# ADVANCED DATA STRUCTURE (R16)

## Unit 3 (Priority Queues - Heaps)

**Syllabus:** Model, Simple Implementation, Binary Heap-Structure Property-Heap-Order Property-BasicHeap Operations- Other Heap Operation, Applications of Priority Queues- The SelectionProblem Event Simulation Problem, Binomial Queues- Binomial Queue Structure – BinomialQueue Operation- Implementation of Binomial Queues.

## Important Questions

1. Define priority queues? Write implementation &applications of priority queues?
2. Define Binary Heap? Write the properties and applications of Binary Heap?
3. Define Binary Tree? Write the properties and applications of Binary Tree?
4. Define Binomial Queue? Write the properties and applications of Binomial Queue?
5. Discuss about operations of Binomial Queue and explain with example?
6. Explain about Heap up and Heap down?
7. How to implement Binomial queues and explain with example?
8. Differentiate between binary tree and binomial tree?
9. Construct Binary Min Heap for 11,45,23,9,4,16,8,29,1,12,21,15.
10. Construct Binomial heap for 1,2,3,4,5,6,7 and delete 1.

## 1. Define priority queues? Write implementation & applications of priority queues?

**Answer:**

**Definition:**

A priority queue is a data structure for maintaining a collection of items and every item has a priority associated with it.

Priority queue contains data items which have some preset priority. While removing an element from a priority queue, the data item with the highest priority is removed first.

In a priority queue, insertion is performed in the order of arrival and deletion is performed based on the priority.

A real-life example of a priority queue would be a hospital queue where the patient with the most critical situation would be the first in the queue. In this case, the priority order is the situation of each patient.

A priority queue supports the following operations:

- Insert(item, priority) - add an item and its key to the priority queue.
- maximum (or minimum) - return the item of highest priority.
- deleteMax (or deleteMin) - remove the item of highest priority and return it.

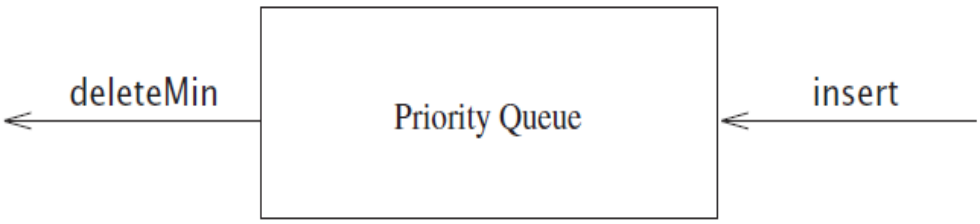



**Figure 6.1** Basic model of a priority queue

**Implementation:**

There are many ways to implement priority queues.

- **Using Arrays:** performing insertions at the end of array in **O (1)** and requires **O(N)** time to delete the item by linear search.
- **Using Linked Lists:** Using simple linked list, performing insertions at the front in **O(1)** and requires **O(N)** time to delete the minimum.
- **Using Binary Heaps:** Using binary search tree, gives an **O(logN)** average running time for both insertion and deletion operations.
- Priority queues are implemented by **Binary Heaps** and **Binomial Heaps**.

**Applications:**

- Implementing Dijkstra's algorithm for finding shortest path.
- Implementing Prims algorithm for finding minimum spanning tree.
- Implementing Huffman Coding for data compression.
- Priority Queues have many applications besides operating systems (Interrupt Handling).
- CPU Scheduling.
- Priority Queues are used to implement external sorting.
- A* algorithm in Artificial Intelligence (finds the shortest path between two vertices of a weighted graph, trying out the most promising routes first.)

## 2. Define Binary Heap? Write the properties and applications of Binary Heap?

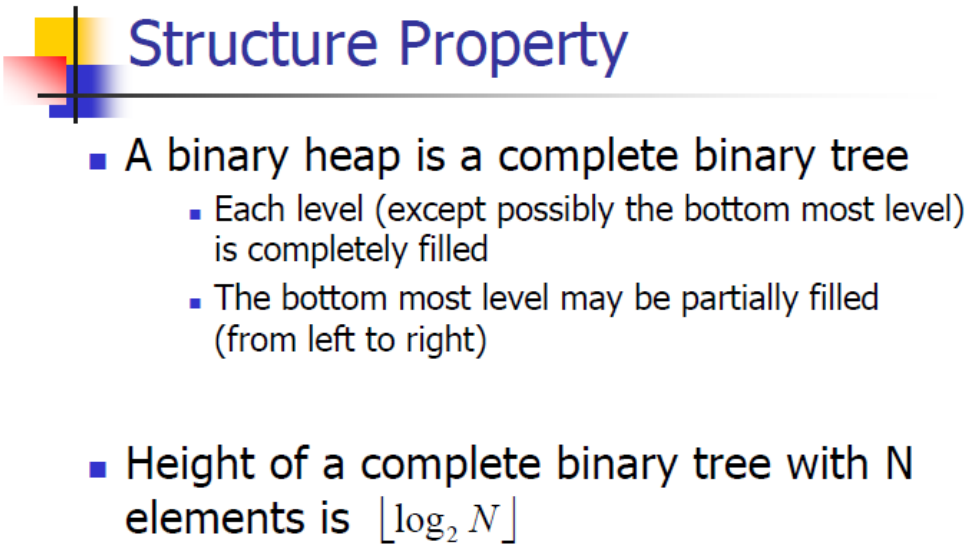
**Answer:**

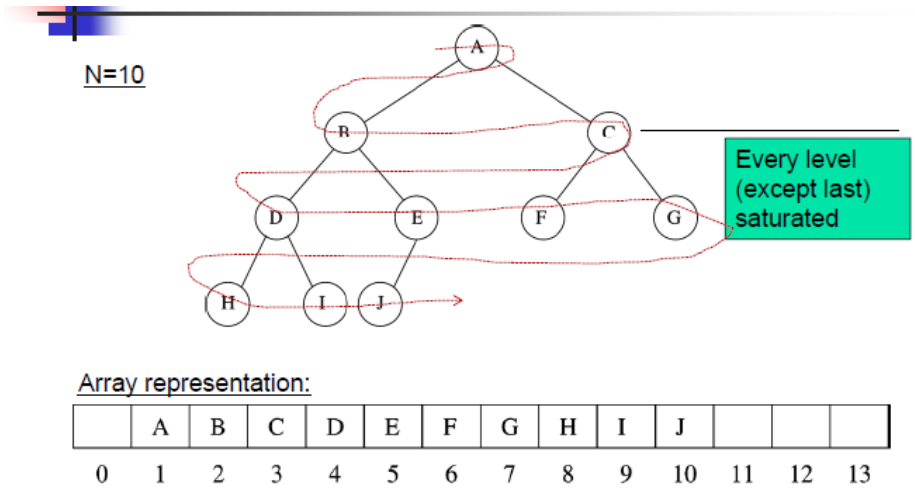
Binary heap or heaps are used to implementing Priority Queues.

Binary Heap is a partially complete binary tree and also satisfies the following two properties.

(i) Heap Structure Property
(ii) Heap Order Property

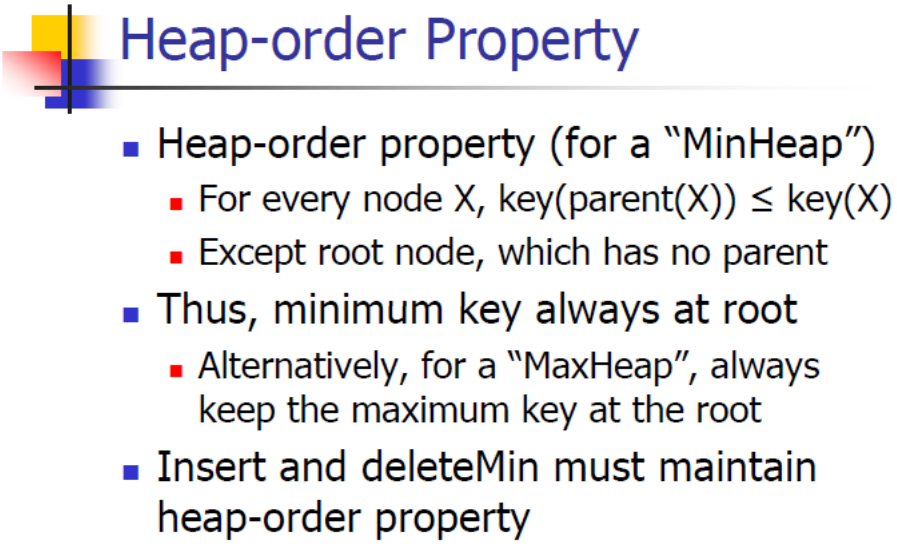
**Heap Structure Property:** It's a partially complete binary tree (All levels are completely filled by two children's except possibly the last level and the last level has all keys as left as possible).

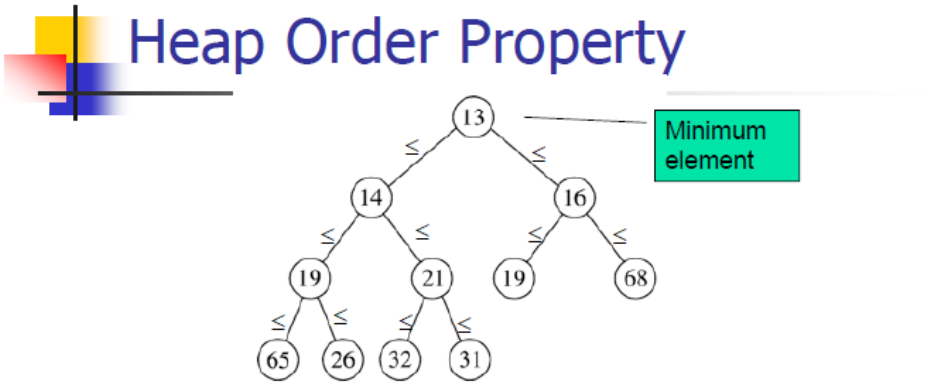


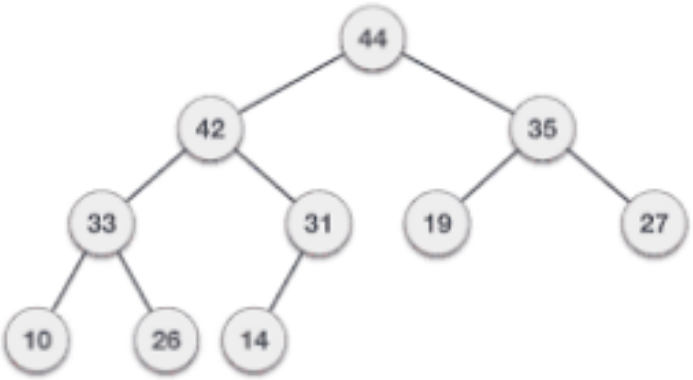



## Heap order Property:

A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to Min Heap.





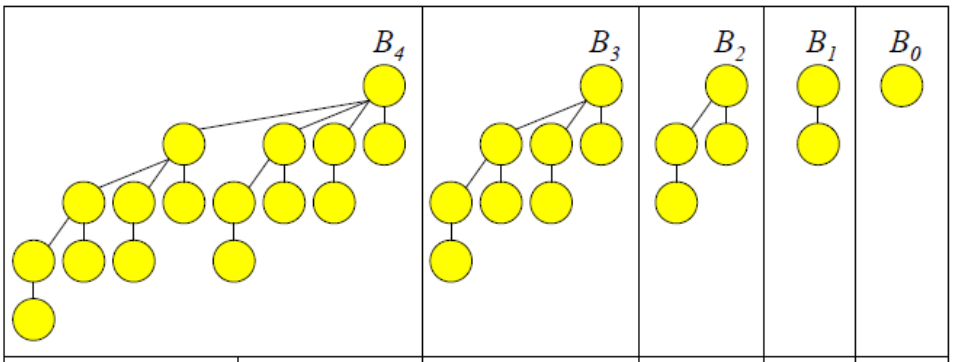

3. **Define Binomial Queue? Write the properties and applications of Binomial Queue?**

**Answer:**

A **binomial queue** is a collection (or *forest*) of heap-ordered binomial trees.

- Not just one tree, but a collection of trees.
- Each tree has a defined structure.
- Each tree has the familiar heap-order property.

# Binomial Queue with 5 Trees




| depth | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| number of elements | $2^4 = 16$ | $2^3 = 8$ | $2^2 = 4$ | $2^1 = 2$ | $2^0 = 1$ |

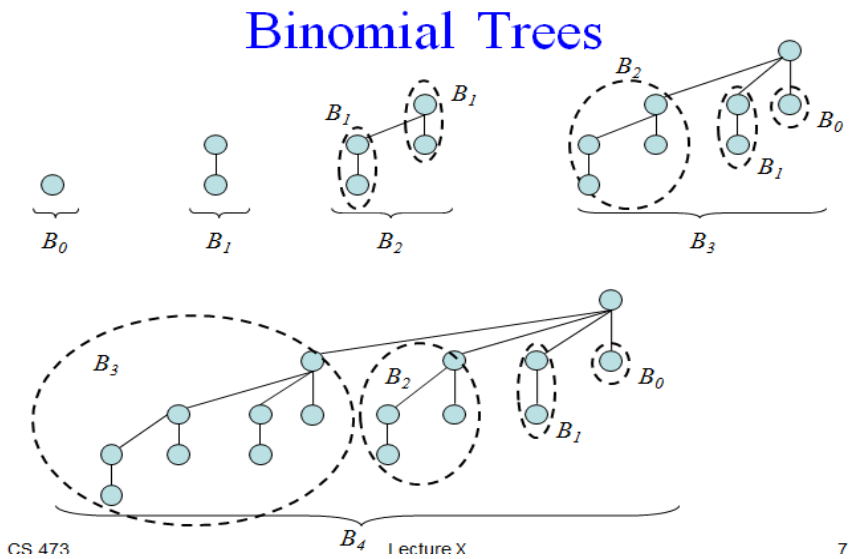
**Binomial Tree:**

The binomial tree Bk is an ordered tree defined recursively.

$B_0$ consists of a single node ……..

$B_k$ consists of two binominal trees Bk-1 linked together.

Root of one is the leftmost child of the root of the other.




## Binomial Queue properties:

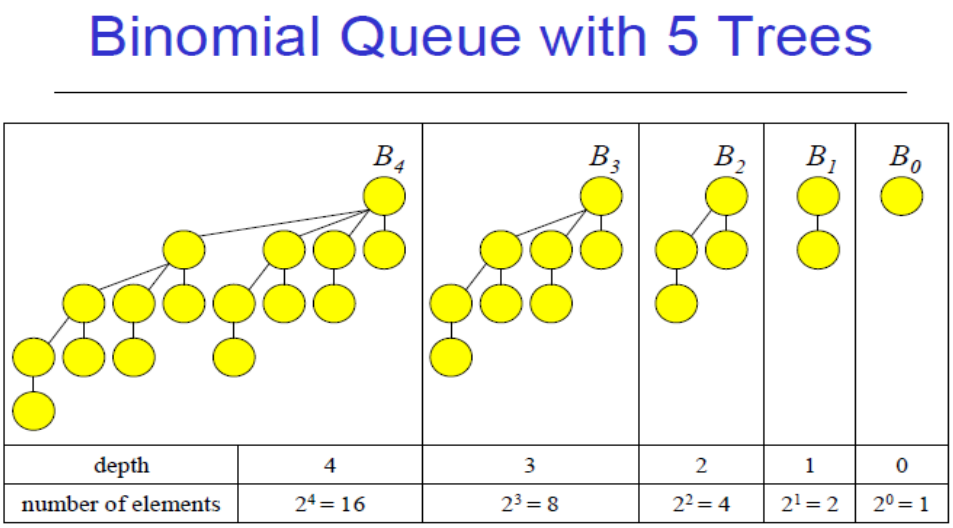
1. **Structure Property:**

   (i). Each tree contains two copies of the previous tree.

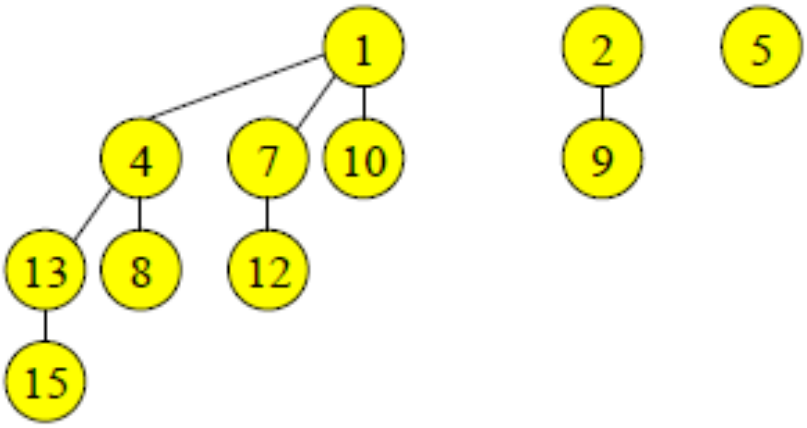
   - The second copy is attached at the root of the first copy.

   (ii). The number of nodes in a tree of heighth is exactly $2^h$.

(iii). The number of nodes in a specific level L is ${}^{N}C_{L} = N!/(L!(N - L)!)$




2. **Heap order Property:**
   Binomial Queue is a collection of min heap ordered binomial trees.




## Applications of Binomial Queue:

1. Implementing priority queues
2. Implementing discrete event simulation

## 4. Discuss about operations of binomial queues?

**Answer: (http://lcm.csa.iisc.ernet.in/dsa/node141.html)**

The main operation in Binomial Heap is union(), all other operations mainly use this operation. The union() operation is to combine two Binomial Heaps into one. Let us first discuss other operations, we will discuss union later.

1) **insert(H, k):** Inserts a key 'k' to Binomial Heap 'H'. This operation first creates a Binomial Heap with single key 'k', then calls union on H and the new Binomial heap.

2) **getMin(H):** A simple way to getMin() is to traverse the list of root of Binomial Trees and return the minimum key. This implementation requires O(Logn) time. It can be optimized to O(1) by maintaining a pointer to minimum key root.

3) **extractMin(H):** This operation also uses union(). We first call getMin() to find the minimum key Binomial Tree, then we remove the node and create a new Binomial Heap by connecting all subtrees of the removed minimum node. Finally we call union() on H and the newly created Binomial Heap. This operation requires O(Logn) time.

4) **delete(H):** Like Binary Heap, delete operation first reduces the key to minus infinite, then calls extractMin().

5) **decreaseKey(H):** decreaseKey() is also similar to Binary Heap. We compare the decreases key with it parent and if parent's key is more, we swap keys and recur for parent. We stop when we either reach a node whose parent has smaller key or we hit the root node. Time complexity of decreaseKey() is O(Logn).

**Union operation in Binomial Heap:**

Given two Binomial Heaps H1 and H2, union(H1, H2) creates a single Binomial Heap.

1) The first step is to simply merge the two Heaps in non-decreasing order of degrees. In the following diagram, figure(b) shows the result after merging.

2) After the simple merge, we need to make sure that there is at-most one Binomial Tree of any order. To do this, we need to combine Binomial Trees of same order. We traverse the list of merged roots, we keep track of three pointers, prev, x and next-x. There can be following 4 cases when we traverse the list of roots.

Case 1: Orders of x and next-x are not same, we simply move ahead.

In following 3 cases orders of x and next-x are same.

Case 2: If order of next-next-x is also same, move ahead.

Case 3: If key of x is smaller than or equal to key of next-x, then make next-x as a child of x by linking it with x.

Case 4: If key of x is greater, then make x as child of next.

(https://www.geeksforgeeks.org/binomial-heap-2/)

## 5. How to implement/ represent Binomial queues and explain with example?

**Answer:**

**Representation of binomial queues:**

As shown in Figure 19.3(b), each binomial tree within a binomial heap is stored in the left-child, right-sibling representation of Section 10.4. Each node has a *key* field and any other satellite information required by the application. In addition, each node $x$ contains pointers $p[x]$ to its parent, $child[x]$ to its leftmost child, and $sibling[x]$ to the sibling of $x$ immediately to its right. If node $x$ is a root, then $p[x] = \text{NIL}$. If node $x$ has no children, then $child[x] = \text{NIL}$, and if $x$ is the rightmost child of its parent, then $sibling[x] = \text{NIL}$. Each node $x$ also contains the field $degree[x]$, which is the number of children of $x$.

As Figure 19.3 also shows, the roots of the binomial trees within a binomial heap are organized in a linked list, which we refer to as the ***root list***. The degrees of the roots strictly increase as we traverse the root list. By the second binomial-heap property, in an $n$-node binomial heap the degrees of the roots are a subset of $\{0, 1, \ldots, \lfloor \lg n \rfloor\}$. The *sibling* field has a different meaning for roots than for nonroots. If $x$ is a root, then $sibling[x]$ points to the next root in the root list. (As usual, $sibling[x] = \text{NIL}$ if $x$ is the last root in the root list.)

A given binomial heap $H$ is accessed by the field $head[H]$, which is simply a pointer to the first root in the root list of $H$. If binomial heap $H$ has no elements, then $head[H] = \text{NIL}$.

**Each node has the following fields:**

**p: parent**

**child: leftmost child**

**sibling**

**Degree**

**Key**

**Roots of the trees are connected using linked list**.

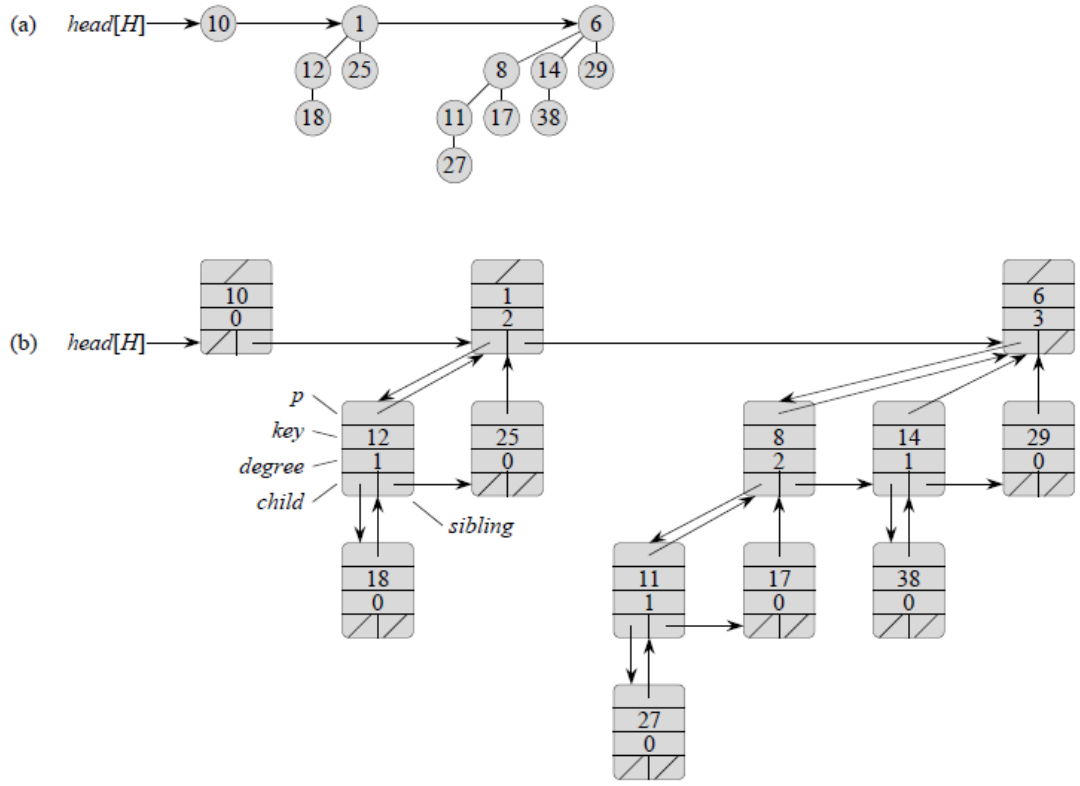
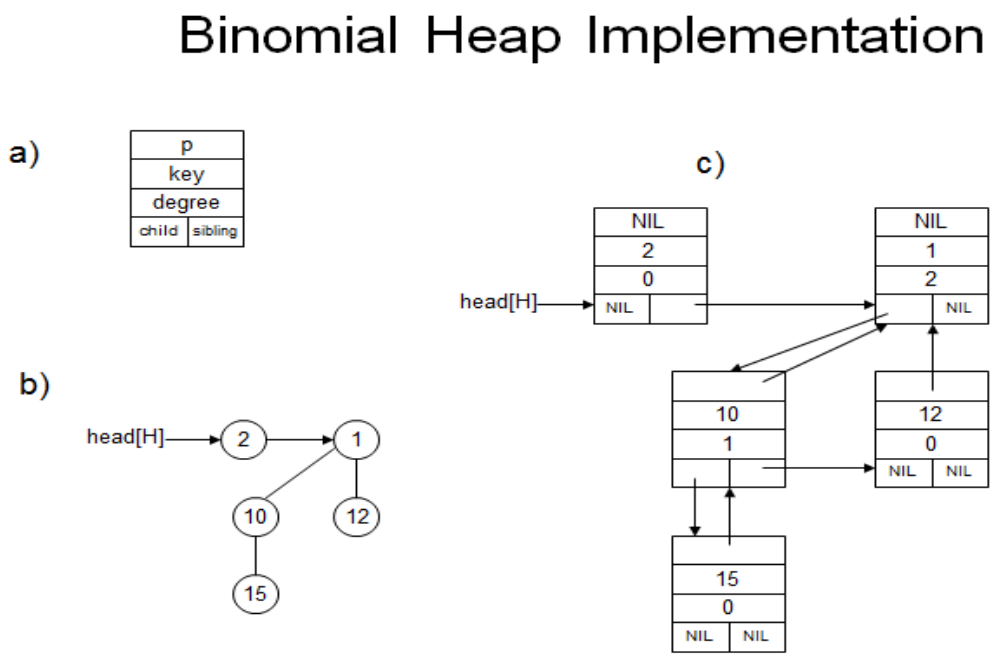



Figure 19.3 A binomial heap H with n = 13 nodes. (a) The heap consists of binomial trees B0, B2, and B3, which have 1, 4, and 8 nodes respectively, totalling n = 13 nodes. Since each binomial tree is min-heap-ordered, the key of any node is no less than the key of its parent. Also shown is the root list, which is a linked list of roots in order of increasing degree. (b) A more detailed representation of binomial heap H. Each binomial tree is stored in the left-child, right-sibling representation, and each node stores its degree.

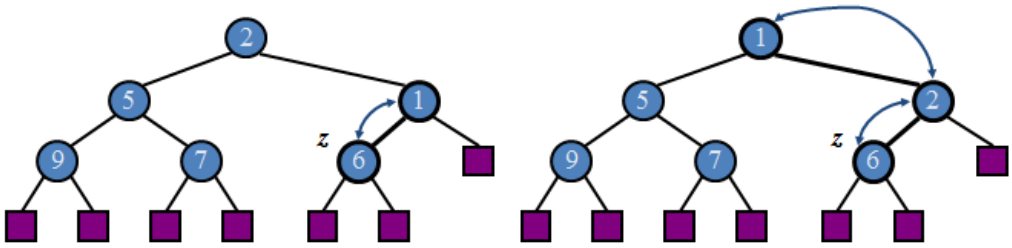


## 6. Explain about Heap UP and Heap Down?

**Answer:**

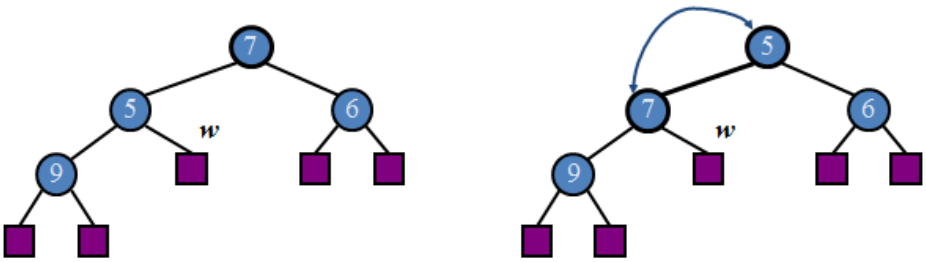
# Heap Up (Percolate up)

- After the insertion of a new key ***k***, the heap-order property may be violated.
- Algorithm heapup restores the heap-order property by swapping ***k*** along an upward path from the insertion node.
- Heapup terminates when the key ***k*** reaches the root or a node whose parent has a key smaller than or equal to ***k***.
- Since a heap has height ***O***(log ***n***), heapup runs in ***O***(log ***n***) time.




# Heap Down (Percolate Down)

- After replacing the root key with the key ***k*** of the last node, the heap-order property may be violated.
- Algorithm heap down restores the heap property by swapping key ***k*** with the child with the smallest key along a downward path from the root.
- Heap down terminates when key ***k*** reaches a leaf or a node whose children have keys greater than or equal to ***k*** .
- Since a heap has height ***O***(log ***n***), heap down runs in ***O***(log ***n***) time.



## 7. Compare Binary Heap and Binomial Queue?

**Answer:**

| Procedure | Binomial Heap | Binary Heap |
|---|---|---|
| Make-Heap | O (1) | O (1) |
| Insert | O (log n) | O (log n) |
| Minimum | O (log n) | O (1) |
| Extract-Min | O (log n) | O (log n) |
| Union | O (log n) | O (n) |
| Decrease-Key | O (log n) | O (log n) |
| Delete | O (log n) | O (log n) |

# Applications of Priority Queues - the Selection Problem

## 1. The problem

Given a list of N elements (here "list" means a sequence, it does not mean ADT list)
and an integer k, $k \leq N$, find the $k^{th}$ largest element.

## 2. Solution 1

Sort the elements in an array and return the element in the $k^{th}$ position.

Complexity:

simple sorting algorithm $O(N^2)$
advanced sorting algorithm O(NlogN)

## 3. Solution 2

Read k elements in an array and sort them. Let $S_k$ be the smallest element. For each next element E do the following:

If $E > S_k$ remove $S_k$ and insert E in the appropriate position in the array
Return $S_k$

Complexity: O(N*k)

$k^2$ for the initial sorting of k elements
(N-k)*k for inserting each next element

$O(k^2)+ O((N-k)*k) = O(k^2 + N*k - k^2) = O(N*k)$

Worst case: k = N/2, complexity: $O(N^2)$

## 4. Solution 3

Assume we change the heap-order property - the highest priority corresponds to the highest key value.

Read N elements in an array
Build a heap of these N elements - O(N)
Perform k DeleteMax operations - O(klogN)

Complexity: O(N) + O(klogN)
For k = N/2 the complexity is O(NlogN)

## 5. Solution 4

We return back to the usual heap-order property – the smallest element is at the top.

Build a heap of k elements. The k-th largest element among k elements is the smallest element in that heap and it will be at the top. Complexity O(k)

Compare each next element with the top element O(1)
If the new element is larger, DeleteMin the top element and insert the new element in the heap. - O(log(k))

At the end of the input the smallest element in the heap is the k-th largest element in the list of N elements.

Complexity: O(k) + O((N-k)*log(k)) = O(Nlog(k))