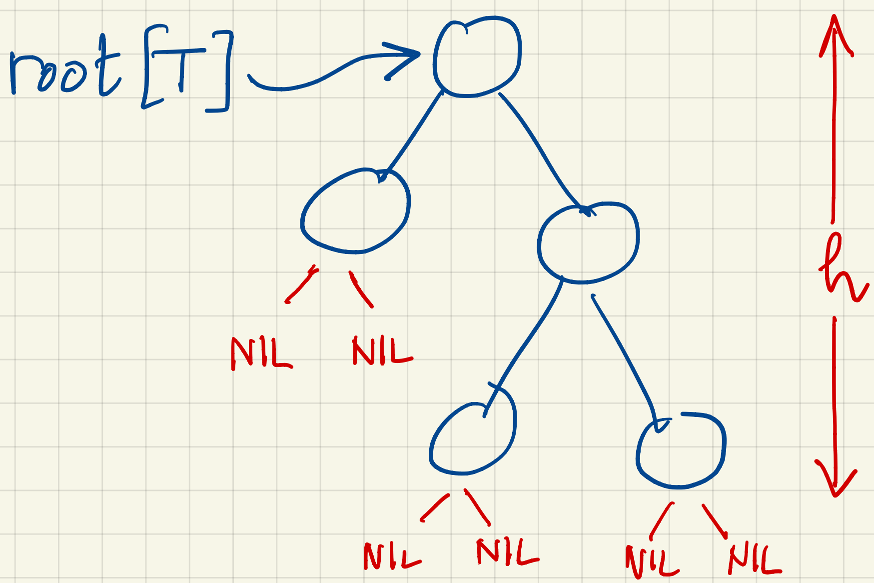
Binary Search Tree

Dynamic set operations such as insert, delete, search and more can be done in $O(h)$ time, where h is height of tree.



each node x has:

$key[x]$

$left[x]$

$right[x]$

$p[x]$

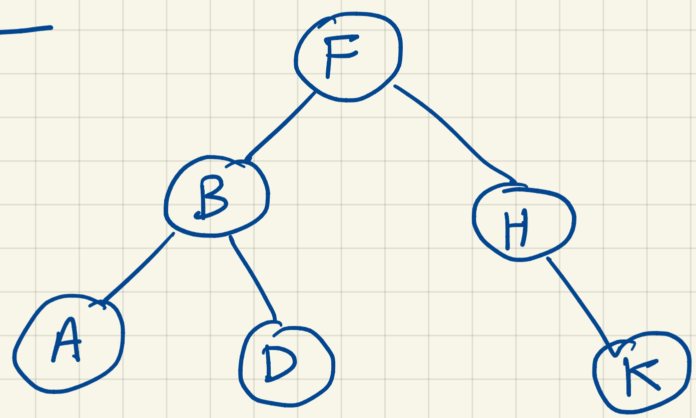
$$p[root[T]] = NIL$$

Binary Search Tree property

$y \in$ left subtree of $x \Rightarrow key[y] \leq key[x]$

$y \in$ right subtree of $x \Rightarrow key[y] \geq key[x]$

Example:



(It's ok to have duplicates too)

Searching and Inserting

Tree-Search (x, k)

    if $x =$ NIL or $k = key[x]$
        then return x
    if $k < key[x]$
        then return Tree-Search$(left[x], k)$
        else return Tree-Search$(right[x], k)$

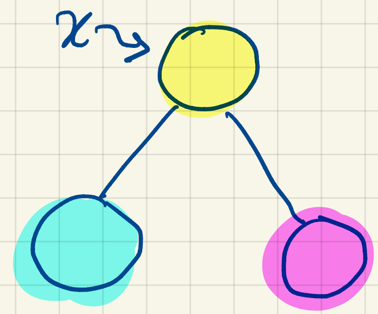
Initial call is for Tree-Search$(root[T], k)$

Tree-Insert(T, x)

- similar code
- Insert x in place of NIL (where it should have been)
- Use trailing pointer to keep track where you came from (see book)

Sorting with BST

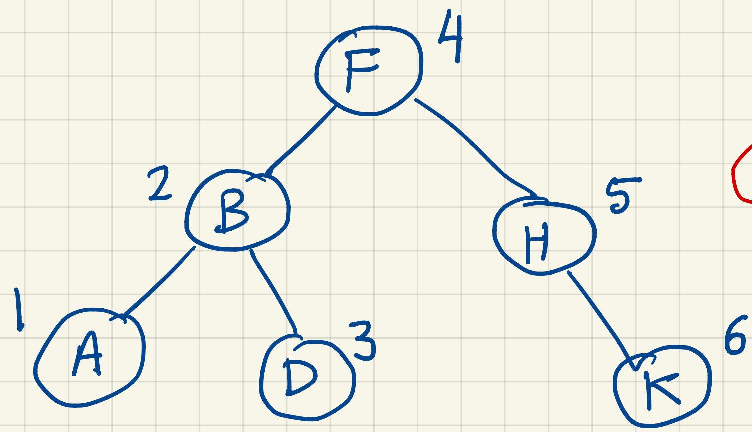
First consider the inorder tree walk




```
Inorder-Tree-Walk (x)
    if x ≠ NIL
        then Inorder-tree_walk(left[x])
             print key[x]
             Inorder-Tree-walk (right[x])
```

Inorder:
- visit left[x]
- process x
- visit right[x]

Inorder-Tree-Walk(root[T]) will print keys in order



A B D F H K

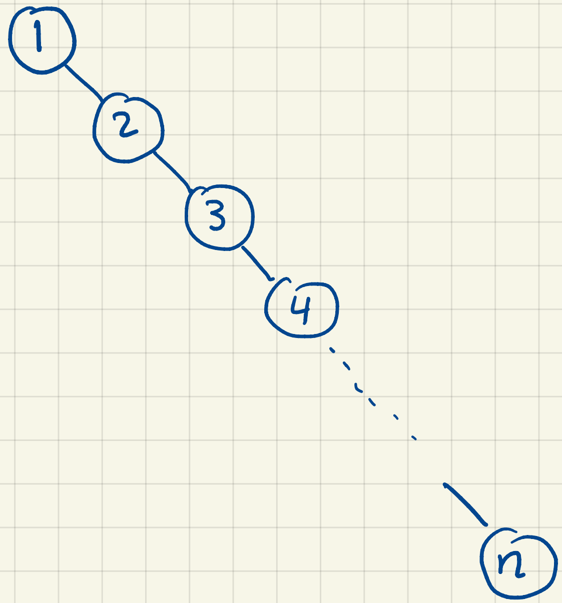
```
BST-Sort(A, n)
  for i ← 1 to n
      do x ← node such that key[x] = A[i]
         Tree-Insert(T, x)
  Inorder-Tree-Walk(root[T])
```

What can we say about this algorithm?

- $O(n^2)$ [mainly due to inserts when tree is bad, $h = O(n)$]
- $\Omega(n \log n)$ [It's comparison based]
- It's $O(n \log n)$ on average, similar to Quicksort.

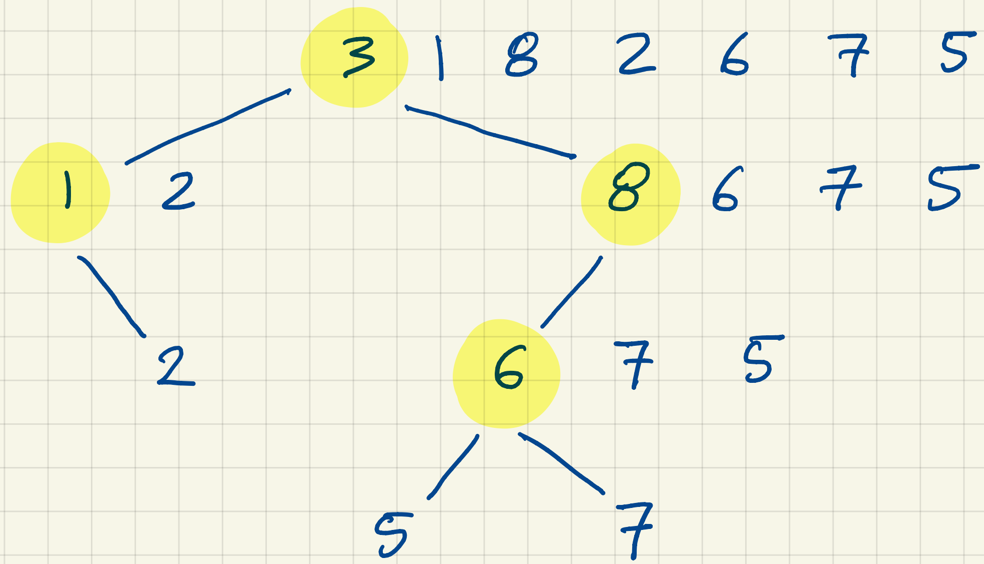
: BST-sort and a stable partition Quicksort with first element as pivot, make exactly the same comparisons (but in different order)

$A = [1, 2, 3, 4, \ldots, n]$



"Bad" tree

Example:



Quicksort comparisons:

- Everyone with 3
- 2 to 1
- $\{6, 7, 5\}$ to 8
- $\{7, 5\}$ to 6

BST comparisons: 1,3 ; 8,3 ; 2,3 ; 2,1 ; 6,3 ; 6,8

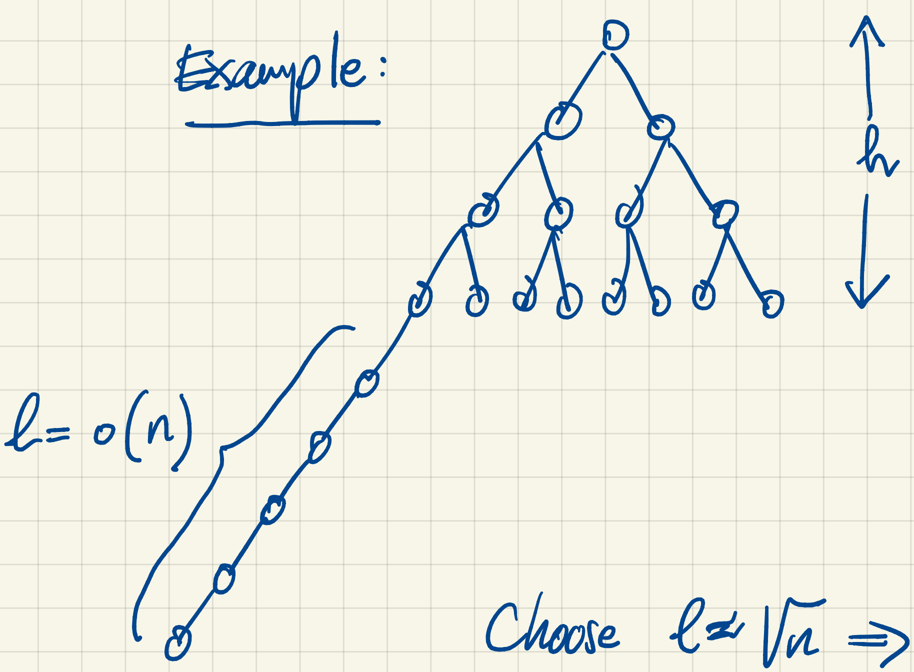
7,3 ; 7,8 ; 7,6 ; 5,3 ; 5,8 ; 5,6

So BST-sort has an $O(n \log n)$ time on random input.

So what did we conclude ?

- A randomly built BST has expected sum of depths equal to $O(n \log n)$, so expected average depth is $O(\log n)$
- This does not mean expected height is $O(\log n)$, height is the maximum depth.

Example:



$n = 2^{h+1} - 1 + \ell \qquad h \approx O(\log n)$

$\text{Height} = h + \ell$

$\text{sum of depth} \approx \Theta\left(hn + \ell^2\right)$

$\text{average depth} \approx \Theta\left(h + \frac{\ell^2}{n}\right)$

Choose $\ell \approx \sqrt{n} \Rightarrow$ average depth is $\Theta(h) = \Theta(\log n)$

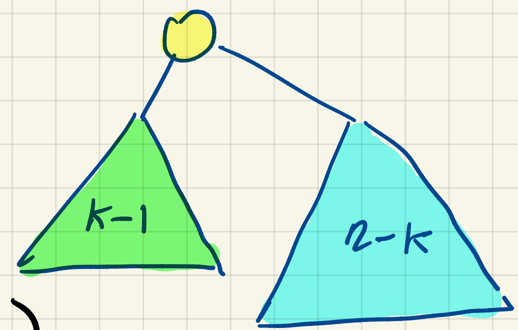
Height is $\Theta(\sqrt{n})$

- It would be great if expected height of randomly built BST is $O(\log n)$, then all operations on tree run in $O(\log n)$.

- In fact, it is!

Let X_n = height of random BST

$$X_n = 1 + \max(X_{k-1}, X_{n-k})$$



Assume $X_0 = -\infty$ ($Y_0 = 0$, below)

To make this look like recurrence we have seen before, let $Y_n = 2^{X_n}$, then

$$Y_n = 2^{1+\max(X_{k-1}, X_{n-k})} = 2 \cdot 2^{\max(X_{k-1}, X_{n-k})}$$

$$= 2 \cdot \max\left(2^{X_{k-1}}, 2^{X_{n-k}}\right) = 2\max(Y_{k-1}, Y_{n-k})$$

$$E[Y_n] = \frac{2}{n}\sum_{k=1}^{n} E[\max(Y_{k-1}, Y_{n-k})]$$

(left subtree equally likely to have $0, 1, \ldots, n-1$ nodes)

$$\Rightarrow \neq \max(E[Y_{k-1}], E[Y_{n-k}])$$

$$\leq \frac{2}{n}\sum_{k=1}^{n} E[Y_{k-1} + Y_{n-k}]$$

$$E[Y_n] \leq \frac{4}{n}\sum_{k=0}^{n-1} E[Y_k]$$

Guess $E[Y_n] \leq dn^3$

$$E[Y_n] \leq \frac{4d}{n}\sum_{k=0}^{n-1} k^3 \leq \frac{4d}{n}\int_0^n x^3\,dx = \frac{4d}{n}\left.\frac{x^4}{4}\right|_0^n = dn^3.$$

Now $E[Y_n] \leq dn^3$

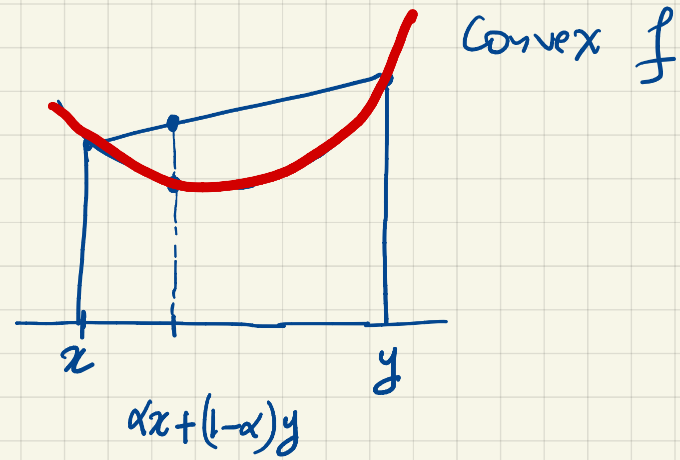
By Jensen's Inequality $E[X_n] = E[\log Y_n] \leq \log E[Y_n]$

$$E[X_n] \leq \log dn^3 = \log d + 3\log n = O(\log n)$$

Jensen's Inequality (Not hard to show)

If $f(x)$ is convex:

$$f(E[X]) \leqslant E[f(X)]$$



log is concave $\Rightarrow$

$$\log E[X] \geqslant E[\log X]$$

$$f(\alpha x+(1-\alpha)y) \leqslant \alpha f(x)+(1-\alpha)f(y)$$

So BST operations run in $O(\log n)$ on a randomly built BST. See book for

- Tree-Insert
- Tree-Delete
- Tree-Successor
- Tree-Predecessor
- Tree-Minimum
- Tree-Maximum