CS 261
Data Structures

Lecture 17

Priority Queue & Heap

Heap Sort

7/26/22, Tuesday

Oregon State University

Lecture Topics:

- Priority Queues & Heaps
- Array-based Heaps
- Build a heap from an arbitrary array
- Heapsort

Priority Queues

- *Priority Queue*: an ADT that associates a priority value with each element.

- The element with the highest priority is the first one dequeued.
  - highest priority – element with the lowest priority value

- Interface:
  - **`insert()`** – insert an element with a specified priority value
  - **`first()`** – return the element with the lowest priority value (the "first" element in the priority queue)
  - **`remove_first()`** – remove (and return) the element with the lowest priority value

Priority Queues Visualization

- The user's view of a priority queue:

- A priority queue is typically implemented using a data structure called a ***heap***

Heaps

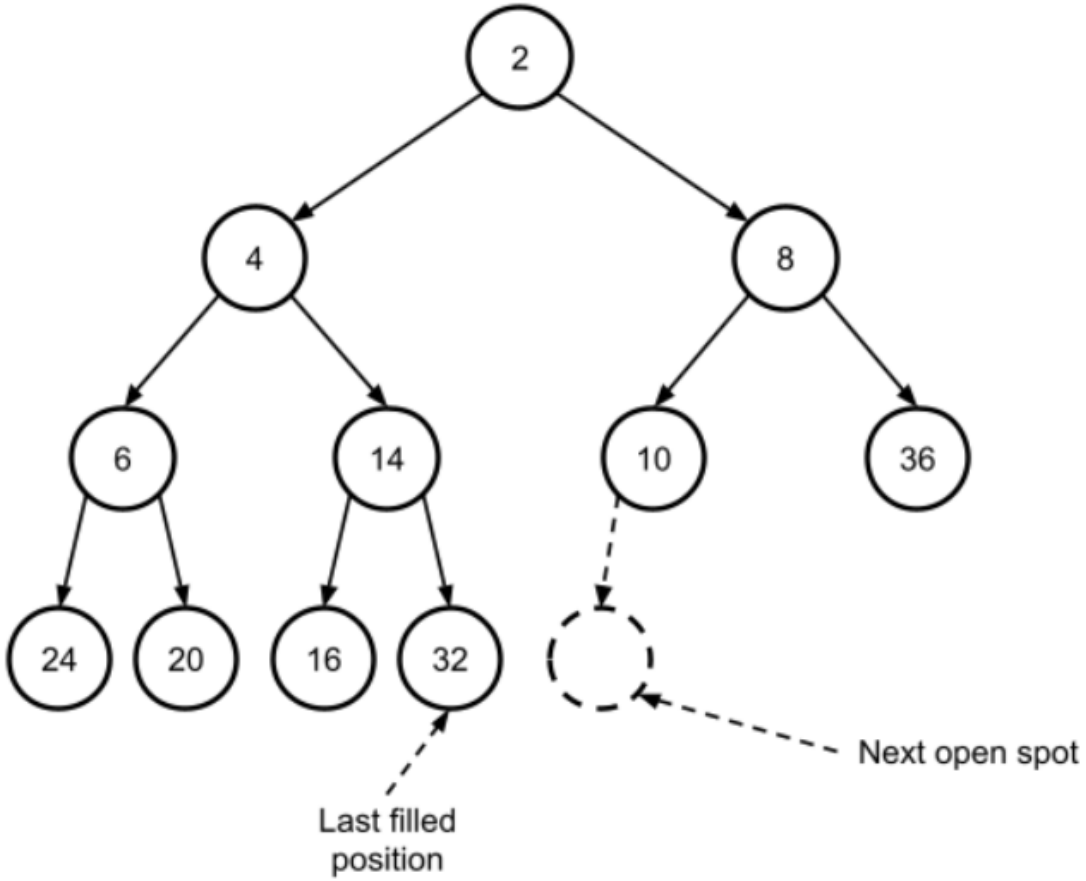
- Caveat: The heap data structure ≠ the dynamic memory space "heap"

- A heap data structure: a ***complete*** binary tree in which every node's value is less than or equal to the values of its children
  - This is called a minimizing binary heap, or just "min heap".
  - max heap: each node's value is greater than or equal to the values of its children

- Recall: a complete binary tree is one that is filled, except for the bottom level, which is filled from left to right
  - The longest path from root to leaf in such a tree is O(log n).

Min Heap Example

- With only priority values displayed:



Add a node to a Heap

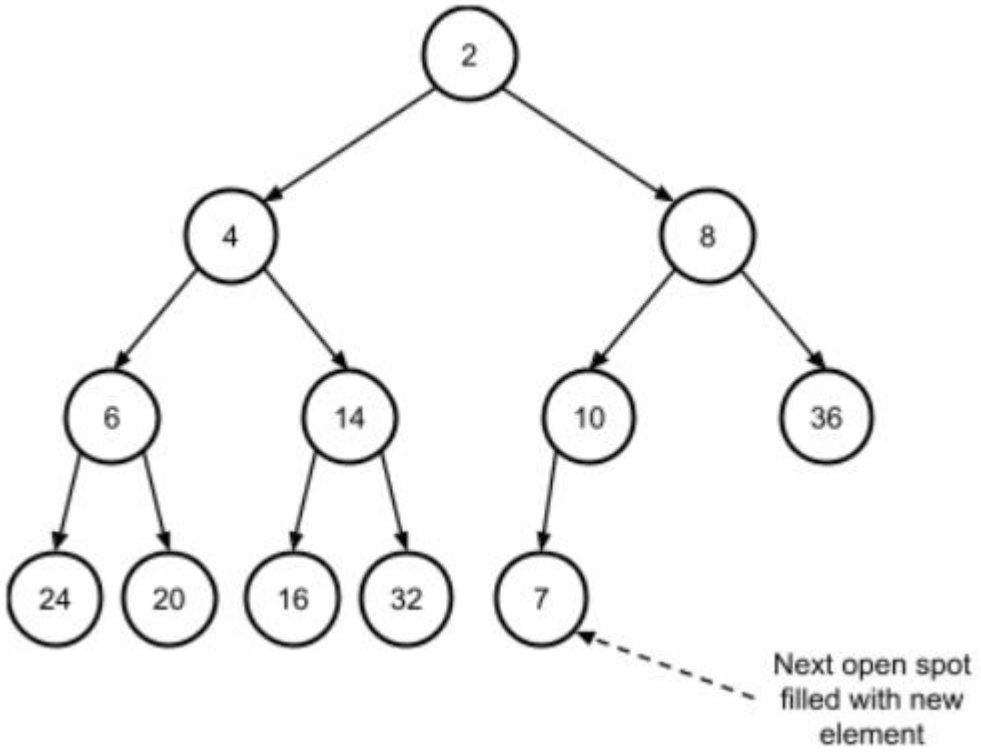
- A min (or max) heap is maintained through the addition and removal of nodes via **percolations**
  - **Percolation – move nodes up and down the tree according to their priority values.**

- When adding a value to a heap,
  - place it into the next open spot
  - percolate it up the heap until its priority value is less than both of its children

Add a node to a Heap

- Example: adding the value 7 to the min heap:

1. place it in the next open spot

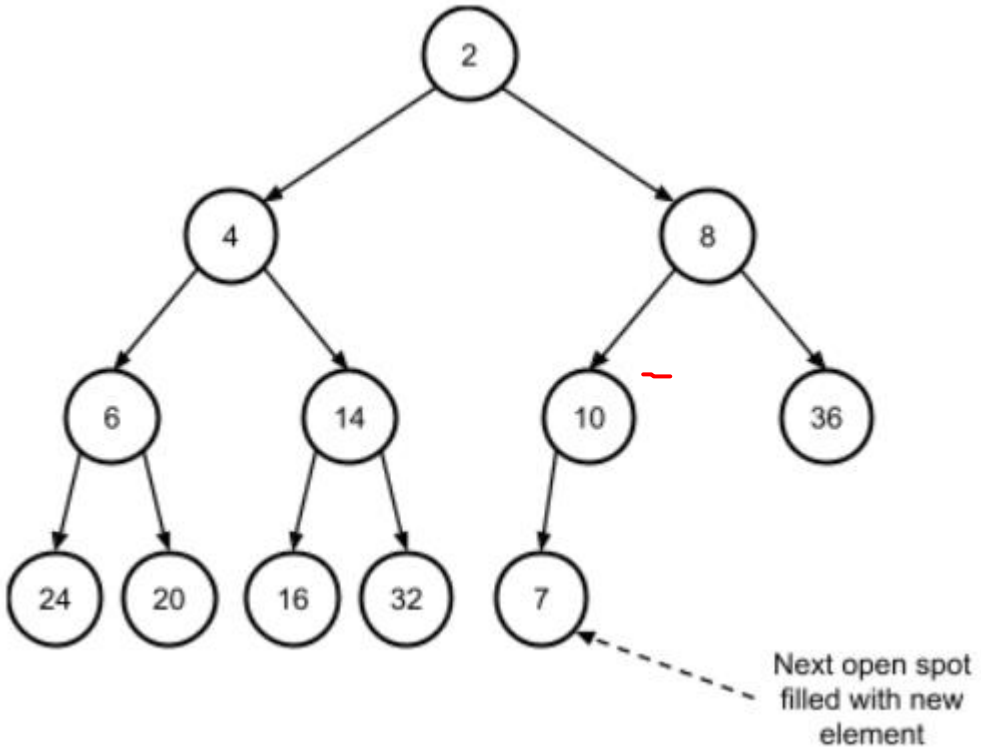


```
while new priority value < parent's priority value:
    swap new node with parent
```

Add a node to a Heap

- Example: adding the value 7 to the min heap:

2. percolate the new element up the tree

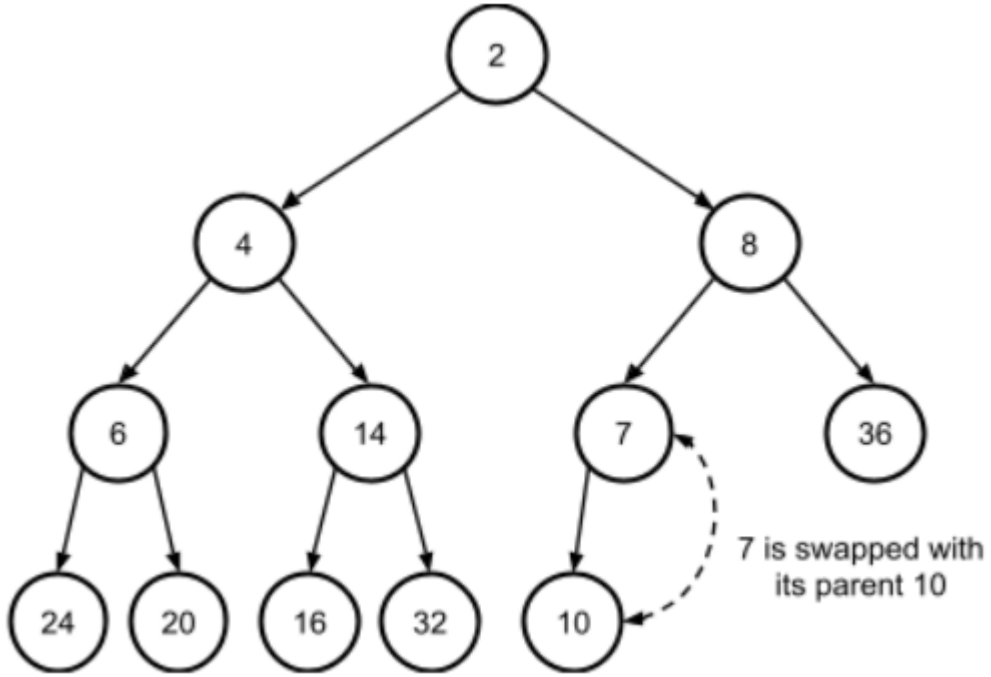


```
while new priority value < parent's priority value:
    swap new node with parent
```

Add a node to a Heap

- Example: adding the value 7 to the min heap:

2.1. compare the new node (7) with its parent (10) and see that they needed to be swapped to maintain the min heap property:

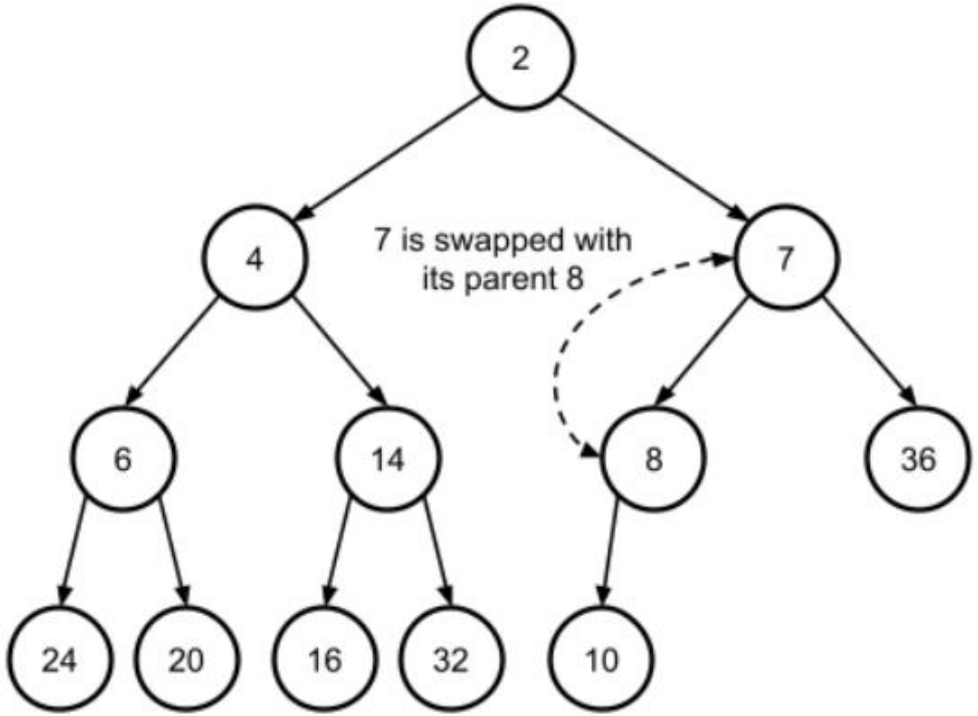


```
while new priority value < parent's priority value:
    swap new node with parent
```

Add a node to a Heap

- Example: adding the value 7 to the min heap:

2.2. compare the new node (7) with its new parent (8) and see that they too needed to be swapped:

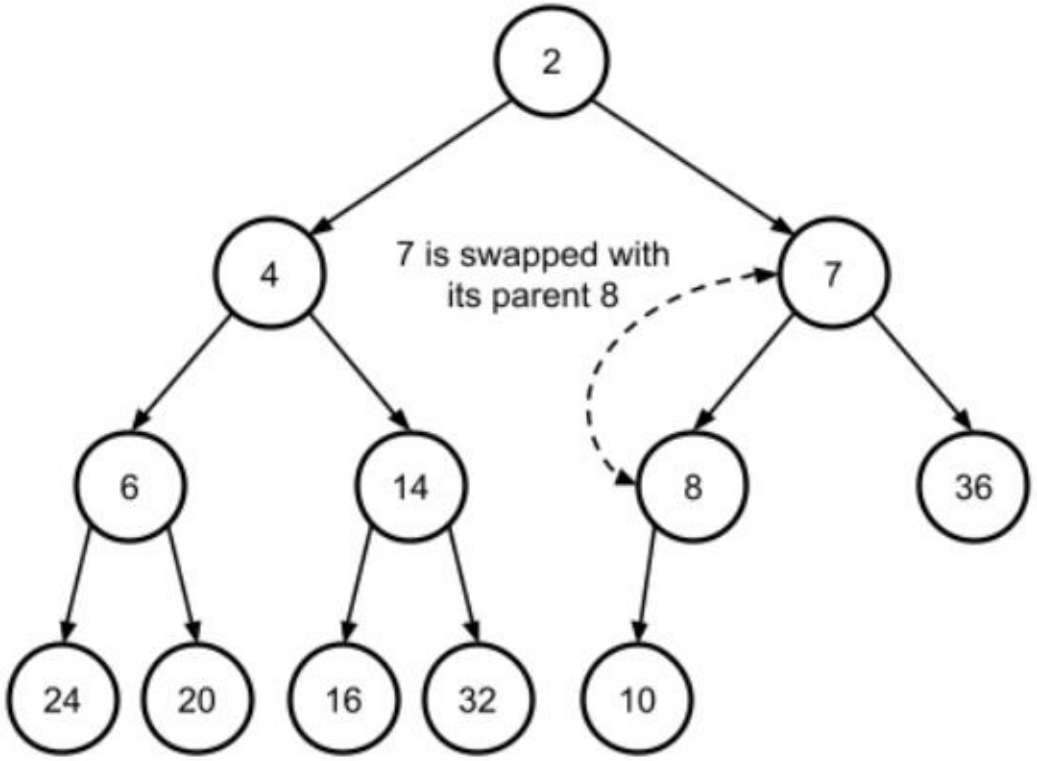


```
while new priority value < parent's priority value:
    swap new node with parent
```

Add a node to a Heap

height

- Runtime Complexity of percolation: O(log n)



Remove a node from a Heap

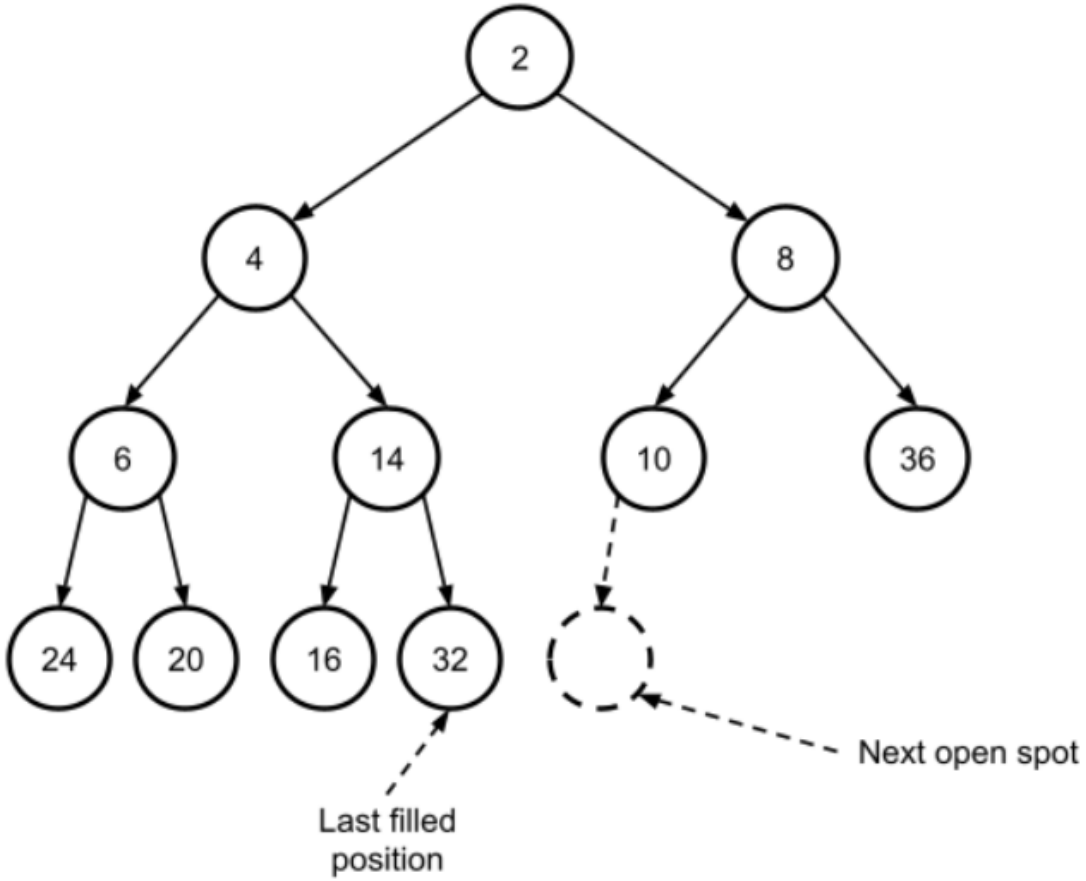
- In a min heap, the root node's priority value is always the lowest
  - the `first()` and `remove_first()` always access and remove the root node

- Question: If we always remove the root node, how do we replace it?
  - Remember, we need to maintain the completeness of the binary tree.

- Answer: replace it with the element last added to the heap and then fix the heap by percolating that node down

Remove a node from a Heap

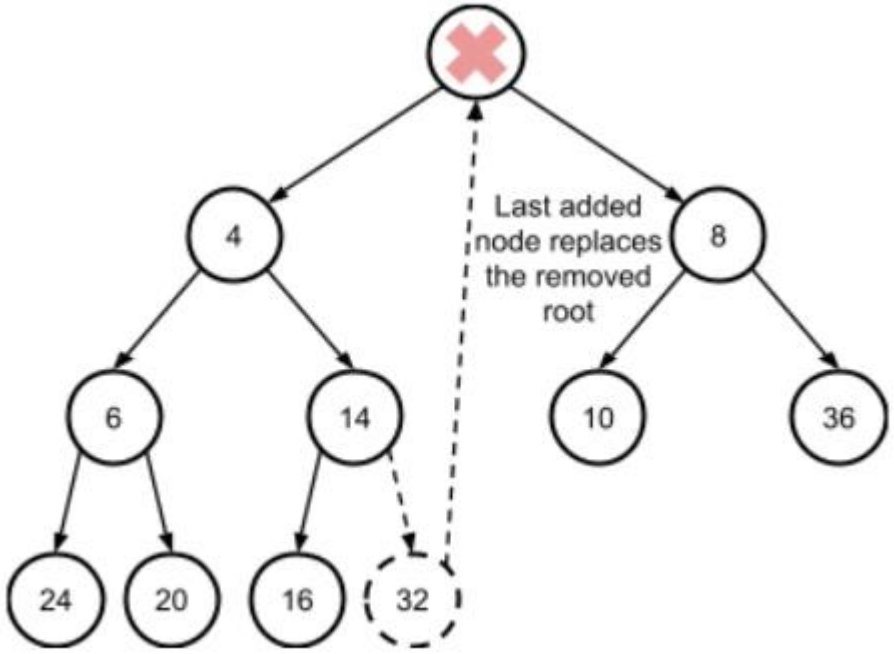
- Example: remove the root node (2) from that heap:

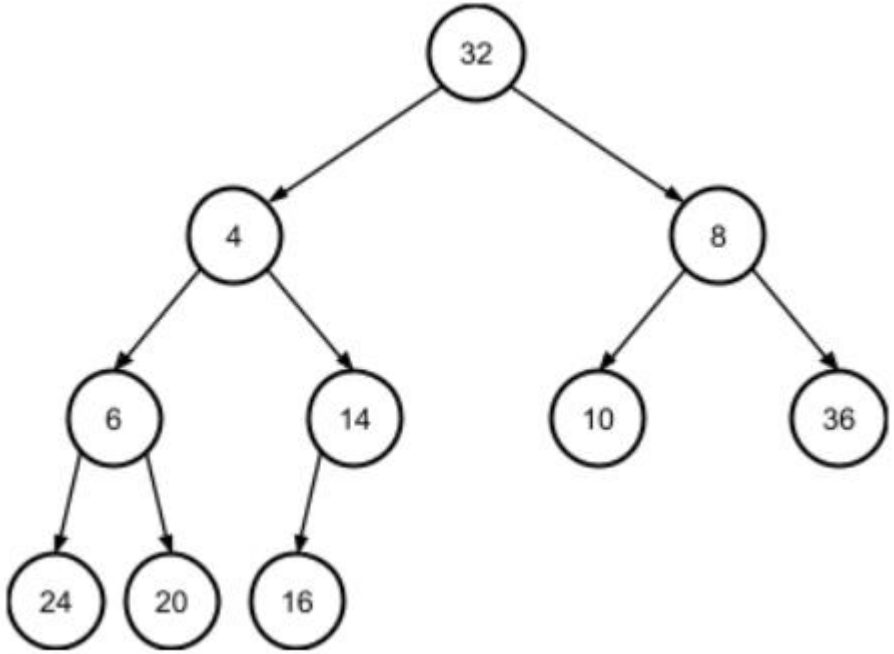


Remove a node from a Heap

- Example: remove the root node (2) from that heap:

1. replace it with the last added node (32)



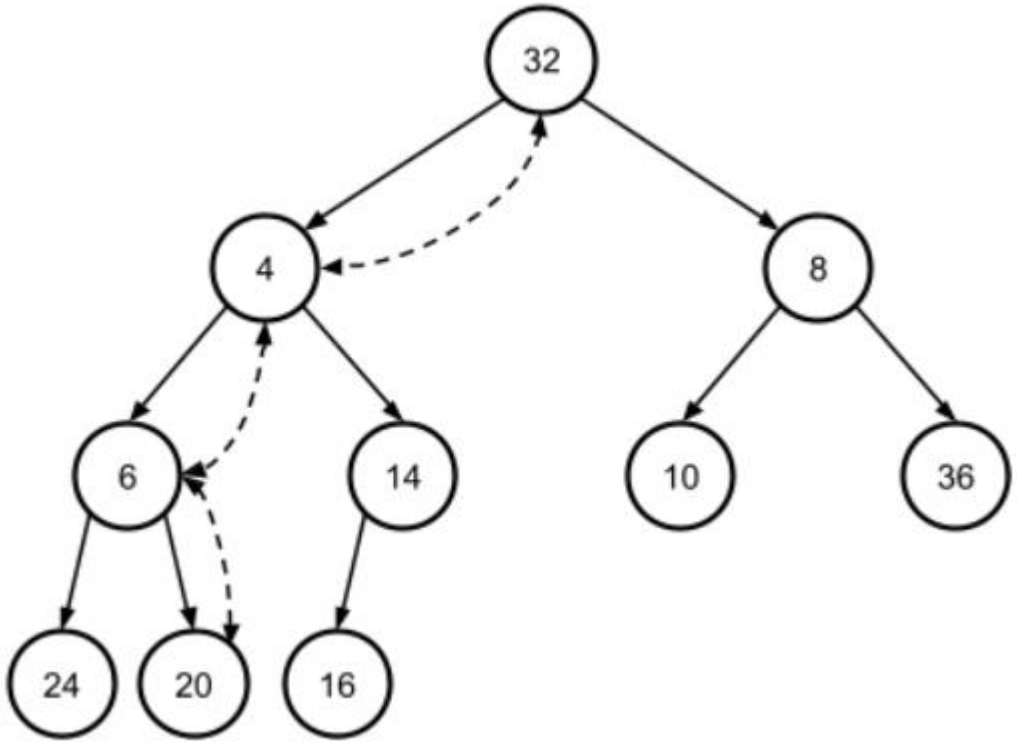


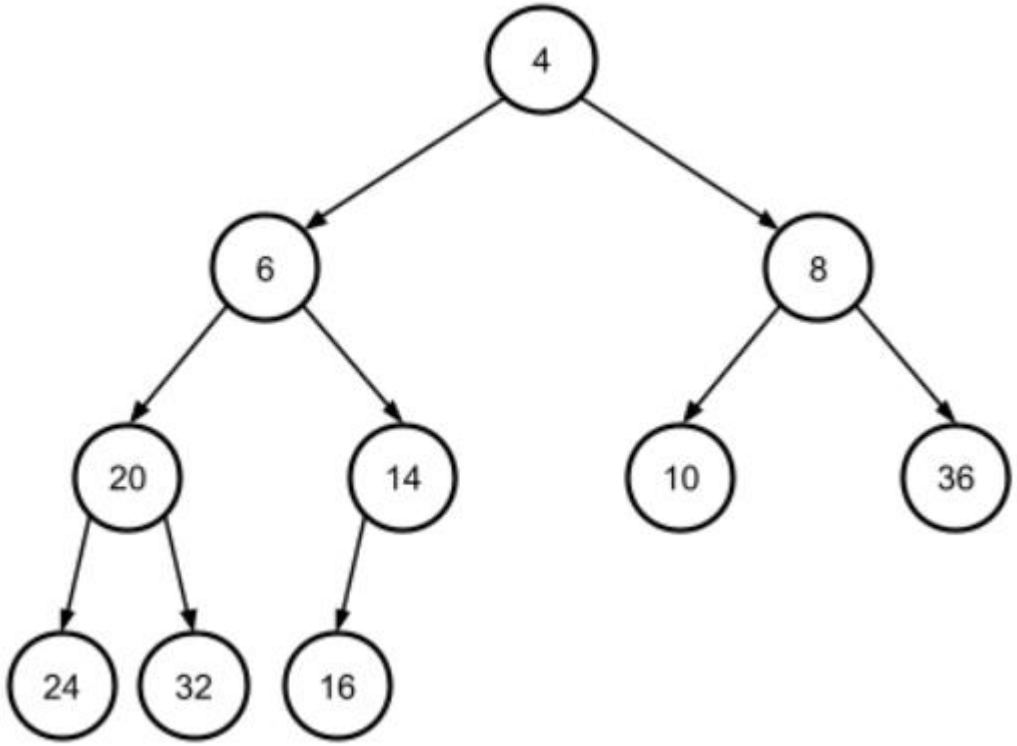
```
while priority > smallest child priority:
      swap with smallest child
```

Remove a node from a Heap

- Example: remove the root node (2) from that heap:

2. percolate the replacement node down the tree





Lecture Topics:

- Priority Queues & Heaps
- Array-based Heaps
- Build a heap from an arbitrary array
- Heapsort

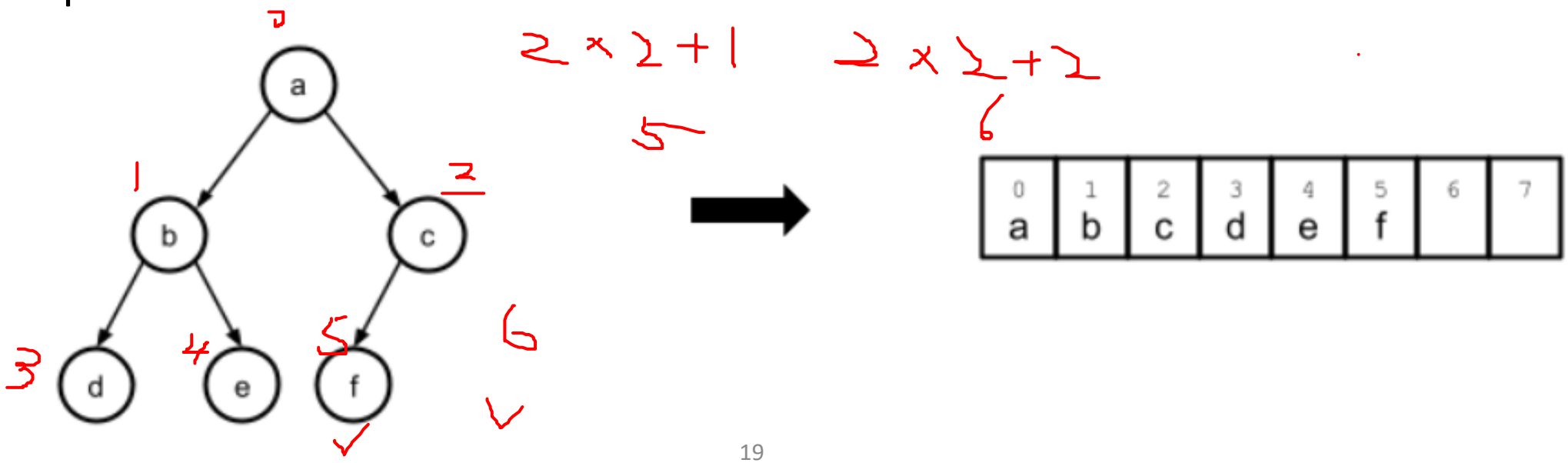
Implement a Heap

- Many ways to implement a heap…
- Recall: a heap data structure contains a complete binary tree
- Then…

Implement a Heap

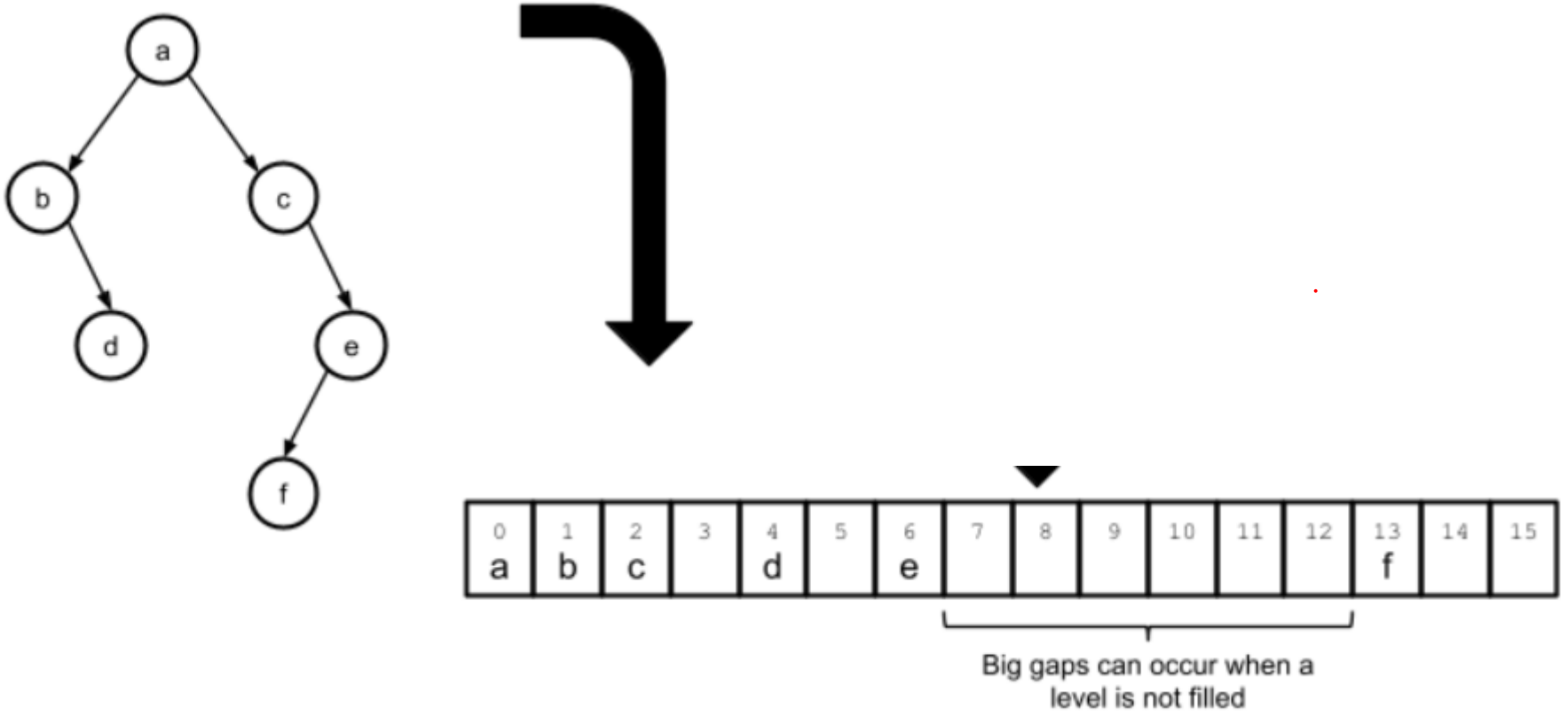
- Implement the complete binary tree representation of a heap using an array:
  - root node of the heap is stored at index 0
  - The left and right children of a node at index i are stored respectively at indices 2 * i + 1 and 2 * i + 2
  - The parent of a node at index i is at (i - 1) / 2 (using the floor that results from integer division).

- Example:



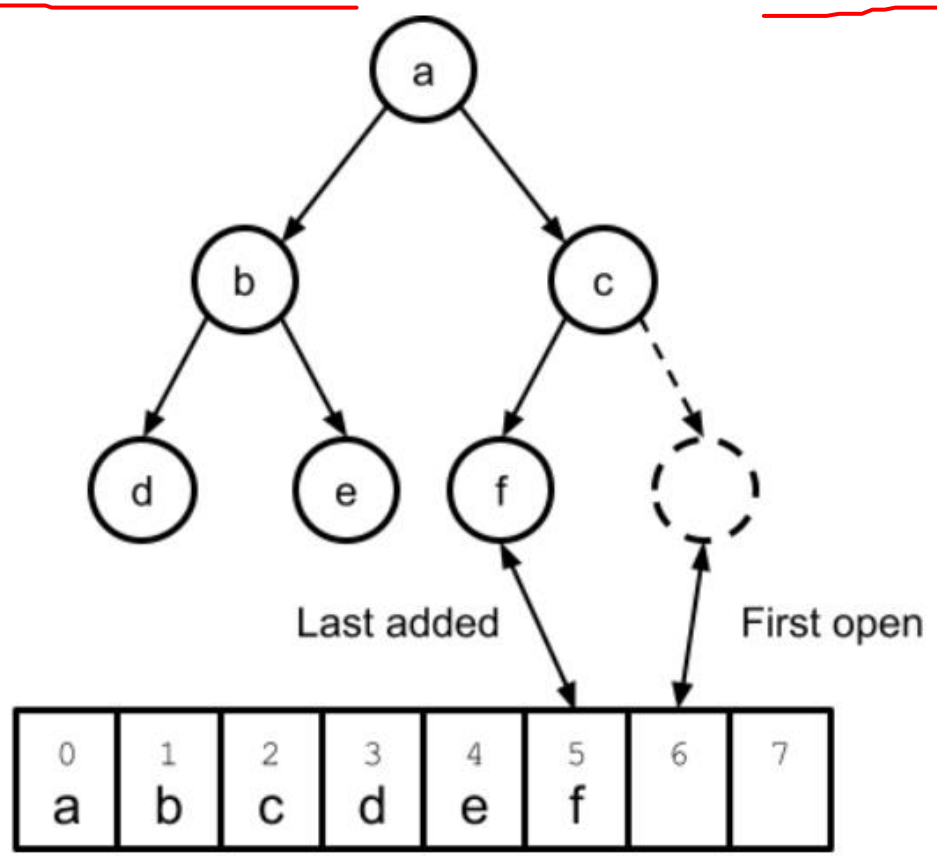
Implement a Heap

- Q: Can you implement a binary tree that was not complete using an array?
- A: No!
- Example:



Implement a Heap

- Keeping track of the last added element and the first open spot in the array representation of the heap is simple
  - simply the last element in the array and the following empty spot
- Example:

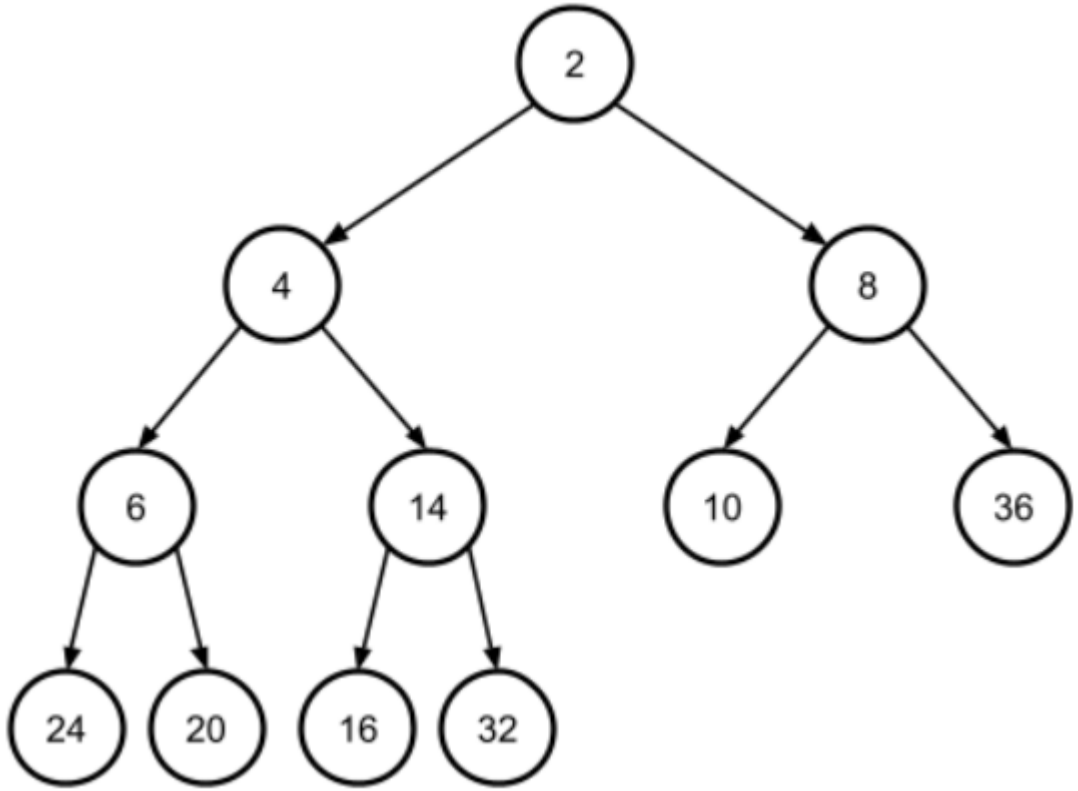


Inserting into an array-based Heap

- Inserting an element into the array representation of the heap follows this procedure:
  1. Put new element at the end of the array.
  2. Compute the inserted element's parent index `((i - 1) / 2)`.
  3. Compare the value of the inserted element with the value of its parent.
  4. If the value of the parent is greater than the value of the inserted element, swap the elements in the array and repeat from step 2.
     - Do not repeat if the element has reached the beginning of the array.

Inserting into an array-based Heap

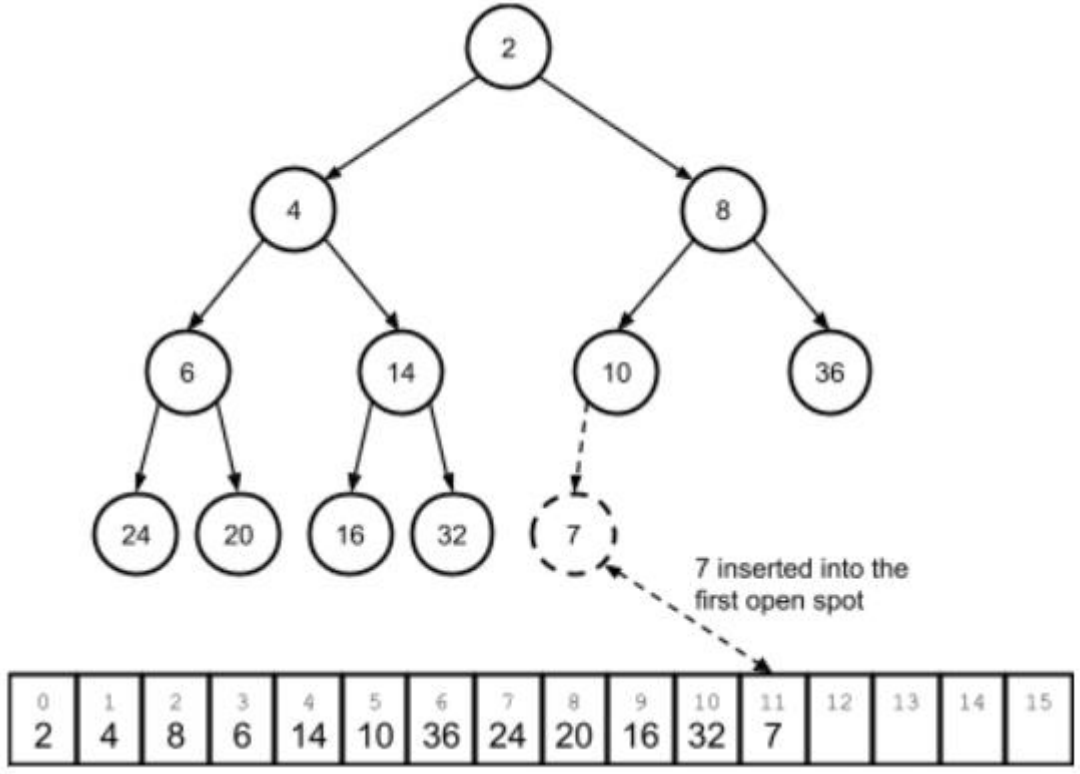
- Example: added 7 to the following heap



Inserting into an array-based Heap

- Example: added 7 to the following heap

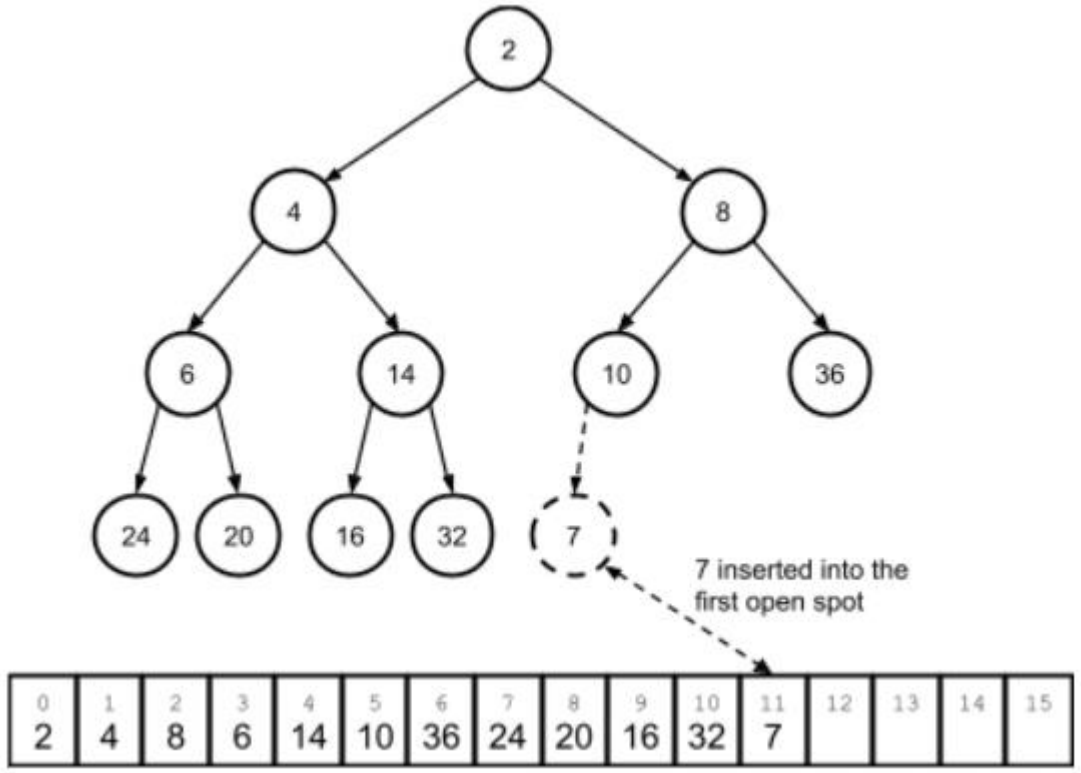
1. insert the new element into the end of the array



Inserting into an array-based Heap

- Example: added 7 to the following heap

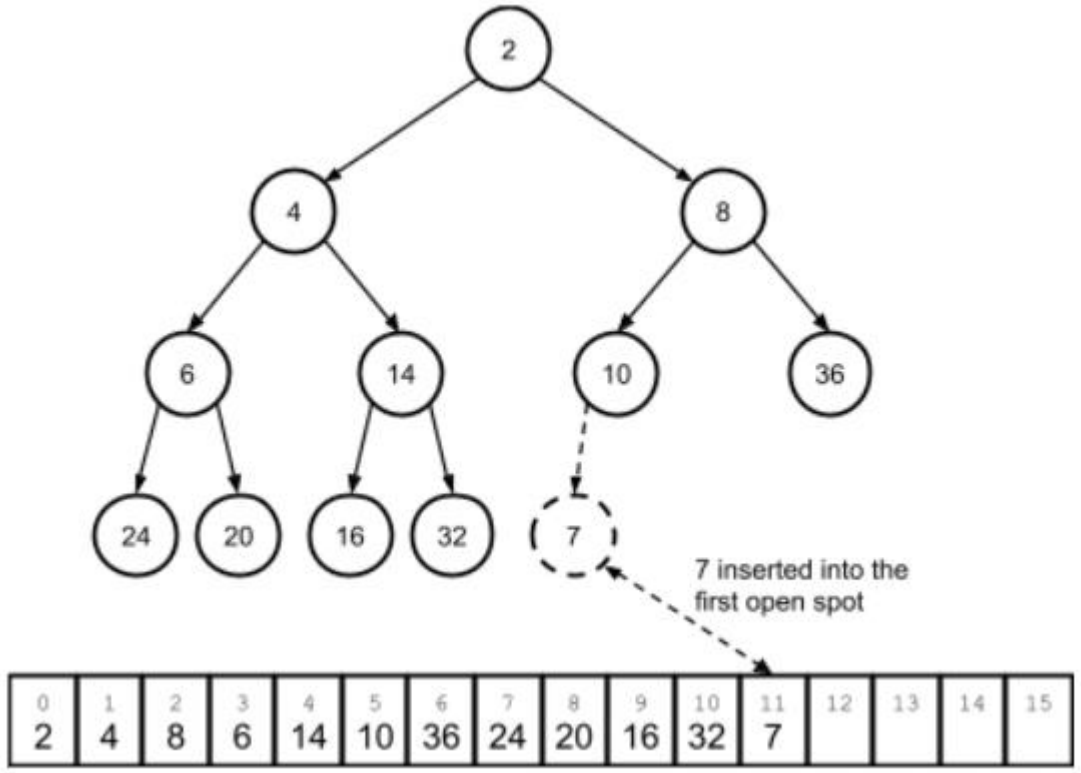
2. compute the index of 7's parent node ((11 - 1) / 2 → 5)



Inserting into an array-based Heap

- Example: added 7 to the following heap

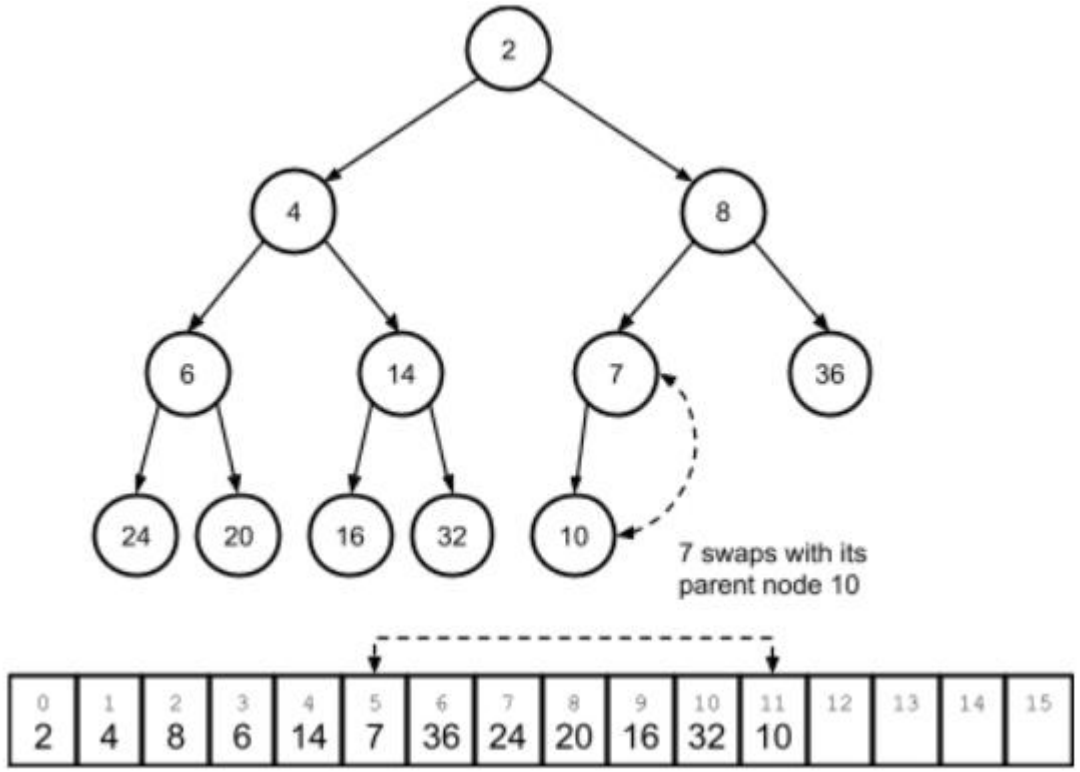
3. compare 7 with the value we found there (at index 5 → 10)



Inserting into an array-based Heap

- Example: added 7 to the following heap

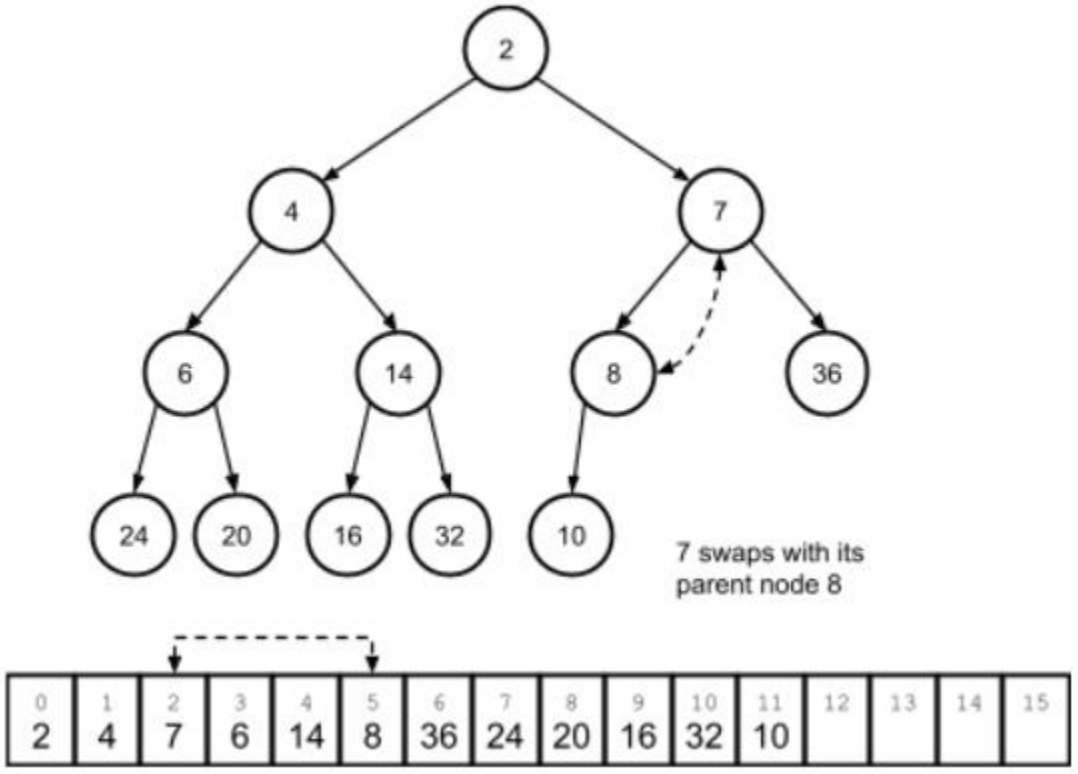
4. Since 7 is less than 10, swap them



Inserting into an array-based Heap

- Example: added 7 to the following heap

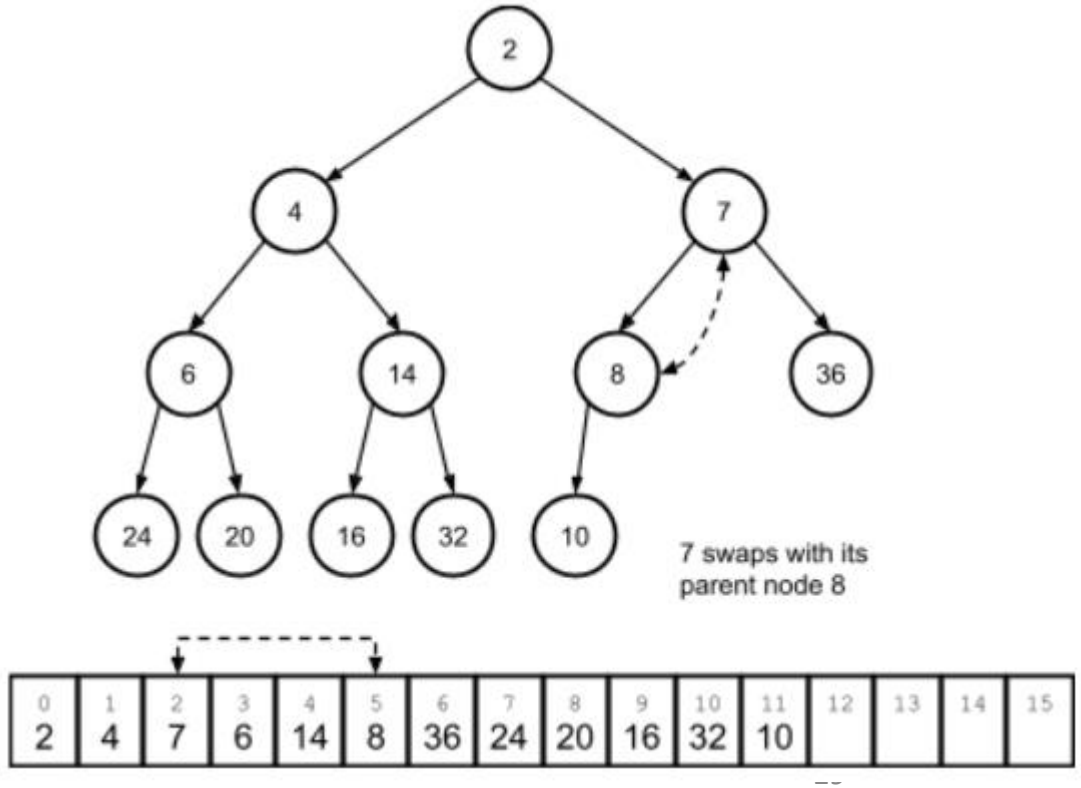
5. Repeat, comparing 7 to its new parent 8 at index (5 - 1) / 2 → 2, and swap again



Inserting into an array-based Heap

- Example: added 7 to the following heap

6. Repeat, compare to 7's new parent node 2 at index (2 - 1) / 2 → 0, and we'd stop, since 2 is less than 7

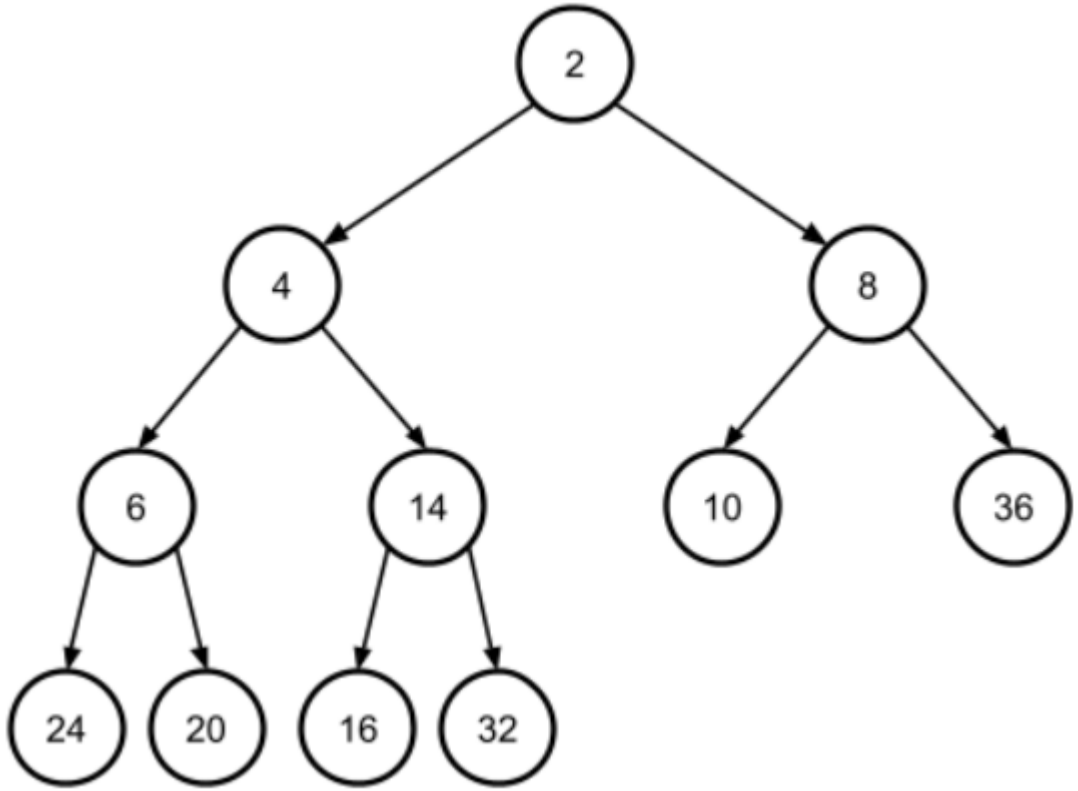


Removing from an array-based Heap

- Recall: in min heap, always remove the node with the lowest priority (i.e., root)
- Remove an element from the array representation of the heap follows this procedure:
  1. Remember the value of the first element in the array (to be returned later).
  2. Replace the value of the first element in the array with the value of the last element and remove the last element.
  3. If the array is not empty (i.e. it started with more than one element), compute the indices of the children of the replacement element (2 * i + 1 and 2 * i + 2).
     - If both of these elements fall beyond the bounds of the array, stop here.
  4. Compare the value of the replacement element with the minimum value of its two children (or possibly one child).
  5. If the replacement element's value is less than its minimum child's value, swap those two elements in the array and repeat from step 3

Removing from an array-based Heap

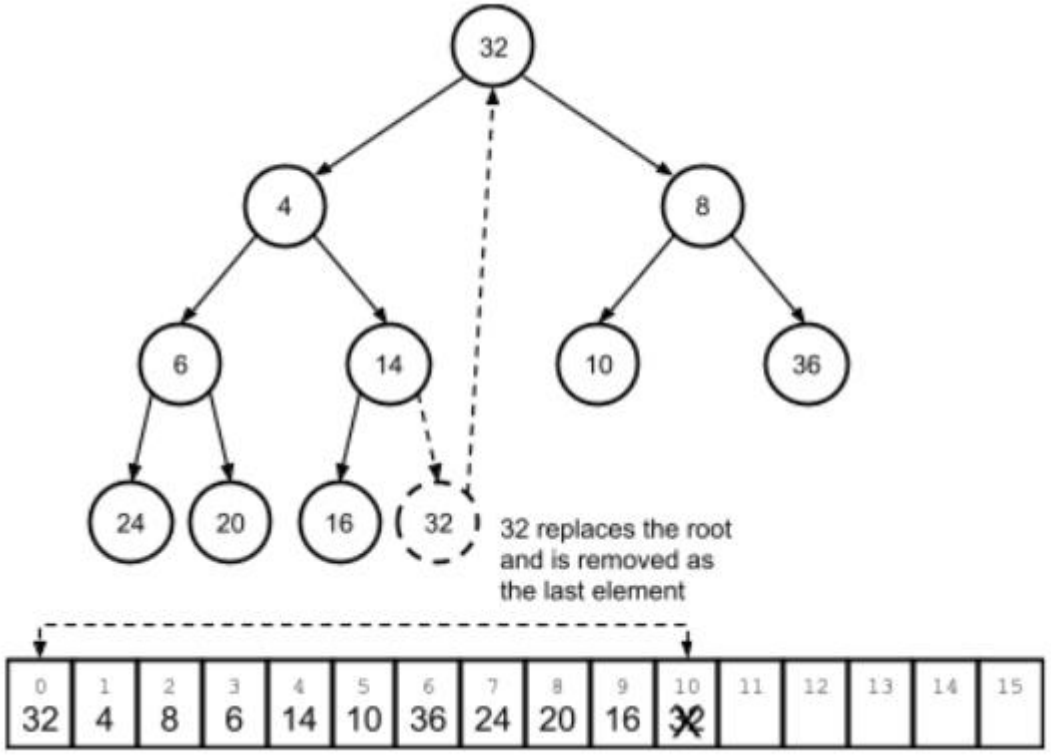
- Example: removing the root (2) from the following heap



Removing from an array-based Heap

- Example: removing the root (2) from the following heap

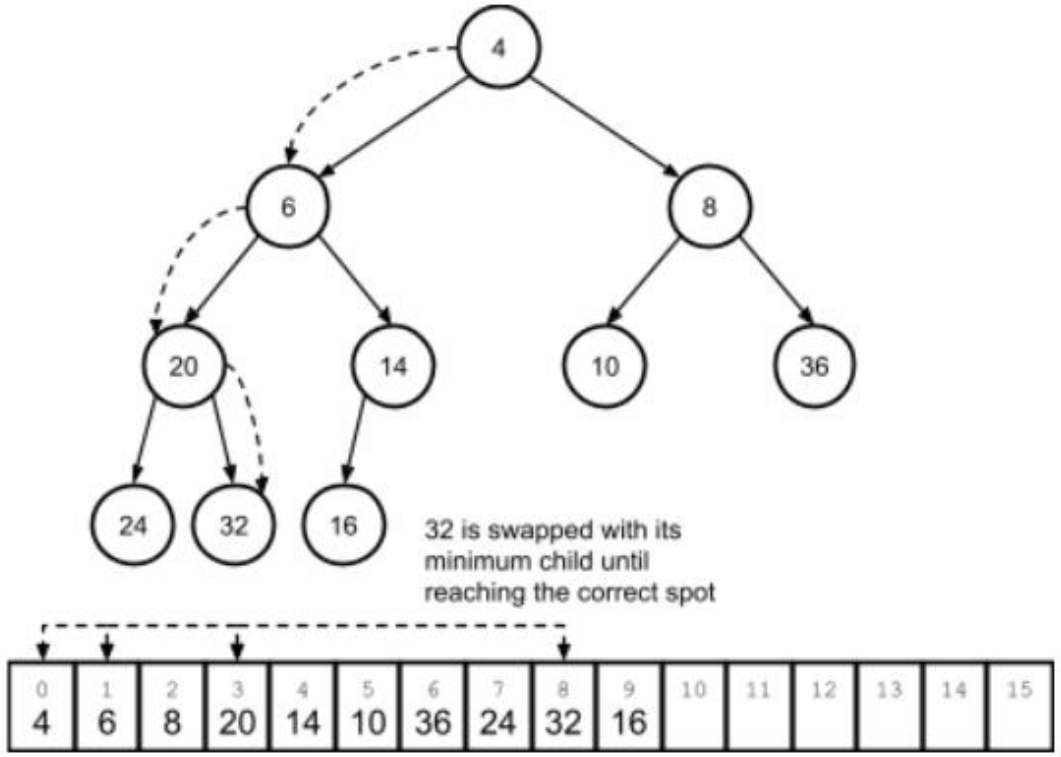
1. replacing the root (the first element in the array) with the last element and then removing the last element



Removing from an array-based Heap

- Example: removing the root (2) from the following heap

2. percolate 32 down the array, comparing it to its minimum-value child and swapping values in the array until 32 reached its correct place



Lecture Topics:

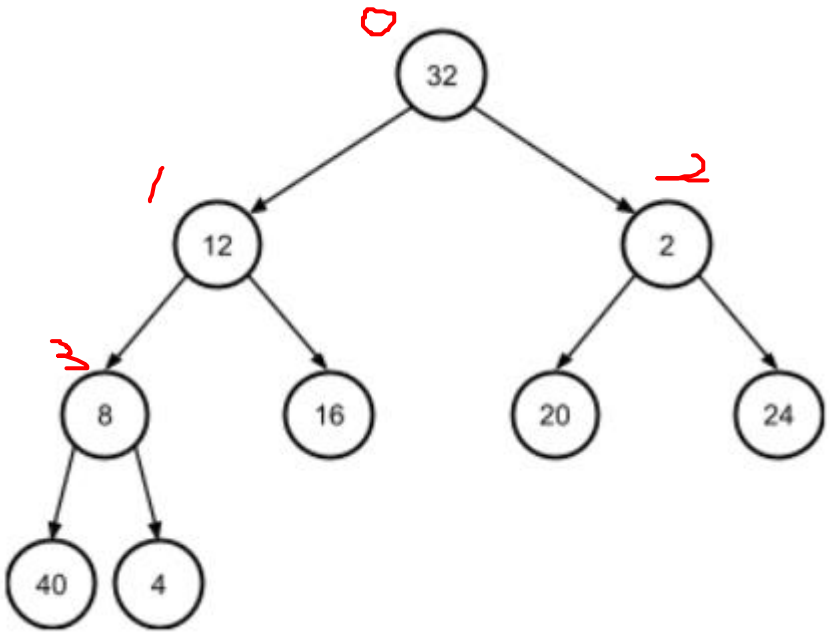
- Priority Queues & Heaps
- Array-based Heaps
- Build a heap from an arbitrary array
- Heapsort

Building a heap from an arbitrary array

- Example: Convert the following arbitrary array to a heap:

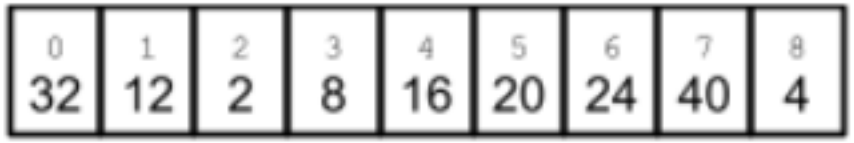
0	1	2	3	4	5	6	7	8
32	12	2	8	16	20	24	40	4

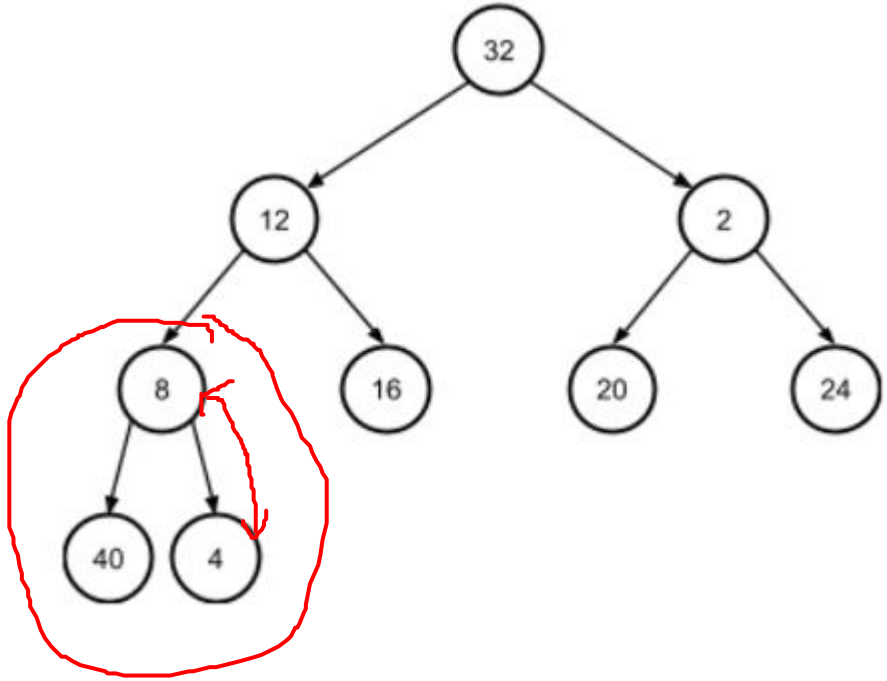
  - First, consider this arbitrary array as a heap:



Building a heap from an arbitrary array

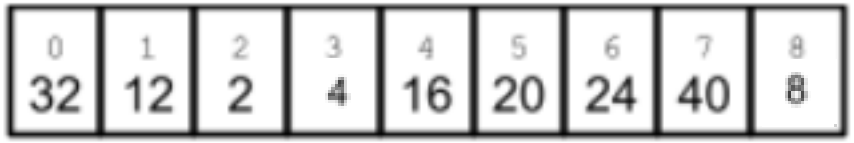
- Percolate down the first non-leaf element, then the subtree rooted at that element's original position will be a proper heap
  - first non-leaf element (from the back of the array) is at n / 2 - 1

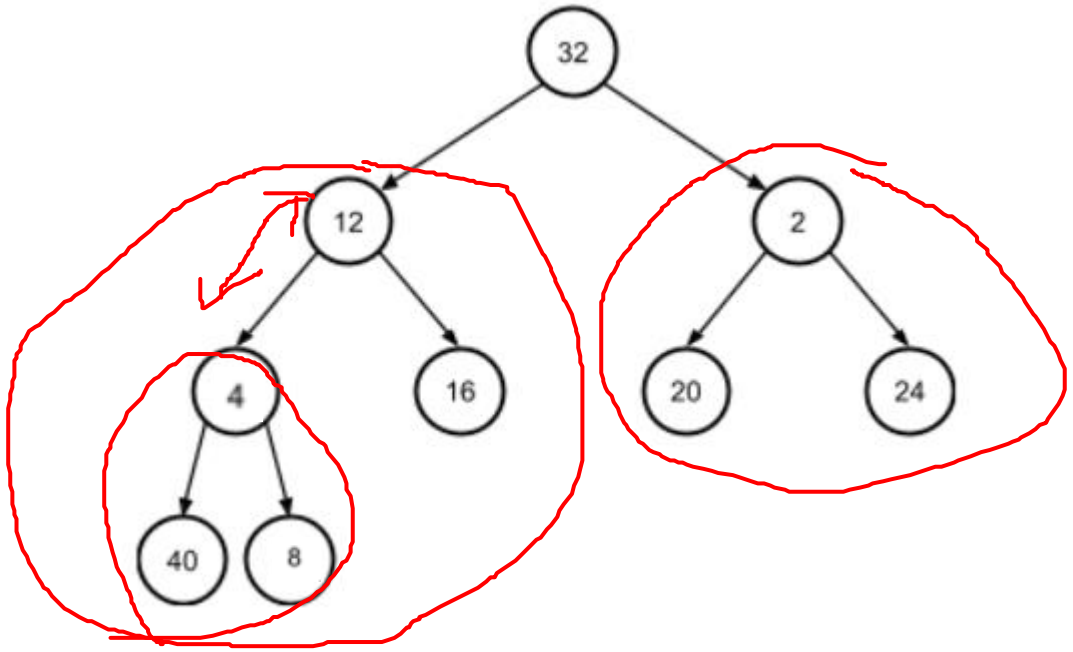




Building a heap from an arbitrary array

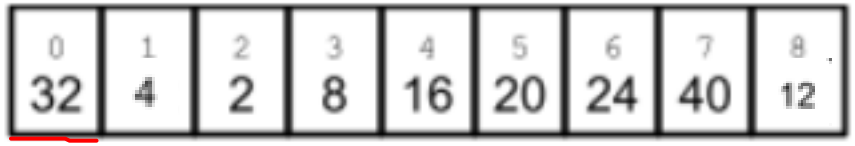
- Percolate down the first non-leaf element, then the subtree rooted at that element's original position will be a proper heap
  - first non-leaf element (from the back of the array) is at n / 2 - 1

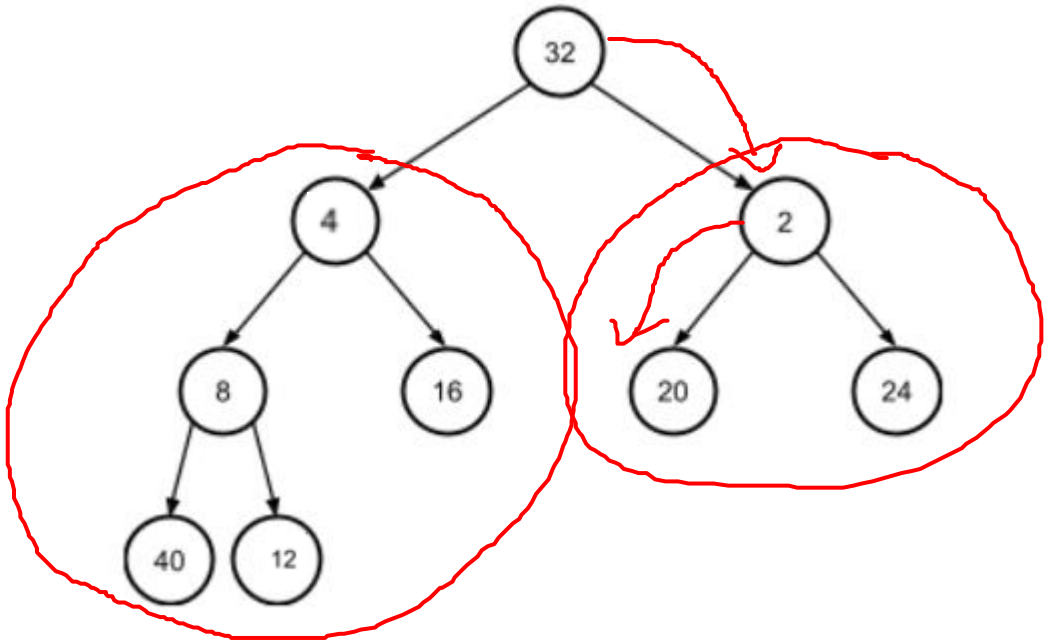




Building a heap from an arbitrary array

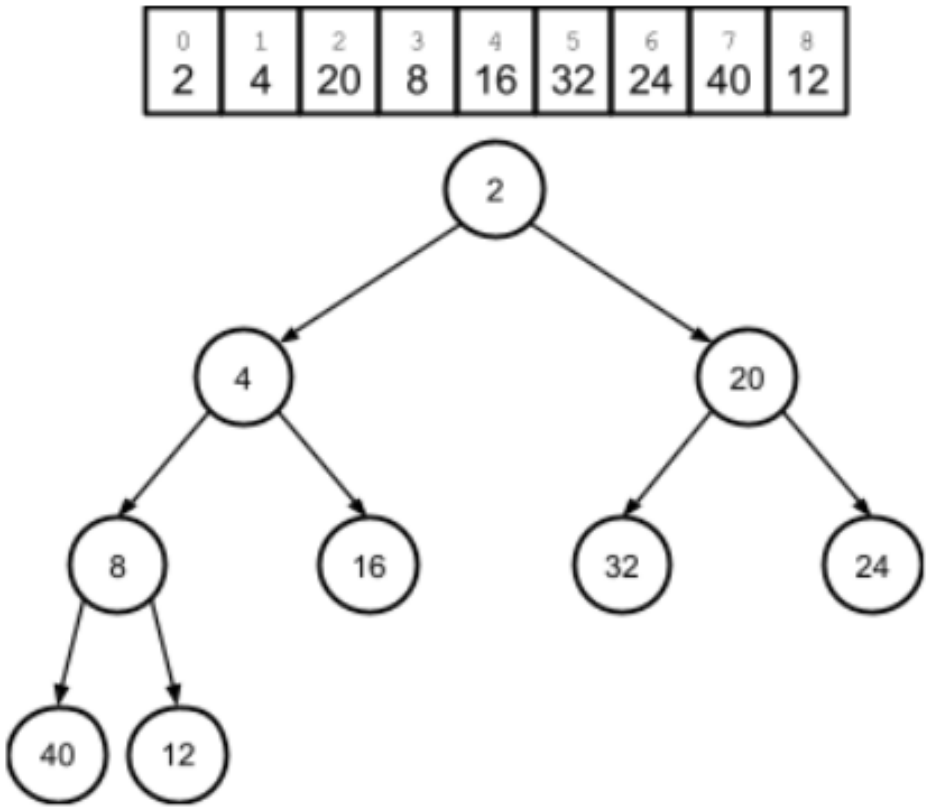
- Percolate down the first non-leaf element, then the subtree rooted at that element's original position will be a proper heap
  - first non-leaf element (from the back of the array) is at n / 2 - 1





Building a heap from an arbitrary array

- Once we percolate down the root element, the entire array will represent a proper heap



Building a heap from an arbitrary array

- Time Complexity:
  - perform n / 2 downward percolation operations.
  - Each of these operations is O(log n).
  - This means the total complexity is O(n log n).

- Space Complexity:
  - No additional space needed and no recursive calls: O(1)

Lecture Topics:

- Priority Queues & Heaps
- Array-based Heaps
- Build a heap from an arbitrary array
- Heapsort

Heap Sort

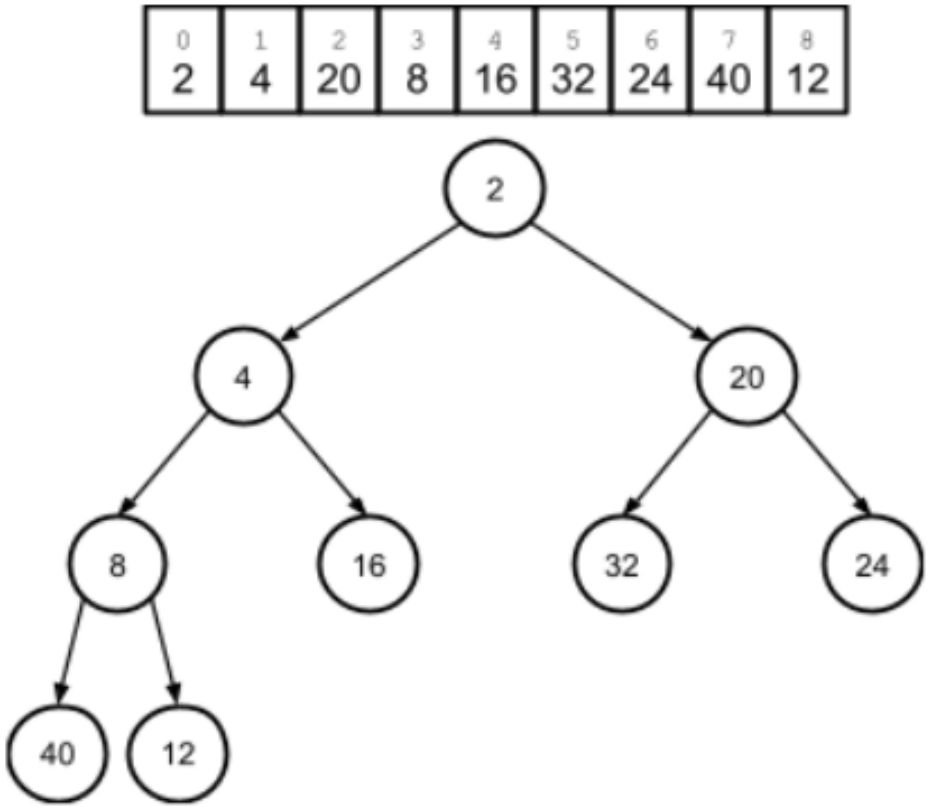
- Given the heap and its operations, we can implement an efficient (O(n log n)), in-place sorting algorithm called heapsort.
- First, build a heap out of the array
- Then, sort:
  - Keep a running counter k that is initialized to one less than the size of the array (i.e. the last element).
  - Swap the first element in the array (the min) with the last element (the kth element).
    - The array itself remains the same size, and we decrement k.
  - Percolate the replacement value down to its correct place in the array, stop at the kth element.
    - Thus, the heap is effectively shrinking by 1 at each iteration
- Repeat this procedure until k reaches the beginning of the array

Heap Sort

- As this sorting procedure runs, it maintains two properties:
  - The elements of the array beyond k are sorted, with the minimum element at the end of the array.
  - The array through element k always forms a heap, with the minimum remaining value at the beginning of the array

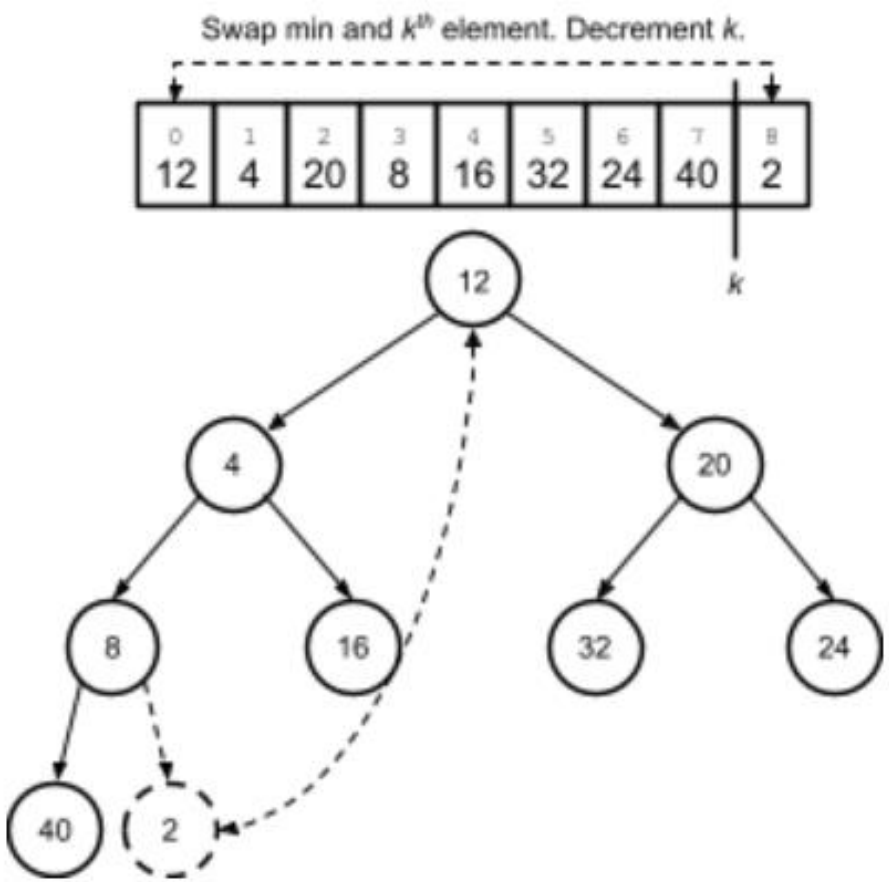
Heap Sort Example

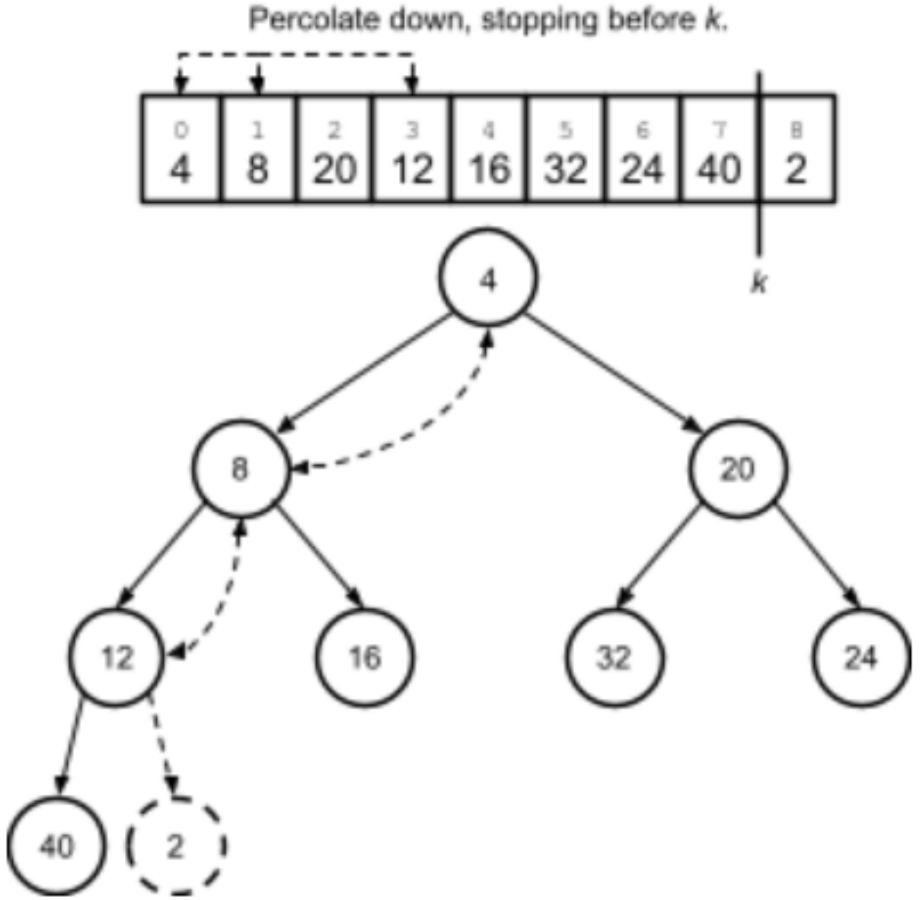
- Apply Heapsort to the following heap array (descending order):



Heap Sort Example

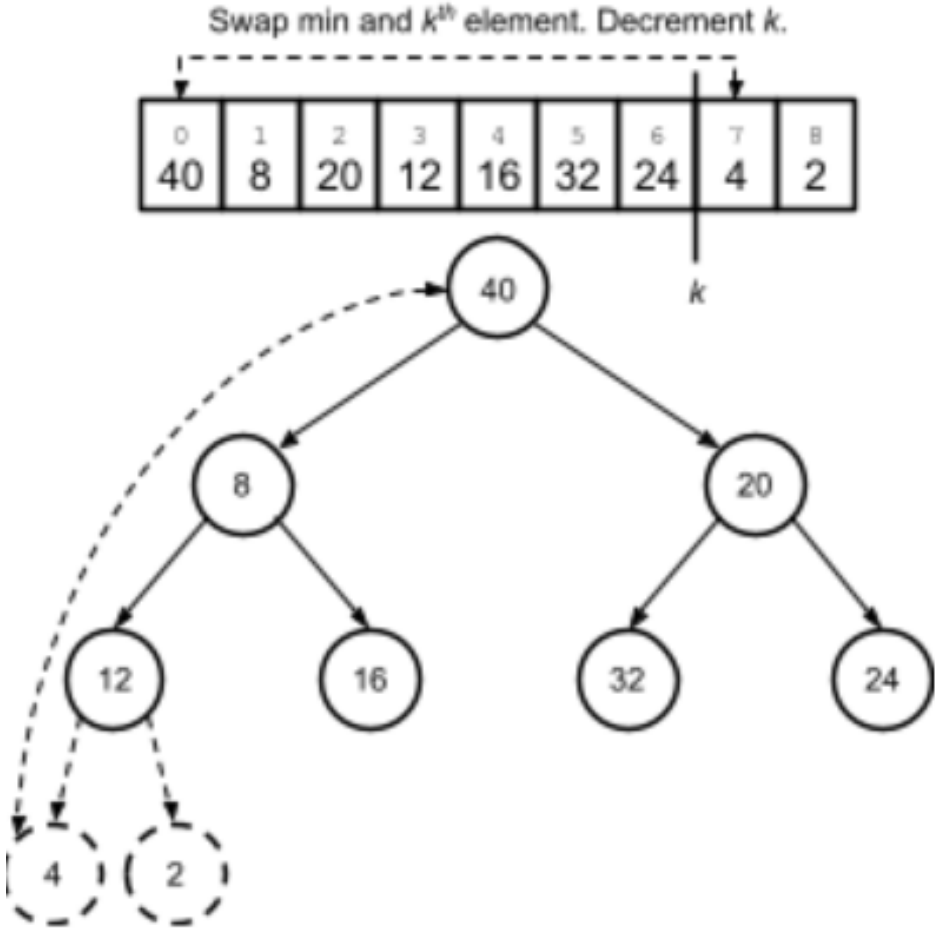
- Apply Heapsort to the following heap array (descending order):

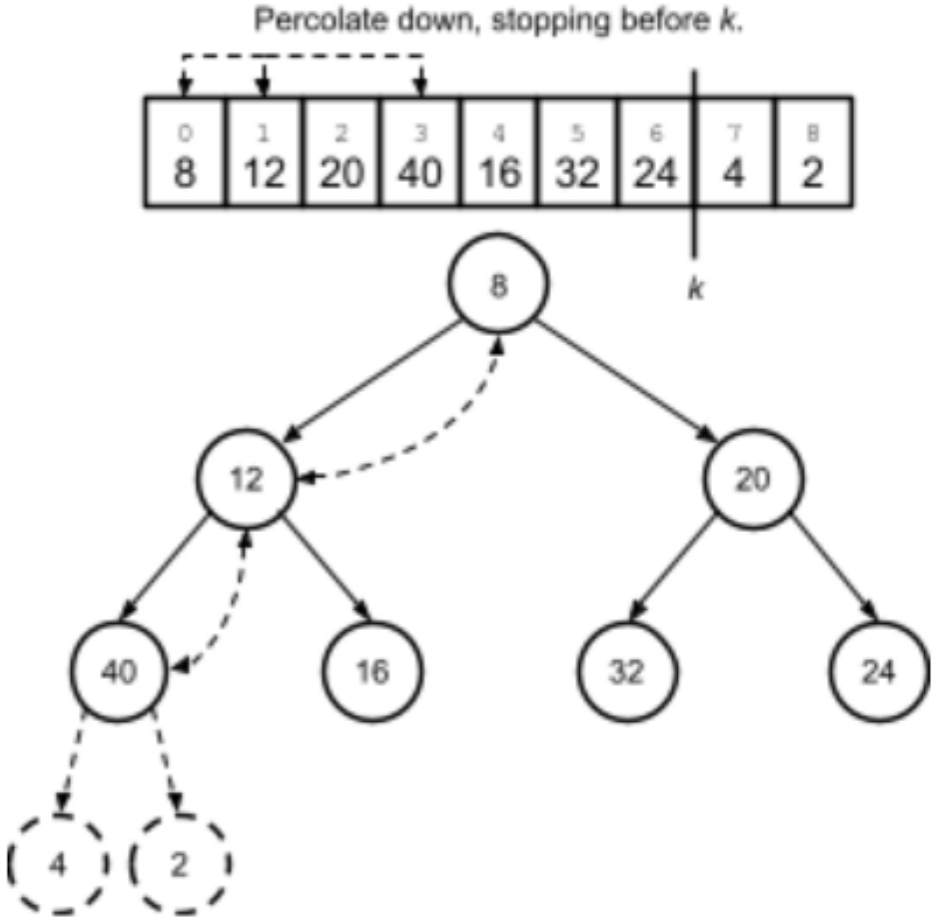




Heap Sort Example

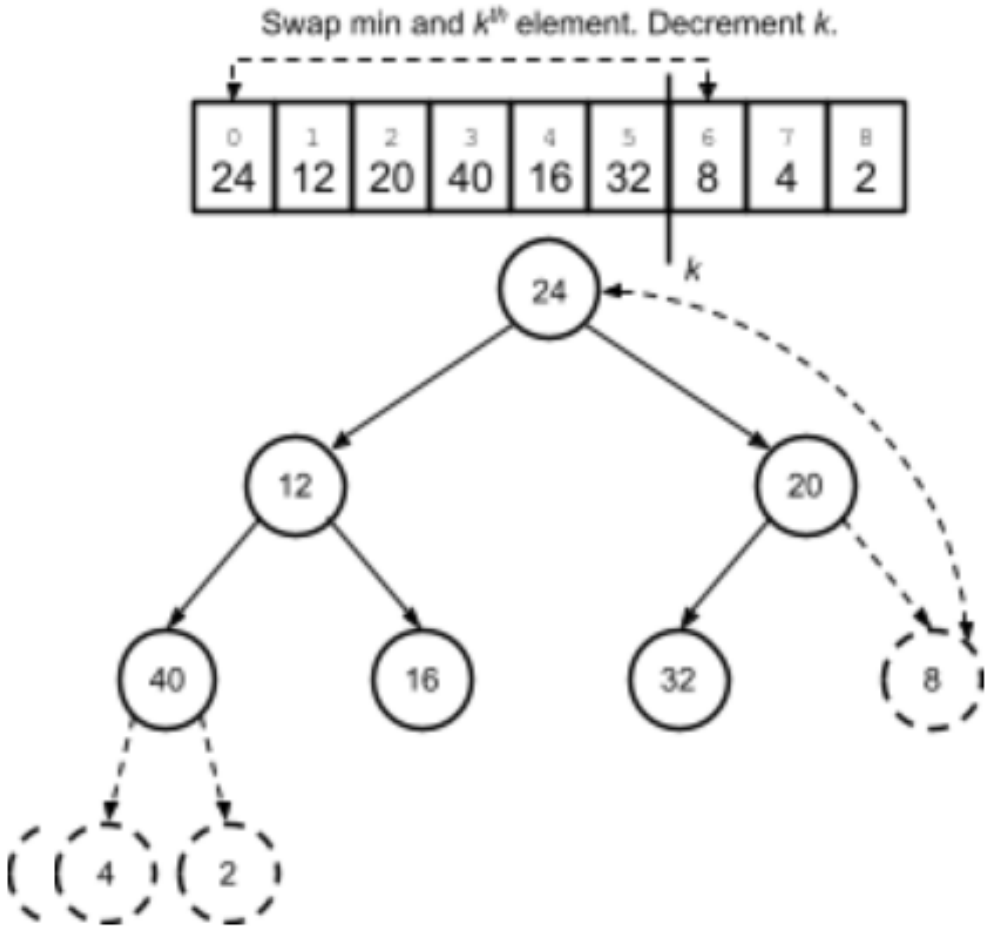
- Apply Heapsort to the following heap array (descending order):

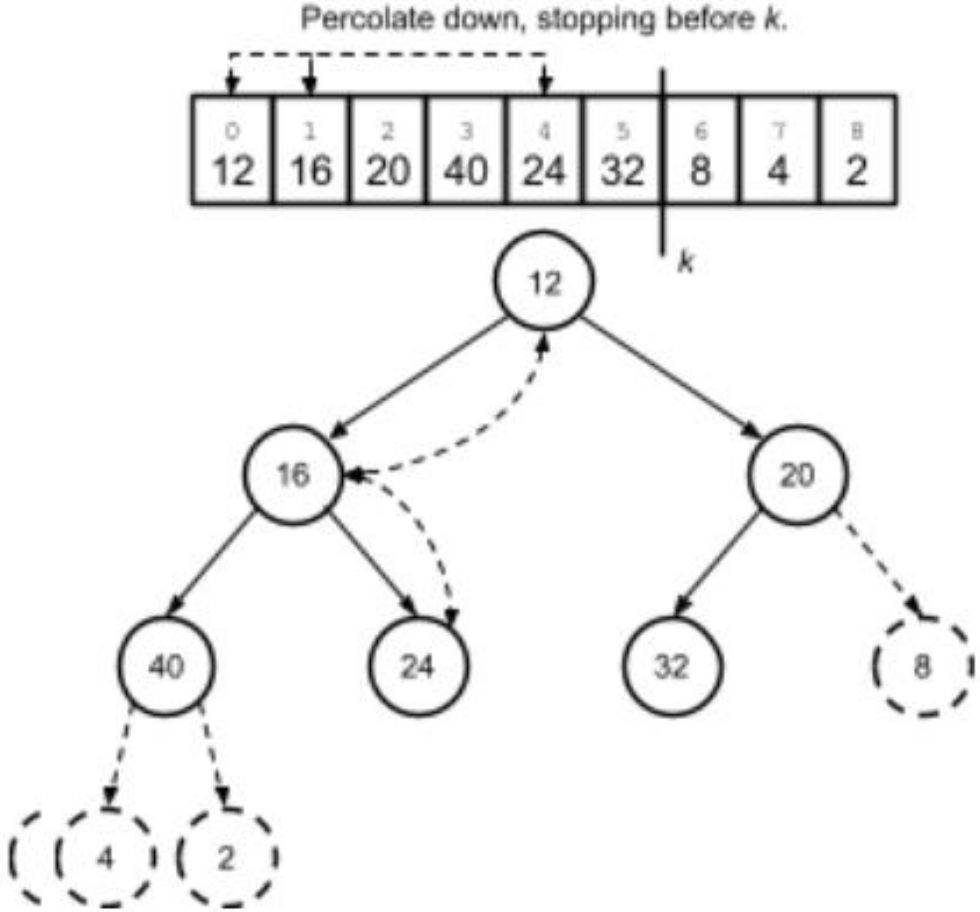




Heap Sort Example

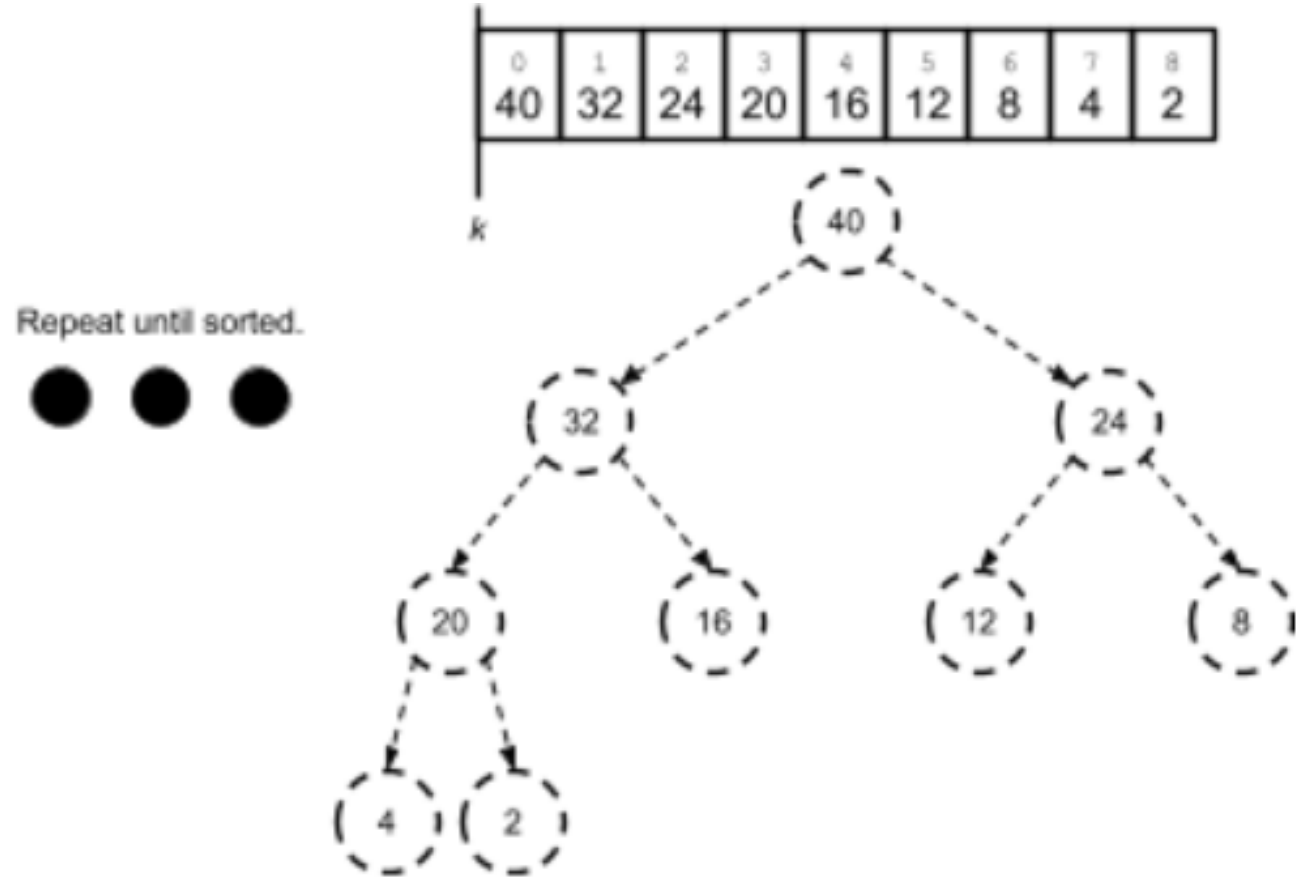
- Apply Heapsort to the following heap array (descending order):





Heap Sort Example

- Apply Heapsort to the following heap array (descending order):



Heap Sort Example

https://youtu.be/MtQL_ll5KhQ