# UCSC Silicon Valley Extension

## Advanced C Programming

### Binary Heaps

Radhika Grover

# Priority queues (Heaps)

- Support at least two operations:
  - insert
  - deleteMin : finds, returns and removes the smallest element in queue

# Priority queue implementations

- Linked list (with insertions at front)
  - insert: O(1), deleteMin: O(N)
- Binary search tree  (deletions can make tree unbalanced)
  - insert (average): O(log N), deleteMin (average): O(log N)
- Arrays
  - insert: O(log N), deleteMin: O(log N)

# Types of heaps

- Binary
- Binomial
- Fibonacci

and others

# Binary and Binomial heap

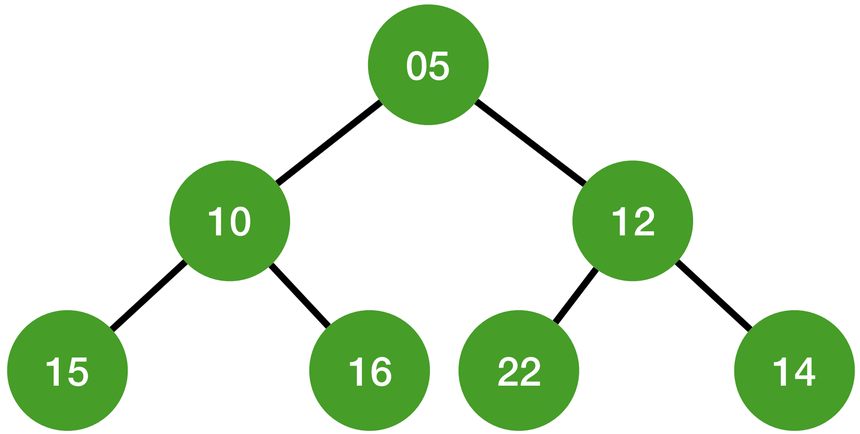
- Operations supported :
  - Create, Delete, Insert
  - Find minimum element
  - Extract  minimum element
  - Union of two heaps
  - Decrease key

# Applications

- Dijkstra’s (shortest path)
- Prim’s (minimum spanning tree)
- Huffman encoding
- Heapsort
- Greedy algorithms

# Binary heap

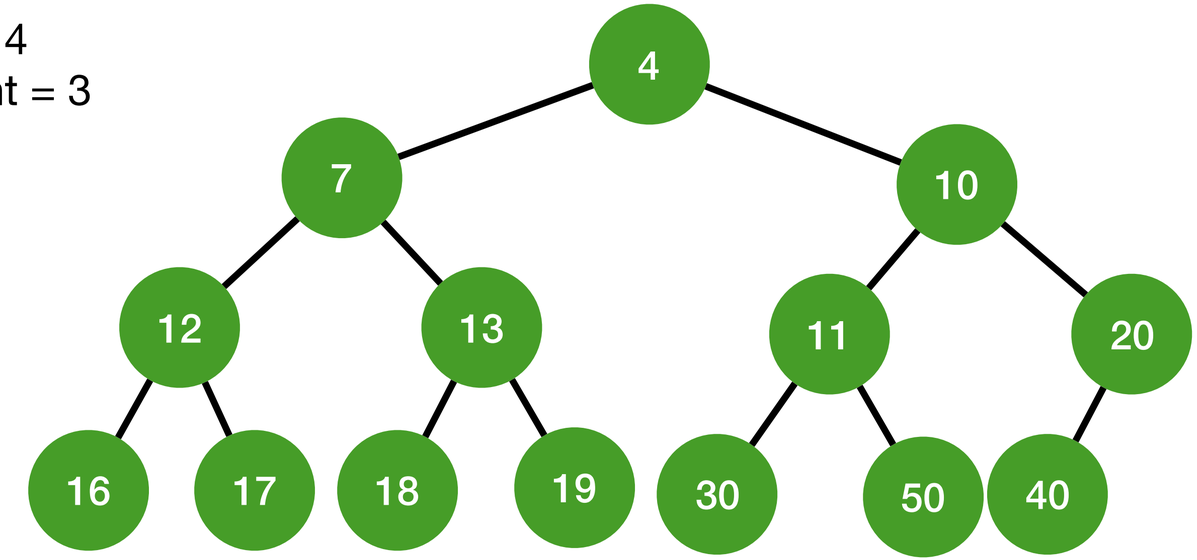
- Binary tree with node
- Every node has a key
- Min-heap : child has a key greater than or equal to parent’s key
- Max-heap : child has key less than or equal to parent’s key



# Binary heap properties

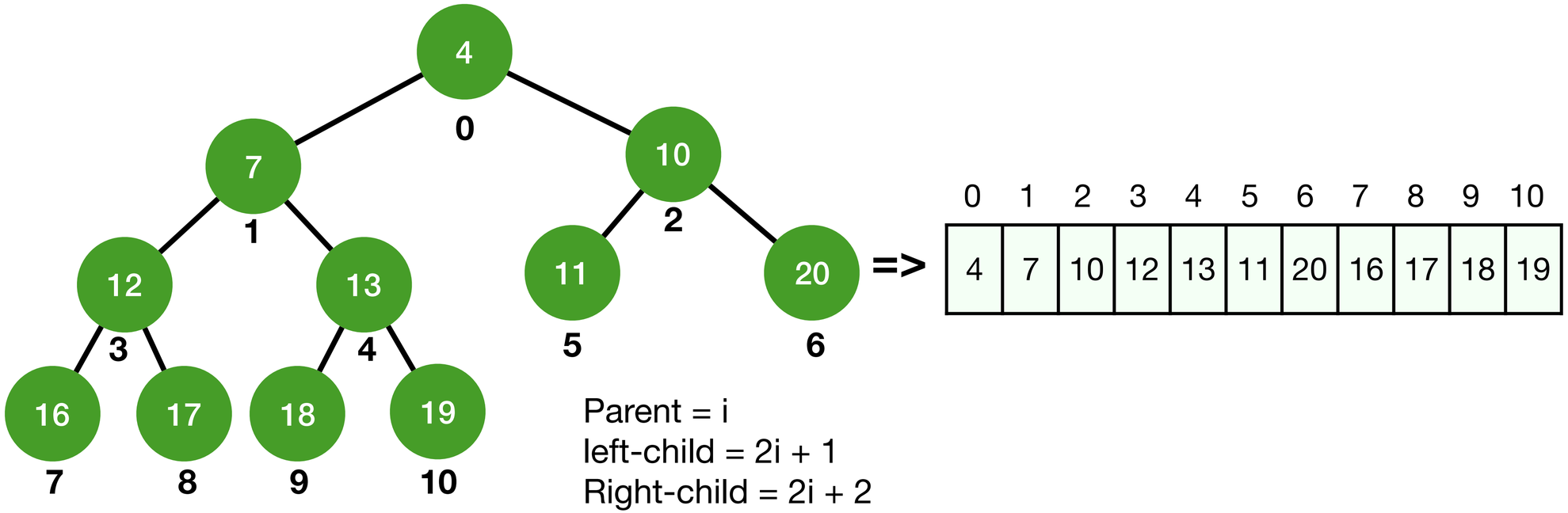
- Smallest min-heap key is in root node
- Height is $\lfloor \log_2 N \rfloor$, N = number of nodes

N = 14
height = 3

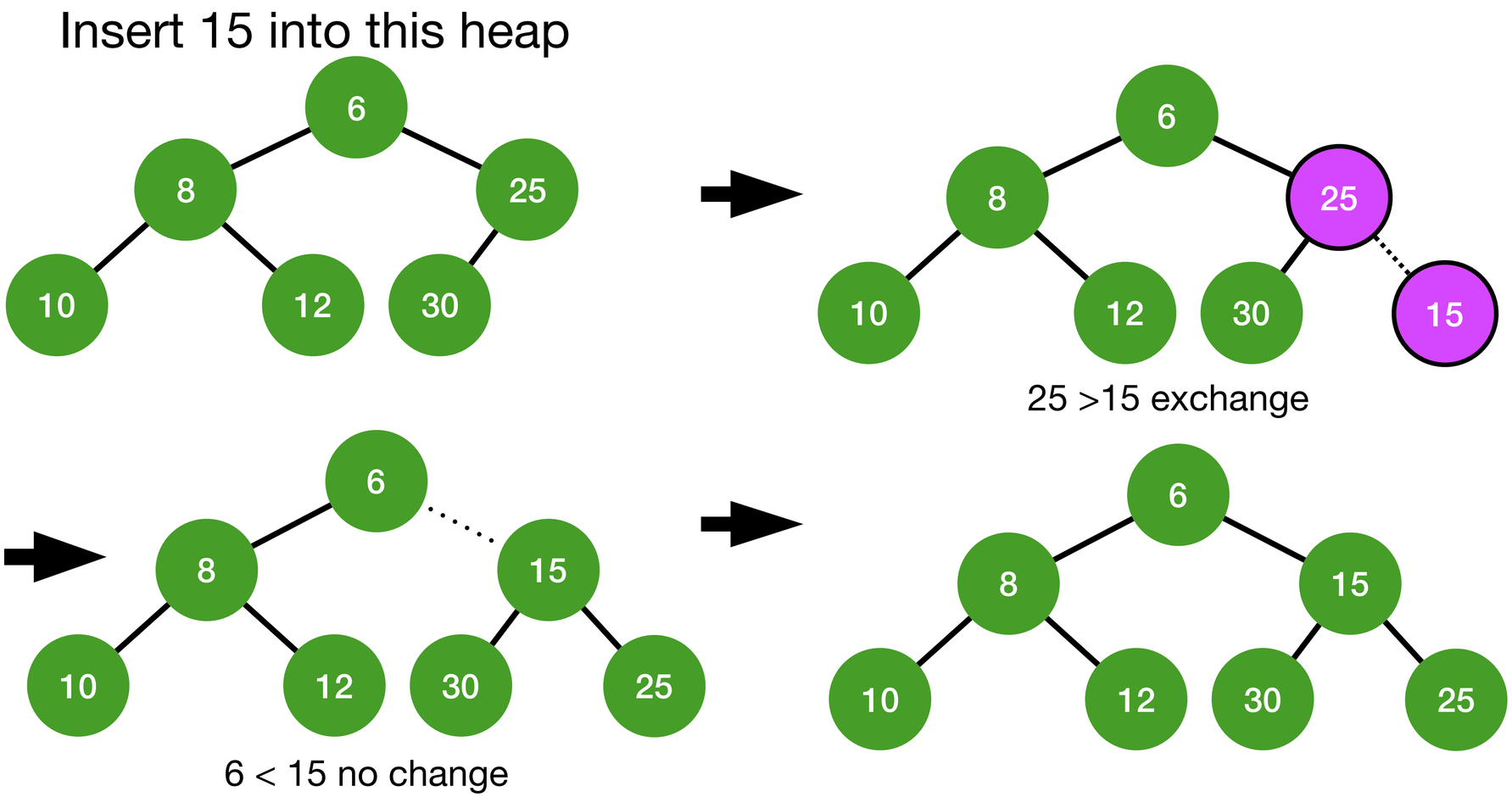


# Binary heap : arrays

Store in an array



# Binary heap : insert



# Binary heap : insert

1. Add element to last available slot

2. Compare element with parent

3. If the order is not correct, swap and go to step 2, otherwise stop

worst-case $O(\log_2 n)$

Average case O(1)

# Binary heap : decrease key

1. Decrease key of element at index i
2. Compare element with parent
3. If order is incorrect swap and go to step 2, otherwise stop

Time complexity : $O(\log_2 N)$

# Binary heap : delete minimum key

- In min-heap (max-heap) the minimum (maximum) key is at root

    1. Replace root with rightmost leaf

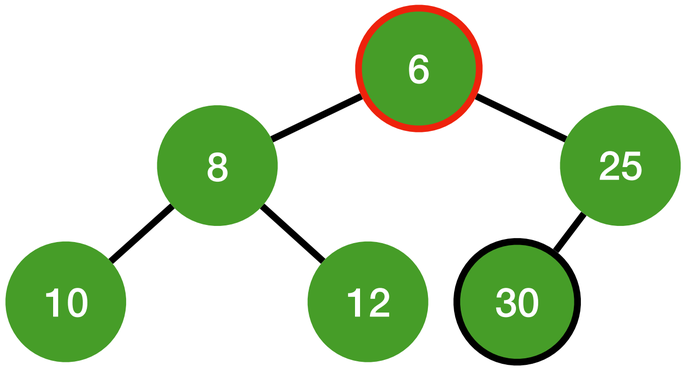
    2. Compare new root with its children

    3. If order is incorrect, swap with smaller (larger) child in a min-heap(max-heap); otherwise, stop

    4. Bubble root down by repeating step 3

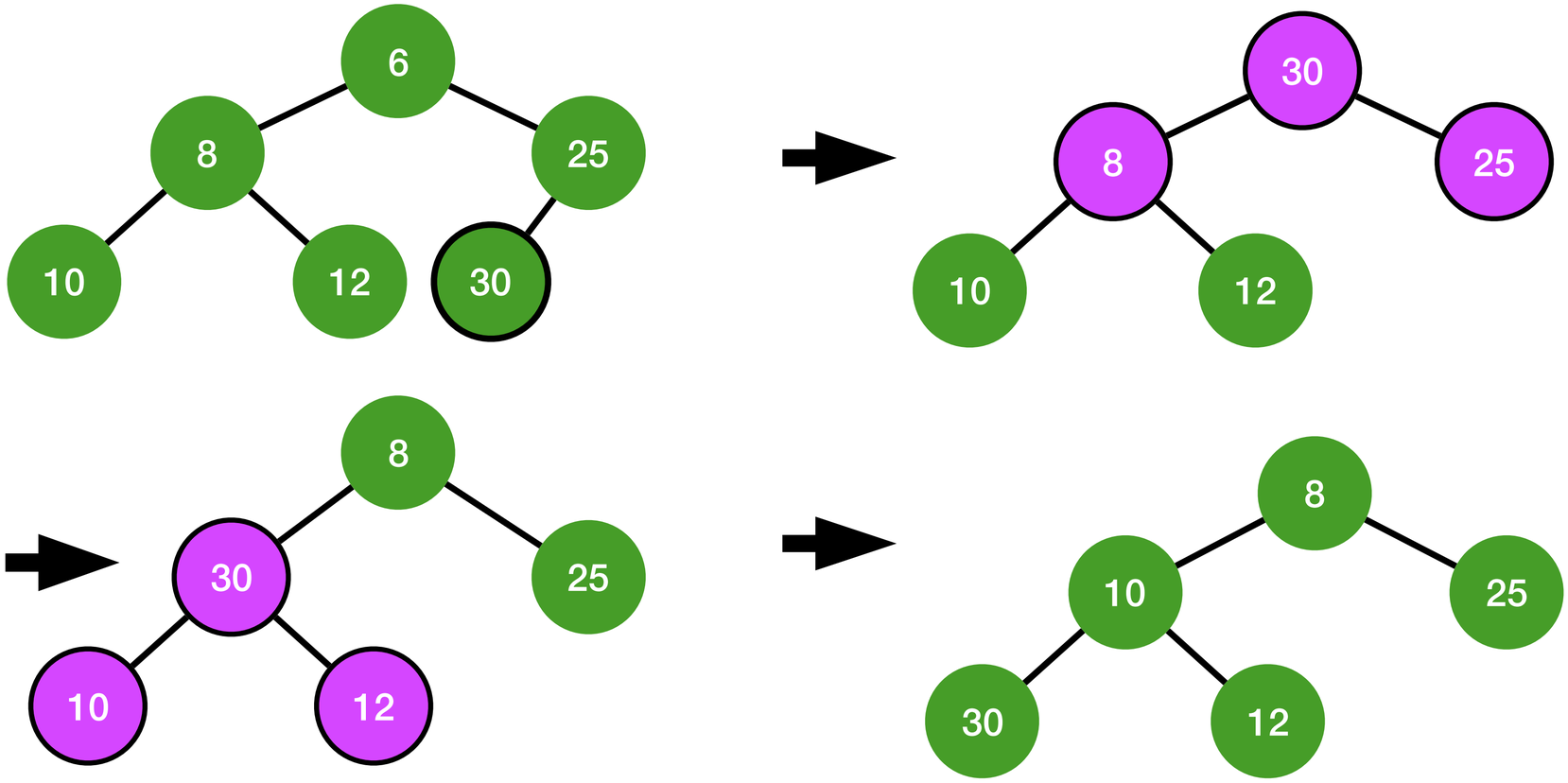
Time complexity : $O(\log_2 N)$

# Delete min




| 6 | 8 | 25 | 10 | 12 | 30 | |
|---|---|---|---|---|---|---|

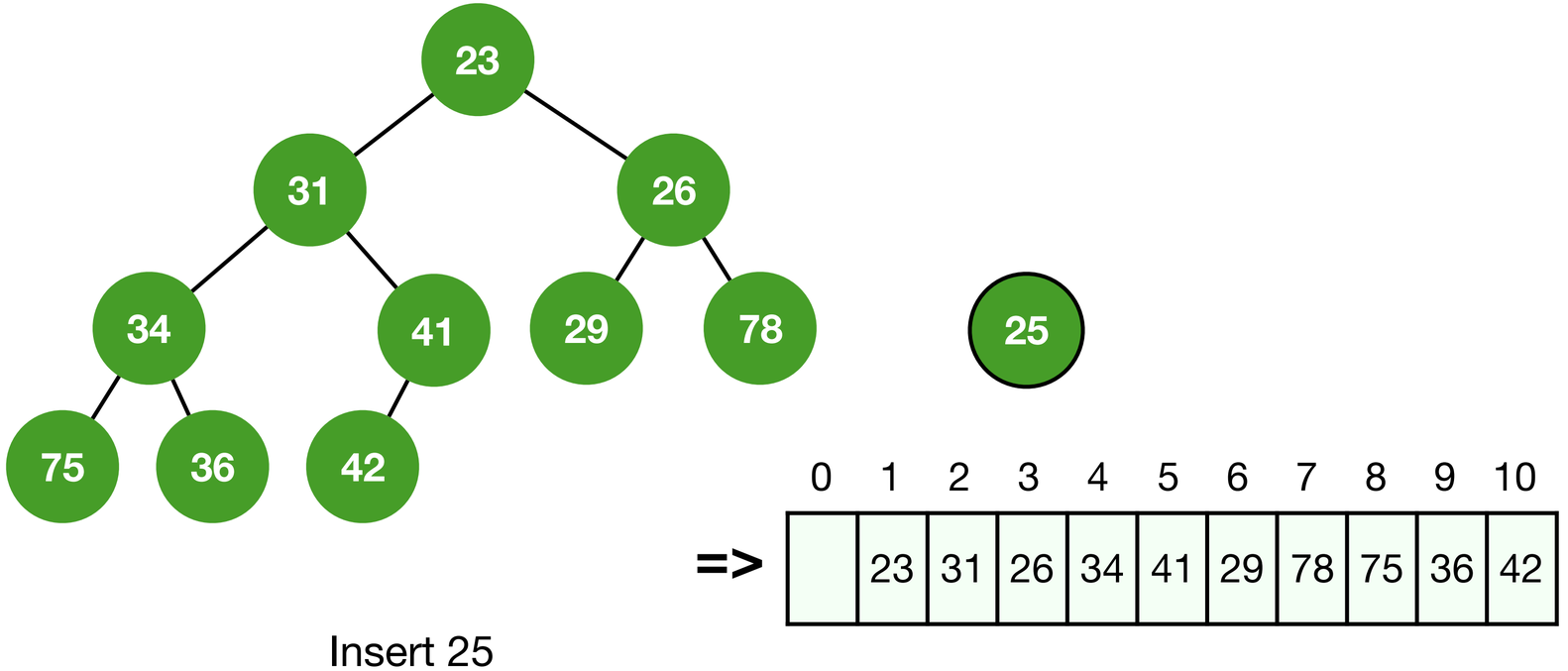
# Binary heap : delete minimum key



# Improved binary heap

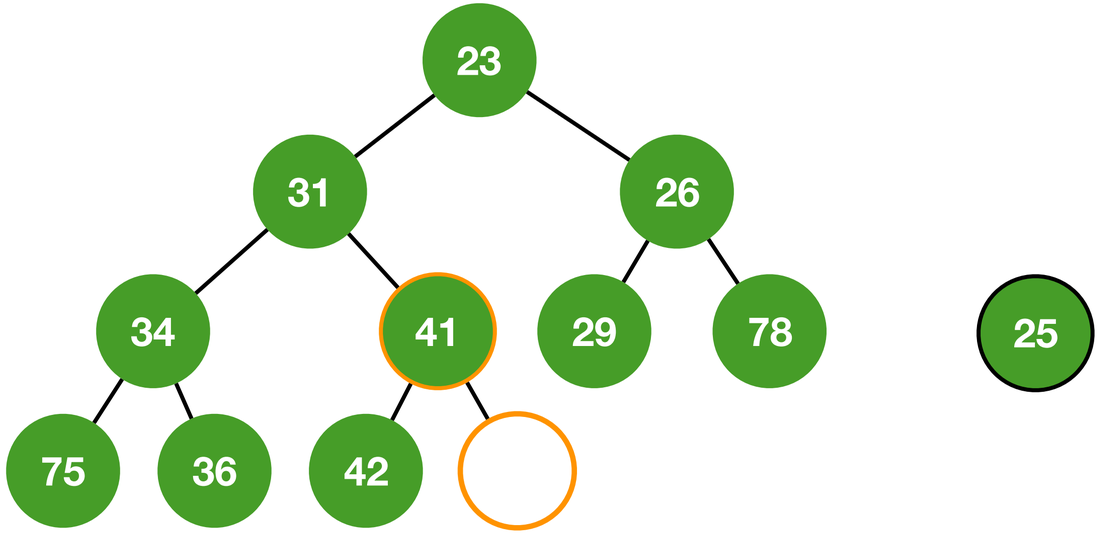
- *Swap* uses 3 assignments
- *Percolate up* or *percolate down* uses only 1 assignment
- Implement binary heap with percolate up/down operations.
- Store a sentinel at index 0, start the heap at index 1 - easier to program.

# Heap insert



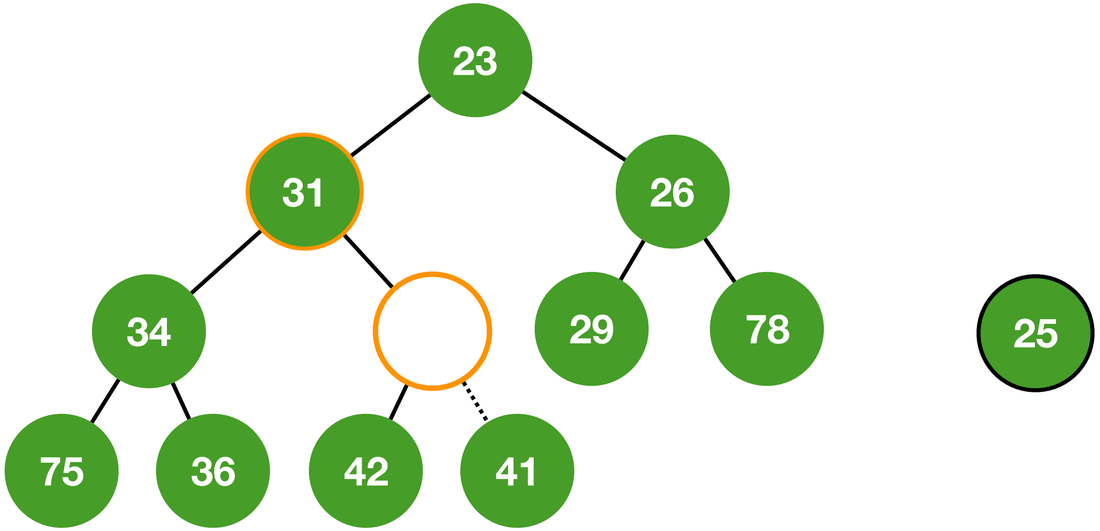

Insert 25

# Heap insert



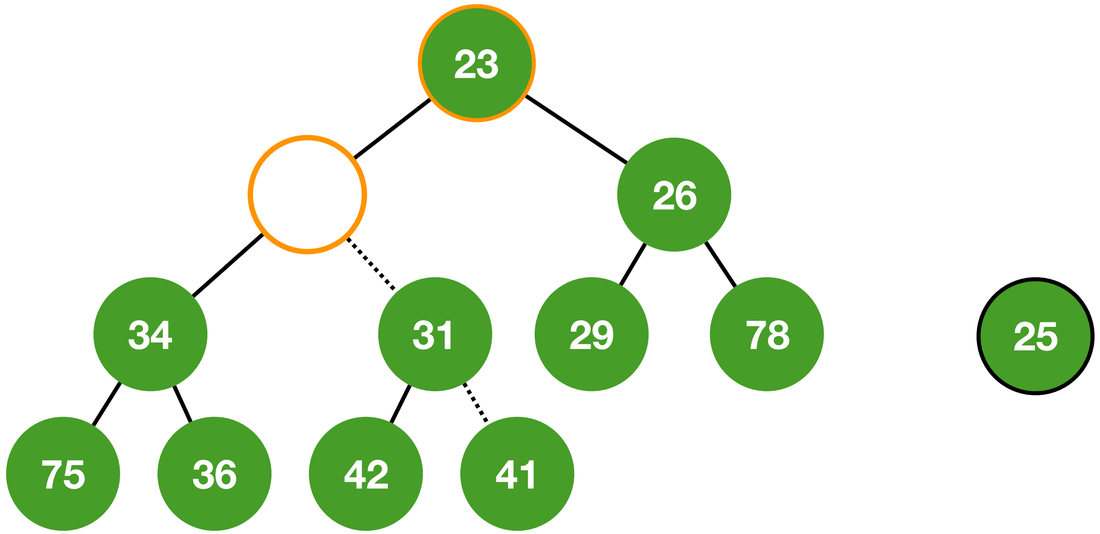

Insert 25 : Percolate up the heap until the correct location is found

# Heap insert



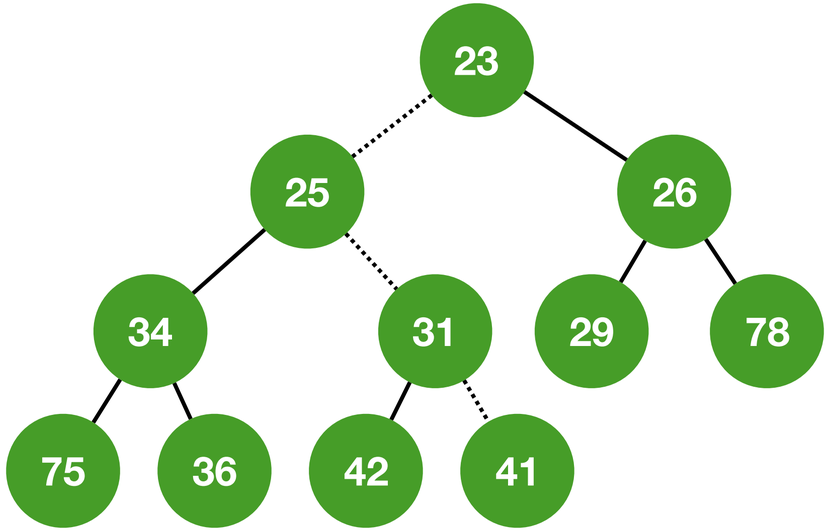

Check order : 41 < 25 ? No, move 41 down

# Heap insert



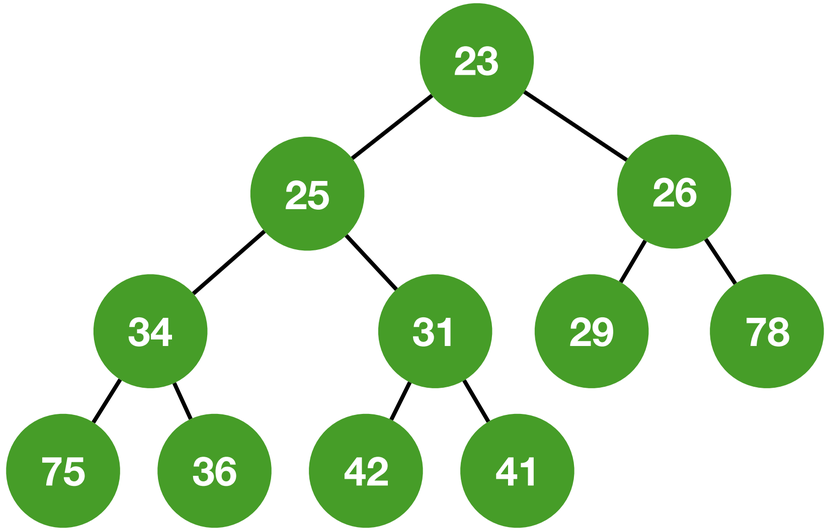

Check order : 31 < 25 ? No, move 31 down

# Heap insert

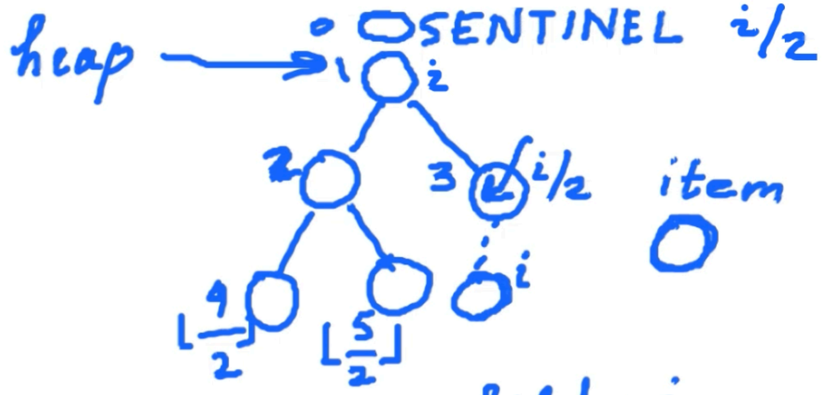



Check order: 23 < 25 ? Yes, insert 25

# Heap insert solution



# Pseudocode for heap insert

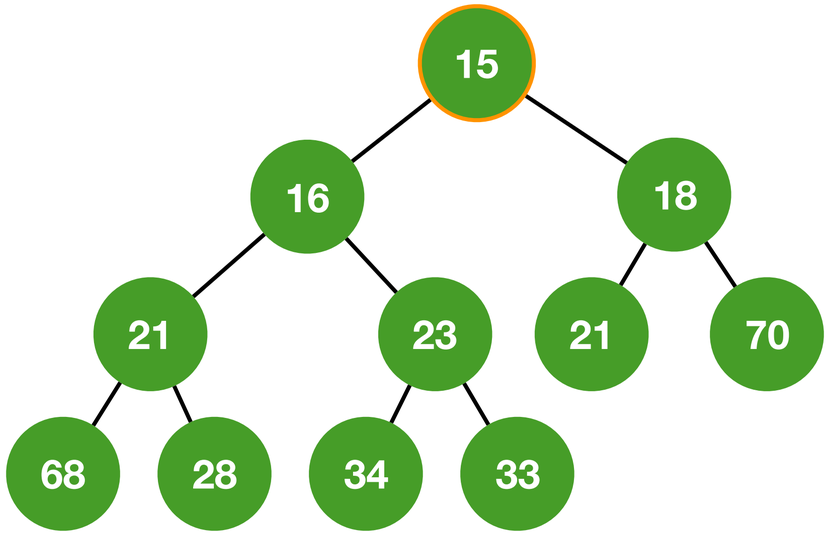



```
struct Heap {
        size
        arr[]
};
struct Heap *heap;
        insert(heap, item) {
                i = ++heap->size;
        while (heap->arr[i/2] > item){
                heap->arr[i] = heap->arr[i/2]
                i = i/2
        }
} heap->arr[i] = item
```

# Heap insert code

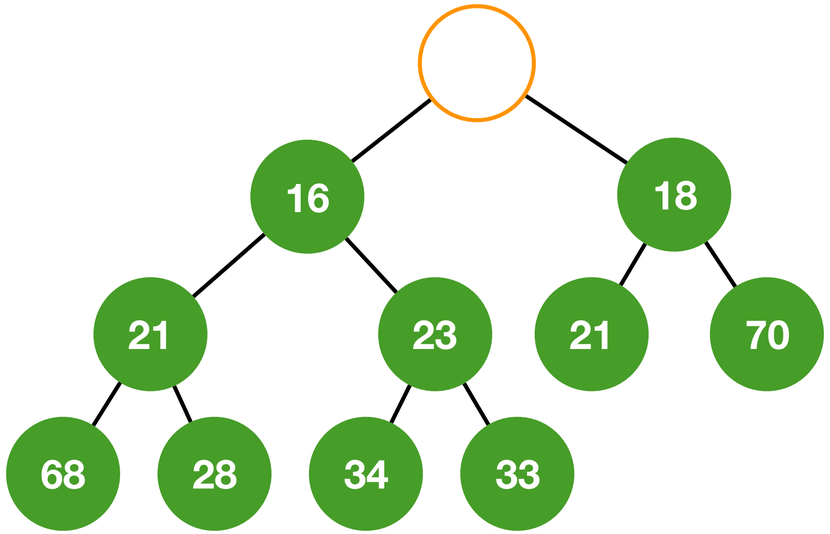
```
//BinaryHeap/binaryHeap.c
// insert the node into the heap maintaining the min heap property
void insert(BinaryHeap *heap, int _distance, int _value) {
   int i;
   Node *newNode = (Node *) malloc(sizeof(Node));
   *newNode = (Node) {.distance = _distance, .value = _value};

   int item = newNode->distance;
   if (isFull(heap)) {
      fprintf(stderr, "Error: priority queue is full");
   }
   else {
      i = ++heap->size;
      // parent has index i/2, percolate up to insert new item
      while (heap->arr[i / 2]->distance > item) {
         heap->arr[i] = heap->arr[i / 2];
         i = i / 2;
      }
      heap->arr[i] = newNode;
   }
}
```

# Heap delete min



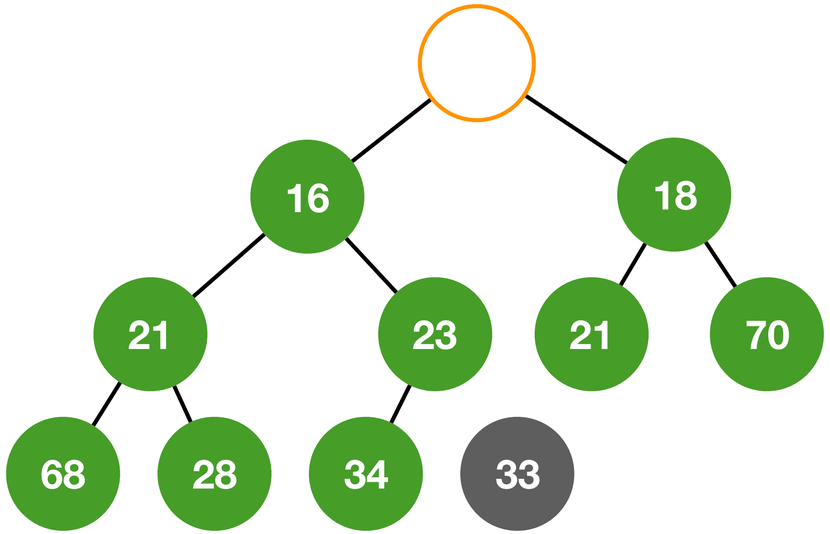

delete_min : Percolate down the heap

# Heap delete min



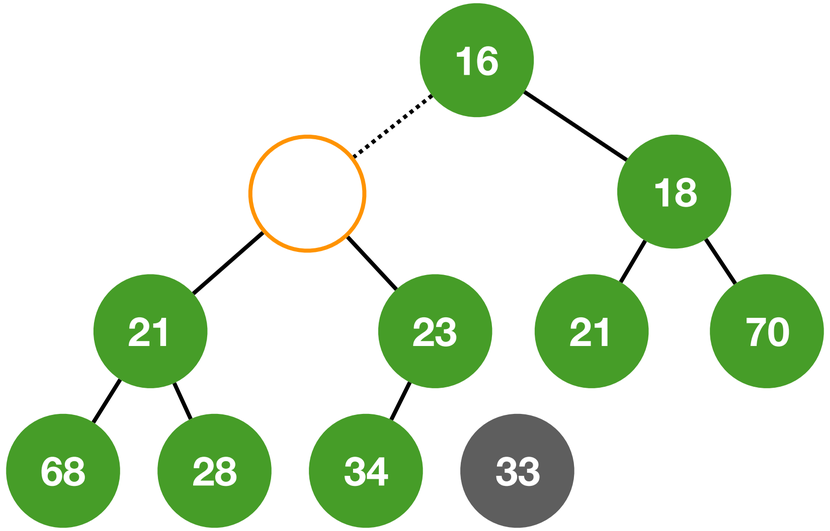

minimum is removed

# Heap delete min




Place 33 in the heap, 33 cannot be placed in the hole (violate heap order)

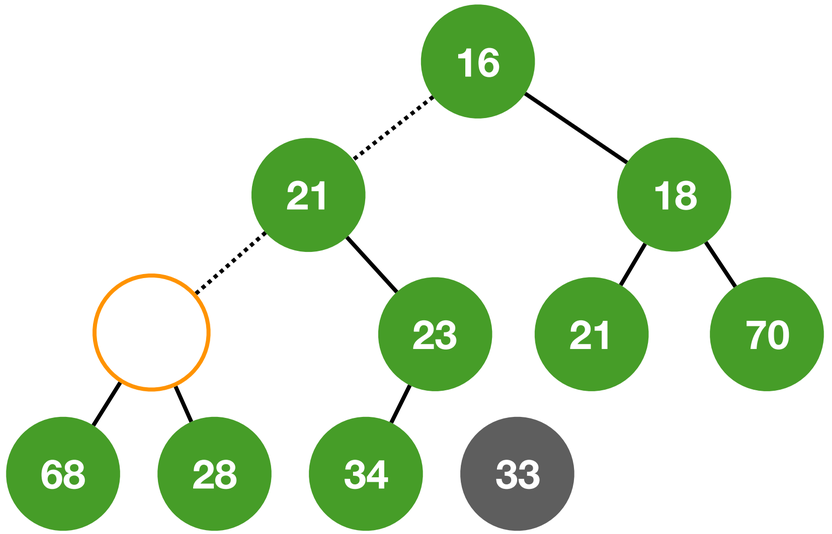
# Heap delete min




Place smaller child 16 into hole, create a new hole on the bottom level

33 cannot be placed in the hole (violate heap order)

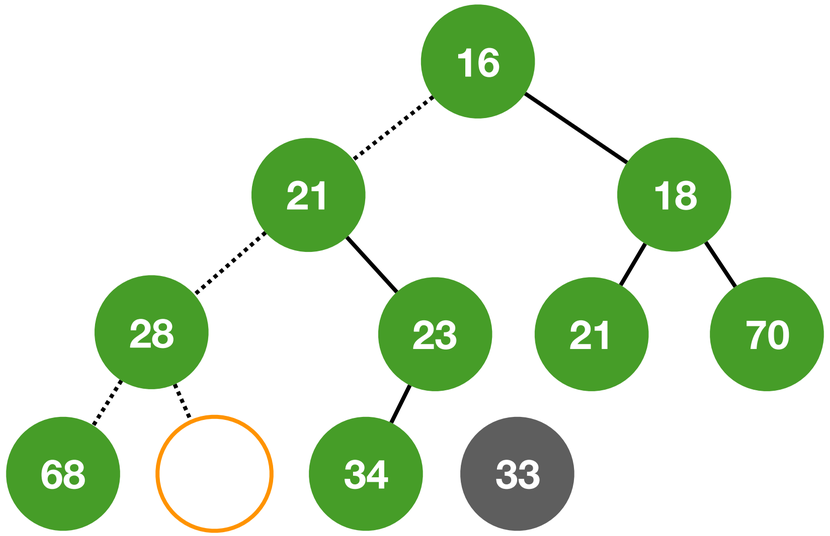
# Heap delete min




Place smaller child 21 into hole, create a new hole on the bottom level

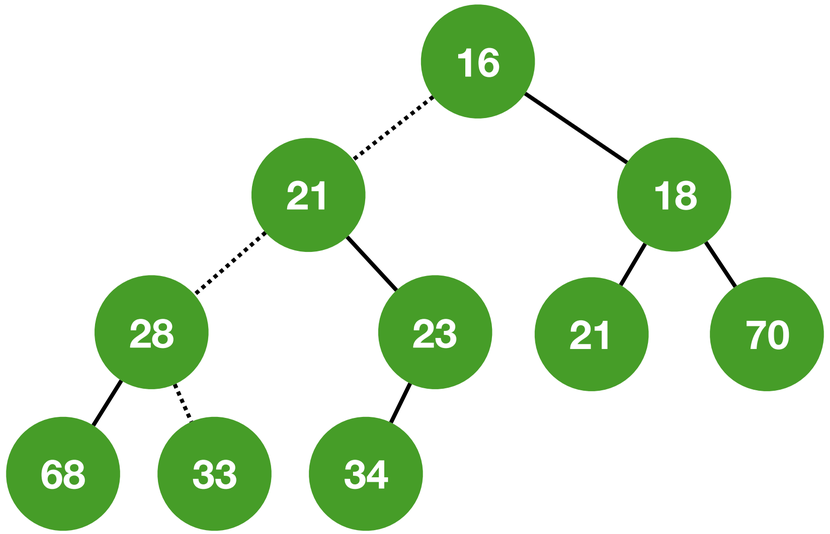
33 cannot be placed in the hole (violate heap order)

# Heap delete min




Place smaller child 28 into hole, create a new hole on the bottom level

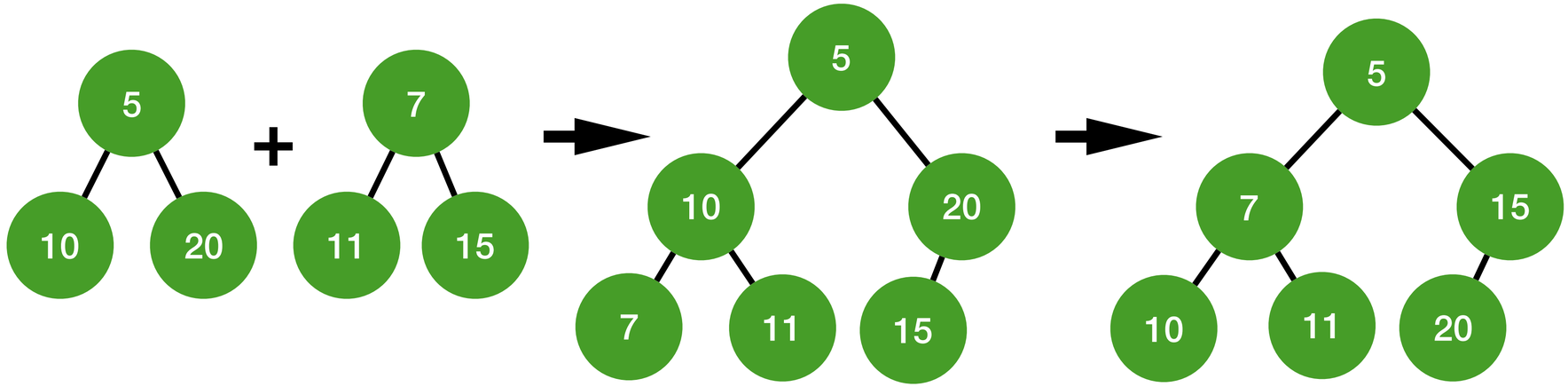
# Heap delete min solution



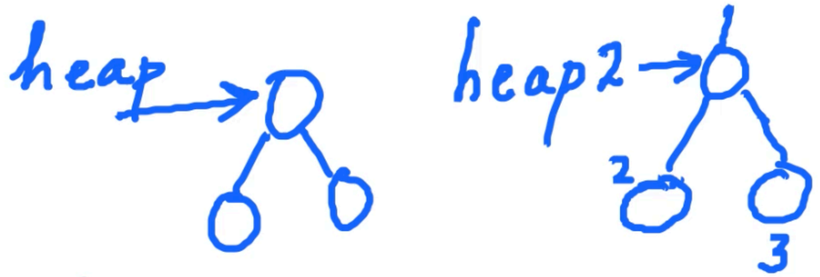

Place 33 in the hole

# Binary heap union

- Combine two binary heaps into a single heap
- Requires  $\Omega(N)$ operations
- Binomial and Fibonacci heaps have better running times



## Heap union




```
size = heap2->size
for (i = 1; i <= size; i++) {
        insert(heap, heap2->arr[i]);
```

# Code for heap union

```
//BinaryHeap/binaryHeap.c
//insert each node from heap2 into this heap
void heapUnion(BinaryHeap *heap, BinaryHeap *heap2) {
  size_t size = getSize(heap2);
  for (int i = 1; i <= size ; i++) {
    int distance = getItem(heap2, i)->distance;
    int value = getItem(heap2, i)->value;
    insert(heap, distance, value);
  }
}
```

# Applications

- Scheduling of jobs by a processor
- Sorting
- Implementation of greedy algorithms
- Any application where we need to find the element with the highest or lowest priority efficiently.