# Binary Search Tree

Autumn 2011

# Binary Search Trees

# Binary Search Trees

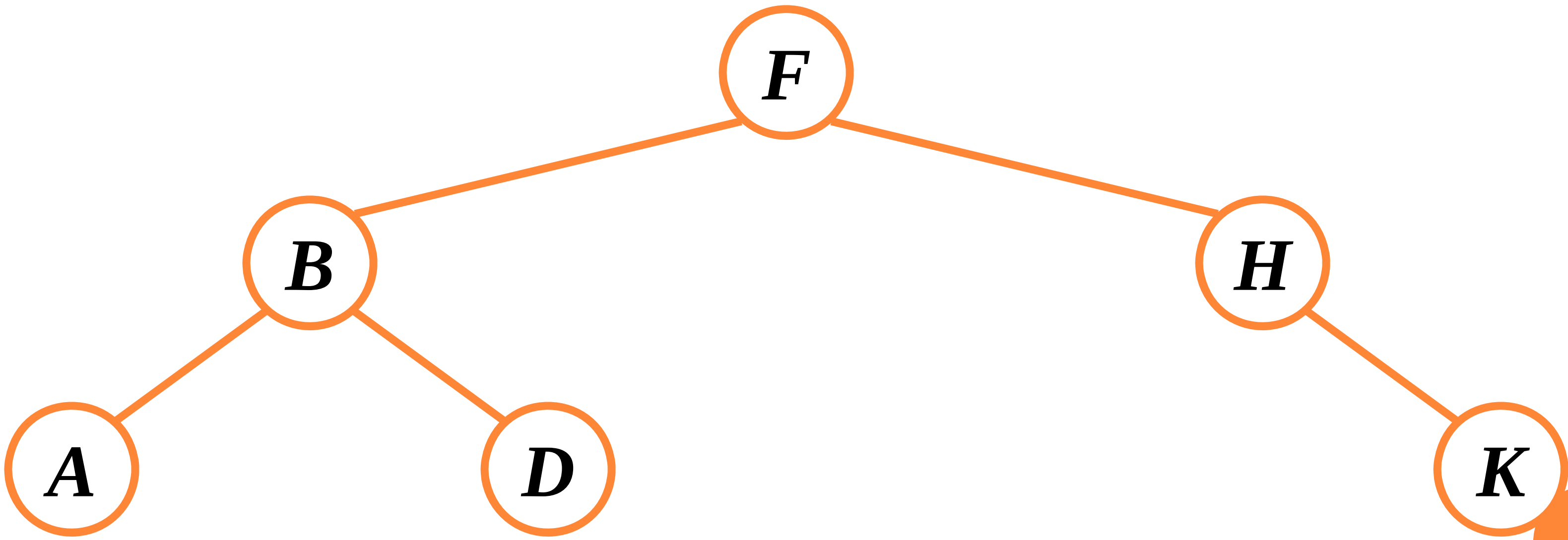
- *Binary Search Trees* (BSTs) are an important data structure for dynamic sets
- In addition to satellite data, elements have:
  - *key*: an identifying field inducing a total ordering
  - *left*: pointer to a left child (may be NULL)
  - *right*: pointer to a right child (may be NULL)
  - *p*: pointer to a parent node (NULL for root)

# Binary Search Trees

- BST property:

  key[left(x)] ≤ key[x] ≤ key[right(x)]

- Example:



# INORDER TREE WALK (TRAVERSAL)

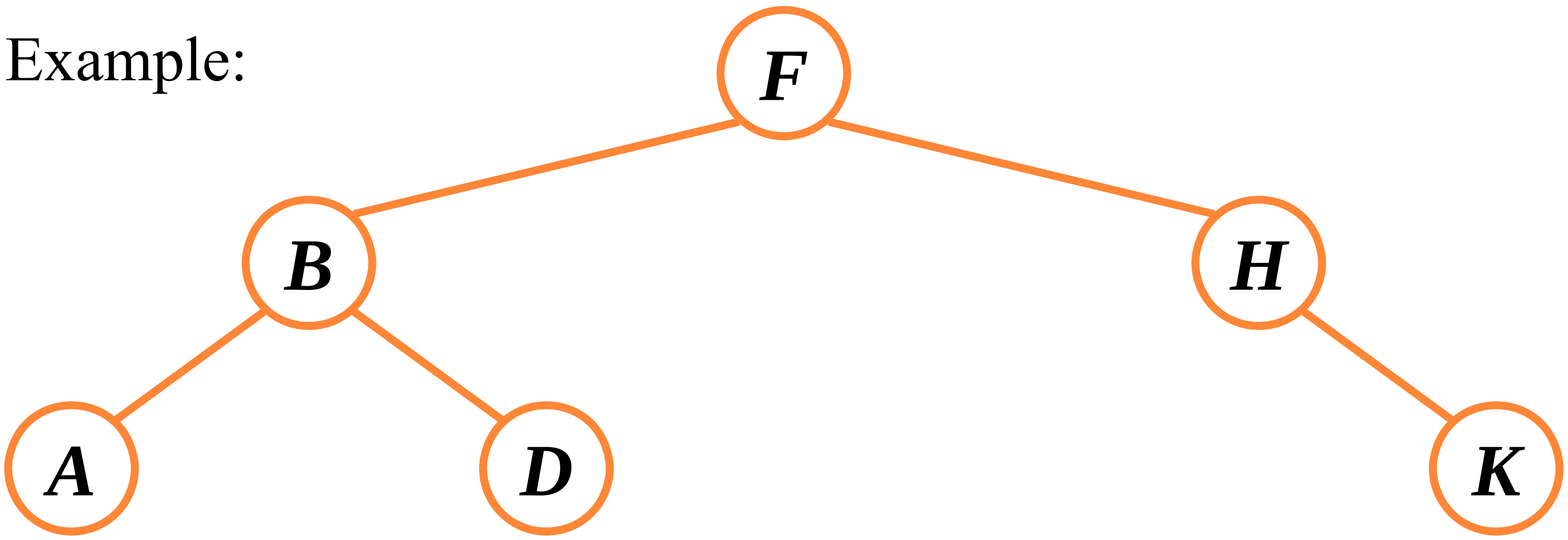
- *What does the following code do?*

```
TreeWalk(x)
    TreeWalk(left[x]);
    print(x);
    TreeWalk(right[x]);
```

- A: prints elements in sorted (increasing) order
- This is called an *inorder tree walk*
  - *Preorder tree walk*: print root, then left, then right
  - *Postorder tree walk*: print left, then right, then root

# INORDER TREE WALK

- Example:




- Output:  A B D F H K
- *How long will a tree walk take?*
  - *Theorem 12.1*

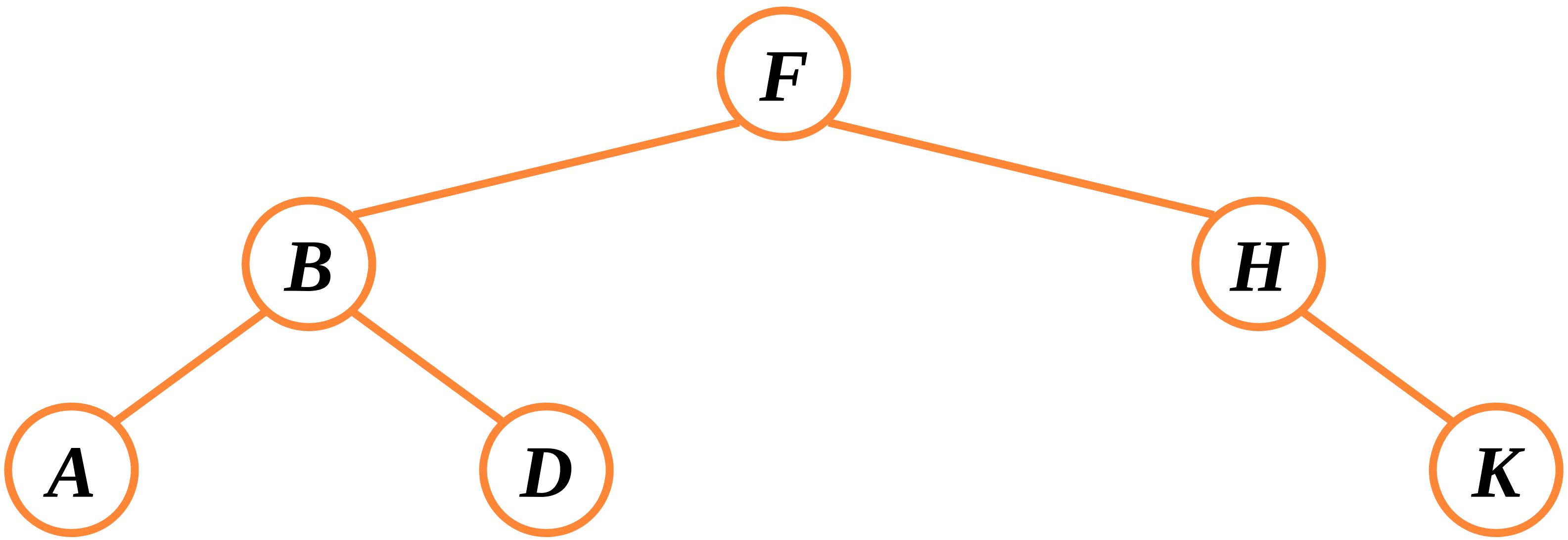
# Operations on BSTs: Search

- Given a key and a pointer to a node, returns an element with that key or NULL:

```
TreeSearch(x, k)
    if (x = NULL  or  k = key[x])
        return x;
    if (k < key[x])
        return TreeSearch(left[x], k);
    else
        return TreeSearch(right[x], k);
```

# BST Search: Example

- Search for *D* and *C*:



# Operations on BSTs: Search

- Here's another function that does the same:

```
TreeSearch(x, k)
    while (x != NULL  and  k != key[x])
        if (k < key[x])
            x = left[x];
        else
            x = right[x];
    return x;
```

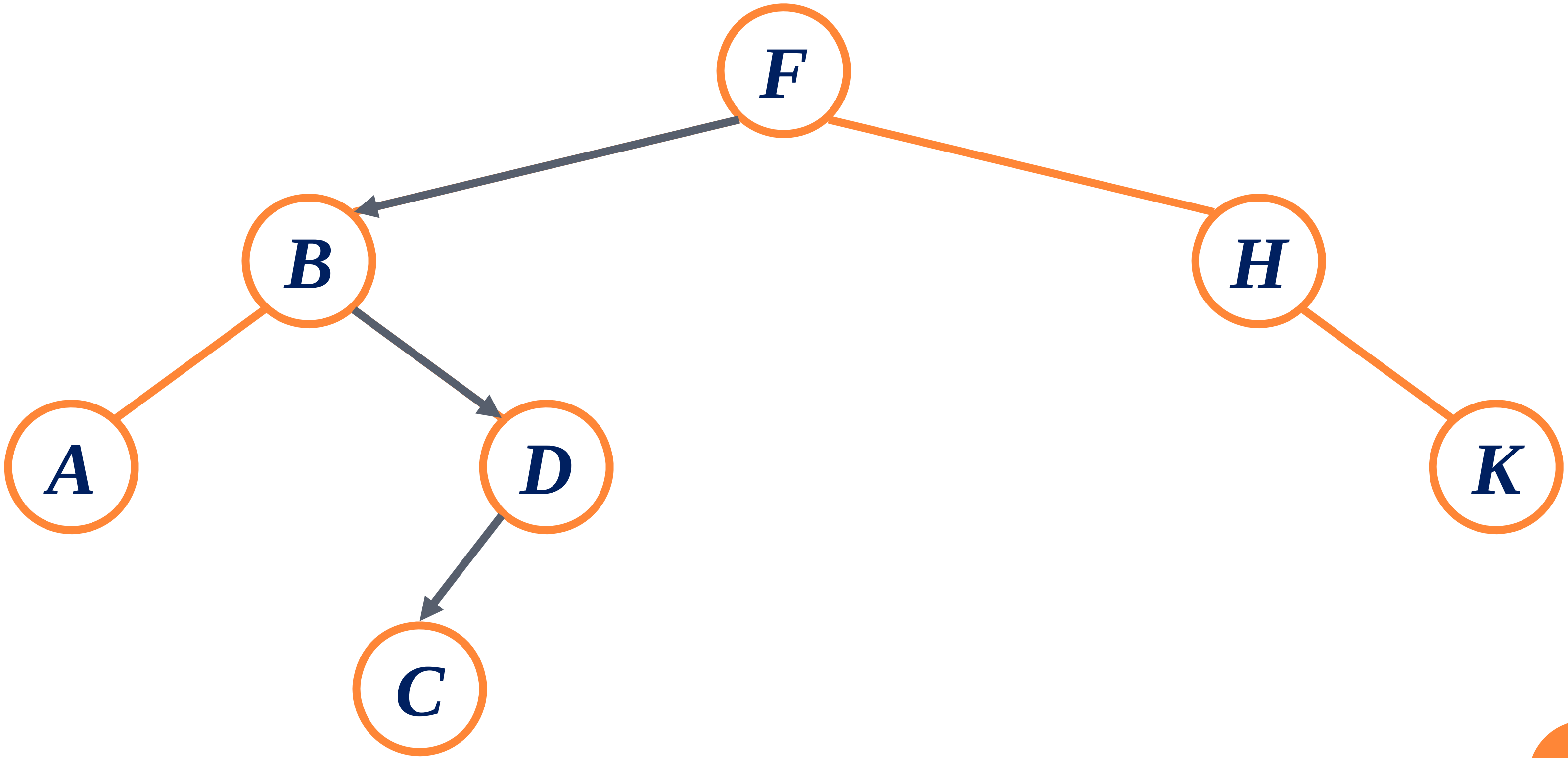
- *Which of these two functions is more efficient?*

# Operations of BSTs: Insert

- Adds an element x to the tree so that the binary search tree property continues to hold
- The basic algorithm
  - Like the search procedure above
  - Insert x in place of NULL

# BST Insert: Example

- Example: Insert *C*



# BST Search/Insert: Running Time

- The height of a binary search tree is $h$
- What is the running time of TreeSearch() or TreeInsert()?
  - $O(h)$
- What is the height of a binary search tree?
  - Worst case: $h = O(n)$ when tree is just a linear string of left or right children

## Minimum of BST

- TREE-Minimum(x)

```
while left(x)≠NIL
      do x←left[x]
Return  x
```

## Maximum of BST

- TREE-Maximum(x)

```
while right(x)≠NIL
      do x←right[x]
Return  x
```

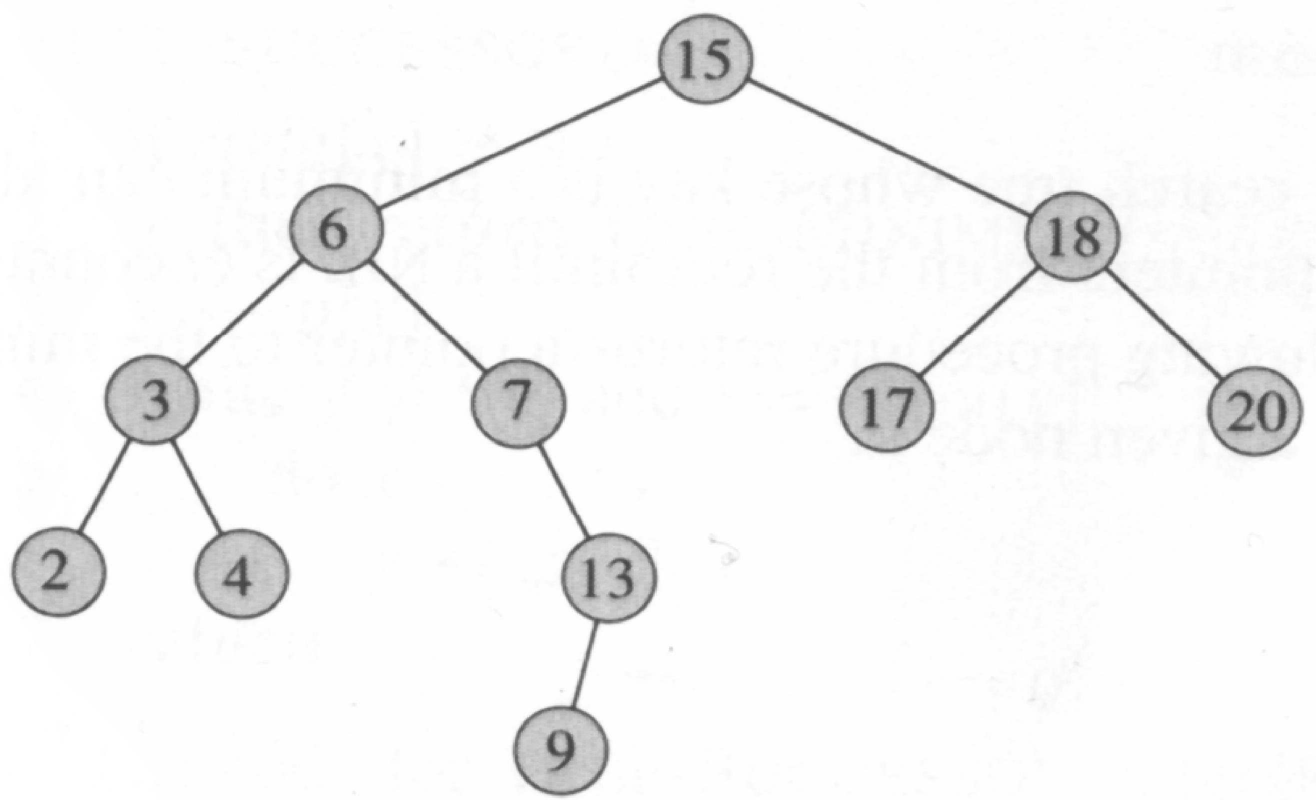
# BST Operations: Successor

- The successor of the current node is the one in the in-order tree walk.
- Two cases:
  - **x has a right subtree**: successor is minimum node in right subtree
  - **x has no right subtree**: successor is lowest ancestor of x whose left child is also one ancestor of x

    (every node is its own ancestor)
  - *Intuition: As long as you move to the left up the tree, you're visiting smaller nodes.*

# BST Operations: Successor

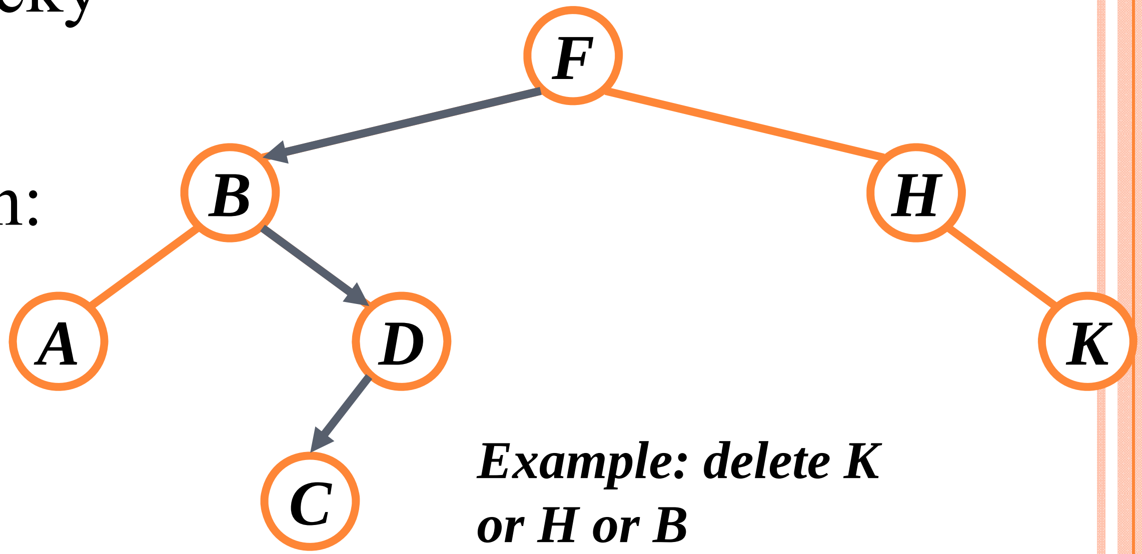
- What is the successor of node 3? 15? 13? 17?




- How about predecessor?

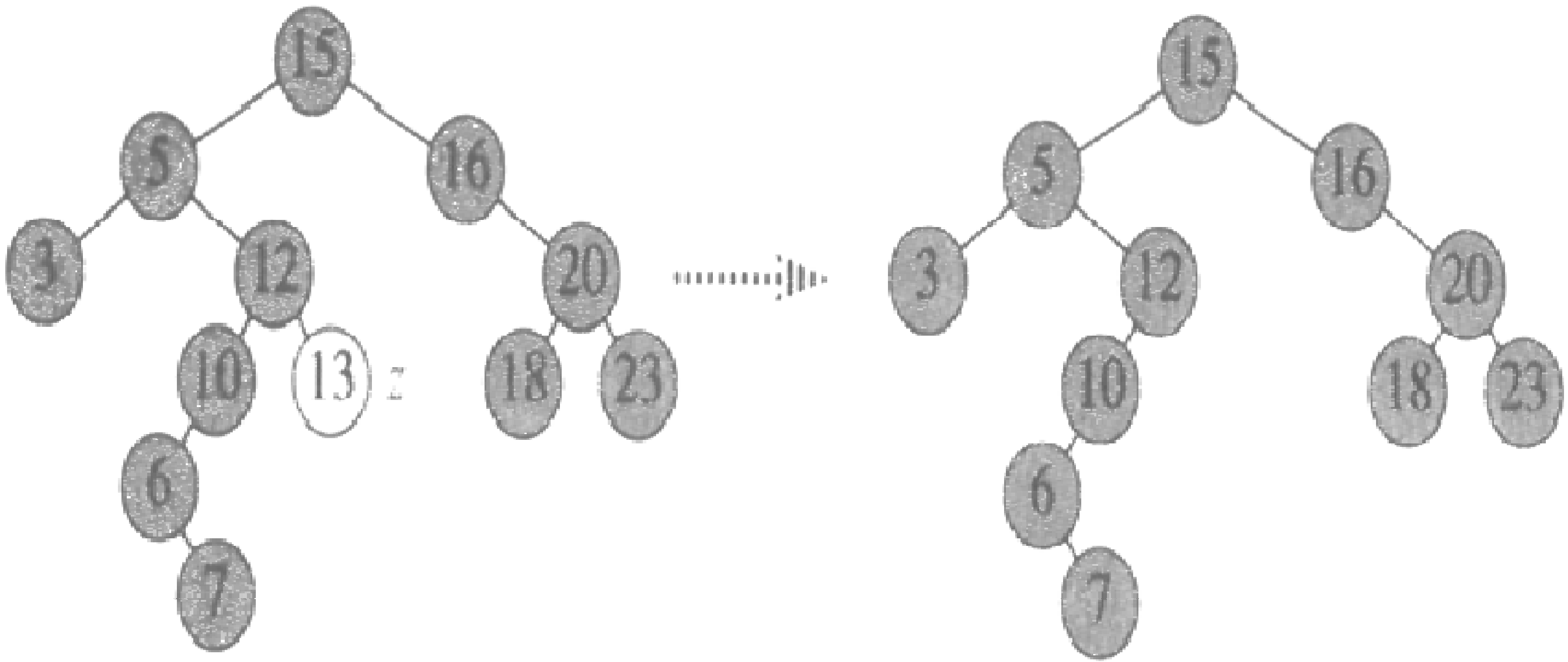
# BST Operations: Delete

- Deletion is a bit tricky
- 3 cases:
  - x has no children:
    - Remove x
  - x has one child:
    - Splice out x
  - x has two children:
    - Swap x with successor
    - Perform case 1 or 2 to delete it

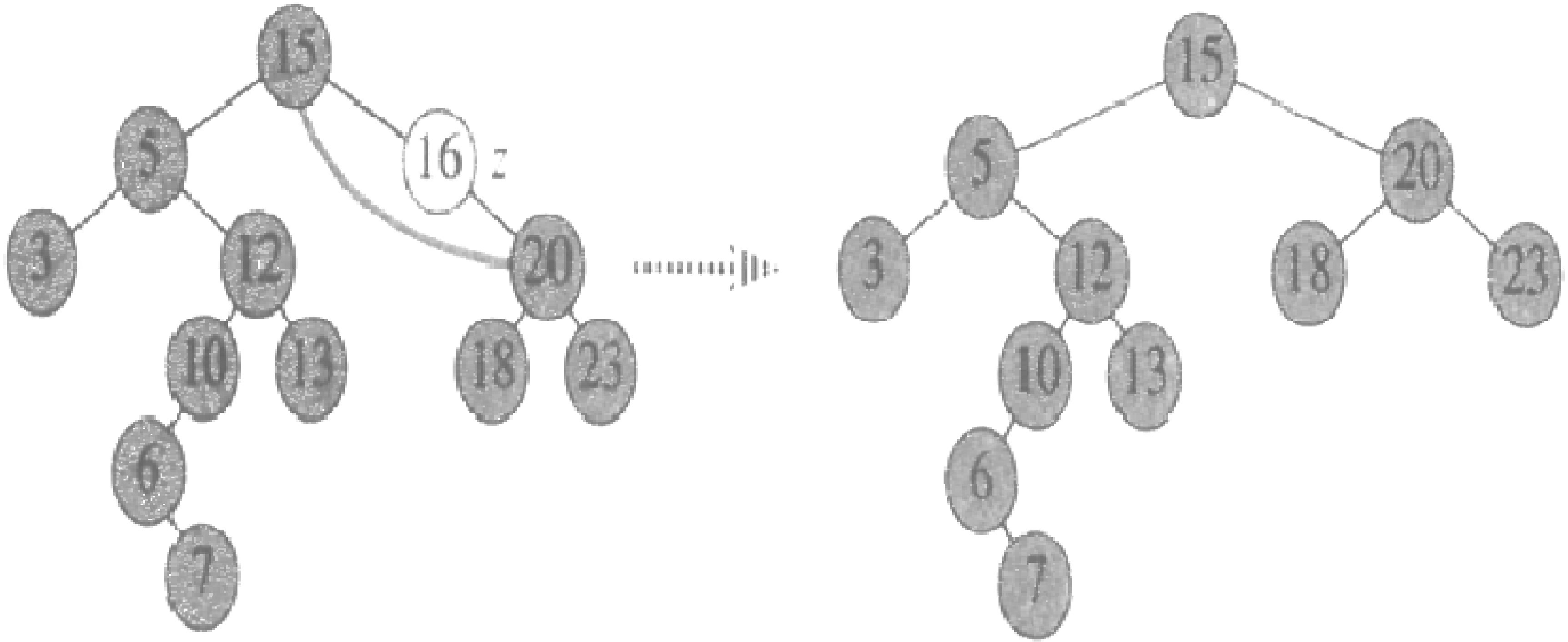



***Example: delete K or H or B***

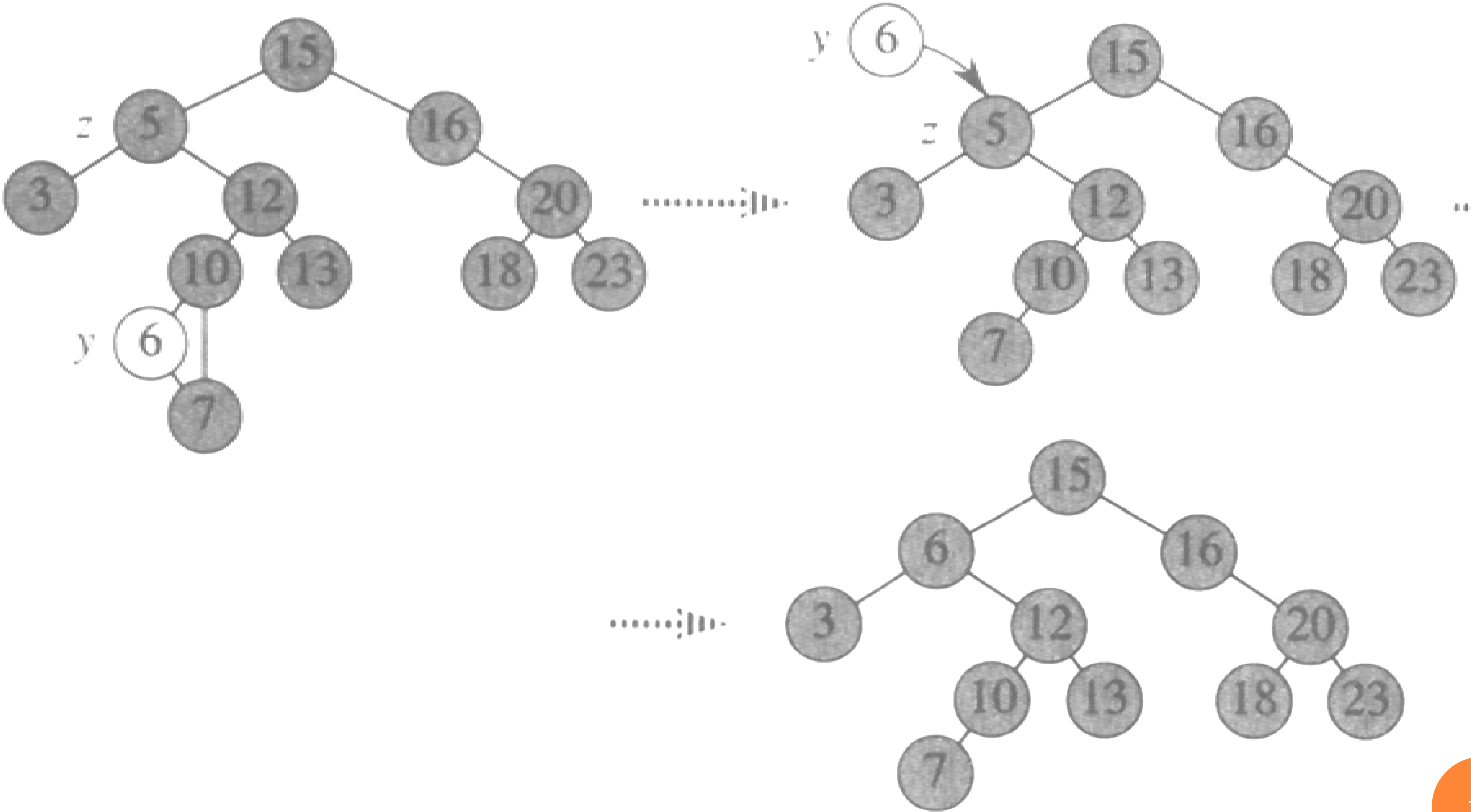
# Z HAS NO CHILDREN



# Z HAS ONLY ONE CHILD



# Z HAS TWO CHILDREN



# BST Operations: Delete

- *Why will case 2 always go to case 0 or case 1?*
  - When x has 2 children, its successor is the minimum in its right subtree.
- *Could we swap x with predecessor instead of successor?*
  - Yes.

# Sorting With Binary Search Trees

- Can you come out an algorithm for sorting by BST?

**3 1 8 2 6 7 5**

# Sorting With Binary Search Trees

- Informal code for sorting array A of length *n*:

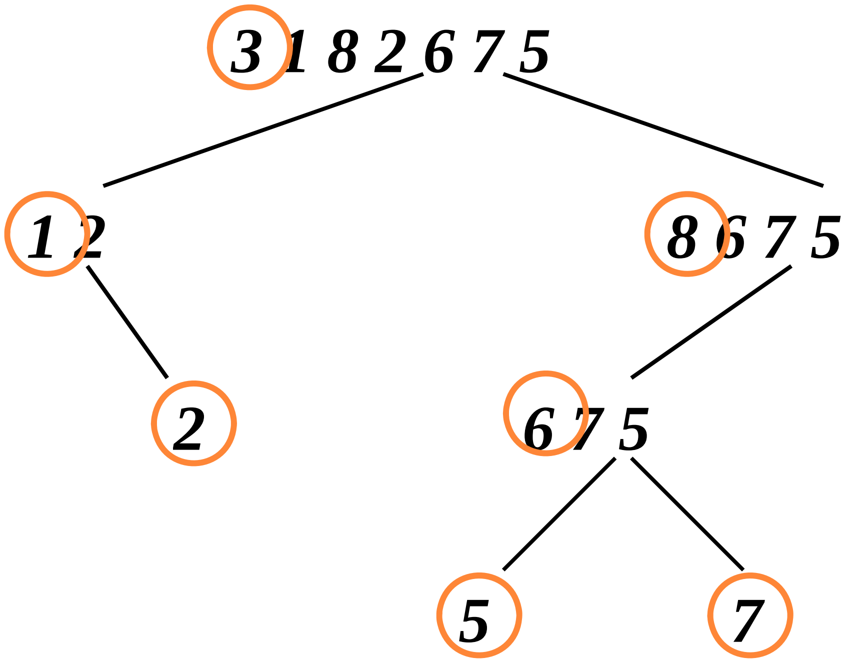
```
BSTSort(A)
    for i=1 to n
        TreeInsert(A[i]);
    InorderTreeWalk(root);
```

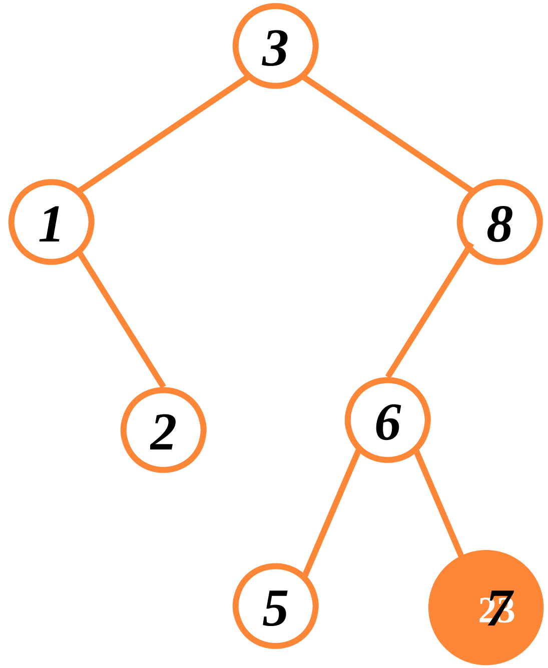
- *What will be the running time in the*
  - *Worst case?*
  - *Best case?*
  - *Average case?*

# SORTING WITH BSTS

```
for i=1 to n
   TreeInsert(A[i]);
InorderTreeWalk(root);
```

- Average case analysis
  - It's a form of quicksort!



# SORTING WITH BSTS

- Inserted nodes are similar to partition pivot used in quicksort, but in a different order.
  - BST does not partition immediately after picking the inserted node.

# SORTING WITH BSTS

- Since run time is proportional to the number of comparisons, same time as quicksort: O(n lg n)
- Which do you think is better, quicksort or BSTSort? Why?

# SORTING WITH BSTS

- Since run time is proportional to the number of comparisons, same time as quicksort: O(n lg n)
- Which do you think is better, quicksort or BSTSort? Why?
  - Quicksort
  - Sorts in place
  - Doesn't need to build data structure

# More BST Operations

- BSTs are good for more than sorting. For example, can implement a priority queue
- *What operations must a priority queue have?*
  - Insert
  - Minimum

# Randomly Built Binary Search Tree

# DEFINITION

- A randomly built binary search tree on $n$ keys as one that arises from inserting the keys in random order into an initially empty tree, where each of the $n!$ permutations of the input keys is equally likely.

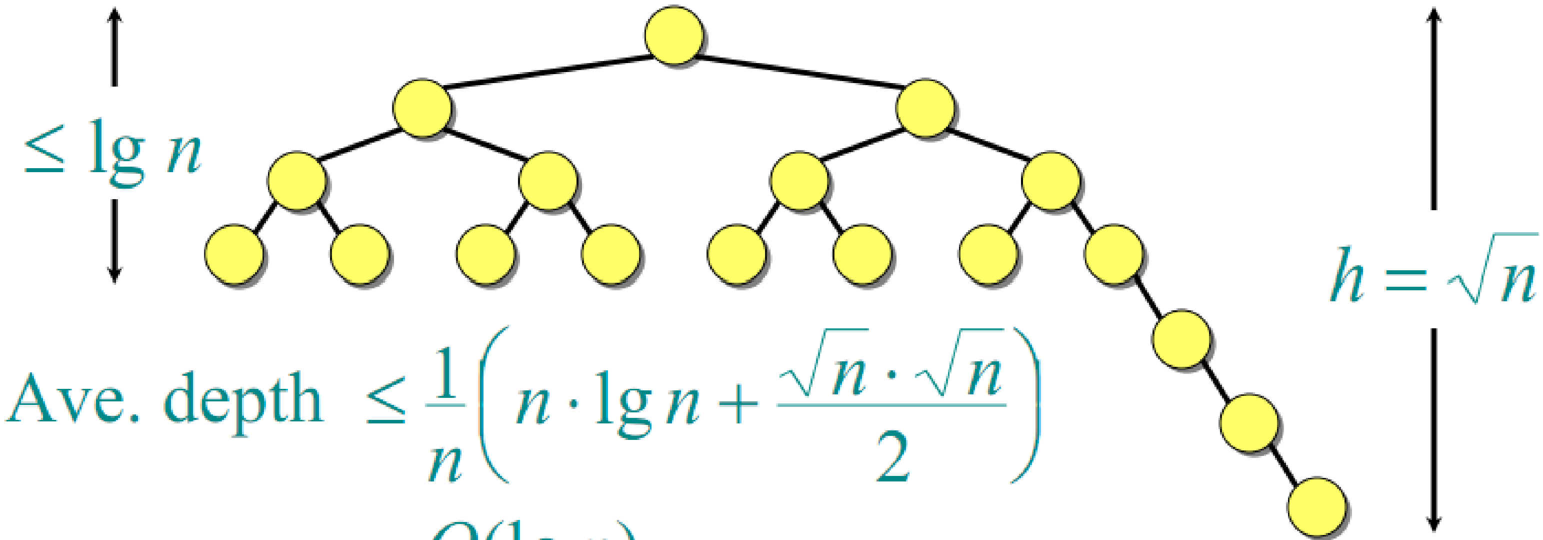
# Randomly Built Binary Search Tree

- **Theorem**: The average height of a randomly-built binary search tree of $n$ distinct keys is $O(\lg n)$
- **Corollary**: The dynamic operations Successor, Predecessor, Search, Min, Max, Insert, and Delete all have $O(\lg n)$ average complexity on randomly-built binary search trees.

# EXPECTED TREE HEIGHT

- Average node depth of a randomly built BST = O(lg n) does not necessarily mean that its expected height is also O(lg n) (although it is).

**Example.**




$$\text{Ave. depth} \leq \frac{1}{n}\left(n \cdot \lg n + \frac{\sqrt{n} \cdot \sqrt{n}}{2}\right)$$
$$= O(\lg n)$$

# Height of a Randomly Built Binary Search Tree

**Outline of the analysis:**

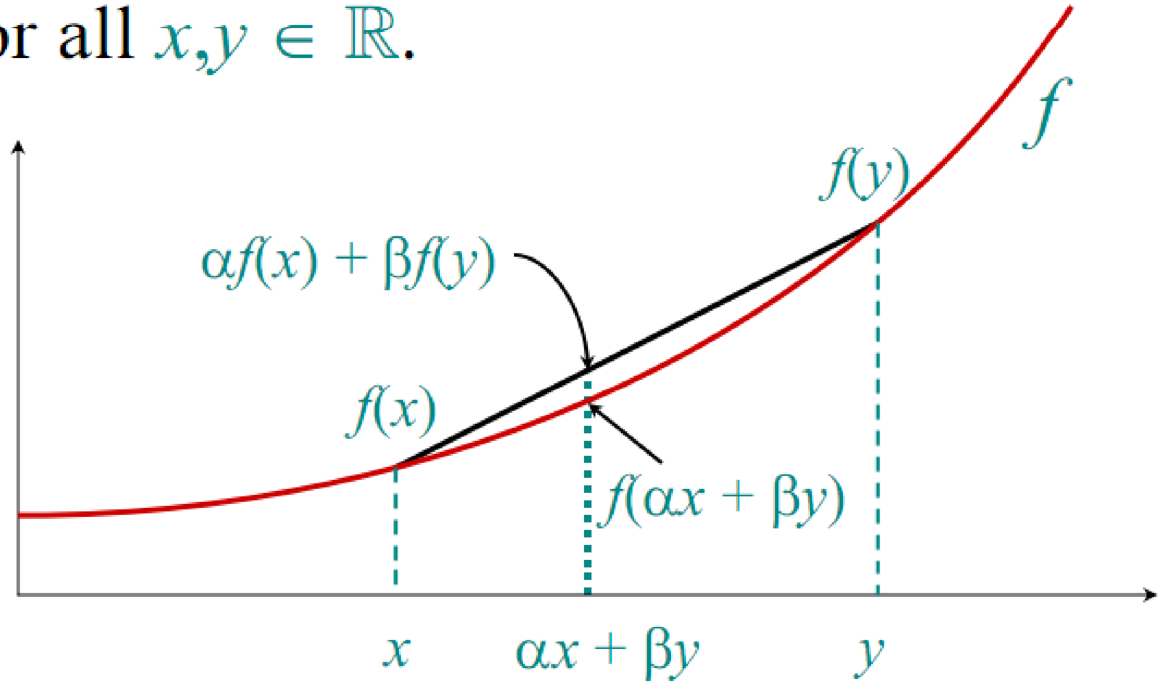
- Prove ***Jensen's inequality***, which says that $f(E[X]) \le E[f(X)]$ for any convex function $f$ and random variable $X$.
- Analyze the ***exponential height*** of a randomly built BST on $n$ nodes, which is the random variable $Y_n = 2^{X_n}$, where $X_n$ is the random variable denoting the height of the BST.
- Prove that $2^{E[X_n]} \le E[2^{X_n}] = E[Y_n] = O(n^3)$, and hence that $E[X_n] = O(\lg n)$.

# Convex functions

A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is ***convex*** if for all $\alpha, \beta \geq 0$ such that $\alpha + \beta = 1$, we have

$$f(\alpha x + \beta y) \leq \alpha f(x) + \beta f(y)$$

for all $x, y \in \mathbb{R}$.



# CONVEXITY LEMMA

**Lemma.** Let $f : \mathbb{R} \to \mathbb{R}$ be a convex function, and let $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ be a set of nonnegative constants such that $\sum_k \alpha_k = 1$. Then, for any set $\{x_1, x_2, \ldots, x_n\}$ of real numbers, we have

$$f\left(\sum_{k=1}^{n} \alpha_k x_k\right) \leq \sum_{k=1}^{n} \alpha_k f(x_k).$$

*Proof.* By induction on $n$. For $n = 1$, we have $\alpha_1 = 1$, and hence $f(\alpha_1 x_1) \leq \alpha_1 f(x_1)$ trivially.

# PROOF (CONTINUED)

Inductive step:

$$f\left(\sum_{k=1}^{n}\alpha_k x_k\right) = f\left(\alpha_n x_n + (1-\alpha_n)\sum_{k=1}^{n-1}\frac{\alpha_k}{1-\alpha_n}x_k\right)$$

Algebra.

# Proof (continued)

Inductive step:

$$f\left(\sum_{k=1}^{n}\alpha_k x_k\right) = f\left(\alpha_n x_n + (1-\alpha_n)\sum_{k=1}^{n-1}\frac{\alpha_k}{1-\alpha_n}x_k\right)$$

$$\leq \alpha_n f(x_n) + (1-\alpha_n) f\left(\sum_{k=1}^{n-1}\frac{\alpha_k}{1-\alpha_n}x_k\right)$$

Convexity.

# Proof (continued)

Inductive step:

$$f\left(\sum_{k=1}^{n}\alpha_k x_k\right)=f\left(\alpha_n x_n+(1-\alpha_n)\sum_{k=1}^{n-1}\frac{\alpha_k}{1-\alpha_n}x_k\right)$$

$$\le \alpha_n f(x_n)+(1-\alpha_n)f\left(\sum_{k=1}^{n-1}\frac{\alpha_k}{1-\alpha_n}x_k\right)$$

$$\le \alpha_n f(x_n)+(1-\alpha_n)\sum_{k=1}^{n-1}\frac{\alpha_k}{1-\alpha_n}f(x_k)$$

Induction.

# PROOF (CONTINUED)

Inductive step:

$$
\begin{aligned}
f\left(\sum_{k=1}^{n}\alpha_k x_k\right) &= f\left(\alpha_n x_n + (1-\alpha_n)\sum_{k=1}^{n-1}\frac{\alpha_k}{1-\alpha_n}x_k\right)\\
&\le \alpha_n f(x_n) + (1-\alpha_n) f\left(\sum_{k=1}^{n-1}\frac{\alpha_k}{1-\alpha_n}x_k\right)\\
&\le \alpha_n f(x_n) + (1-\alpha_n)\sum_{k=1}^{n-1}\frac{\alpha_k}{1-\alpha_n}f(x_k)\\
&= \sum_{k=1}^{n}\alpha_k f(x_k). \quad \square
\end{aligned}
$$

Algebra.

# JENSEN'S INEQUALITY

**Lemma.** Let $f$ be a convex function, and let $X$ be a random variable. Then, $f(E[X]) \le E[f(X)]$.

*Proof.*

$$f(E[X]) = f\left( \sum_{k=-\infty}^{\infty} k \cdot \Pr\{X = k\} \right)$$

Definition of expectation.

# JENSEN'S INEQUALITY

**Lemma.** Let $f$ be a convex function, and let $X$ be a random variable. Then, $f(E[X]) \le E[f(X)]$.

*Proof.*

$$f(E[X]) = f\left(\sum_{k=-\infty}^{\infty} k \cdot \Pr\{X = k\}\right)$$

$$\le \sum_{k=-\infty}^{\infty} f(k) \cdot \Pr\{X = k\}$$

Convexity lemma (generalized).

# JENSEN'S INEQUALITY

**Lemma.** Let $f$ be a convex function, and let $X$ be a random variable. Then, $f(E[X]) \le E[f(X)]$.

*Proof.*

$$f(E[X]) = f\left(\sum_{k=-\infty}^{\infty} k \cdot \Pr\{X = k\}\right)$$

$$\le \sum_{k=-\infty}^{\infty} f(k) \cdot \Pr\{X = k\}$$

$$= E[f(X)]. \quad \square$$

Tricky step, but true—think about it.

# Analysis of BST height

Let $X_n$ be the random variable denoting the height of a randomly built binary search tree on $n$ nodes, and let $Y_n = 2^{X_n}$ be its exponential height.

If the root of the tree has rank $k$, then

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

since each of the left and right subtrees of the root are randomly built. Hence, we have

$$Y_n = 2 \cdot \max\{Y_{k-1}, Y_{n-k}\} .$$

# ANALYSIS (CONTINUED)

Define the indicator random variable $Z_{nk}$ as

$$Z_{nk} = \begin{cases} 1 & \text{if the root has rank } k, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, $\Pr\{Z_{nk} = 1\} = \mathrm{E}[Z_{nk}] = 1/n$, and

$$Y_n = \sum_{k=1}^{n} Z_{nk}\left(2 \cdot \max\{Y_{k-1}, Y_{n-k}\}\right) .$$

# Exponential height recurrence

$$E[Y_n] = E\left[\sum_{k=1}^{n} Z_{nk}\left(2 \cdot \max\{Y_{k-1}, Y_{n-k}\}\right)\right]$$

Take expectation of both sides.

# Exponential height recurrence

$$E[Y_n] = E\left[\sum_{k=1}^{n} Z_{nk}\left(2\cdot\max\{Y_{k-1}, Y_{n-k}\}\right)\right]$$

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

Linearity of expectation.

# Exponential height recurrence

$$E[Y_n] = E\left[\sum_{k=1}^{n} Z_{nk}\,(2 \cdot \max\{Y_{k-1}, Y_{n-k}\})\right]$$

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

$$= 2\sum_{k=1}^{n} E[Z_{nk}] \cdot E[\max\{Y_{k-1}, Y_{n-k}\}]$$

Independence of the rank of the root from the ranks of subtree roots.

# Exponential height recurrence

$$E[Y_n] = E\left[\sum_{k=1}^{n} Z_{nk}\left(2 \cdot \max\{Y_{k-1}, Y_{n-k}\}\right)\right]$$

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

$$= 2\sum_{k=1}^{n} E[Z_{nk}] \cdot E[\max\{Y_{k-1}, Y_{n-k}\}]$$

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

The max of two nonnegative numbers is at most their sum, and $E[Z_{nk}] = 1/n$.

# Exponential height recurrence

$$E[Y_n] = E\left[\sum_{k=1}^{n} Z_{nk}\left(2\cdot\max\{Y_{k-1}, Y_{n-k}\}\right)\right]$$

$$= \sum_{k=1}^{n} E[Z_{nk}\left(2\cdot\max\{Y_{k-1}, Y_{n-k}\}\right)]$$

$$= 2\sum_{k=1}^{n} E[Z_{nk}]\cdot E[\max\{Y_{k-1}, Y_{n-k}\}]$$

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

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

Each term appears twice, and reindex.

# Solving the Recurrence

Use substitution to show that $E[Y_n] \leq cn^3$ for some positive constant $c$, which we can pick sufficiently large to handle the initial conditions.

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

# SOLVING THE RECURRENCE

Use substitution to show that $E[Y_n] \le cn^3$ for some positive constant $c$, which we can pick sufficiently large to handle the initial conditions.

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

$$\le \frac{4}{n}\sum_{k=0}^{n-1} ck^3$$

Substitution.

# Solving the recurrence

Use substitution to show that $E[Y_n] \le cn^3$ for some positive constant $c$, which we can pick sufficiently large to handle the initial conditions.

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

$$\le \frac{4}{n}\sum_{k=0}^{n-1} ck^3$$

$$\le \frac{4c}{n}\int_0^n x^3\,dx$$

Integral method.

# Solving the recurrence

Use substitution to show that $E[Y_n] \le cn^3$ for some positive constant $c$, which we can pick sufficiently large to handle the initial conditions.

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

$$\le \frac{4}{n}\sum_{k=0}^{n-1} ck^3$$

$$\le \frac{4c}{n}\int_0^n x^3\,dx$$

$$= \frac{4c}{n}\left(\frac{n^4}{4}\right)$$

Solve the integral.

# SOLVING THE RECURRENCE

Use substitution to show that $E[Y_n] \le cn^3$ for some positive constant $c$, which we can pick sufficiently large to handle the initial conditions.

$$
\begin{aligned}
E[Y_n] &= \frac{4}{n}\sum_{k=0}^{n-1} E[Y_k] \\
&\le \frac{4}{n}\sum_{k=0}^{n-1} ck^3 \\
&\le \frac{4c}{n}\int_0^n x^3\,dx \\
&= \frac{4c}{n}\left(\frac{n^4}{4}\right) \\
&= cn^3. \quad \text{Algebra.}
\end{aligned}
$$

# THE GRAND FINALE

Putting it all together, we have

$$2^{E[X_n]} \leq E[2^{X_n}]$$

Jensen's inequality, since $f(x) = 2^x$ is convex.

# THE GRAND FINALE

Putting it all together, we have

$$2^{E[X_n]} \leq E[2^{X_n}]$$

$$= E[Y_n]$$

Definition.

# THE GRAND FINALE

Putting it all together, we have

$$
\begin{aligned}
2^{E[X_n]} &\leq E[2^{X_n}] \\
&= E[Y_n] \\
&\leq cn^3 .
\end{aligned}
$$

What we just showed.

# THE GRAND FINALE

Putting it all together, we have

$$
\begin{aligned}
2^{E[X_n]} &\le E[2^{X_n}] \\
&= E[Y_n] \\
&\le cn^3 .
\end{aligned}
$$

Taking the lg of both sides yields

$$E[X_n] \le 3 \lg n + O(1) .$$

# HOMEWORK

- 12.1-5,12.2-5

- Due day:10.10