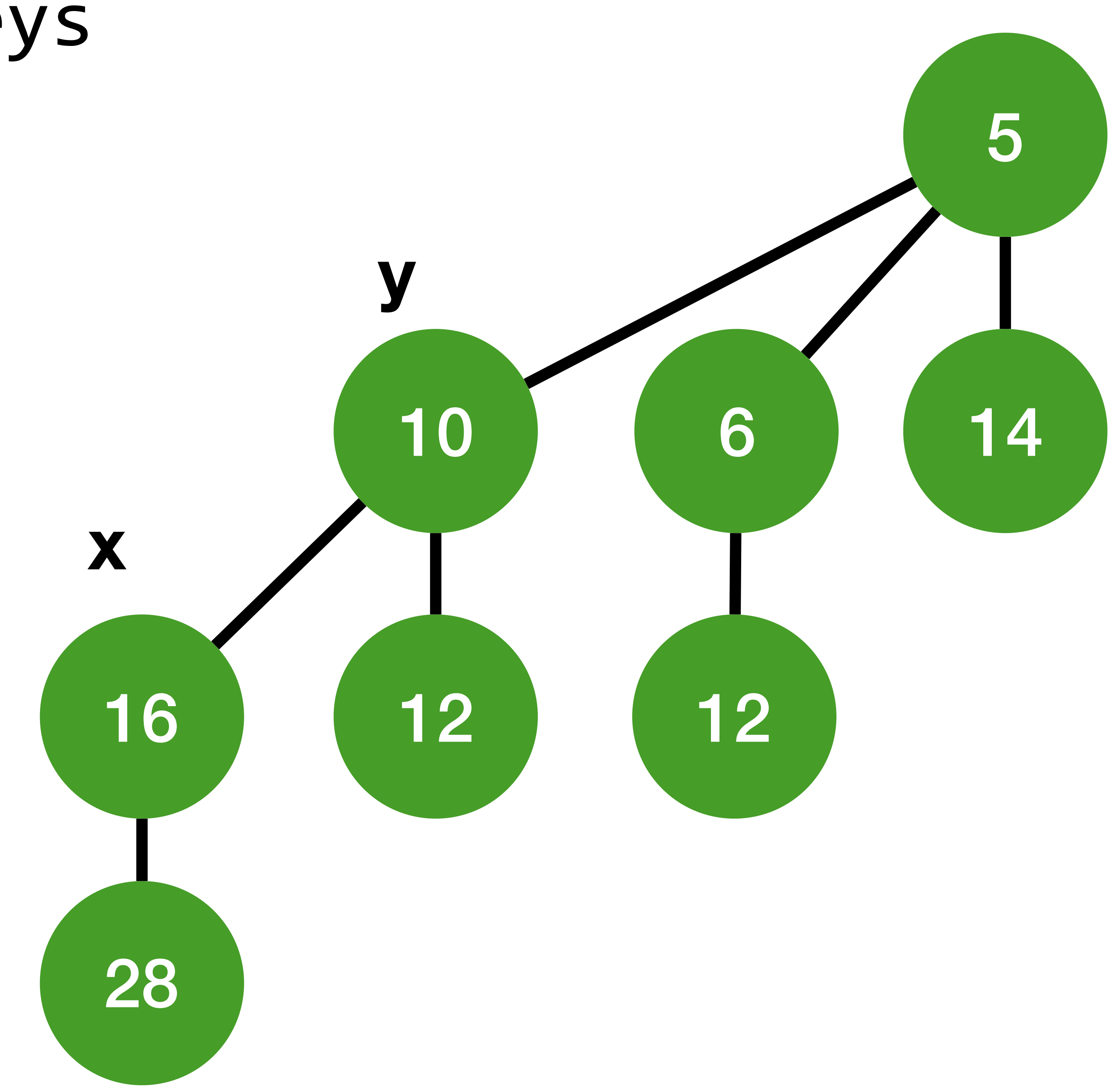
```
BinomialHeapExtractMin(H){
    x = BinomialHeapMinimum(H);
    unlink x from H;
    H’ = MakeBinomialHeap();

    L = reverse order of linked list of x’s children;

    set H’ to point to new list L;
    H = BinomialHeapUnion(H, H’)

    return x;
}
```

Each operation takes $O(\lg N)$ where H has N nodes.

Binomial heap decreasing a key

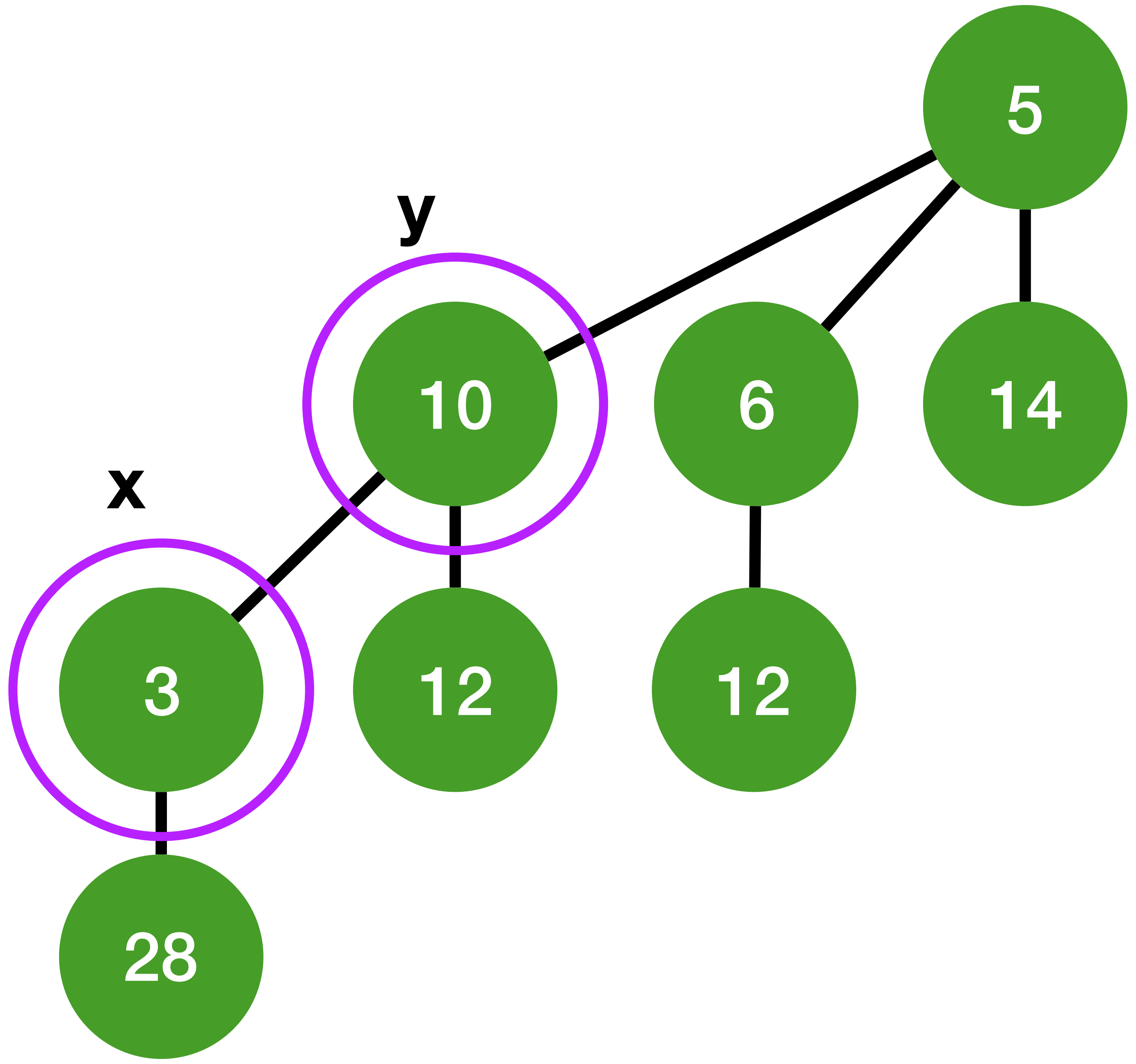
Decrease x's key to 3

```
if (x's key < parent[x]'s key)
swap x and y keys
```



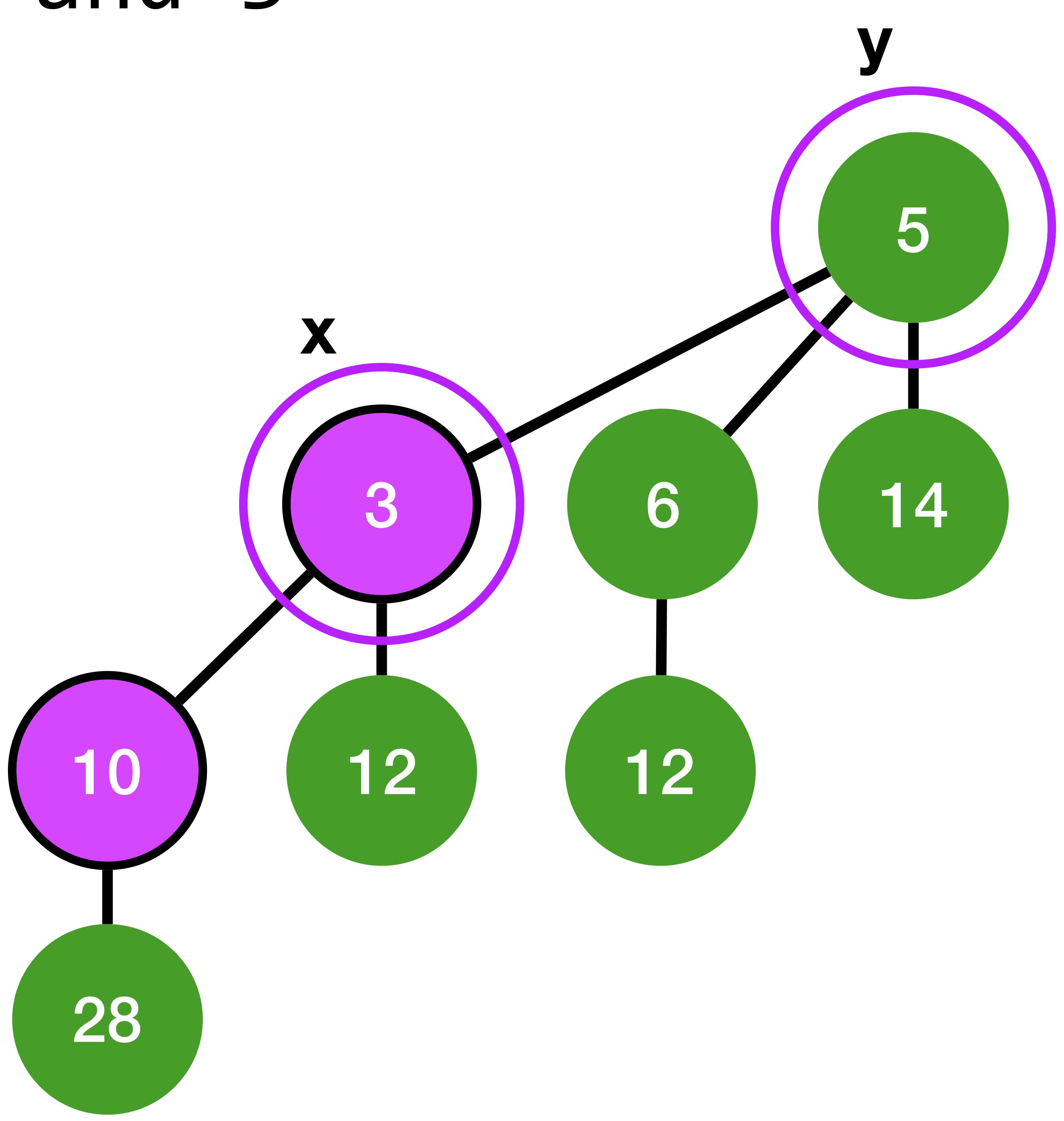
Binomial heap decreasing a key

```
3 > 10 swap 3 and 10
```

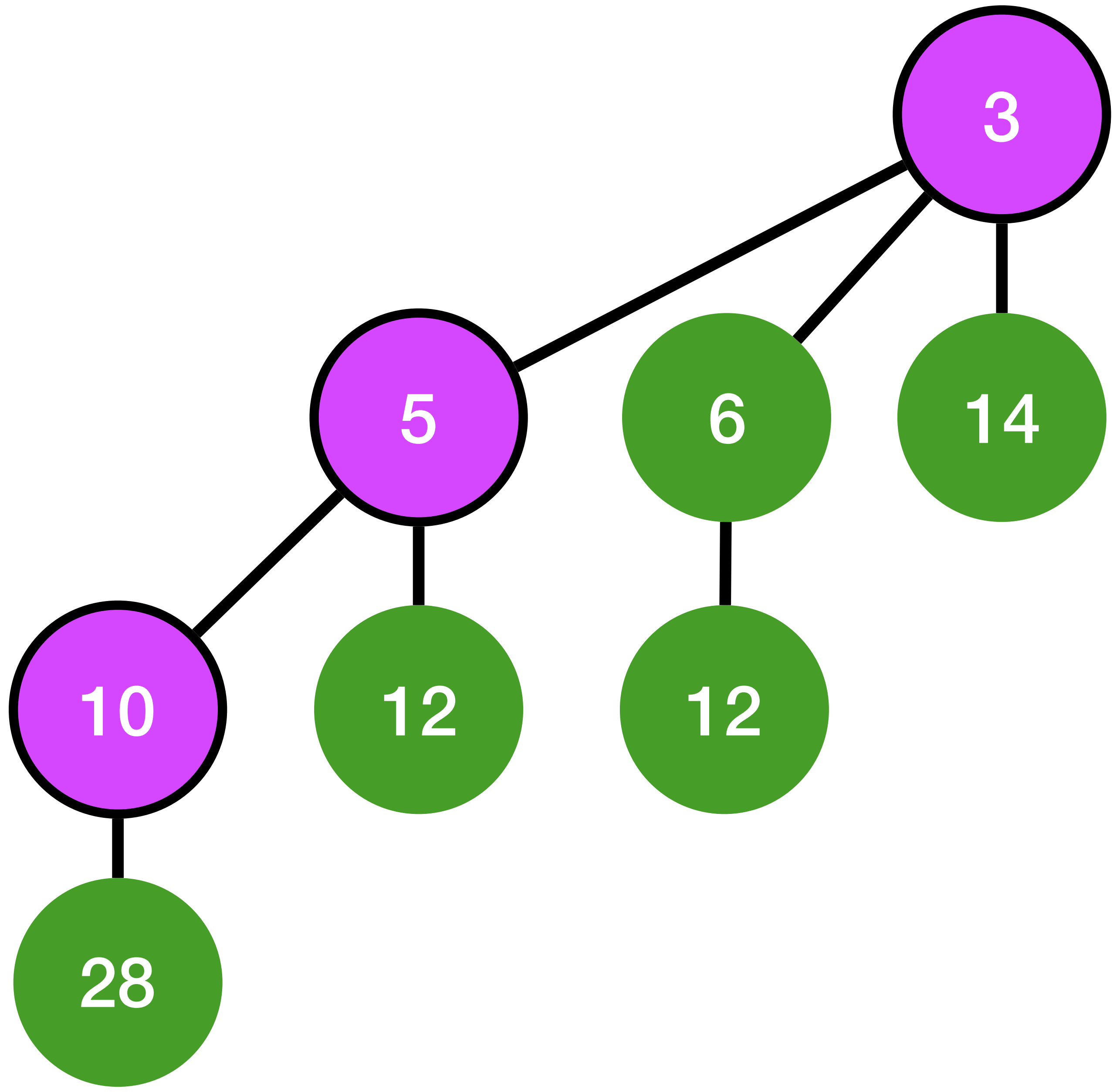


Binomial heap decreasing a key

`5 > 3 swap 5 and 3`



Binomial heap decreasing a key



Binomial heap decreasing a key pseudocode

```
BinomialHeapDecreaseKey(H, x, K){
   if(K > key[x])
      display error message;

   key[x] = K;
   y = P[x];

   while(y != NULL && ( key[x] < key[y] )){
      swap(key[x], key[y]);
      x = y;
      y = P[x];
    }
 }

Time Complexity : O(lg N)
```


Binomial heap deleting a key

1. Decrease the key to minimum possible value (-∞) by using the BinomialHeapDecreaseKey method

2. Delete this key by calling the Binomial Heap Extract Min method

Time Complexity : O(lg N)

References

- Cormen, Thomas H., et al. *Introduction to Algorithms*. The MIT Press, 2014.
- M. A. Weiss, Data Structures and Algorithm Analysis in C, Pearson, 4th edition, 2014.

