In-place Heap Sort

Heap sorting N items:

- Bottom-up heapify input array.
- Repeat N times:
  - Delete largest item from the max heap, swapping root with last item in the heap.

Input:

In-place Heap Sort: Phase 1: Heapification

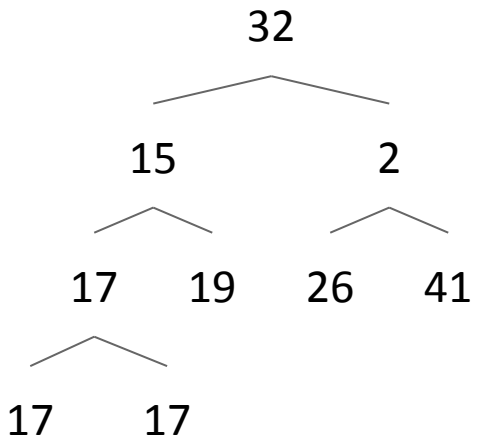
Heap sorting N items:

- **Bottom-up heapify input array:**
  - Sink nodes in reverse level order: sink(k)
  - After sinking, guaranteed that tree rooted at position k is a heap.

Note: This is not a heap yet!
That's why we're heapifying.

Input:

32	15	2	17	19	26	41	17	17

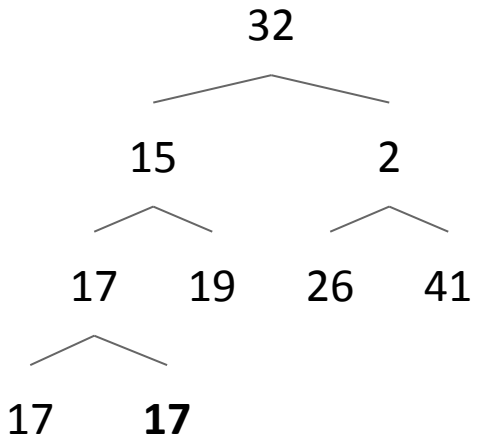


In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - **Sink nodes in reverse level order: sink(k)**
  - After sinking, guaranteed that tree rooted at position k is a heap.

Input:

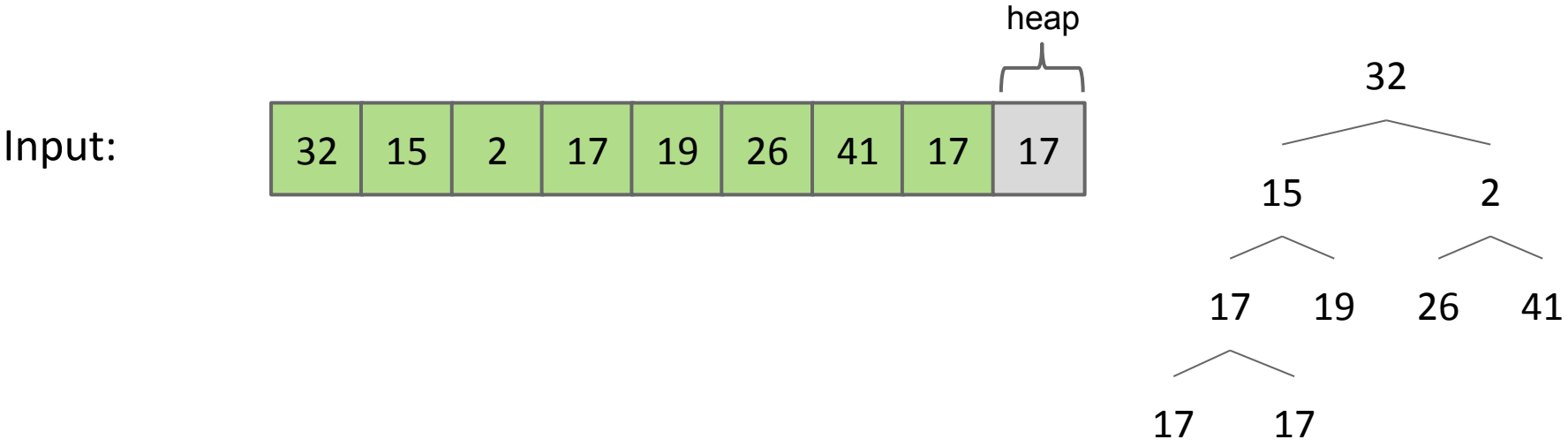


Sinking 17 has no effect.

In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

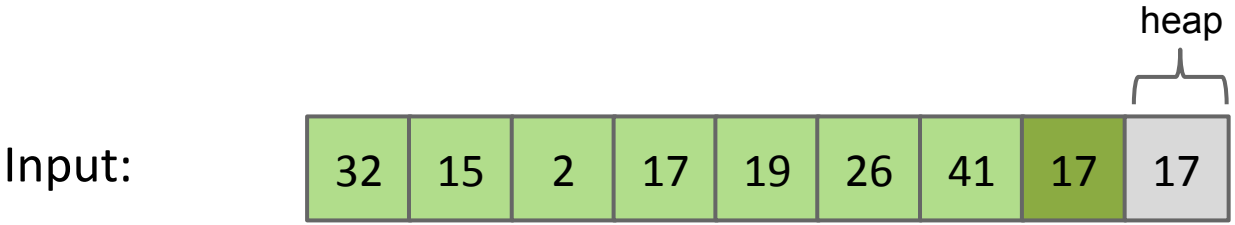
- Bottom-up heapify input array:
  - Sink nodes in reverse level order: sink(k)
  - **After sinking, guaranteed that tree rooted at position k is a heap.**

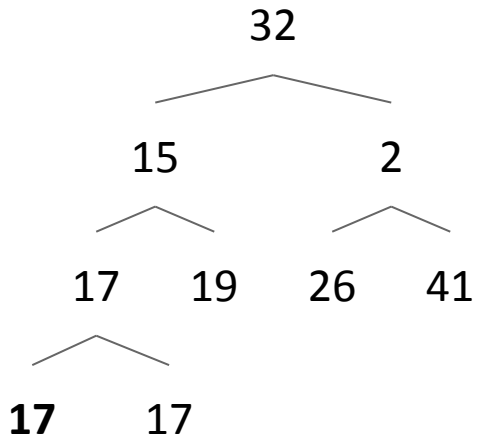


In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - **Sink nodes in reverse level order: sink(k)**
  - After sinking, guaranteed that tree rooted at position k is a heap.



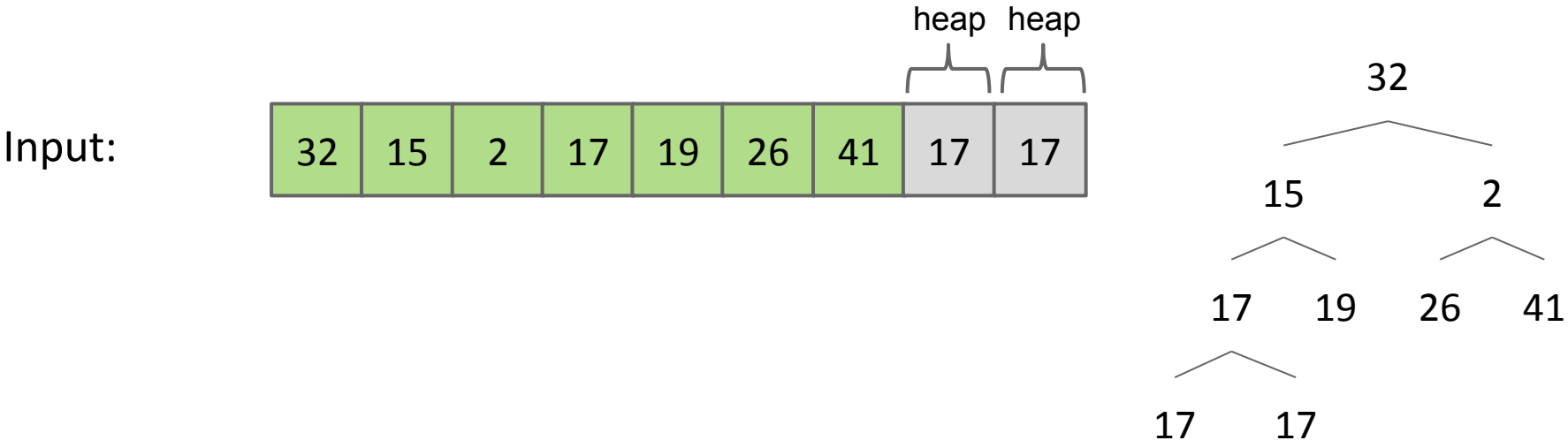


Sinking 17 has no effect.

In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

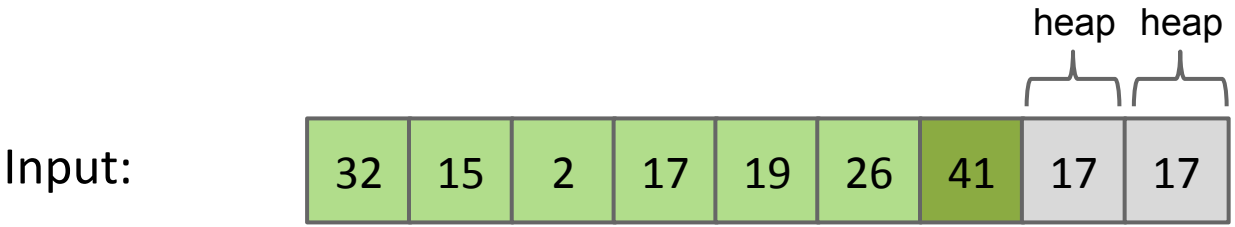
- Bottom-up heapify input array:
  - Sink nodes in reverse level order: sink(k)
  - **After sinking, guaranteed that tree rooted at position k is a heap.**

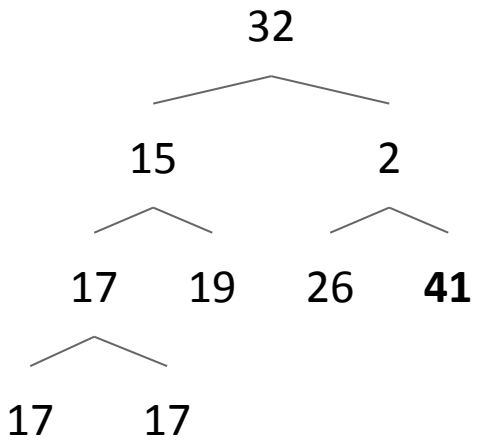


In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - **Sink nodes in reverse level order: sink(k)**
  - After sinking, guaranteed that tree rooted at position k is a heap.



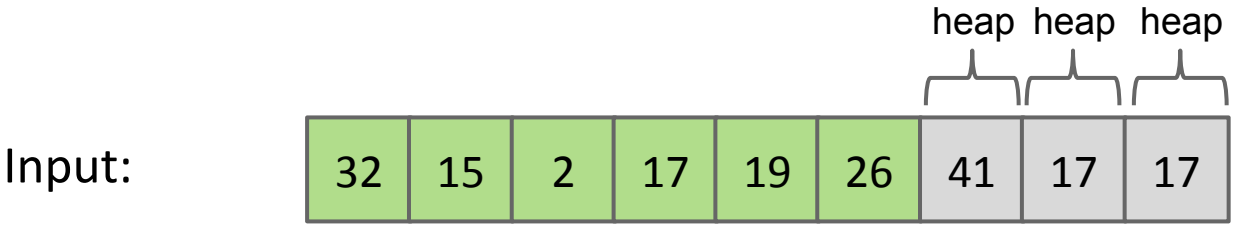


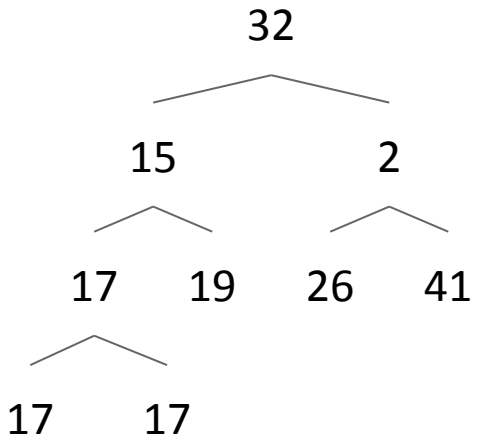
Sinking 41 has no effect.

In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - Sink nodes in reverse level order: sink(k)
  - **After sinking, guaranteed that tree rooted at position k is a heap.**

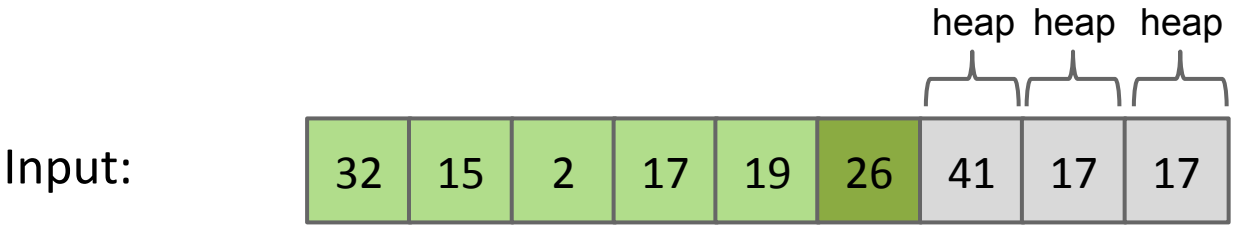


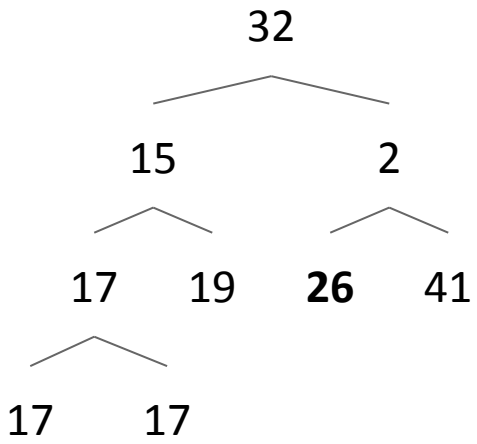


In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - **Sink nodes in reverse level order: sink(k)**
  - After sinking, guaranteed that tree rooted at position k is a heap.



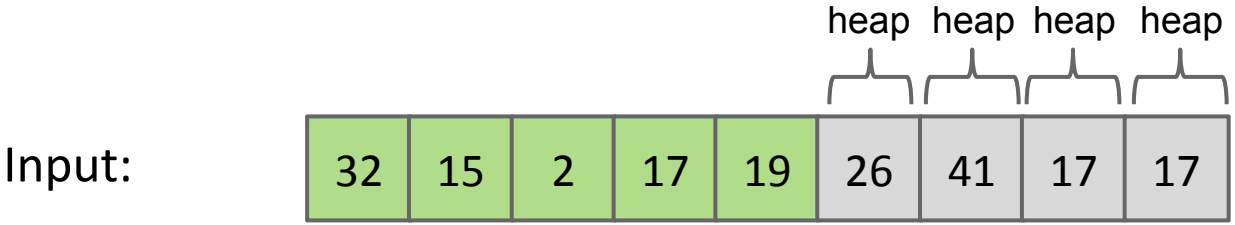


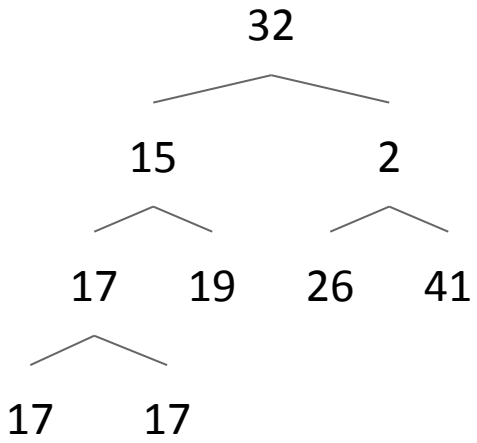
Sinking 26 has no effect.

In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - Sink nodes in reverse level order: sink(k)
  - **After sinking, guaranteed that tree rooted at position k is a heap.**

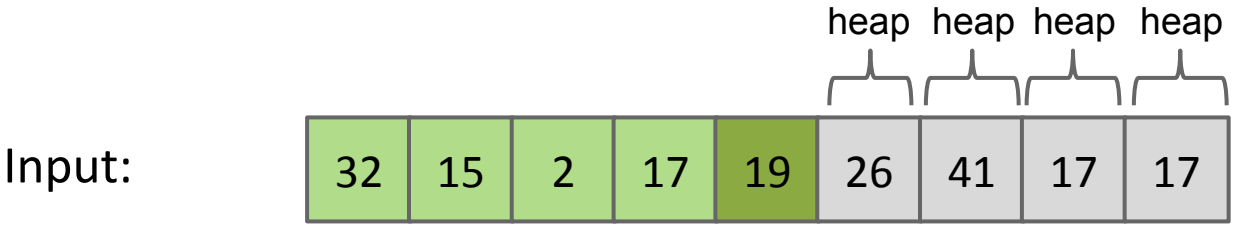


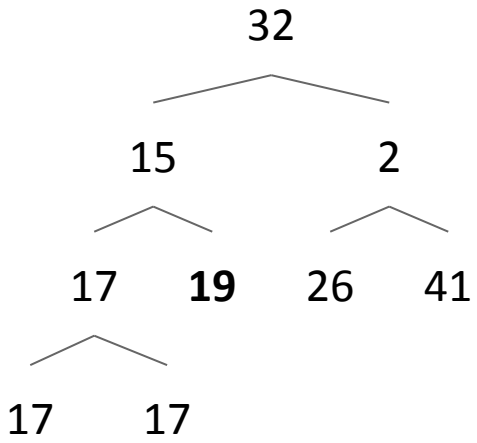


In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - **Sink nodes in reverse level order: sink(k)**
  - After sinking, guaranteed that tree rooted at position k is a heap.



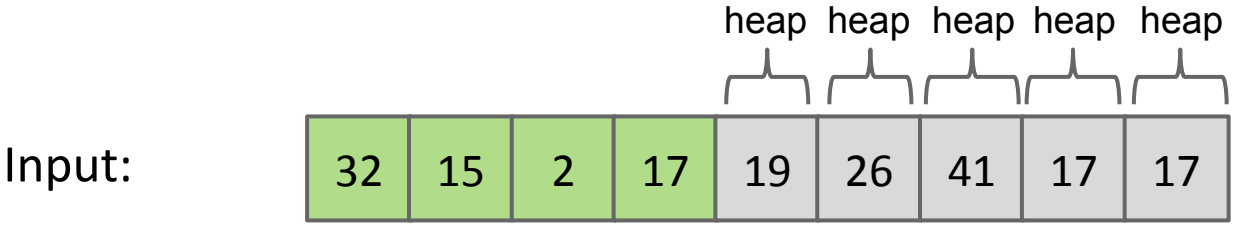


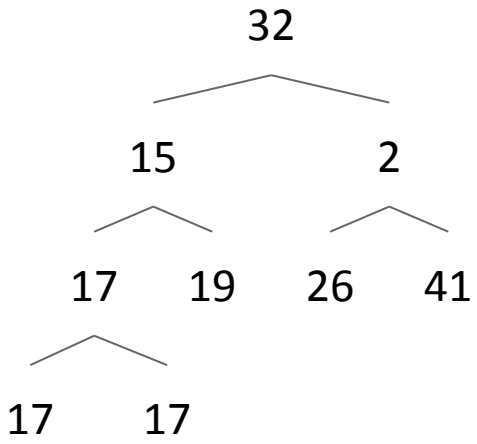
Sinking 19 has no effect.

In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - Sink nodes in reverse level order: sink(k)
  - **After sinking, guaranteed that tree rooted at position k is a heap.**

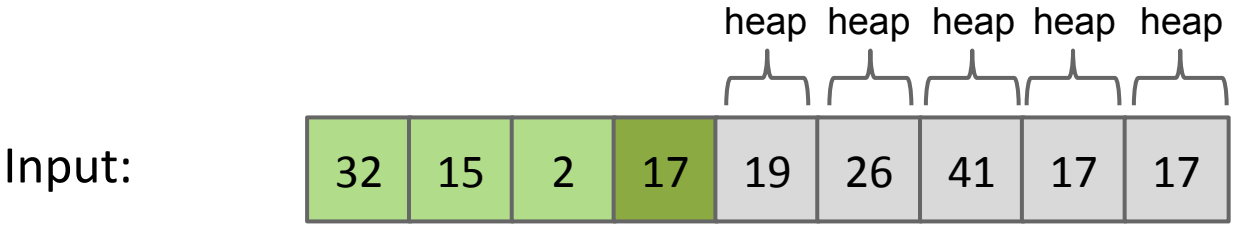


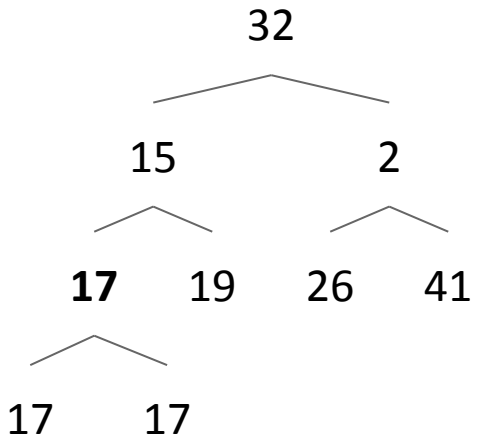


In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - **Sink nodes in reverse level order: sink(k)**
  - After sinking, guaranteed that tree rooted at position k is a heap.



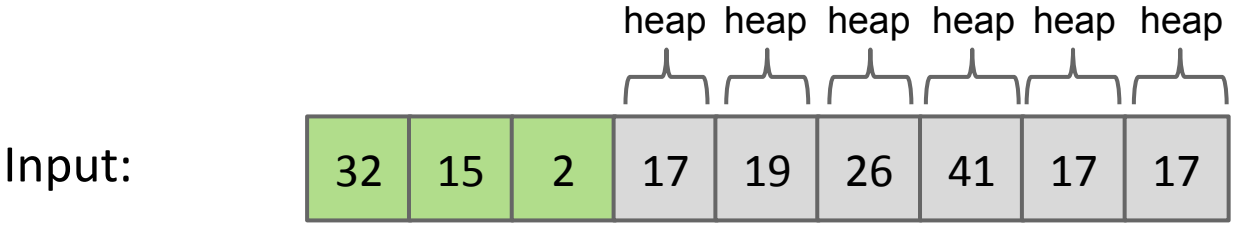


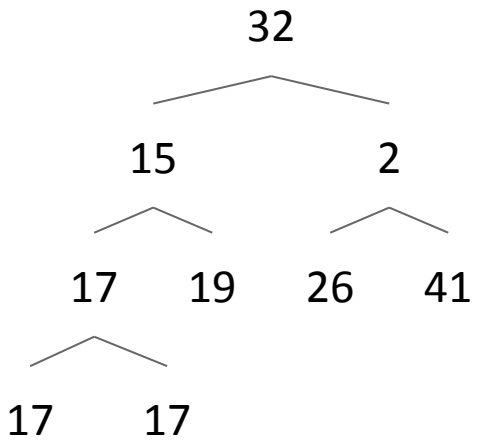
Sinking 17 has no effect.

In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - Sink nodes in reverse level order: sink(k)
  - **After sinking, guaranteed that tree rooted at position k is a heap.**



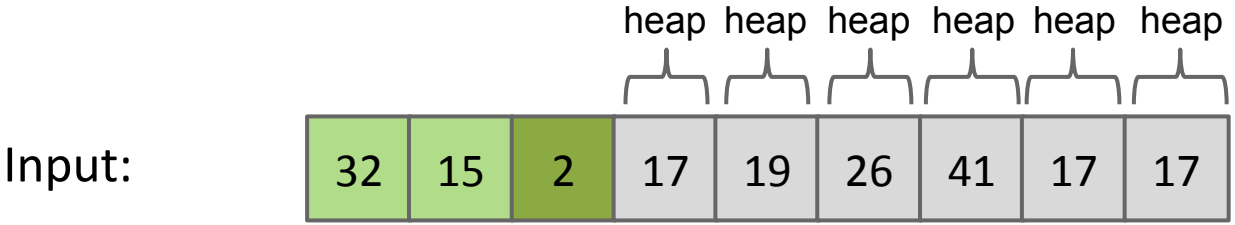


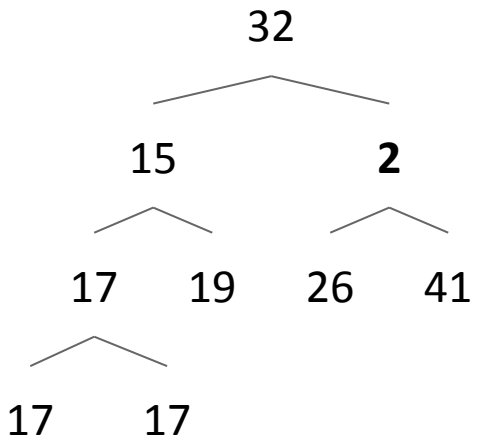
To be clear, each box marked with “heap” is the root of a heap. The leftmost 17 is the root of a 3 element heap, and the last 5 elements are roots of a 1 element heap.

In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - **Sink nodes in reverse level order: sink(k)**
  - After sinking, guaranteed that tree rooted at position k is a heap.



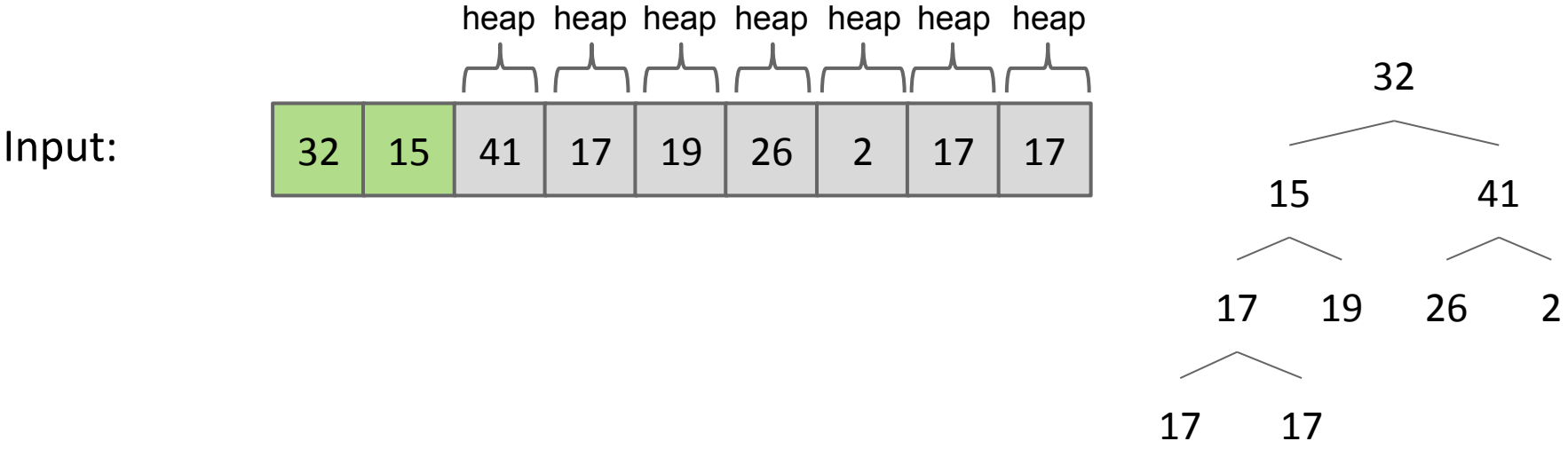


Sinking 2 does something!

In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

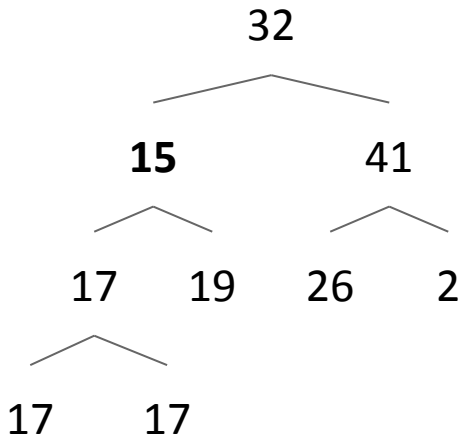
- Bottom-up heapify input array:
  - Sink nodes in reverse level order: sink(k)
  - **After sinking, guaranteed that tree rooted at position k is a heap.**



In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - **Sink nodes in reverse level order: sink(k)**
  - After sinking, guaranteed that tree rooted at position k is a heap.

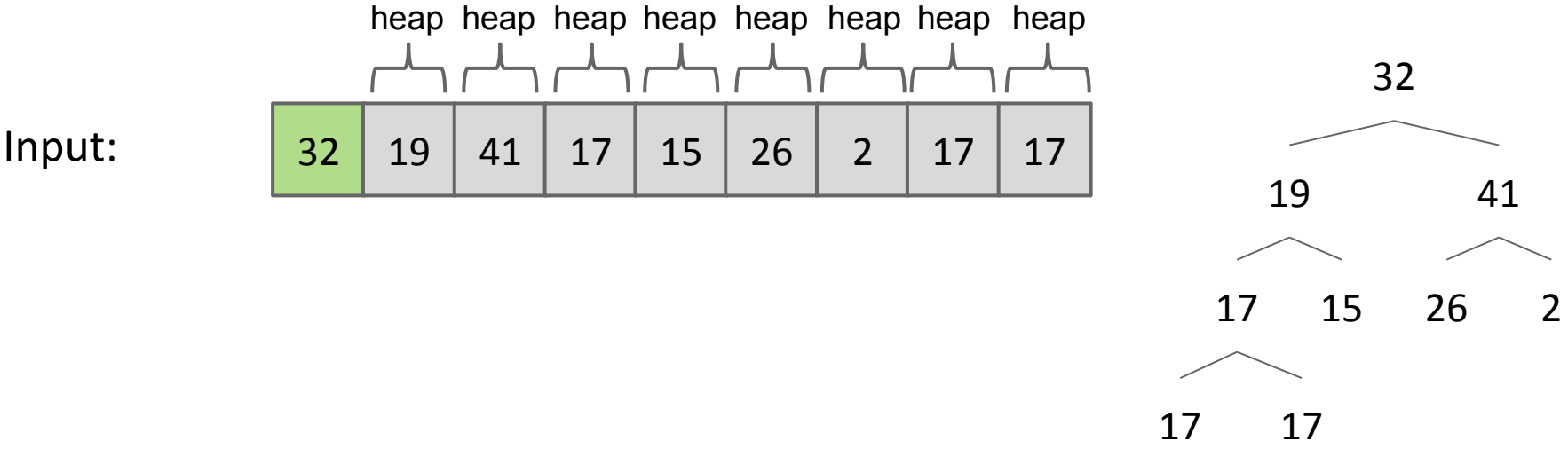


Sinking 15 does something!

In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

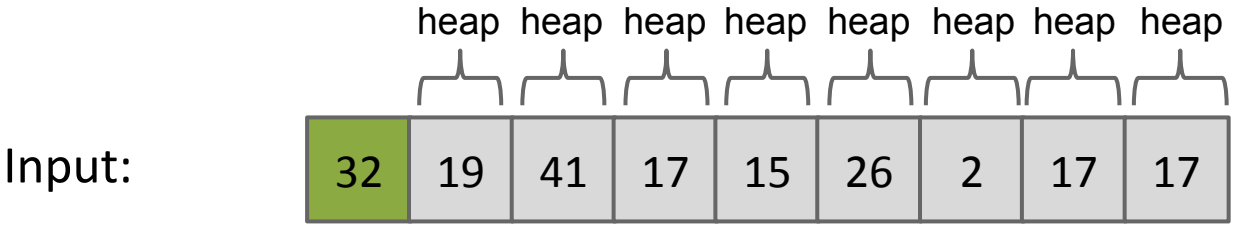
- Bottom-up heapify input array:
  - Sink nodes in reverse level order: sink(k)
  - **After sinking, guaranteed that tree rooted at position k is a heap.**

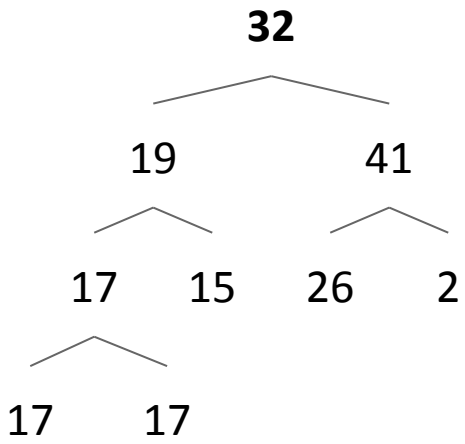


In-place Heap Sort: Phase 1: Heapification

Heap sorting N items:

- Bottom-up heapify input array:
  - **Sink nodes in reverse level order: sink(k)**
  - After sinking, guaranteed that tree rooted at position k is a heap.





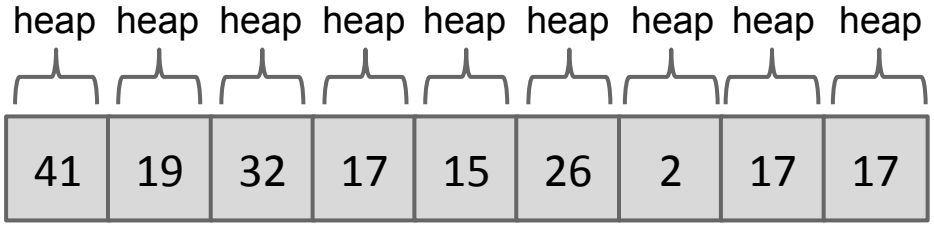
Sinking 32 does something!

In-place Heap Sort: Phase 1: Heapification

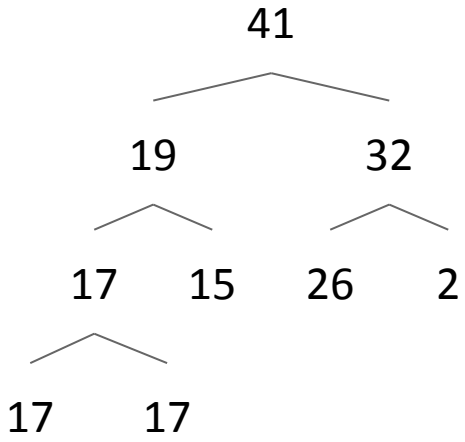
Heap sorting N items:

- Bottom-up heapify input array:
  - Sink nodes in reverse level order: sink(k)
  - **After sinking, guaranteed that tree rooted at position k is a heap.**

Input:



(No room to leave an unused, spot, so we will actually use position zero for this algorithm!)



Punchline: Since tree rooted at position 0 is a heap, then entire array is a heap.

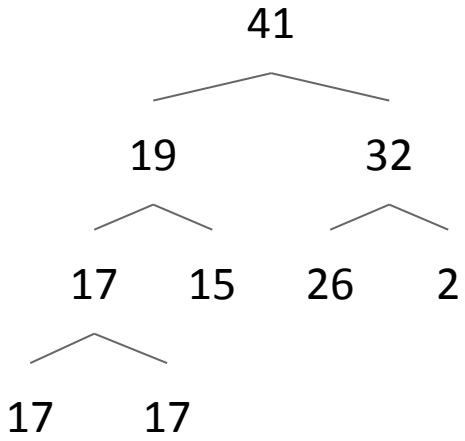
In-place Heap Sort

Heap sorting N items:

- **Bottom-up heapify input array (done!).**
- Repeat N times:
  - Delete largest item from the max heap, swapping root with last item in the heap.

Input:

Size: 9



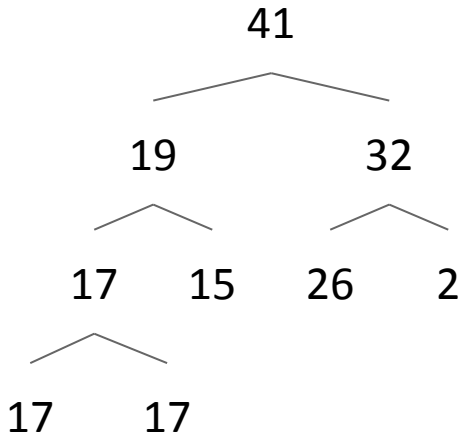
In-place Heap Sort

Heap sorting N items:

- Bottom-up heapify input array (done!).
- Repeat N times:
  - Delete largest item from the max heap, swapping root with last item in the heap.

Input:

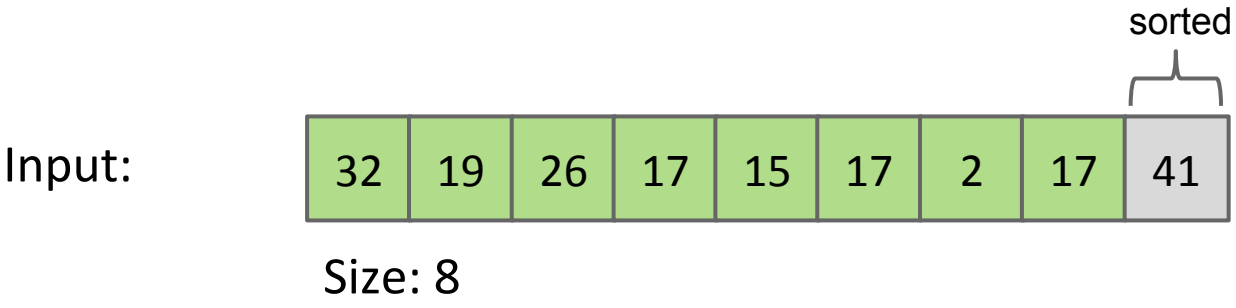
Size: 9

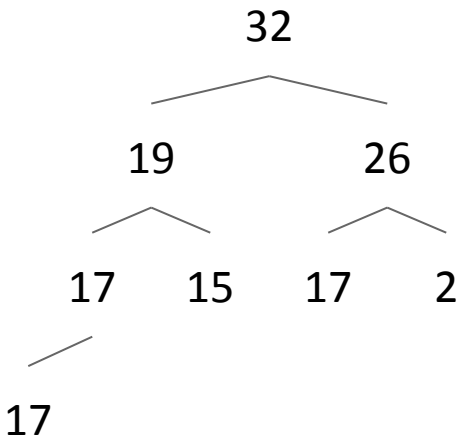


In-place Heap Sort

Heap sorting N items:

- Bottom-up heapify input array (done!).
- Repeat N times:
  - **Delete largest item from the max heap, swapping root with last item in the heap.**

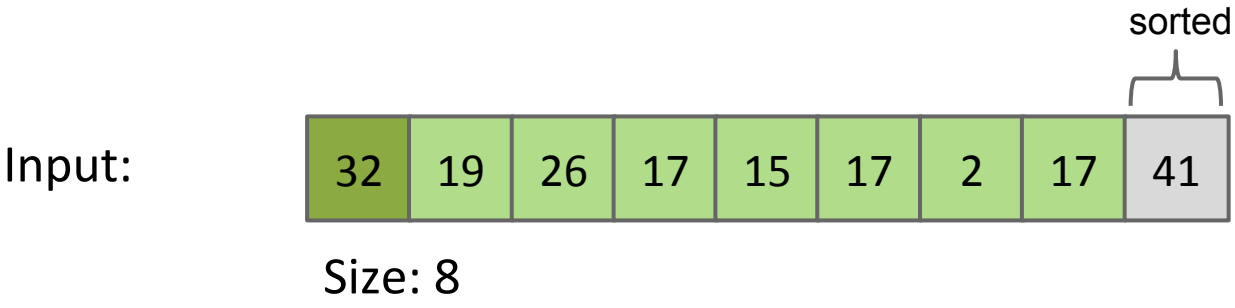


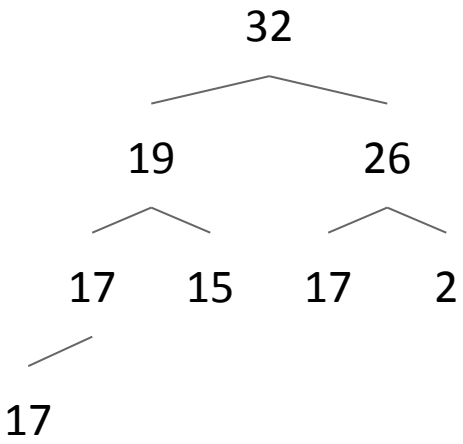


In-place Heap Sort

Heap sorting N items:

- Bottom-up heapify input array (done!).
- Repeat N times:
  - Delete largest item from the max heap, swapping root with last item in the heap.

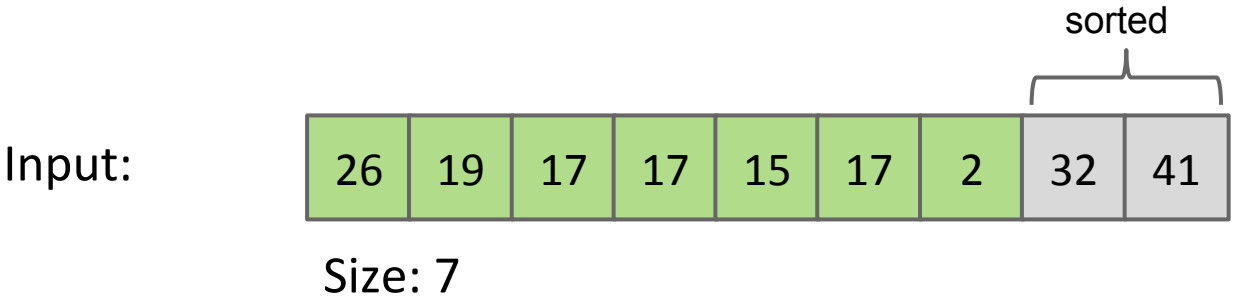


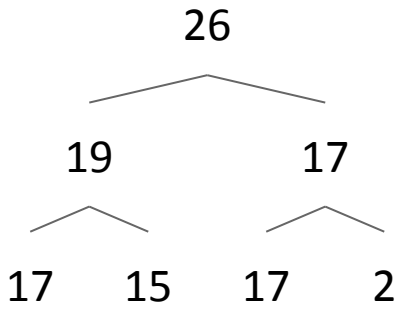


In-place Heap Sort

Heap sorting N items:

- Bottom-up heapify input array (done!).
- Repeat N times:
  - **Delete largest item from the max heap, swapping root with last item in the heap.**

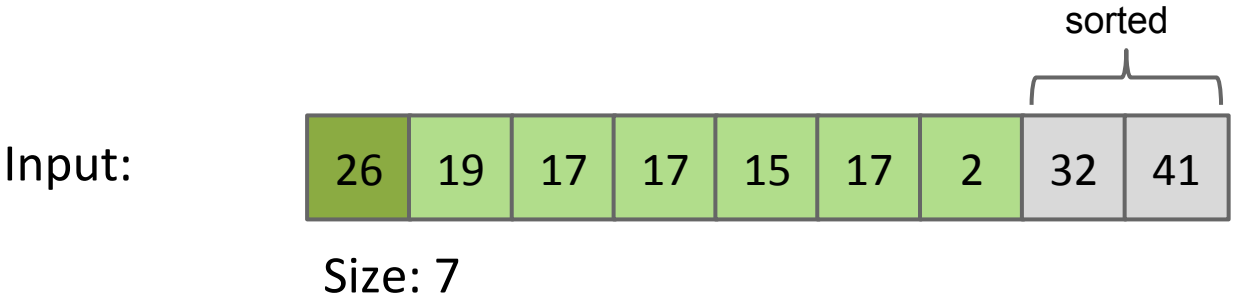


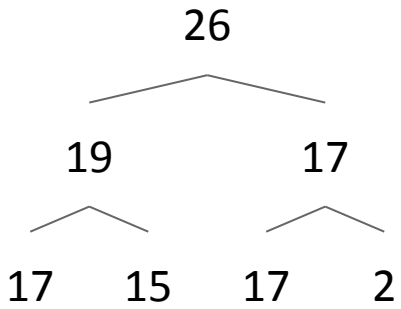


In-place Heap Sort

Heap sorting N items:

- Bottom-up heapify input array (done!).
- Repeat N times:
  - Delete largest item from the max heap, swapping root with last item in the heap.



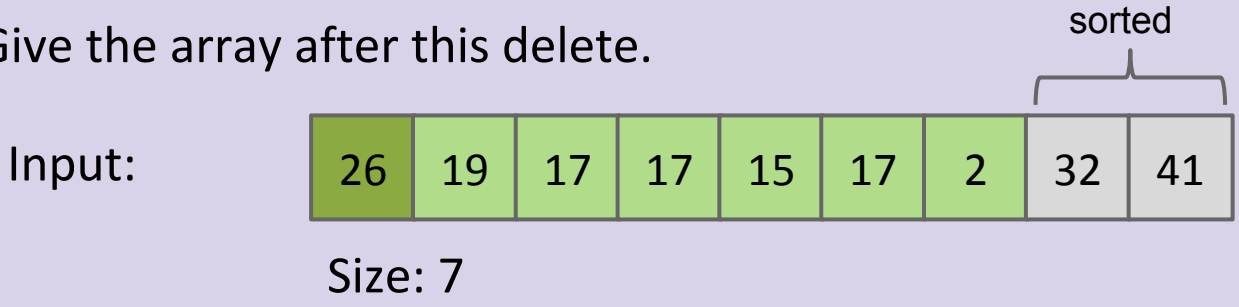


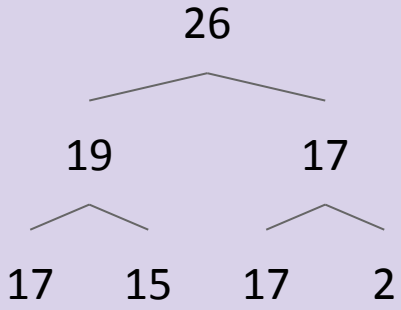
In-place Heap Sort

Heap sorting N items:

- Bottom-up heapify input array (done!).
- Repeat N times:
  - **Delete largest item from the max heap, swapping root with last item in the heap.**

Give the array after this delete.

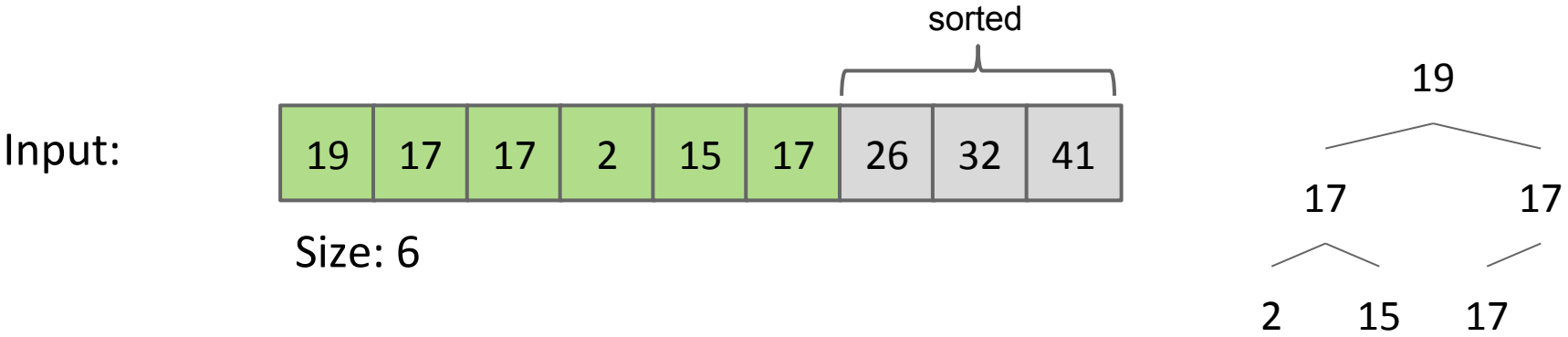




In-place Heap Sort

Heap sorting N items:

- Bottom-up heapify input array (done!).
- Repeat N times:
  - Delete largest item from the max heap, swapping root with last item in the heap.



From here on out, the process is just the same, so verbose steps are omitted...

In-place Heap Sort

Heap sorting N items:

- Bottom-up heapify input array (done!).
- Repeat N times:
  - Delete largest item from the max heap, swapping root with last item in the heap.