Heap

Notes by Yan Yan

Contents

1. Heap
2. Insertion and Removal in Heap
3. Upheap and Downheap
4. Heap using Array
5. Heapsort

Learning Objectives

1. Remember the definition of max-heap and mini-heap
2. Identify if a tree is heap
3. Describe process of upheap and downheap
4. Implement array-based heap
5. Describe heapsort process using array implementation

Heap Definition

- A **max-heap** is a complete binary tree that every node satisfies the heap property:
  - If B is a child of A, then key(A) ≥ key(B)
  - (A node's key is greater than or equal to the node's children's keys).
  - In a max-heap, the root node has the highest key value in the heap
- A **min-heap** is a heap that elements at every node will be either less than or equal to the element at its left and right child.
  - the root has the lowest key value

Heap Examples

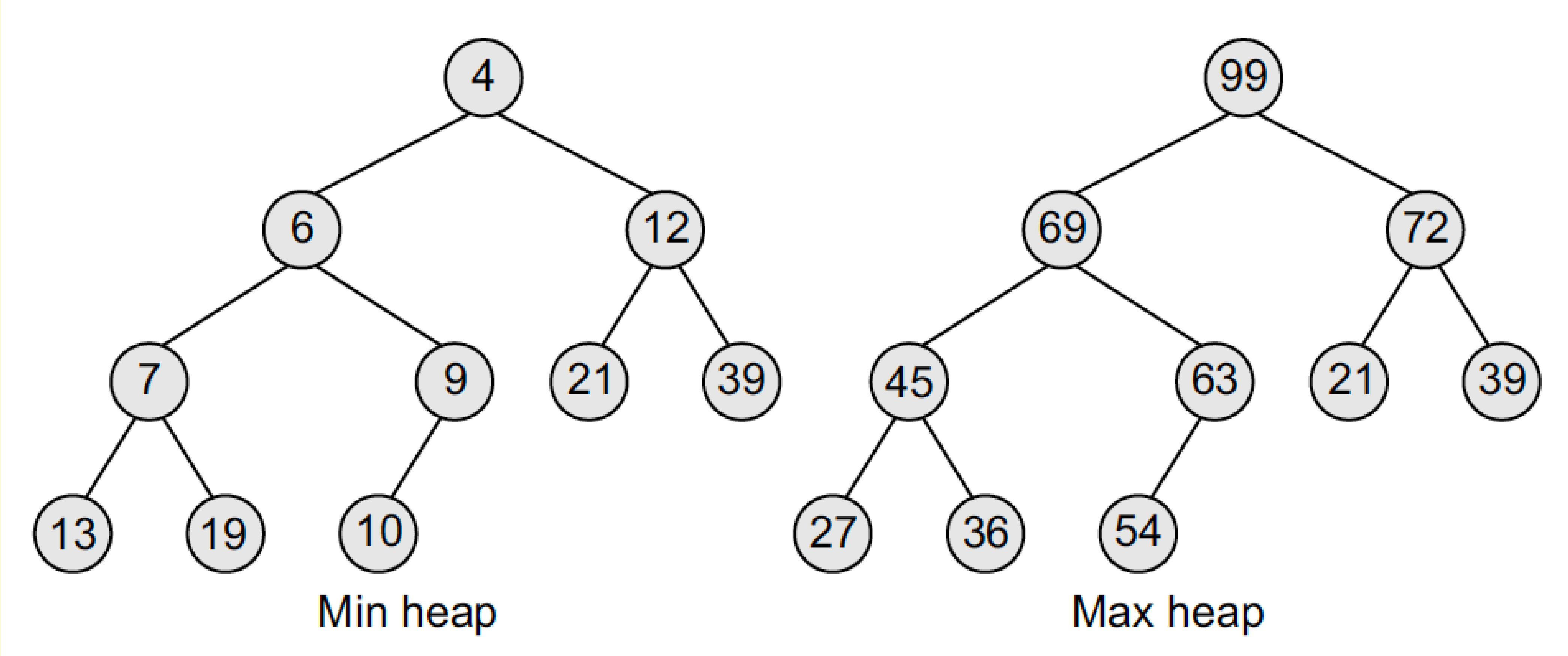


Figure from book Data Structures Using C, 2nd edition (Chapter 12, Figure 12.1), by Reema Thareja

Heap Applications

- A computer may execute jobs one at a time; upon finishing a job, the computer executes the pending job having maximum priority.
- Implement priority queue ADT
  - A priority queue is a queue where each item has a priority, and items with higher priority are closer to the front of the queue than items with lower priority.
  - **Enqueue:** insert and maintain the priority property
  - **Dequeue:** remove and return the front item of the queue (highest priority).

Heap Applications

- Manage prioritized queues of customers awaiting support.
- Customers that entered the line earlier and/or have a more urgent issue get assigned a lower number, which corresponds to a higher priority.

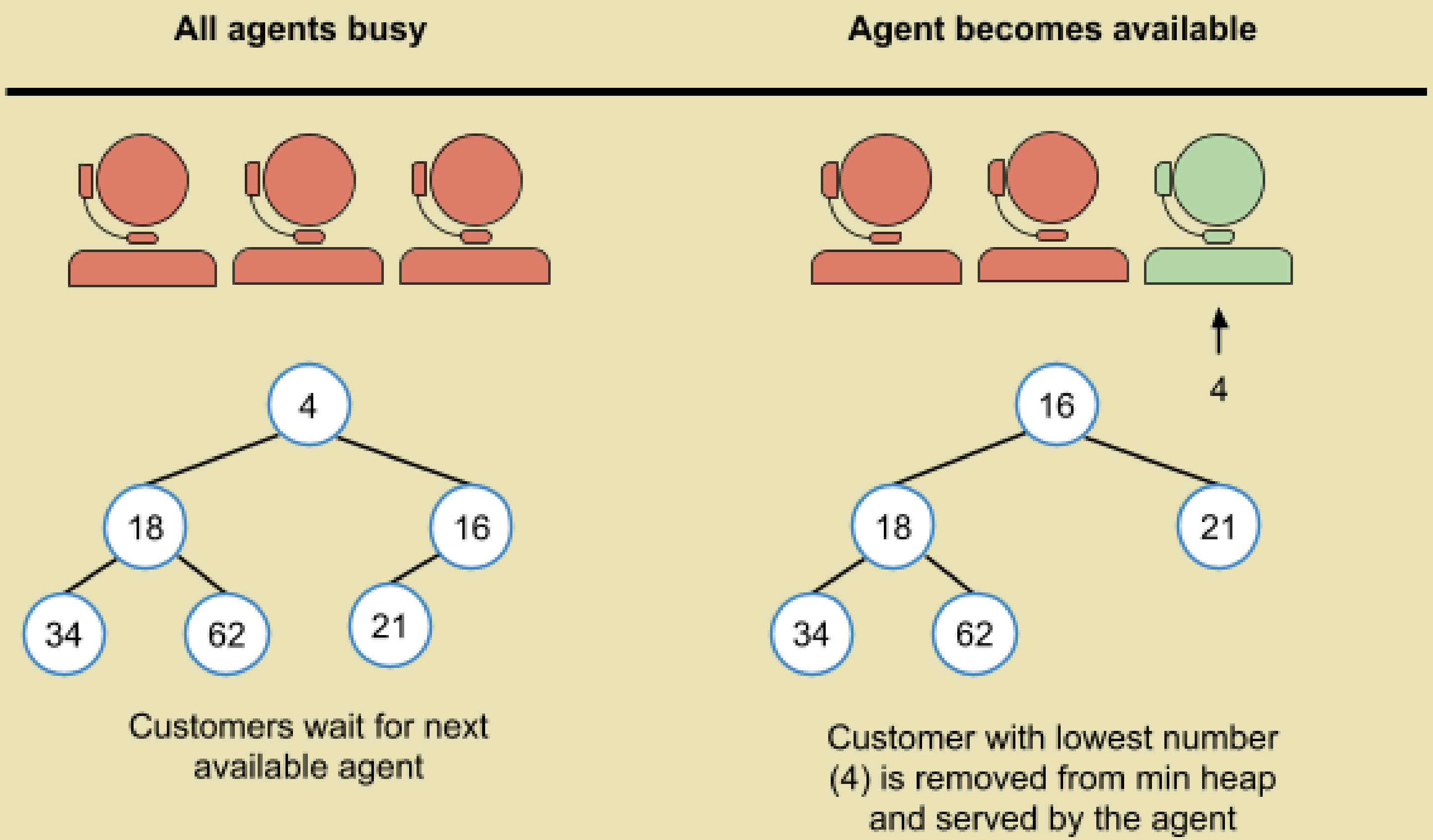


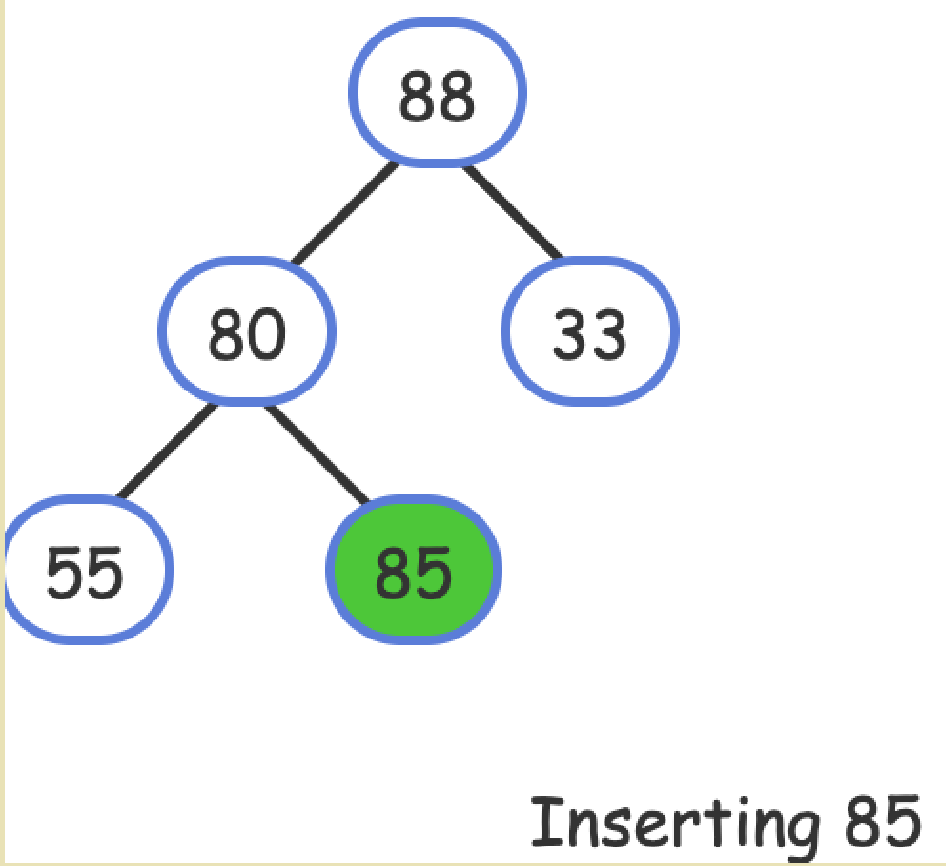
Figure from Zybook 9.1 Heaps

Heap Insertion

- Inserting the node in the tree's last level, and then swapping the node with its parent until no max-heap property violation occurs.
- Inserts fill a level (left-to-right) before adding another level, so the tree is still a complete binary tree.
- The upward movement of a node in a max-heap is called percolating.

Heap Insertion -- Example

- Insert 85



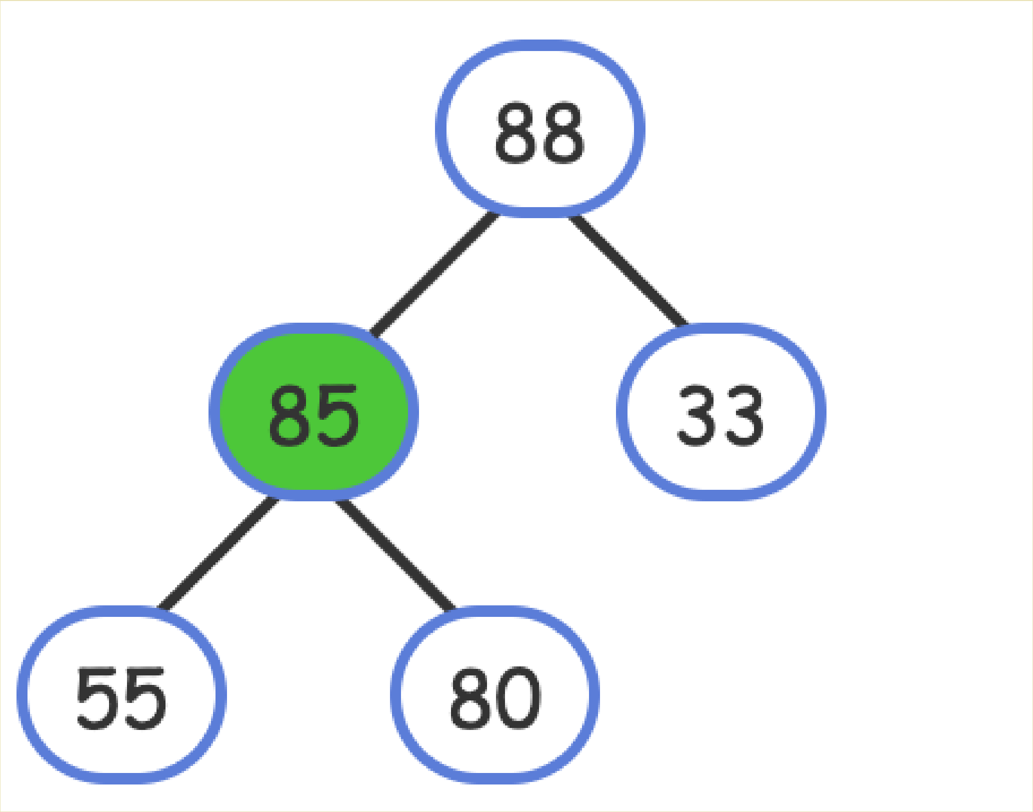


Figure from Zybook 9.1 Heaps

Heap Insertion -- Example

- Insert 99

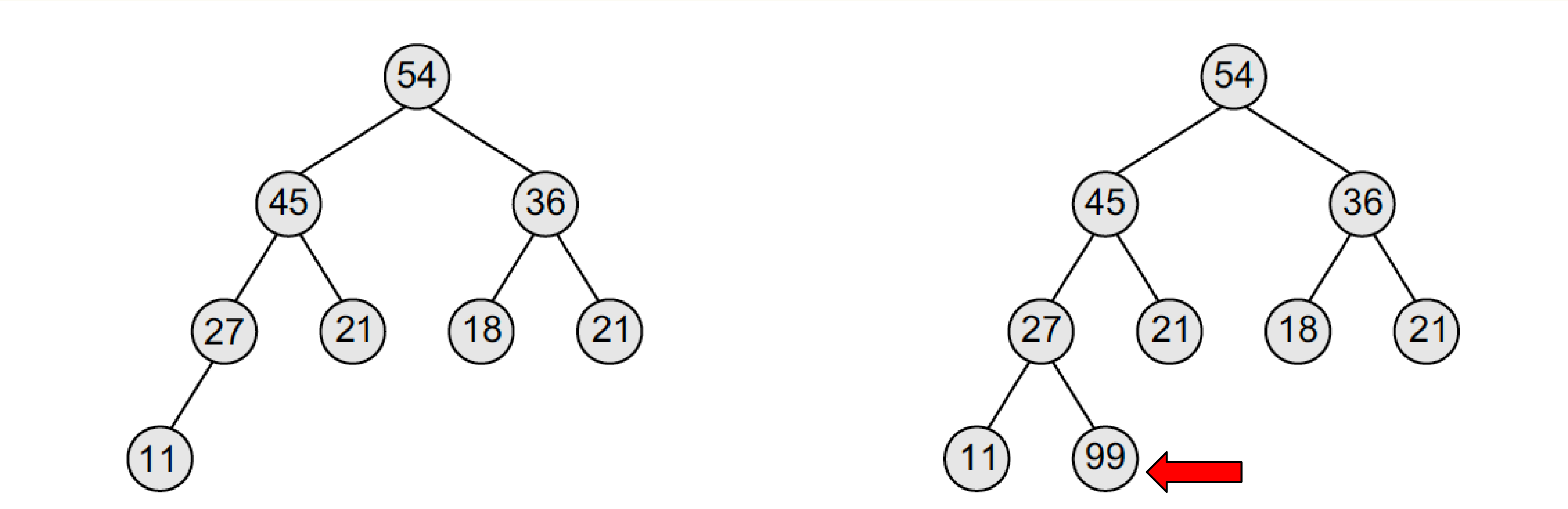
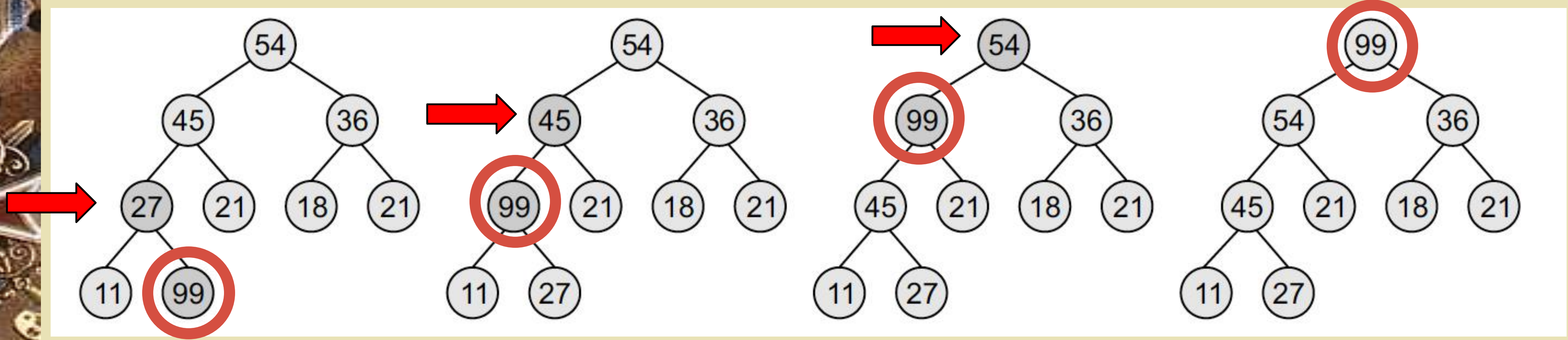


Figure from book Data Structures Using C, 2nd edition (Chapter 12, Figures 12.2 and 12.3), by Reema Thareja

Heap Insertion -- Example

- Insert 99



- This process is also called upheap

Figure from book Data Structures Using C, 2nd edition (Chapter 12, Figures 12.4), by Reema Thareja

Heap Removal

- A remove from a max-heap is always a removal of the root. – why?
- Replace the root with the last level's last node (so the tree remains a complete binary tree)
- Change the new root node with its greatest child until no max-heap property violation occurs.
  - Move the new node to the “correct” spot

Heap Removal -- Example

- Remove 54

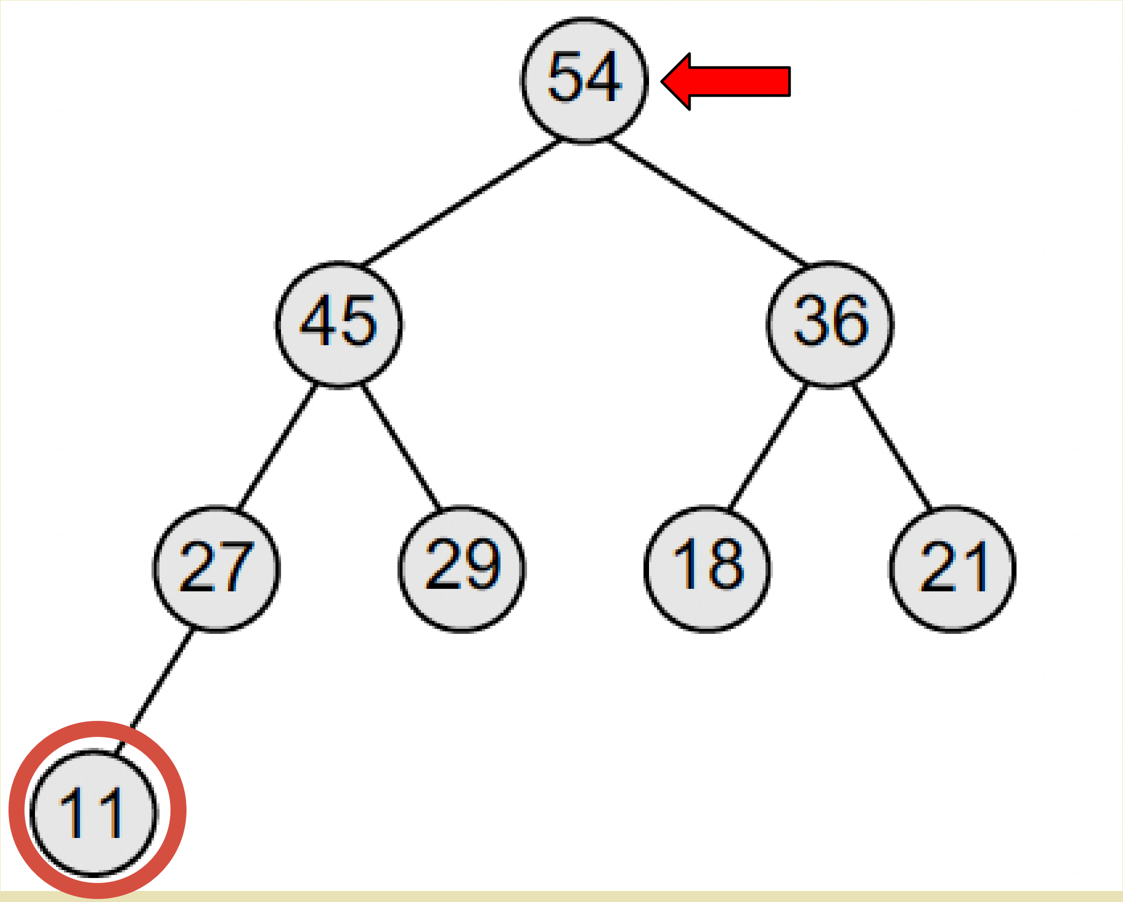
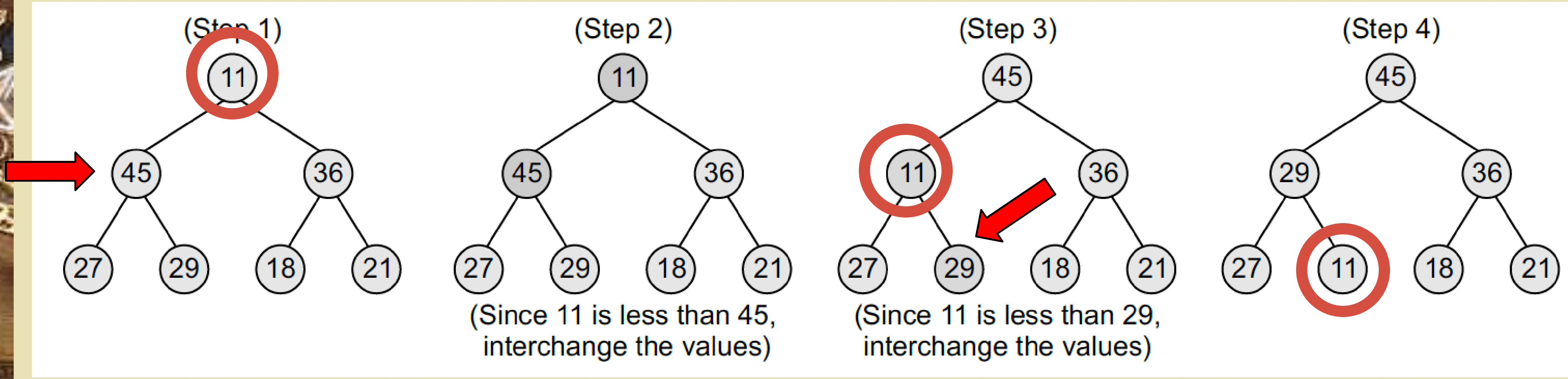


Figure from book Data Structures Using C, 2nd edition (Chapter 12, Figures 12.8), by Reema Thareja

Heap Removal -- Example

- Remove 54



- This process is also called downheap

Figure from book Data Structures Using C, 2nd edition (Chapter 12, Figures 12.9), by Reema Thareja

Heap using Array

- Heaps are typically stored using arrays.
- Given a tree representation of a heap, the heap's array form is produced by traversing the tree's levels from left to right and top to bottom.
- The root **node** is always the entry at index **0** in the array, the root's left child is the entry at index 1, the root's right child is the entry at index 2, and so on.

Heap using Array -- Example

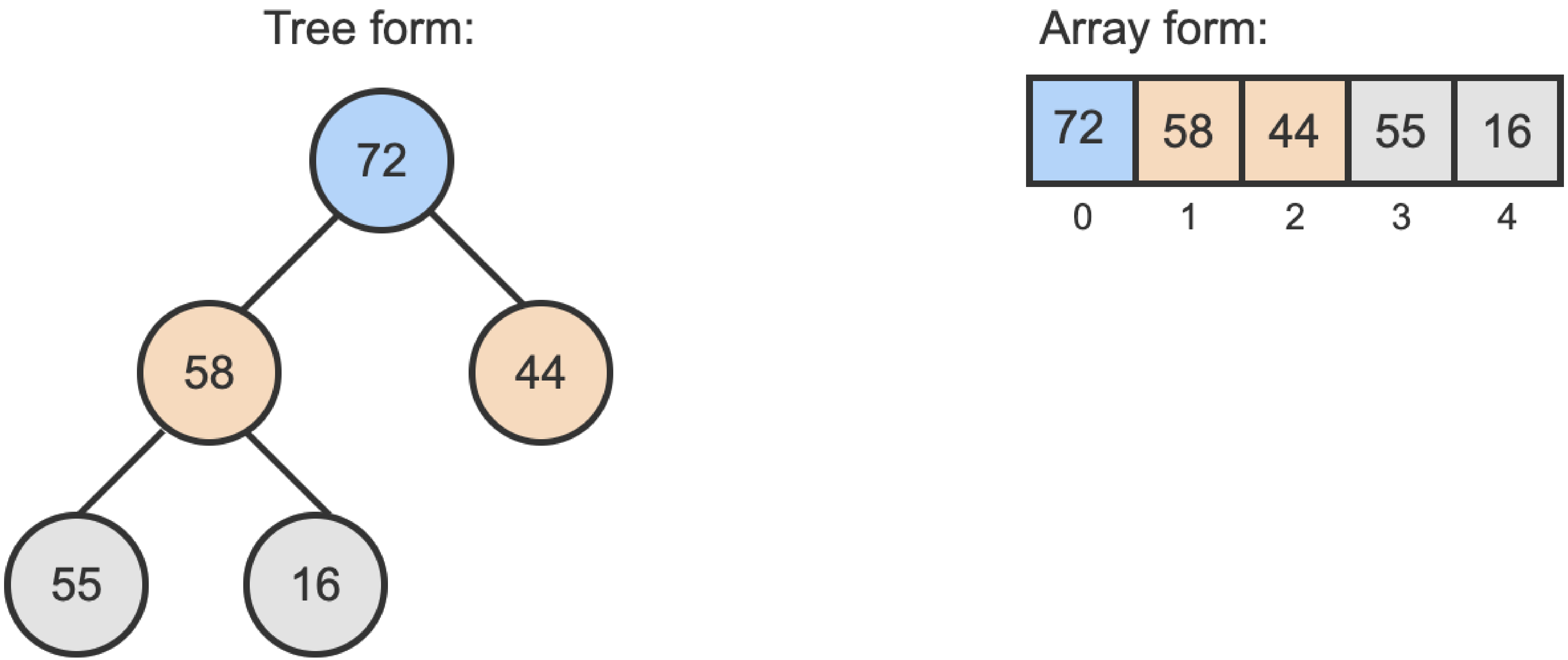


Figure from Zybook 9.2 Heaps

Heap using Array -- Example

- Add 63

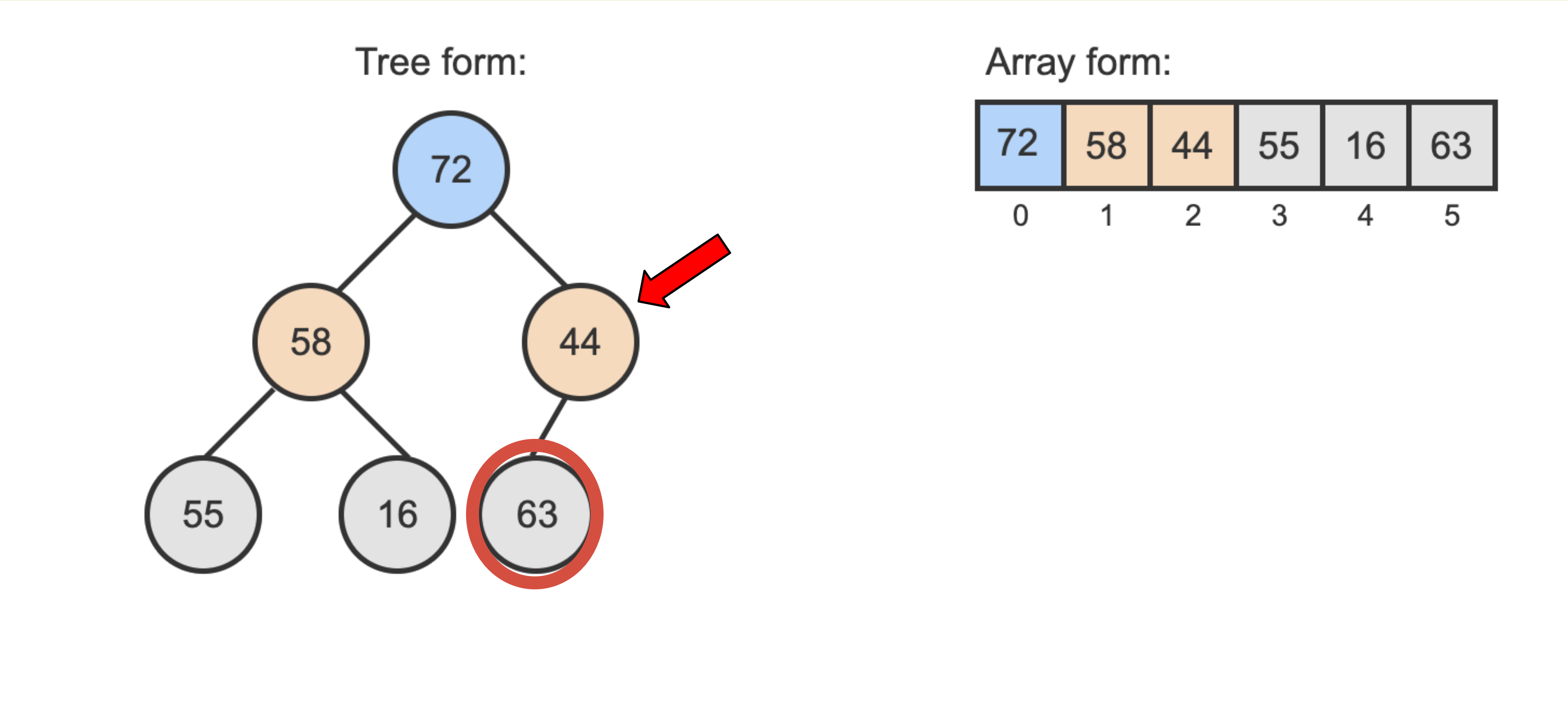


Figure from Zybook 9.2 Heaps

Heap using Array -- Example

- Add 63

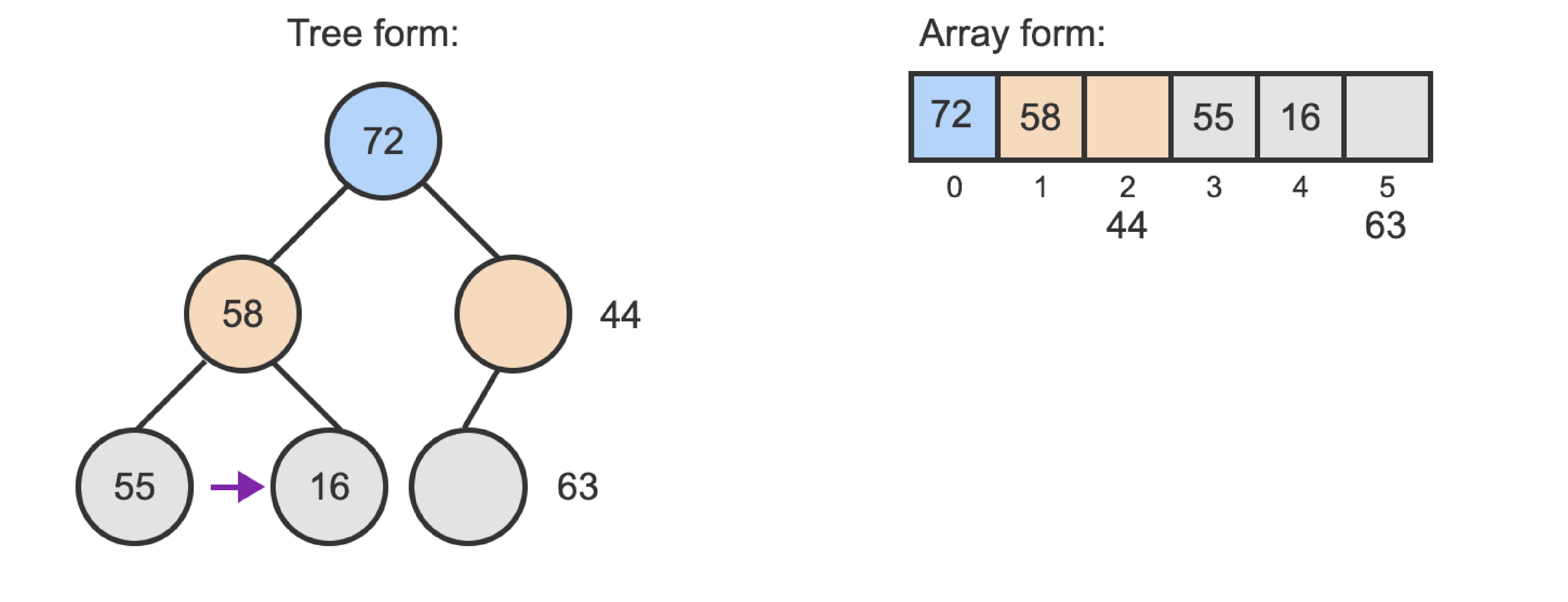


Figure from Zybook 9.2 Heaps

Heap using Array -- Example

- Add 63

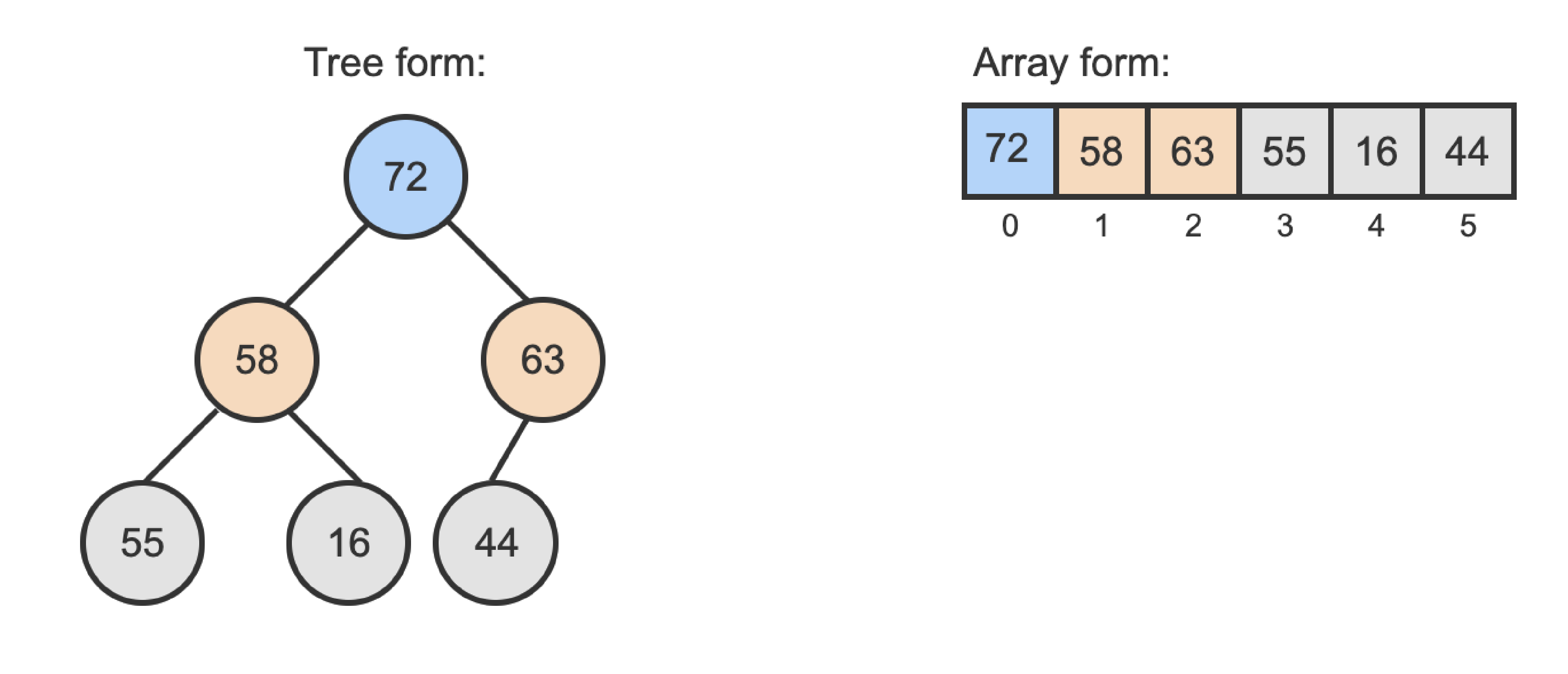


Figure from Zybook 9.2 Heaps

Heap using Array

- Parent child node indices for a heap.

Node index	Parent index	Child indices
0	N/A	1, 2
1	0	3, 4
2	0	5, 6
3	1	7, 8
4	1	9, 10
5	2	11, 12
...	...	...
i	$\lfloor (i-1)/2 \rfloor$	2 * i + 1, 2 * i + 2

Figure from Zybook 9.2 Heaps

Heap using Array -- percolate-up

- Pseudocode for the array-based percolate-up function

```
MaxHeapPercolateUp(nodeIndex, heapArray)
    while nodeIndex > 0 do
        parentIndex = (nodeIndex - 1) // 2   // Find the parent index

        if heapArray[nodeIndex] <= heapArray[parentIndex] then
            return                                // Heap property is satisfied
        else
            swap(heapArray[nodeIndex], heapArray[parentIndex])  // Swap with parent
            nodeIndex = parentIndex             // Move up to the parent's position
        end if
    end while
end function
```

Heap using Array -- percolate-down

- Pseudocode for the array-based percolate-down function

```
MaxHeapPercolateDown(nodeIndex, heapArray, arraySize)
    childIndex = 2 * nodeIndex + 1      // Left child index
    value = heapArray[nodeIndex]

    while childIndex < arraySize do    // Loop until we reach the end of the heap
        maxValue = value                // Initialize maxValue as the current node's value
        maxIndex = -1                   // Initialize maxIndex to an invalid index

        // Check both children (if they exist) to find the larger one
        for i = 0 to 1 do
            if childIndex + i < arraySize then
                if heapArray[childIndex + i] > maxValue then
                    maxValue = heapArray[childIndex + i]
                    maxIndex = childIndex + i
                end if
            end if
        end for

(continue…)
```

Heap using Array -- percolate-down

- Pseudocode for the array-based percolate-down function

(continue…)

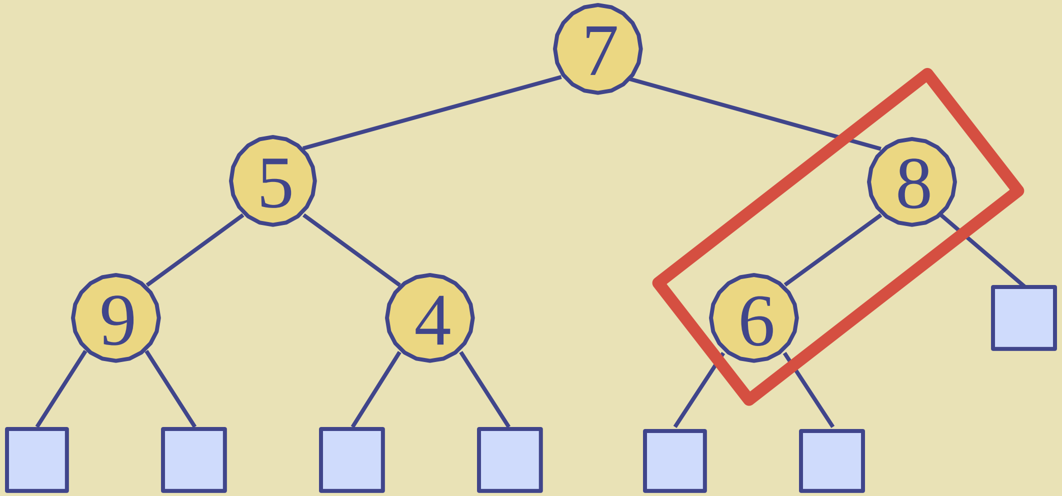
```
  if maxValue == value then        // If the largest value is already at the current node
       return                      // The heap property is satisfied, so stop
     else
       // Swap with the larger child
       swap(heapArray[nodeIndex], heapArray[maxIndex])
       nodeIndex = maxIndex              // Move down to the child position
       childIndex = 2 * nodeIndex + 1    // Update to the new left child index
     end if
   end while
end function
```

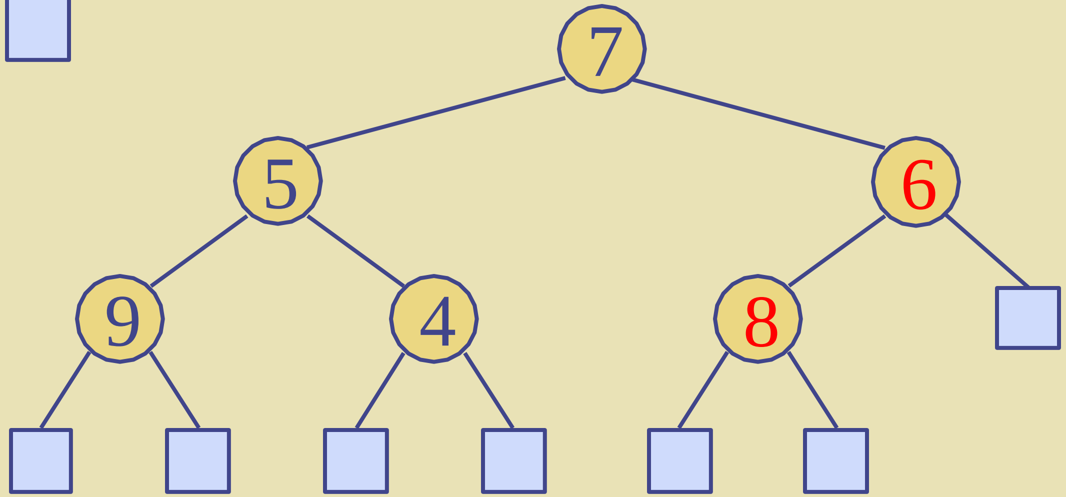
Heap Construction by Downheap

1. Construct a complete binary tree with the list from the root down and from left to right.
2. Apply downheap to every parental node starting with the last one and backwards to the root. (heapify)

Heap Construction by Downheap

- Example: Construct a mini-heap from list 7, 5, 8, 9, 4, 6

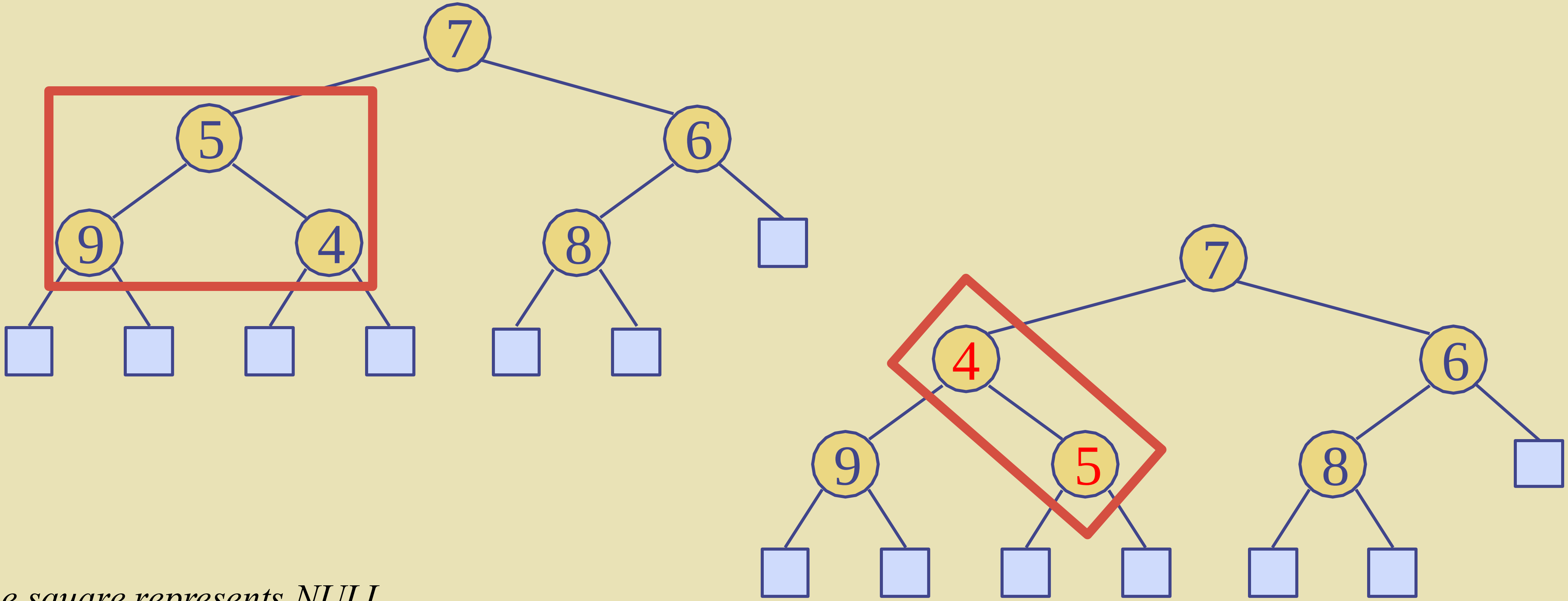




Blue square represents NULL

Heap Construction by Downheap

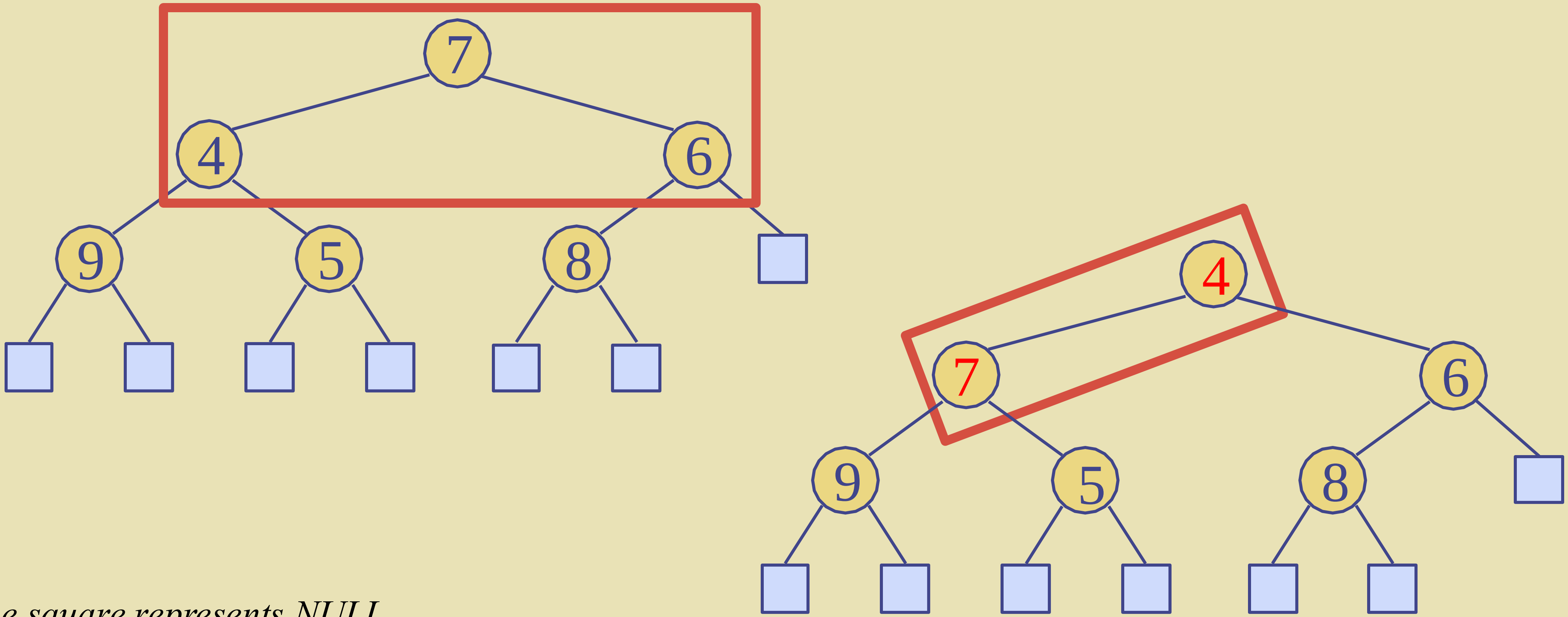
- Example: Construct a mini-heap from list 7, 5, 8, 9, 4, 6



Blue square represents NULL

Heap Construction by Downheap

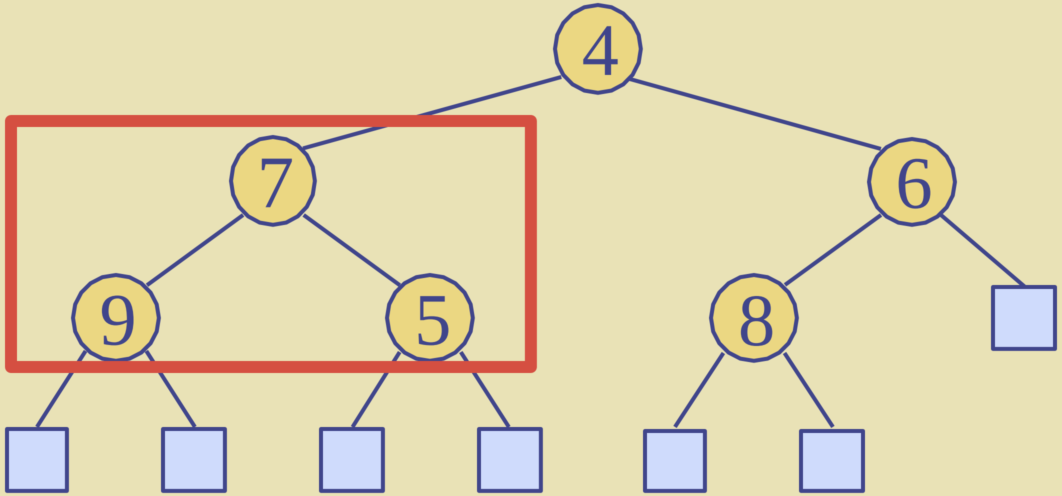
- Example: Construct a mini-heap from list 7, 5, 8, 9, 4, 6

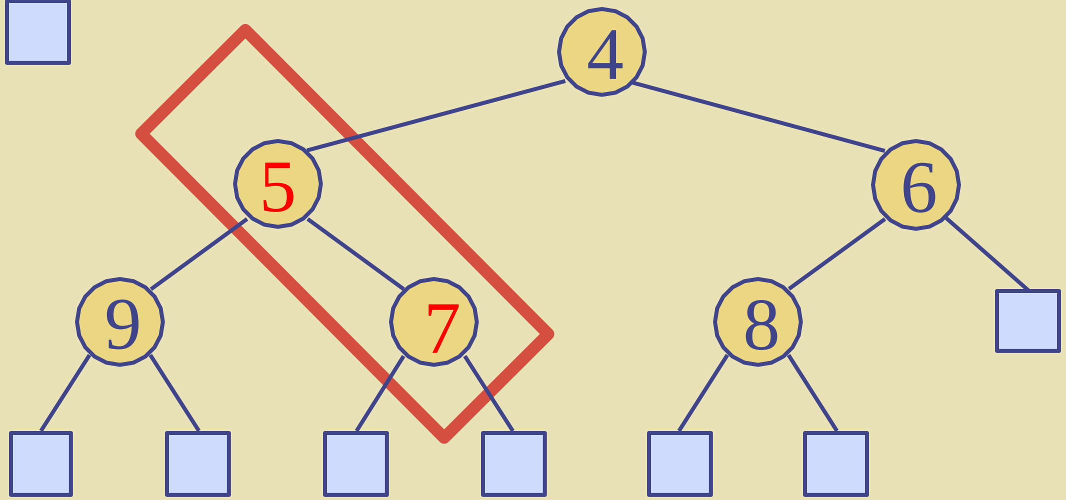


Blue square represents NULL

Heap Construction by Downheap

- Example: Construct a mini-heap from list 7, 5, 8, 9, 4, 6





Blue square represents NULL

Heapsort

Increasing order (mini-heap)

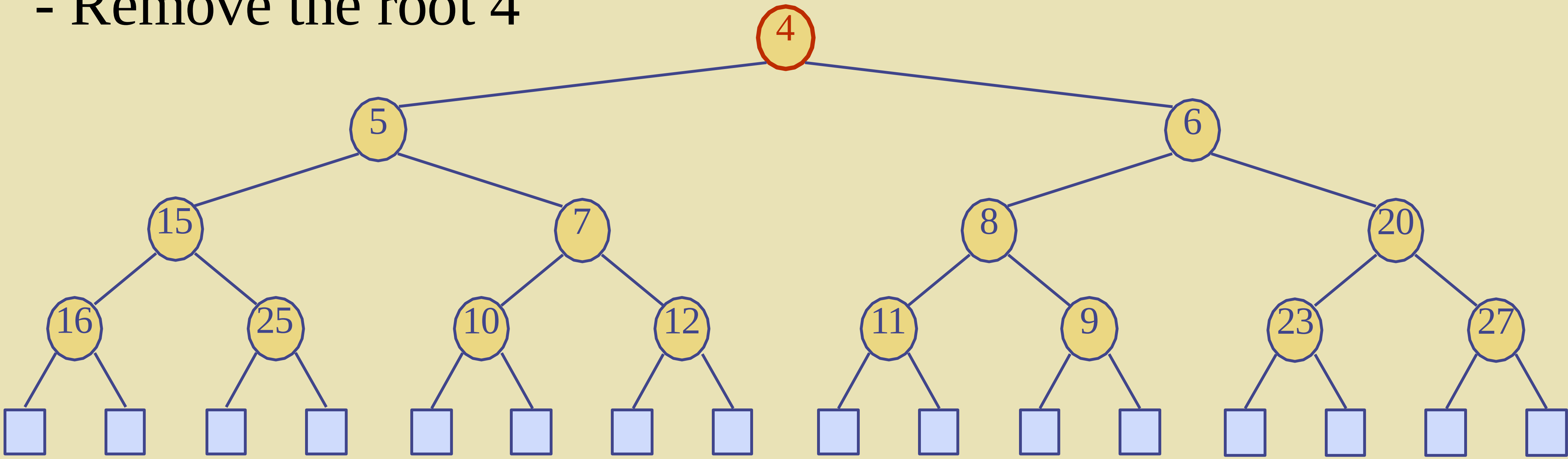
1. Construct a mini-heap from a list (e.g. array).
2. The minimum at the heap root is removed and placed in the result, then the downheap algorithm is used to restore the heap-order property.

Decreasing order is built on a max-heap

Heapsort

Example for a mini-heap

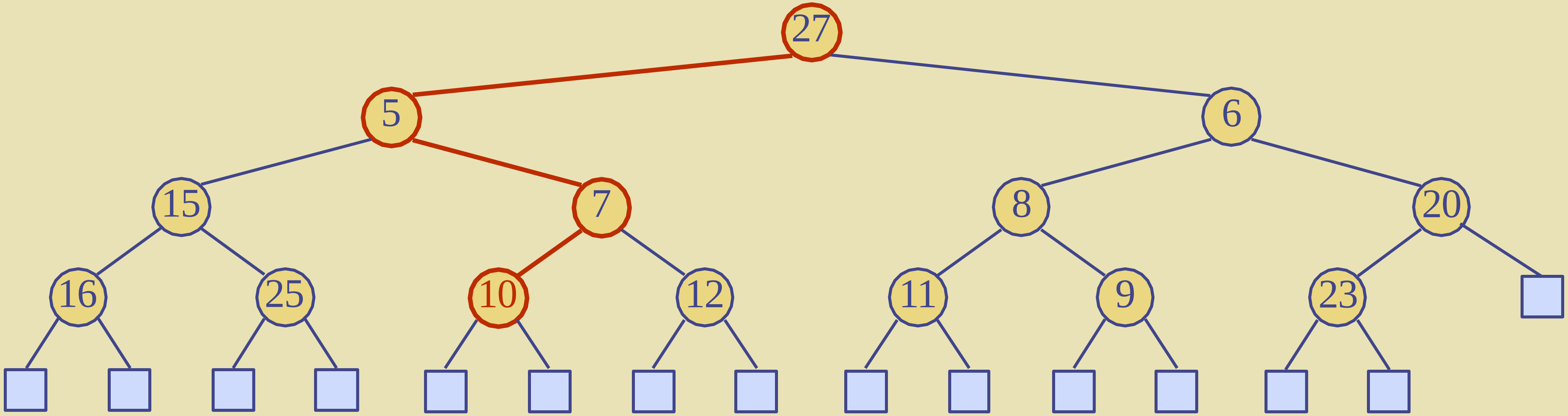
- Remove the root 4



Heapsort

Example for a mini-heap

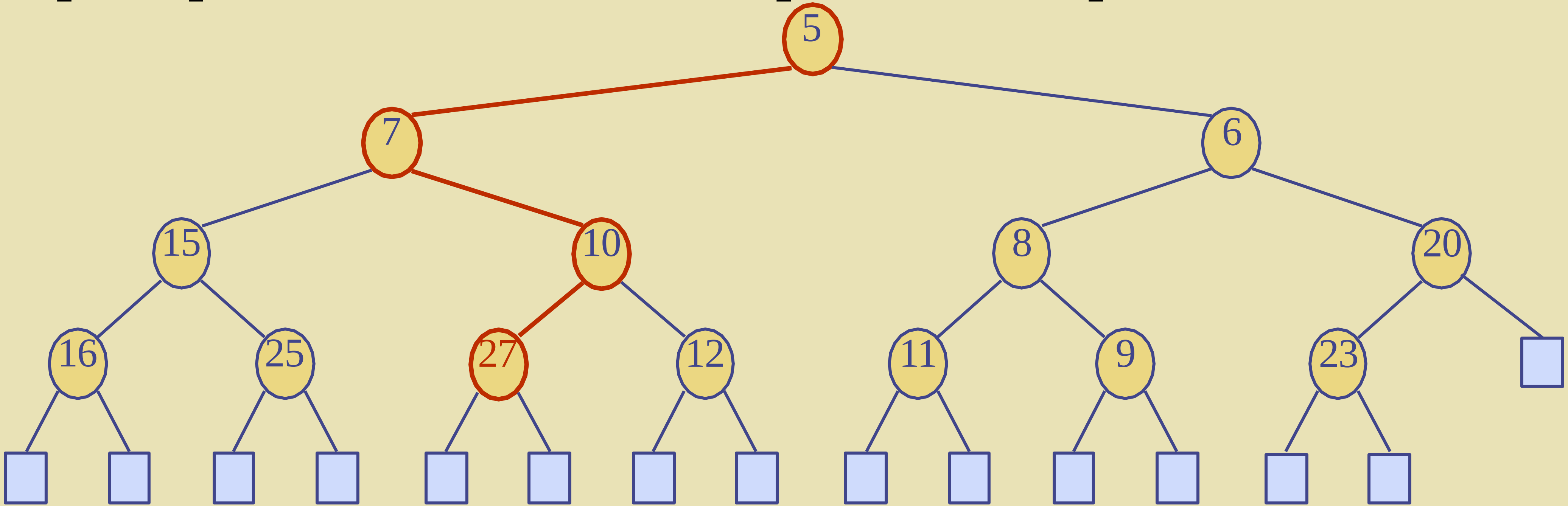
- Put up the last leaf, 27, as the new root
- Restore the heap



Heapsort

Example for a mini-heap

- Heap restored, and remove the new root 5
- put up 23, and continue the previous steps

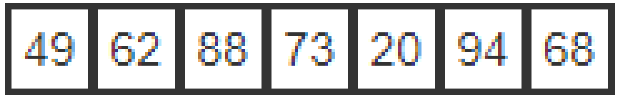


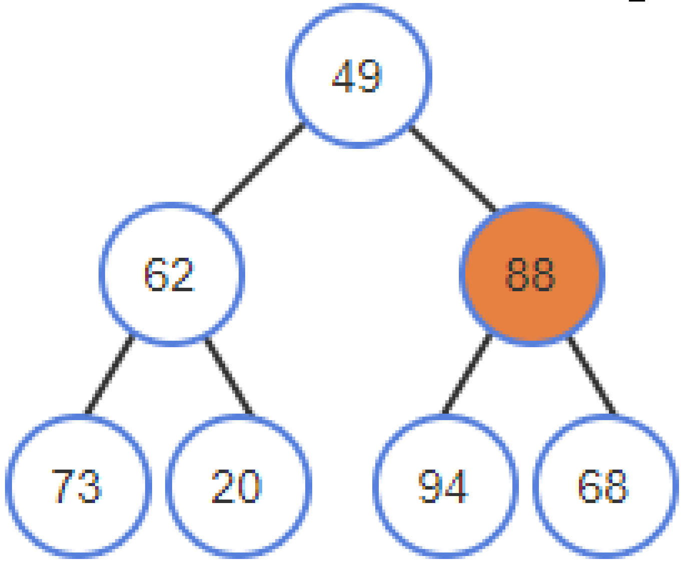
Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

1. Build complete binary tree
2. Build a Heap.

2.a. Find the last internal (parent) node, 88, check and swap with 94

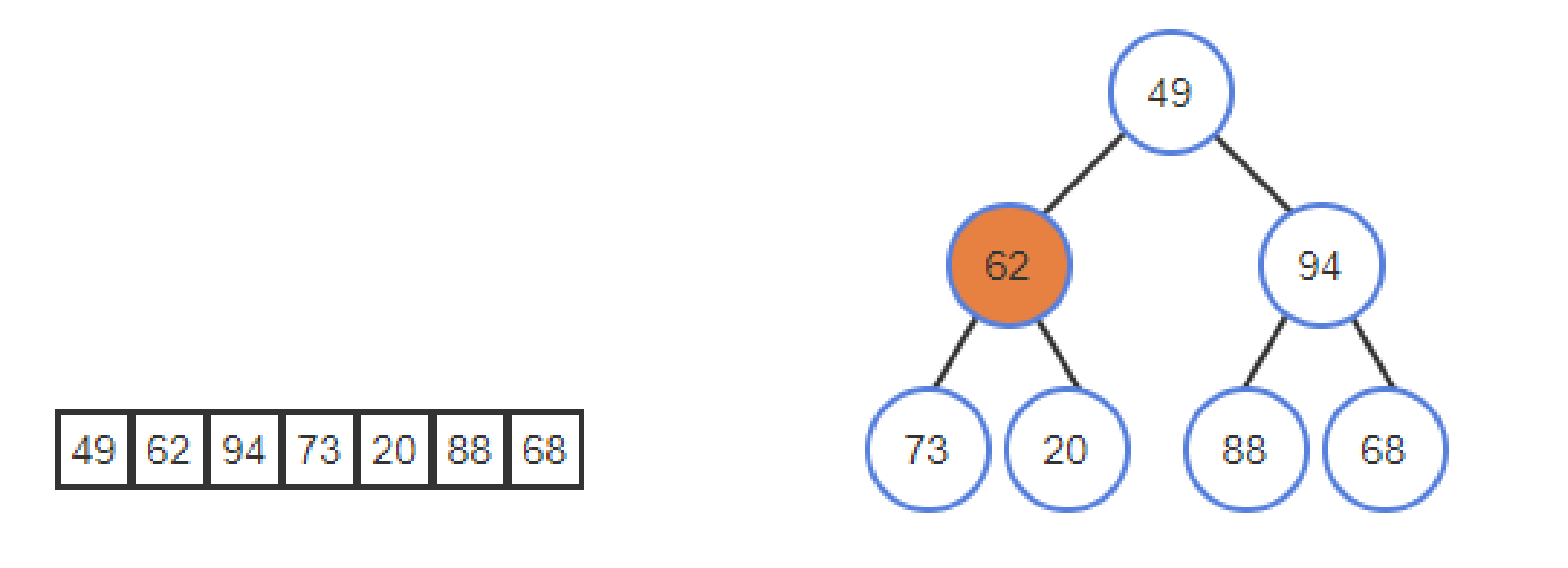




Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

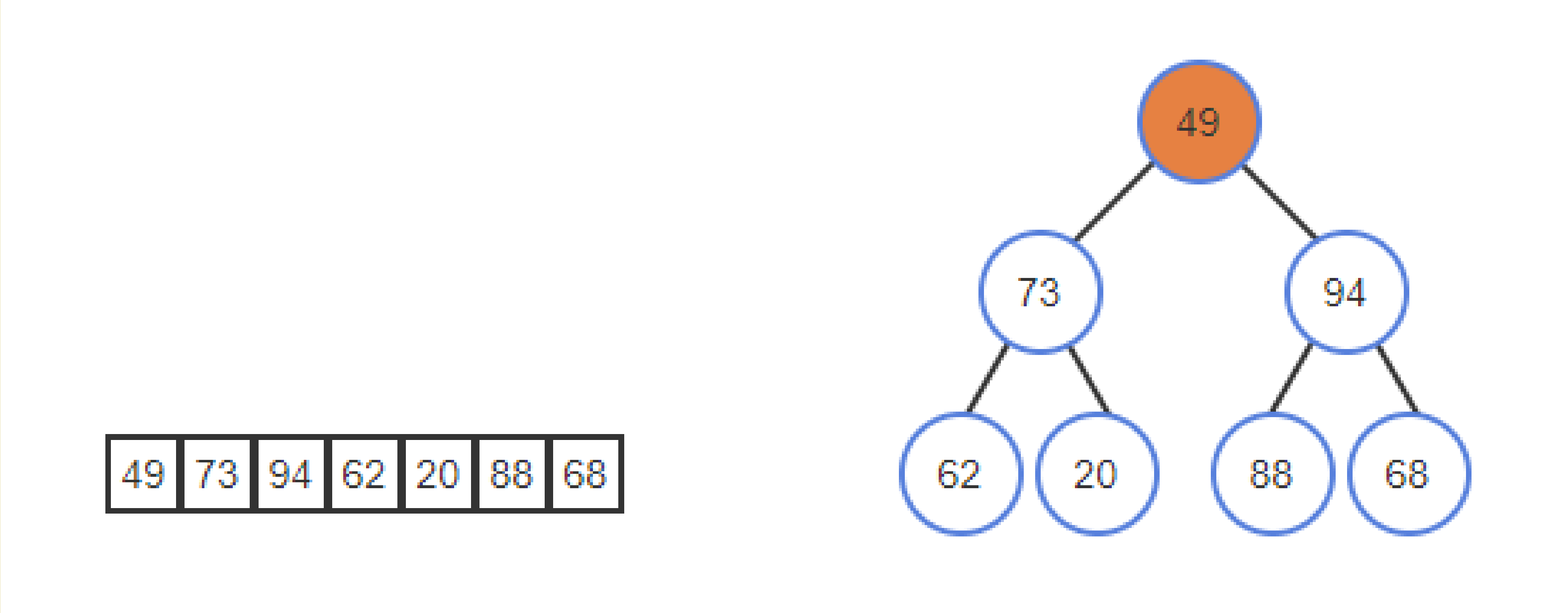
- 2.b. Find the previous internal (parent) node, 62, check and swap with 73



Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

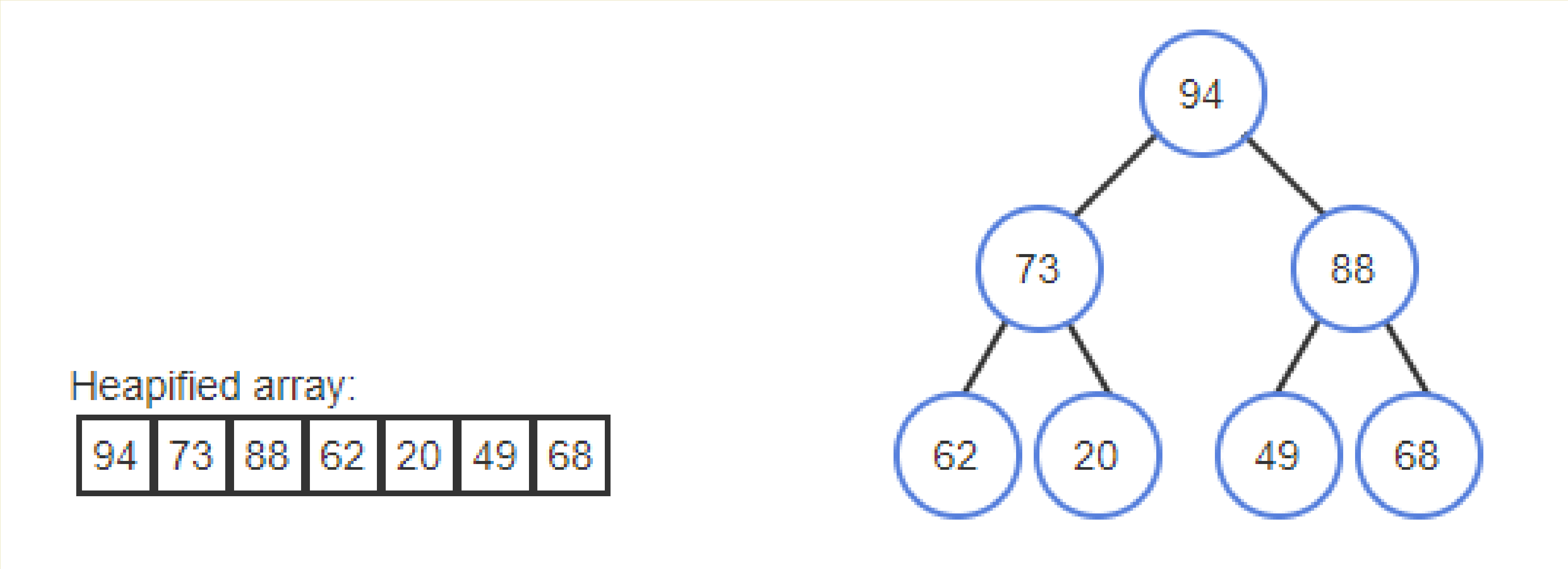
2.c. Find the previous internal (parent) node, 49, check and swap with 94, and 88



Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

2.d. Heap is built

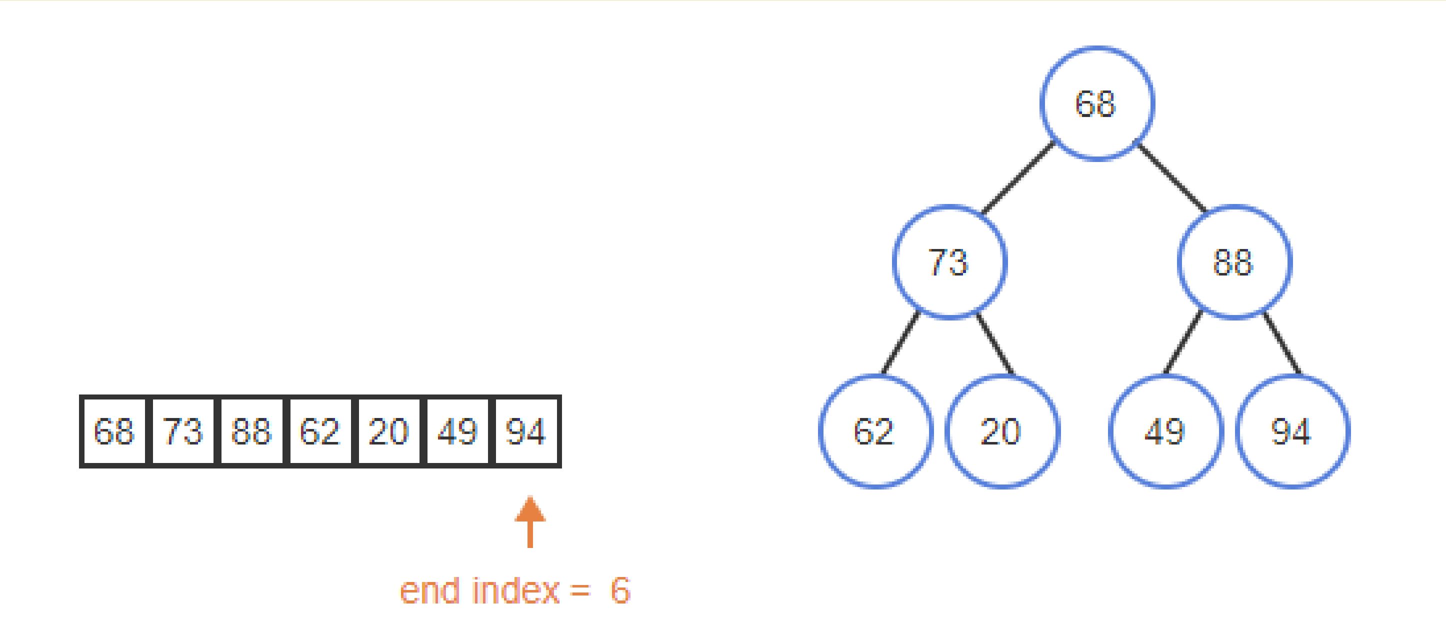


Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

3. Sort the heap by removing root and restoring the heap recursively.

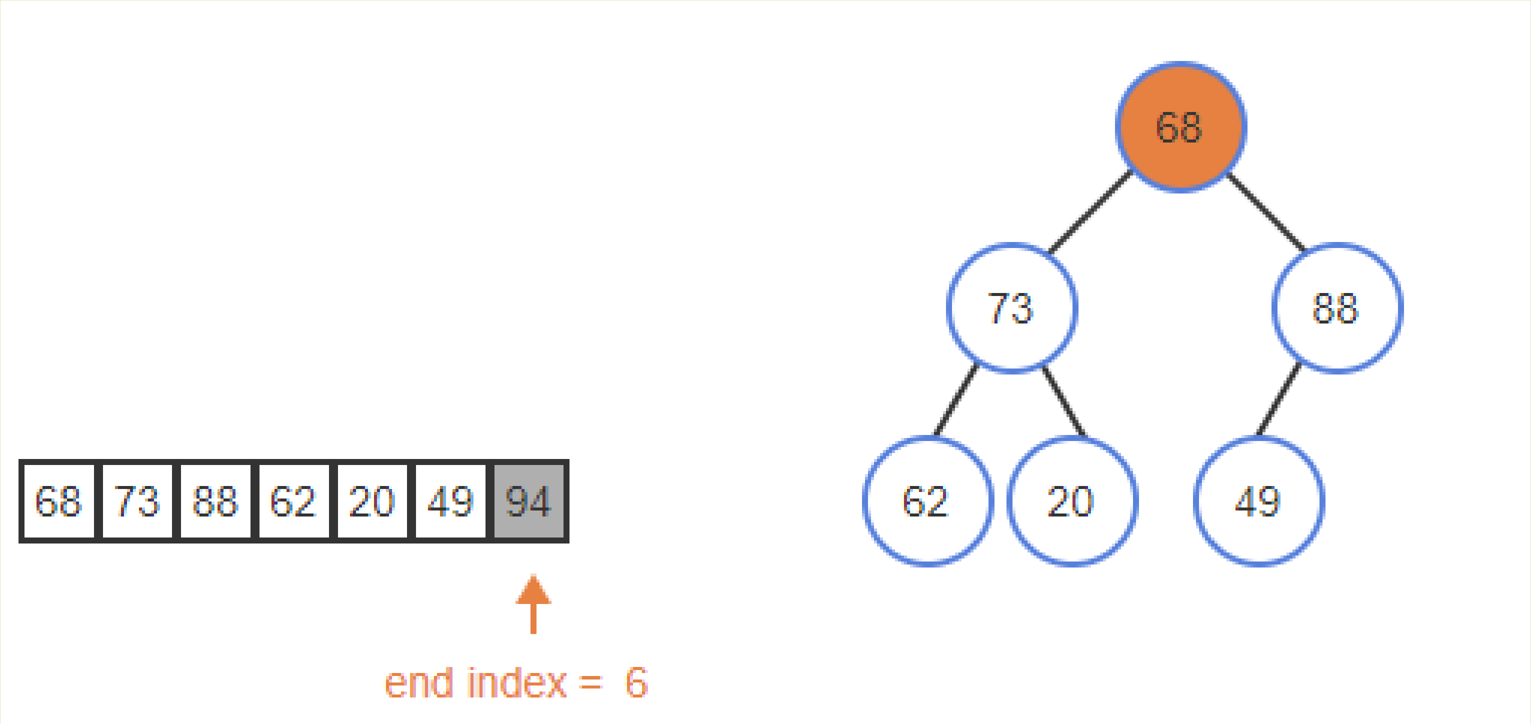
3.a. Remove root 94, and put last leaf node, 68, as the root. In the array, swap 94 and 68



Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

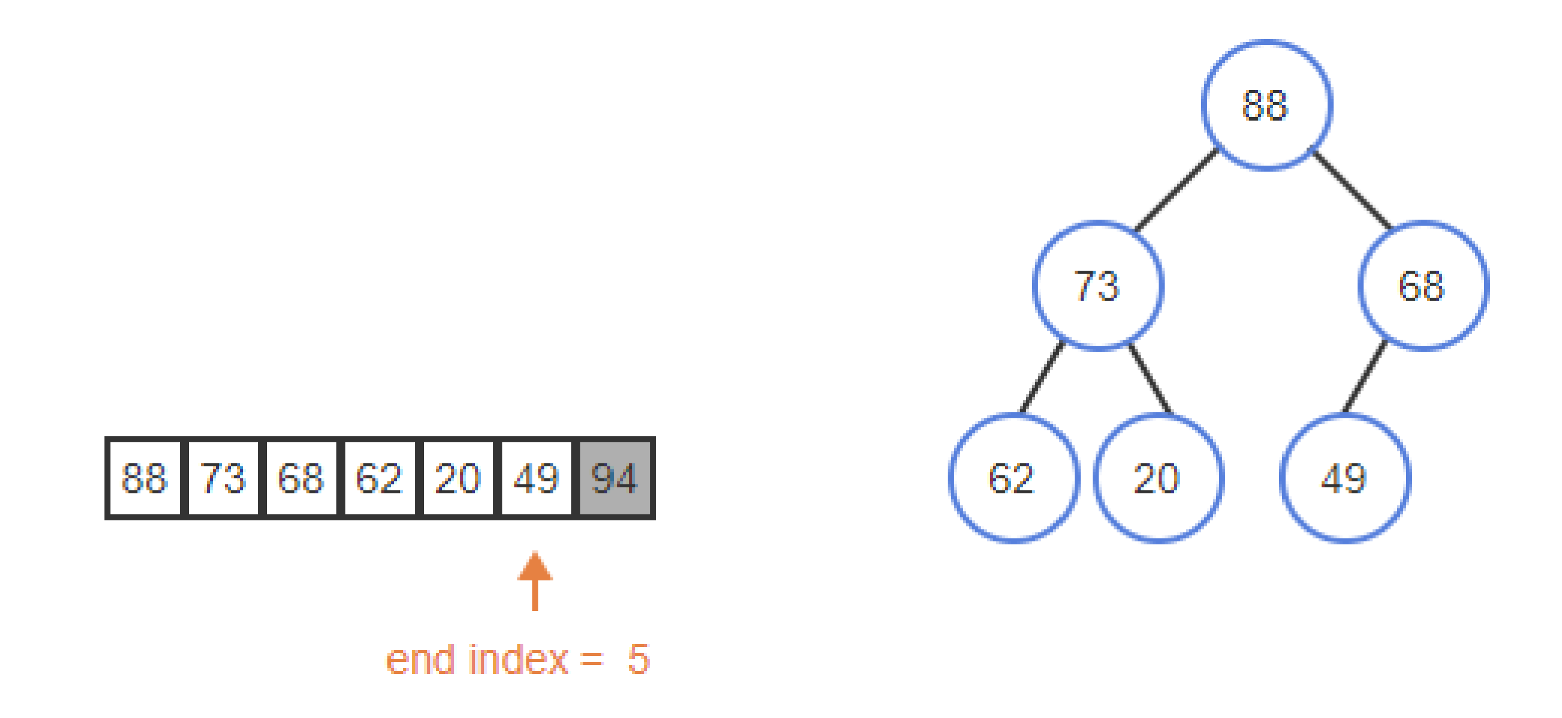
3.b. Restore heap with the new root 68, swap with 88



Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

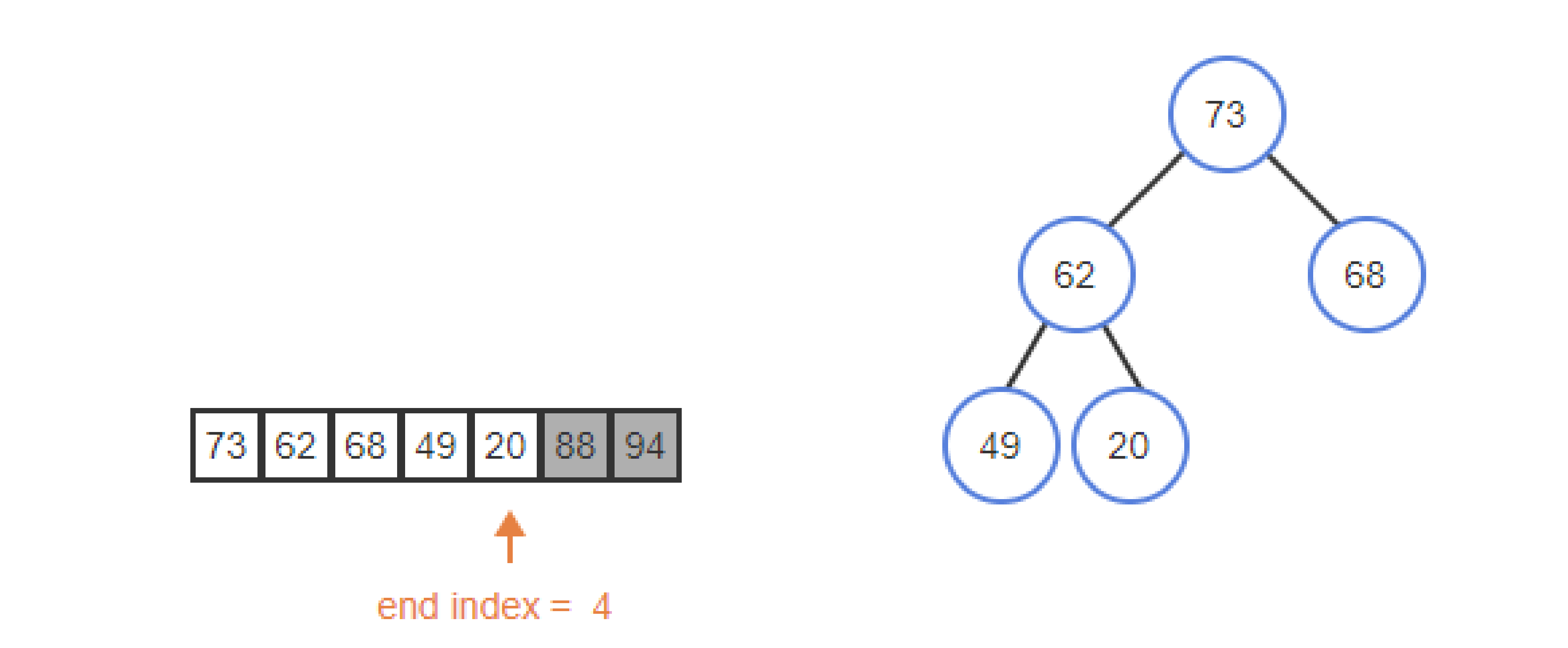
3.c Remove the new root 88, move 49 up to the root (in array, swap with 49), restore heap (49 will swap with 73, 62)



Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

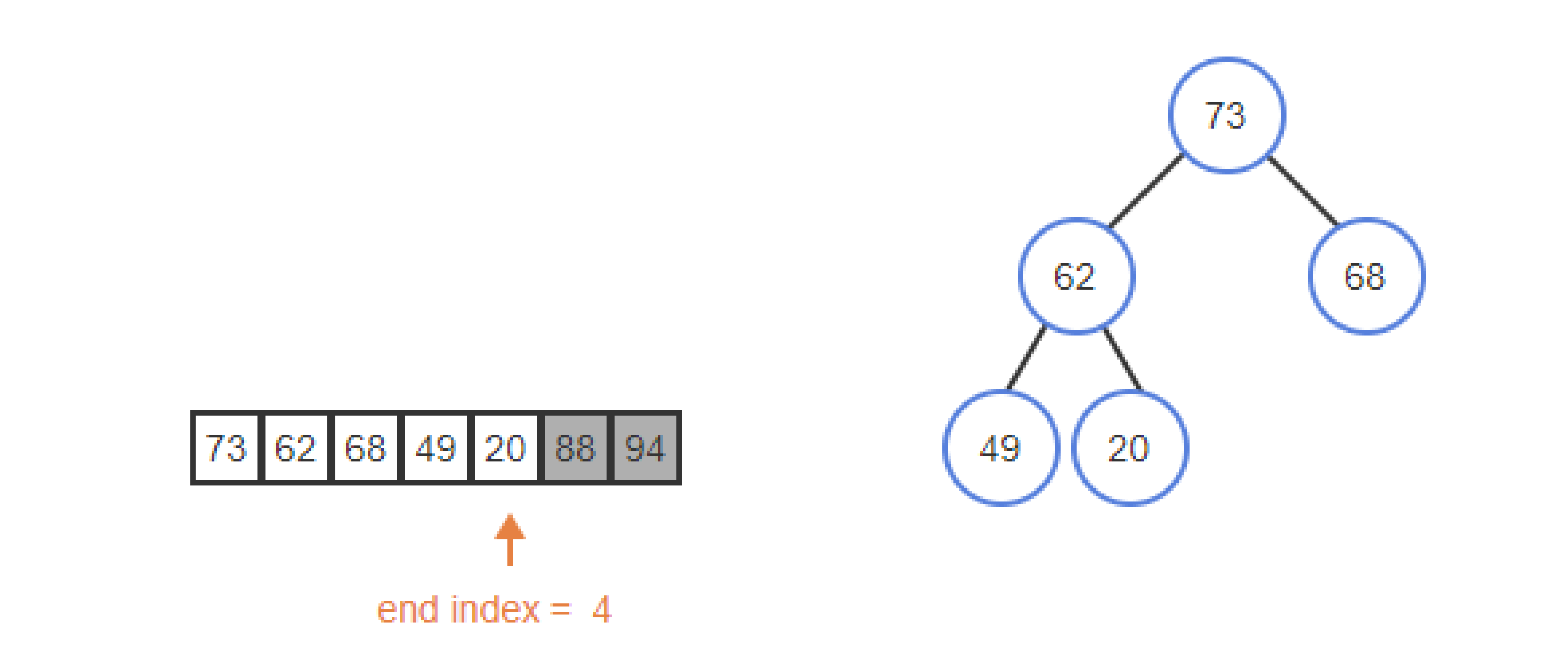
3.d Restore heap with root 73



Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

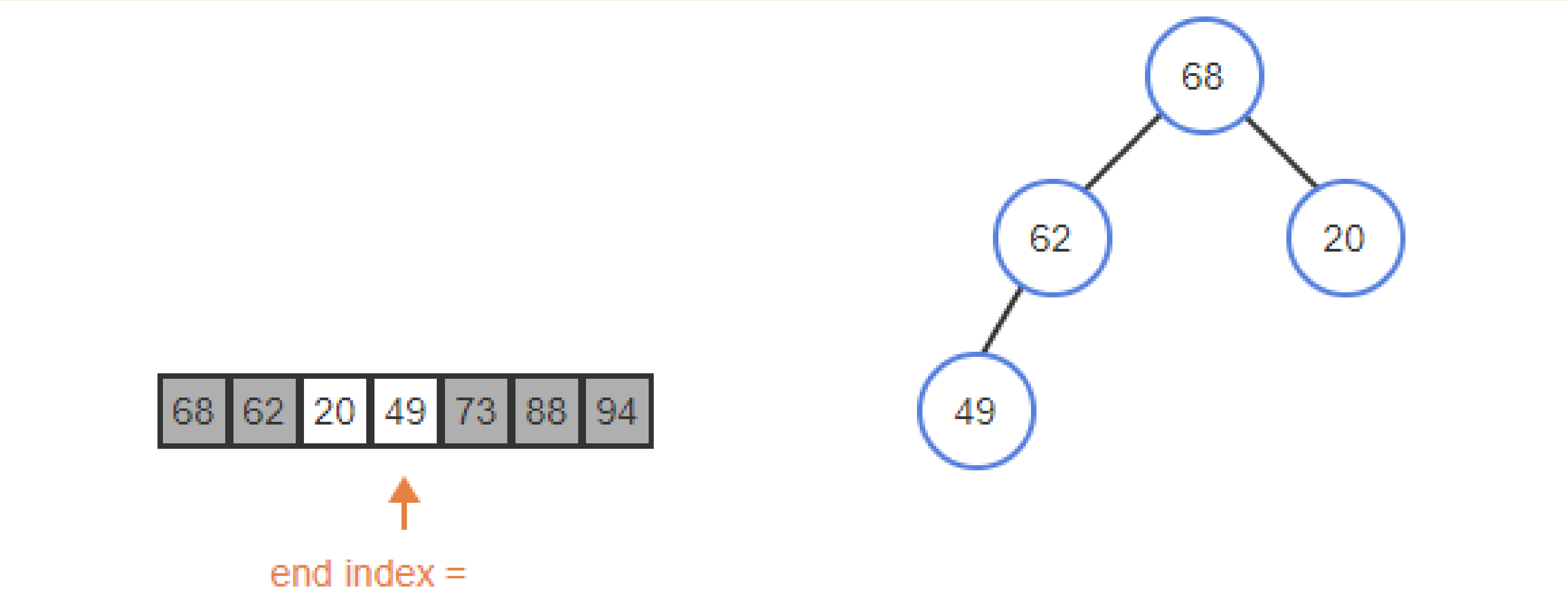
3.e Remove the root 73, move 20 up (swap with 20 in the arrary), restore the heap (20 swap with 68)



Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

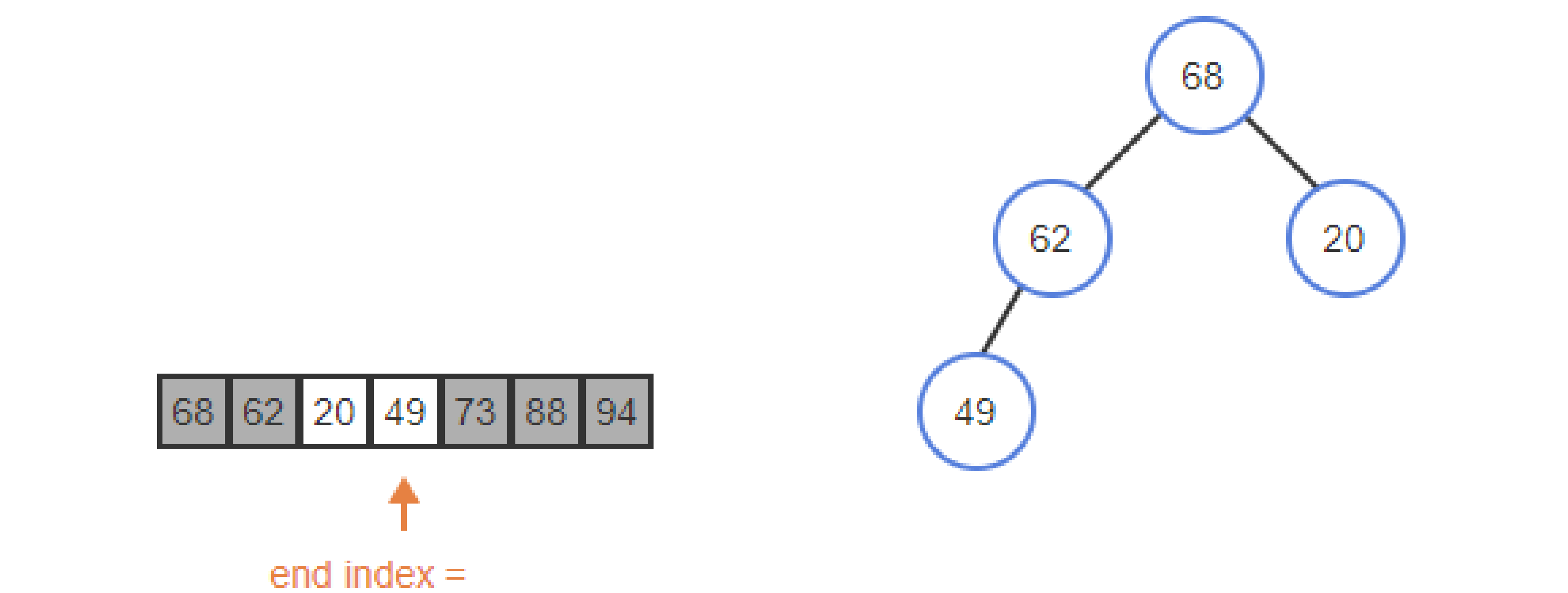
3.f Restore the heap with new root 68



Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

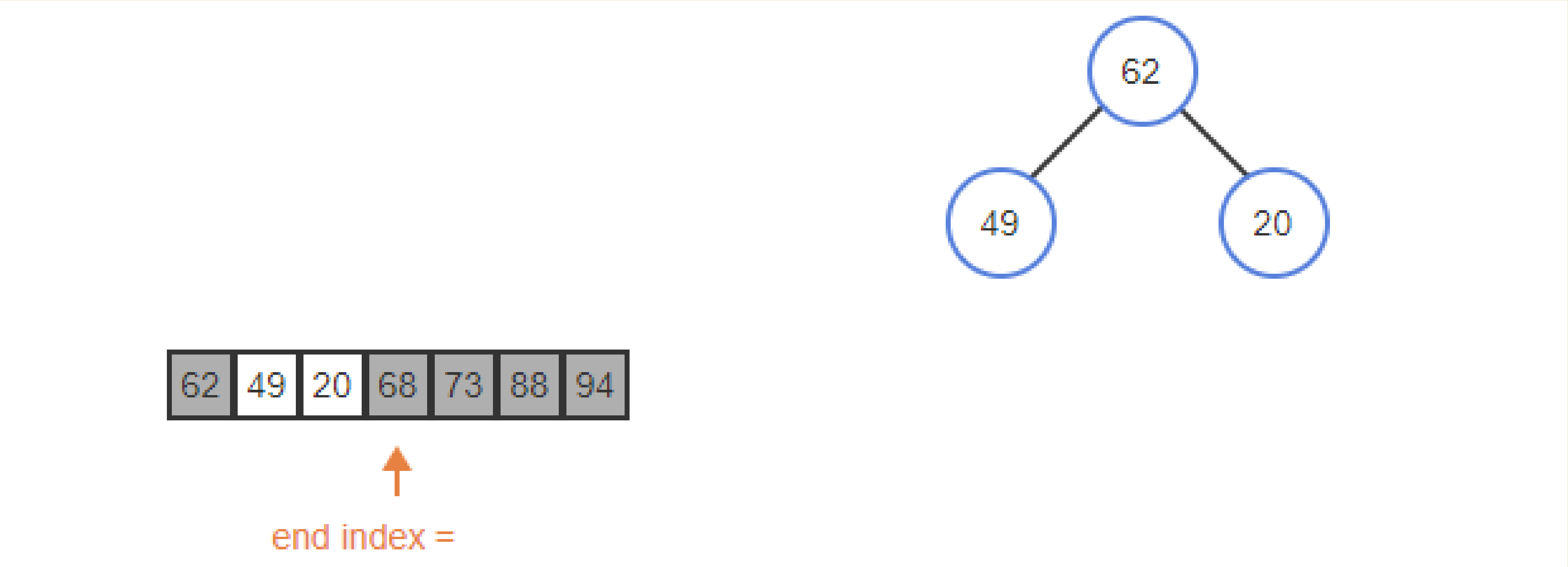
3.g Remove the root 68, move 49 up (swap with 49 in the array), restore the heap (49 swap with 62)



Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

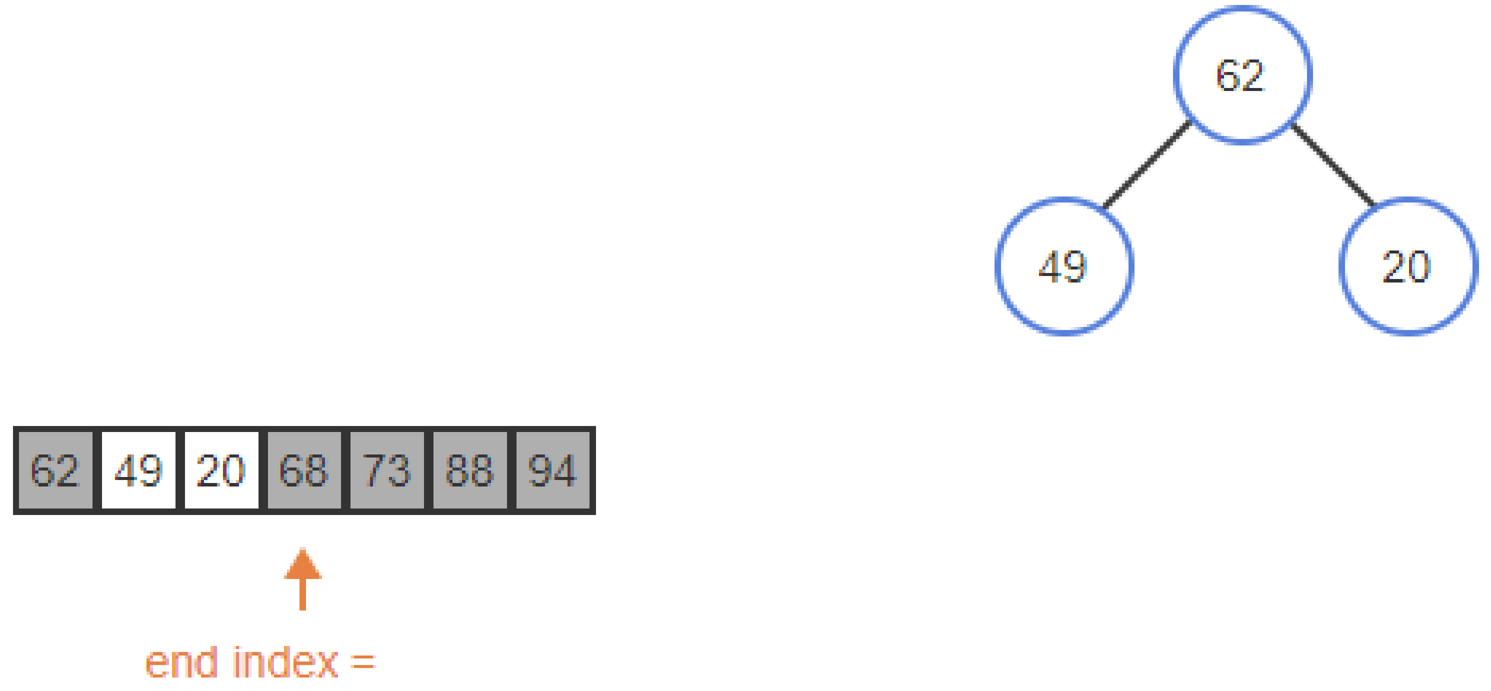
3.h Restore the heap with new root 62



Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

3.i Remove the root 62, move 20 up (swap with 20 in the arrary), restore the heap (20 swap with 49)

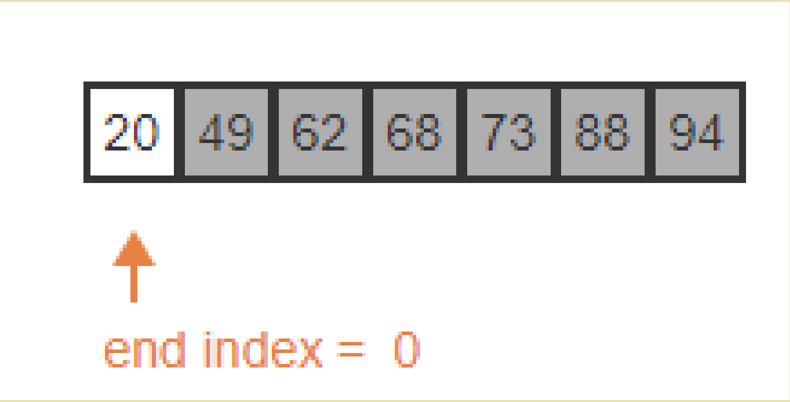


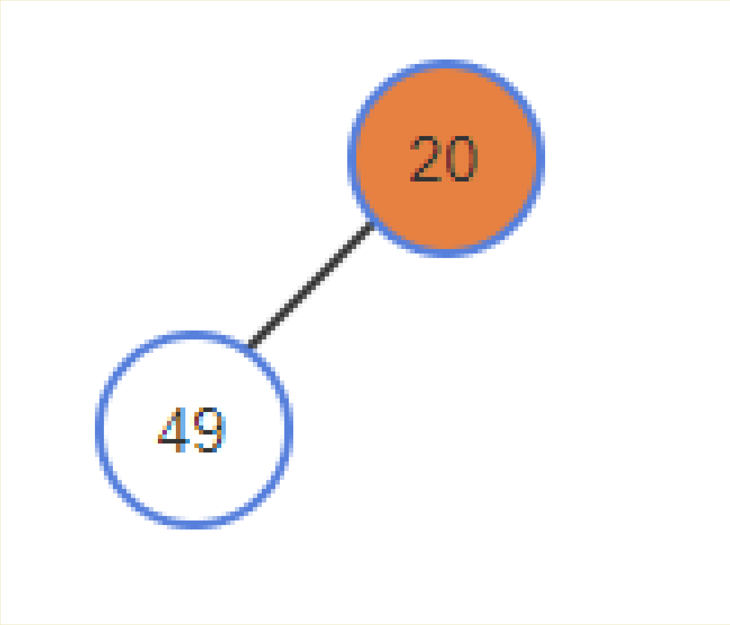
Array-Based Heapsort Example

Sort the array [49, 62, 88, 73, 20, 94, 68] in ascending order by heapsort

3.j Swap the root 20 with 49, remove 49, and remove 20.

Sort completes





References and Useful Resources

- Data Structures Using C, 2nd edition by Reema Thareja
- Differences between mini- and max- heap. https://www.geeksforgeeks.org/difference-between-min-heap-and-max-heap/

That’s about this lecture!