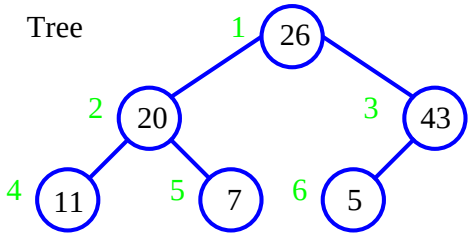
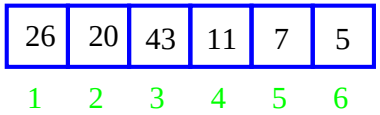
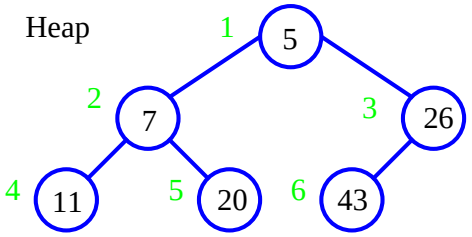
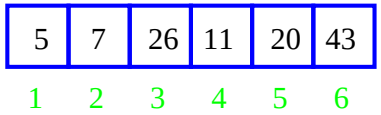
Heap

A complete binary tree with possibly some of the rightmost leaves removed (nearly complete binary tree) and the key at any node is at least as small as its children.









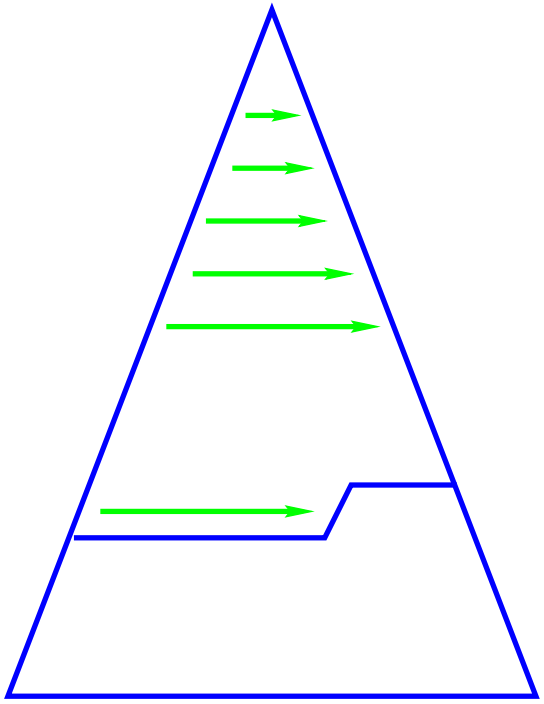
Pointerless Tree: Array$[1, \ldots, n]$

- parent of node i: $\lfloor i/2 \rfloor$
- left child of node i: $2i$
- right child of node i: $2i + 1$

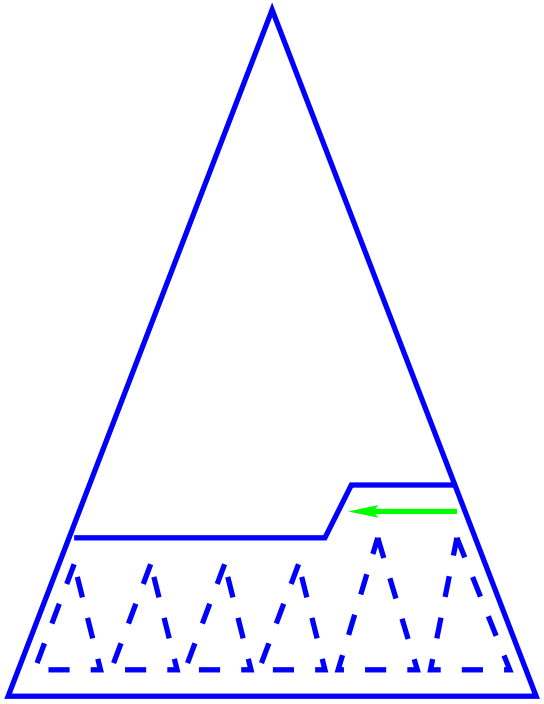
If $2i + 1 > n$, node i has no right child.
If $2i > n$ node i has no children.

"Heapification:"

Top down



Bottom up



Heap Building

Top Down

- Induction base: $A[1]$ is a heap
- Induction hypothesis: the array $A[1, \ldots, i]$ is a heap
- How to add $A[i+1]$ to the heap $A[1, \ldots, i]$?
  Compare to parent $A[\lfloor (i+1)/2 \rfloor]$
  Exchange if parent is bigger
  Until new parent is smaller
- Now $A[1, \ldots, i, i+1]$ is a heap.
  $\lceil \log_2(i) \rceil$ possible comparisons and exchanges.
- Time: $\sum_{i=2}^{n} \log_2 i = O(n \log n)$

Bottom Up

- Induction hypothesis: all trees in the array $A[i+1, \ldots n]$ satisfy the heap condition.
- Induction base: $A[n]$ is a heap! ($A[\lfloor n/2 \rfloor + 1, \ldots, n]$ are leaves in the tree, hence each corresponds to a singleton tree. Heap condition is satisfied.)

Start induction from $\lfloor n/2 \rfloor$.

† Consider $A[i]$: it has at most two children, $A[2i], A[2i+1]$. Both are roots to valid heaps (induction hypothesis!).

† Compare $A[i]$ to the minimal of its children. If $A[i]$ is larger, exchange it with its smaller child.

† Continue this process until $A[i]$ reaches a place where either it is a leaf or it is smaller than both its children

† Now $A[i, \ldots n]$ satisfy heap condition.

Heapify(A, $Size$, i)

– Assume that binary trees rooted at $2i$ and $2i+1$ (children of i) are heaps, but $A[i]$ may be larger than its children.

(violating the heap property)

– Heapify(A, $Size$, i) let the value of $A[i]$ "float down" the heap so that subtree rooted at i becomes a heap.

Heapify(A, $Size$, i)

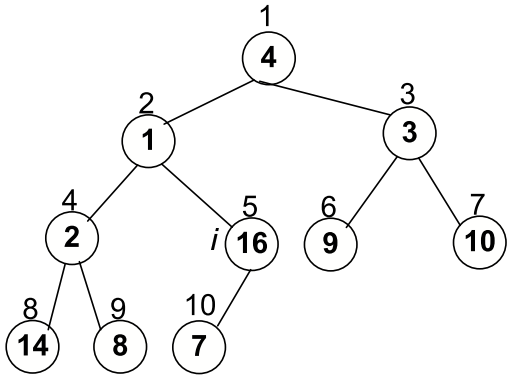
While ($i \leq \lfloor Size/2 \rfloor$)

1. If $A[i] \leq$ min of A[$2i$] and A[$2i+1$] then terminate
   (need to check if $2i+1 \leq size$)
2. Else swap $A[i]$ with the minimum of its children
   Set i to the index of the minimum child

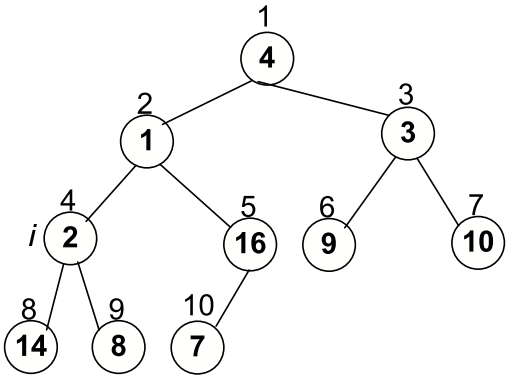
End While

Example: max_heap

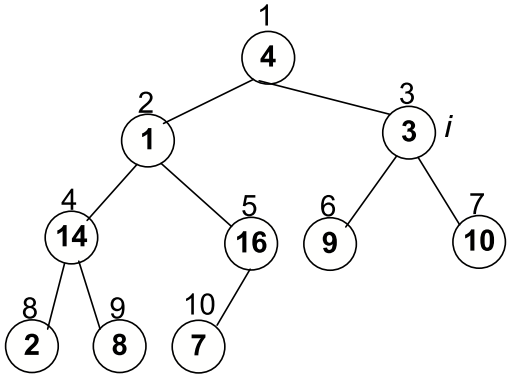
A	4	1	3	2	16	9	10	14	8	7



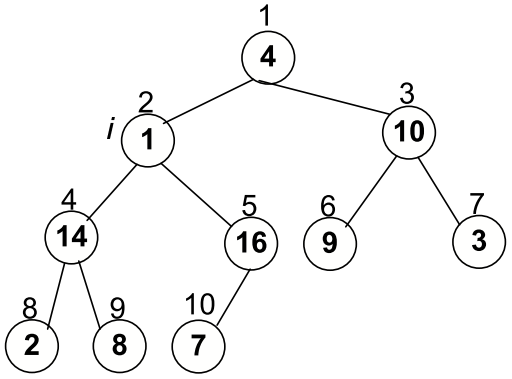
(a)



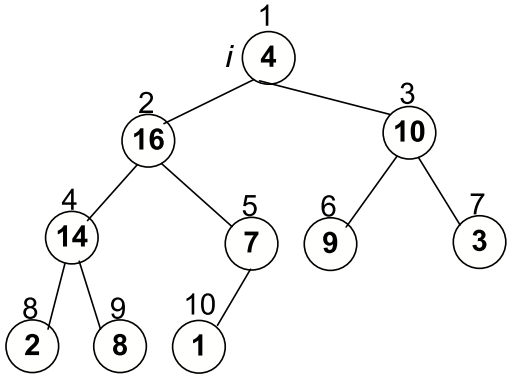
(b)



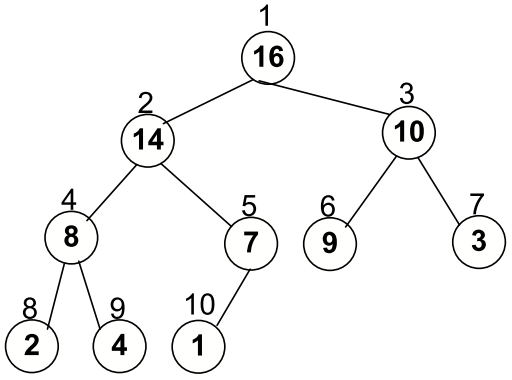
(c)



(d)



(e)



(f)

Time complexity of bottom up

- Each step: (from $A[i+1, \ldots n]$ to $A[i, \ldots n]$)
  Comparisons & exchanges $\leq 2\cdot$ height of the node i in the tree
- Complexity of whole heapification:
  Comparisons & exchanges $\leq \sum_{i=1}^{n} 2\cdot$ height of the node i in the tree.

  Height of a node in a tree: the maximal distance from the node to its leaf descendant.
- If $\lfloor n/2 \rfloor + 1 \leq i \leq n$ height is 0
  If $\lfloor n/4 \rfloor + 1 \leq i \leq \lfloor n/2 \rfloor$ height is 1
  If $\lfloor n/2^{k+1} \rfloor + 1 \leq i \leq \lfloor n/2^{k} \rfloor$ height is k
- $$\sum_{i=0}^{\lfloor \log n \rfloor} i \cdot \frac{n}{2^{i+1}} \leq n(0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{4} + 2 \cdot \frac{1}{8} + 3 \cdot \frac{1}{16} \ldots) \leq cn \qquad c\text{: constant}$$
- Can prove that $\sum_{i=1}^{n}$ height of the node i in the heap $< n$.

Extract_Minimum operation

- Swap $A[1]$ and $A[n]$
- Heapify(A, $n-1$, 1)
- Return $A[n]$

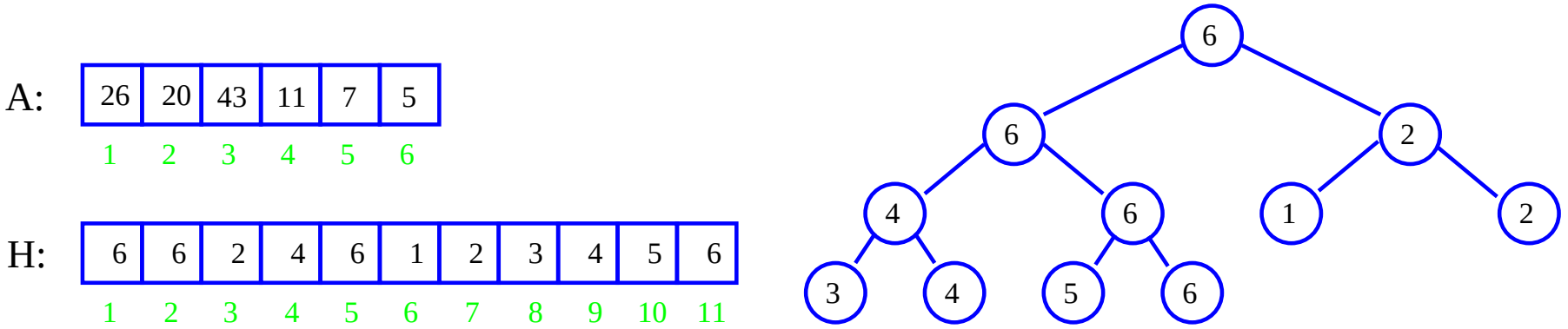
Get_Minimum operation

- Return $A[1]$

A Slightly Different Heap

- After heapification of an array A, the original order of the elements in A is not kept.
- Operations like the following cannot be done efficiently:
  Add 1 to the element which is the i-th element in the original array.
- Solution: another type of the heap structure.
  - † The original array $A[1..n]$ remains the same.
  - † Another array $H[1..2n-1]$ is used to represent a heap of indices.
  - † Each node records an index in the original array A.
  - † The leaves, $H[n..2n-1]$, have index values $1, 2, \ldots, n$.
  - † For an internal node u with two children l and r, if $A[H[l]] < A[H[r]]$ then $H[u] = H[l]$; else $H[u] = H[r]$.
  - † This can be used for the algorithm of single source shortest path problem.

A Slightly Different Heap



This is still called a heap. The following operations are supported: Heapification, Update, Minimum_Index, Get_Key and Extract_Minimum.

- Heapification:
  1. for i from n to $2n-1$ do $H[i] = i - n + 1$.
  2. for i from $n-1$ to 1 do
     if $A[H[2i]] < A[H[2i+1]]$ then
     $H[i] = H[2i]$;
     else $H[i] = H[2i+1]$.

Operations

- Minimum_Index: return $H[1]$.
- Get_Minimum: return $A[H[1]]$.
- Get_Key(i): return $A[i]$.
- $Update(i, v)$: (change $A[i]$ to v)
  1. $A[i] = v$;
  2. $i = \lfloor (i + n - 1)/2 \rfloor$;
  3. while $i \geq 1$ do
     if $A[H[2i]] < A[H[2i + 1]]$ then
     $H[i] = H[2i]$;
     else
     $H[i] = H[2i + 1]$;
     $i = \lfloor i/2 \rfloor$;

- $Extract\_Minimum()$:
  (remove the element with minimum key)
  1. $A[0] = \infty$;
  2. $H[H[1] + n - 1] = 0$;
  3. $v = A[H[1]]$;
  4. $i = \lfloor (H[1] + n - 1)/2 \rfloor$;
  5. while $i \geq 1$ do
     if $A[H[2i]] < A[H[2i + 1]]$ then
     $H[i] = H[2i]$;
     else
     $H[i] = H[2i + 1]$;
     $i = \lfloor i/2 \rfloor$;
  6. return v;

- Time Complexity:
  - † Heapification: $O(n)$.
  - † Update: $O(\log n)$.
  - † Extract_Minimum: $O(\log n)$.
  - † Minimum_Index: $O(1)$.
  - † Get_Minimum: $O(1)$.
  - † Get_Key: $O(1)$.

An Example

n people are playing a game. Array $A[1..n]$ has the score of n people. Each round there is only one random person who wins the game, and the score of that person is increased by one. Print out in each round, the person who has the lowest score.

Naive Algorithm For each time, update the score of the person who wins. Search through A to find the loser. Print it.
Time Complexity: $O(n^2)$.

Use the new heap structure. Construct a heap H. For each time, update the score of the person who wins, and print the MinimumIndex of H.
Time Complexity: $O(n \log n)$.

Why cannot we use the ordinary heap here?