Heap Sort

• Recall Selection Sort:
  While !S.isEmpty() {
    k := S.DeleteMin();
    T.addlast(k); // An easy simplification of Insert(k)
  }

• Let S be a heap and T be the target
  \( O(n \log n) \) since DeleteMin is \( O(\log n) \)
  But how do we build a heap?

Build Heap

```
BuildHeap {
  for i = N/2 to 1 by -1 PercDown(i,A[i])
}
```

Analysis of Build Heap

• Each node can percolate down at most its own height
• Let \( N = 2^{k+1} - 1 \) (height of complete heap is \( k \))
• Then sum of heights is
  \[
  \sum_{j=0}^{k} (k-j) = 2^k - 1 \quad \text{or} \quad N - (k+1)
  \]
Other Heap Operations

- **Find**(X, H): Find the element X in heap H of N elements
  
  > What is the running time? O(N)

- **FindMax**(H): Find the maximum element in H where FindMin is O(1)
  
  > What is the running time? O(N)

- We sacrificed performance of these operations in order to get O(1) performance for FindMin

Other Heap Operations

- **DecreaseKey**(P, ∆, H): Decrease the key value of node at position P by a positive amount ∆, e.g., to increase priority
  
  > First, subtract ∆ from current value at P
  > Heap order property may be violated
  > so percolate up to fix
  > Running Time: O(log N)

Other Heap Operations

- **IncreaseKey**(P, ∆, H): Increase the key value of node at position P by a positive amount ∆, e.g., to decrease priority
  
  > First, add ∆ to current value at P
  > Heap order property may be violated
  > so percolate down to fix
  > Running Time: O(log N)

Other Heap Operations

- **Delete**(P, H): E.g. Delete a job waiting in queue that has been preemptively terminated by user
  
  > Use DecreaseKey(P, ∞, H) followed by DeleteMin
  > Running Time: O(log N)

Other Heap Operations

- **Merge**(H1, H2): Merge two heaps H1 and H2 of size O(N). H1 and H2 are stored in two arrays.
  
  > Can do O(N) Insert operations: O(N log N) time
  > Better: Copy H2 at the end of H1 and use BuildHeap. Running Time: O(N)