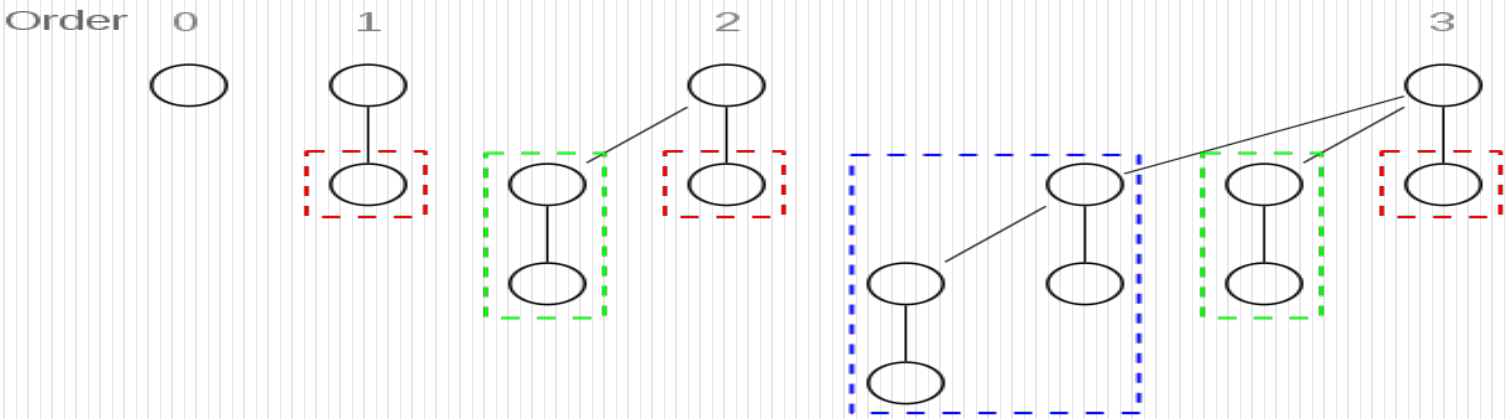
Binomial heaps

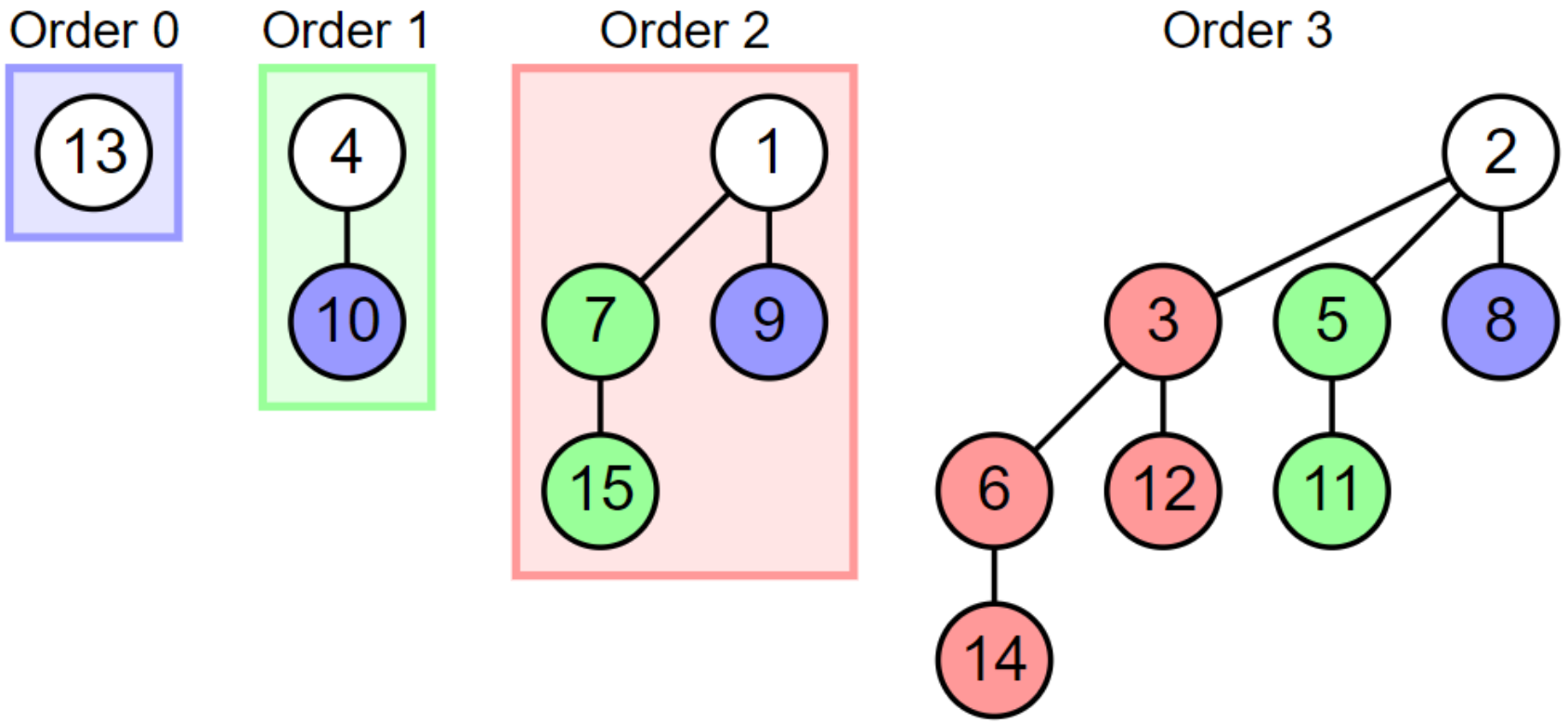


Dr. Bibhudatta Sahoo

CS215, Department of CSE, NIT Rourkela

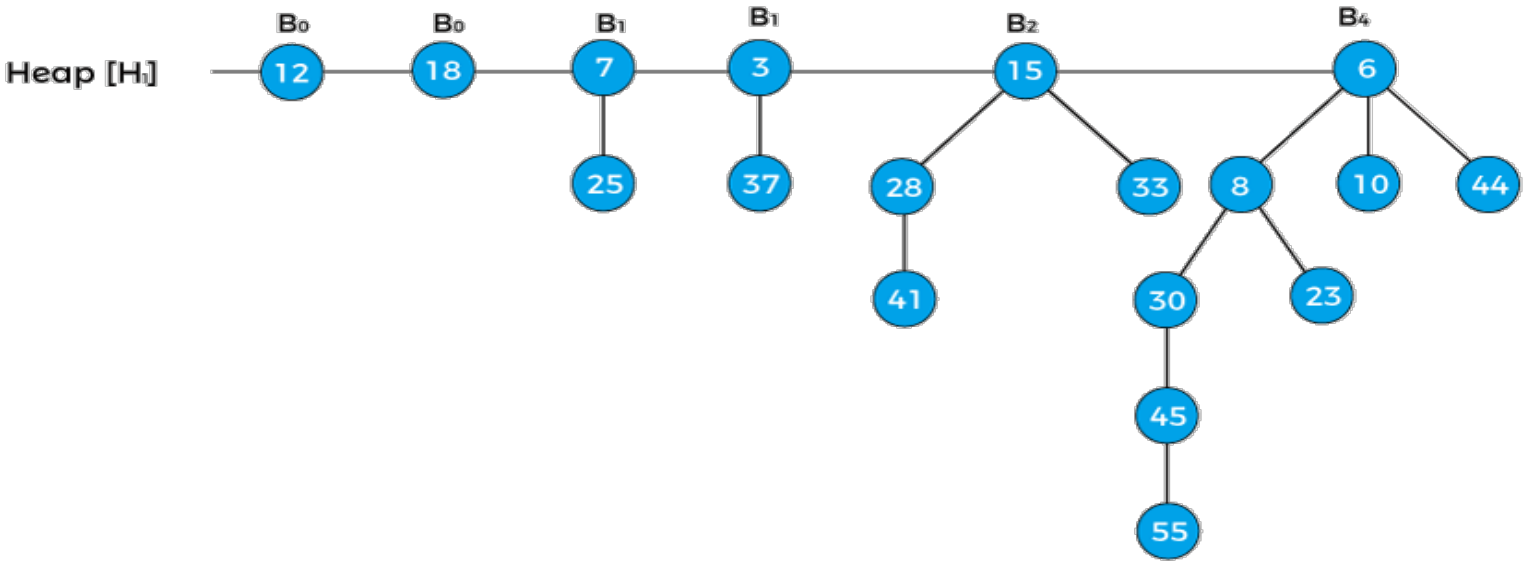
Email: bdsahu@nitrkl.ac.in, 9937324437, 2462358

Binomial heap(Min heap)



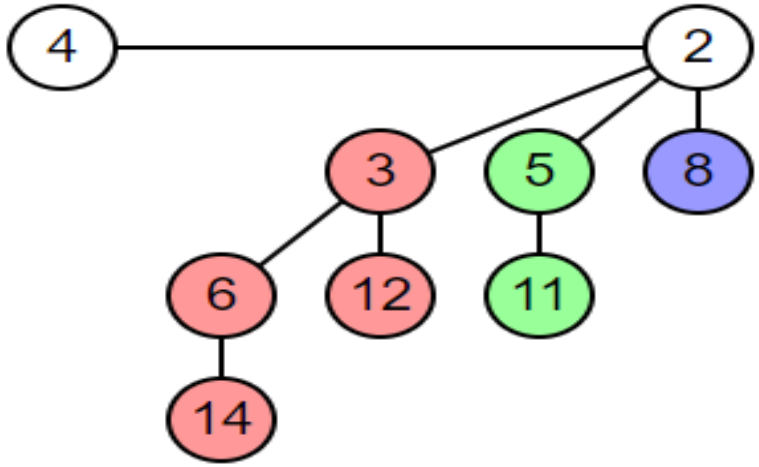
Binomial heap

- A **Binomial Heap** is a specific kind of heap data structure that merges properties of both **binary** and **binomial trees** to efficiently support a set of operations, including insertion, deletion, and merging of heaps.
- Due to their efficient merging and structured properties, binomial heaps are suitable for applications like network routing, spanning tree algorithms, and any problem where frequent merging of priority queues is required.



Binomial heap

- A binomial heap is made of a series of binomial trees each of which have a unique order.
- The order of a binomial tree defines how many elements it can contain, namely 2^{order}. Each tree of the order ***x*** is constructed by linking trees of the order together.
- The binomial heap can be either a **min heap** or a **max heap**.
- binomial heap also follows the properties of the heap data structure; all nodes must be smaller than their children for a min heap, or larger for a max heap.



Binomial heap

Key Characteristics of Binomial Heaps

1. **Collection of Binomial Trees**: A binomial heap is composed of multiple binomial trees, each of which has a specific order. A binomial tree of order k has 2^k nodes and consists of a root node with k child trees, each of orders $k-1, k-2, \ldots, 0$.

2. **Min-Heap or Max-Heap Property**: Binomial heaps generally maintain a min-heap or max-heap property. In a min-heap, the key of a node is less than or equal to the keys of its children, ensuring that the root node of each binomial tree contains the minimum key value.

3. **Structure of Trees**: The structure of the trees allows for efficient merging operations. Each binomial tree is structured such that it has one node as a root and child trees that follow the order mentioned above, which allows binomial heaps to store data efficiently.

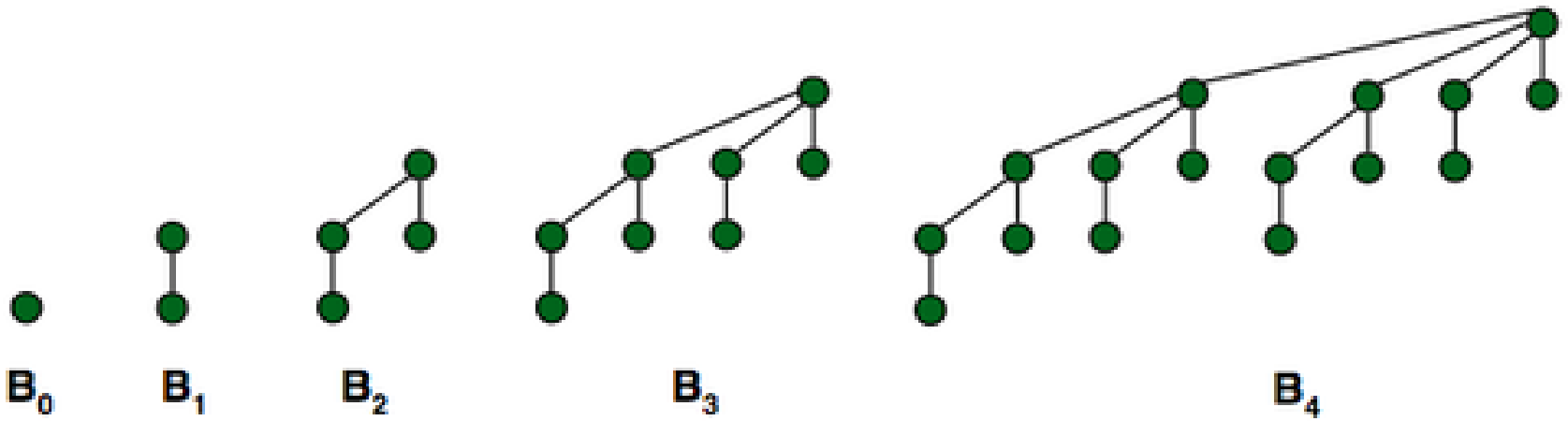
Binomial Tree

- A **Binomial Tree** is a specific type of ordered tree that has unique structural properties and forms the foundation of binomial heaps.
- Each binomial tree has an order, defined by the number of nodes it contains, and follows a recursive structure based on the **binomial coefficient**.

Key Characteristics of a Binomial Tree

1. **Order**: A binomial tree of order k, denoted B_k, has 2^k nodes.
2. **Recursive Structure**: A binomial tree of order k can be formed by linking two binomial trees of order $k-1$, with the root of one tree becoming the leftmost child of the root of the other tree. This recursive linking makes the structure efficient for operations in a binomial heap.
3. **Height**: The height of a binomial tree of order k is k, meaning a binomial tree of order 0 is a single node, an order 1 tree has one level of depth, and so on.

Binomial Tree

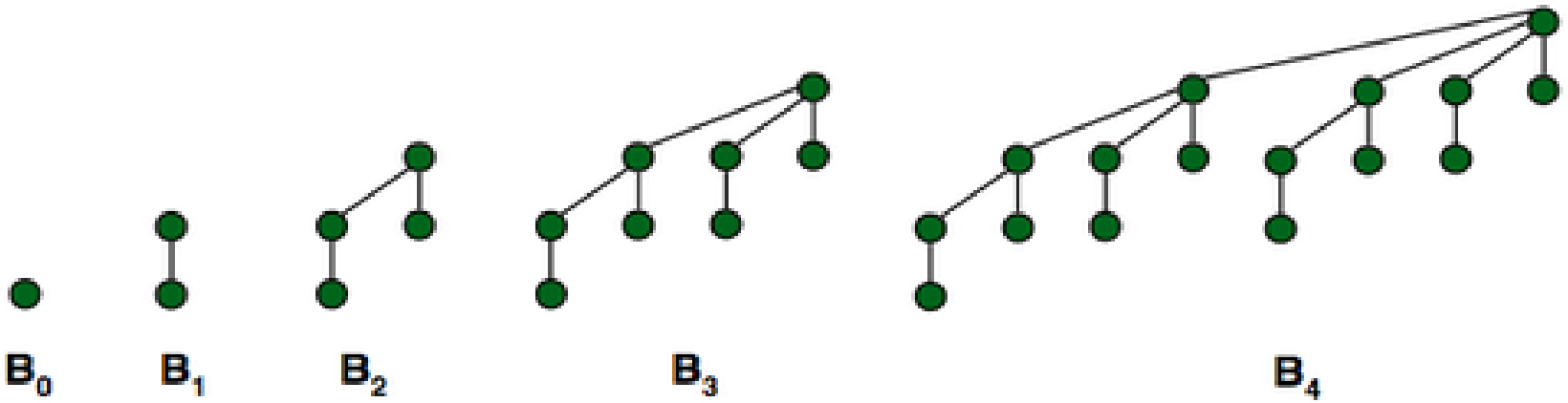


Key Characteristics of a Binomial Tree

1. **Order**: A binomial tree of order k, denoted B_k, has 2^k nodes.
2. **Recursive Structure**: A binomial tree of order k can be formed by linking two binomial trees of order $k-1$, with the root of one tree becoming the leftmost child of the root of the other tree. This recursive linking makes the structure efficient for operations in a binomial heap.
3. **Height**: The height of a binomial tree of order k is k, meaning a binomial tree of order 0 is a single node, an order 1 tree has one level of depth, and so on.

Binomial Tree

- A binomial tree is a recursive data structure: a tree of degree zero is just a single node and a tree of degree k is two trees of degree k-1, connected.
- A tree of degree 1 is just **two nodes**, i.e., two trees of degree 0.
- A tree of degree 2 is **four nodes**, i.e., two trees of degree 1 (or two trees of two trees of degree zero = four nodes).
- A tree of degree 3…

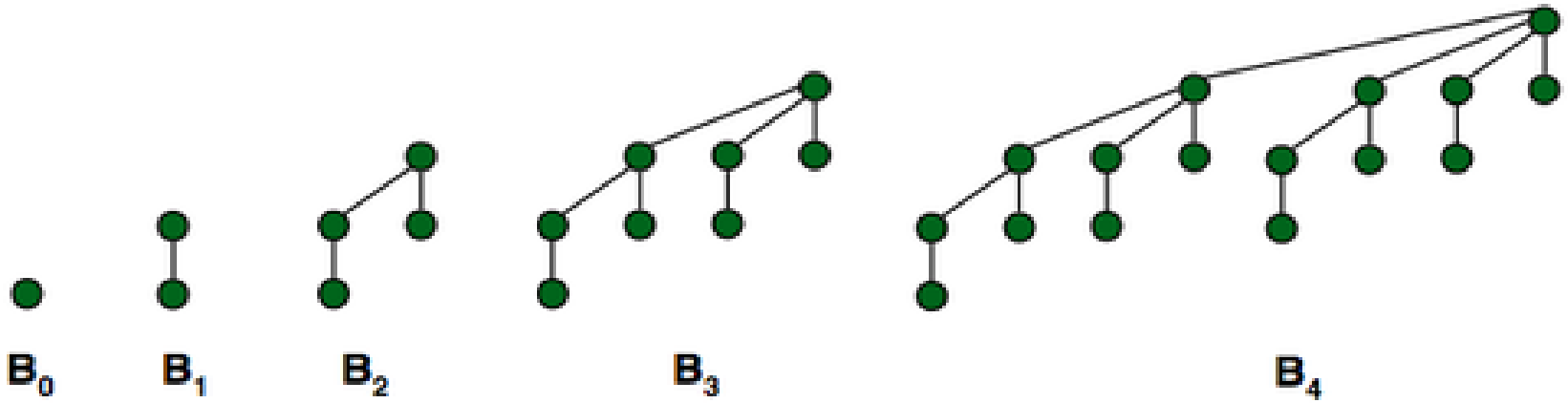


Properties of Binomial Trees

- **Node Arrangement**: In a binomial tree of order k, the root node has k children, each of which is the root of a binomial tree with decreasing orders from $k-1$ to 0.
- **Shape and Balance**: Binomial trees are balanced, meaning that all paths from the root to the leaves have a similar length.
- **Binary Representation of Orders**: Binomial trees are useful in heaps because a binomial heap can be represented as a collection of binomial trees, with each tree order corresponding to the binary representation of the number of nodes.

Binomial heap

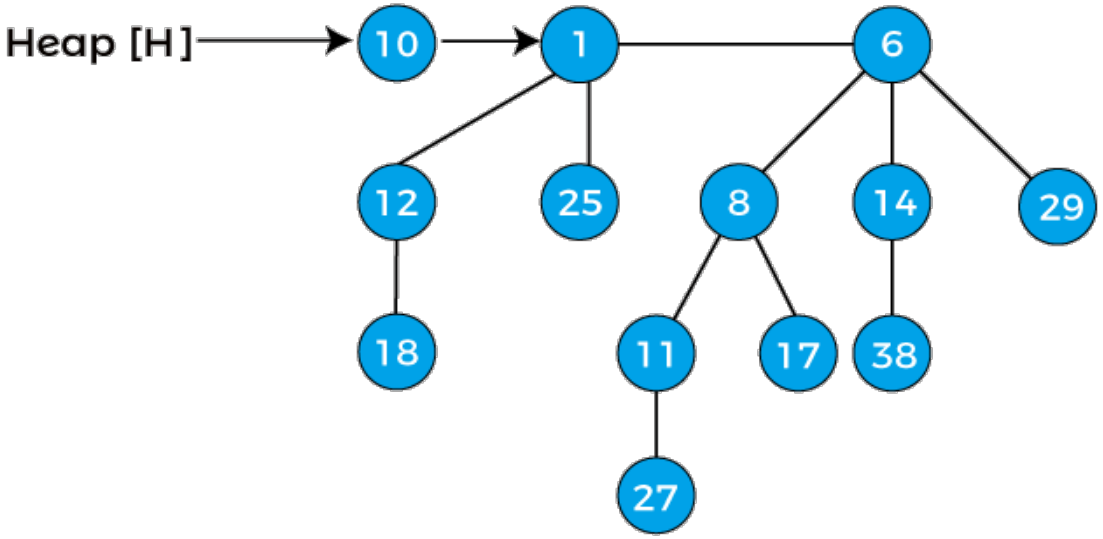
- A binomial *heap* is made up of a list of binomial *trees*.



In Binomial heaps, one can perform an insert and a combine in O (1) actual and amortized time and a delete min in O(log n) amortized time.

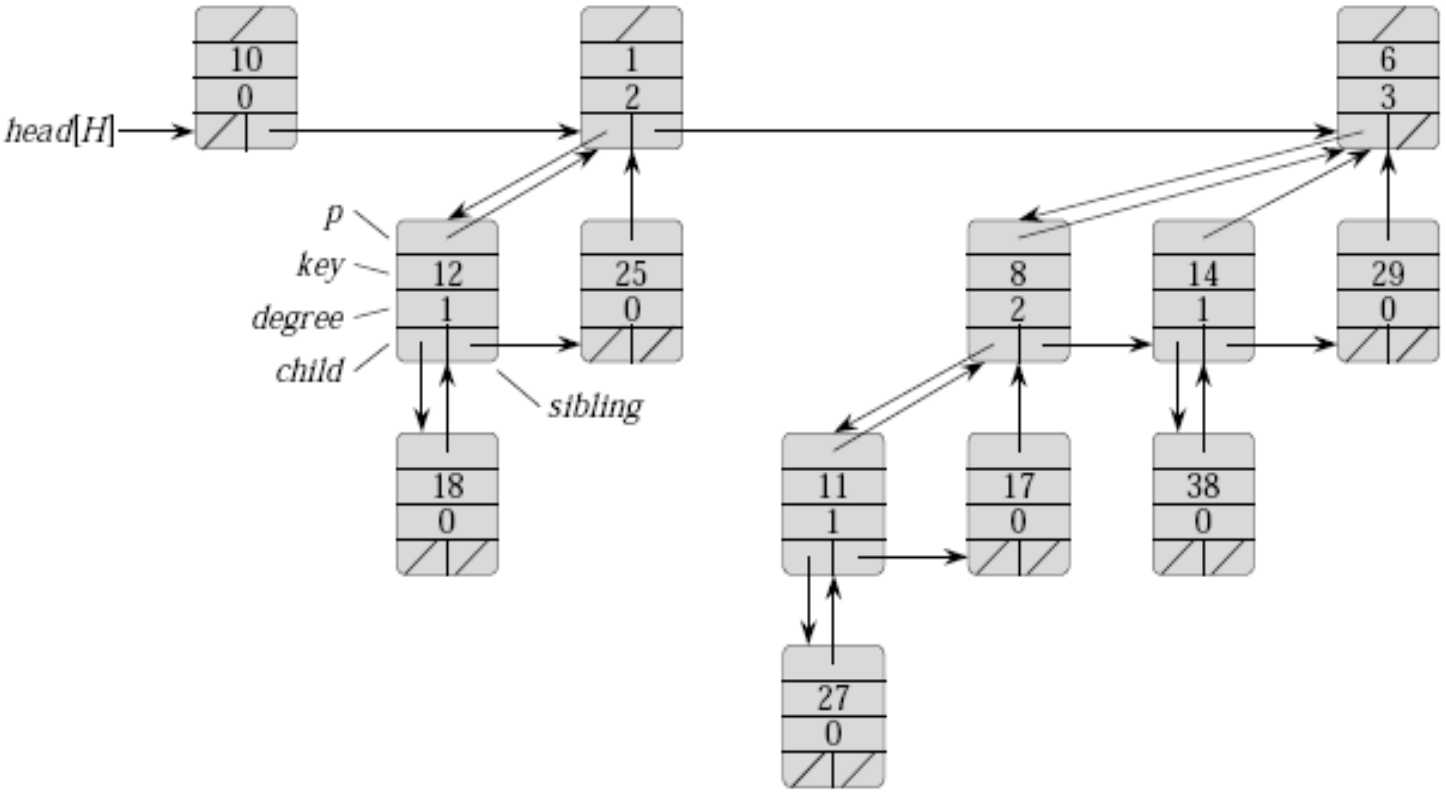
Properties of Binomial heap with n nodes

- Every binomial tree in the heap must follow the **min-heap** property, i.e., the key of a node is greater than or equal to the key of its parent.
- For any non-negative integer k, there should be at least one binomial tree in a heap where root has degree k.
- The first property of the heap ensures that the min-heap property is hold throughout the heap. Whereas the second property listed above ensures that a binary tree with **n** nodes should have at most **1 + $\log_2$ n** binomial trees, here **$\log_2$** is the binary logarithm.



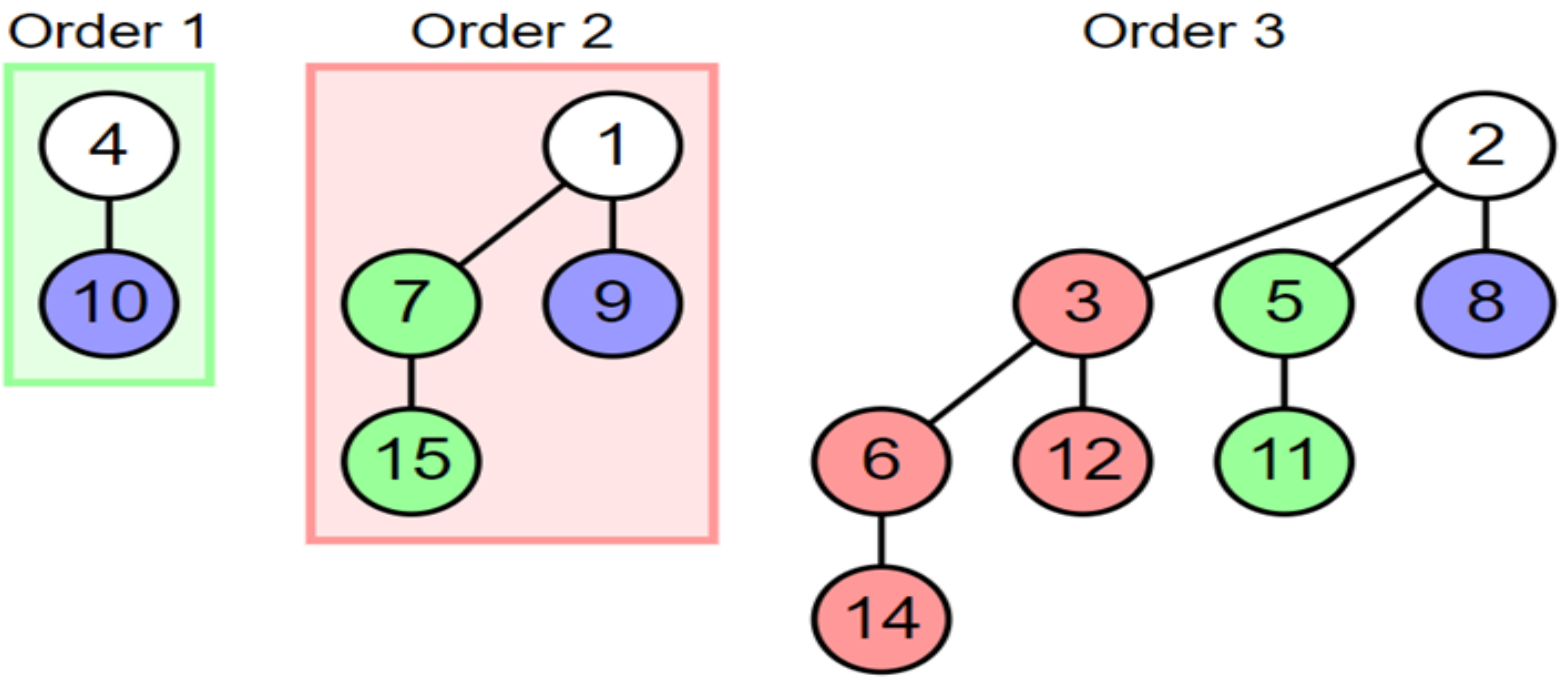
Binomial heap(Min heap)

- The main application of **Binary Heap** is as implement a **priority queue.**
- Binomial Heap is an extension of Binary Heap that provides faster union or merge operation with other operations provided by Binary Heap



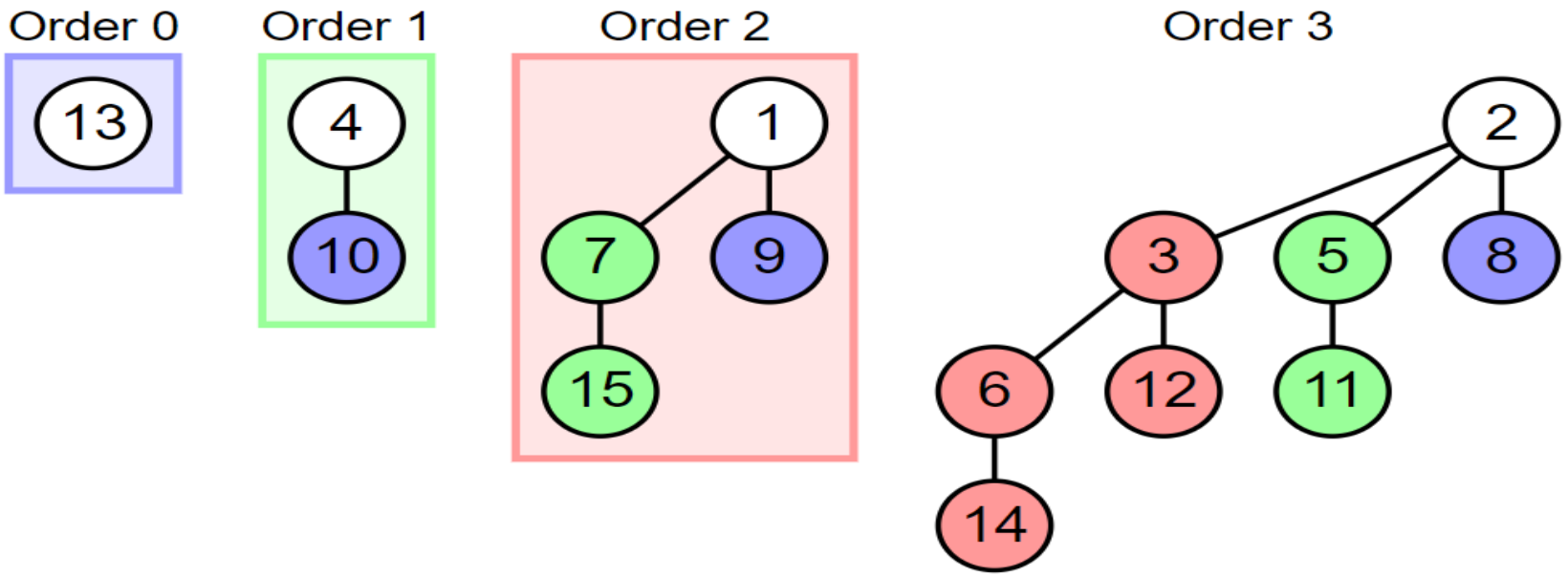
Binomial heap with 14 elements

- A binomial heap with **14 elements** will have binomial trees of the order 1, 2, and 3, which are in the same positions as the number 14 in binary '**1110**'.
- 1 Tree of order 3, 1 Tree of order 2, 1 Tree of order 1, No tree of order 0.



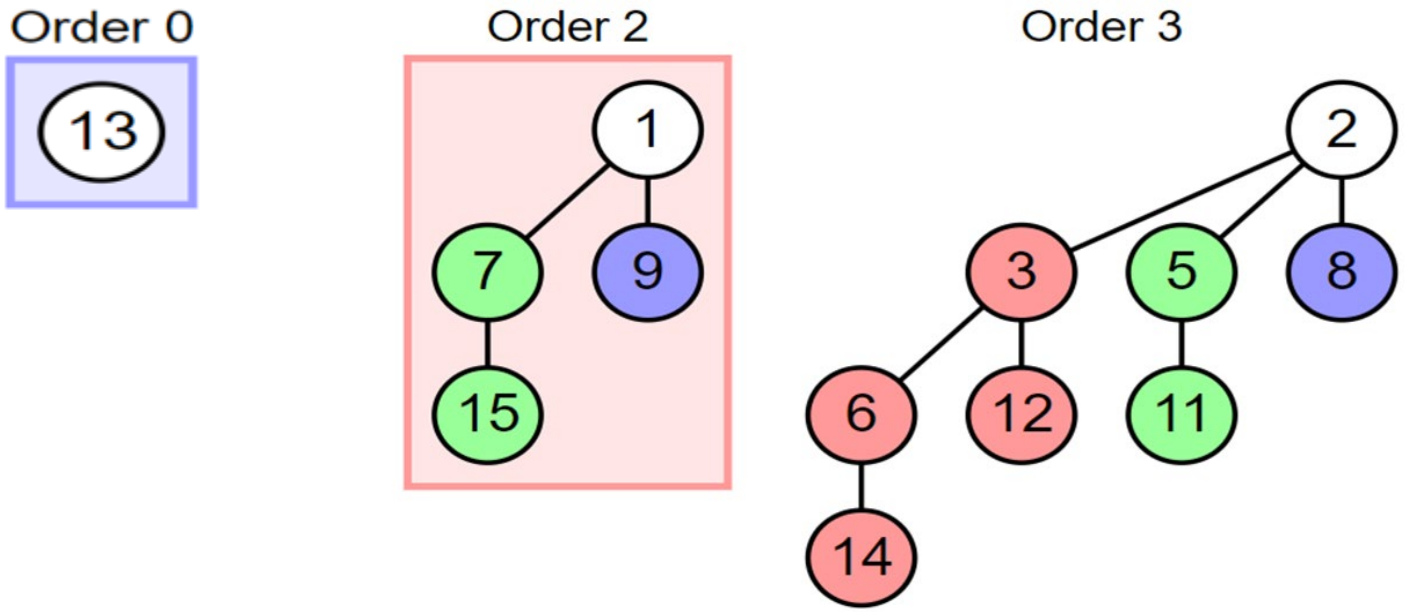
Binomial heap with 15 elements

- A binomial heap with **15 elements** will have binomial trees of the order 1, 2, and 3, which are in the same positions as the number 15 in binary '1111'.
- 1 Tree of order 3, 1 Tree of order 2, 1 Tree of order 1, 1tree of order 0.



Binomial heap with 13 elements

- If we have a heap with **13** items, we can express this in binary as 1101.
- This would translate to a binary tree of degree 3, a tree of degree 2, and a tree of degree 0 (with $2^3 + 2^2 + 2^0 = 8 + 4 + 1$ items respectively = 13 total items).
- 1 Tree of order 3, 1 Tree of order 2, and 1tree of order 0



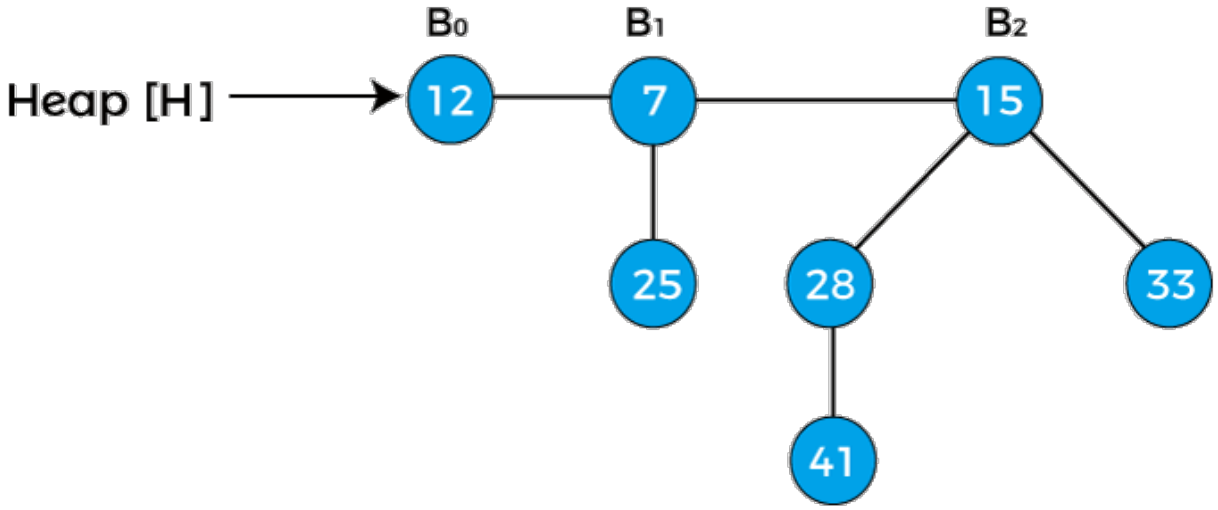
Operations on Binomial Heap ADT

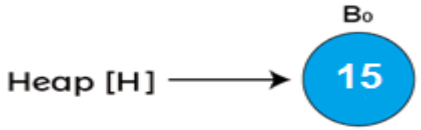
1. **Make-Heap**:
   - **Description**: Creates and returns a new, empty binomial heap.
   - **Complexity**: $O(1)$.
2. **Insert(heap, key)**:
   - **Description**: Inserts a new element with a specified key into the binomial heap.
   - **Procedure**: Creates a new binomial heap containing a single node with the given key, then merges this single-node heap with the existing heap.
   - **Complexity**: $O(\log n)$, where n is the number of nodes in the heap.

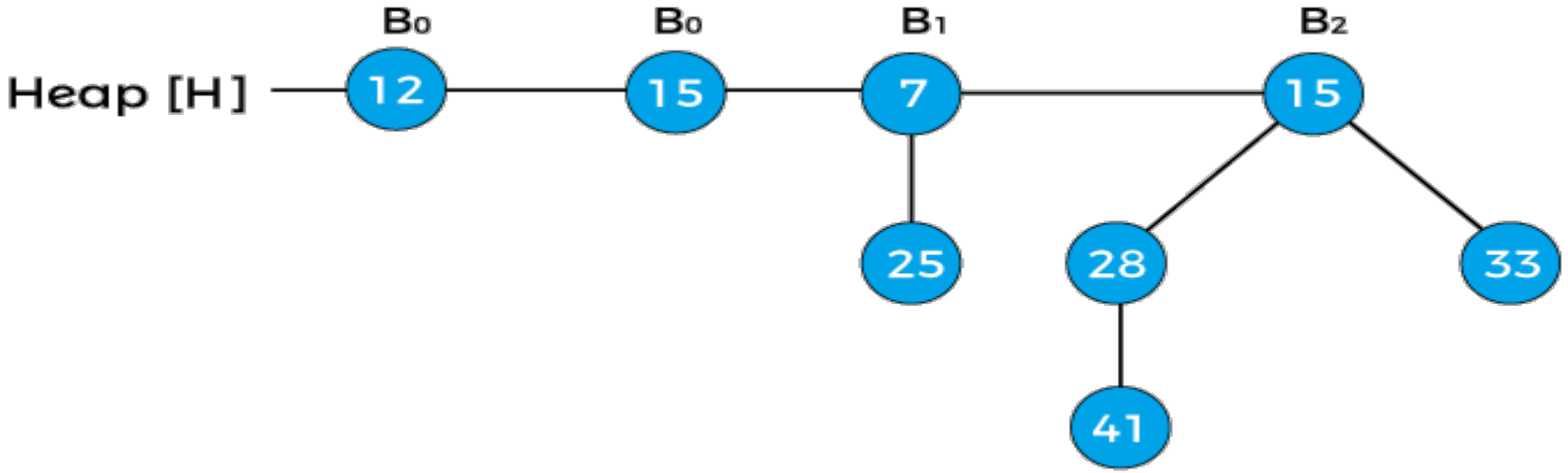
Insert to create Heap

1. If the heap doesn't have a rank *0* tree, then directly insert the new singleton tree (with rank *0*) to the head of the list.
2. If the heap has a rank *0* tree, then the two rank *0* tree need to be linked and promoted to a new rank *1* tree. Then continue to try to insert the rank *1* tree with the rest of the list that potential starts with a existing rank *1* tree.
3. If there is already a rank *1* tree, then link and promoted to rank *2*… so on so forth, until the newly promoted tree has a slot to fit in.

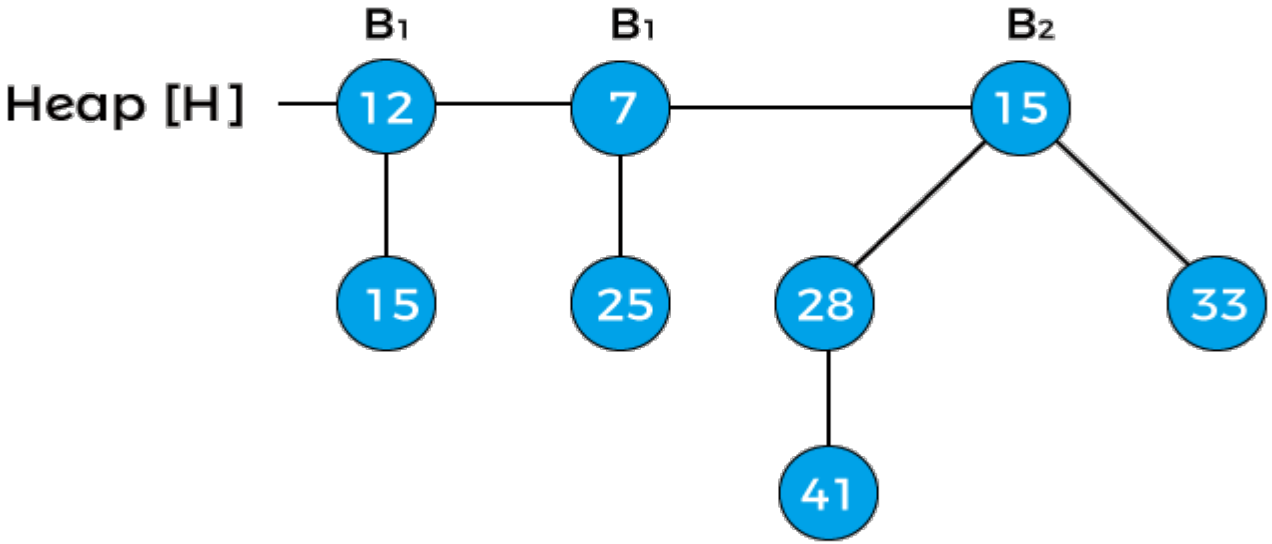
2: Insert

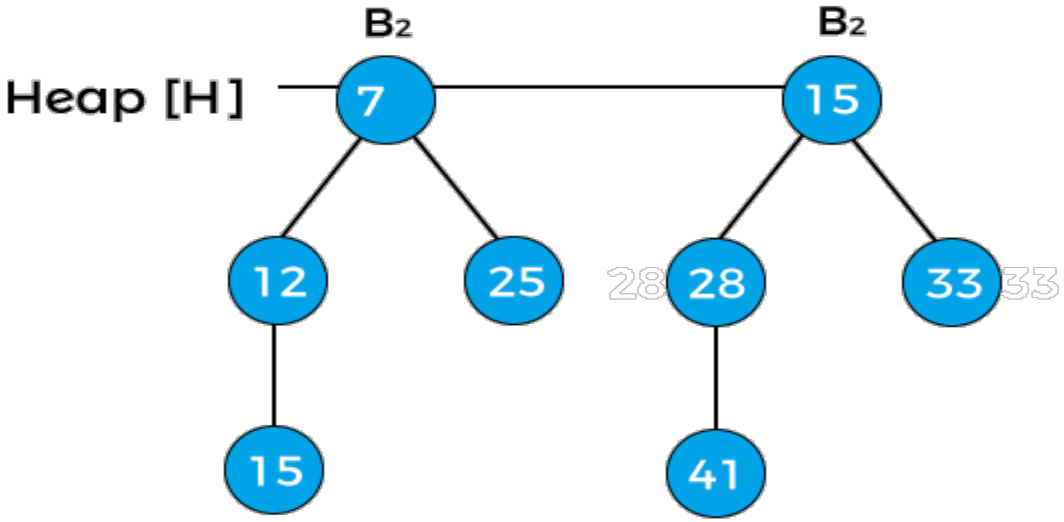

B0
B1
B2
Heap [H]
12
7
15
25
28
33
41


B0
Heap [H]
15

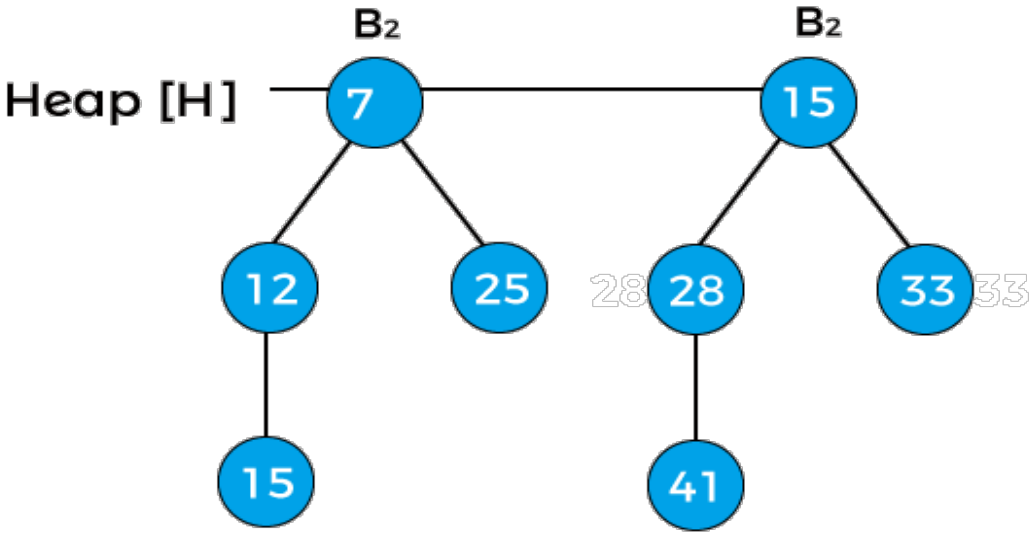

B0
B0
B1
B2
Heap [H]
12
15
7
15
25
28
33
41

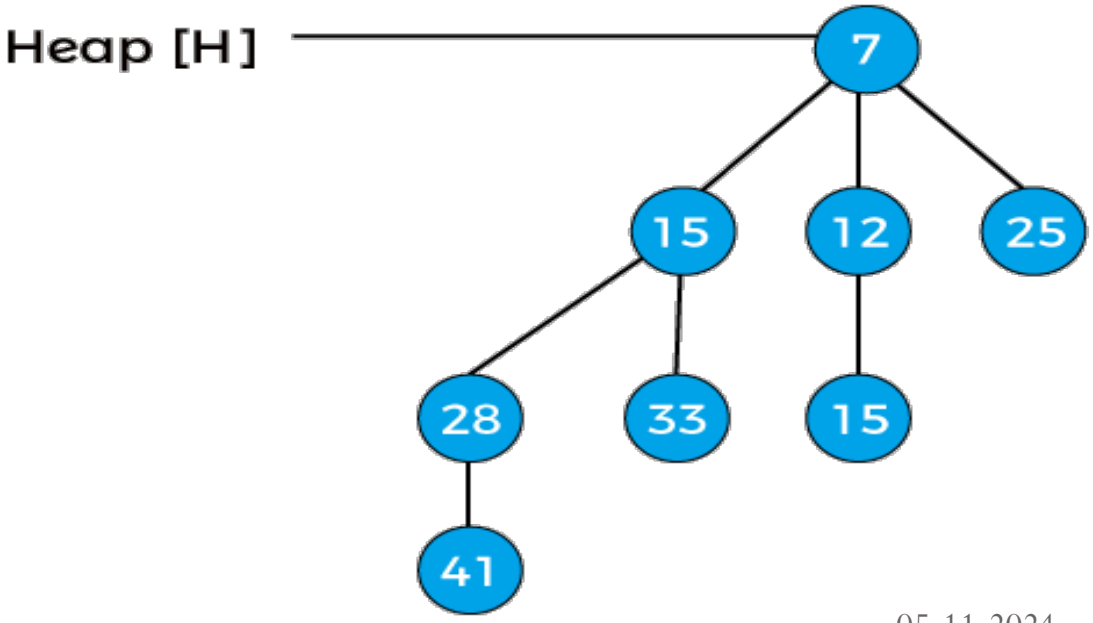
2: Insert


B1
B1
B2
Heap [H]
12
7
15
15
25
28
33
41


B2
B2
Heap [H]
7
15
12
25
28
33
15
41

2: Insert

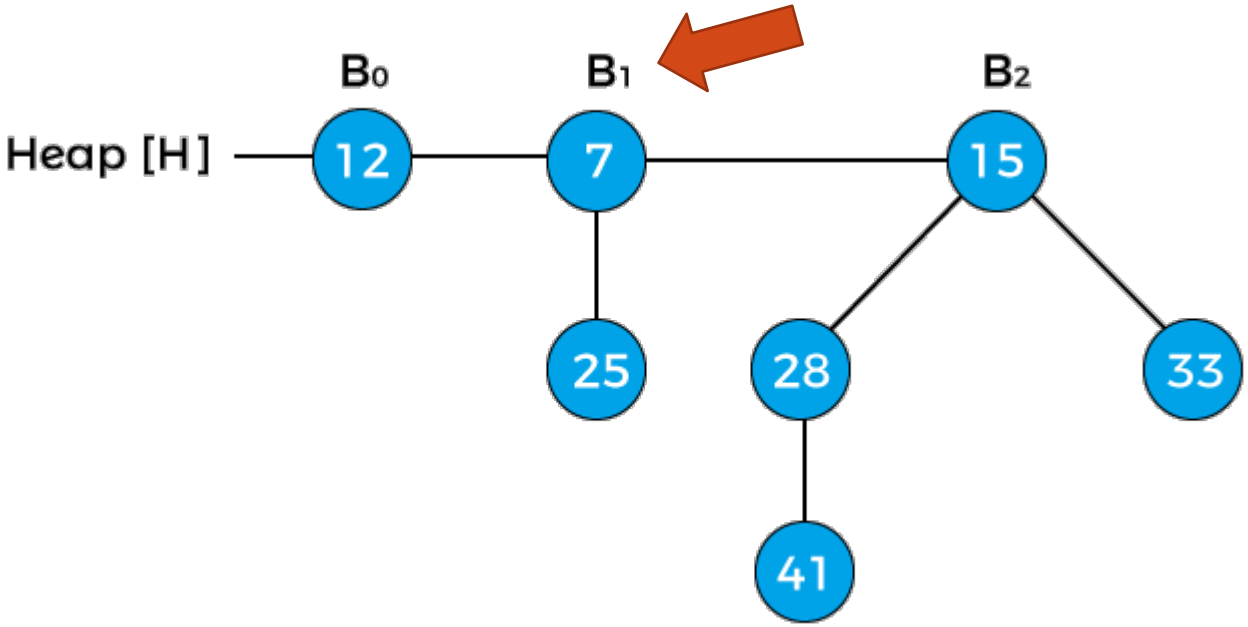




Operations of Binomial Heap ADT

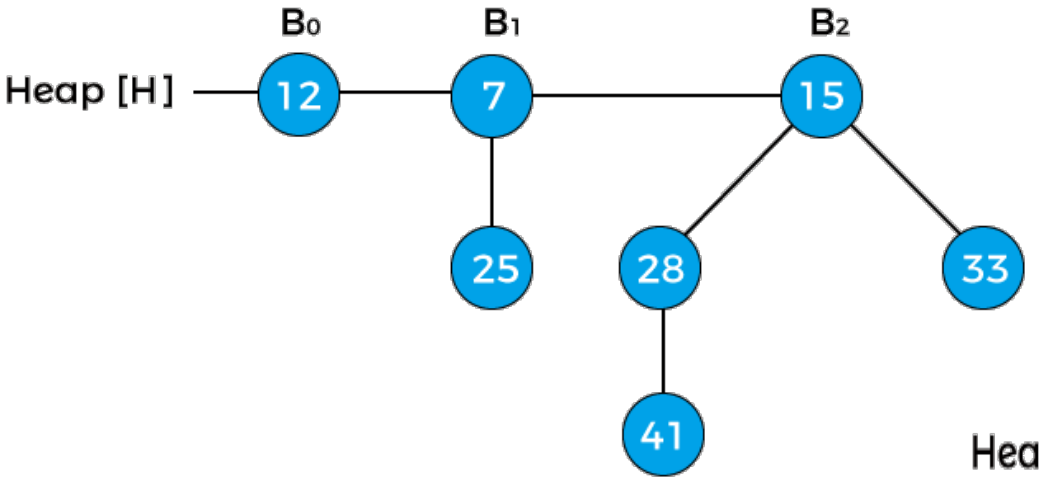
3. **Minimum(heap):**
   - **Description**: Returns the minimum key in the binomial heap without removing it.
   - **Procedure**: Finds the root with the smallest key among the binomial trees in the heap.
   - **Complexity**: $O(\log n)$, as it involves scanning the roots of all binomial trees.
4. **Extract-Min(heap):**
   - **Description**: Removes and returns the minimum key from the binomial heap.
   - **Procedure**: Finds and removes the root with the minimum key, breaks the remaining trees under this root into separate binomial heaps, and merges these heaps back into the original.
   - **Complexity**: $O(\log n)$.

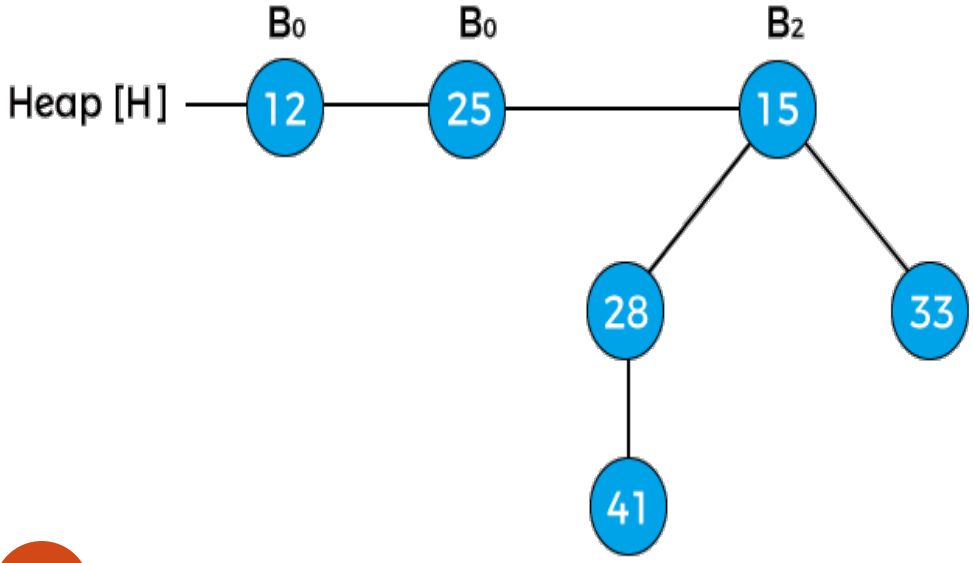
3: Returns minimum key

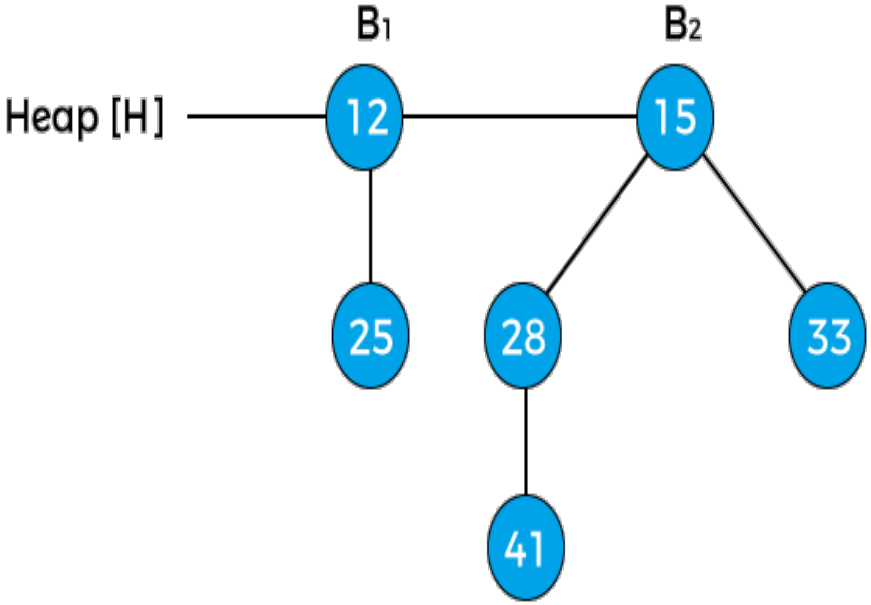


4: Extracting the minimum key

- To remove an element with the minimum key value







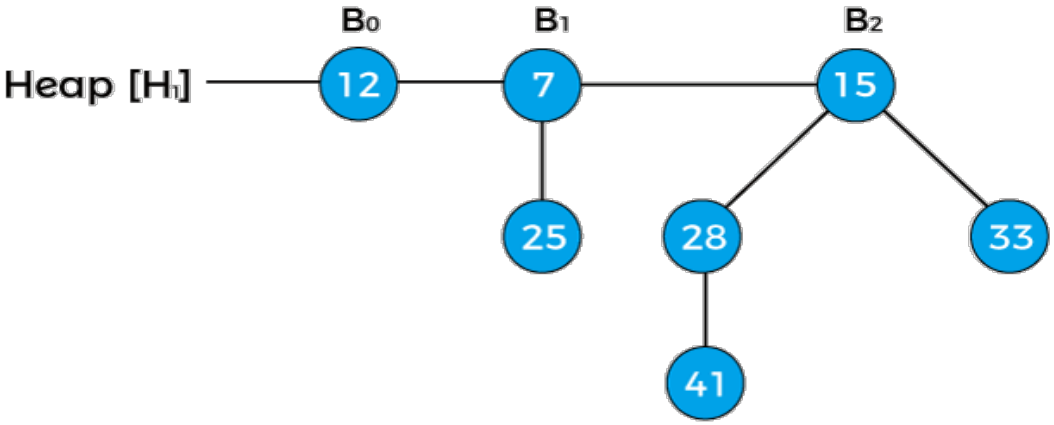
Operations of Binomial Heap ADT

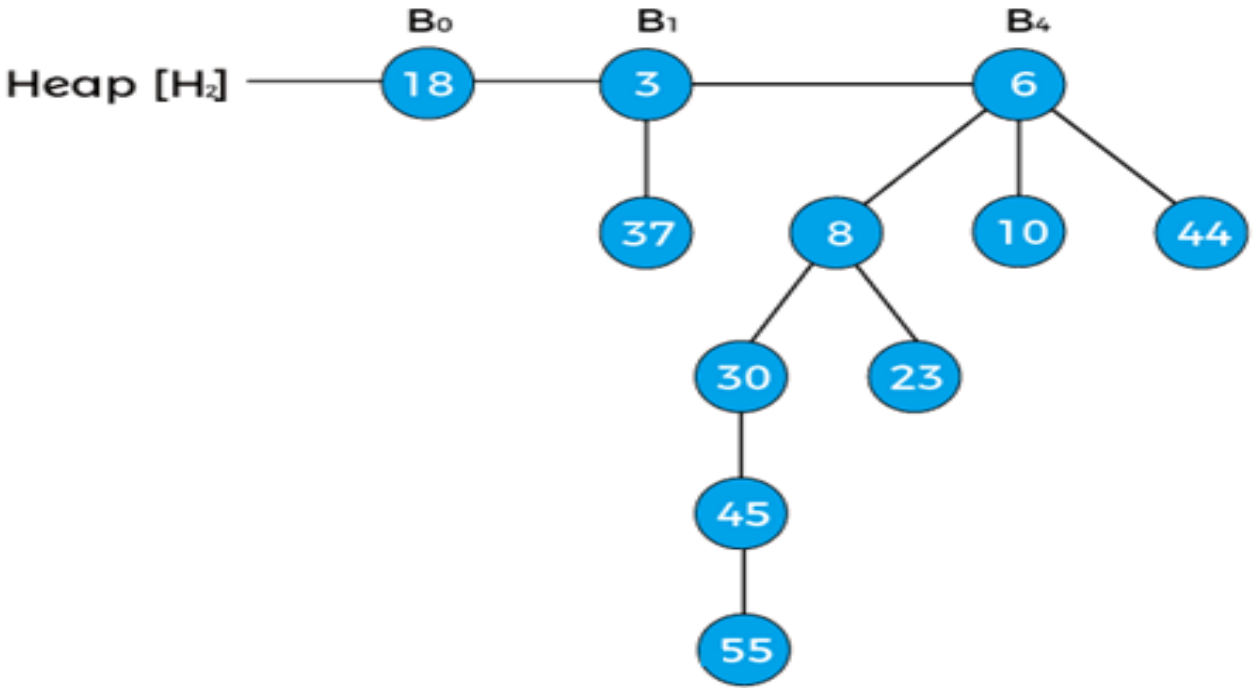
5. **Union(heap1, heap2):**
   - **Description**: Merges two binomial heaps, $heap1$ and $heap2$, into a single binomial heap.
   - **Procedure**: Combines the two heaps' binomial trees in a similar way to binary addition, combining trees of the same order recursively.
   - **Complexity**: $O(\log n)$, where n is the total number of nodes in the combined heap.
6. **Decrease-Key(heap, node, new_key):**
   - **Description**: Decreases the key of a specified node to a new, lower value.
   - **Procedure**: The key is updated, and the node is "bubbled up" through parent nodes until the min-heap property is restored.
   - **Complexity**: $O(\log n)$.
7. **Delete(heap, node):**
   - **Description**: Deletes a specified node from the binomial heap.
   - **Procedure**: First decreases the node's key to negative infinity (or a minimum possible value) using the `Decrease-Key` operation, then removes the minimum element using `Extract-Min`.
   - **Complexity**: $O(\log n)$.

5: Union or Merging of two binomial heap

- To perform the union of two binomial heaps, we have to consider the below cases -
- **Case 1:** If degree[x] is not equal to degree[next x], then move pointer ahead.
- **Case 2:** if degree[x] = degree[next x] = degree[sibling(next x)] then, Move the pointer ahead.
- **Case 3:** If degree[x] = degree[next x] but not equal to degree[sibling[next x]] and key[x] < key[next x] then remove [next x] from root and attached to x.
- **Case 4:** If degree[x] = degree[next x] but not equal to degree[sibling[next x]] and key[x] > key[next x] then remove x from root and attached to [next x].

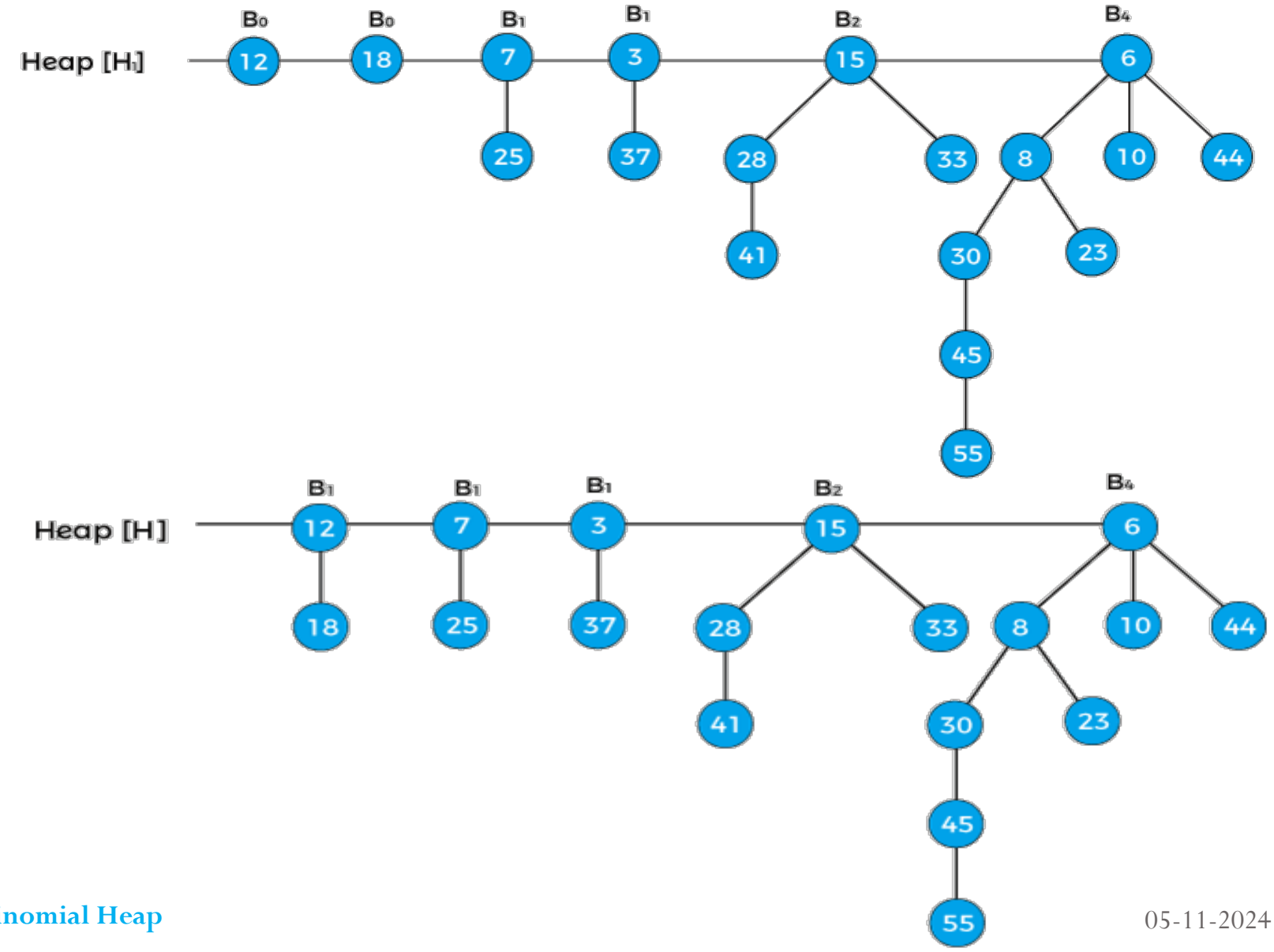
5: Union or Merging of two binomial heap



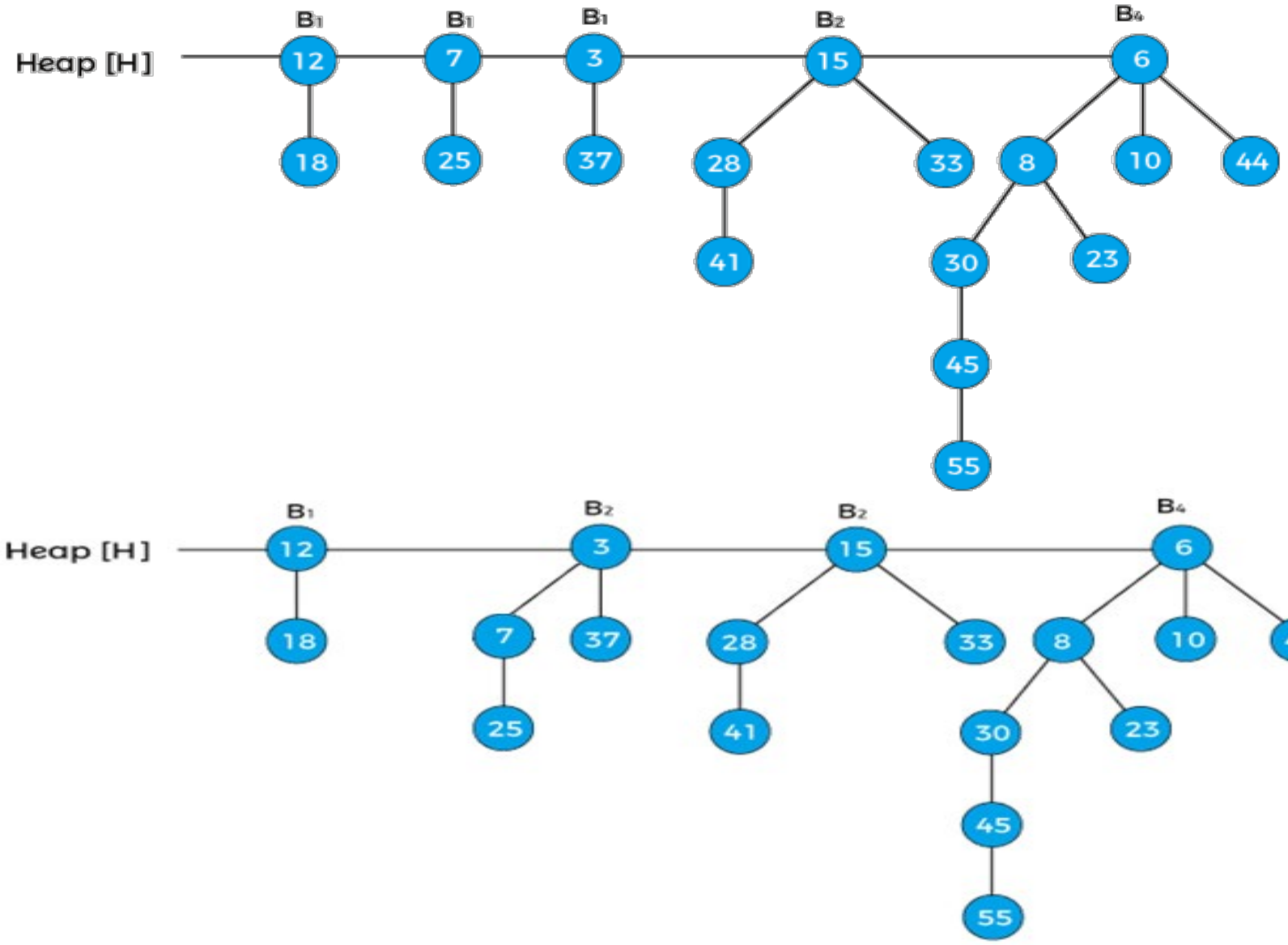


5: Union or Merging of two binomial heap

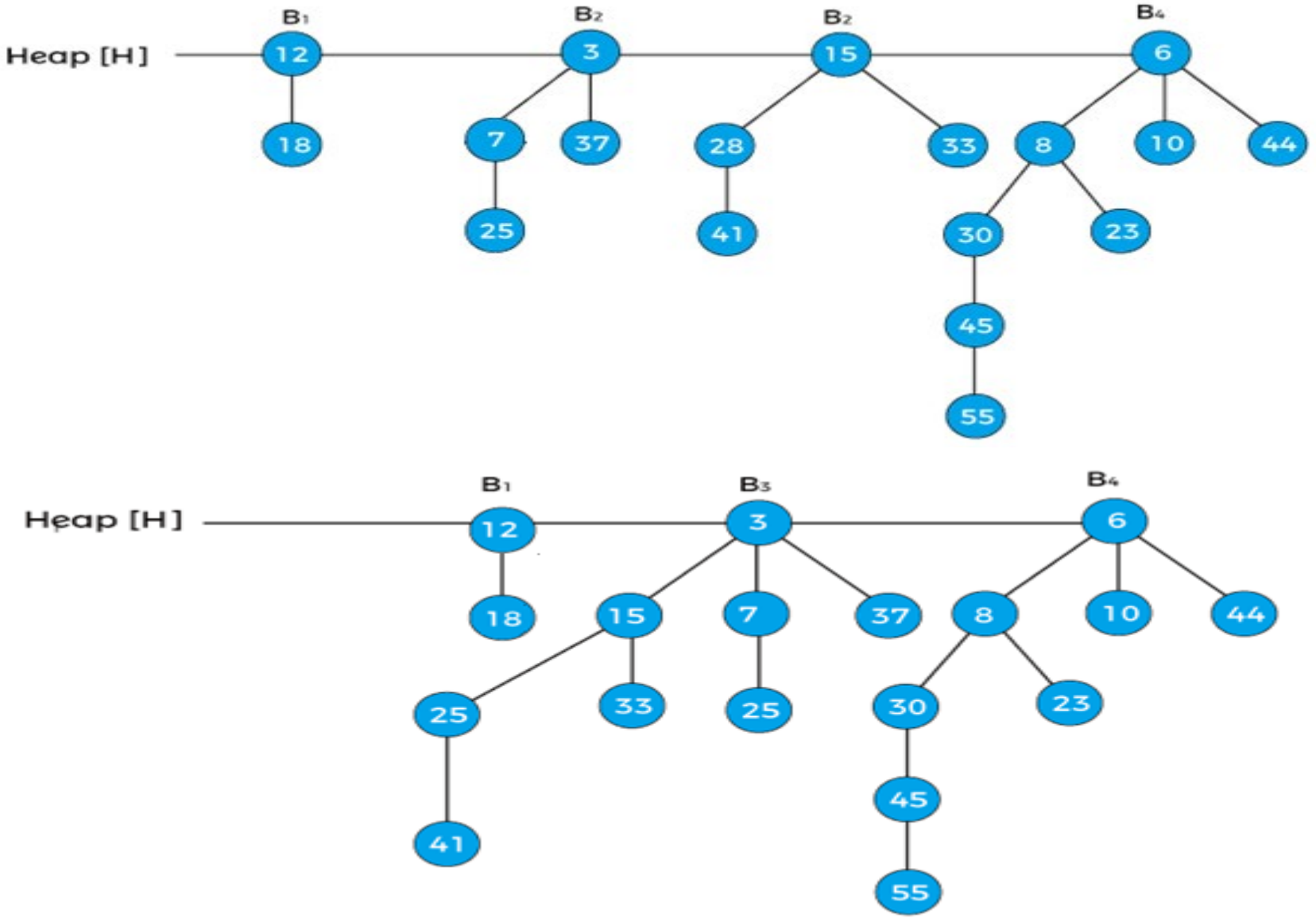
arrange their binomial trees in increasing order.



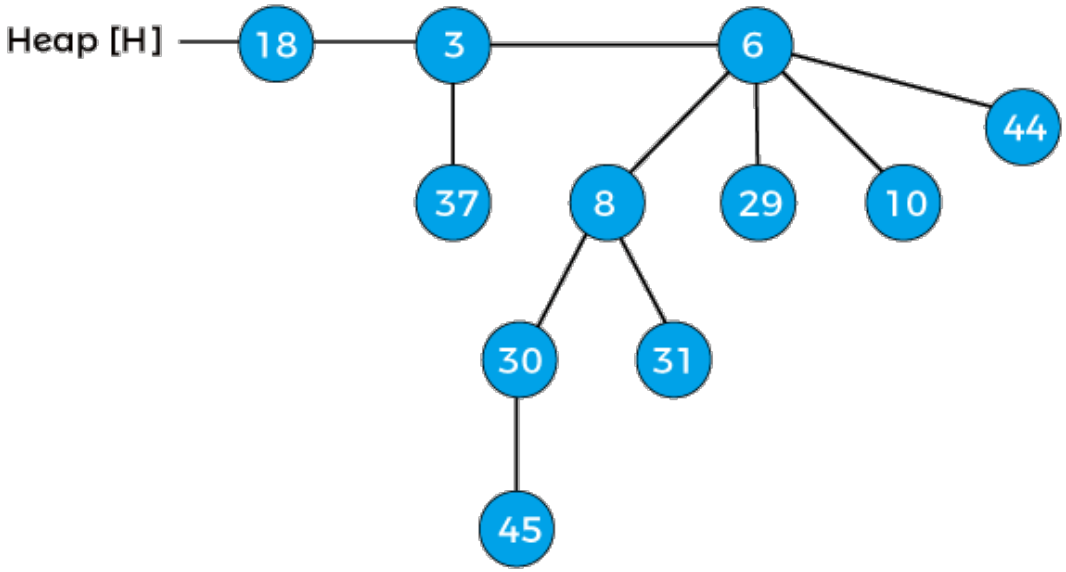
5: Union or Merging of two binomial heap



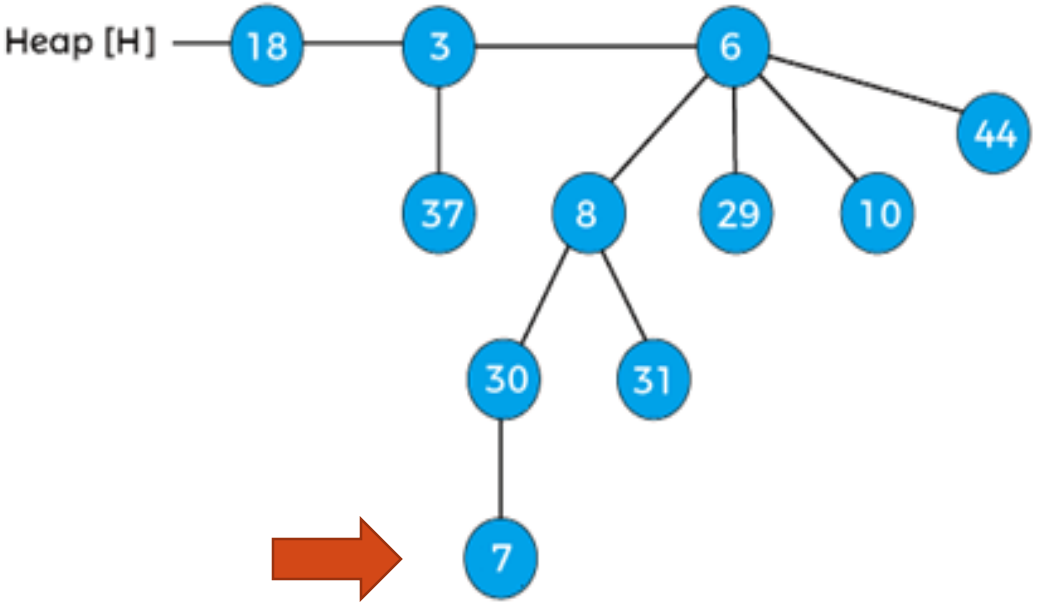
5: Union or Merging of two binomial heap



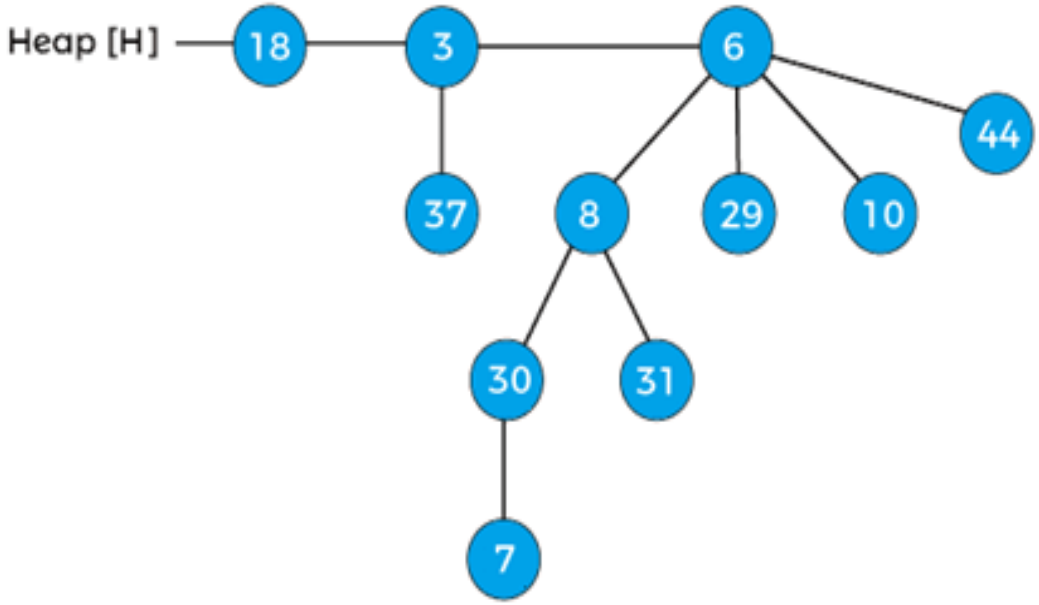
6: Decreasing a key

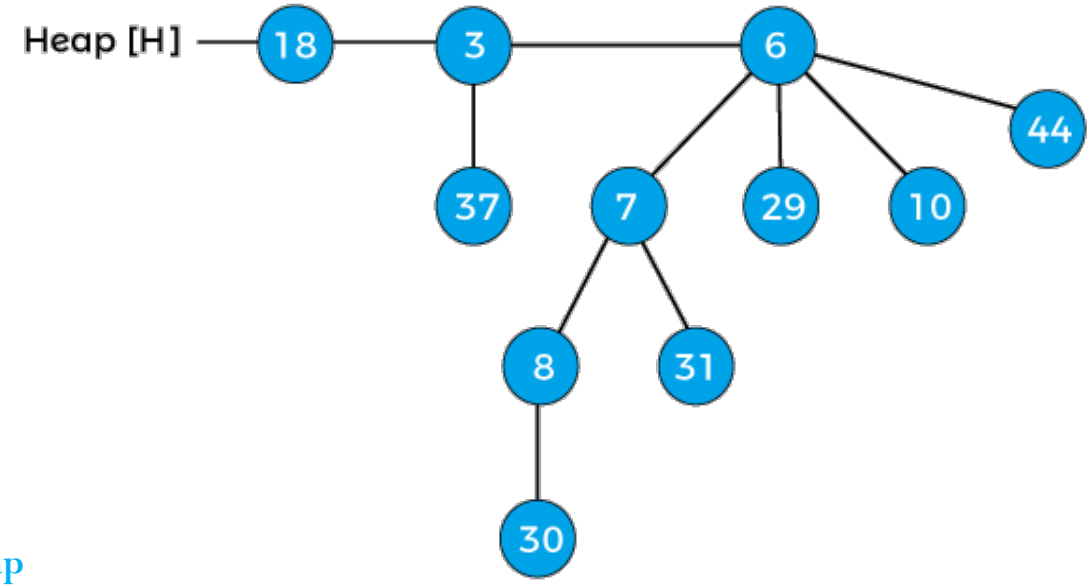


Decrease the key 45 by 7 of the above heap.

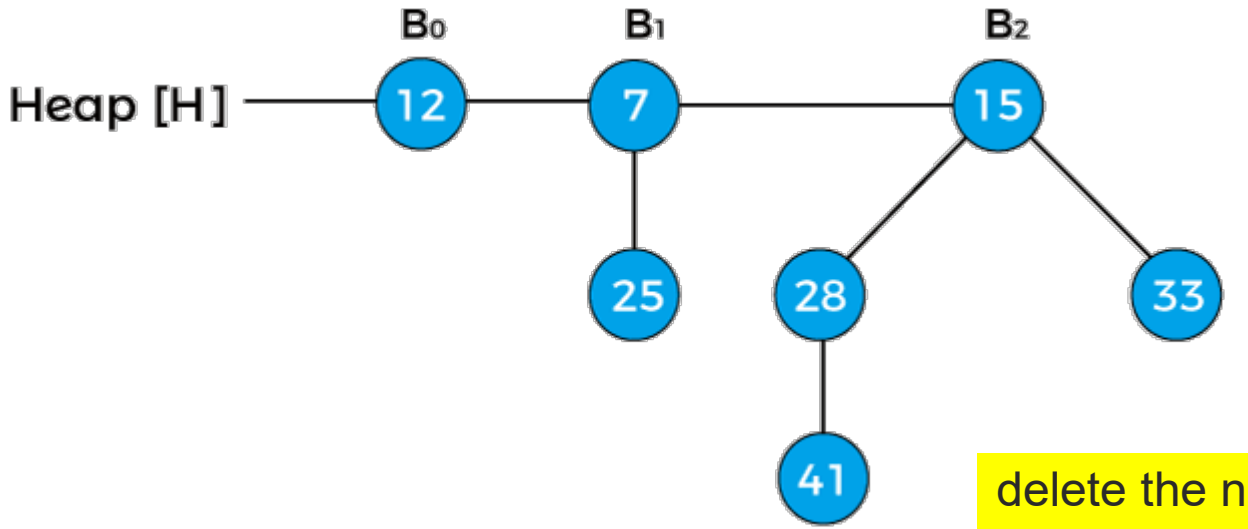


6: Decreasing a key

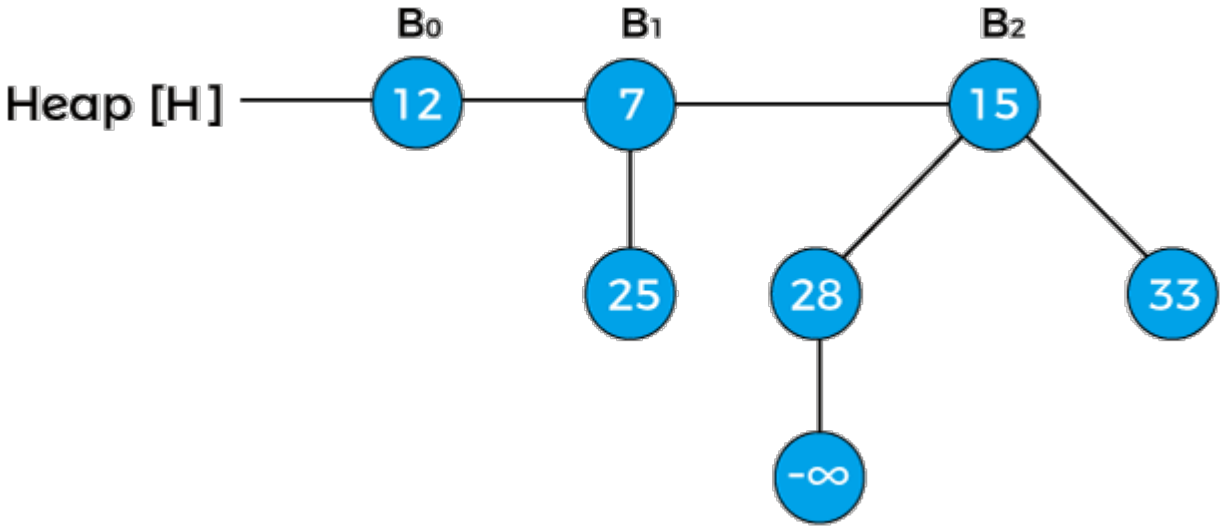




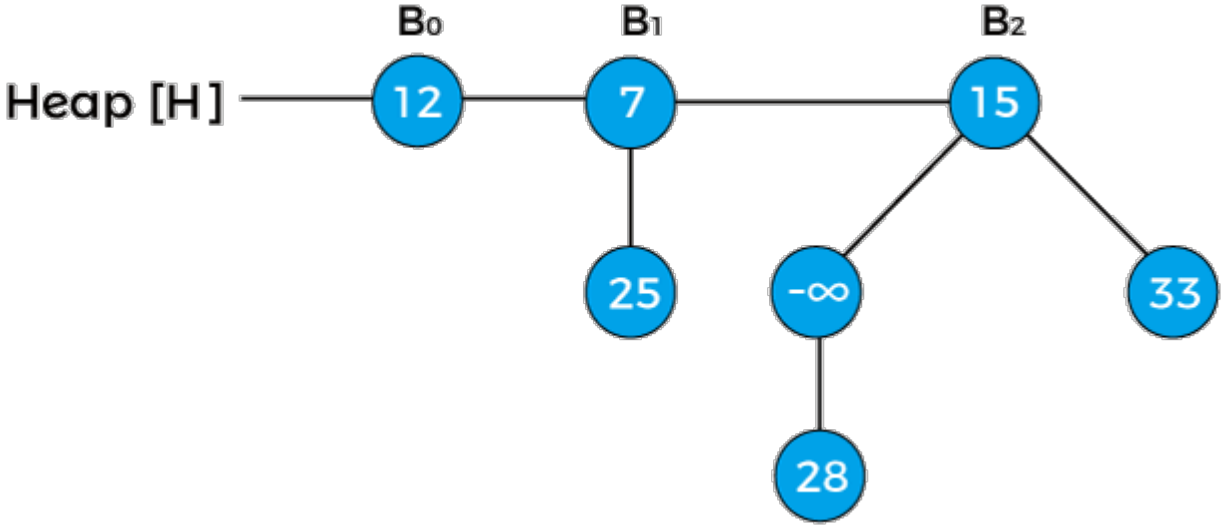
7: Deleting a node from the heap

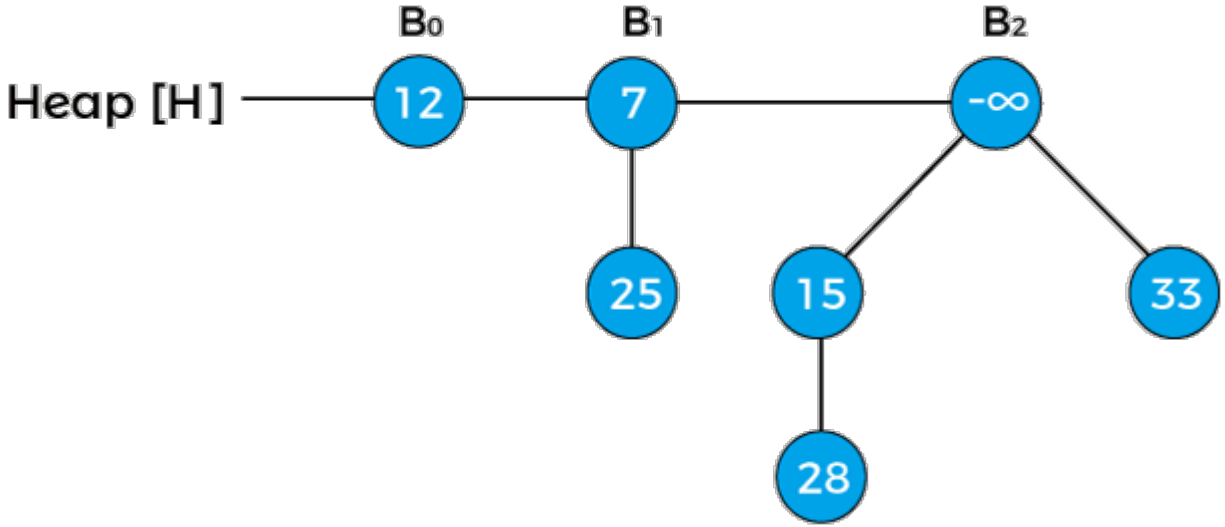


delete the node 41 from the heap

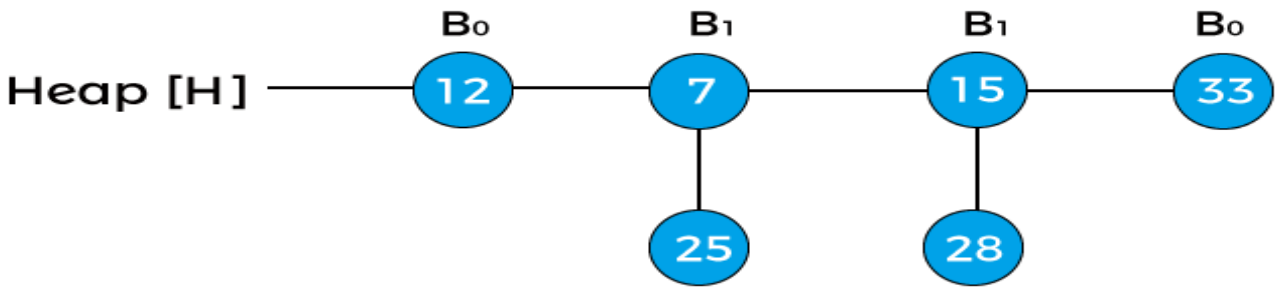


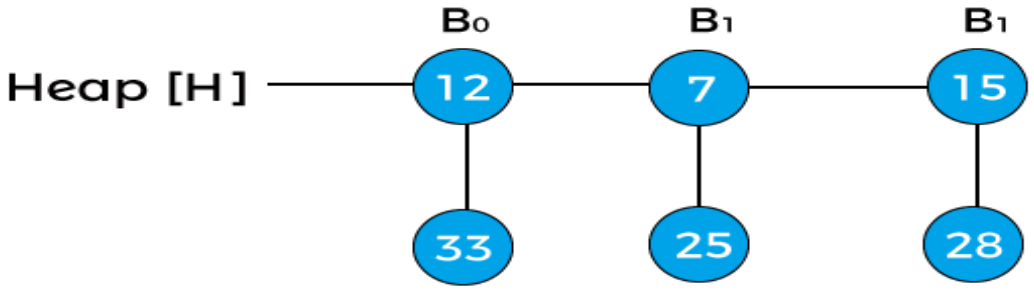
7: Deleting a node from the heap

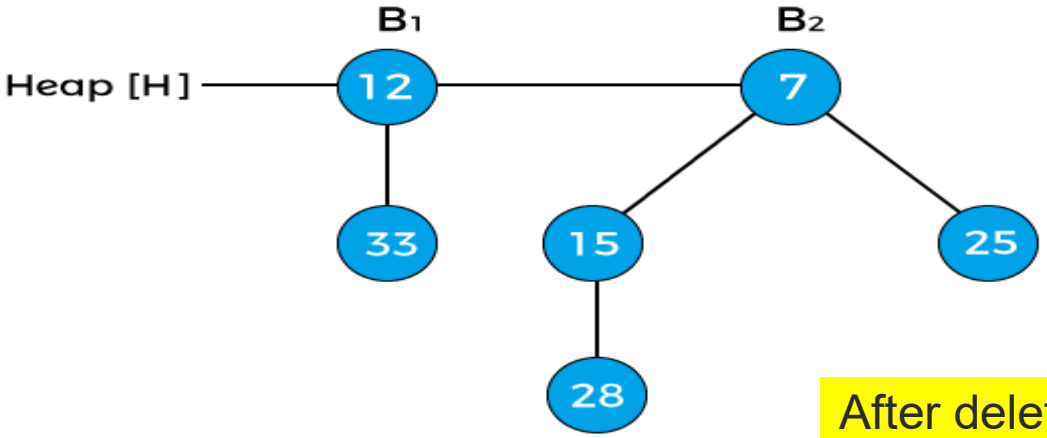




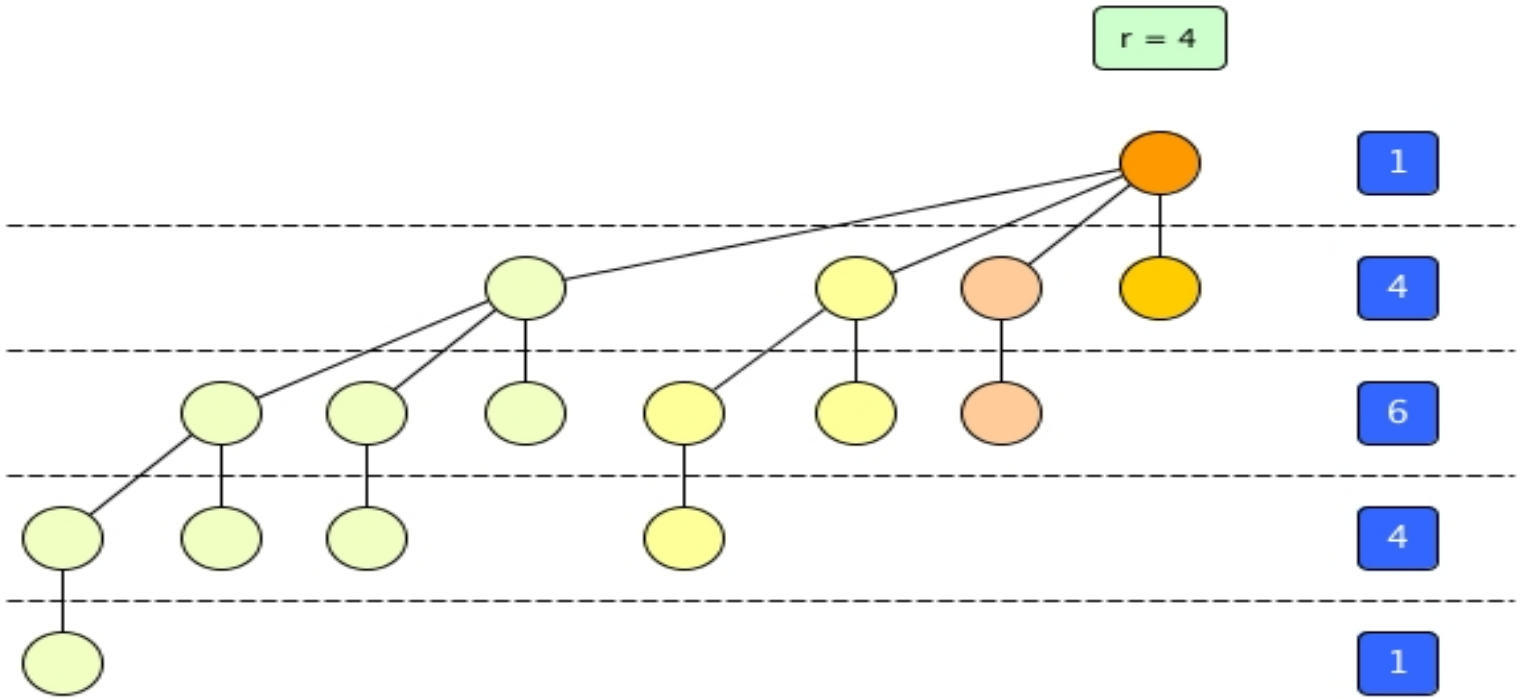
7: Deleting a node from the heap

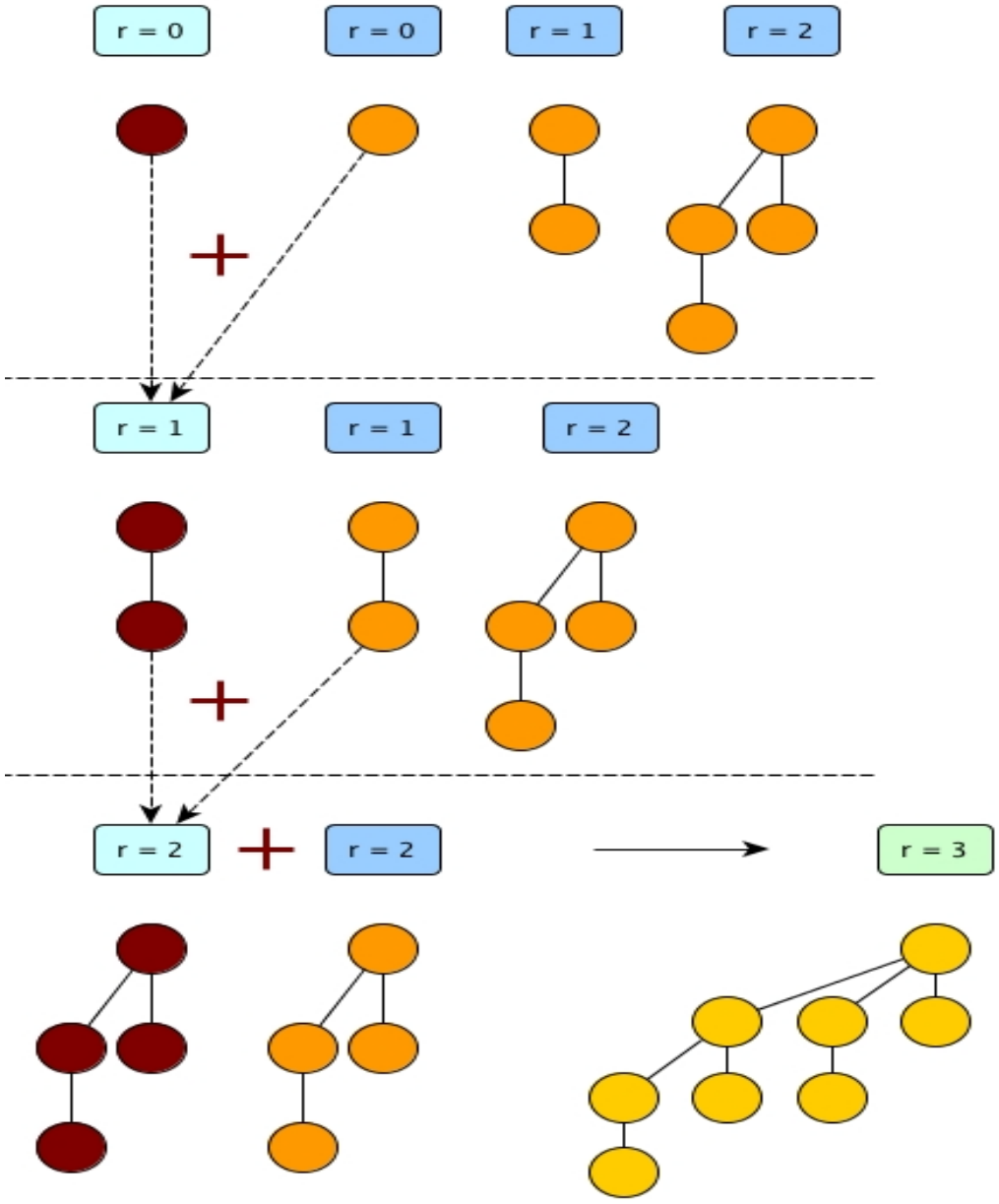




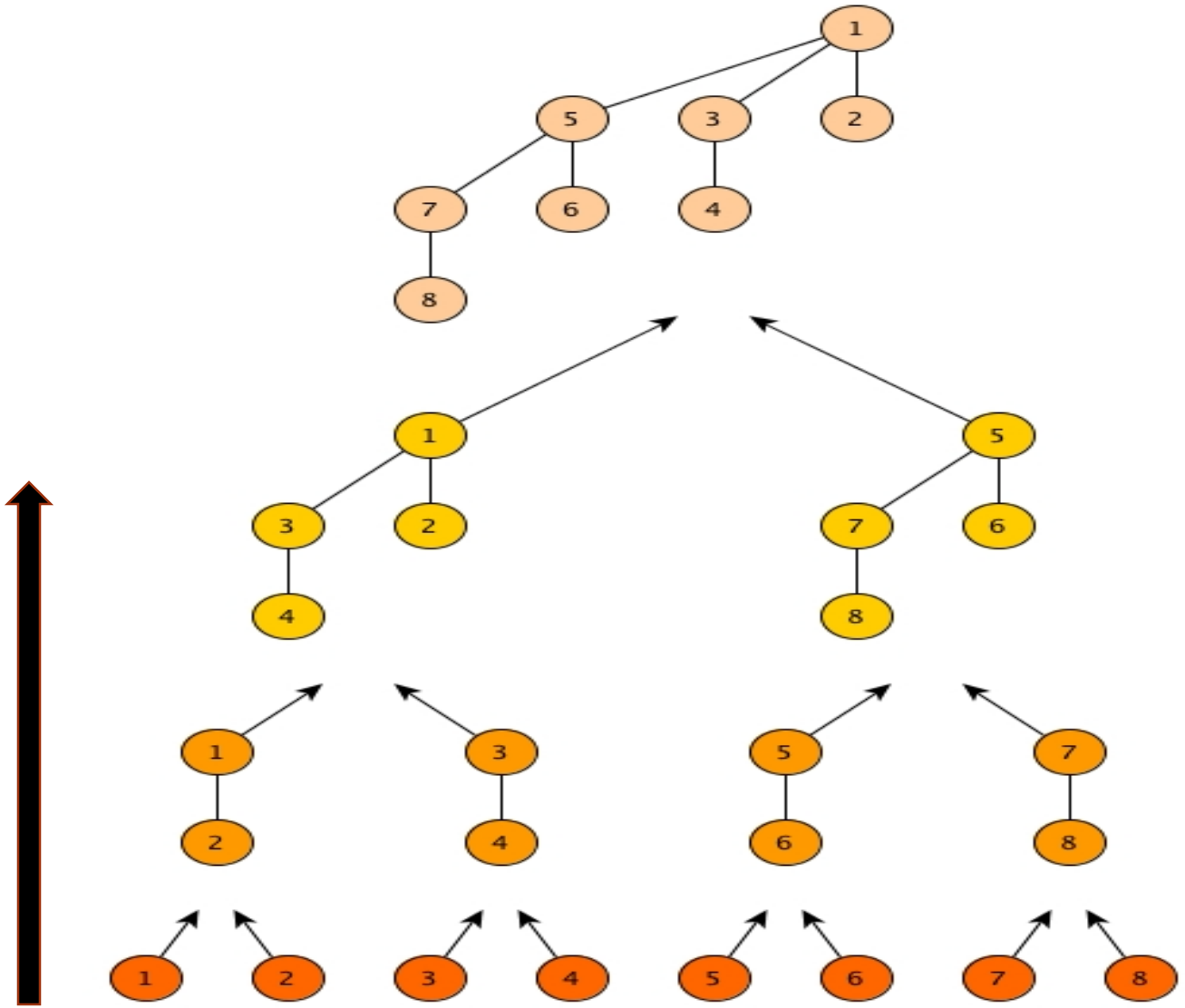


After deleting the node 41 from the heap


r = 4
1
4
6
4
1


r = 0
r = 0
r = 1
r = 2
r = 1
r = 1
r = 2
r = 2
r = 2
r = 3

Construct a Binomial Tree from a list of 2^r element



Complexity of binomial heap

Operations	Time complexity
Finding the minimum key	O(log n)
Inserting a node	O(log n)
Extracting minimum key	O(log n)
Decreasing a key	O(log n)
Union or merging	O(log n)
Deleting a node	O(log n)

Space Complexity
The space complexity of a binomial heap with 'n' elements is O(n).

Comparing Binomial heap(Min heap)

Binary Heap	Differences	Binomial Heap
A complete binary tree	Structure	Forest of trees
Each tree can hold 2^N	Number of Elements	More than 2^N
log(N)	Height of Heap	N
Olog(N)	Insert Time Complexity	Olog(N)
O(1)	Min Time Complexity	Olog(N)
Olog(N)	Remove Time Complexity	Olog(N)
Olog(N)	Decrease Time Complexity	Olog(N)
O(N)	Union Time Complexity	Olog(N)

Application of Binomial heap

1. Priority Queues: Binomial heaps are widely used in implementing priority queues, particularly in scenarios where frequent merging of heaps is required. They efficiently support operations like insertion, deletion of the minimum element, and decrease-key, making them useful for tasks that require dynamic priority management.

2. Graph Algorithms: Many graph algorithms, such as Dijkstra's shortest path algorithm and Prim's minimum spanning tree algorithm, benefit from the efficient priority queue operations provided by binomial heaps. They help manage the list of vertices with the shortest tentative distances efficiently, optimizing the performance of these algorithms.

Application of binomial heap

3. Job Scheduling: In job scheduling systems, where tasks with varying priorities are dynamically managed, binomial heaps provide an effective data structure for maintaining and updating task priorities. They allow easy insertion of new tasks, removal of the highest-priority task, and adjustment of task priorities.

4. Network Routing: Binomial heaps are used in network routing protocols that require quick access to the least-cost path. For example, in link-state routing protocols, binomial heaps can be used to efficiently manage and update routing tables by selecting the shortest path to each node.

Application of binomial heap

4. Event Simulation Systems: Binomial heaps are also useful in discrete event simulation systems, where events are managed in a priority queue based on their occurrence time. The heap structure allows for the efficient scheduling of future events and processing of events in a priority order.

5. Dynamic Set Operations: Binomial heaps are suitable for applications where dynamic sets need to be merged frequently, as they offer an efficient merge operation (union of two heaps). This makes them advantageous for scenarios where multiple data sets need to be combined and processed dynamically.


Thank you for your attention

References

- http://typeocaml.com/2015/03/17/binomial-heap/
- https://www.geeksforgeeks.org/binomial-heap-2/

Exercises

Exercises

1. What is a binomial heap, and how does it differ from a regular binary heap?
2. Explain the structure of a binomial tree. How are binomial trees arranged in a binomial heap?
3. Describe the min-heap and max-heap properties in the context of binomial heaps.
4. Why are binomial heaps useful in applications where merging of heaps is frequent?
5. What is the time complexity of merging two binomial heaps, and why?
6. Describe the process of inserting a new element into a binomial heap.
7. How is the minimum (or maximum) element located in a binomial heap, and what is its time complexity?
8. Explain the steps involved in deleting the minimum element from a binomial heap.
9. How does the union (or merge) operation work in a binomial heap? Why is this operation efficient?

Exercises

10. How can we implement a decrease-key operation in a binomial heap, and what is its complexity?
11. Explain how the delete operation works in a binomial heap and its complexity.
12. Describe a scenario where a binomial heap would be more efficient than a binary heap.
13. What would be the binomial tree structure of a binomial heap containing 13 elements?
14. Compare binomial heaps to Fibonacci heaps in terms of efficiency and use cases.
15. In what types of algorithms or problems would using a binomial heap be particularly advantageous?
16. Explain the trade-offs between a binomial heap and other heap structures, like Fibonacci heaps or binary heaps, for different applications.

Exercises

17. How would you implement Dijkstra's shortest path algorithm using a binomial heap?
18. Describe how Prim's algorithm for minimum spanning trees can benefit from using a binomial heap.
19. How would a binomial heap be used in a priority queue for a scheduling system where tasks frequently need re-prioritization?
20. Why is each binomial tree in a binomial heap uniquely defined by its order?
21. How does this uniqueness help in merging operations?
22. What is the maximum number of binomial trees that a binomial heap with n nodes can contain? Explain your reasoning.
23. Explain why a binomial tree of order k has exactly 2^k nodes and a height of k.
24. In what cases would you end up with two binomial trees of the same order after an insertion or merge operation, and how is this resolved?
25. How does the binary representation of a number influence the structure of binomial heaps?

Exercises

26. Describe the procedure for merging two binomial trees of the same order in a binomial heap. What makes this operation efficient?
27. Explain how you would handle the "carry" during a merge operation when you encounter two binomial trees of the same order.
28. What is the difference between the decrease-key operation in a binomial heap versus other types of heaps, and why is it more complex?
29. What data structure(s) could you use to track the minimum element in a binomial heap efficiently?
30. Discuss how binomial heaps can be implemented with pointers versus array-based structures. What are the trade-offs?
31. What edge cases should you consider when merging two binomial heaps, particularly if one is empty?
32. How would you implement the delete operation in a binomial heap, and what specific challenges would you face?
33. Explain a method for finding the second minimum element in a binomial heap without using additional storage.

Exercises

34. Analyze the space complexity of a binomial heap and discuss if and how it could be optimized.
35. Why might binomial heaps be less efficient than Fibonacci heaps for decrease-key operations, and how does this impact their use cases?
36. For what range of applications would binomial heaps offer performance advantages, and where would they fall short compared to other heap structures?
37. How does the amortized time complexity of binomial heap operations compare to their worst-case time complexity?
38. Compare and contrast the use of binomial heaps in Prim's algorithm versus Dijkstra's algorithm. Would one benefit more than the other?
39. Why might a binomial heap not be ideal for an application that requires frequent decrease-key operations, such as a dynamically changing priority queue?
40. Discuss situations in which merging multiple heaps is necessary and how binomial heaps provide a solution to this problem.