Heaps, Binary Heaps and Heapsort

Subhabrata Samajder

IIIT, Delhi

Summer Semester,

20^{th} June, 2022

Heapsort

Heapsort

- Like Merge-sort it's worst case time complexity is $\mathcal{O}(n \log n)$.
- Like Quick-sort it is an in place algorithm.
    - # of elements stored outside the input array at any time: $\mathcal{O}(1)$.
- Combines the better attributes of the two sorting algorithms.

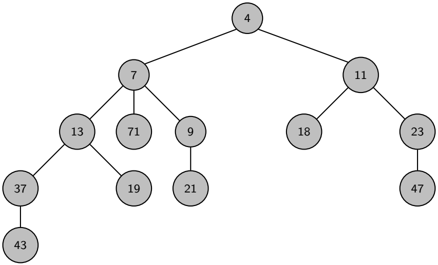
Heapsort (Cont.)

- Introduces a different algorithm design technique:
  - the use of a data structure to manage information during the execution of the algorithm.
  - This data structure is called a heap.
- Apart from heapsort it also makes an efficient priority queue.
- The term heap was originally coined in the context of heapsort.
- But it has since come to refer to as garbage-collection storage provided by programming languages like Lisp and Java.
- But heap data structure is not garbage-collected storage!

Heaps

Heap

A **Min-Heap** is a (rooted) tree data structure where the value stored in a node less than or equal to the value stored in each of its children.



Heap (Cont.)

- The lowest/highest priority element is always stored at the root.
- It is not a sorted structure.
- It can be regarded as being partially ordered.
- It is useful when it is necessary to repeatedly remove the object with the lowest/highest priority.

Basic Operations

Query Operations:

- FIND-MIN(H): Report the smallest key stored in the heap.

Modifying Operations:

- CREATEHEAP(H): Create an empty heap H.
- INSERT(x, H): Insert a new key with value x into the heap H.
- EXTRACT-MIN(H): Delete the smallest key from H.
- DECREASE-KEY(p, Δ, H): Decrease the value of the key p by amount Δ.
- MERGE(H_1, H_2): Merge two heaps H_1 and H_2.

Variants

- 2-3 heap
- B-heap
- Beap
- Binary heap
- Binomial heap
- Brodal queue
- *d*-ary heap
- Fibonacci heap
- K-D Heap
- Leaf heap
- Leftist heap
- Pairing heap
- Radix heap
- Randomized meldable heap
- Skew heap
- Soft heap
- Ternary heap
- Treap
- Weak heap

Variants

- 2-3 heap
- B-heap
- Beap
- Binary heap
- Binomial heap
- Brodal queue
- *d*-ary heap
- Fibonacci heap
- K-D Heap
- Leaf heap
- Leftist heap
- Pairing heap
- Radix heap
- Randomized meldable heap
- Skew heap
- Soft heap
- Ternary heap
- Treap
- Weak heap

Complete Binary Tree

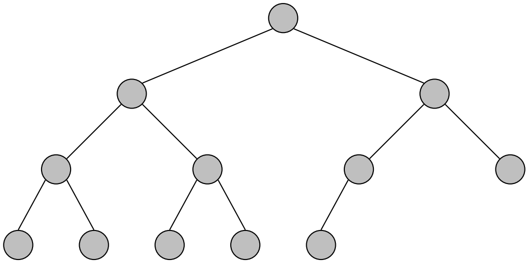
Can we implement a binary tree using an array?

Complete Binary Tree

Can we implement a binary tree using an array?

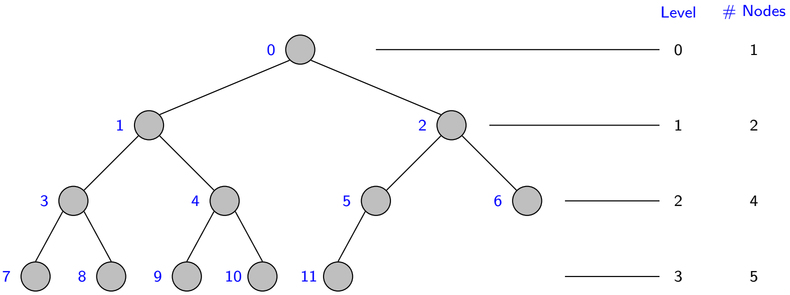
Yes, in some special cases.

A Complete Binary Tree



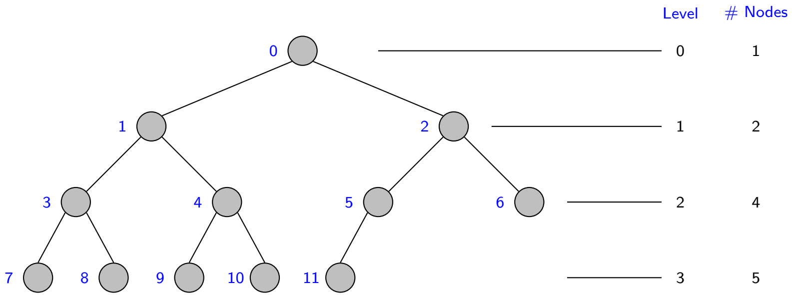
A complete binary of 12 nodes.

A Complete Binary Tree



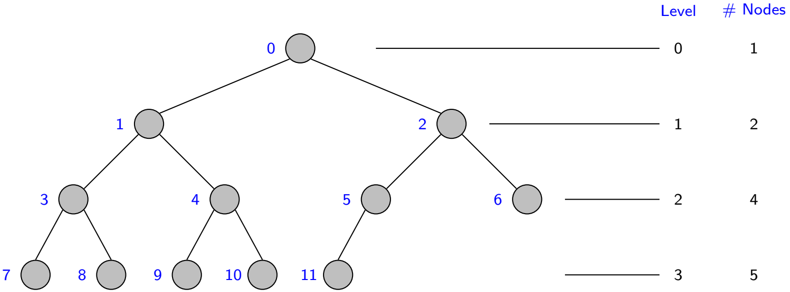
Can you see a relationship between label of a node and labels of its children?

A Complete Binary Tree



- The label of the **leftmost node** at level $i = 2^i - 1$.
- The label of a **node** v **at level** i **occurring at** k^{th} **place from left** $= 2^i + k - 2$.
- The label of the **left** child of v is $= 2 \cdot (2^i + (k-2)) + 1$.
- The label of the **right** child of v is $= 2 \cdot (2^i + (k-2)) + 2$.

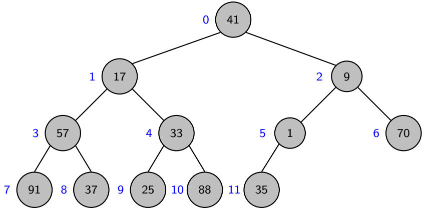
A Complete Binary Tree



- Let v be a node with label j.
- Label of **left child**$(v) = 2j + 1$.
- Label of **right child**$(v) = 2j + 2$.
- Label of **parent**$(v) = \lfloor (j-1)/2 \rfloor$.

A Complete Binary Tree and An Array

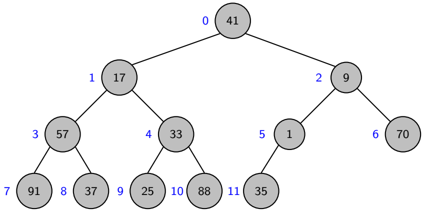
Can we implement a complete binary tree using an array?

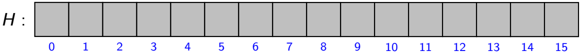


A Complete Binary Tree and An Array

Can we implement a complete binary tree using an array? Yes!

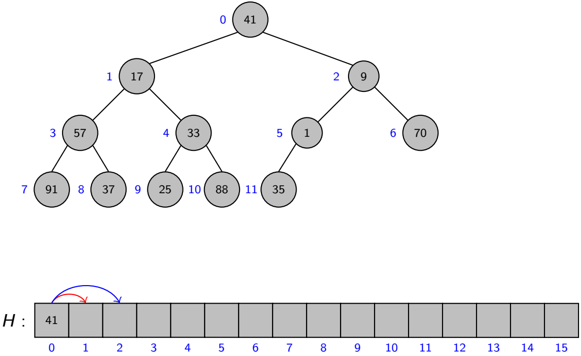
Advantage: It is the most compact representation.





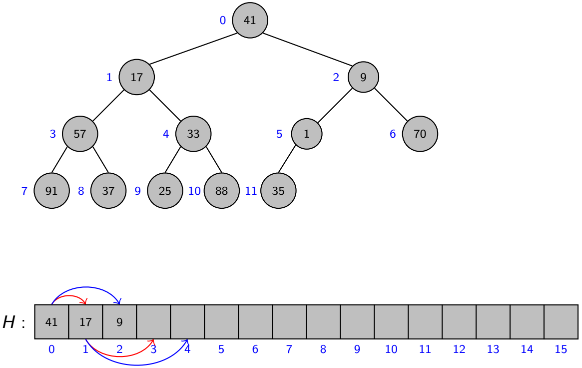
A Complete Binary Tree and An Array

Can we implement a complete binary tree using an array?



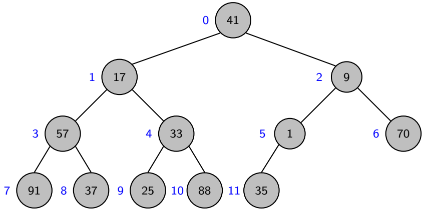
A Complete Binary Tree and An Array

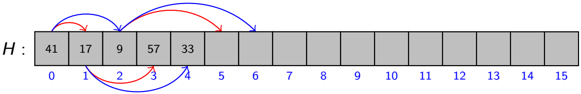
Can we implement a complete binary tree using an array?



A Complete Binary Tree and An Array

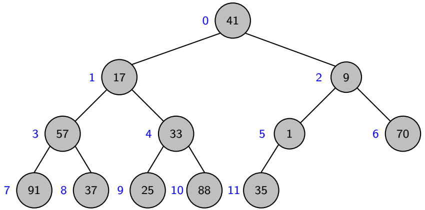
Can we implement a complete binary tree using an array?

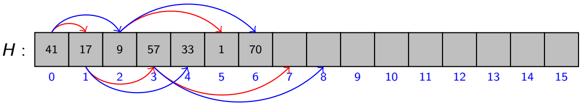




A Complete Binary Tree and An Array

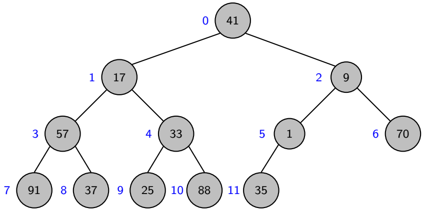
Can we implement a complete binary tree using an array?

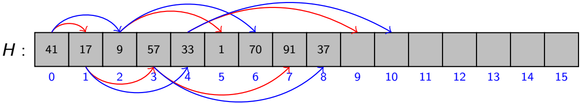




A Complete Binary Tree and An Array

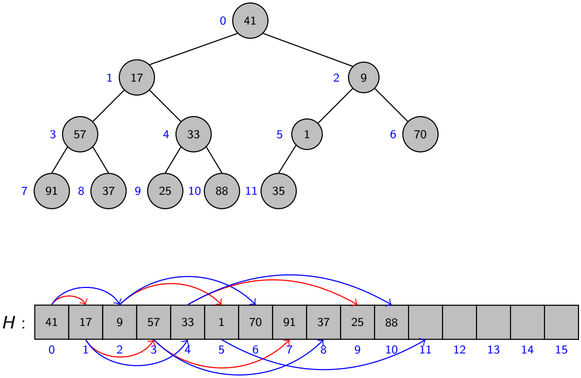
Can we implement a complete binary tree using an array?





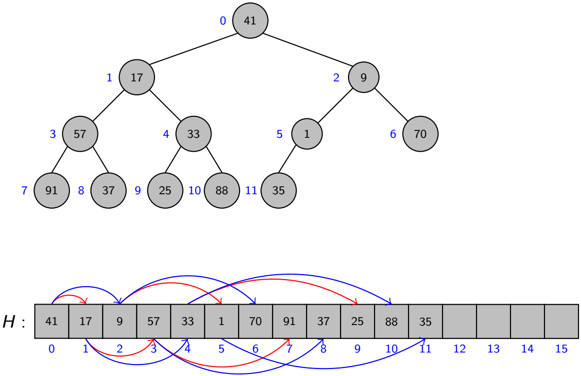
A Complete Binary Tree and An Array

Can we implement a complete binary tree using an array?



A Complete Binary Tree and An Array

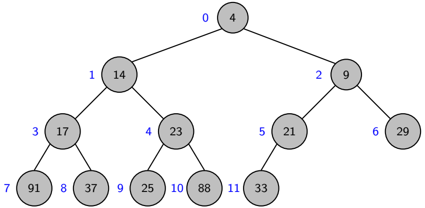
Can we implement a complete binary tree using an array?



Binary Heap

Binary (Min) Heap

Definition: It is a complete binary tree satisfying the heap property at each node.



H :

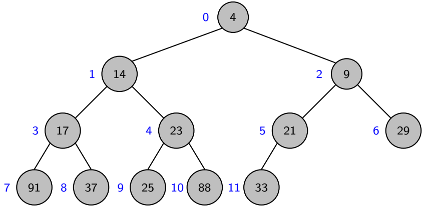
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	14	9	17	23	21	29	91	37	25	88	33				

Implementation of a Binary Heap

H[ ] : An array of size n used for storing the binary heap.

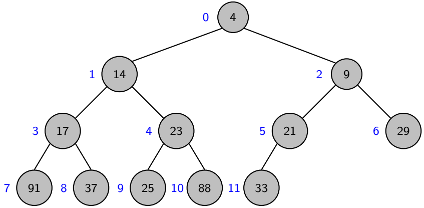
size : A variable for the total number of keys currently in the heap.

Find-Min(H)



H :	4	14	9	17	23	21	29	91	37	25	88	33				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Find-Min(H)



H :

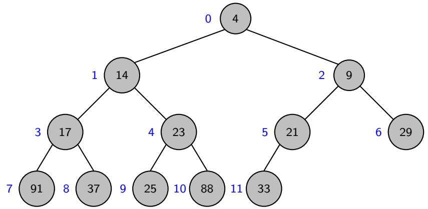
4	14	9	17	23	21	29	91	37	25	88	33				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Return $H[0]$

EXTRACT-MIN(H)

Goal: Deletes the smallest key from H.

Challenge: Preserve the complete binary tree structure as well as the heap property!

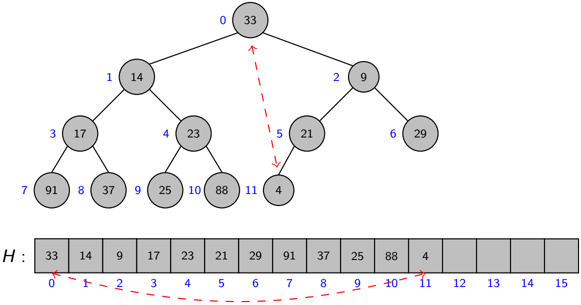


H :	4	14	9	17	23	21	29	91	37	25	88	33				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

EXTRACT-MIN(H)

Goal: Deletes the smallest key from H.

Challenge: Preserve the complete binary tree structure as well as the heap property!

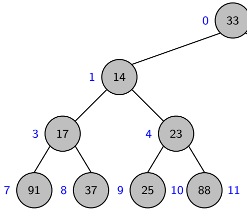


- $swap(H[0], H[size - 1])$.

EXTRACT-MIN(H)

Goal: Deletes the smallest key from H.

Challenge: Preserve the complete binary tree structure as well as the heap property!



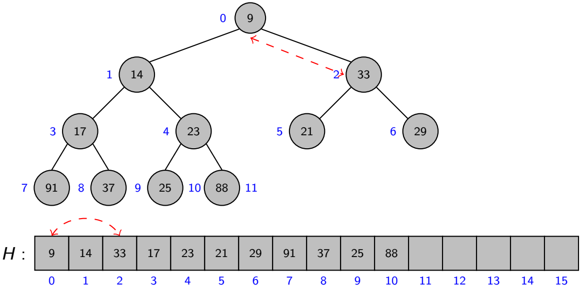
H :	33	14	9	17	23	21	29	91	37	25	88					
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- $swap(H[0], H[size-1])$.
- $size = size - 1$.

EXTRACT-MIN(H)

Goal: Deletes the smallest key from H.

Challenge: Preserve the complete binary tree structure as well as the heap property!

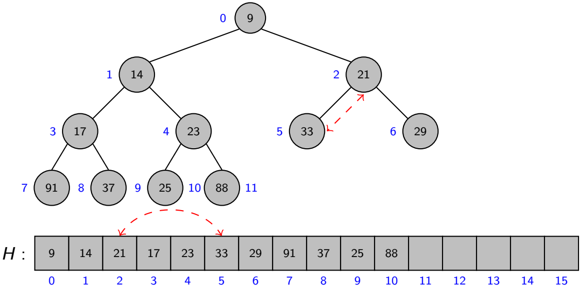


- $swap(H[0], H[size-1])$.
- $size = size - 1$.
- **While** $x > key[left[x]]$ or $x > key[right[x]]$, **then**
  - $swap(x, \min\{left[x], right[x]\})$.

EXTRACT-MIN(H)

Goal: Deletes the smallest key from H.

Challenge: Preserve the complete binary tree structure as well as the heap property!

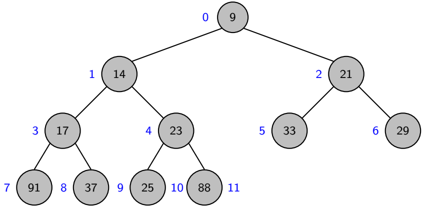


- $swap(H[0], H[size-1])$.
- $size = size - 1$.
- **While** $x > key[left[x]]$ or $x > key[right[x]]$, **then**
  - $swap(x, \min\{left[x], right[x]\})$.

EXTRACT-MIN(H)

Goal: Deletes the smallest key from H.

Challenge: Preserve the complete binary tree structure as well as the heap property!



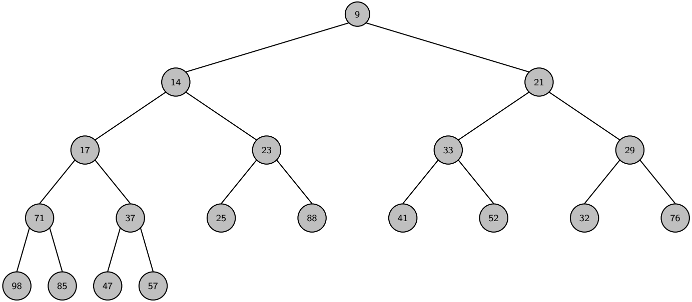
H :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
9	14	21	17	23	33	29	91	37	25	88					

- $swap(H[0], H[size - 1])$.
- $size = size - 1$.
- **While** $x > key[left[x]]$ or $x > key[right[x]]$, **then**
  - $swap(x, \min\{left[x], right[x]\})$.
- **Complexity:** $\#$ swaps $= \mathcal{O}(\#$ levels in binary heap$) = \mathcal{O}(\log n)$ (show it!).

INSERT(x, H)

Let $x = 11$.

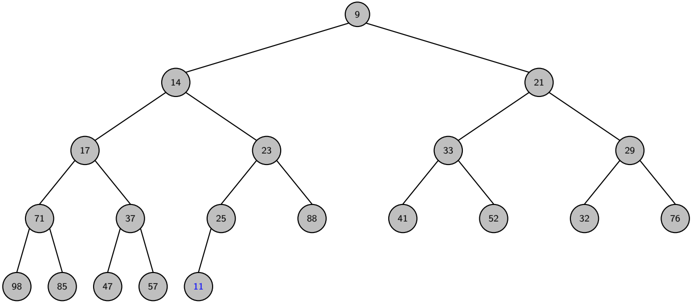


H :

9	14	21	17	23	33	29	71	37	25	88	41	52	32	76	98	85	47	57			...	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31

INSERT(x, H)

Let $x = 11$.



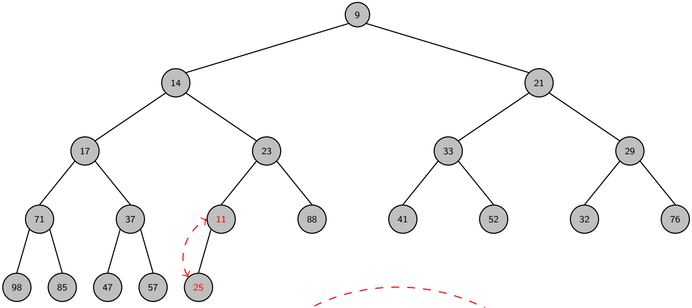
H :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31
9	14	21	17	23	33	29	71	37	25	88	41	52	32	76	98	85	47	57	11		...	

- $H[size] = x$.
- $size = size + 1$

INSERT(x, H)

Let $x = 11$.

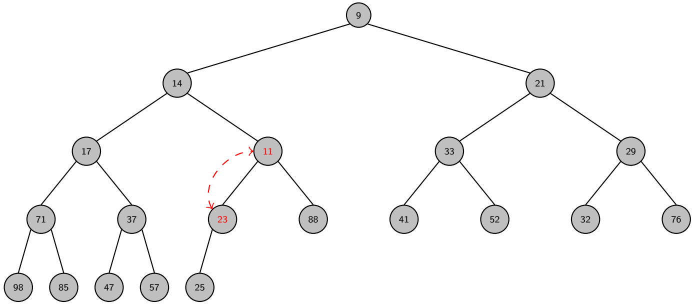


H :	9	14	21	17	23	33	29	71	37	11	88	41	52	32	76	98	85	47	57	25		...	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31

- $H[size] = x$.
- $size = size + 1$
- **While** $parent[x] > x$, **then**
  - $swap(x, parent[x])$.

INSERT(x, H)

Let $x = 11$.

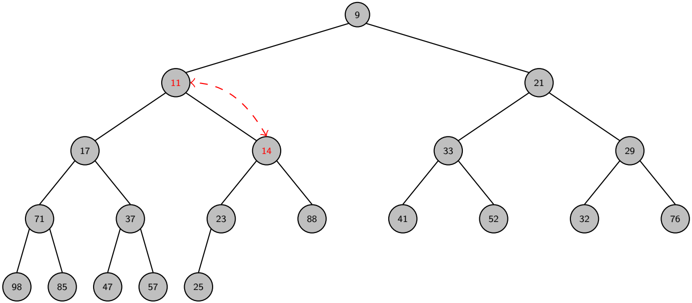


H :	9	14	21	17	11	33	29	71	37	23	88	41	52	32	76	98	85	47	57	25		...	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31

- $H[size] = x$.
- $size = size + 1$
- **While** $parent[x] > x$, **then**
  - $swap(x, parent[x])$.

INSERT(x, H)

Let $x = 11$.



H :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31
9	11	21	17	14	33	29	71	37	23	88	41	52	32	76	98	85	47	57	25		···	

- $H[size] = x$.
- $size = size + 1$
- **While** $parent[x] > x$, **then**
  - $swap(x, parent[x])$.

INSERT(x, H) (Cont.)

Begin
$i \leftarrow size(H)$;
$H[size] \leftarrow x$;
$size(H) \leftarrow size(H) + 1$;

while $(i > 0$ and $H[i] < H[\lfloor (i-1)/2 \rfloor])$
$swap(H[i], H[\lfloor (i-1)/2 \rfloor])$;
$i \leftarrow \lfloor (i-1)/2 \rfloor$;
End

Complexity?

INSERT(x, H) (Cont.)

Begin
  $i \leftarrow size(H)$;
  $H[size] \leftarrow x$;
  $size(H) \leftarrow size(H) + 1$;

  while $(i > 0$ and $H[i] < H[\lfloor (i-1)/2 \rfloor])$
    $swap(H[i], H[\lfloor (i-1)/2 \rfloor])$;
    $i \leftarrow \lfloor (i-1)/2 \rfloor$;
End

Complexity: $\mathcal{O}(\log n)$.

The Remaining Operations on Binary Heap

- $\textsc{Decrease-Key}(p, \Delta, H)$: Decrease the value of the key p by amount Δ.
  - Similar to $\textsc{Insert}(x, H)$.
  - Do it as an exercise!
  - **Complexity?**

- $\textsc{Merge}(H_1, H_2)$: Merge two heaps H_1 and H_2.

The Remaining Operations on Binary Heap

- DECREASE-KEY(p, Δ, H): Decrease the value of the key p by amount Δ.
  - Similar to INSERT(x, H).
  - Do it as an exercise!
  - **Complexity:** $\mathcal{O}(n)$.
  - **Note:** Searching for p takes $\mathcal{O}(n)$
  - Can you do it in $\mathcal{O}(\log n)$?

- MERGE(H_1, H_2): Merge two heaps H_1 and H_2.

The Remaining Operations on Binary Heap

- DECREASE-KEY(p, Δ, H): Decrease the value of the key p by amount Δ.
  - Similar to INSERT(x, H).
  - Do it as an exercise!
  - **Complexity:** $\mathcal{O}(n)$.
  - Needs some additional information called MAP!
  - MAP: Stores the index corresponding to each key.

- MERGE(H_1, H_2): Merge two heaps H_1 and H_2.

The Remaining Operations on Binary Heap

- DECREASE-KEY(p, Δ, H): Decrease the value of the key p by amount Δ.
  - Similar to INSERT(x, H).
  - Do it as an exercise!
  - **Complexity:** $\mathcal{O}(n)$.
  - Needs some additional information called MAP!
  - MAP: Stores the index corresponding to each key.

- MERGE(H_1, H_2): Merge two heaps H_1 and H_2.
  - **Complexity?**

The Remaining Operations on Binary Heap

- DECREASE-KEY(p, Δ, H): Decrease the value of the key p by amount Δ.
  - Similar to INSERT(x, H).
  - Do it as an exercise!
  - **Complexity:** $\mathcal{O}(n)$.
  - Needs some additional information called MAP!
  - MAP: Stores the index corresponding to each key.

- MERGE(H_1, H_2): Merge two heaps H_1 and H_2.
  - **Complexity:** $\mathcal{O}(n \log n)$
  - Can you do it in $\mathcal{O}(n)$?

Building a Binary heap

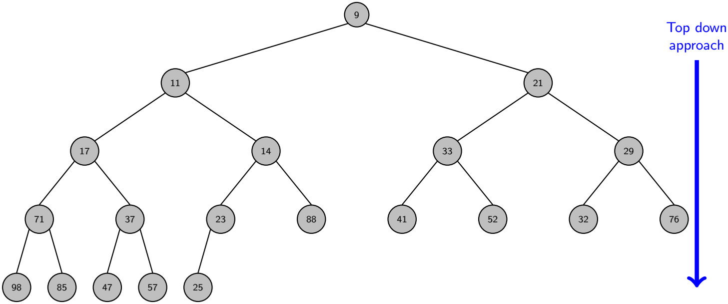
Building a Binary Heap Incrementally

Problem: Given elements $\{x_0, \dots, x_{n-1}\}$, build a binary heap H storing them.

Building a Binary Heap Incrementally

Problem: Given elements $\{x_0, \ldots, x_{n-1}\}$, build a binary heap H storing them.
Trivial Solution: Build the Binary heap incrementally.

$\textsc{CreateHeap}(H)$:
  for $i = 0$ to $n - 1$
    $\textsc{Insert}(x_i, H)$;



H:	9	11	21	17	14	33	29	71	37	23	88	41	52	32	76	98	85	47	57	25		$\cdots$	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31

Time Complexity

- Consider a complete binary tree of height h with k leaf nodes in the last level.
- The total number of nodes $n = (2^h - 1) + k$.
- Therefore, number of leaf nodes is equal to

$$\begin{aligned} k + (2^{h-1} - \lceil k/2 \rceil) &= 2^{h-1} + \lfloor k/2 \rfloor \\ &= \left\lceil \frac{1}{2}\{2^h + k - 1\} \right\rceil = \lceil n/2 \rceil \end{aligned}$$

Time Complexity

- Consider a complete binary tree of height h with k leaf nodes in the last level.
- The total number of nodes $n = (2^h - 1) + k$.
- Therefore, number of leaf nodes is equal to

$$
\begin{aligned}
k + (2^{h-1} - \lceil k/2 \rceil) &= 2^{h-1} + \lfloor k/2 \rfloor \\
&= \left\lceil \frac{1}{2}\{2^h + k - 1\} \right\rceil = \lceil n/2 \rceil
\end{aligned}
$$

- Time complexity for inserting a leaf node $= \mathcal{O}(\log n)$
- Time complexity for building a heap incrementally is $\mathcal{O}(n \log n)$.

Time Complexity

- Consider a complete binary tree of height h with k leaf nodes in the last level.
- The total number of nodes $n = (2^h - 1) + k$.
- Therefore, number of leaf nodes is equal to

$$\begin{aligned} k + (2^{h-1} - \lceil k/2 \rceil) &= 2^{h-1} + \lfloor k/2 \rfloor \\ &= \left\lceil \frac{1}{2}\{2^h + k - 1\} \right\rceil = \lceil n/2 \rceil \end{aligned}$$

- Time complexity for inserting a leaf node $= \mathcal{O}(\log n)$
- Time complexity for building a heap incrementally is $\mathcal{O}(n \log n)$.

A Binary Heap can be build in $\mathcal{O}(n)$ time.

Time Complexity

- Consider a complete binary tree of height h with k leaf nodes in the last level.
- The total number of nodes $n = (2^h - 1) + k$.
- Therefore, number of leaf nodes is equal to

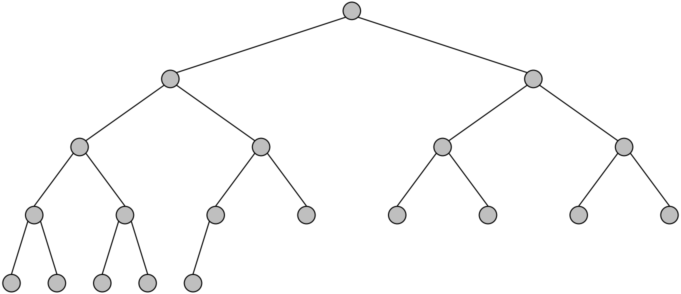
$$k + (2^{h-1} - \lceil k/2 \rceil) \;=\; 2^{h-1} + \lfloor k/2 \rfloor$$
$$= \left\lceil \frac{1}{2}\{2^h + k - 1\} \right\rceil \;=\; \lceil n/2 \rceil$$

- Time complexity for inserting a leaf node $= \mathcal{O}(\log n)$
- Time complexity for building a heap incrementally is $\mathcal{O}(n \log n)$.

A Binary Heap can be build in $\mathcal{O}(n)$ time.

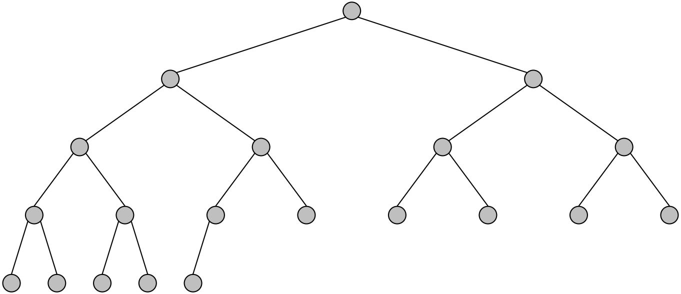
Conclusion: $\mathcal{O}(n)$ algorithm $\Rightarrow$ each leaf nodes must take $\mathcal{O}(1)$.

An Alternate Approach



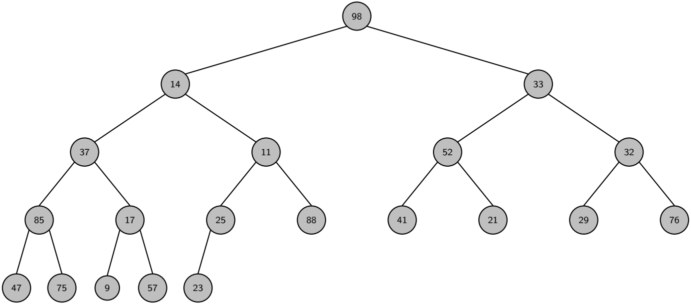
- Heap Property: Every node stores values smaller than its children.

An Alternate Approach



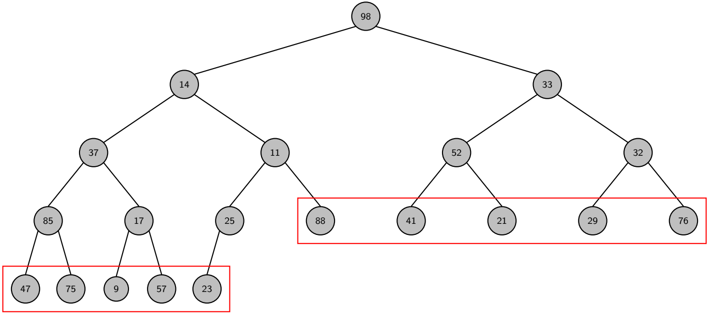
- Heap Property: Every node stores values smaller than its children.
- **Note:** Only need to ensure this property at each node.

An Alternate Approach



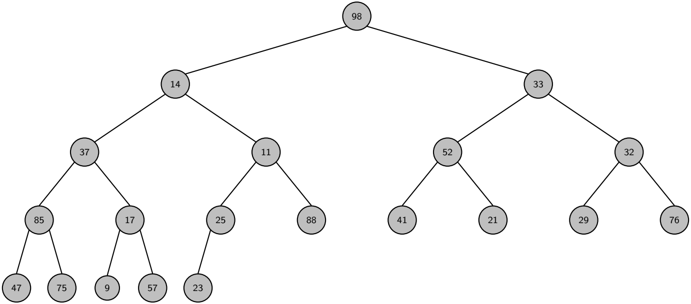
- Heap Property: Every node stores values smaller than its children.
- **Note:** Only need to ensure this property at each node.
- **Question:** In any complete binary tree, how many nodes satisfy the heap property?

An Alternate Approach



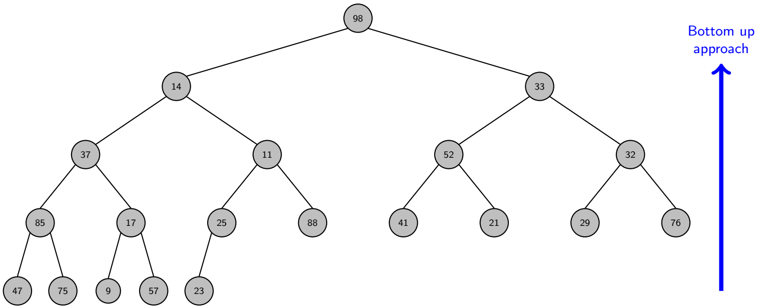
- Heap Property: Every node stores values smaller than its children.
- **Note:** Only need to ensure this property at each node.
- **Question:** In any complete binary tree, how many nodes satisfy the heap property?
  All the leaf nodes surely does!

An Alternate Approach



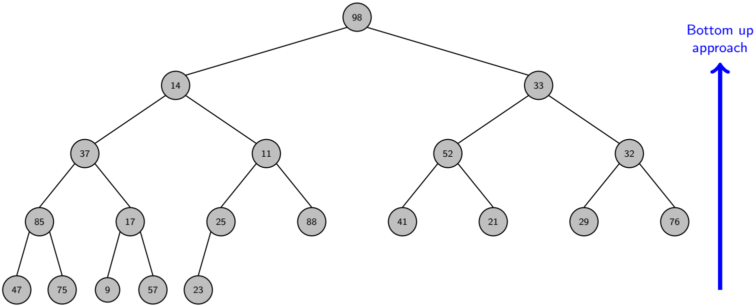
- Heap Property: Every node stores values smaller than its children.
- **Note:** Only need to ensure this property at each node.
- **Question:** In any complete binary tree, how many nodes satisfy the heap property?
  All the leaf nodes surely does!
- **Question:** Does this suggest a new approach to build binary heap?

An Alternate Approach



- Heap Property: Every node stores values smaller than its children.
- **Note:** Only need to ensure this property at each node.
- **Question:** In any complete binary tree, how many nodes satisfy the heap property? All the leaf nodes surely does!
- **Question:** Does this suggest a new approach to build binary heap?

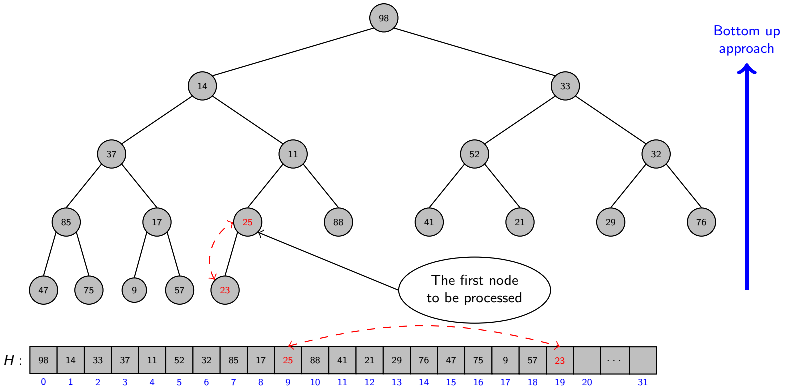
An Alternate Approach



H :	98	14	33	37	11	52	32	85	17	25	88	41	21	29	76	47	75	9	57	23		···	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31

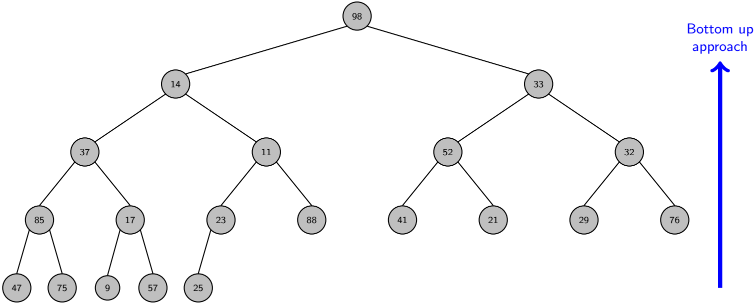
- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.

An Alternate Approach



- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.

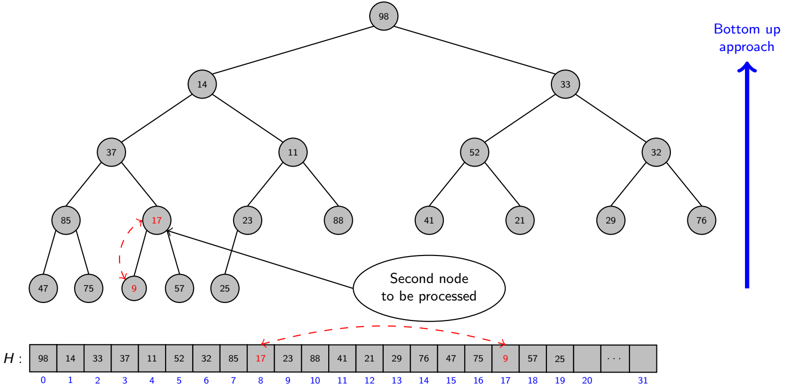
An Alternate Approach



H :	98	14	33	37	11	52	32	85	17	23	88	41	21	29	76	47	75	9	57	25		⋯	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31

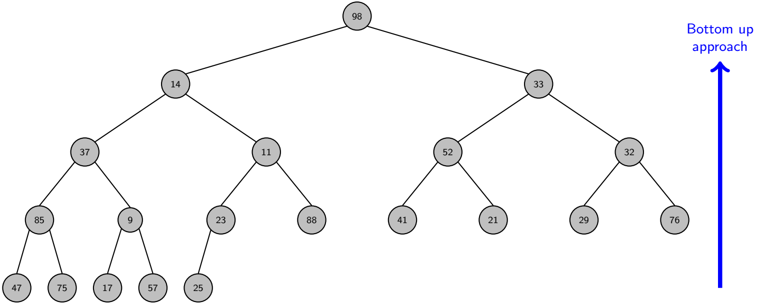
- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.

An Alternate Approach



- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.

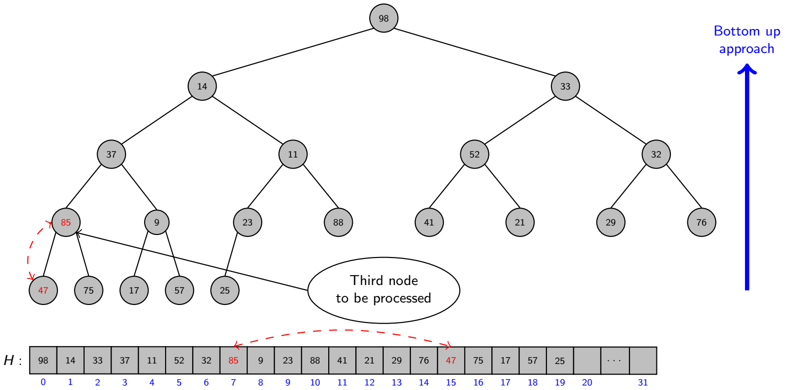
An Alternate Approach



H :	98	14	33	37	11	52	32	85	9	23	88	41	21	29	76	47	75	17	57	25		$\cdots$	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31

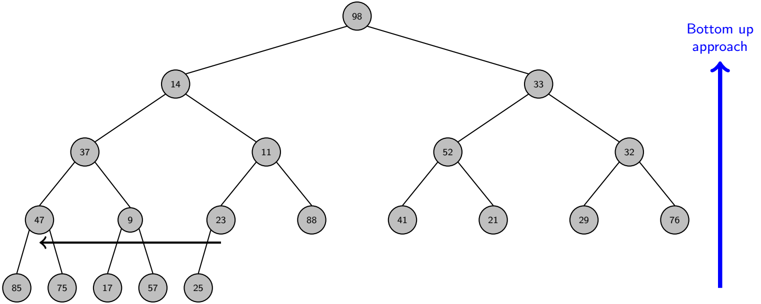
- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.

An Alternate Approach



- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.

An Alternate Approach

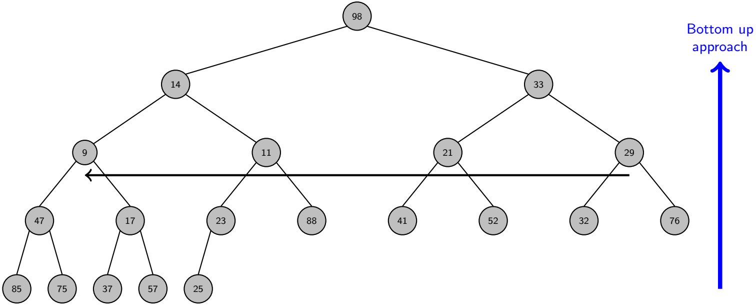


H :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31
98	14	33	37	11	52	32	47	9	23	88	41	21	29	76	85	75	17	57	25		$\cdots$	

- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.

An Alternate Approach

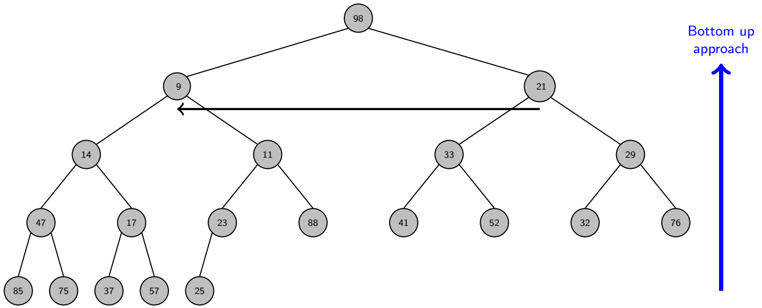


H :

98	9	21	14	11	33	29	47	17	23	88	41	52	32	76	85	75	37	57	25		$\cdots$	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31

- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.

An Alternate Approach

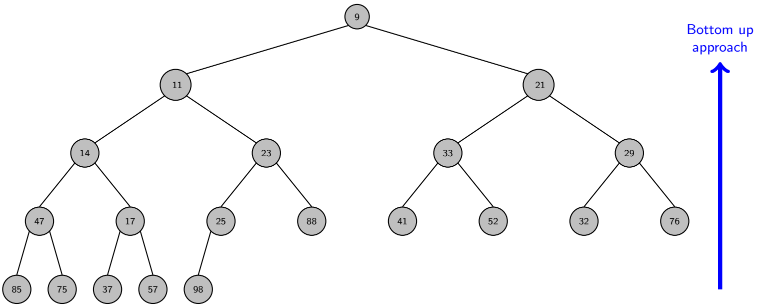


H :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31
98	9	21	14	11	33	29	47	17	23	88	41	52	32	76	85	75	37	57	25		$\cdots$	

- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.

An Alternate Approach



H :	9	11	21	14	23	33	29	47	17	25	88	41	52	32	76	85	75	37	57	98		$\cdots$	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		31

- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.
- Let v be a node corresponding to index i in H.
- The process of restoring heap property at i called $\textsc{Heapify}(i, H)$.

HEAPIFY(i, H)

For node i, compare its value with those of its children

- If it is greater than any of its children
  - Swap it with smallest child
  - and move down ...
- Else stop.

HEAPIFY(i, H)

```
Begin
  n ← size(H) − 1;
  Flag ← true;

  while (i ≤ ⌊(n − 1)/2⌋ and Flag = true)
    min ← i;
    if (H[i] > H[2i + 1])
      min ← 2i + 1;

    if (2i + 2 ≤ n and H[min] > H[2i + 2])
      min ← 2i + 2;

    if (min ≠ i)
      swap(H[i], H[min]);
      i ← min;
    else
      Flag ← false;

End
```

Complexity

- How many nodes of height h can there be in a complete binary tree of n nodes?

Complexity

- How many nodes of height h can there be in a complete binary tree of n nodes?
- **Note:** Each sub-tree is also a complete binary tree.
  - A sub-tree of height h has at least 2^h nodes.
  - No two sub-tree of height h have any element in common.

Complexity

- How many nodes of height h can there be in a complete binary tree of n nodes?
- **Note:** Each sub-tree is also a complete binary tree.
  - A sub-tree of height h has at least 2^h nodes.
  - No two sub-tree of height h have any element in common.
- $\therefore$ the $\#$ nodes of height h is bounded above by $\frac{n}{2^h}$.
- Hence, time complexity of building a heap is given by

$$\begin{aligned}\sum_{h=1}^{\log n} \frac{n}{2^h} \cdot \mathcal{O}(h) \;\leq\; cn \sum_{h=1}^{\log n} \frac{h}{2^h} \;<\; cn \sum_{h=1}^{\infty} \frac{h}{2^h} \\ = \; cn \cdot \frac{(1/2)}{(1-1/2)^2} \quad [\because \textstyle\sum_{i=1}^{\infty} i x^i = \frac{x}{(1-x)^2} \text{ for } |x| < 1] \\ = \; 2cn \;=\; \mathcal{O}(n).\end{aligned}$$

Heapsort

Heapsort

- Build heap H on the given n elements.
- **While** (H is not empty)
  $x \leftarrow \textsc{Extract-Min}(H)$;
  print x;
- **Complexity:** $\mathcal{O}(n \log n)$.

Homework:

- Implement a BINARY-MAX-HEAP in C.
- Use it to sort numbers in an decreasing order.
- For a given n,
  - Take (fixed) m many random inputs of size n each.
  - Compute the average time take by your Heapsort program.
- Repeat the above process for $n = 4, 5, \ldots, 1000$.
- Plot the values in a graph where x-axis is n and y-axis denotes the average time taken for each n.

Thank You for your kind attention!

Books and Other Materials Consulted

1. *Introduction to Algorithms* by Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein.
2. Taken from Prof. Surendar Baswana (CSE, IIT Kanpur) lecture slides.
3. Taken from Prof. Surendar Baswana (CSE, IIT Kanpur) lecture slides.

Questions!!