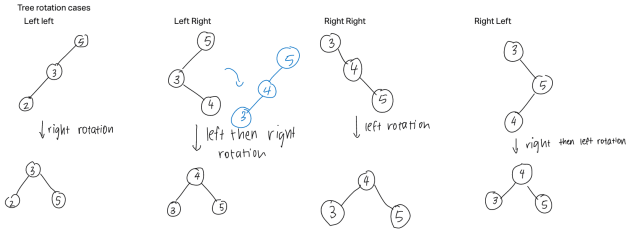
AVL Trees

- A self-balancing binary search tree
- Unbalanced height: *|height(left) - height(right)| > 1*
  - Height of each node can be stored
- Average and worst case search time complexity is of O(log(n))

```c
typedef struct Node *Link; // Links are pointers to nodes
typedef struct Node *Tree; // a Tree is a pointer to its root node

struct Node { int data; int height; Link left; Link right; };
```

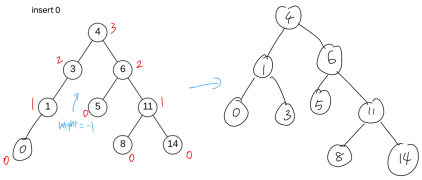
Tree rotation cases

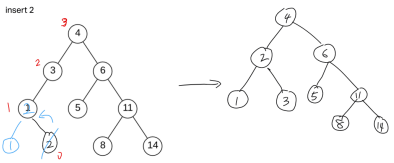


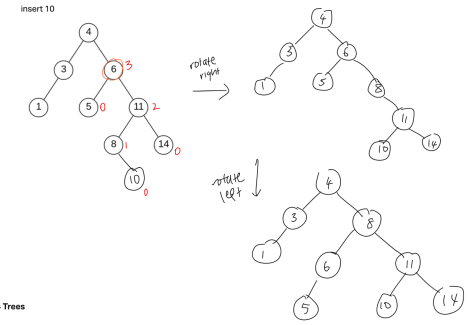
Inserting into AVL:

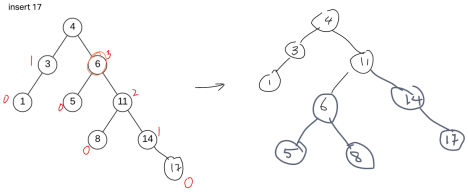
1. Insert the node and update heights
2. Check if heights are balanced
3. Use one of the 4 rotations if unbalanced

1. Annotate each node with the height of its subtree, and insert the integers 0, 2, 10, 17.





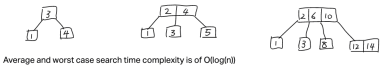




2-3-4 Trees

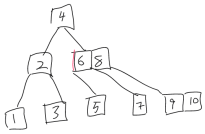
```c
typedef struct Node *Link; // Links are pointers to nodes
typedef struct Node *Tree; // a Tree is a pointer to its root node

struct Node { int order; int data[3]; Link child[4]; };
```

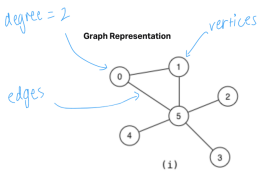
- Can have at a node:
  - 1 item + 2 children
  - 2 items + 3 children
  - 3 items + 4 children

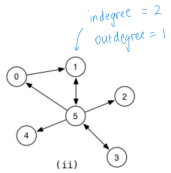


- Average and worst case search time complexity is of O(log(n))

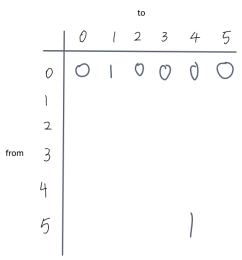
Insert 1 2 3 4 5 8 6 7 9 10 into an empty tree.







Adjacency Matrix Representation



	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	0	0
3	0	0	0	0	0	1
4	0	0	0	0	0	0
5	1	1	1	1	1	0

Adjacency List Representation