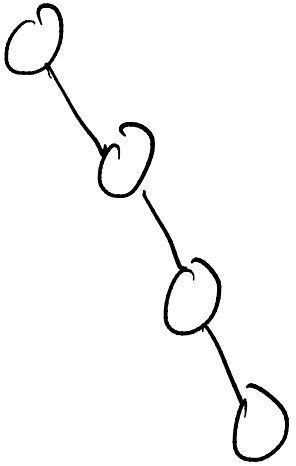
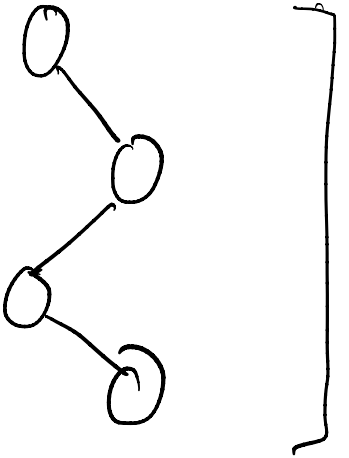
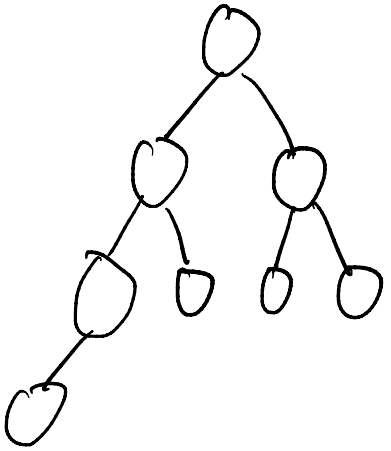
Binary Trees





height = 3

height is $O(n)$, max height is $n-1$



n	min h
4	2
5	2
6	2
7	2
8	3

min height is $O(\lg n)$

$\lfloor \log_2(n) \rfloor$

Binary heap - a child's key is $\leq$ its parent's key

Binary search tree - the key of a left child $\leq$ the key of the parent, key of right child $>$ key of the parent

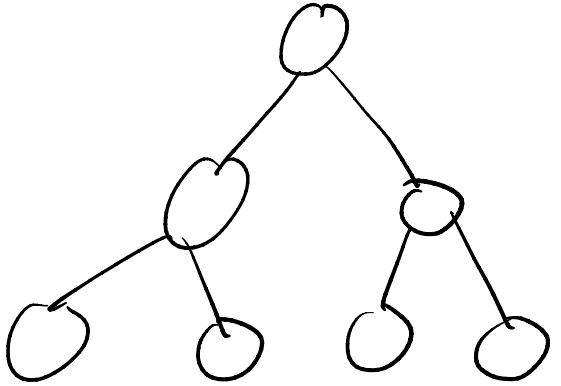
Red-black tree - Self-balancing, performs rotations when the height grows too high, guarantees the height is at most $2 \cdot \lg(n+1)$ (still $O(\lg n)$)

"Normal" binary search tree - no guarantee about height, but if keys are random the expected height is $O(\lg n)$

Binary heap - guaranteed to have the minimum possible height

Searching in a BST is $O(h)$

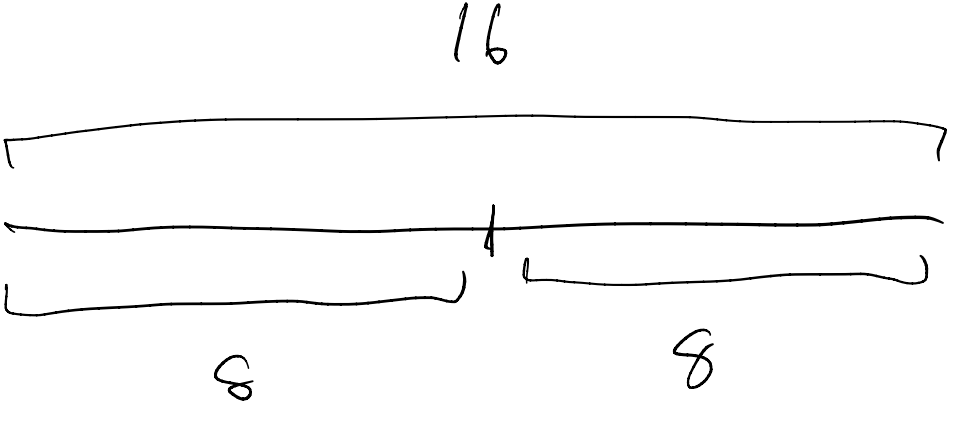
Recursively traversing a binary tree is depth-first search



Dynamic programming

- Solves problems by combining solutions to subproblems
- Subproblems may overlap
    - Solutions to already-solved subproblems are stored
- If doing it recursively, the storage strategy is called memoization
    - Storage can't be stack-allocated, because different recursive calls need to access the same data

Optimal substructure - an optimal solution to a problem contains within it optimal solutions to subproblems



For both DP and greedy strategies, the problems must exhibit optimal substructure

Greedy-choice property - we can assemble a globally optimal solution by making locally optimal choices

We need both optimal substructure and greedy-choice property to hold to use a greedy solution