Binomial Heap

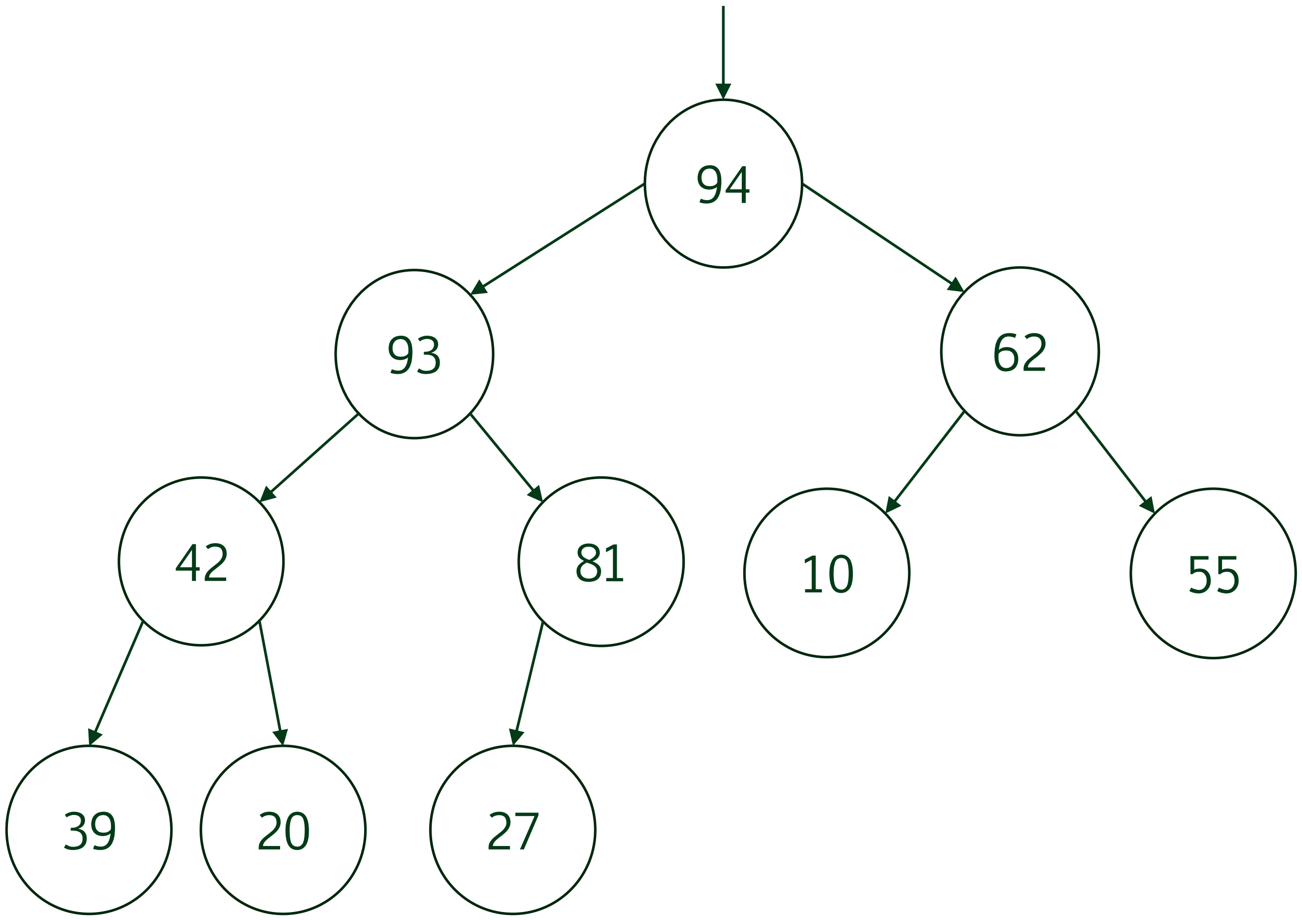
Vuillemin (1978)

Quick Review

- A binary heap, implemented as a linear array, represents a binary tree that satisfies two properties:
  - The binary tree is nearly complete
  - The priority of each node (except the root) is less than or equal to the priority of its parent
- The operations to insert an item and to remove the item with the highest priority take O(log n) time

Example

where higher-valued items have higher priorities



Motivation

- The binomial heap behaves in the same way as a binary heap with one additional capability

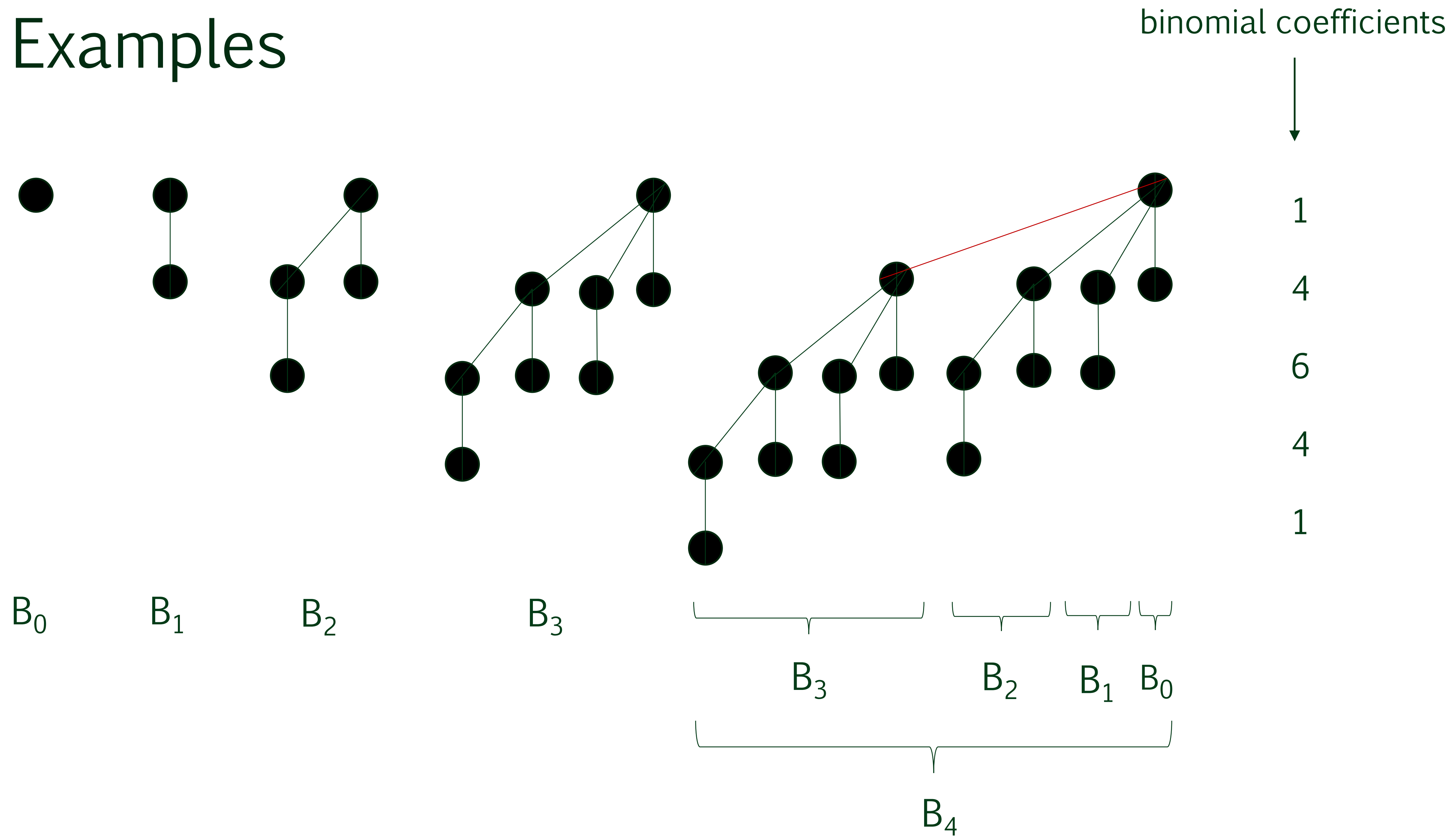
- A binomial heap is an example of a mergeable data structure whereby two binomial heaps can be efficiently merged into one

- To merge two binary heaps takes O(n) time, but to merge two binomial heaps takes only O(log n) time

Binomial Tree

- A binomial tree B_k is defined recursively in the following way:
  - The binomial tree B_0 consists of a single node
  - The binomial tree B_k consists of two binomial trees B_{k-1} where one is the leftmost child of the root of the other

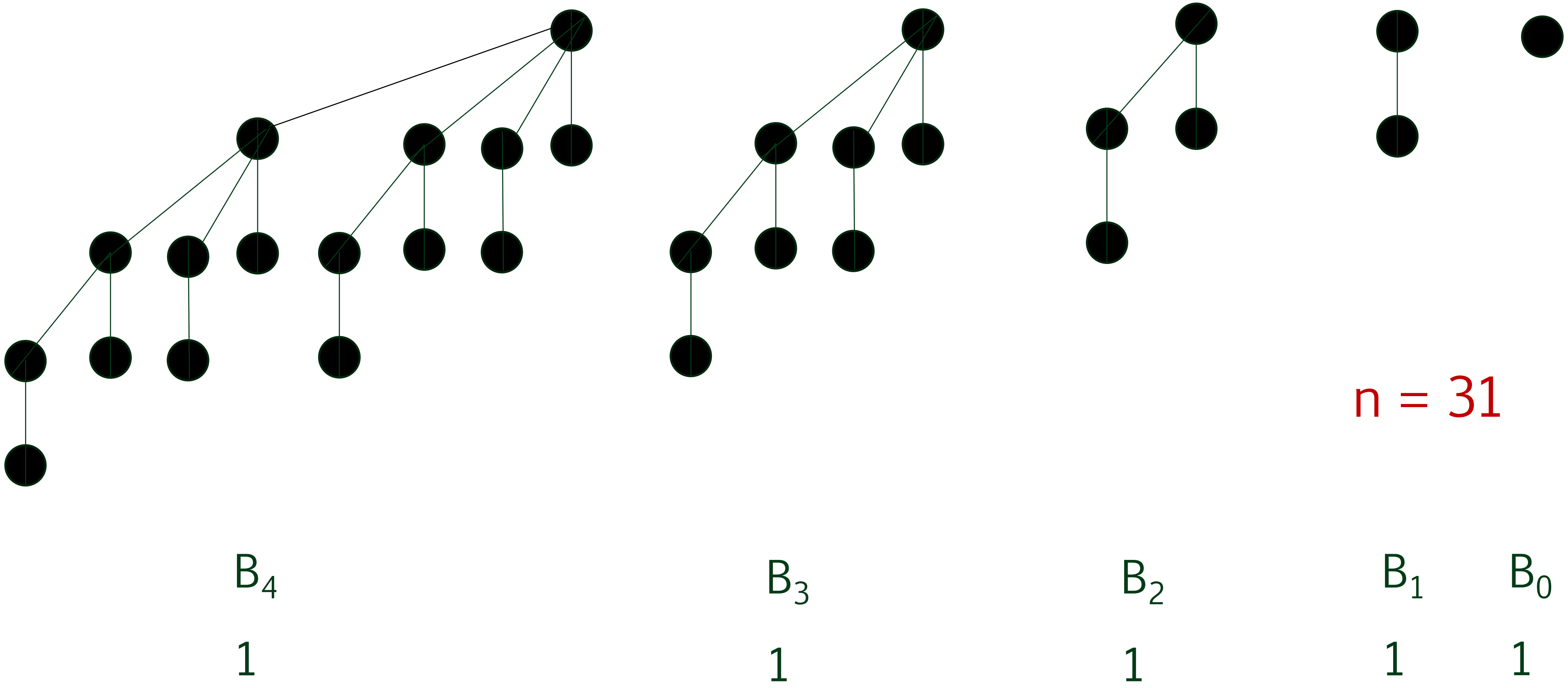
Examples

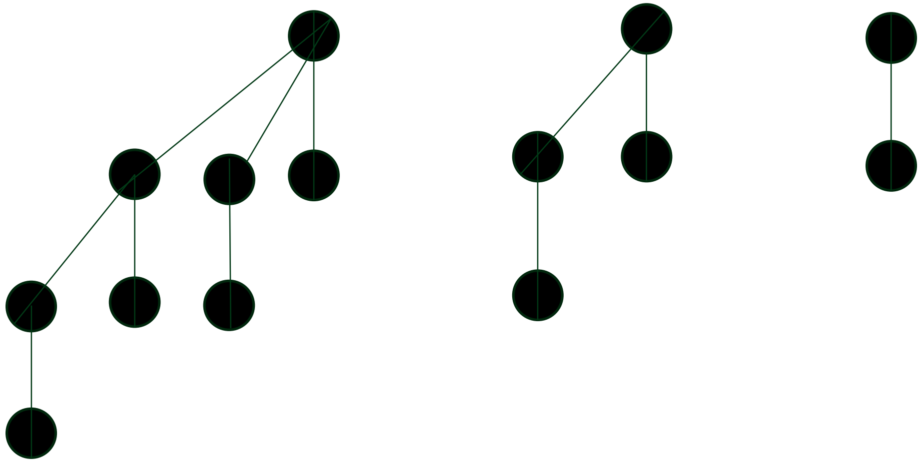


For any binomial tree B_k

- There are 2^k nodes
- The height of the tree is k
- There are exactly $\binom{k}{i}$ nodes at depth i, i = 0,1,2,...,k
- The root has degree k

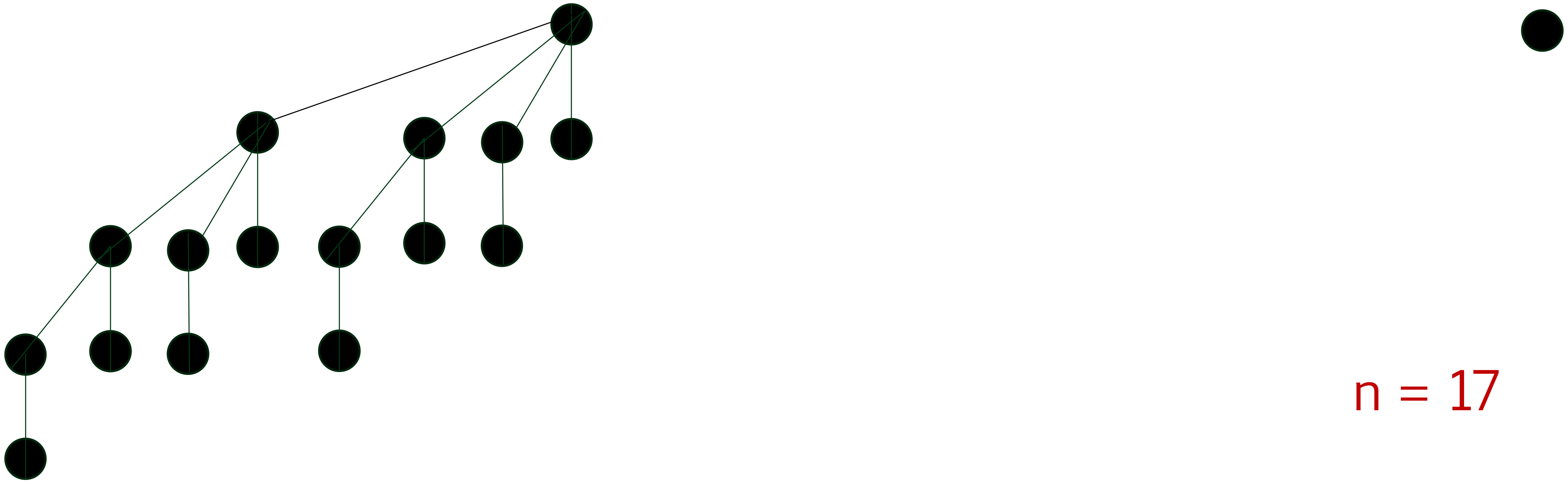
How does a set of binomial trees represent n items?



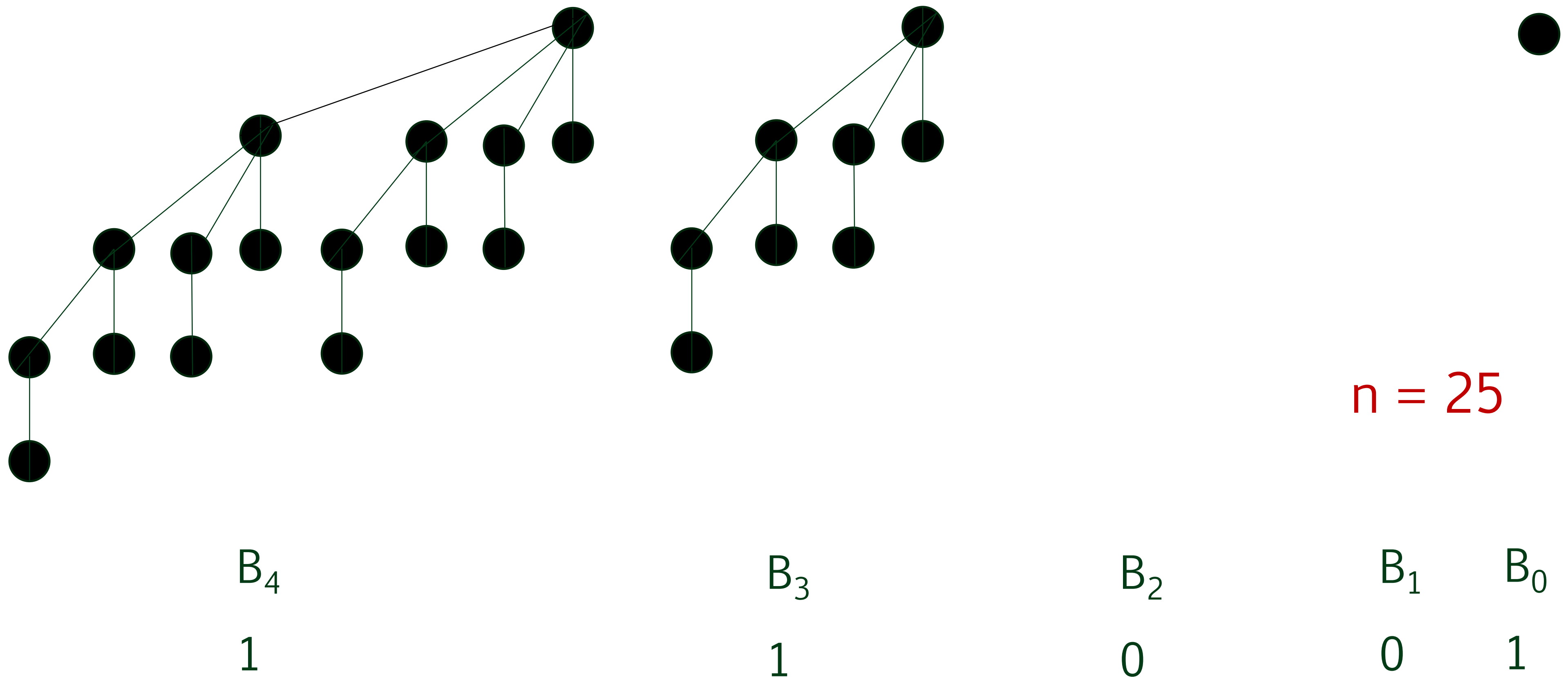


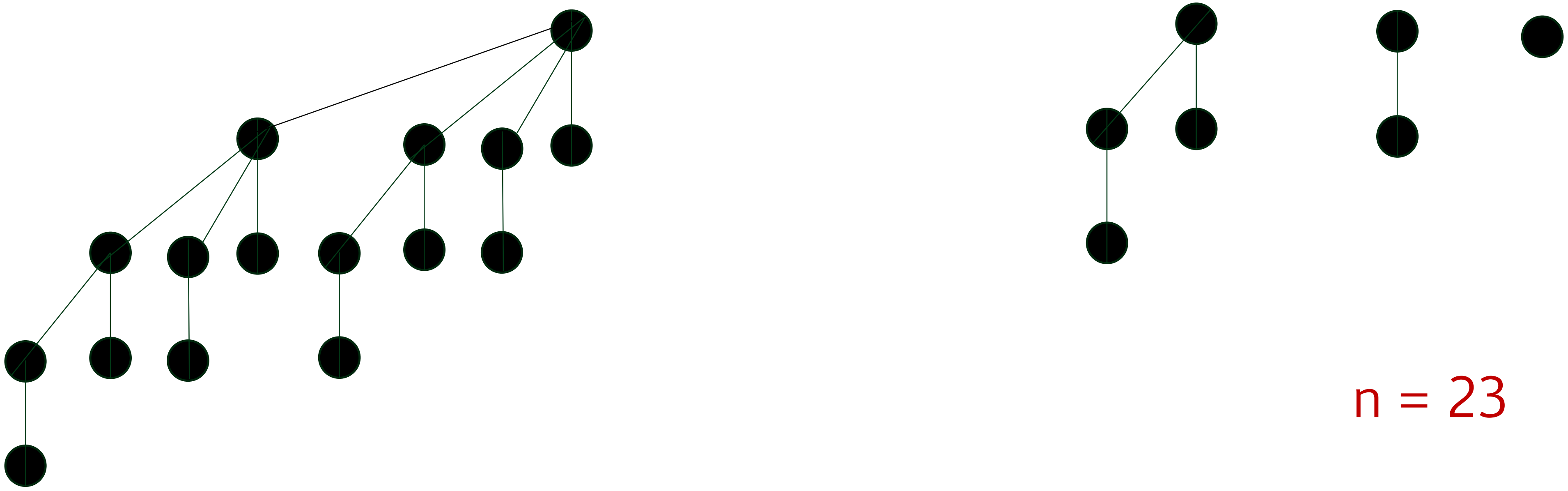
n = 14

B_4	B_3	B_2	B_1	B_0
0	1	1	1	0



B_4	B_3	B_2	B_1	B_0
1	0	0	0	1


n = 25
B_4
1
B_3
1
B_2
0
B_1
0
B_0
1



$n = 23$

B_4	B_3	B_2	B_1	B_0
1	0	1	1	1

Now many binomial trees are needed to represent n items?

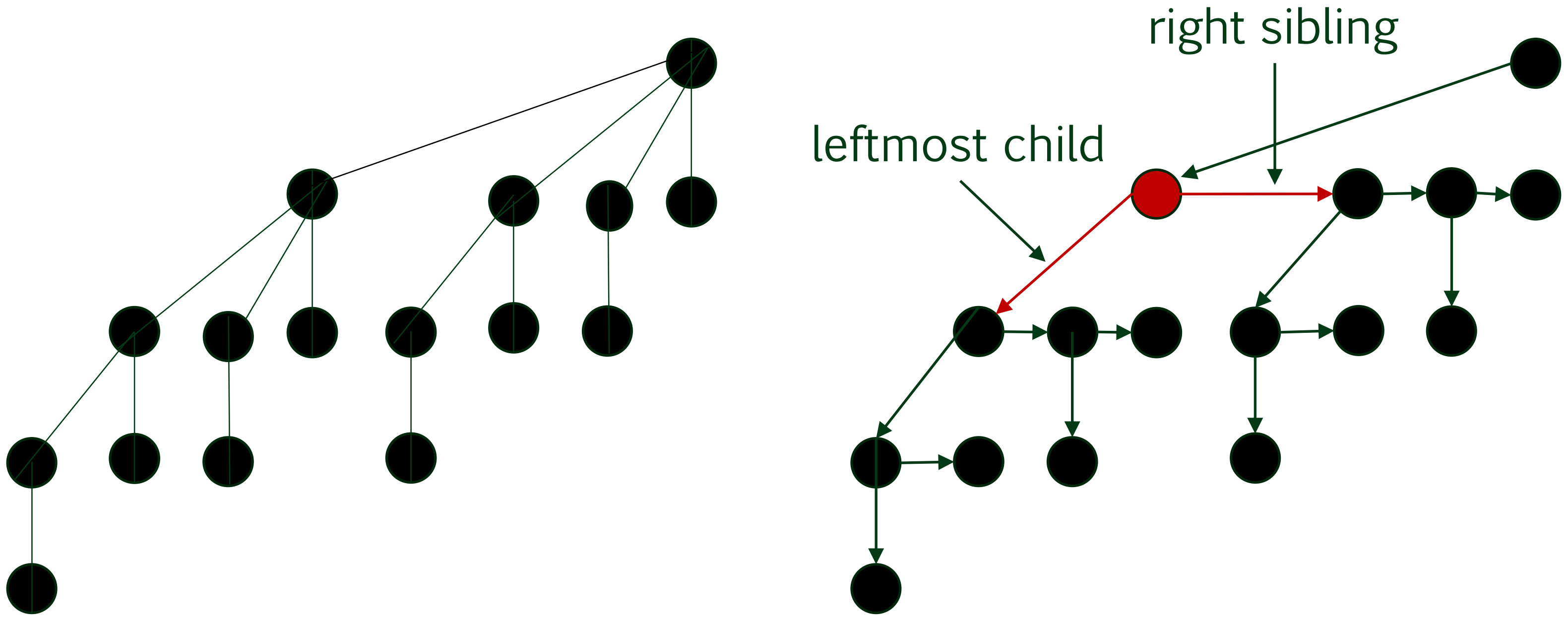
at most $\lfloor \log_2 n \rfloor + 1$

useful to know for time complexity analysis

n	Number of binomial trees
3	2
19	3
32	1
184	4
2525	8

Data structure

› Leftmost-child, right sibling (cf File Systems)



B_4 B_4

```
public class BinomialNode<T>
{
        public T Item                         { get; set; }
        public int Degree                     { get; set; }
        public BinomialNode<T> LeftMostChild { get; set; }
        public BinomialNode<T> RightSibling  { get; set; }
         ...
}
```

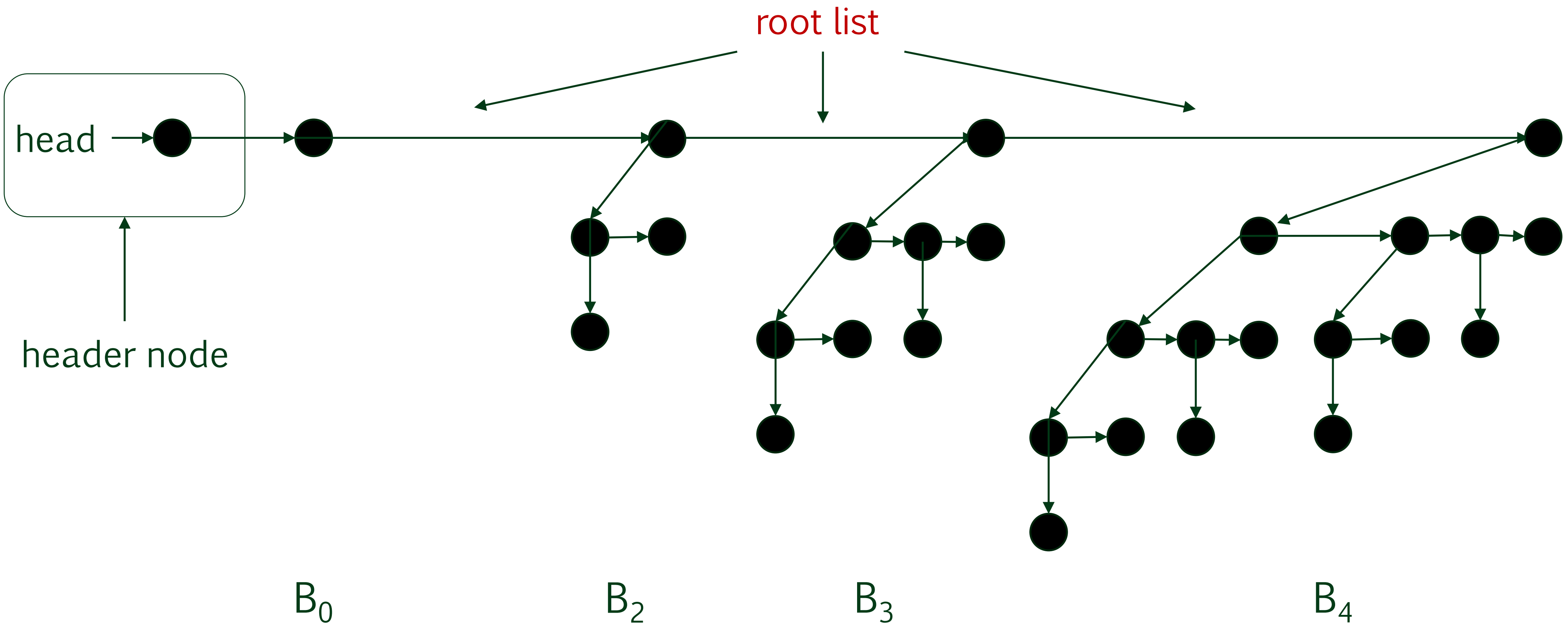
Exercises

- Explain why the merger of two binary heaps requires O(n) time.
- List the binomial trees B_k needed to represent $n = 3, 19, 32, 184$ and 2525 items.

Binomial Heap

- A binomial heap is a set of binomial trees that satisfies two properties:
  - There is at most one binomial tree B_k for each k
  - Each binomial tree is heap-ordered that is, the priority of the item at each node (except the root) is less than or equal to the priority of its parent (same as the binary heap)

Example of a binomial heap for n = 29

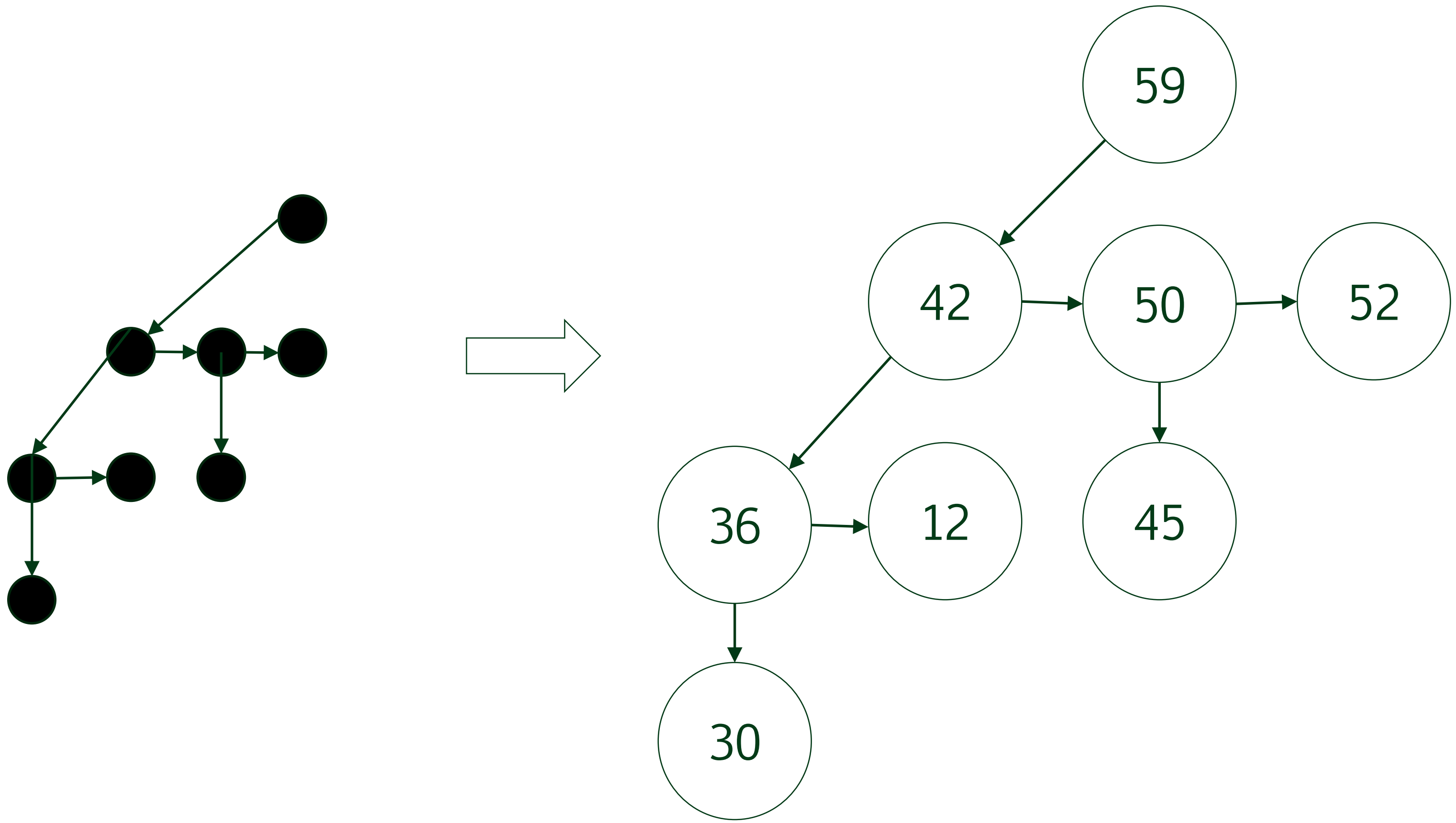


Notes:

1) The rightmost child of each root (except the last) refers to the next binomial tree
2) The binomial trees are order by increasing k

Example of a heap-ordered binomial tree

where higher-valued items have higher priorities



Note:

The priority of a parent must be greater than or equal to the priority of its leftmost child and its right siblings

Data structure

```
public class BinomialHeap<T> : IBinomialHeap<T> where T : IComparable
{
    private BinomialNode<T> head;  // Head of the root list
    private int size;              // Size of the binomial heap
    ...
}
```

Primary methods

```
// Adds an item to a binomial heap
void Add(T item)

// Removes the item with the highest priority
void Remove()

// Returns the item with the highest priority
T Front()

// Merges H with the current binomial heap
void Merge(BinomialHeap<T> H)
```

same as a binary heap

NEW

Supporting methods

```
// Returns the reference to the root preceding the highest priority item
private BinomialNode<T> FindHighest()

// Takes the union (without consolidation) of the current and given binomial heap H
private void Union(BinomialHeap<T> H)

// Consolidates (combines) binomial trees of the same degree
private void Consolidate()

// Merges the given binomial heap H into the current heap (used by Add and Remove)
public void Merge(BinomialHeap<T> H)
```

Let’s start with the private methods

FindHighest

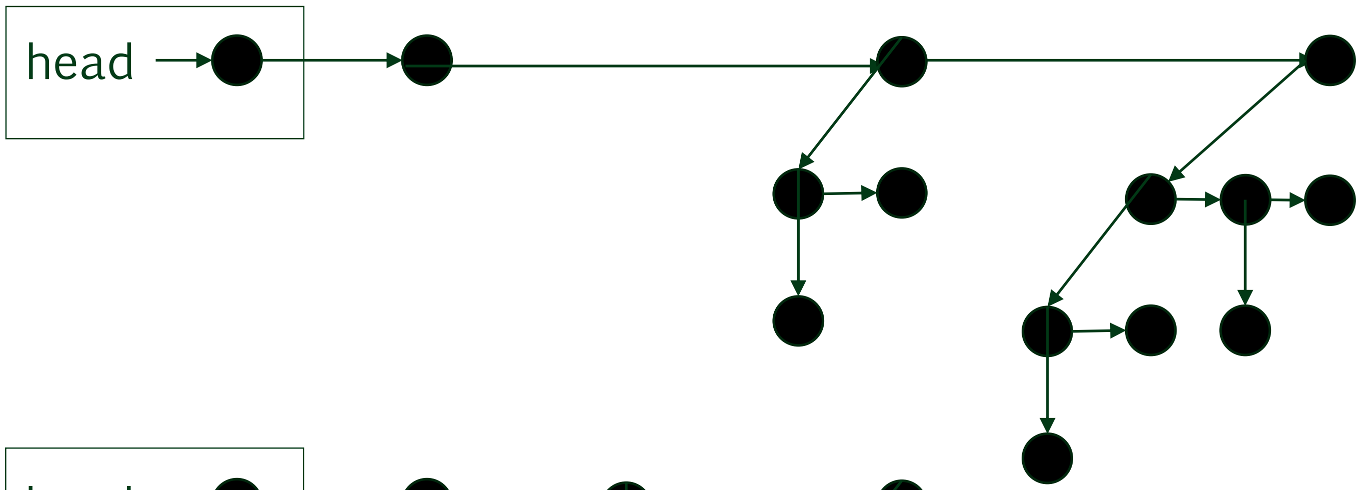
› Basic strategy

– Starting at the header node, traverse the root list and keep track of the item with the highest priority. Note: The item with the highest priority in the binomial heap will be found at the root of one of the binomial trees (Why?)

– Return the reference to the root node that precedes the one with the highest priority

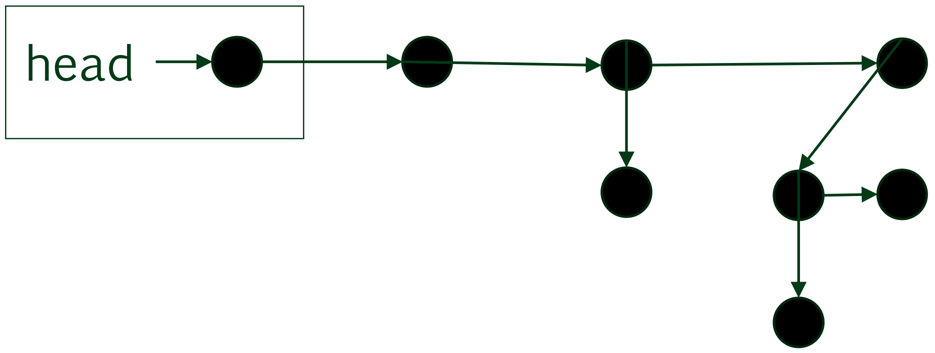
Union

- Basic strategy
  - Given two binomial heaps, merge the root lists into one, maintaining the order of the binomial trees. Note: The resultant list will have at most two binomial trees with root degrees of k (Why?)

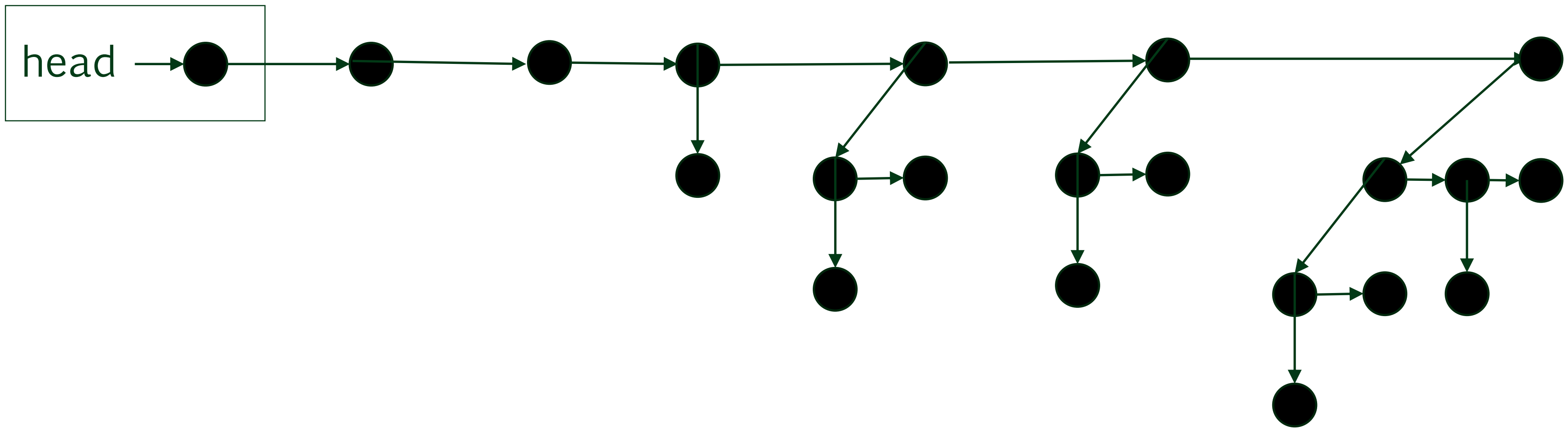


1101

+

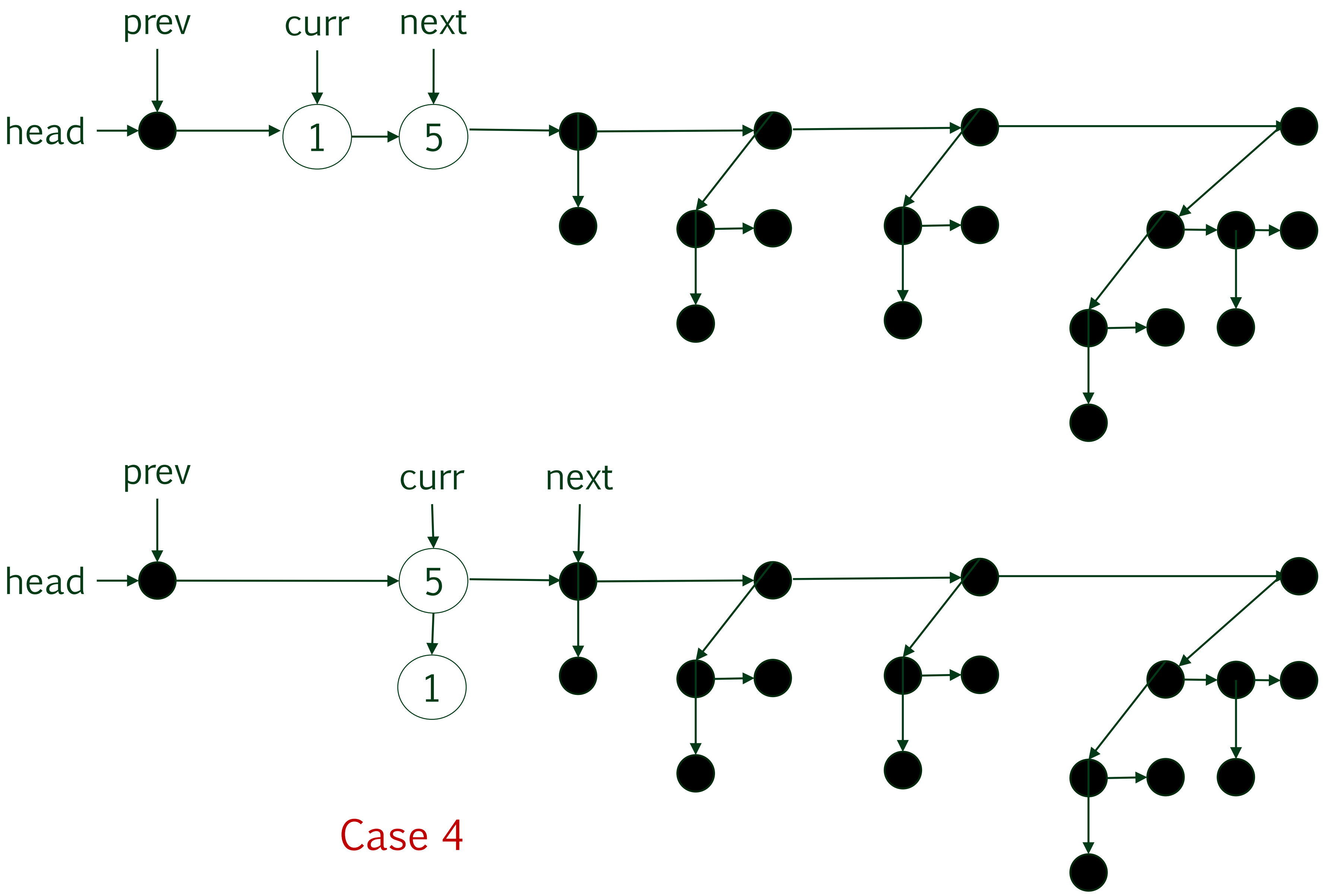


0111

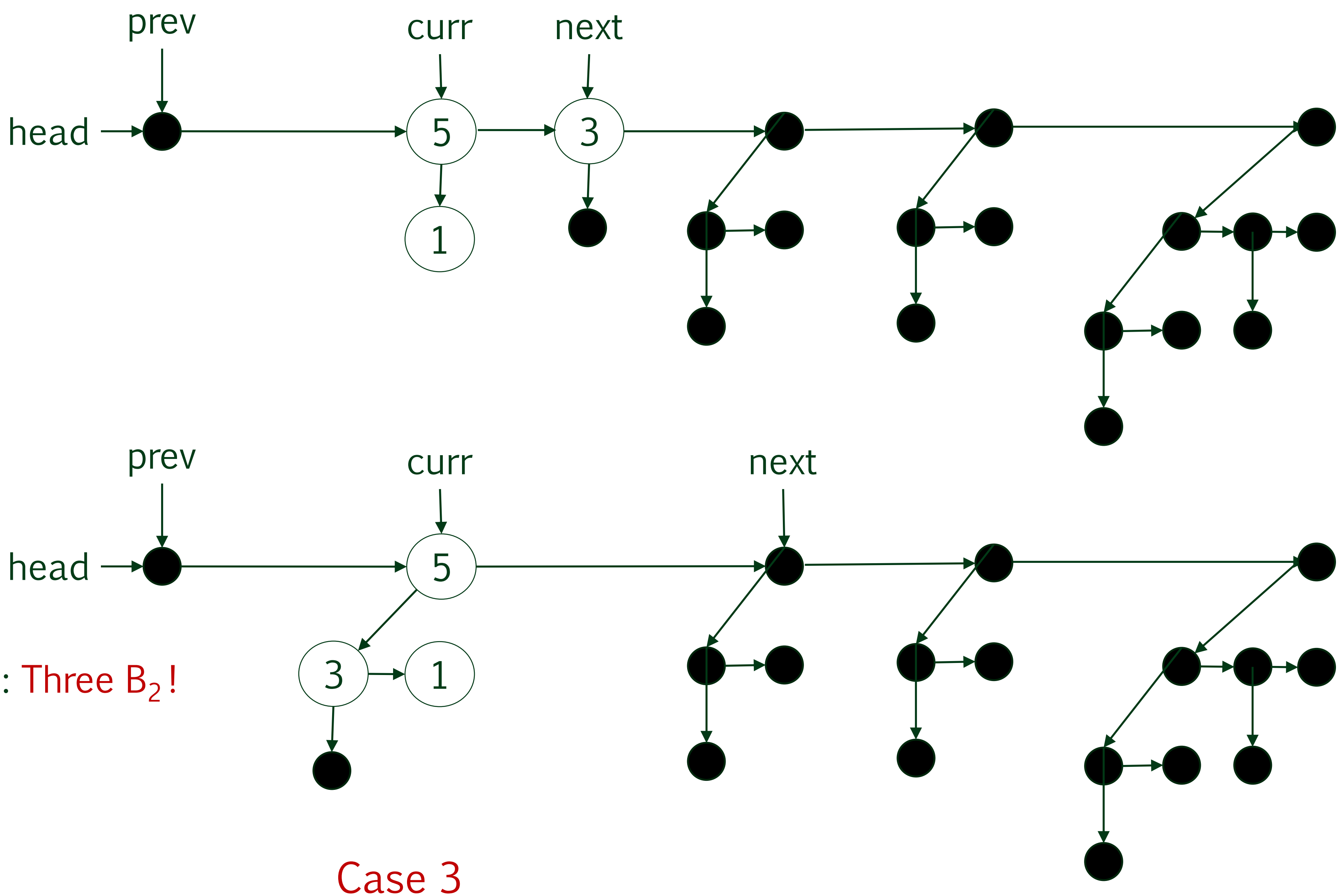


Consolidate

- Basic strategy
  - Run through a root list and combine two binomial trees that have the same root degree k into one binomial tree of degree k+1. Note: There may be (at least temporarily) three binomial trees of the same degree (Why?)

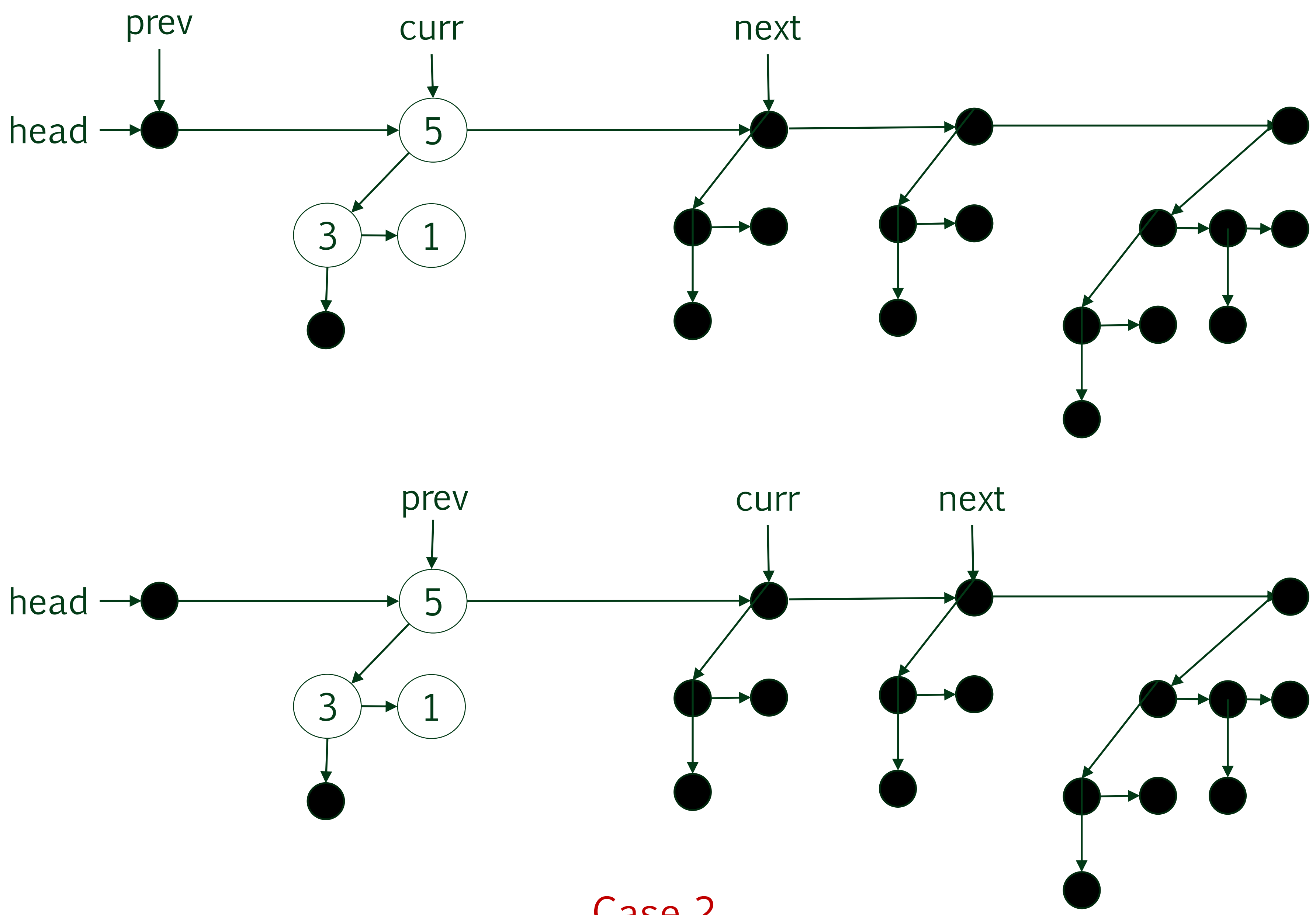


Case 4

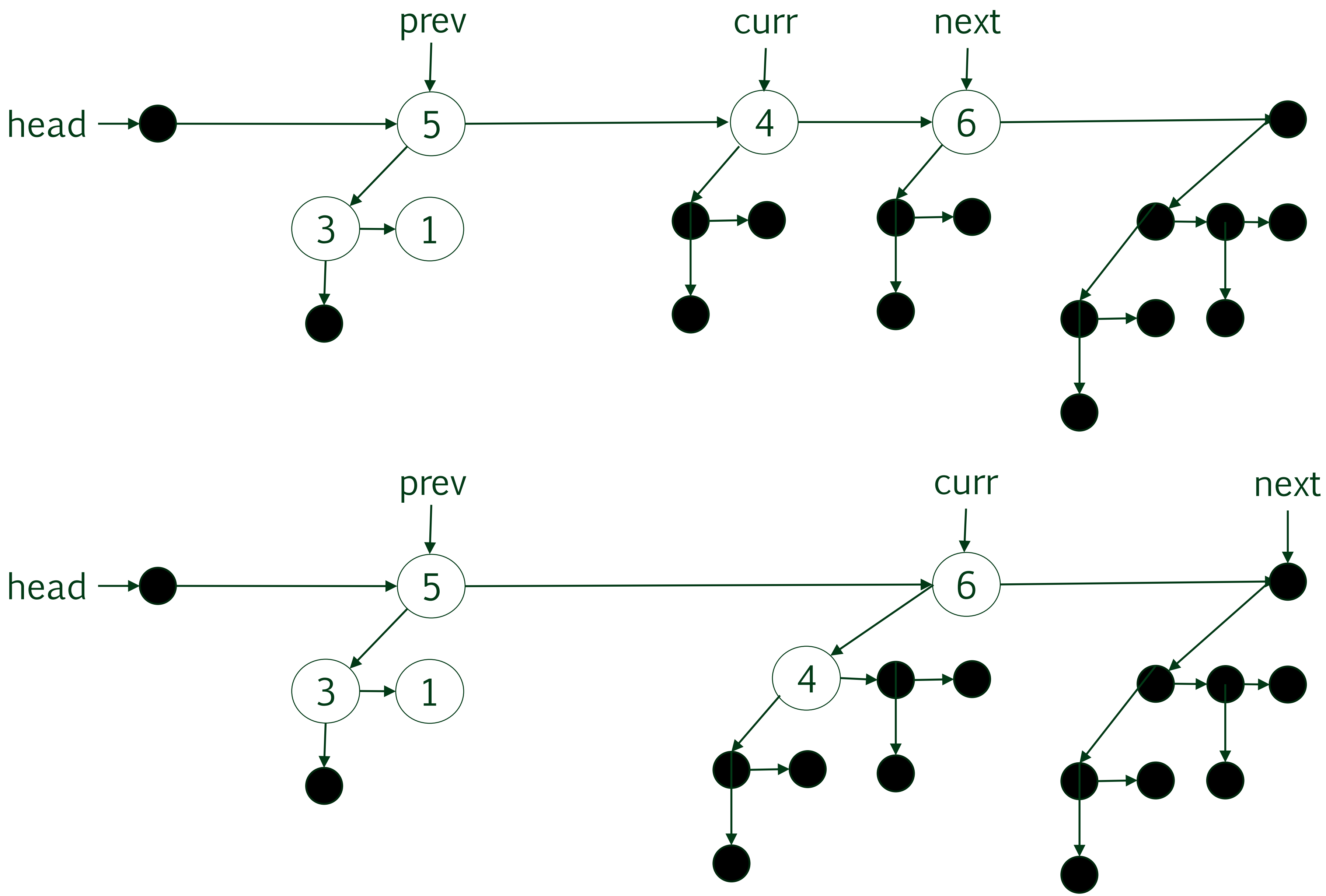


Note: Three B_2!

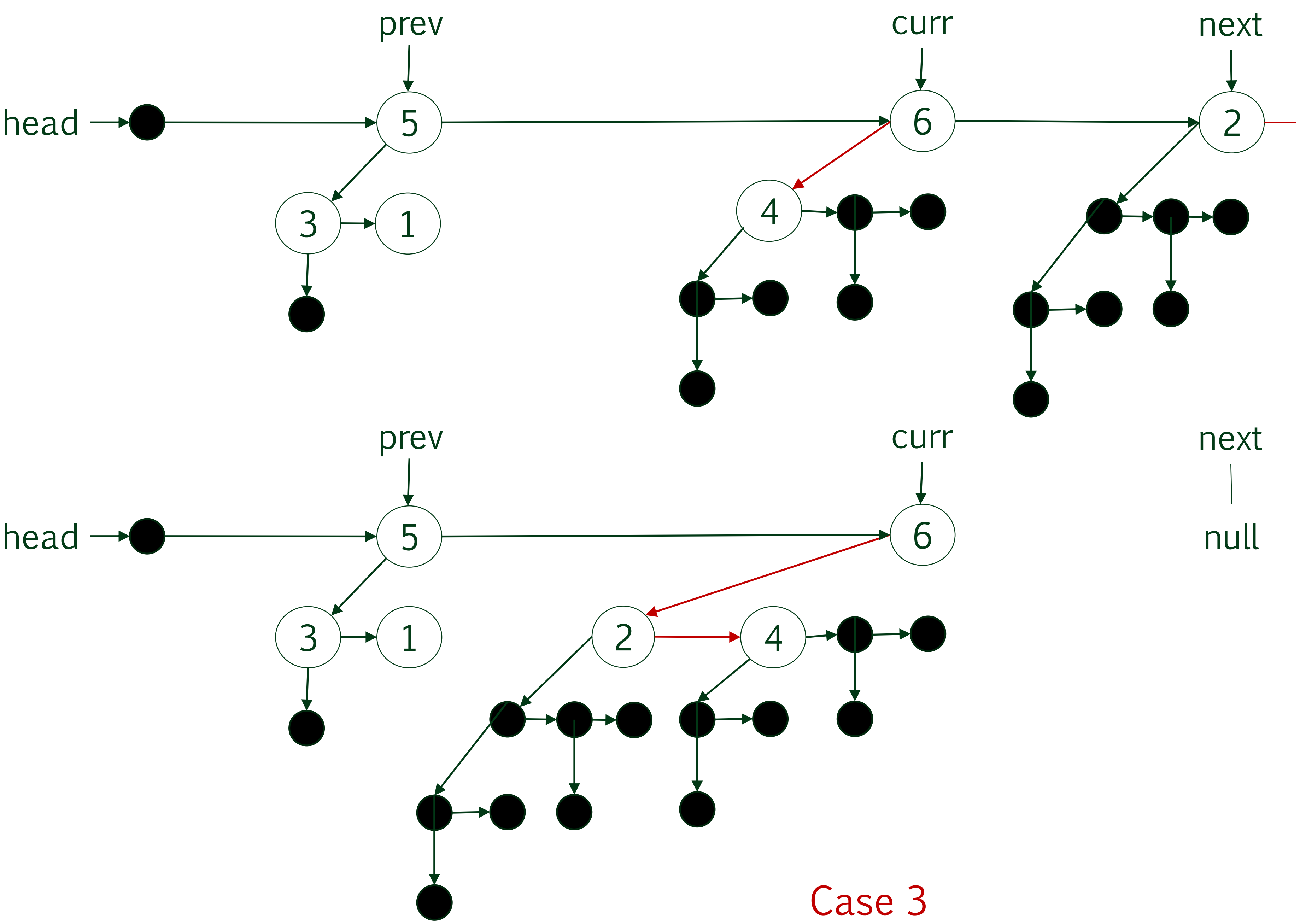
Case 3



Case 2

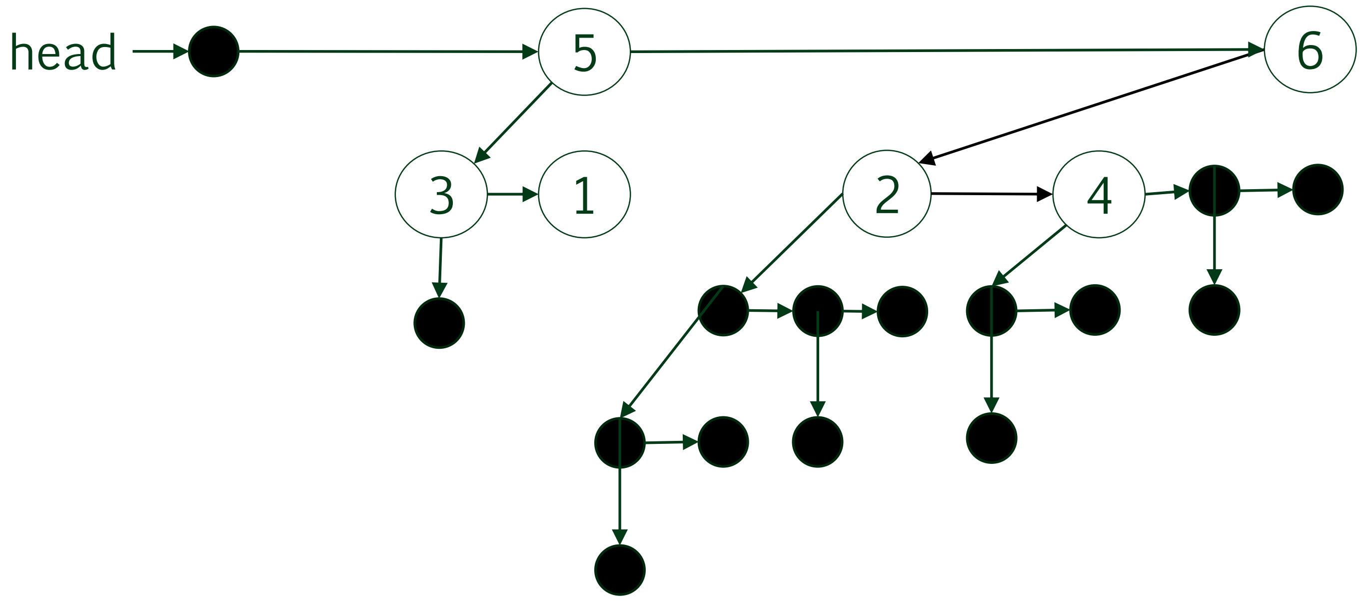


Case 4



Case 3

Final Result



$\boxed{1101}$

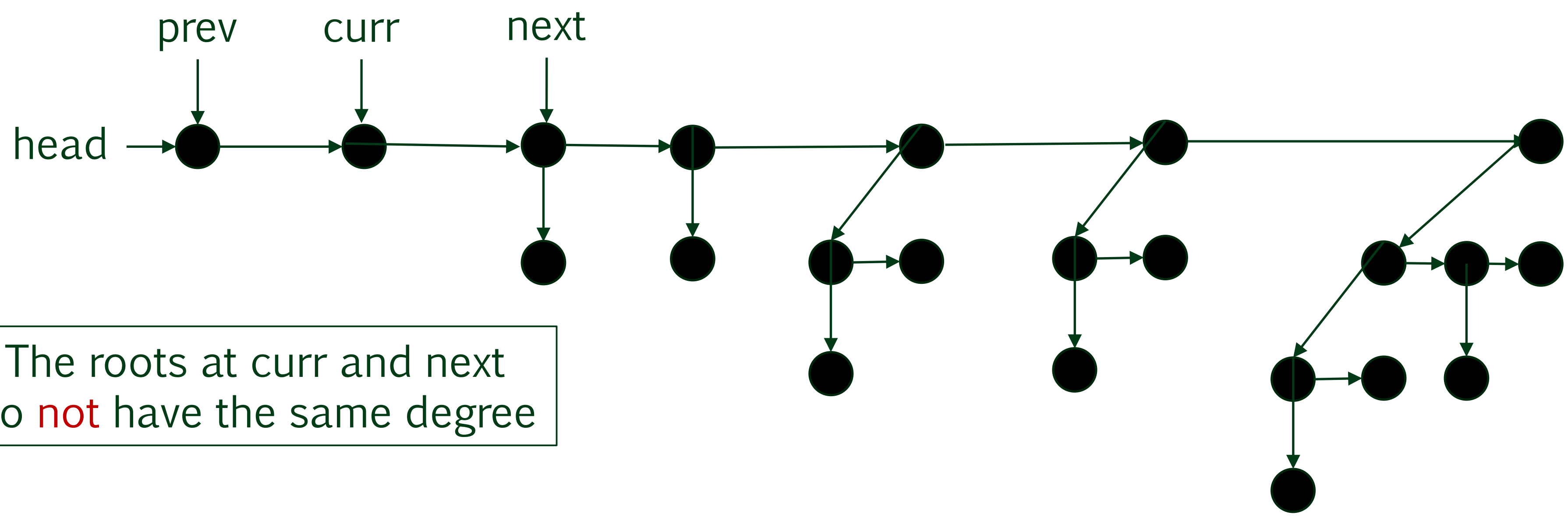
$+$

$\boxed{0111}$

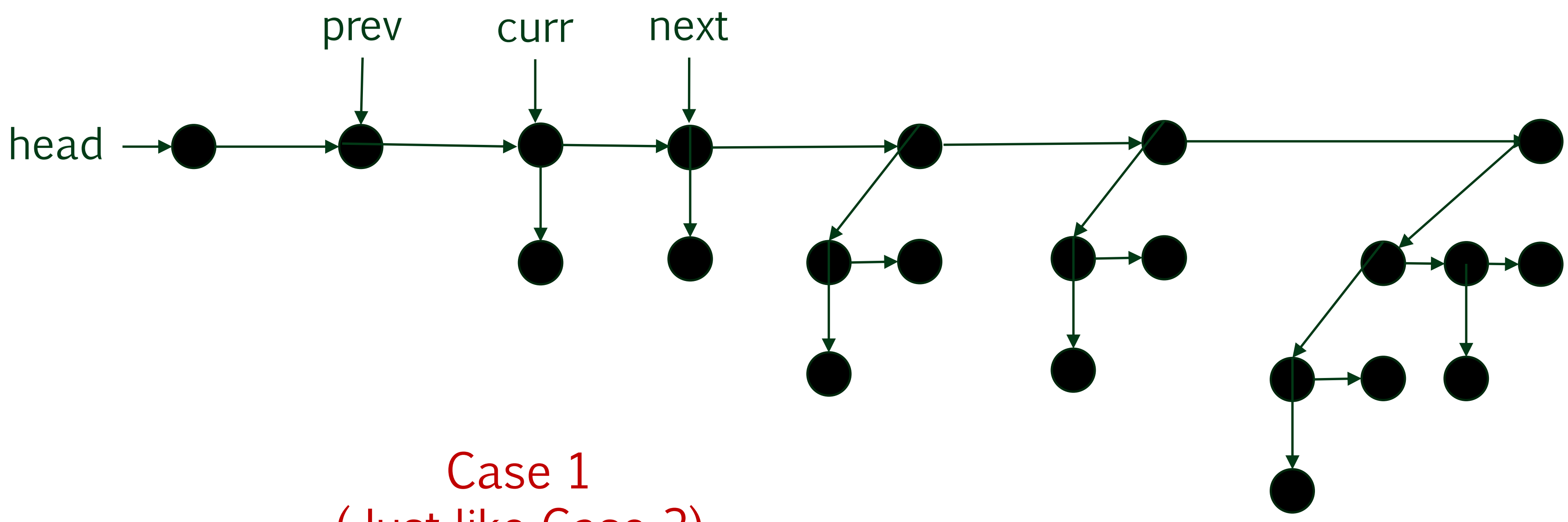
$=$

$\boxed{10100}$

What about Case 1?



The roots at curr and next
do not have the same degree



Case 1
(Just like Case 2)

Exercises

- Work through Union and Consolidate methods for the following pairs of binomial heaps. Only show the structure of the resultant binomial heap.

  - First heap: B0, B1, B2, B3 Second heap: B1, B3
  - First heap: B0, B1, B2 Second heap: B0, B1, B2

- Add items to each node of the binomial trees above and work through the Union and Consolidate methods.

- Suppose a binomial heap before consolidation has a maximum depth of k. Argue that the maximum depth of a binomial heap after consolidation is never more that k+1.

Now, on to the public methods

Merge

- Basic strategy
  - Take the union of the given binomial heap H with the current binomial heap and consolidate.

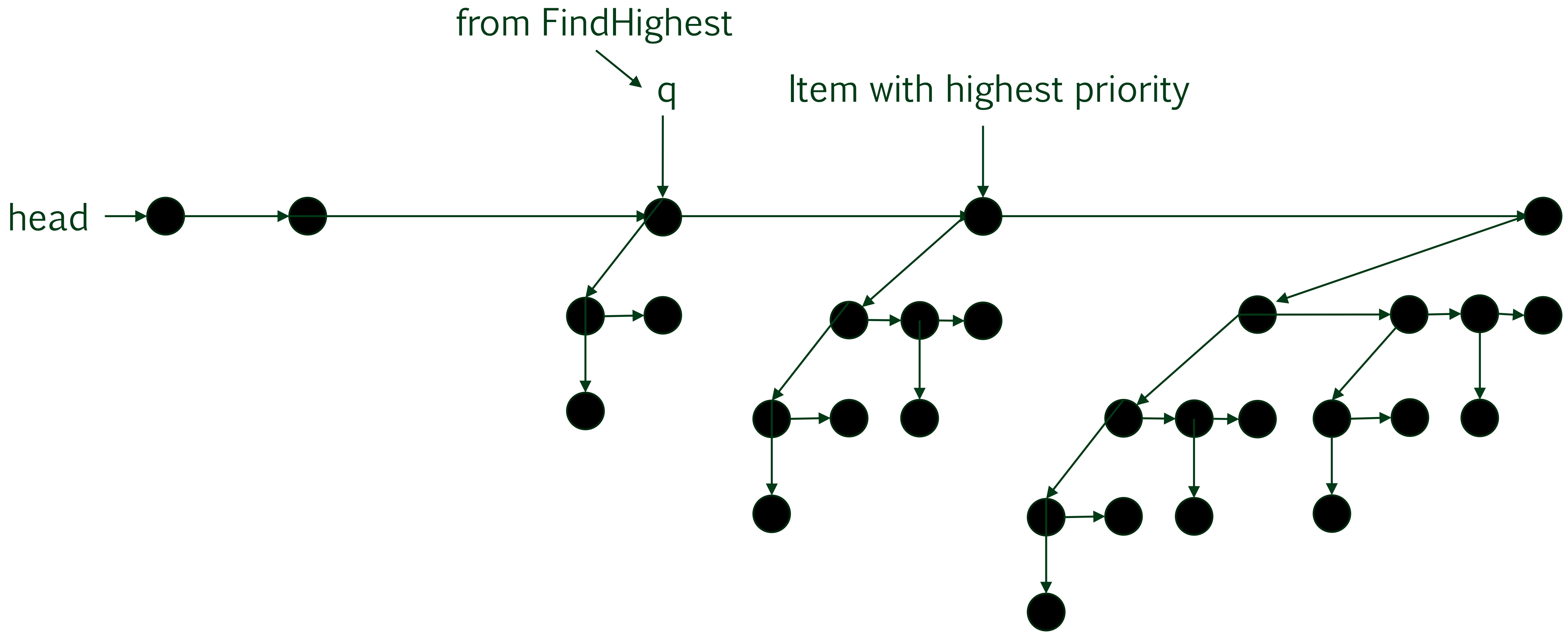
```
Union(H);
Consolidate( );
```

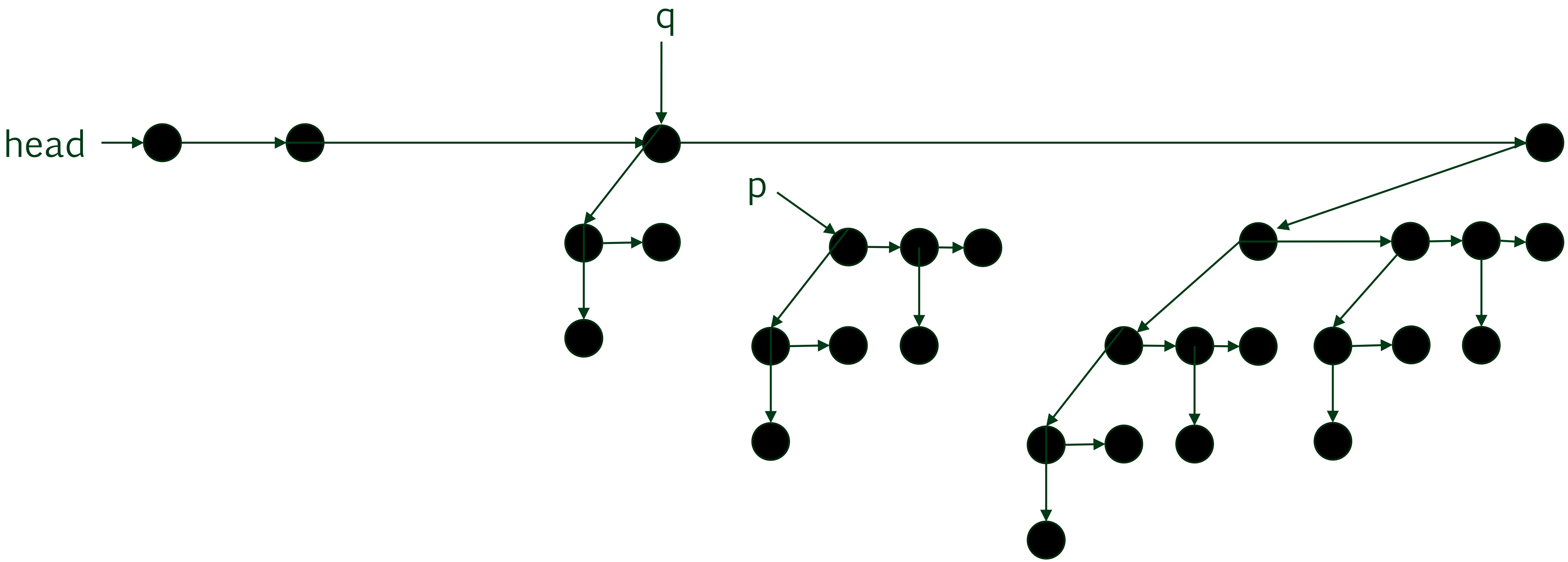
Insert

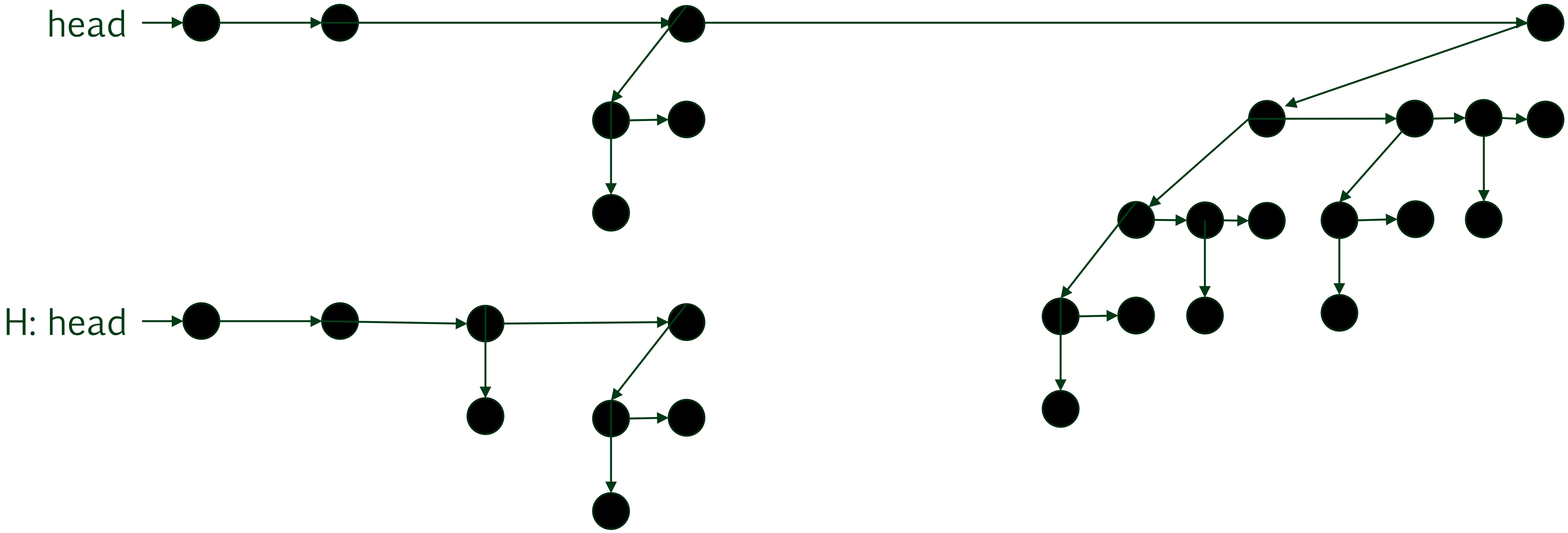
- Basic strategy
  - Create a binomial heap H with the given item (only).
  - Merge H with the current binomial heap and increase size by 1.

Remove

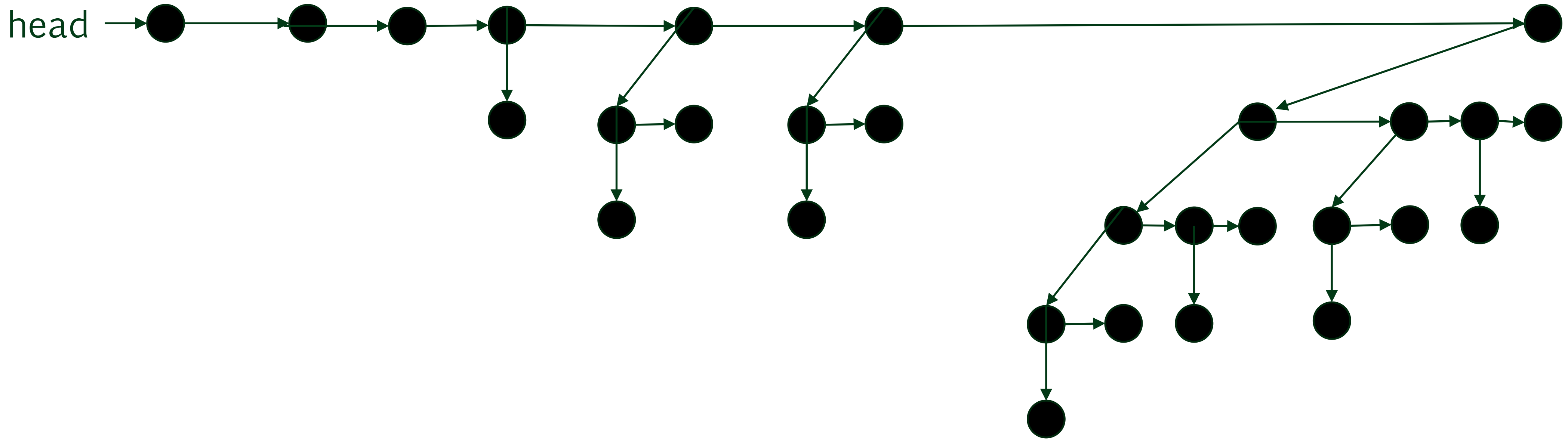
- Basic strategy
  - Call the method FindHighest and remove the next binomial tree T from the current root list.
  - Insert the children of T (in reverse order) into a new binomial heap H, effectively removing the item at the root.
  - Merge H (Union + Consolidate) with the current binomial heap and reduce size by 1.


from FindHighest
q
Item with highest priority
head

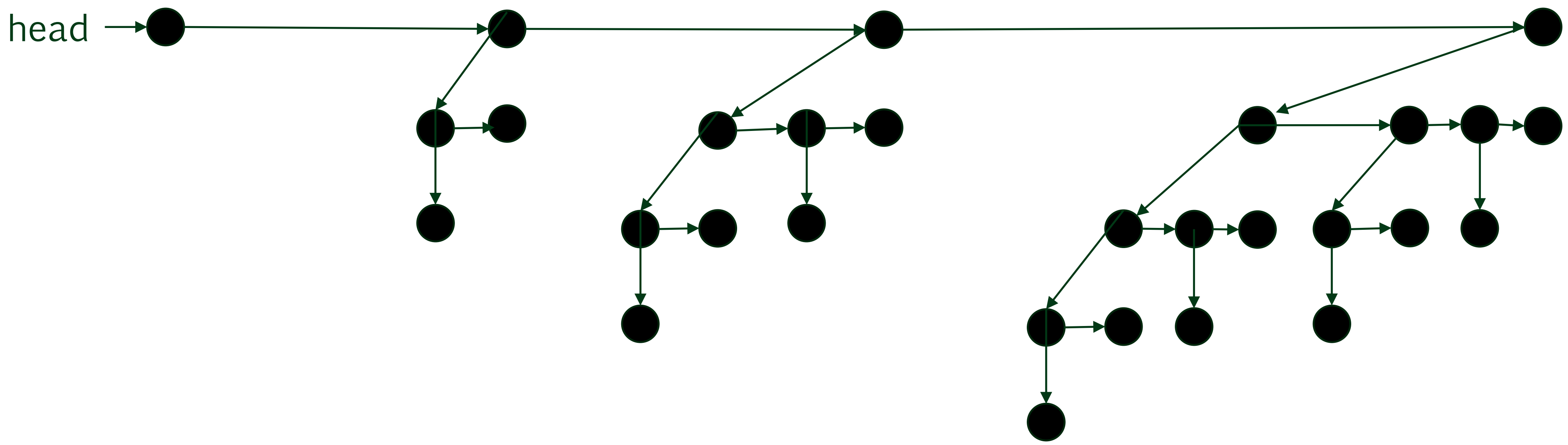

q
head
p


head
H: head

Union



Consolidation



Front

- Basic strategy
  - Call the method FindHighest and return the item at the root of the next binomial tree.

Exercises

- Work through the Insert method for n = 15 where the item to be inserted has the lowest priority. Where is the item in the final binomial heap?
- Work through the Remove method for n = 3, 19 and 31. When assigning items to the binomial heaps, place the item with the highest priority at the root of the last binomial tree.

Time complexities

Method	Binary Heap	Binomial Heap *
Insert	O(log n)	O(log n)
Remove	O(log n)	O(log n)
Front	O(1)	O(log n)
Merge	O(n)	O(log n)
FindHighest		O(log n)
Union		O(log n)
Consolidate		O(log n)

* All time complexities depend on the observation that the maximum length of the root list is O(log n)