Binary Heaps (Cont.), Heapsort and Huffman Encoding

Subhabrata Samajder

IIIT, Delhi
Winter Semester,
28th April, 2023

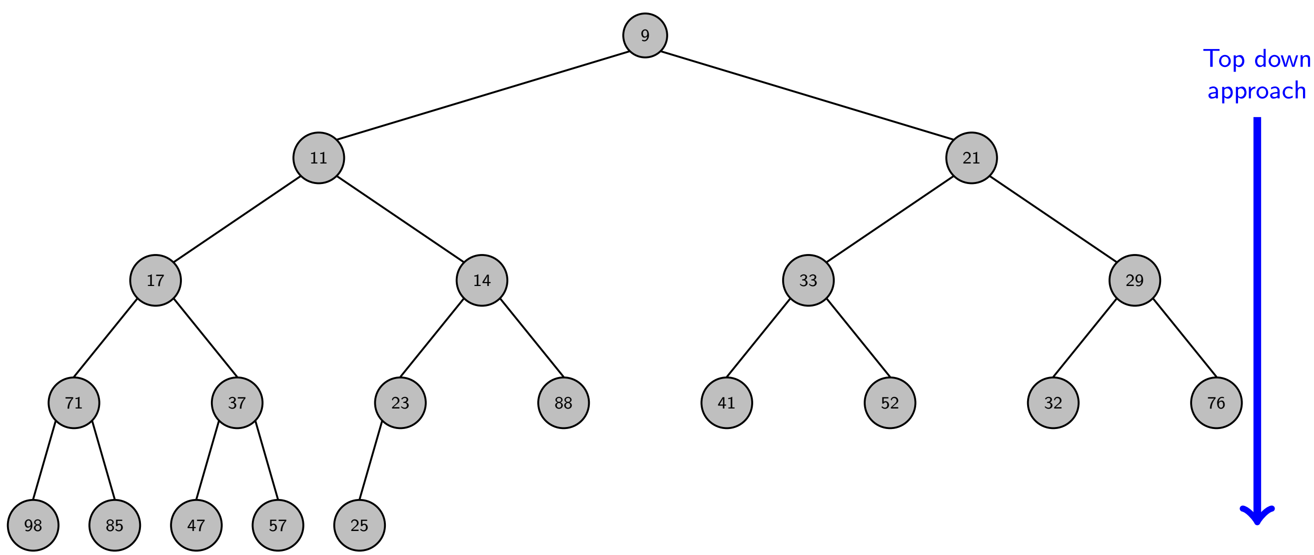
Building a Binary Heap Incrementally

Problem: Given elements $\{x_0, \ldots, x_{n-1}\}$, 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 :

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		···	

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

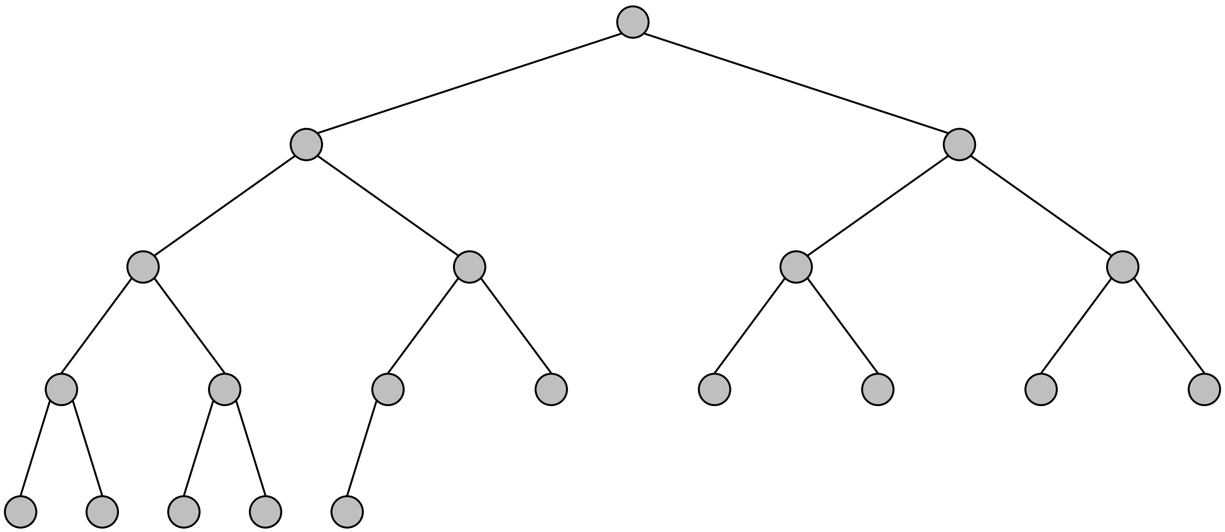
$$
\begin{aligned}
k + (2^{h-1} - \lceil k/2 \rceil) &= 2^{h-1} + \lfloor k/2 \rfloor \\
&= \left\lceil \frac{1}{2}\{2^h + k - 1\} \right\rceil = \lceil n/2 \rceil
\end{aligned}
$$

- 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.

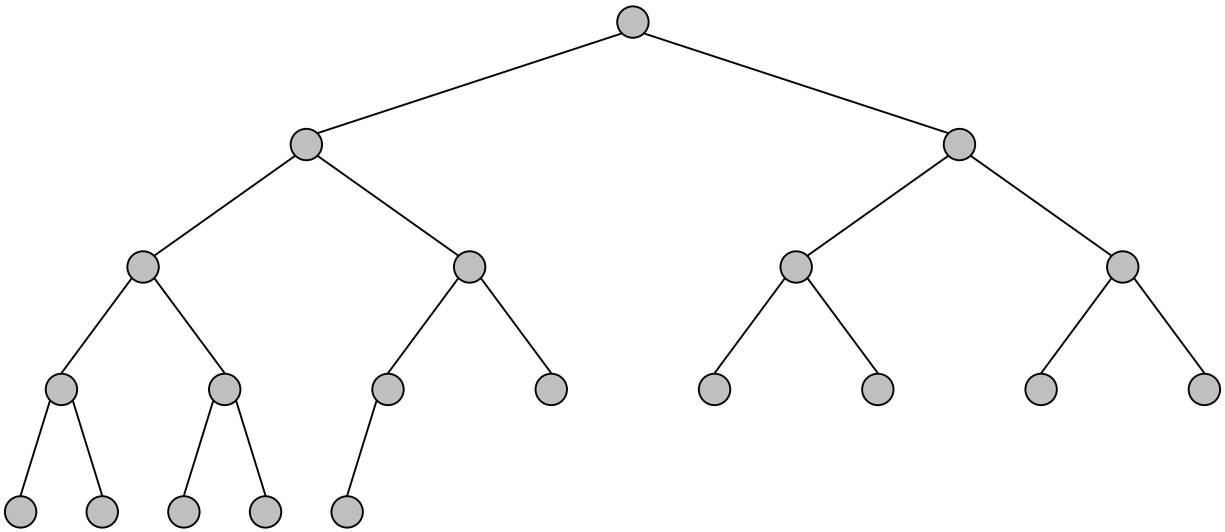
Conclusion: $\mathcal{O}(n)$ algorithm $\Rightarrow$ each leaf nodes must take $\mathcal{O}(1)$.

An Alternate Approach



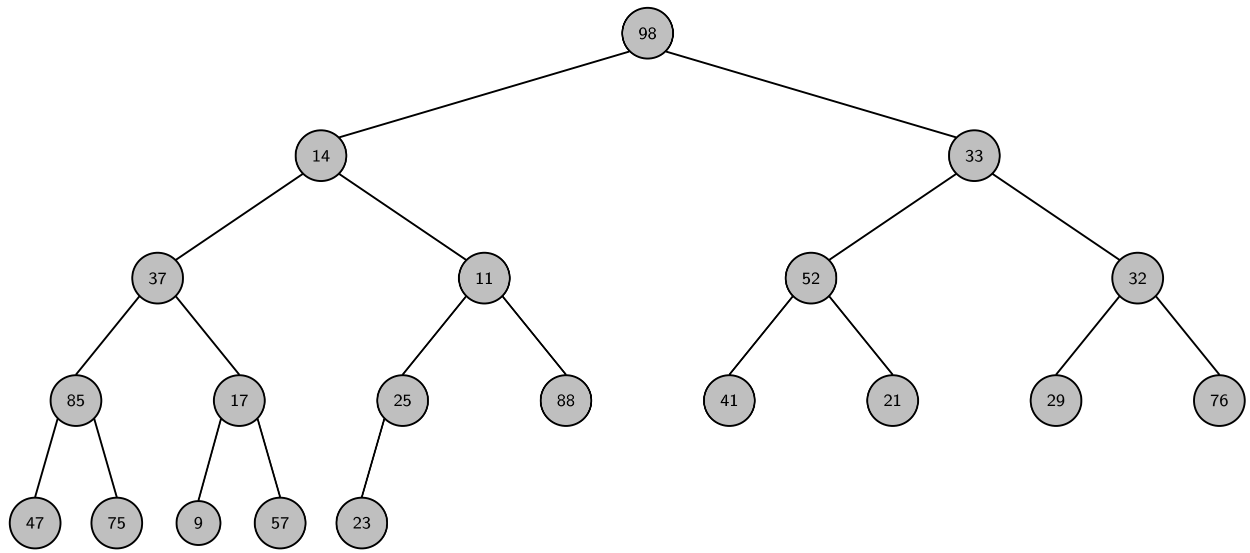
- Heap Property: Every node stores values smaller than its children.

An Alternate Approach



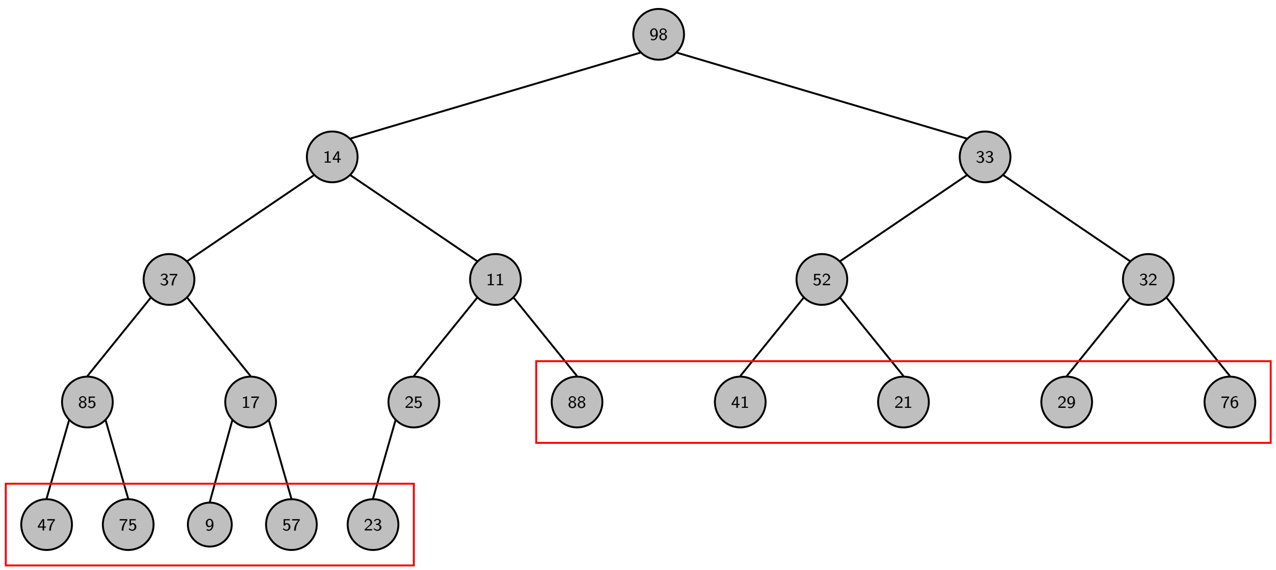
- Heap Property: Every node stores values smaller than its children.
- **Note:** Only need to ensure this property at each node.

An Alternate Approach



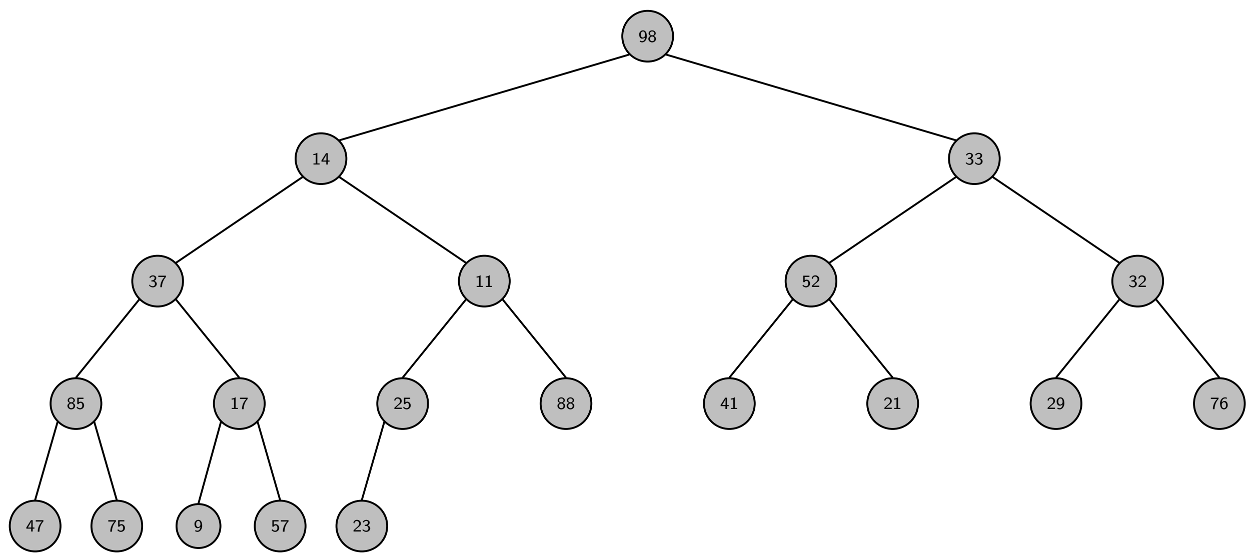
- Heap Property: Every node stores values smaller than its children.
- **Note:** Only need to ensure this property at each node.
- **Question:** In any complete binary tree, how many nodes satisfy the heap property?

An Alternate Approach



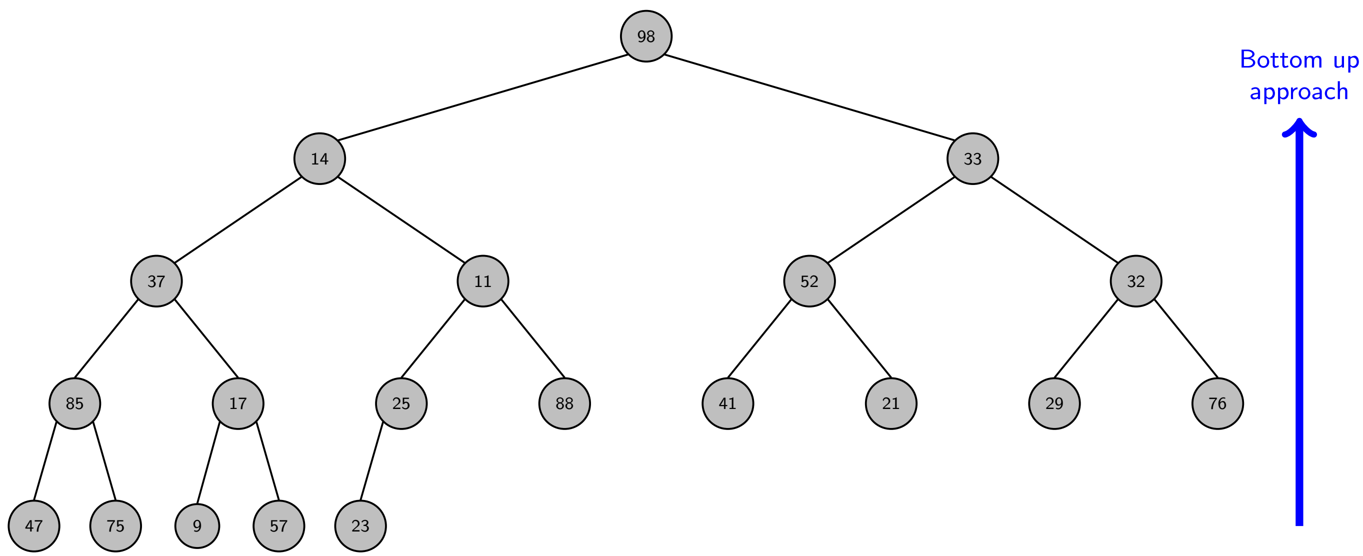
- Heap Property: Every node stores values smaller than its children.
- **Note:** Only need to ensure this property at each node.
- **Question:** In any complete binary tree, how many nodes satisfy the heap property?
  All the leaf nodes surely does!

An Alternate Approach



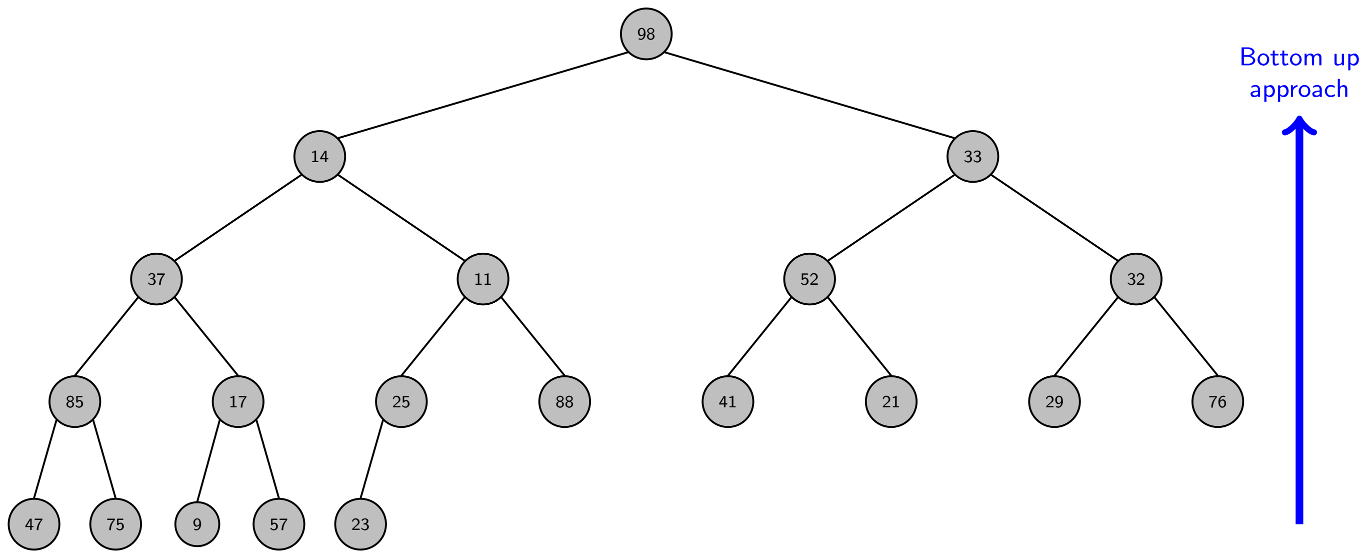
- Heap Property: Every node stores values smaller than its children.
- **Note:** Only need to ensure this property at each node.
- **Question:** In any complete binary tree, how many nodes satisfy the heap property? All the leaf nodes surely does!
- **Question:** Does this suggest a new approach to build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

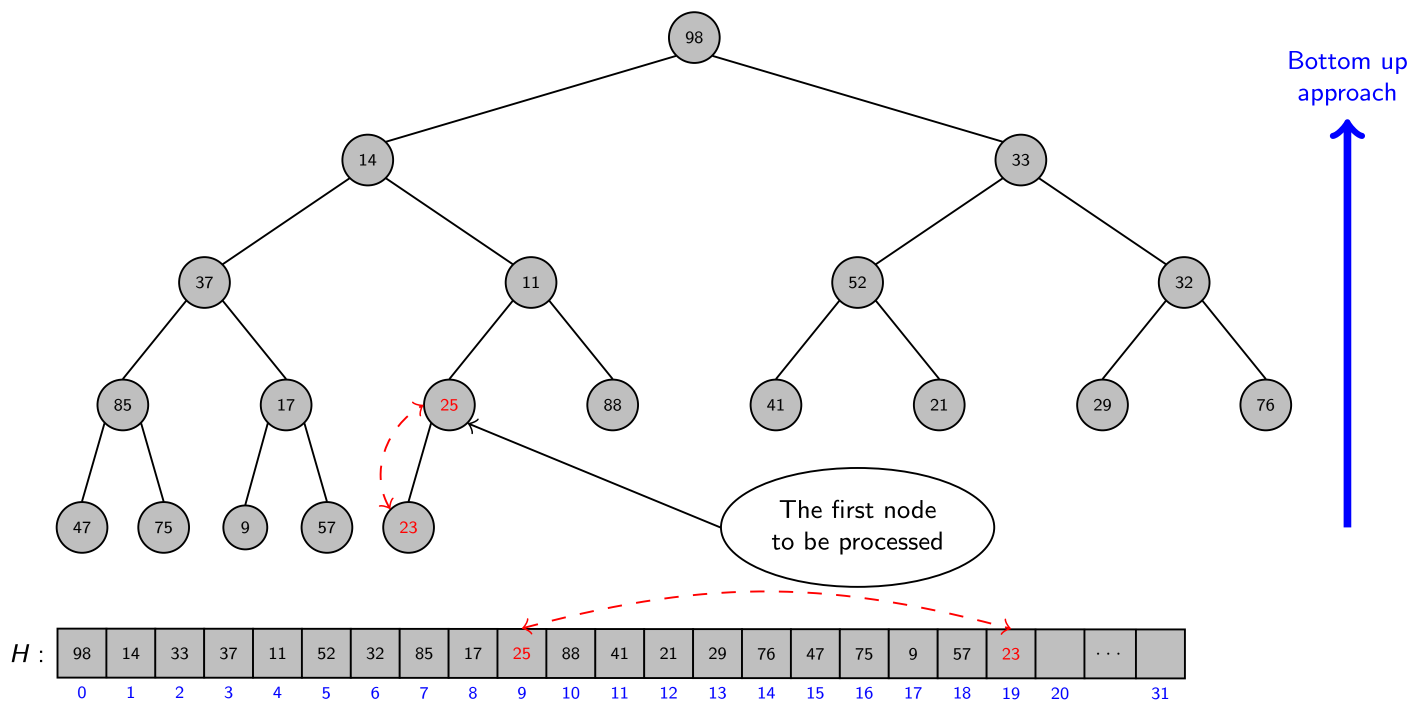


H :

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

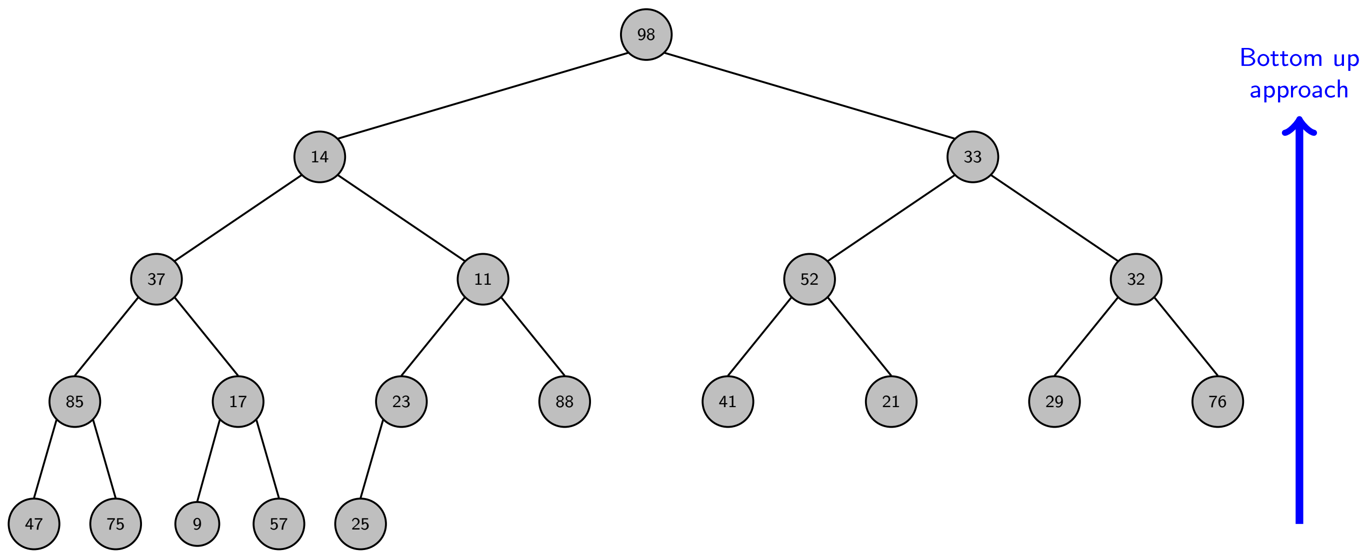
- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.

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.

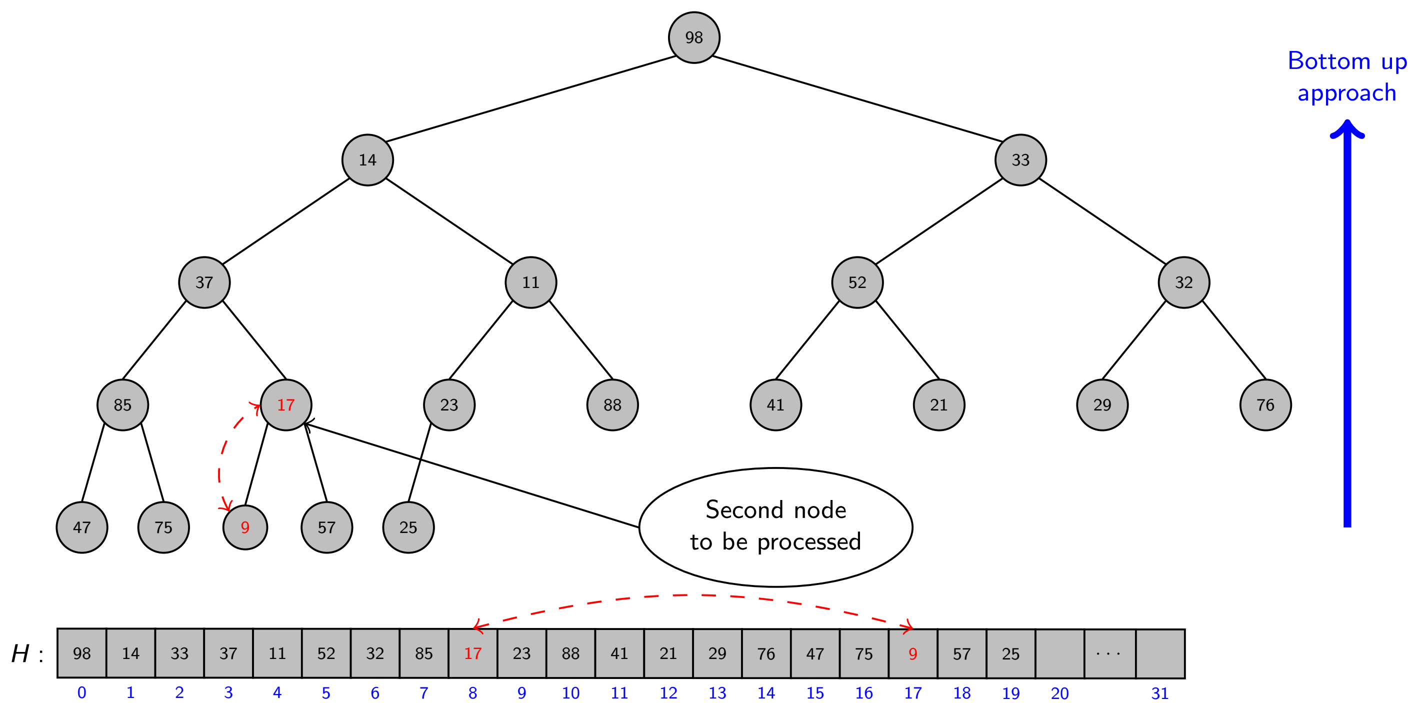
An Alternate Approach



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

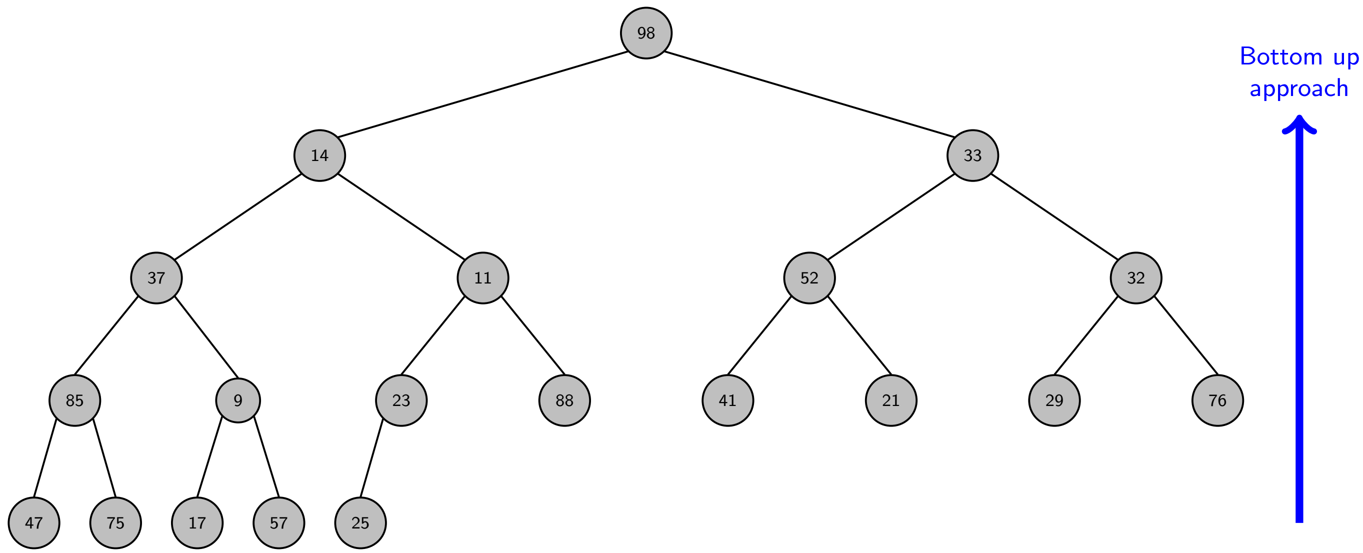
- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.

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.

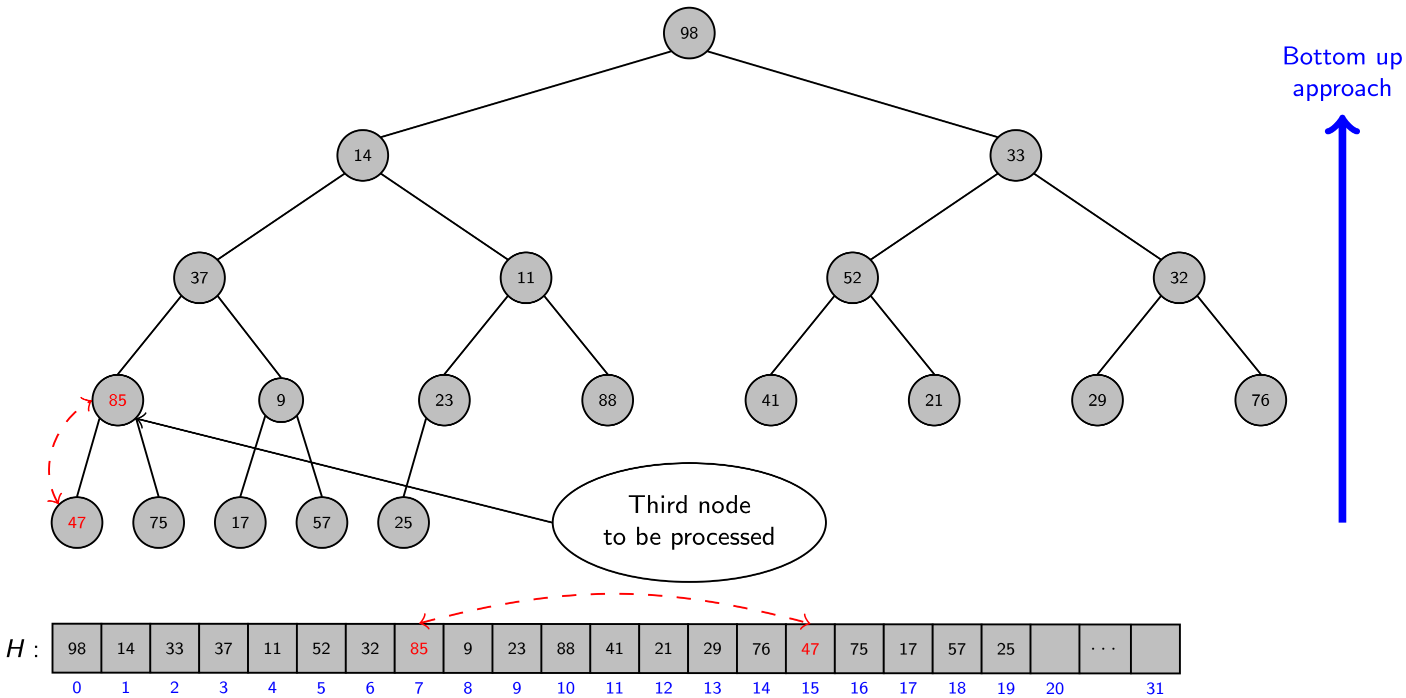
An Alternate Approach



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

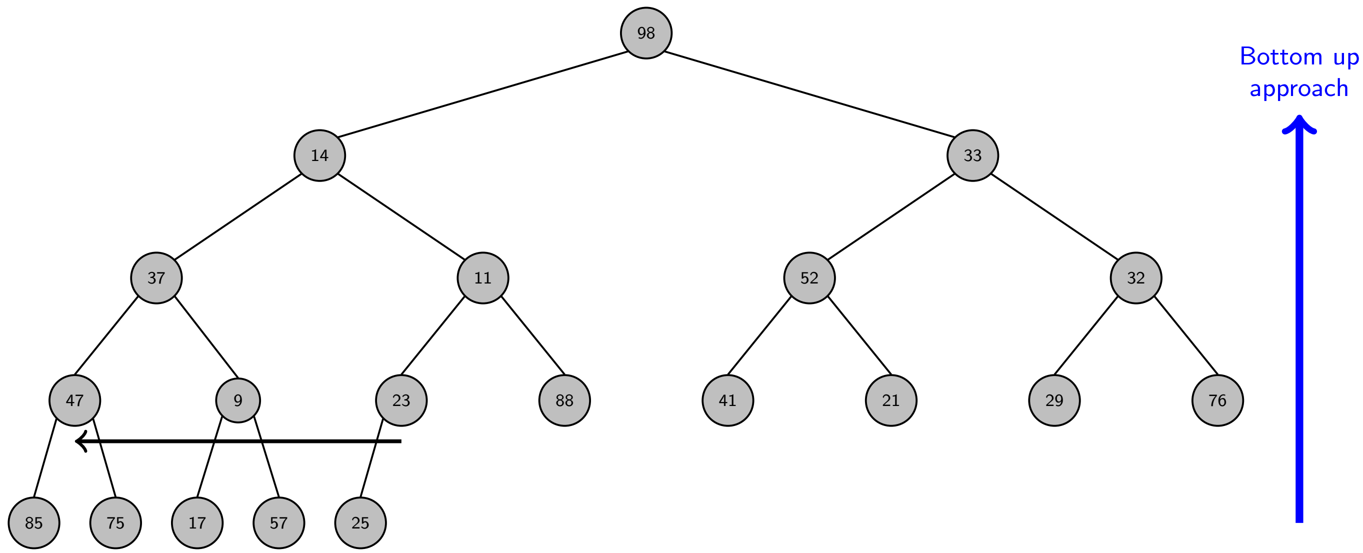
- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.

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.

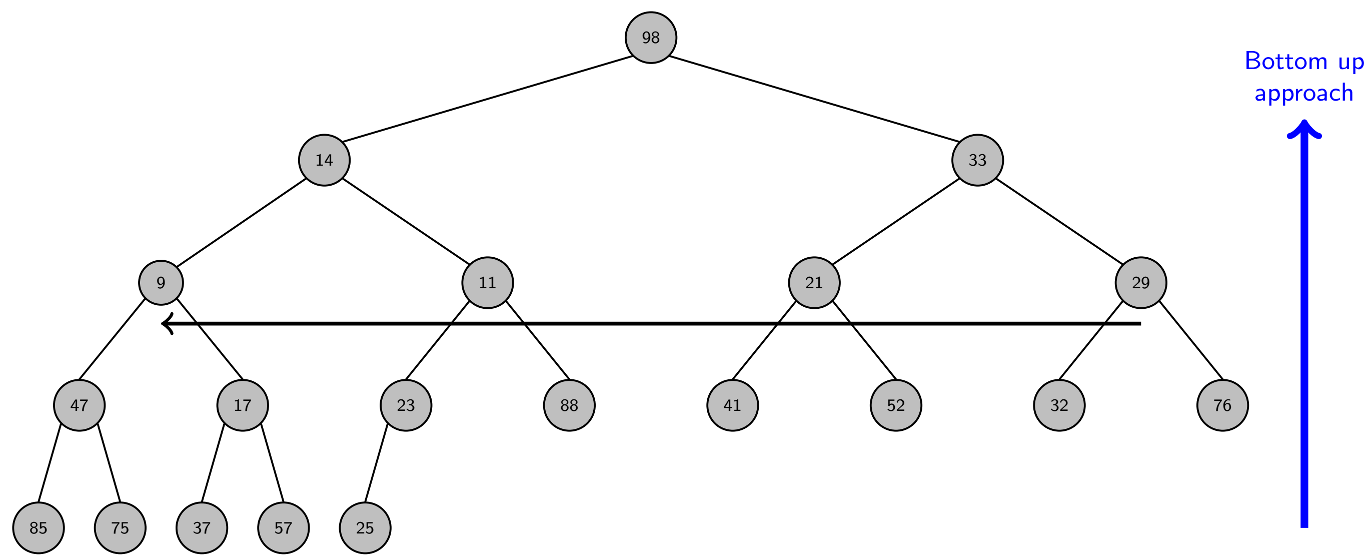
An Alternate Approach



H :	98	14	33	37	11	52	32	47	9	23	88	41	21	29	76	85	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.

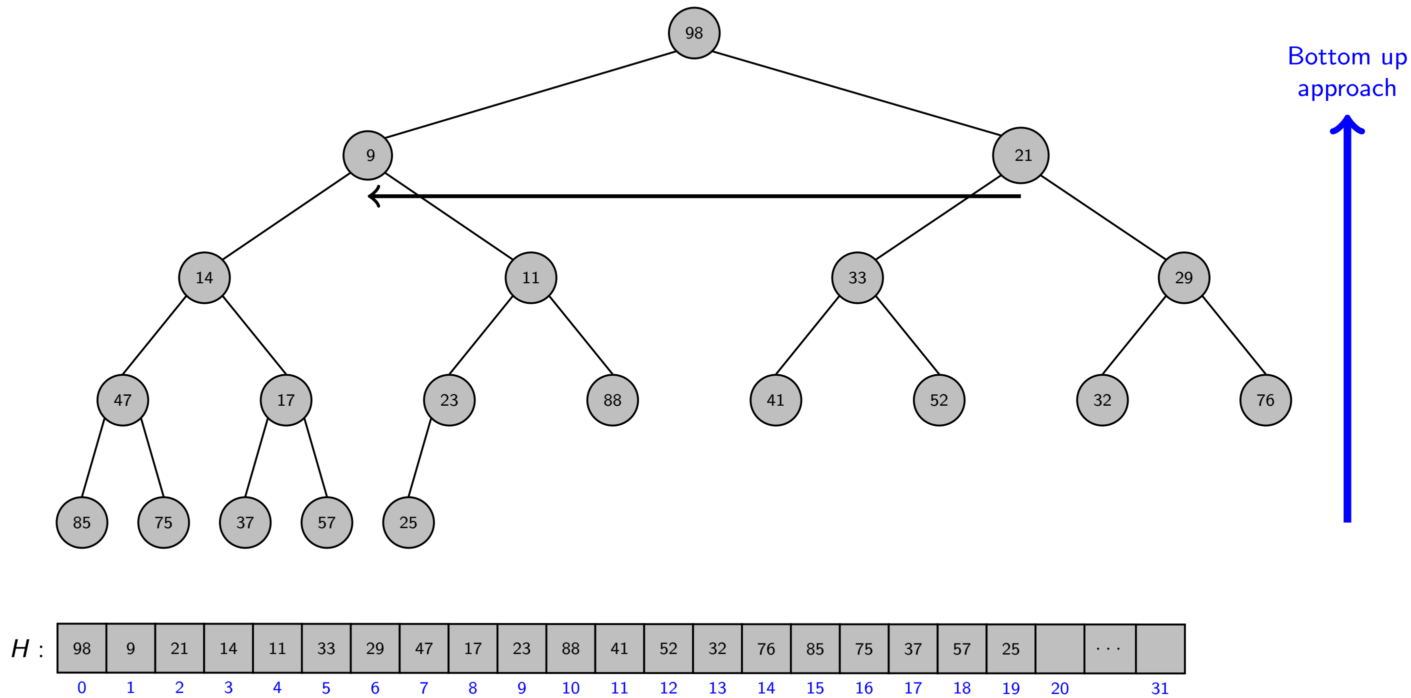
An Alternate Approach



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

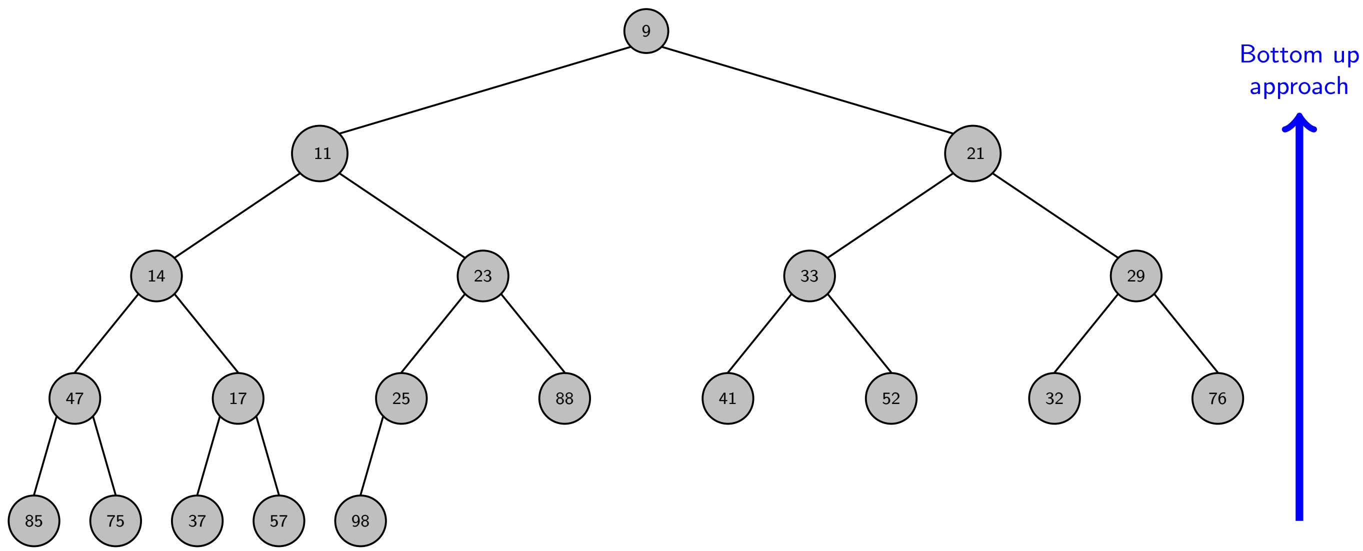
- Copy the given n elements $\{x_0, \ldots, x_{n-1}\}$ into an array H.
- The heap property holds for all the leaf nodes.
- Leaving all the leaf nodes, process the elements in the decreasing order of their index and set the heap property for each of them.

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 :	9	11	21	14	23	33	29	47	17	25	88	41	52	32	76	85	75	37	57	98		...	
	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
  $n \leftarrow size(H) - 1$;
  Flag $\leftarrow$ true;

  while ($i \leq \lfloor (n-1)/2 \rfloor$ and Flag $=$ true)
    min $\leftarrow i$;
    if ($H[i] > H[2i+1]$)
      min $\leftarrow 2i+1$;

    if ($2i+2 \leq n$ and $H[min] > H[2i+2]$)
      min $\leftarrow 2i+2$;

    if (min $\neq i$)
      $swap(H[i], H[\text{min}])$;
      $i \leftarrow min$;
    else
      Flag $\leftarrow$ 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 $\#$ trees of height h is bounded above by $\frac{n}{2^h}$.
- Hence, time complexity of building a heap is given by

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

Heapsort

Heapsort

- Build heap H on the given n elements.

- **While** (H is not empty)
  $x \leftarrow \textsc{Extract-Min}(H)$;
  print x;

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

Heapsort

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.

Huffman Coding

Huffman Coding

- Are used to compress information.
  - Like WinZip (although WinZip doesn't use the Huffman algorithm).
  - JPEGs do use Huffman as part of their compression process.

- **Basic idea:** Instead of storing characters in a file as 8-bit ASCII value, store the more frequently occurring characters using fewer bits and less frequently occurring characters using more bits
  - On average this should decrease the filesize (usually by 1/2).

Huffman Coding (Cont.)

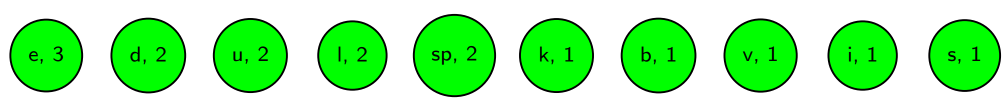
An Example: Consider the string,

"duke blue devils"

- Do a frequency count of the characters:

e	d	u	l	space	k	b	v	i	s
3	2	2	2	2	1	1	1	1	1

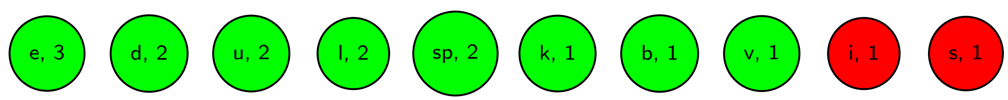
- Next we use a Greedy algorithm to build up a Huffman Tree.
  - Start with nodes for each character.



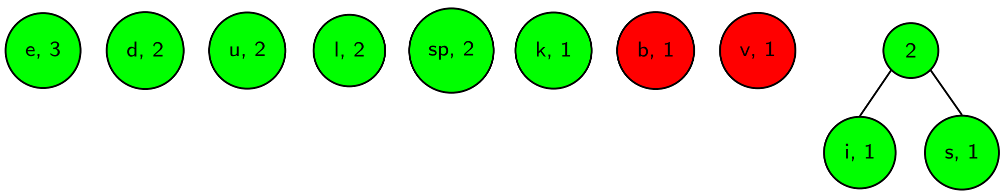
Huffman Coding (Cont.)

- Pick the nodes with the smallest frequency and combine them together to form a new node.
  - The selection of these nodes is the Greedy part.
- Remove the two selected nodes are from the set and replace it with a combined node.
- Continue until only 1 node left in the set.

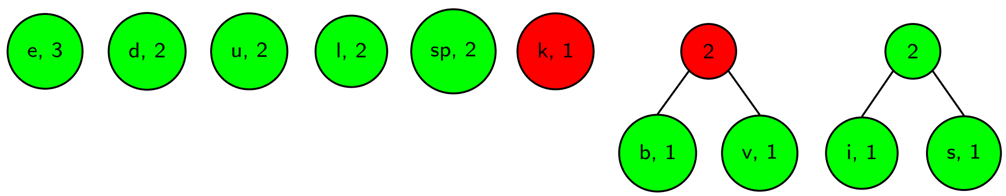
Huffman Coding (Cont.)



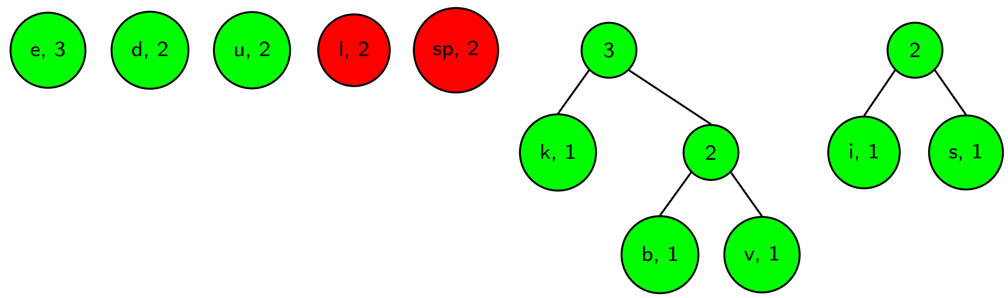
Huffman Coding (Cont.)



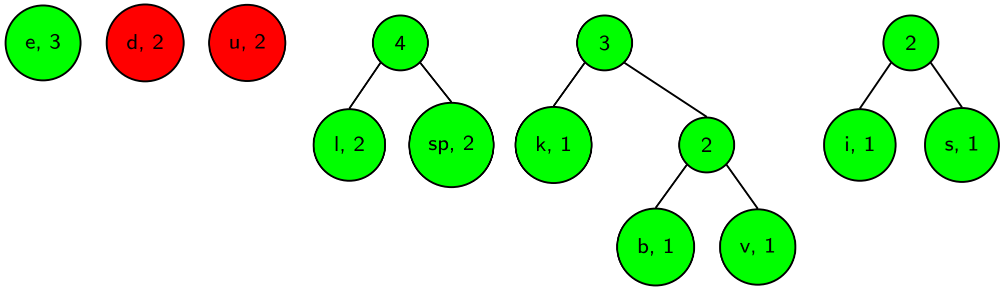
Huffman Coding (Cont.)



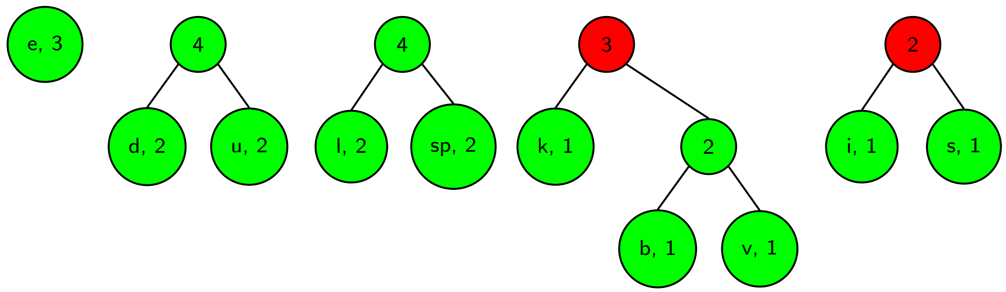
Huffman Coding (Cont.)



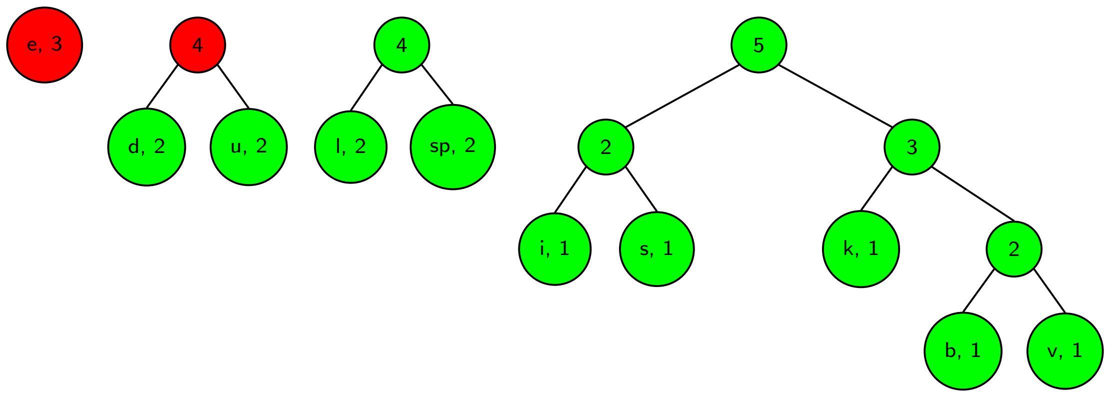
Huffman Coding (Cont.)



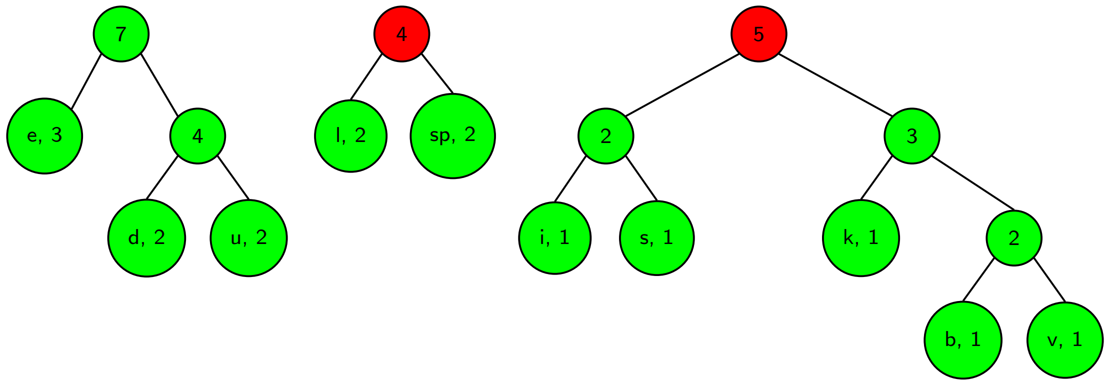
Huffman Coding (Cont.)



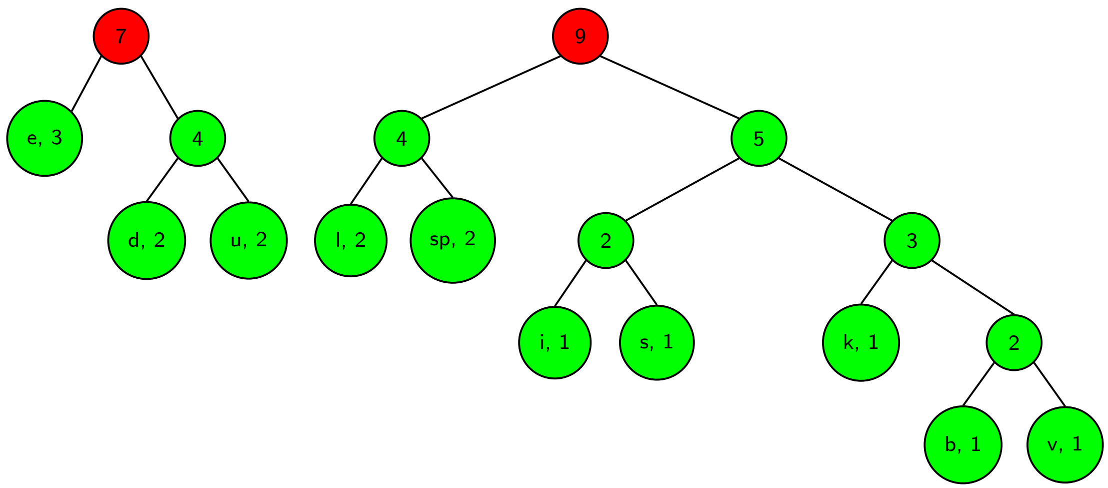
Huffman Coding (Cont.)



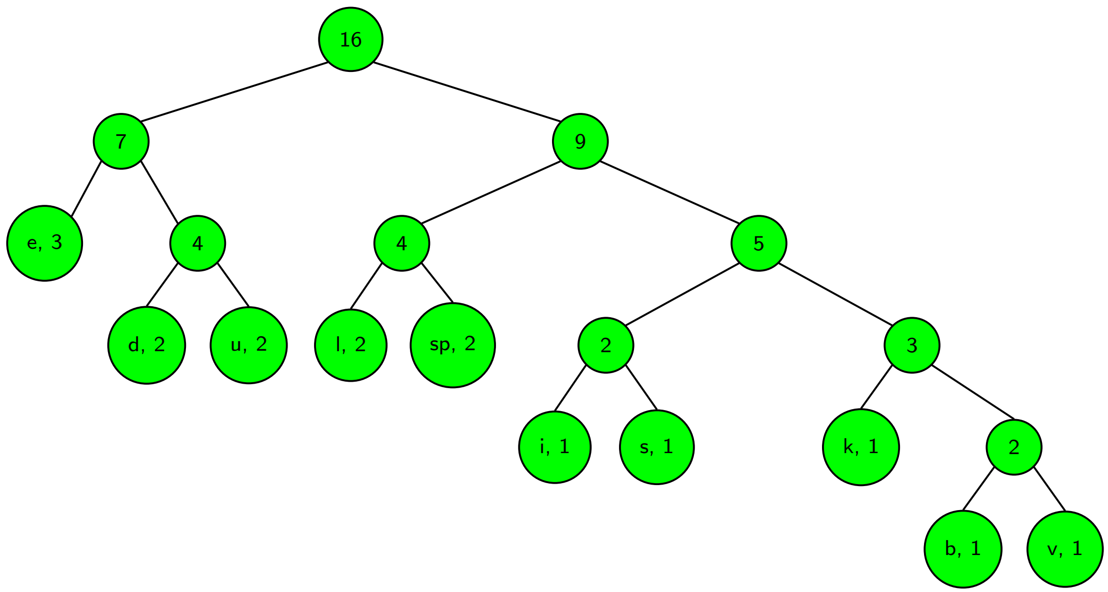
Huffman Coding (Cont.)



Huffman Coding (Cont.)



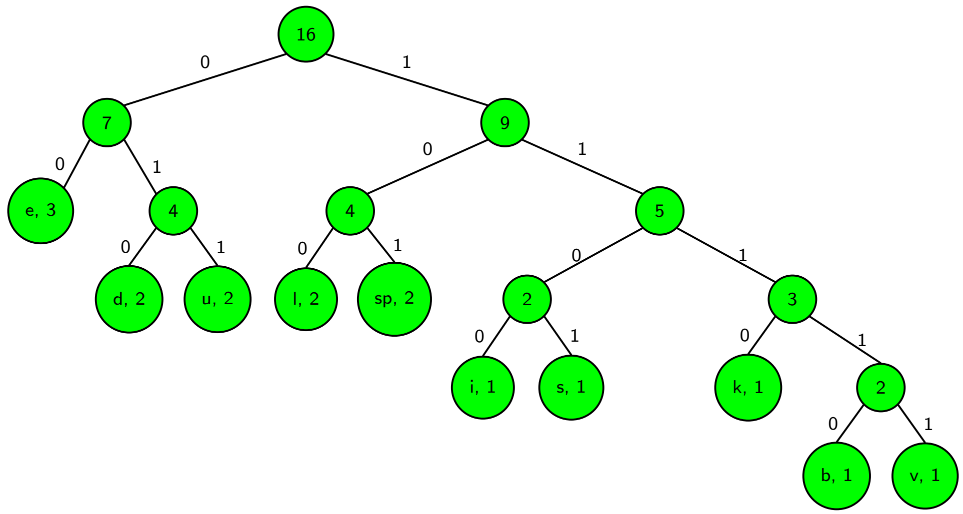
Huffman Coding (Cont.)



Huffman Coding: Assign Codes

- **Assign codes to the tree:**
  - 0: For left child.
  - 1: For Right child.
- A traversal of the tree from root to leaf gives the Huffman code for that particular leaf character.
- **Note:** No code is the prefix of another code.

Huffman Coding: Code Assignment



e	d	u	l	space	i	s	k	b	v
00	010	011	100	101	1100	1101	1110	11110	11111

Huffman Coding: Compression

- Use these codes to encode the string.
- Encoding of "duke blue devils":

$010\ 011\ 1110\ 00\ 101\ 11110\ 100\ 011\ 00\ 101\ 010\ 00\ 11111\ 1100\ 100\ 1101$

- Grouped then into bytes:

$01001111\quad 10001011\quad 11101000\quad 11001010\quad 10001111\quad 11100100\quad 1101xxxx$

- $\therefore$ it takes 7 bytes of space.
- In contrast, the uncompressed string takes 16 characters $\times$ 1 byte/char = 16 bytes.

Huffman Decoding: Uncompression

- Reading the compressed file bit by bit.
  - Start at the root of the tree.
  - If a 0 is read, head left.
  - If a 1 is read, head right.
  - When a leaf is reached decode that character and start over again at the root of the tree

- $\therefore$ Huffman table information needs to be saved as a header in the compressed file.
  - Doesnt add a significant amount of size to the file for large files (which are the ones you want to compress anyway).
  - Or we could use a fixed universal set of codes/freqencies.

Homework

- Implement Huffman encoding and decoding in C.
- Using it create a compression software that takes as input a text file and outputs a compressed file with the Huffman encoding table in the header of the file.
- Also create a uncompression software that will take the compressed file created above and output the original text file.

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.
4. Huffman Coding part taken from the following website.

Questions!!