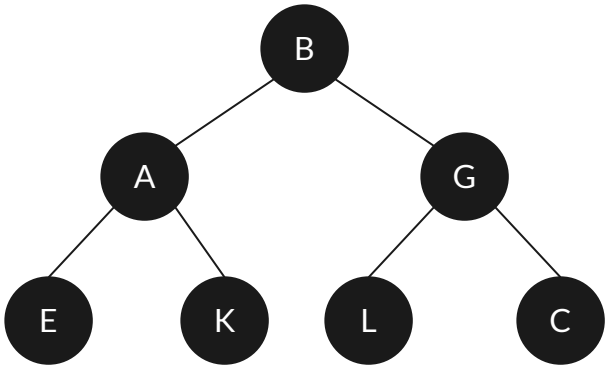
CIS 263 Introduction to Data Structures and Algorithms

Heap, and Heap Sort

Outline

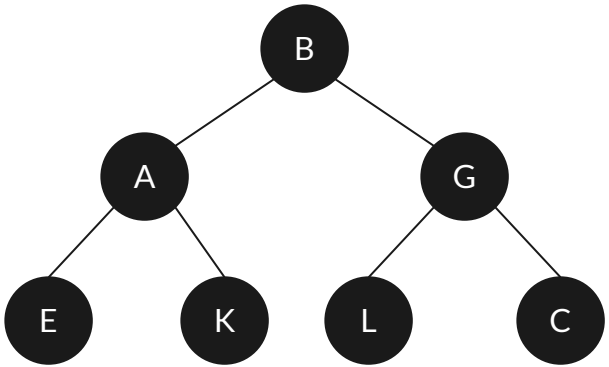
- Heap Data Structure
  - Full Binary Tree
  - Complete Binary Tree
- Operations
  - Insertion
  - Deletion
- Heap Sort

Full Binary Tree



- **Binary Tree**: Two children max; ordering is random
- **Binary** Search **Tree** is an special case of the **Binary Tree**
- **Full Binary Tree**
  - Every level is complete
  - Leaf nodes have equal depth

Full Binary Tree



Assuming index starting at: 1

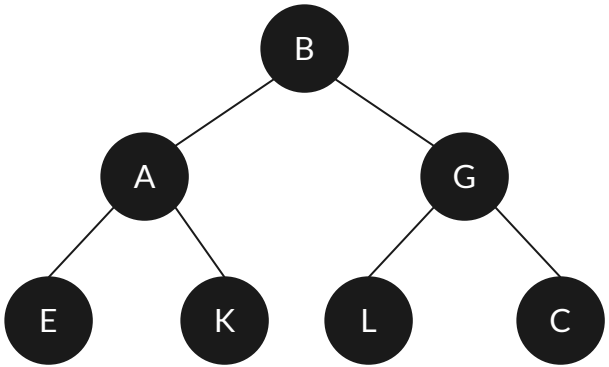
If a node is at index: *i*,

- The left child is at: *2i*
- The right child is at: *2i + 1*
- It's parent is at: i // 2 (floor value)

B	A	G	E	K	L	C

We have a Binary Tree using Array

Full Binary Tree



Assuming index starting at: 1

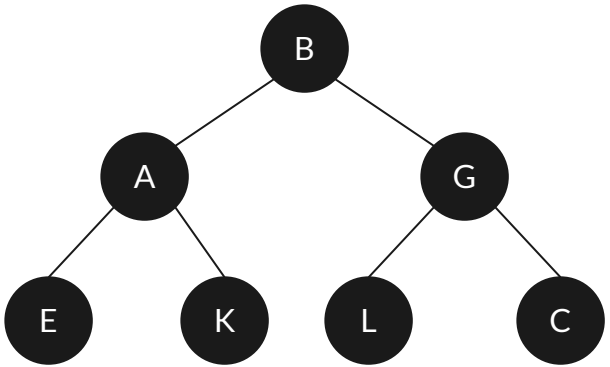
If a node is at index: *i*,

- The left child is at: *2i*
- The right child is at: *2i + 1*
- It's parent is at: i // 2 (floor value)

B	A	G	E	K	L	C

Heap wants to take advantage of both: **Array**, and **Binary Tree**

Complete Binary Tree



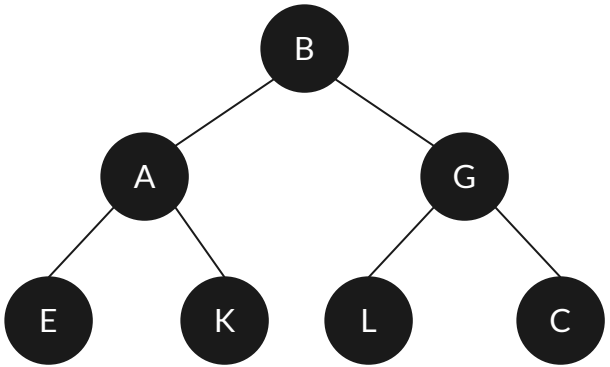
A Full Binary Tree is a Complete Binary Tree

If a node is at index: *i*,

- The left child is at: *2i*
- The right child is at: *2i + 1*
- Its parent is at: i // 2 (floor value)

B	A	G	E	K	L	C

Complete Binary Tree



A Full Binary Tree is a Complete Binary Tree

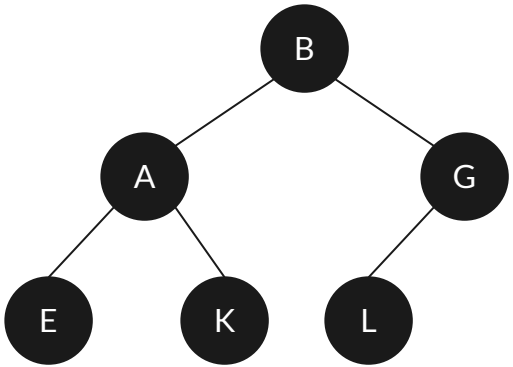
If a node is at index: *i*,

- The left child is at: *2i*
- The right child is at: *2i + 1*
- Its parent is at: i // 2 (floor value)

B	A	G	E	K	L	C

A complete Binary Tree not needs to be a Complete BT

Complete Binary Tree



What are the other properties?

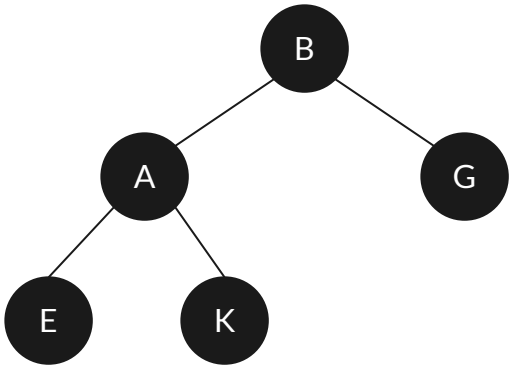
If a node is at index: *i*,

- The left child is at: *2i*
- The right child is at: *2i + 1*
- Its parent is at: i // 2 (floor value)

- Empty space only allowed at the end of the Array
- Deleting elements from right side (level by level still keeps the definition valid

B	A	G	E	K	L	

Complete Binary Tree



What are the other properties?

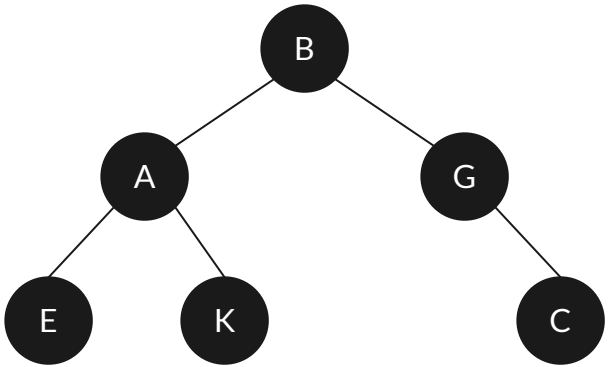
If a node is at index: *i*,

- The left child is at: *2i*
- The right child is at: *2i + 1*
- Its parent is at: i // 2 (floor value)

- Empty space only allowed at the end of the Array
- Deleting elements from right side (level by level still keeps the definition valid

B	A	G	E	K		

Not a Complete Binary Tree



What are the other properties?

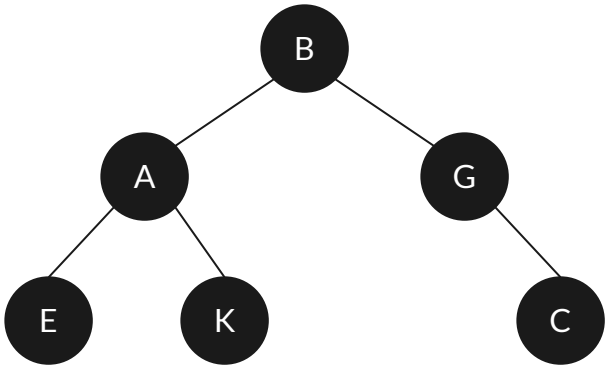
If a node is at index: *i*,

- The left child is at: *2i*
- The right child is at: *2i + 1*
- Its parent is at: i // 2 (floor value)

- What if the rule is not followed (example, if we remove L)?
- No more a Complete Binary Tree
- If we want to move C to left, the Tree Data structure gets broken

B	A	G	E	K		C

Not a Complete Binary Tree



What are the other properties?

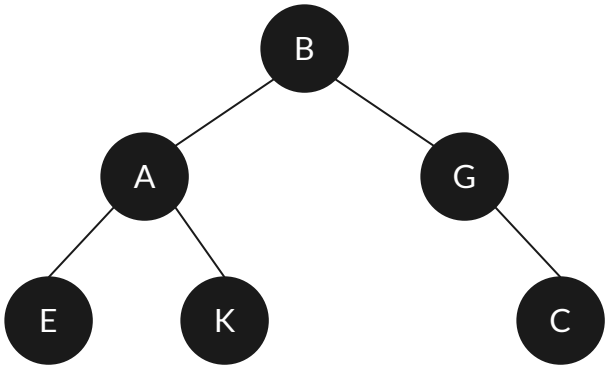
If a node is at index: *i*,

- The left child is at: *2i*
- The right child is at: *2i + 1*
- Its parent is at: i // 2 (floor value)

- What if the rule is not followed (example, if we remove L)?
- Not any more a Complete Binary Tree
- If we want to move C to left, the Tree Data structure gets broken

B	A	G	E	K		C

Not a Complete Binary Tree



What are the other properties?

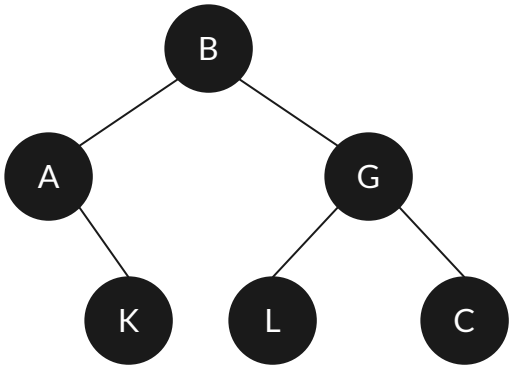
If a node is at index: *i*,

- The left child is at: *2i*
- The right child is at: *2i + 1*
- Its parent is at: i // 2 (floor value)

- What if the rule is not followed (example, if we remove L)?
- Not any more a Complete Binary Tree
- If we want to move C to left, the Tree Data structure gets broken

B	A	G	E	K		C

Not a Complete Binary Tree



What are the other properties?

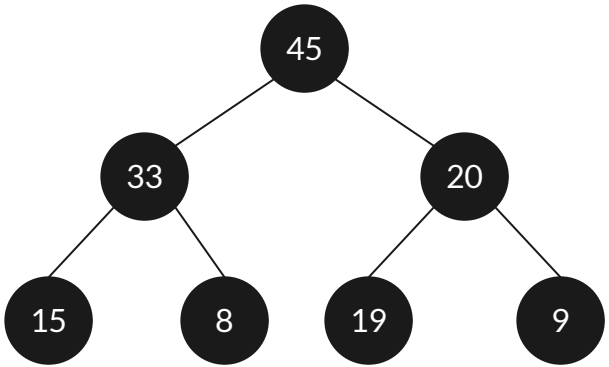
If a node is at index: *i*,

- The left child is at: *2i*
- The right child is at: *2i + 1*
- Its parent is at: i // 2 (floor value)

- What if the rule is not followed (example, if we removed L)
- Not any more a Complete Binary Tree
- If we want to move C to left, the Tree Data structure gets broken
- Same if any other removal is performed, say E

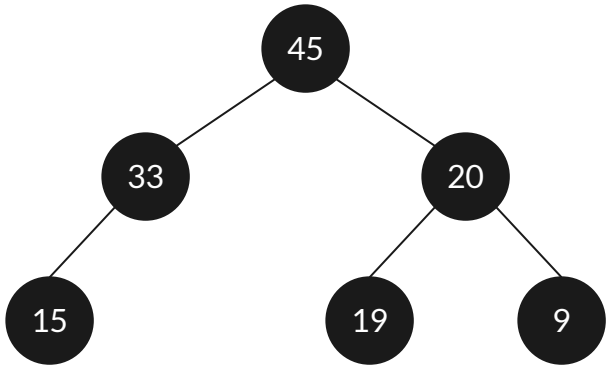
B	A	G		K	L	C

Heap



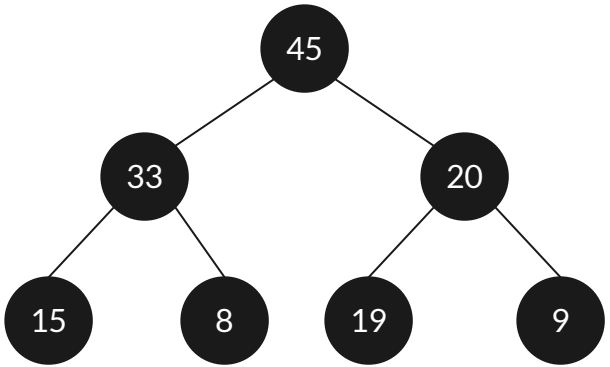
- Heap is a Complete Binary Tree
- Two different formats depending on the ordering preference
  - Max Heap
  - Min Heap
- Left is an example of a Max Heap

Heap



- Heap is a Complete Binary Tree
- Two different formats depending on the ordering preference
  - Max Heap
  - Min Heap
- Left is not an example of a Heap (because it is not a Complete Binary Tree)

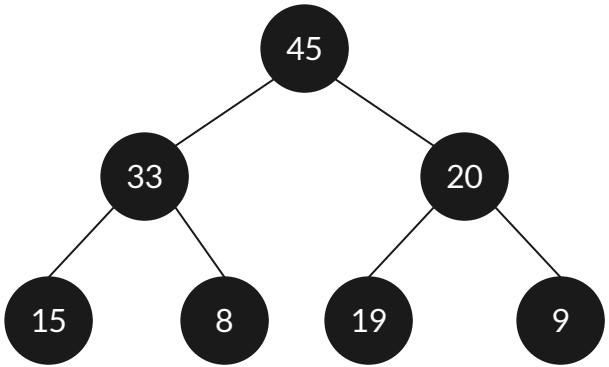
Insertion in a Heap



- Left to right tracking
- First availability at the very last level (all upper levels must be complete to be a valid Complete Binary Tree)

45	33	20	15	8	19	9

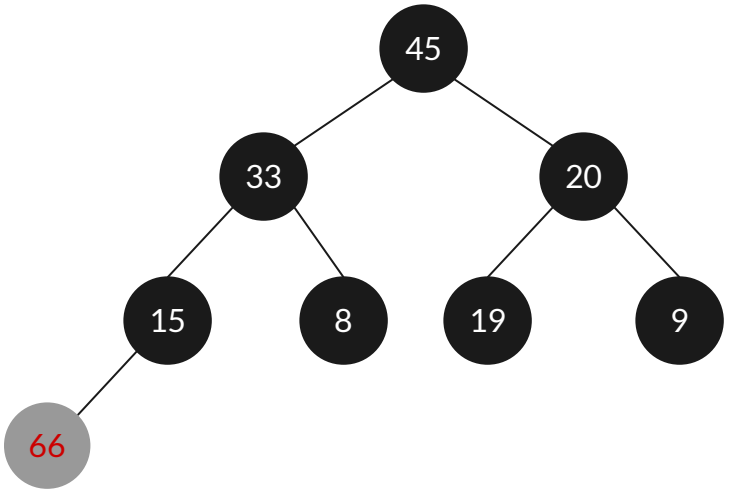
Insertion in a Heap



- Left to right tracking
- First availability at the very last level (all upper levels must be complete to be a valid Complete Binary Tree)
- We are inserting 66

45	33	20	15	8	19	9

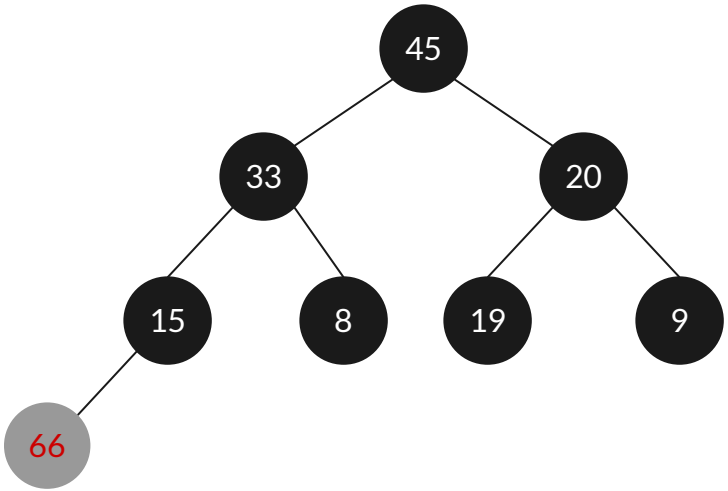
Insertion in a Heap



- Left to right tracking
- First availability at the very last level (all upper levels must be complete to be a valid Complete Binary Tree); end of the array - we can compute the parent
- We are inserting 66

45	33	20	15	8	19	9	66

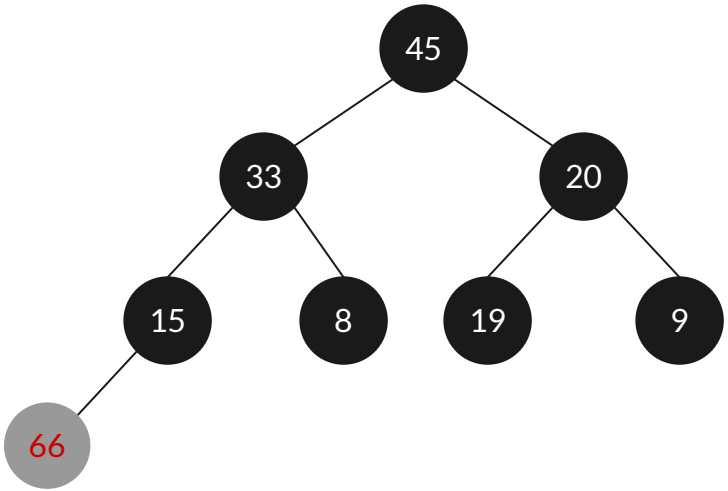
Insertion in a Heap



- Left to right tracking
- First availability at the very last level (all upper levels must be complete)
- We are inserting 66
- Still a Complete BT, but not a Heap (Max)

45	33	20	15	8	19	9	66

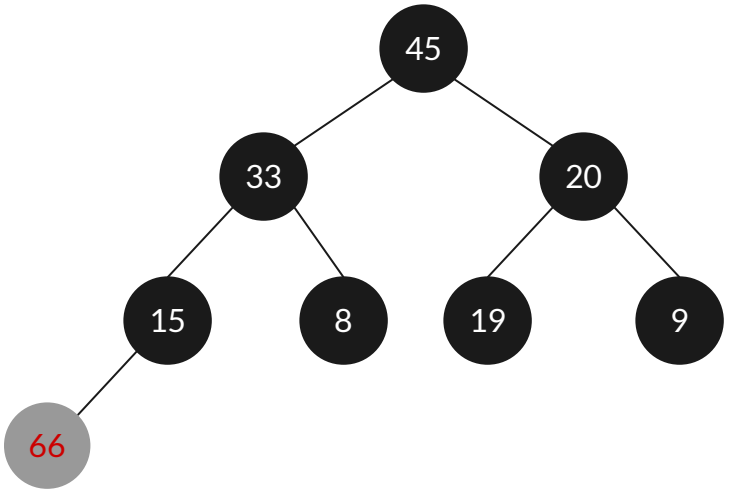
Insertion in a Heap



- Left to right tracking
- First availability at the very last level (all upper levels must be complete)
- We are inserting 66
- Still a Complete BT, but not a Heap (Max)
- We have to re-adjust

45	33	20	15	8	19	9	66

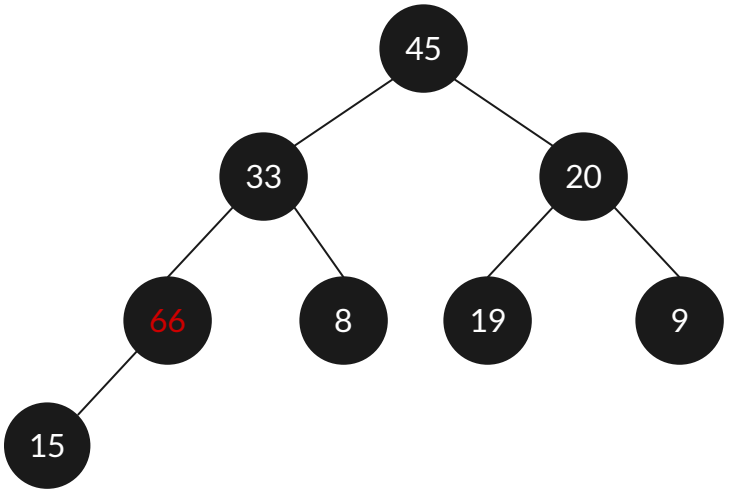
Insertion in a Heap



- Left to right tracking
- First availability at the very last level (all upper levels must be complete)
- We are inserting 66
- Still a Complete BT, but not a Heap (Max)
- We have to re-adjust
- Compare and propagate upwards

45	33	20	15	8	19	9	66

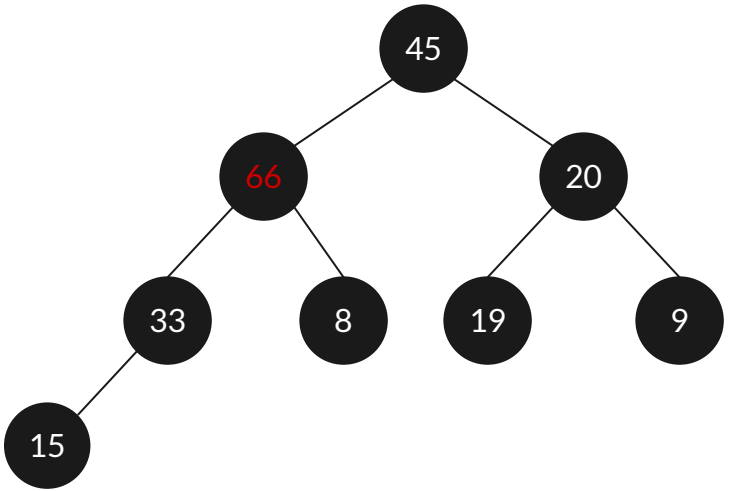
Insertion in a Heap



- Left to right tracking
- First availability at the very last level (all upper levels must be complete)
- We are inserting 66
- Still a Complete BT, but not a Heap (Max)
- We have to re-adjust
- Compare and propagate upwards

45	33	20	66	8	19	9	15

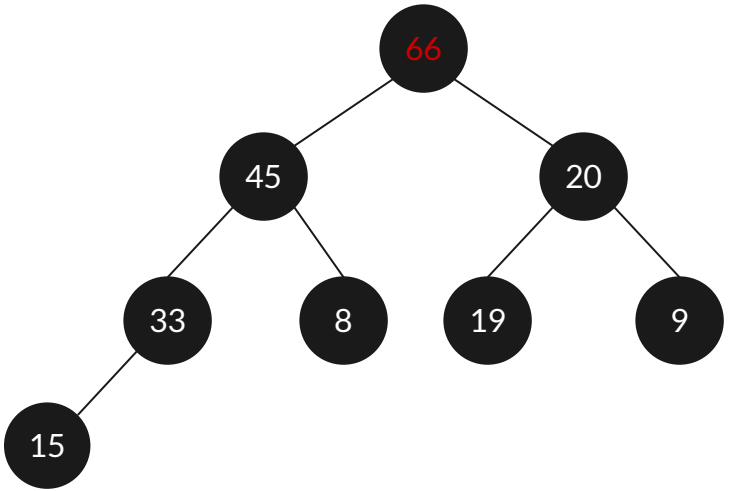
Insertion in a Heap



- Left to right tracking
- First availability at the very last level (all upper levels must be complete)
- We are inserting 66
- Still a Complete BT, but not a Heap (Max)
- We have to re-adjust
- Compare and propagate upwards

45	66	20	33	8	19	9	15

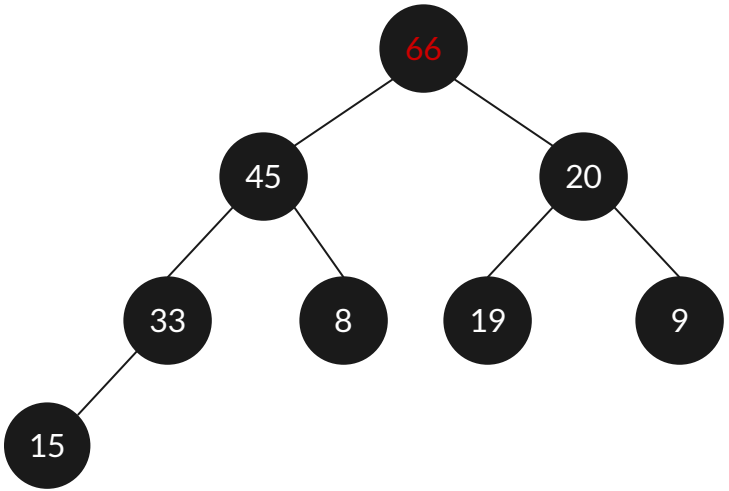
Insertion in a Heap



- Left to right tracking
- First availability at the very last level (all upper levels must be complete)
- We are inserting 66
- Still a Complete BT, but not a Heap (Max)
- We have to re-adjust
- Compare and propagate upwards

66	45	20	33	8	19	9	15

Insertion in a Heap

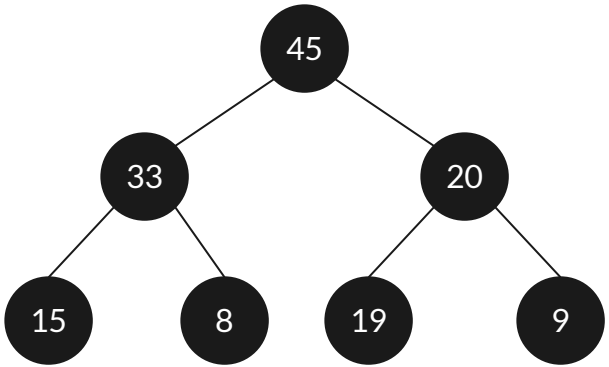


- Left to right tracking
- First availability at the very last level (all upper levels must be complete)
- We are inserting 66
- Still a Complete BT, but not a Heap (Max)
- We have to re-adjust
- Compare and propagate upwards

66	45	20	33	8	19	9	15

Deletion

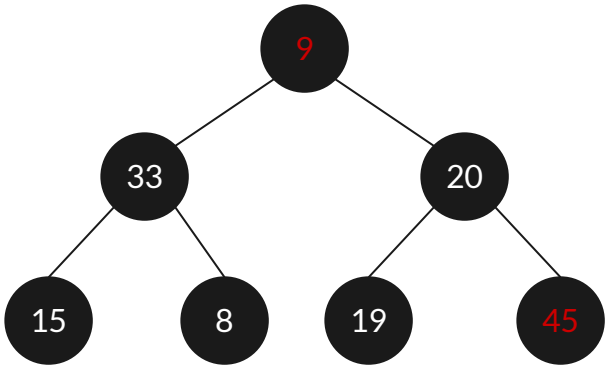
Deletion in a Heap



- We are only allowed to delete the root
- Swap values with the right-most leaf
- Delete the right-most leaf
- Re-adjust values:
  - First compare the two child, and swap with the highest one

45	33	20	15	8	19	9

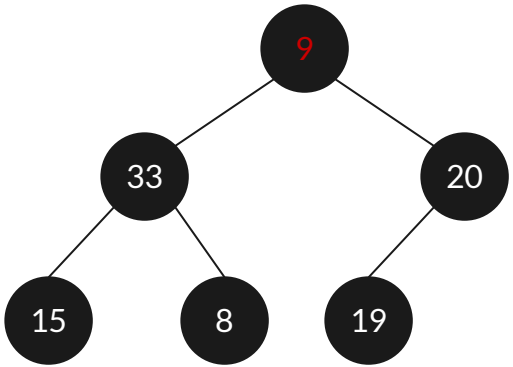
Deletion in a Heap



- We are only allowed to delete the root
- Swap values with the right-most leaf
- Delete the right-most leaf
- Re-adjust values:
  - First compare the two child, and swap with the highest one

45	33	20	15	8	19	9

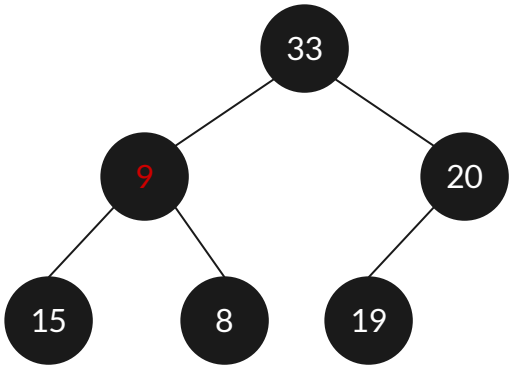
Deletion in a Heap



- We are only allowed to delete the root
- Swap values with the right-most leaf
- Delete the right-most leaf
- Re-adjust values:
  - First compare the two child, and swap with the highest one

9	33	20	15	8	19	

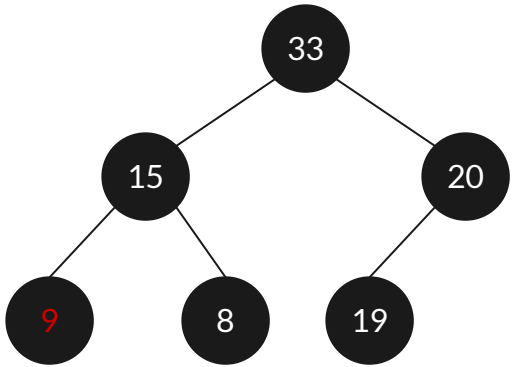
Deletion in a Heap



- We are only allowed to delete the root
- Swap values with the right-most leaf
- Delete the right-most leaf
- Re-adjust values:
  - First compare the two child, and swap with the highest one
  - iterate

33	9	20	15	8	19	

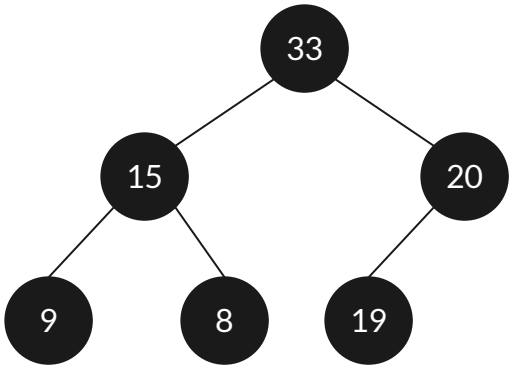
Deletion in a Heap



- We are only allowed to delete the root
- Swap values with the right-most leaf
- Delete the right-most leaf
- Re-adjust values:
  - First compare the two child, and swap with the highest one
  - iterate

33	15	20	9	8	19	

Deletion in a Heap



- We are only allowed to delete the root
- Swap values with the right-most leaf
- Delete the right-most leaf
- Re-adjust values:
  - First compare the two child, and swap with the highest one
  - iterate

33	15	20	9	8	19	

Heap Sort

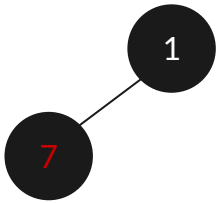
- First build a Max Heap
- Then delete one element at a time

Build a Heap

Build the Heap

1	7	0	5	10		

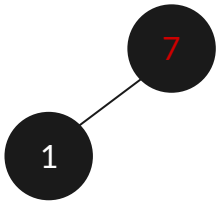
Build a Heap



Build the Heap

1	7	0	5	10		

Build a Heap

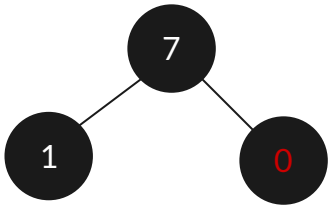


Build the Heap

Swap (1, 7)

7	1	0	5	10		

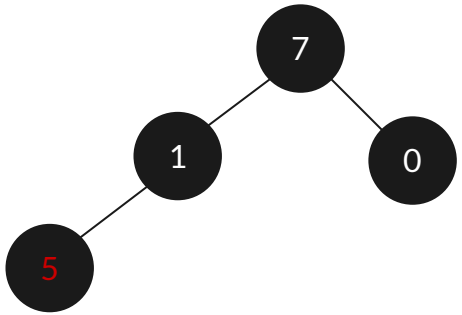
Build a Heap



Build the Heap

7	1	0	5	10		

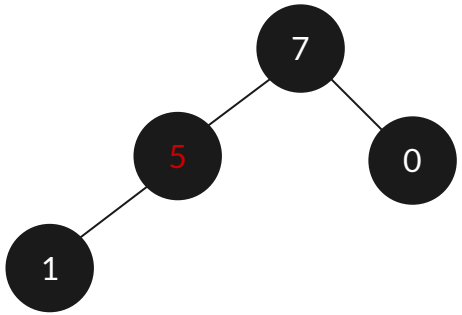
Build a Heap



Build the Heap

7	1	0	5	10		

Build a Heap

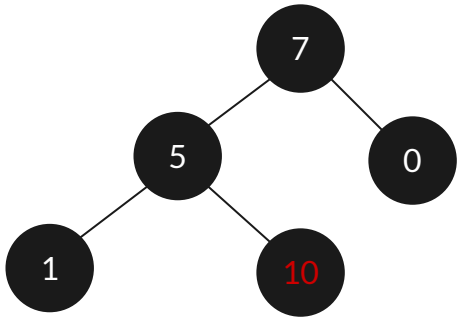


Build the Heap

Swap (5, 1)

7	5	0	1	10		

Build a Heap

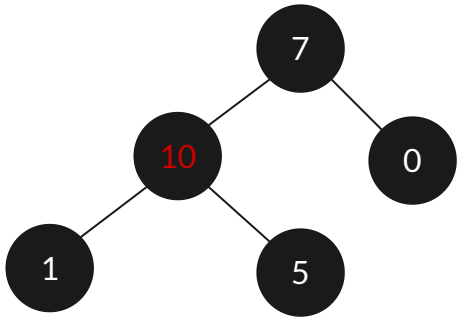


Build the Heap

Add 10

7	5	0	1	10		

Build a Heap

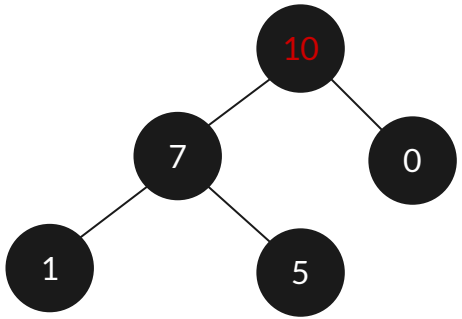


Build the Heap

Swap

7	10	0	1	5		

Build a Heap

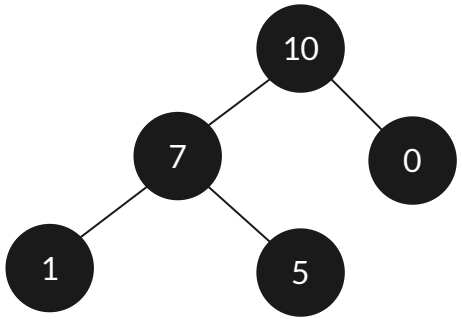


Build the Heap

Swap (7, 10)

10	7	0	1	5		

Build a Heap

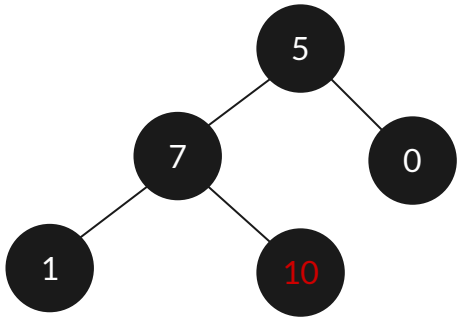


Build the Heap

Complete Max Heap

10	7	0	1	5		

Delete one at a step

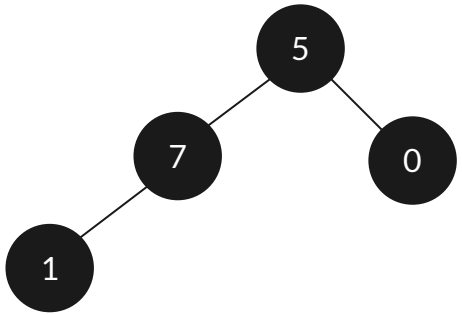


Now Apply Deletion

Swap (10, 5)

5	7	0	1	10		

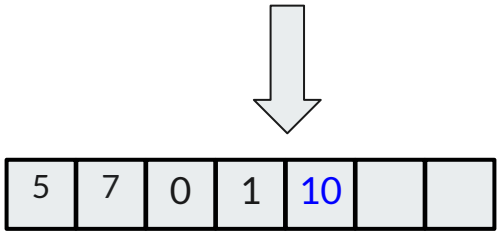
Delete one at a step



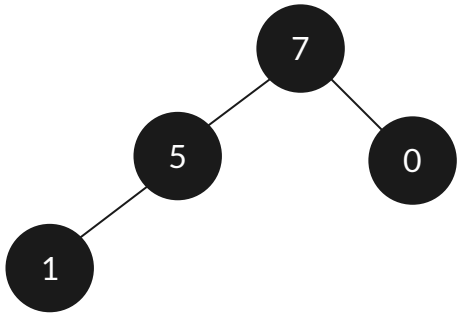
Now Apply Deletion

Remove 10

Heap ends here



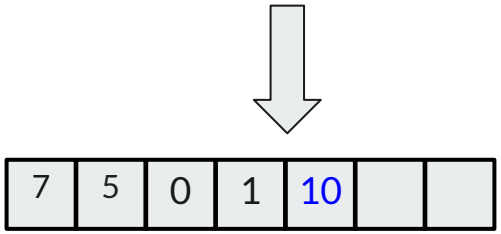
Delete one at a step



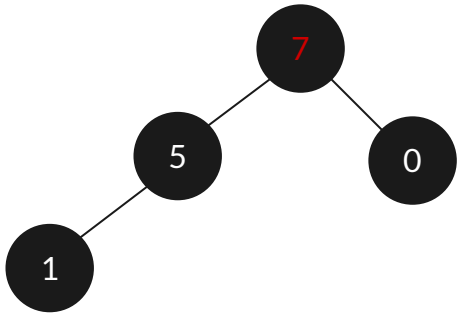
Now Apply Deletion

Rearrange (swap 5, 7)

Heap ends here



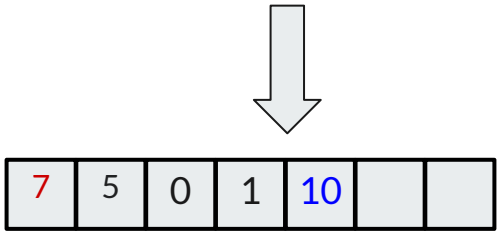
Delete one at a step



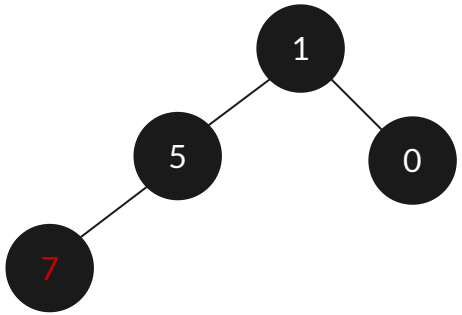
Now Apply Deletion

Remove 7 (Swap 1, 7)

Heap ends here



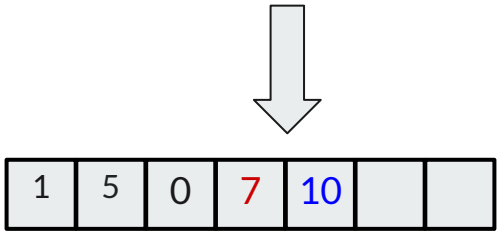
Delete one at a step



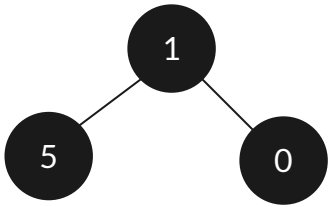
Now Apply Deletion

Remove 7 (Swap 1, 7)

Heap ends here



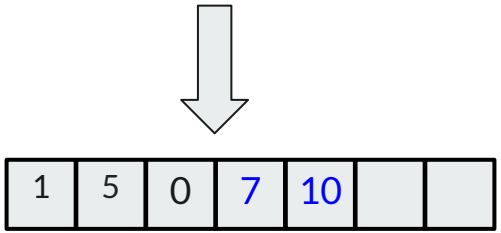
Delete one at a step



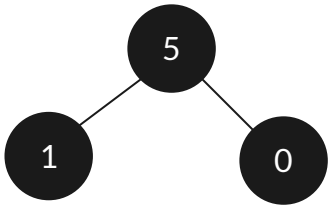
Now Apply Deletion

Remove 7

Heap ends here



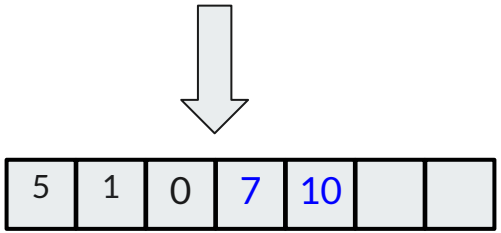
Delete one at a step



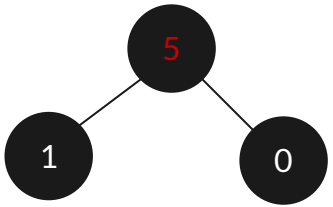
Now Apply Deletion

Readjust (swap 1, 5)

Heap ends here



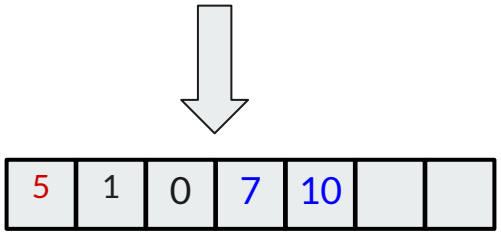
Delete one at a step



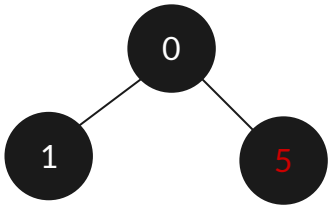
Now Apply Deletion

Remove 5 (swap 5, 0)

Heap ends here



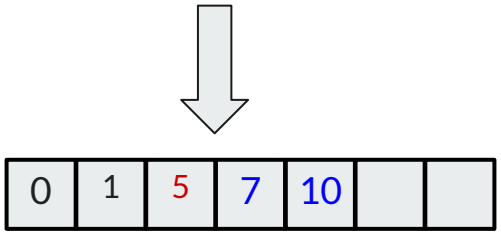
Delete one at a step



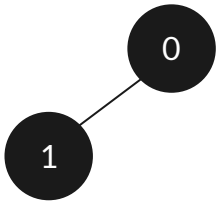
Now Apply Deletion

Remove 5 (swap 5, 0)

Heap ends here



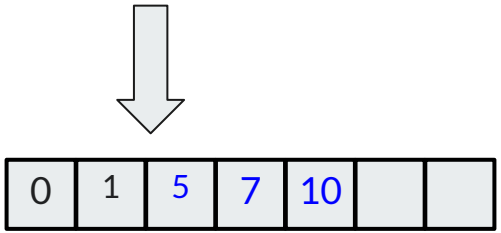
Delete one at a step



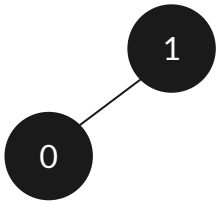
Now Apply Deletion

Remove 5

Heap ends here



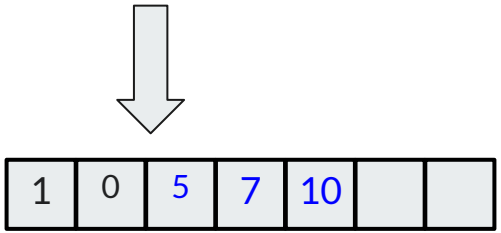
Delete one at a step



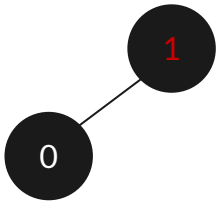
Now Apply Deletion

Readjust (swap 0, 1)

Heap ends here



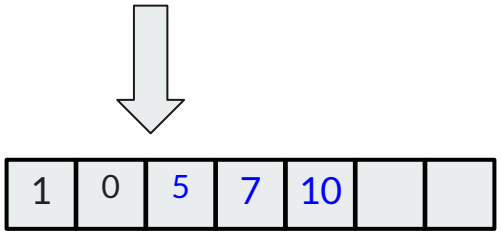
Delete one at a step



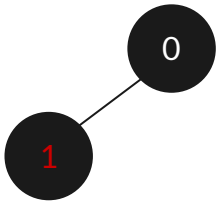
Now Apply Deletion

Delete 1

Heap ends here



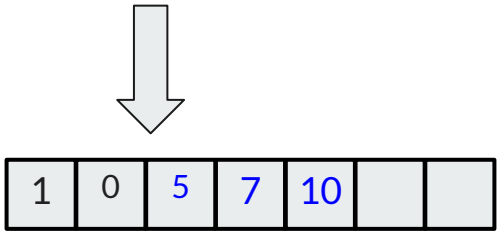
Delete one at a step



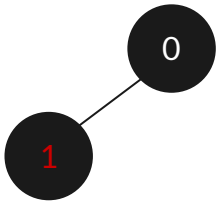
Now Apply Deletion

Delete 1 (swap 1, 0)

Heap ends here



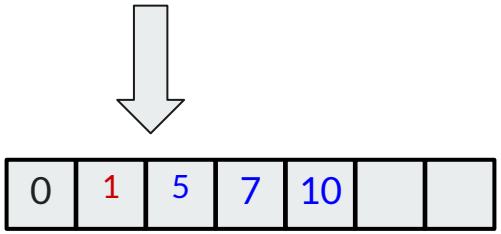
Delete one at a step



Now Apply Deletion

Delete 1 (swap 1, 0)

Heap ends here



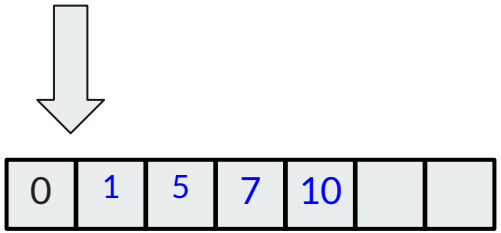
Delete one at a step

0

Now Apply Deletion

Delete 1

Heap ends here

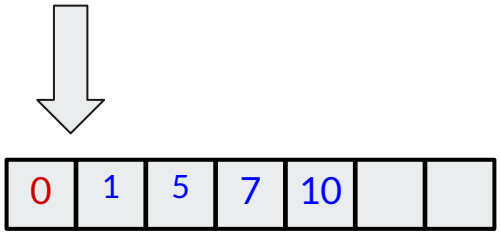


Delete one at a step

Now Apply Deletion

Delete 0

Heap ends here

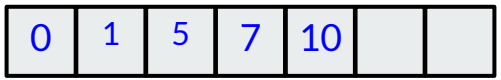


Delete one at a step

Deletion complete

Delete 0

Sorted array
(Heap sort)



QA