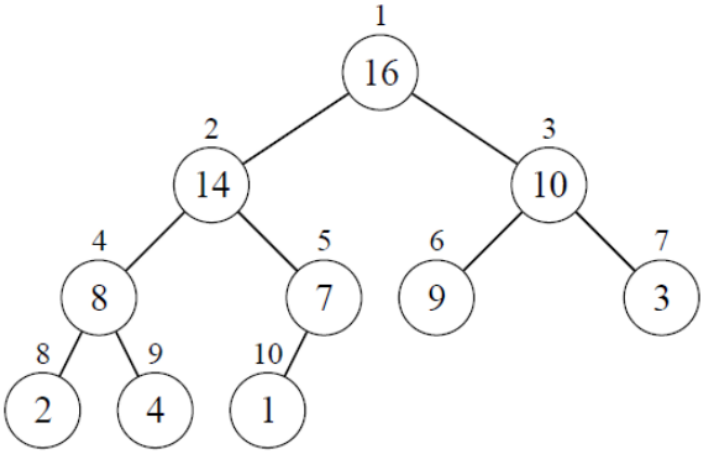
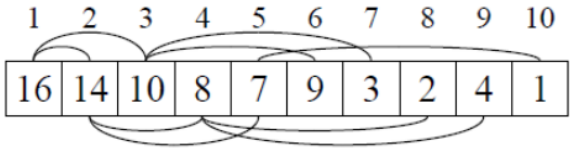
PROBLEM SOLVING 2

More Sorting, Heaps, Heapsort, Priority Queues

Heap stored as array

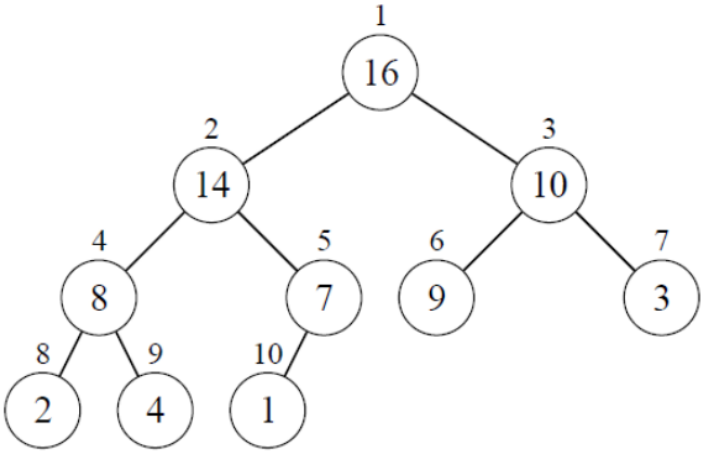
- **Root** = max value = A[1]
- **Parent** of A[i] = A[i/2]
- **Left child** of A[i] = A[2i]; **Right child** of A[i] = A[2i+1]
- Computing is fast with binary representation

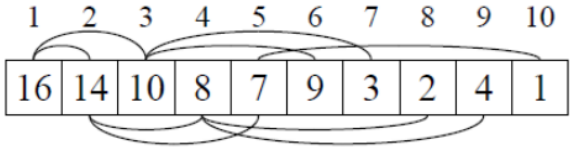




Interview Questions (groups?)

- Minimum and maximum number of elements in a heap of height h?
- Height of an n-element heap
- Where in a max-heap might the smallest element reside, assuming that all elements are distinct?





Is an array in sorted order a min heap, and is a min heap a sorted array?

1. The sorted array is a min heap but the reverse is not true
2. The min heap is a sorted array, but the reverse is not true
3. Both are true
4. Neither are true

Interview Questions

- With the array representation for storing an n-element heap, what is the index of the first leaf?
- Is the array with values {23,17,14,6,13,10,1,5,7,12} a max-heap?

Heap property: parent node >= its children; **parent node** violates heap property

Maintaining the heap property

- Check that A[i] is larger than its children
- If not exchange it with its larger child, and recursively call MAX-HEAPIFY
- lets the value at A[i] “float down” in the max-heap
- Makes node i a max-heap root

```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap-size and A[l] > A[i]
 4      largest = l
 5  else largest = i
 6  if r ≤ A.heap-size and A[r] > A[largest]
 7      largest = r
 8  if largest ≠ i
 9      exchange A[i] with A[largest]
10      MAX-HEAPIFY(A, largest)
```

When does calling max-heapify(A,i) have no effect on the heap?

1. When it is called on A[i] with a value greater than its children
2. When it is called on element A[i] where i > A.heapsize/2
3. All of the above
4. None of the above

Building a heap

- Starting from an unordered heap, build a heap.
- Why does it operate on A.length/2 downto 1?
- Running time seems to be O(n lg n) but it's really **O(n)**
  - Observe that the time for MAX-HEAPIFY to run at a node varies with the height of the node in the tree, and the heights of most nodes are small.

```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

Build a heap!

Given:

{4, 10, 3, 5, 1}

What is the resulting array?

Heapsort

1. Build a max-heap from the array (O(n))
2. Place max element in its correct position by swapping it with the last item (O(1))
3. Decrease heap size by 1
4. Call max-heapify on root O(lg n) each time, n times
5. Calculate time complexity

```
HEAPSORT(A)
1  BUILD-MAX-HEAP(A)
2  for i = A.length downto 2
3      exchange A[1] with A[i]
4      A.heap-size = A.heap-size − 1
5      MAX-HEAPIFY(A, 1)
```

Interview Questions

1. What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order?
2. What about decreasing order?
3. Is one of these a best case?
4. Does this help with finding the median?
5. Does it help us find the largest (like bubblesort did)?
    1. Does it help us find the k largest?

Priority Queue methods

```
HEAP-INCREASE-KEY(A, i, key)
1  if key < A[i]
2       error "new key is smaller than current key"
3  A[i] = key
4  while i > 1 and A[PARENT(i)] < A[i]
5       exchange A[i] with A[PARENT(i)]
6       i = PARENT(i)
```

```
MAX-HEAP-INSERT(A, key)
1  A.heap-size = A.heap-size + 1
2  A[A.heap-size] = −∞
3  HEAP-INCREASE-KEY(A, A.heap-size, key)
```

Time complexities?

Priority Queue

- (in groups)
- Given a sequence of numbers: 19, 6, 8, 11, 4, 5
  - Draw a binary **min**-heap (in a tree form) by inserting the above numbers reading them from left to right.
- Show a tree that can be the result after the call extract() on the above heap
- Show a tree after another call to extract()

Interview Questions

1. Show how to implement a first-in, first-out queue with a priority queue.
2. Show how to implement a stack with a priority queue.

Interview Questions

- Give an O(n lg k)-time algorithm to merge k sorted lists into one sorted list, where n is the total number of elements in all the input lists.
- (Hint: Use a min-heap for k-way merging.)

Consider the extremes:

- What is the result if we merge (n) 1-element lists?
- What is the result if we merge (2) n/2 element lists?