CSC2001F: Data Structures II

Omowunmi Isafiade

Email: omowunmi.isafiade@uct.ac.za

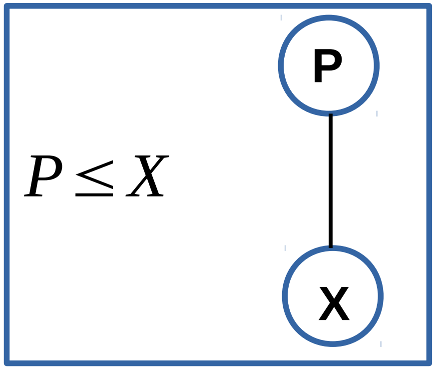
Office: Room 306

Outline

- The Binary Heap
  - Insertions
  - Deletions
- Implementation Considerations...
- Building Binary Heaps (from unsorted to sorted)

Priority Queue – Build Heap Implementation

- The heap-order property allows a priority queue to perform operations quickly
- So it makes sense – use to find min/max quickly
- Heap-order property - "in a heap, for every node *X* with parent *P*, the key in P is never larger than the key in X ($P \leq X$)"



- **Note:** A ***max heap*** supports access to the maximum. Can be implemented with minor changes i.e $P \geq X$

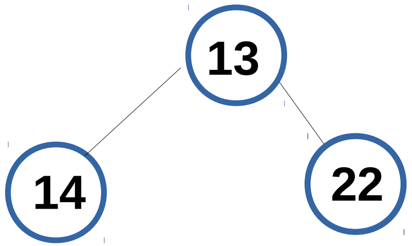
Heap Operations - Insertions

- **Note:** <u>Structure and order properties </u>must always be obeyed
- Methodology:
  - Create a new node in the tree in next available position (to avoid violating structure property – complete binary tree)
  - Check to ensure that ordering property is satisfied

- General Strategy ("***Percolate up"***)
  - Create a hole at the next available location
    - If heap order is not violated, place item in the hole
    - else "bubble-up" the hole toward the root

Heap Operations – Insertion (Example)

- Consider a binary heap formed from the set {14, 13, 22}

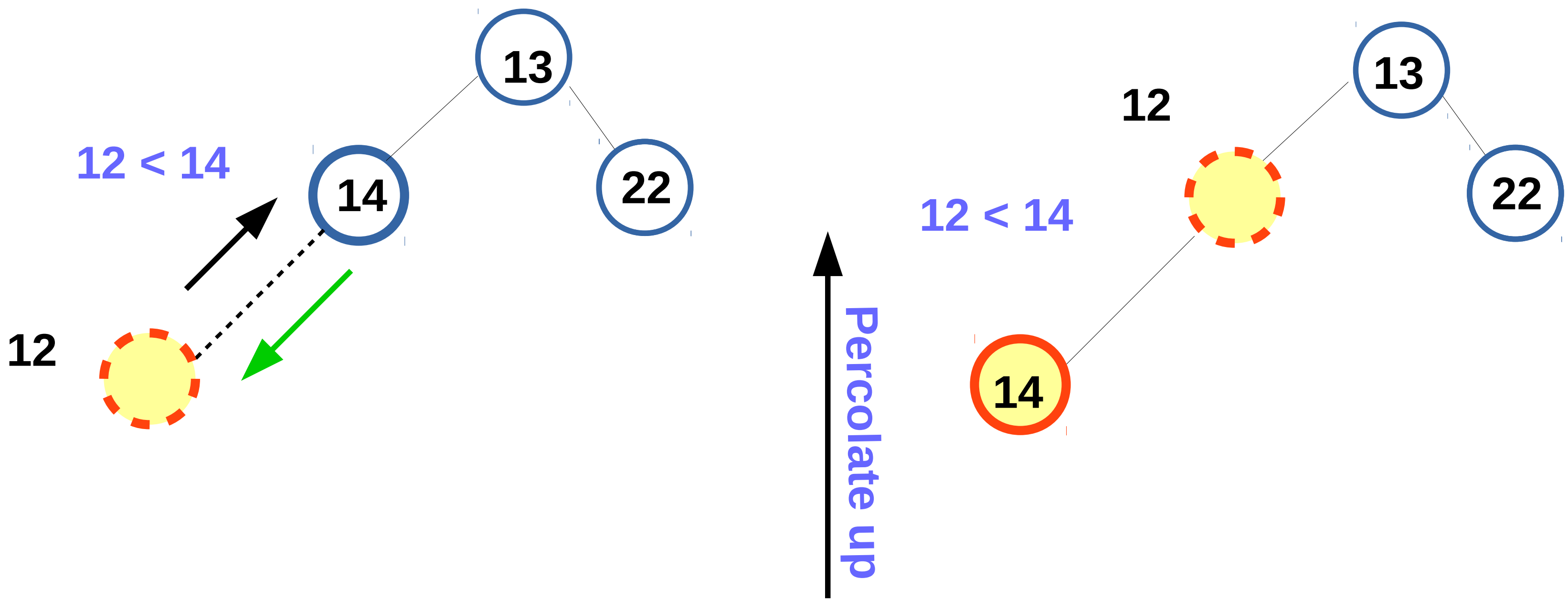


- Insert the elements {12, 11, 10, 20} into the heap above

Exercise Corrections

Note:
*** All operations are aimed at finding a new slot for "12"**
*** Ordering and structure property must be strictly obeyed**

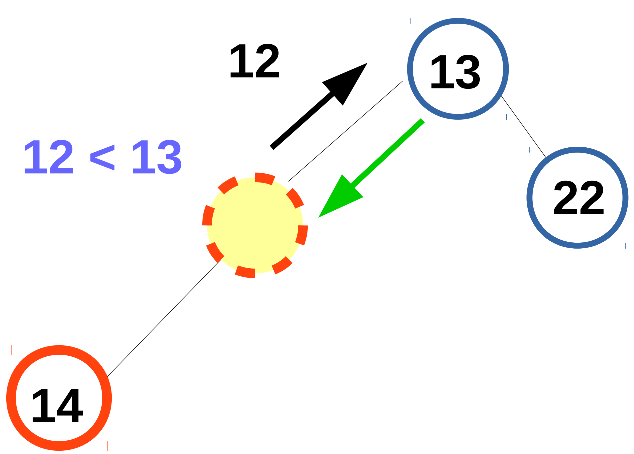
- **Case 1**: Inserting 12
- Step 1: add a node (hole) at next available location
- Step 2: compare "12" to immediate parent node
- Step 3: bubble up as necessary (until correct location is found)

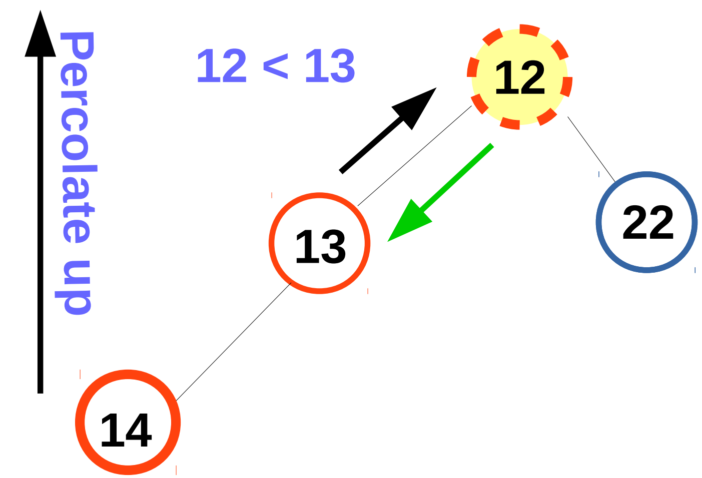


Exercise Corrections

Note:
*** All operations are aimed at finding a new slot for "12"**
*** <u>Ordering and structure property</u> must be strictly obeyed**

- **Case 1**: Inserting 12
- Step 4: compare "12" to immediate parent node
- Step 5: bubble up as necessary



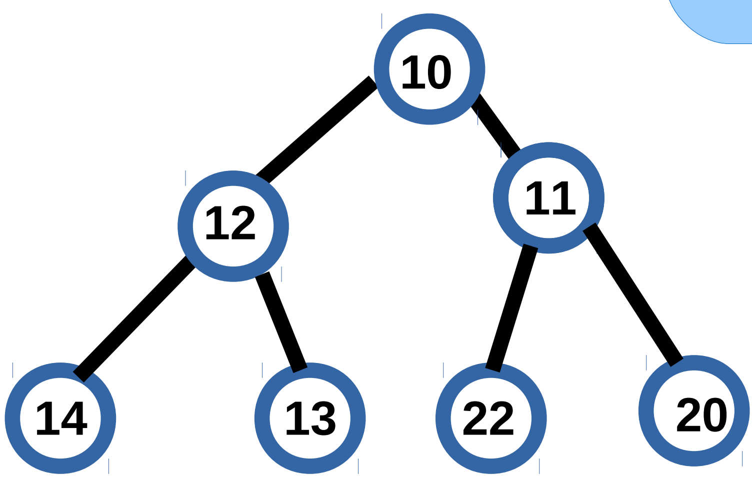


12 is finally at the correct position.
<u>Note</u>: structure and order properties obeyed

Exercise Corrections

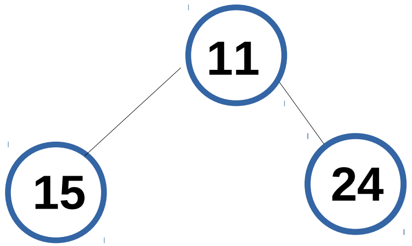
- Repeat the procedure for other items
- Resulting binary heap...

Note:
*** All operations are aimed at finding a new slot for "an item"**
*** Ordering and structure property must be strictly obeyed**



Exercise in Class – Insertion Operation

- Consider a binary heap formed from the set {15, 11, 24}

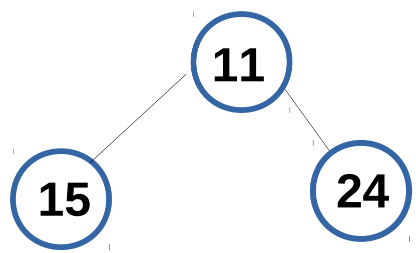


Note:
*** All operations are aimed at finding a new slot for "an item"**
*** Ordering and structure property must be strictly obeyed**

- Insert the elements {14, 10, 17, 20, 19} into the heap above

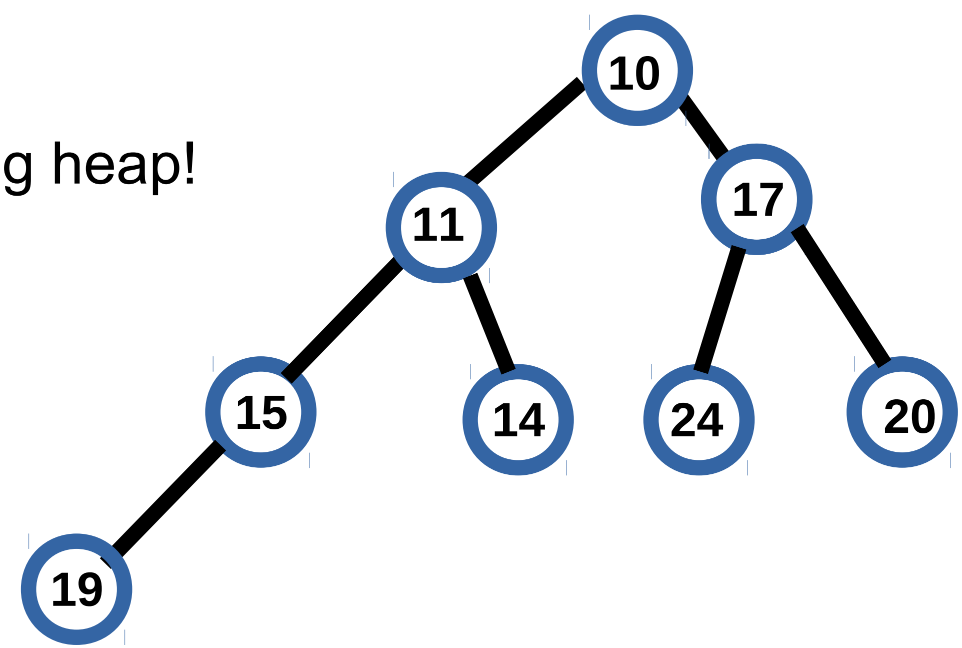
Exercise Solution – Insertion Operation

- Consider a binary heap formed from the set {15, 11, 24}
- Insert the elements {14, 10, 17, 20, 19} into the heap above



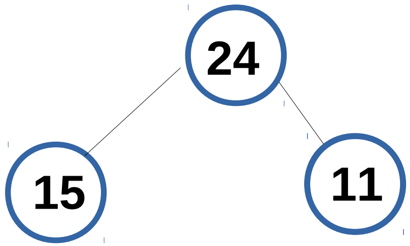
Note:
*** All operations are aimed at finding a new slot for "an item"**
*** Ordering and structure property must be strictly obeyed**

- Resulting heap!



Exercise in Class – Insertion Operation

- Consider a binary heap formed from the set {15, 24, 11}

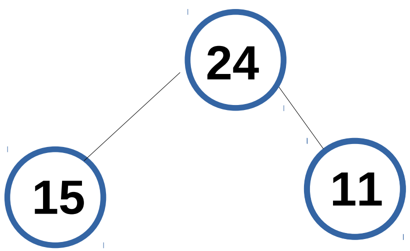


Note:
*** All operations are aimed at finding a new slot for "an item"**
*** Ordering and structure property must be strictly obeyed**

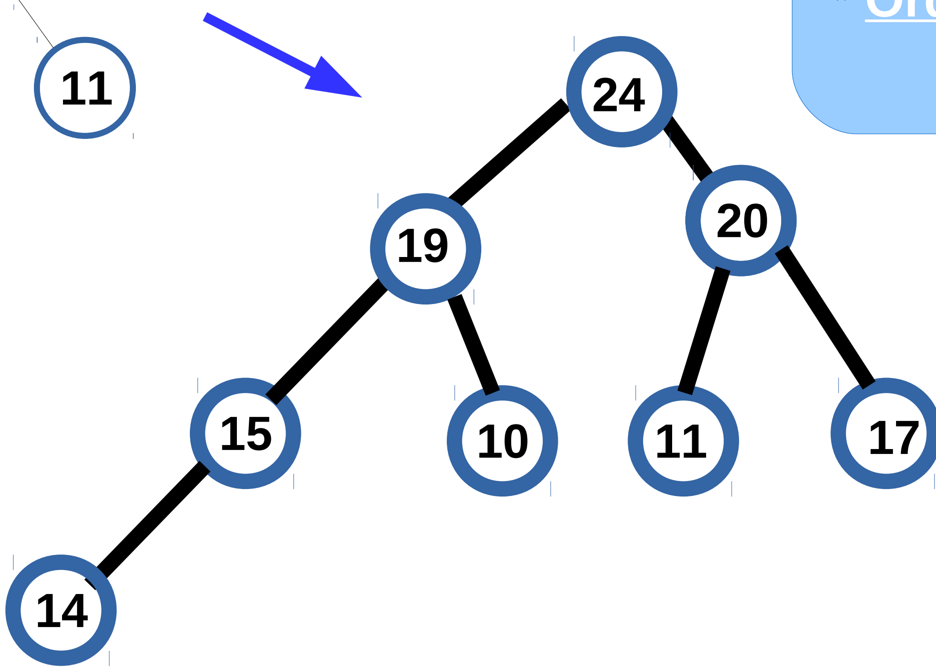
- Insert the elements {14, 10, 17, 20, 19} into the heap above
- Note: assume **max-heap (I.e maximum element @ root node)**

Exercise in Class – Insertion Operation

- Consider a binary heap formed from the set {15, 24, 11}
- Insert the elements {14, 10, 17, 20, 19} into the heap above
- **Assume max-heap** (P >= X)



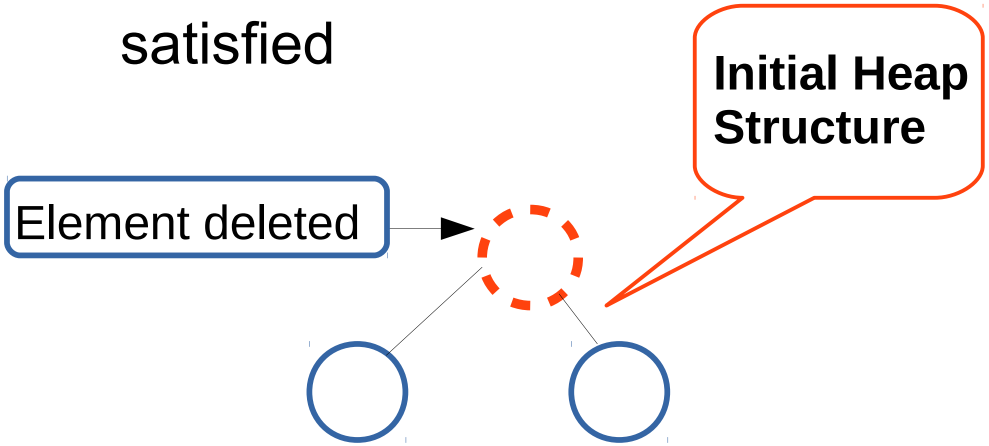
Note:
*** All operations are aimed at finding a new slot for "an item"**
*** Ordering and structure property must be strictly obeyed**

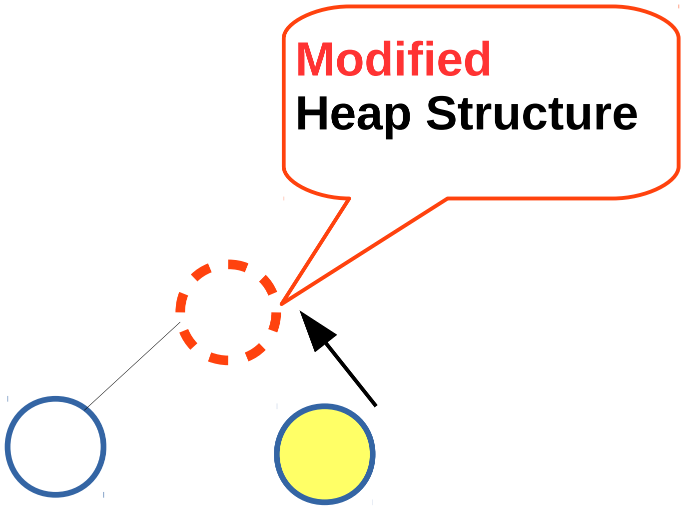


- Result!

Heap Operations – Deletions

- Easy to find “min” (@ root)
- But!!! deleting “min” creates a hole at the root.
  - Heap shrinks by 1 (find a new slot for last item @ bottom level)
- Restructuring Principle:
  - Re-order items to ensure structure and ordering properties are satisfied

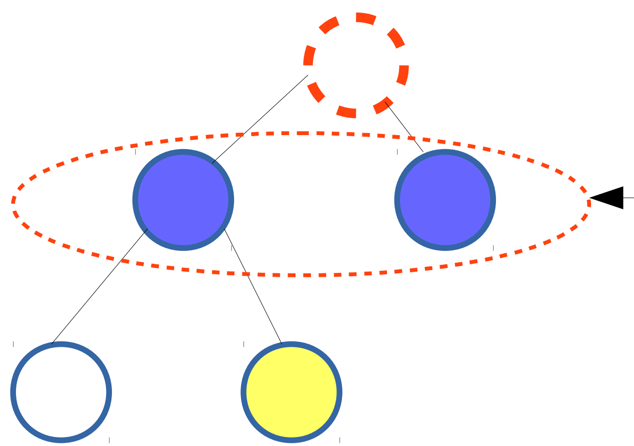




Heap Operations – Deletions

- Restructuring Principle (“***percolate down***”):
  - Re-order items to ensure **structure** and **ordering** properties are satisfied.
  - Comparison is between yellow and blue nodes

2. If the smaller of the blue nodes is <= the yellow node, It is moved up to the empty slot (root) and the empty slot moved down



1. **Modified heap structure**: Compare **nodes at next level (blue nodes)** to decide which one is smaller than “yellow node”

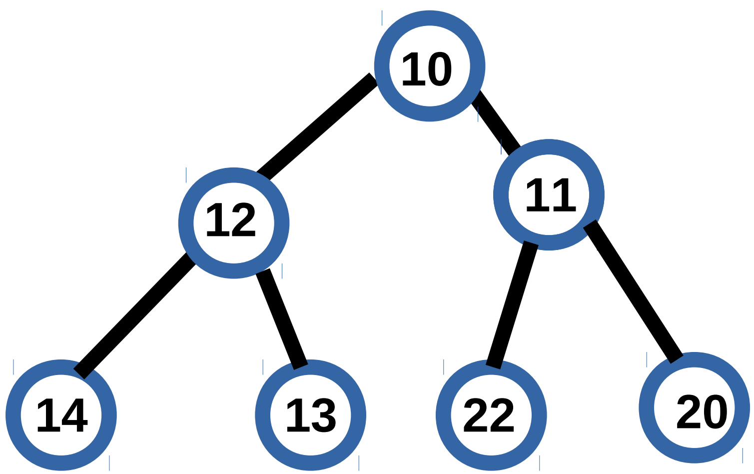
3. The procedure is repeated until the item (yellow node) can be correctly placed – a Process called percolate down

Heap Operations - Deletions

- Restructuring Principle:
  - Re - order items to ensure structure and ordering properties are obeyed
  - Exercise: Delete the items {10, 11, 12, 13, 14} from the binary heap hereafter
    - <u>Think about how you would re-organise the heap to obey both the structuring and ordering properties!</u>

Heap Operations – Deletions Example

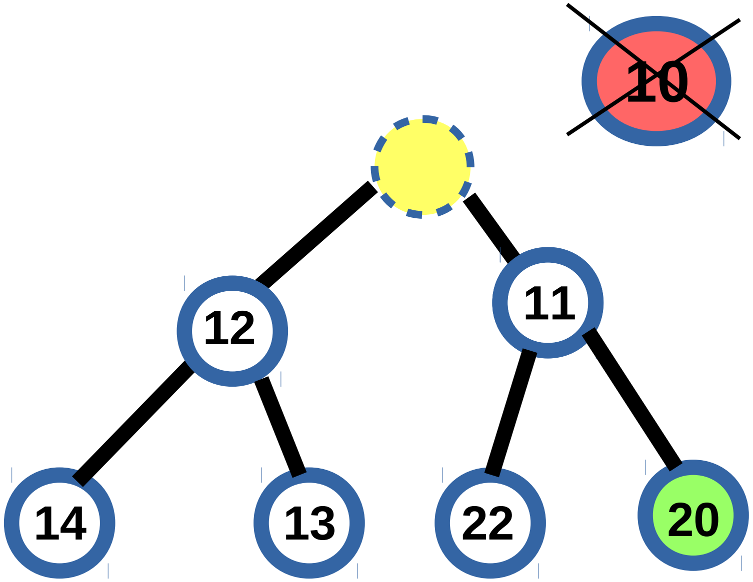
- Restructuring Principle (***Percolate down***):
  - Re-order items to ensure **<u>structure</u>** and **<u>ordering</u>** properties are satisfied.
- Exercise: Delete {10, 11, 12, 13, 14} from the binary heap below



  - Be sure to re-organise the heap to obey both the structuring and ordering properties!

Heap Operations – Deletions Example

- **Step 1**: Deleting 10 ...



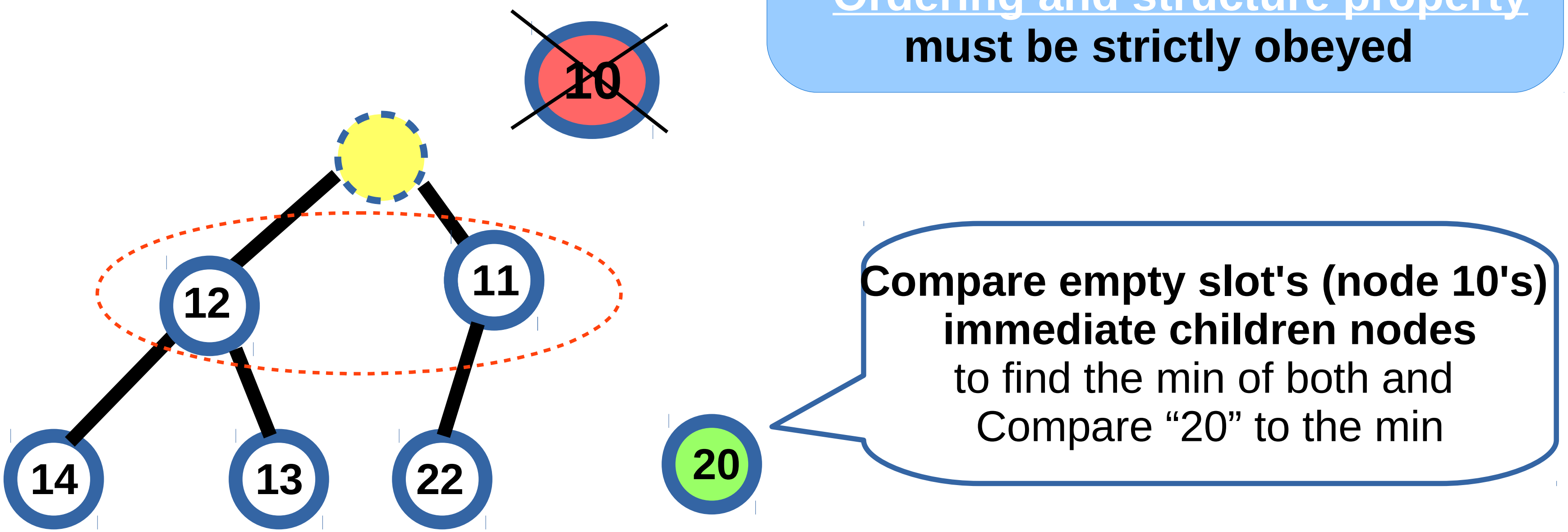
Note:
*** All operations are aimed at finding a new slot for "20"**
*** Ordering and structure property must be strictly obeyed**

- Find new position for **"20"**

Heap Operations – Deletions Example

- **Step 2**: Deleting 10 ...

Note:
* **All operations are aimed at finding a new slot for “20”**
* **Ordering and structure property must be strictly obeyed**

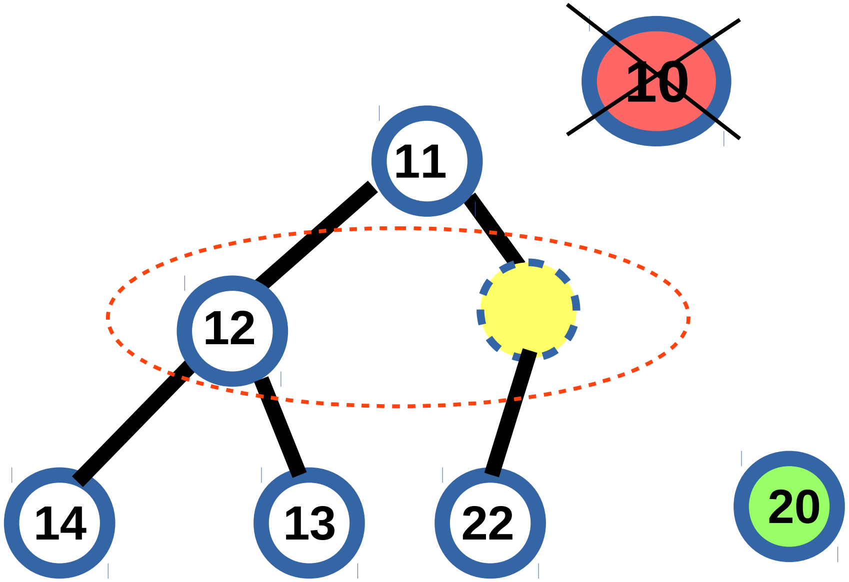


- Find new position for **“20”**

Heap Operations – Deletions Example

- **Step 3**: Deleting 10 ...

Note:
*** All operations are aimed at finding a new slot for "20"**
*** Ordering and structure property must be strictly obeyed**

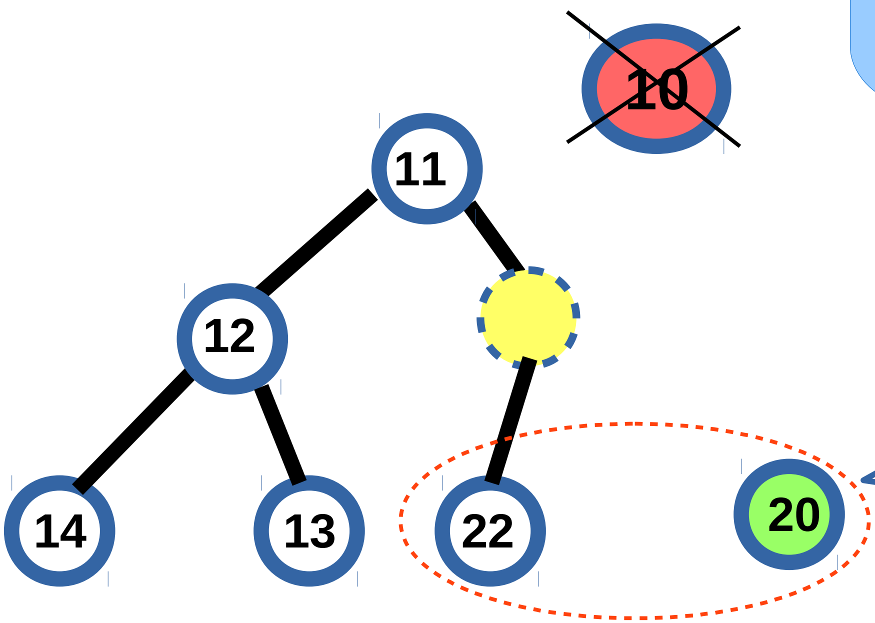


11 < 12 and also less than 20.
So, 11 is moved to the "hole",
pushing the hole down one level

- Find new position for **"20"**

Heap Operations – Deletions Example

- **Step 4**: Deleting 10 ...



Note:
*** All operations are aimed at finding a new slot for "20"**
*** Ordering and structure property must be strictly obeyed**

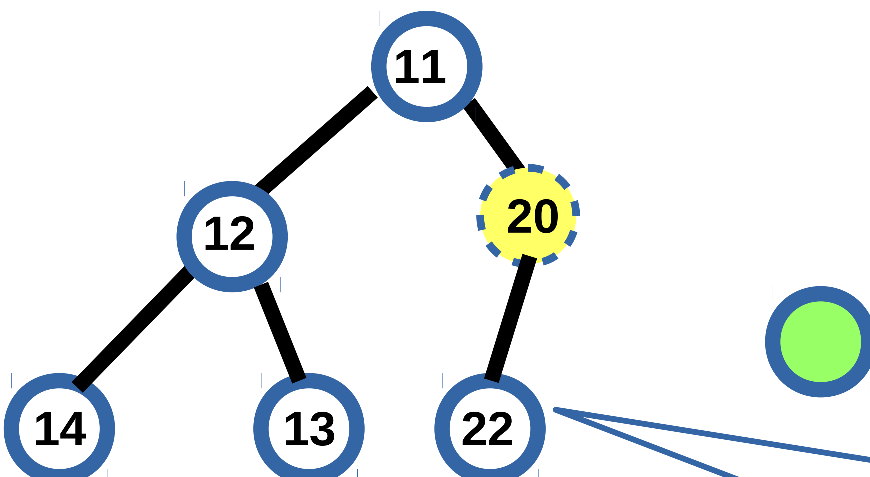
Now compare the **children nodes of the current empty slot** to find the min. **20< 22**, so 20 moves into the empty position

- Find new position for **"20"**

Heap Operations – Deletions Example

Note:
* **All operations are aimed at finding a new slot for "20"**
* **Ordering and structure property must be strictly obeyed**

- **Step 4**: Deleting 10 ...

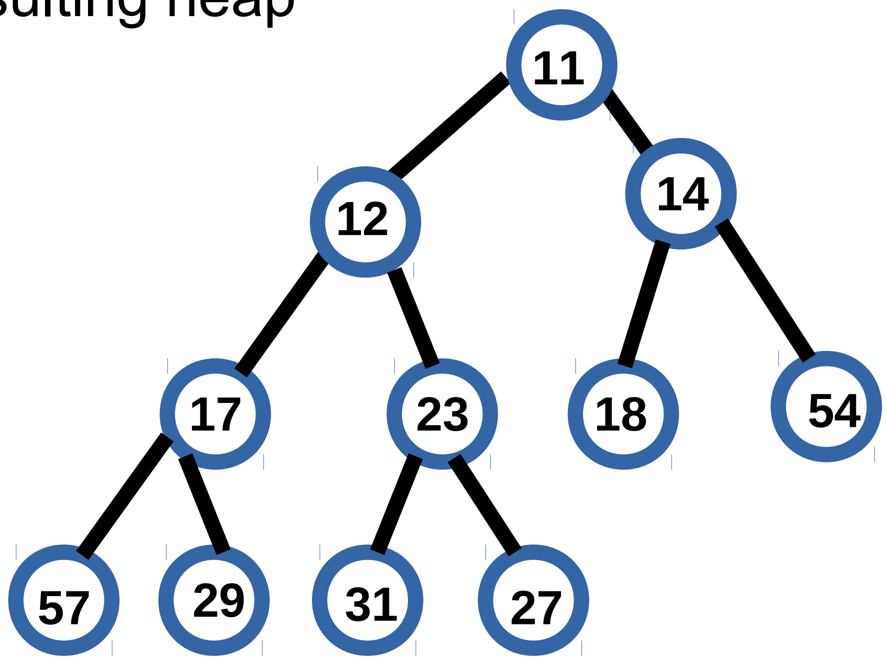


Final binary heap: satisfying both Ordering and structure property

- Find new position for **"20"**

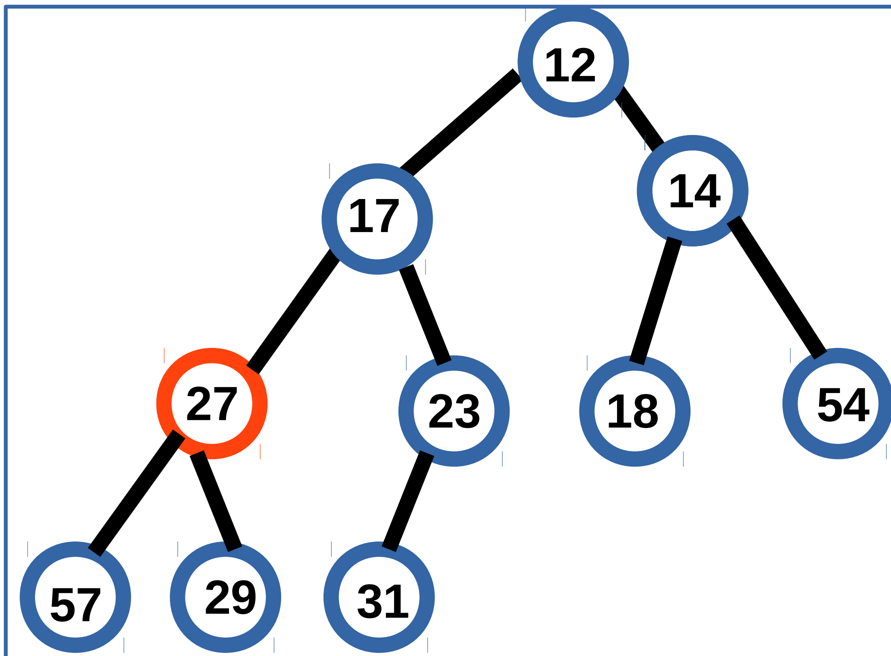
Binary Heap – Deletion Exercise

- Exercise: Delete "11" and "12" from the binary heap below and show the resulting heap

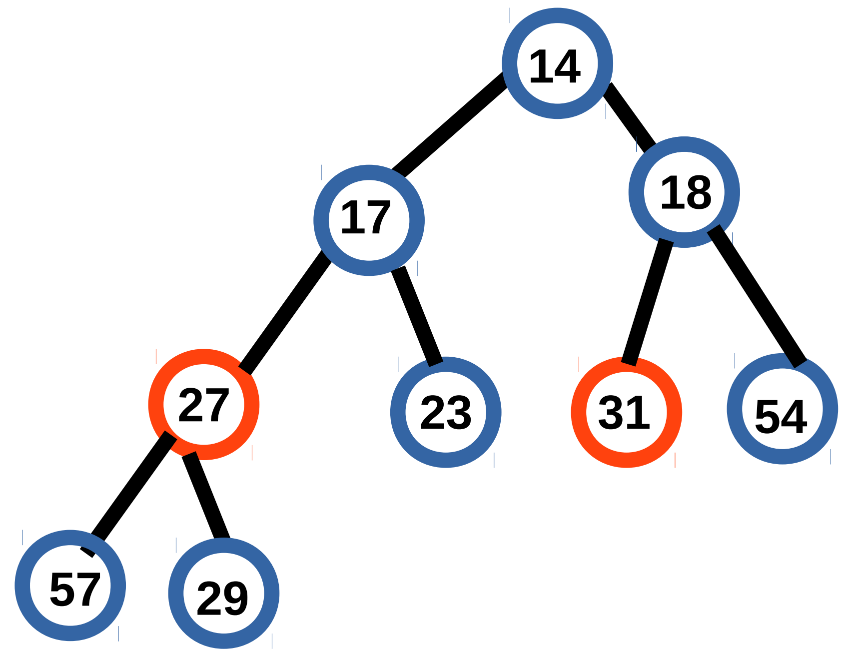


Binary Heap – Deletion Exercise (Solution)

- If you follow the procedure you will eventually get the following:



After deleting 11



After deleting 12

__Note__: Structure and ordering property must be maintained at all times.

Binary Heaps – Some Considerations

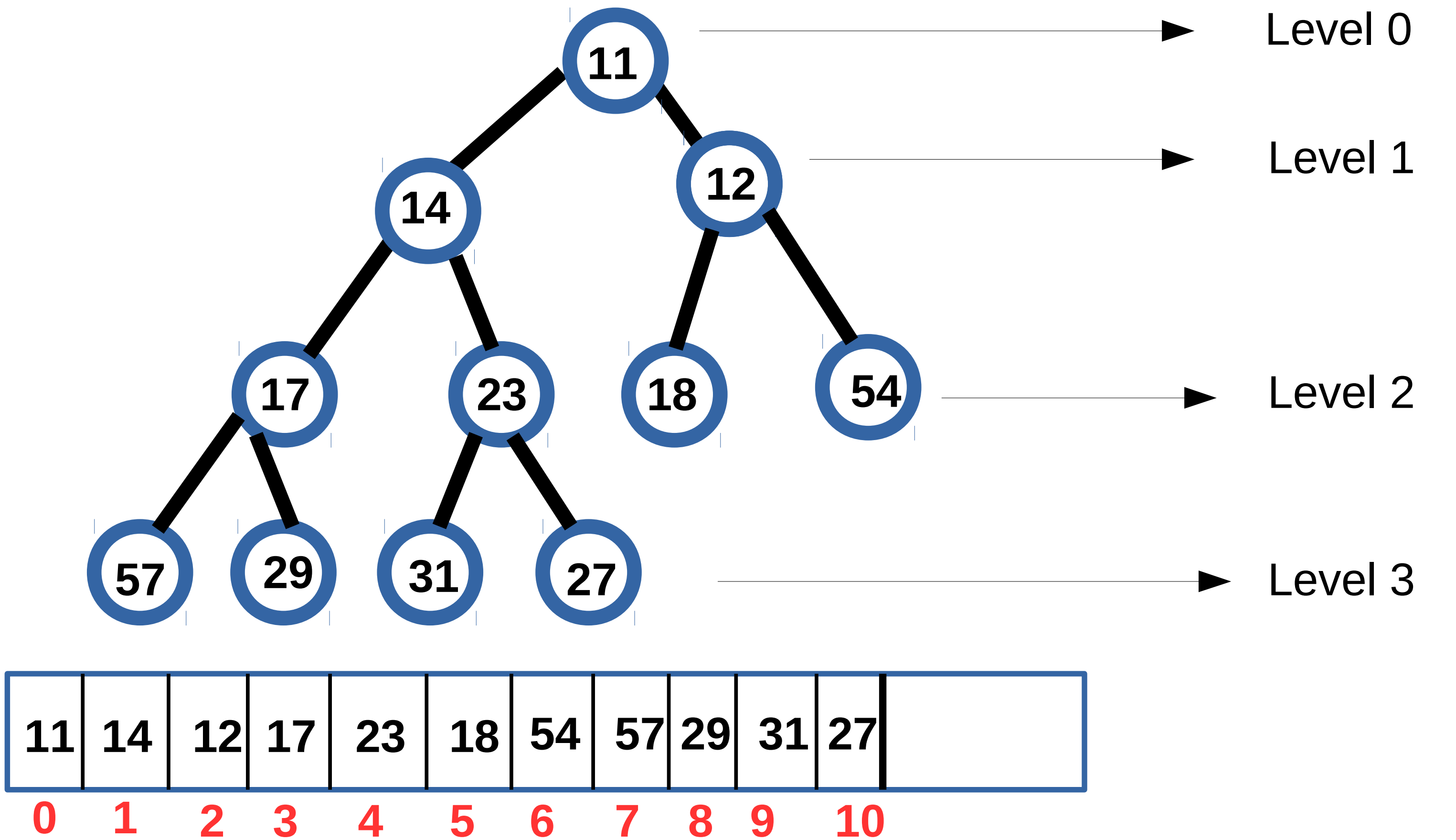
- A priority queue is not a heap (or a binary heap)
- Priority queue is an abstract concept like a list
- A list can be implemented as a linked list or an array
- Likewise – a heap is just a (classical) method of implementing the concept of a priority queue

Binary Heaps – Some Considerations

- An array can be used to store a tree (e.g instead of doing it with a linked list)
- Advantage:
  - No child links required
  - Operations required to traverse tree are simple to implement and efficient (performance)
- Disadvantage:
  - Dynamic adjustments of table size can become expensive

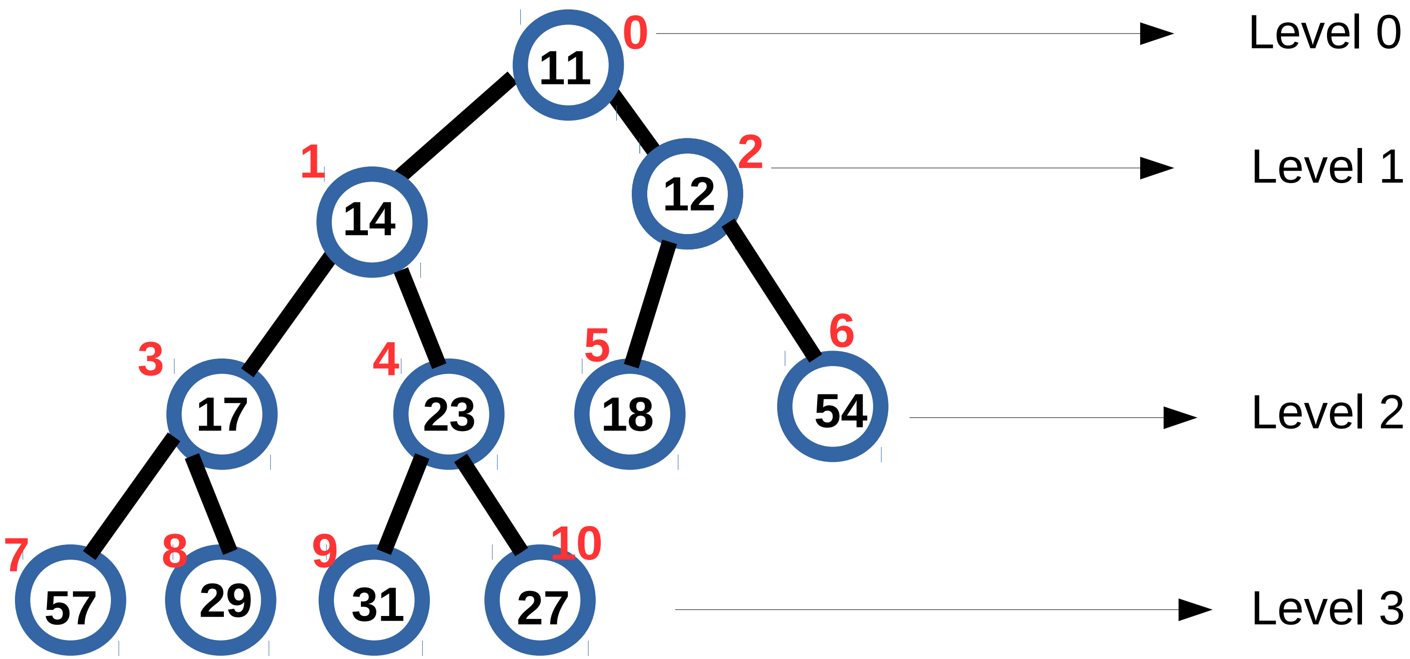
Binary Heap – Implementation Considerations

- Example: The binary heap below can be represented using



Binary Heap – Implementation Considerations

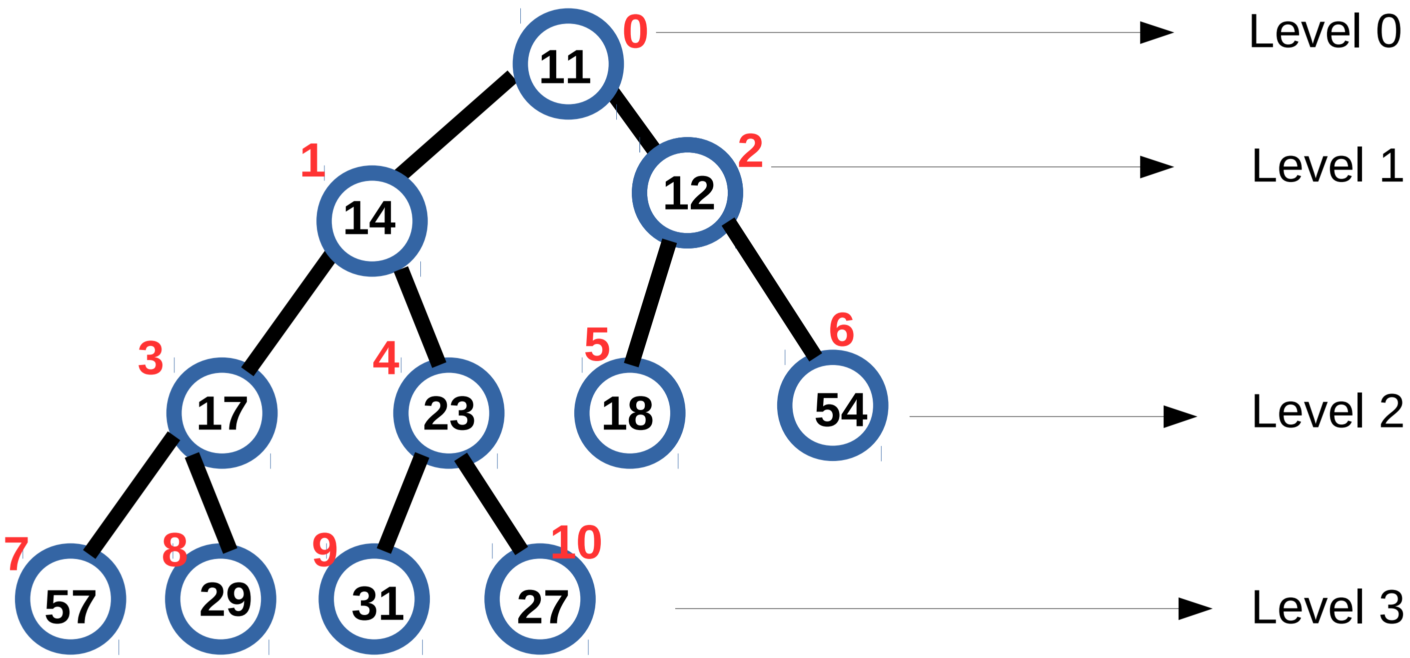
- Example: The binary heap below can be represented using an array



11	14	12	17	23	18	54	57	29	31	27	
0	1	2	3	4	5	6	7	8	9	10	

Note (root node at index 0):
*** For every node n in position "i" the left child is at position 2i +1**
*** Similarly, the right child is at position 2i + 2**
(Provided they are internal nodes)

Binary Heap – Implementation Considerations



11	14	12	17	23	18	54	57	29	31	27
0	1	2	3	4	5	6	7	8	9	10

Note:

*** Root is labelled 0 and for every node n with position i > 0, the parent of n is at position** $\lfloor ((i-1)/2) \rfloor$

*** Items of the heap are stored in an array of size H and in this case the last node is at position H-1**

Binary Heaps – Build Heap Operation

- **Goal**: Take a binary heap that violates the heap order and reinstates it
- **Advantage**:
  - Reduces the cost of insertions from O(N log N) to O(N)
- Recall: An insertion takes O(log N) time (particularly if new element to be inserted is new “min”)
  - Implies N insertions take O(N log N)
- Insertions becomes costly since heap order must be maintained after every insertion

Binary Heaps – Build Heap Operation

- **Goal**: Take a binary heap that violates the heap order and reinstate it

- Next Lecture: More on building a binary heap!