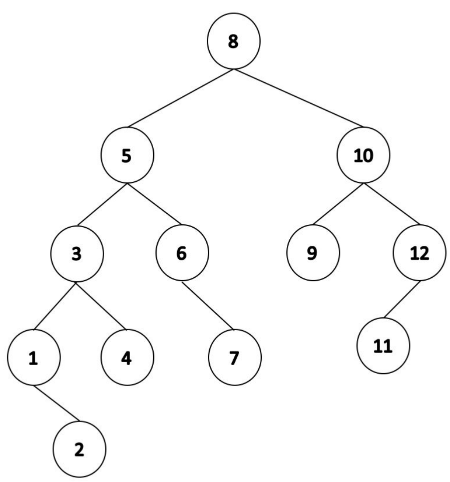
CS 225 Spring 2019 :: TA Lecture Notes
3/1 AVL

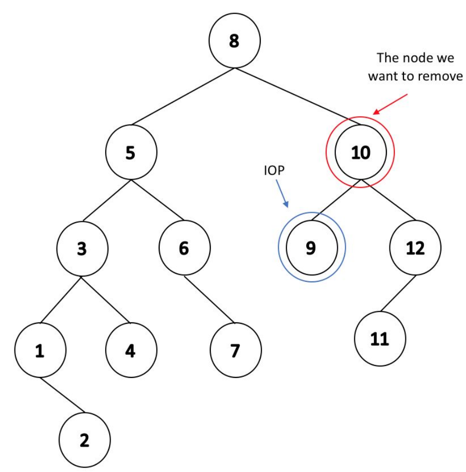
By Wenjie

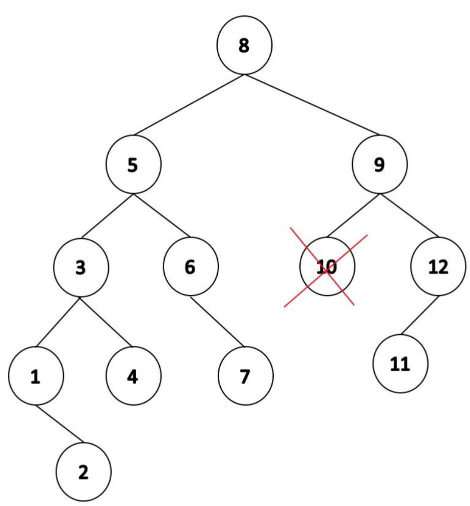
AVL Remove

- 2 children: swap with IOP then remove
- 1 child: swap with child then remove
- No child: remove
- We remove the same way we would remove a node in a BST.

Eg: remove (10)



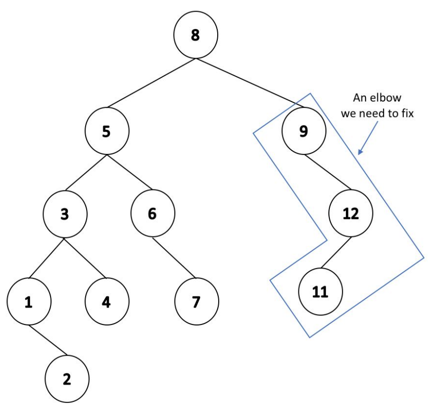


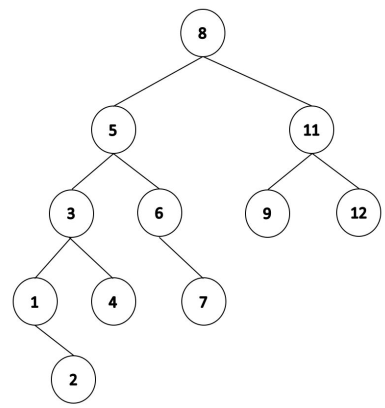


CS 225 Spring 2019 :: TA Lecture Notes
3/1 AVL

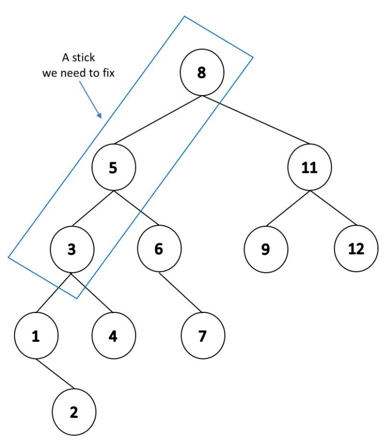
By Wenjie

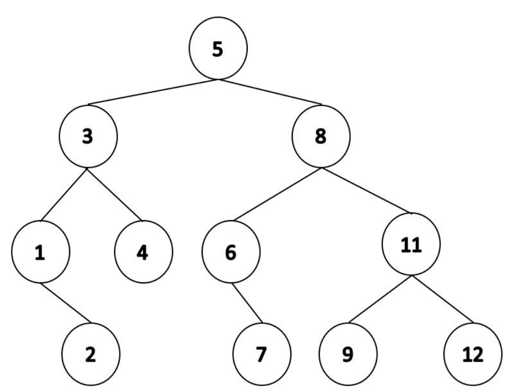
- After we removed the node, we can see that we are left with an elbow with the lowest point of imbalance at 9. We know how to fix a case where b(9) = 2 and b(12) = -1 (RL rotation).





- We have corrected the imbalance at 9, but is the tree as a whole balanced?
    - If we check b(8), we will see that it is two. We corrected one imbalance, but we have created another one. Again we know how to fix this: b(8) = -2 and b(5) = -1 (rotate to the right)





CS 225 Spring 2019 :: TA Lecture Notes
3/1 AVL

By Wenjie

AVL summary

- _find(...) → O(h) + 0 rotation
- _insert(...) → O(h) + up to 1 rotation
- _remove(...) → O(h) + up to h rotations
  - Each rotation is O(1).
  - Doing h rotations is h * O(1) = O(h)
  - O(h) + O(h) = 2 * O(h) = O(h)

where O(h) = O(logN)

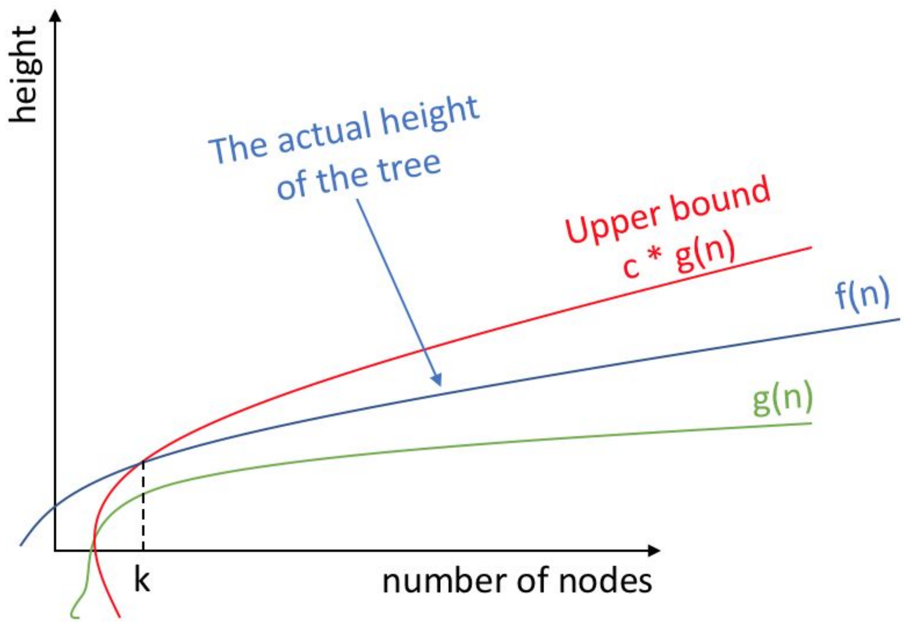
Big-O definition

If we say a function is a Big O of another ie f(n) = O(g(n)) if and only if there exists some constants variable c,k such that f(n) ≤ c * g(n) and for all n > k.

- In another works, c*g(n) is the upper bound of the f(n) and f(n) is always below the upper bound
- For all trees that are at least k nodes big, we know that the height h of that tree will be less than c * g(h). The definition implies that small values of k don't matter.

CS 225 Spring 2019 :: TA Lecture Notes
3/1 AVL

By Wenjie

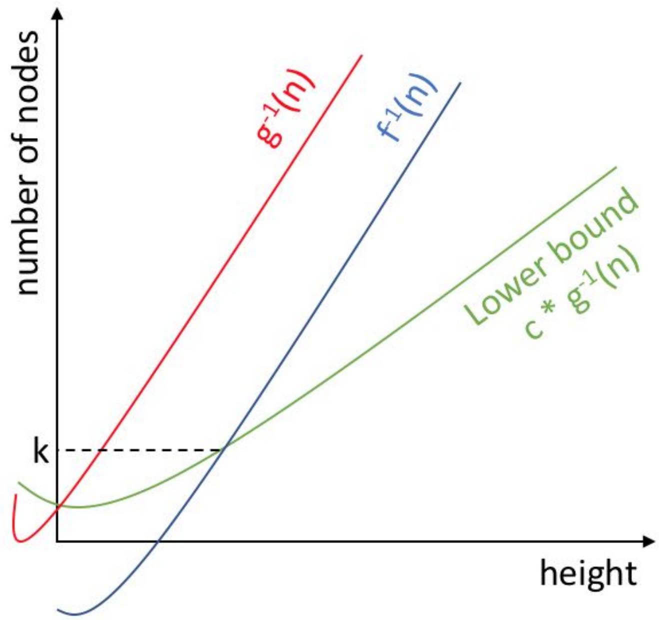


However the definition above is hard to prove the maximal value of the height and we should invert the definition above.
The inverted definition of above definition:

- For all integer n and some int k such that n > k, we have $n > c * g^{-1}(h)$.
- In other words, give the tree of height h, what is the minimal number of nodes of that tree.
- There is a unique representation of a tree with minimum number of nodes.

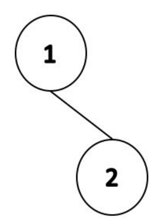
CS 225 Spring 2019 :: TA Lecture Notes
3/1 AVL

By Wenjie



- The representation of the function that has the smallest number of nodes in an AVL tree of height h → N(h):

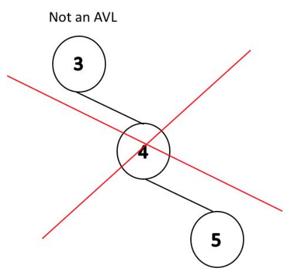
N(h = -1) = 0 N(h = 0) = 1 N(h = 1) = 2

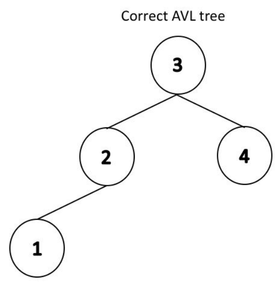


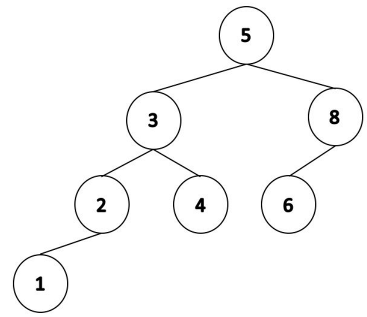
N(h = 2) = 4 N(h = 3) = 7

CS 225 Spring 2019 :: TA Lecture Notes
3/1 AVL

By Wenjie







A minimal AVL tree is going to have a balance of -1.

- Looking at these couple of cases, we can notice a recursive relation:
  - **$\underline{N(h) = 1 + N(h - 1) + N(h - 2)}$**

    left-side right-side
- Simplify the above function
  - $N(h) > N(h - 1) + N(h - 2)$ → we can drop a constant, but now N(h) is greater and not equal anymore.
  - We know that $N(h - 1) > N(h - 2)$ → because AVL min tree has balance of -1, we know that left side is longer than the right side.
  - $N(h) > 2 * N(h - 2)$ → we can't drop N(h - 1), but above we concluded it is larger than N(h - 2), this new form is correct.
  - The last step would be to find closed form which is $2^{h/2}$ and the following are the steps to get the closed form:

Theorem: An AVL tree of height h has at least $2^{h/2}$ nodes for h > 0 → $N(h) > 2^{h/2}$.

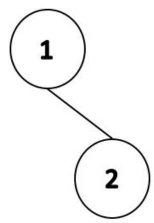
Proof:

- Consider an AVL tree and let h denote it's height
- Base case: h = 1 and h = 2

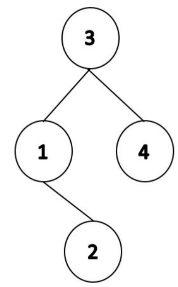
AVL	Thm	AVL	Thm
h = 1	$2^{1/2} = \sqrt{2} \cong 1.4$	h = 2	$2^{2/2} = 2$

CS 225 Spring 2019 :: TA Lecture Notes
3/1 AVL

By Wenjie



An AVL tree of height 2
has at least 2 nodes



An AVL tree of height 3
has at least 3 nodes

Inductive hypothesis: for $h > 2$, $\forall\ j < h$, $N(j) > 2^{j/2}$.
We want to show: $N(h) = 1 + N(h - 1) + N(h - 2)$

$$> 2 * N(h - 2)$$
$$> 2 * 2^{(h-2)/2}$$
$$> 2^{h/2}$$

Therefore an AVL tree of height h has at least $2^{h/2}$ nodes.

We have proved that $n \geq N(h) > 2^{h/2} \rightarrow n > 2^{h/2}$
Now we invert back: $h < 2\log(n)$ QED

Note: This theorem will give a very loose bound as a result of our lower bounding of the formula. But if we want to calculate N(h), we need to calculate using the recursive formula:

$N(h) = 1 + N(h - 1) + N(h - 2)$,

which is a more precise bound.