Binary Search Trees

Prof. Zheng Zhang

Harbin Institute of Technology, Shenzhen

BINARY SEARCH TREES

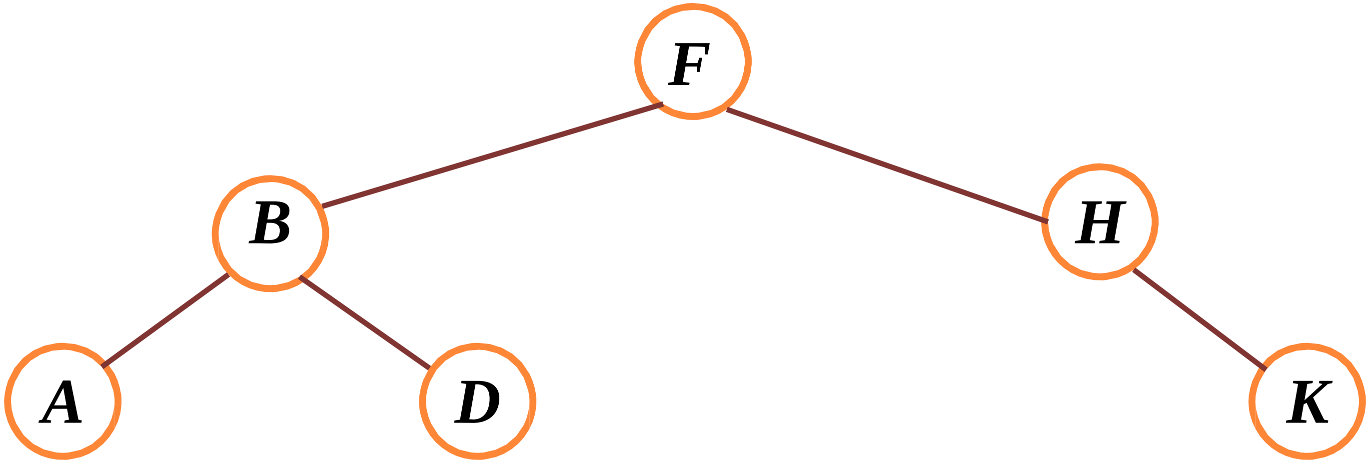
- *Binary Search Trees* (BSTs) are an important data structure for dynamic sets (Operations).
- In addition to satellite data, elements have:
  - *key*: an identifying field inducing a total ordering (other satellite data)
  - *left*: pointer to a left child (may be NULL)
  - *right*: pointer to a right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

BINARY SEARCH TREES

- BST property:

  $$key[\text{left}(x)] \le key[x] \le key[\text{right}(x)]$$

- Example:



BINARY SEARCH TREES

A=[05,13,19,21,37,56,64,75,80,88,92]

Key=21

05 13 19 21 37 56 64 75 80 88 92

Low=1 **mid=$\lfloor$(Low+Up)/2$\rfloor$=6** **Up=11**

mid=3 **Up=5**

Low=4, mid=4

Key=85

05 13 19 21 37 56 64 75 80 88 92

Low=1 **mid=6** **Up=11**

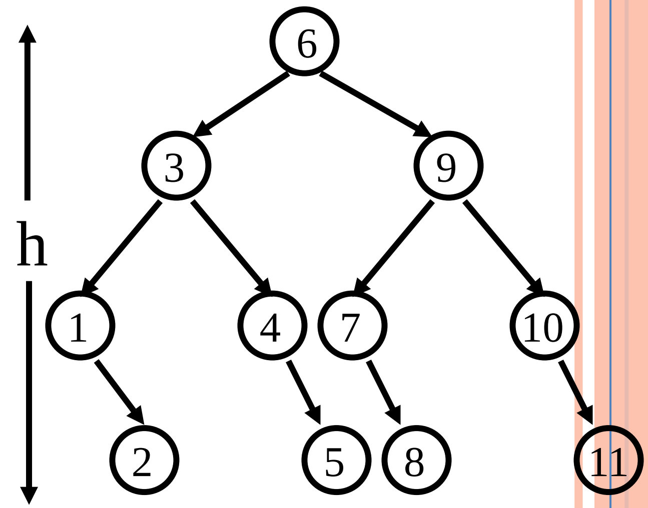
Low=7 **mid=9**

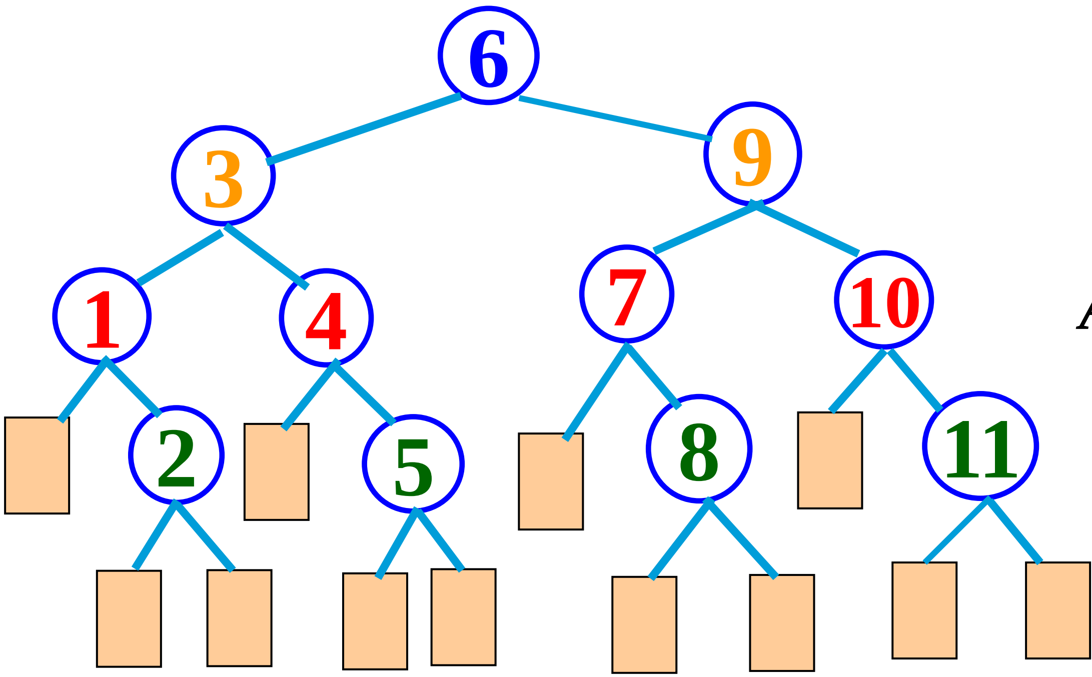
Low=10, mid=10

Up=9 **Low=10**

Binary Search Trees

1	2	3	4	5	6	7	8	9	10	11
05	13	19	21	37	56	64	75	80	88	92
1		3			6			9		11





$$ASL_{bs} = \sum_{i=1}^{n} P_i \cdot C_i \qquad P_i = 1/n$$

$$= 1/n \cdot \sum_{j=1}^{h} j \cdot 2^{j-1}$$

$$=(n+1)/n \cdot \log_2(n+1)-1$$

$$\approx \log_2(n+1)-1$$

Inorder Tree Walk (Traversal)

- *What does the following code do?*

TreeWalk(*x*)

1. if $x \neq$ NIL

2. TreeWalk(*left*[*x*]);

3. print(*x*);

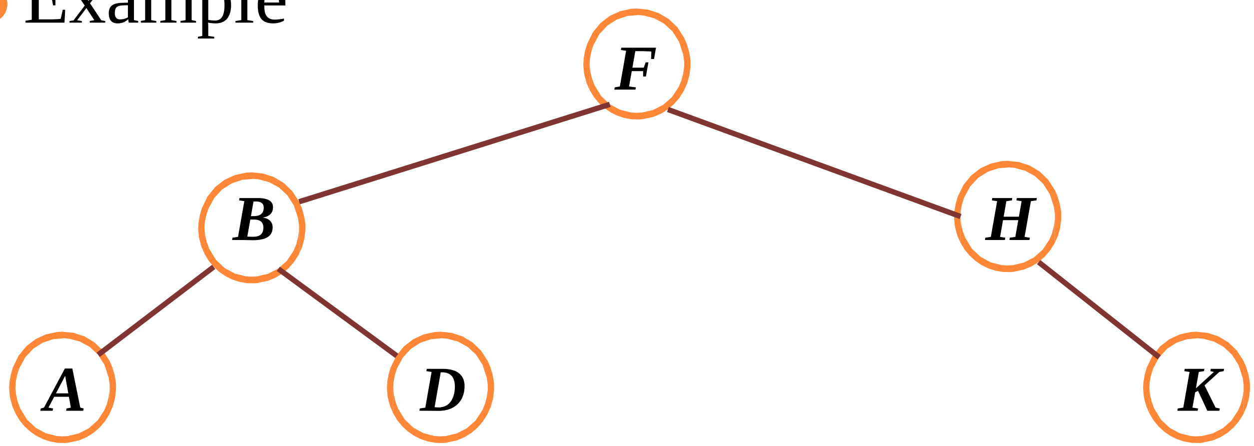
4. TreeWalk(*right*[*x*]);

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

If x is the root of an n-node subtree, then the call INORDER-TREE-WALK(x) takes $\Theta(n)$ time.

Operations on BSTs: Search

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

TreeSearch(*x*, *k*)

1. if (*x* = NULL *or* *k* = *key*[*x*]) return *x*;

2. if (*k* < *key*[*x*])

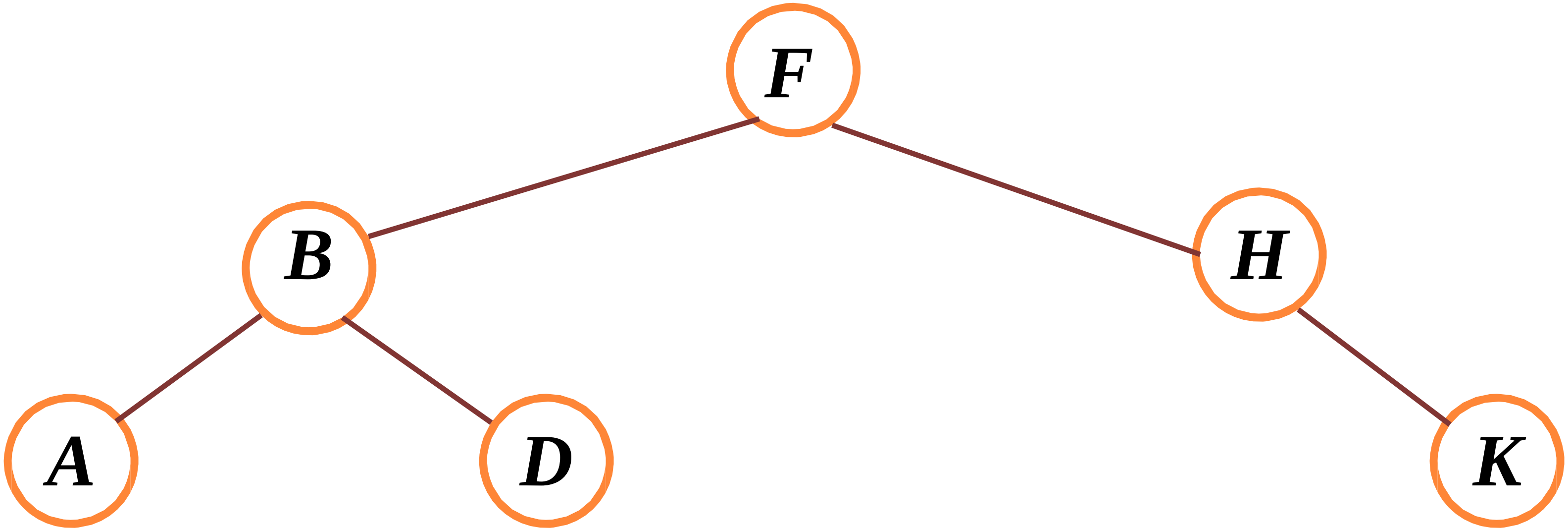
3.   return TreeSearch(*left*[*x*], *k*);

4. else

5.   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*;

Minimum and maximum?

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

Operations on BSTs: Min-Max

Minimum of BST

- TREE-Minimum(x)

1 while *left*(x) $\neq$ NIL

2 do $x \leftarrow$ *left*[x]

3 Return x

Maximum of BST

- TREE-Maximum(x)

1 while *right*($\underline{x}$) $\neq$ NIL

2 do $x \leftarrow$ *right*[x]

3 Return x

OPERATIONS OF BSTS: INSERT

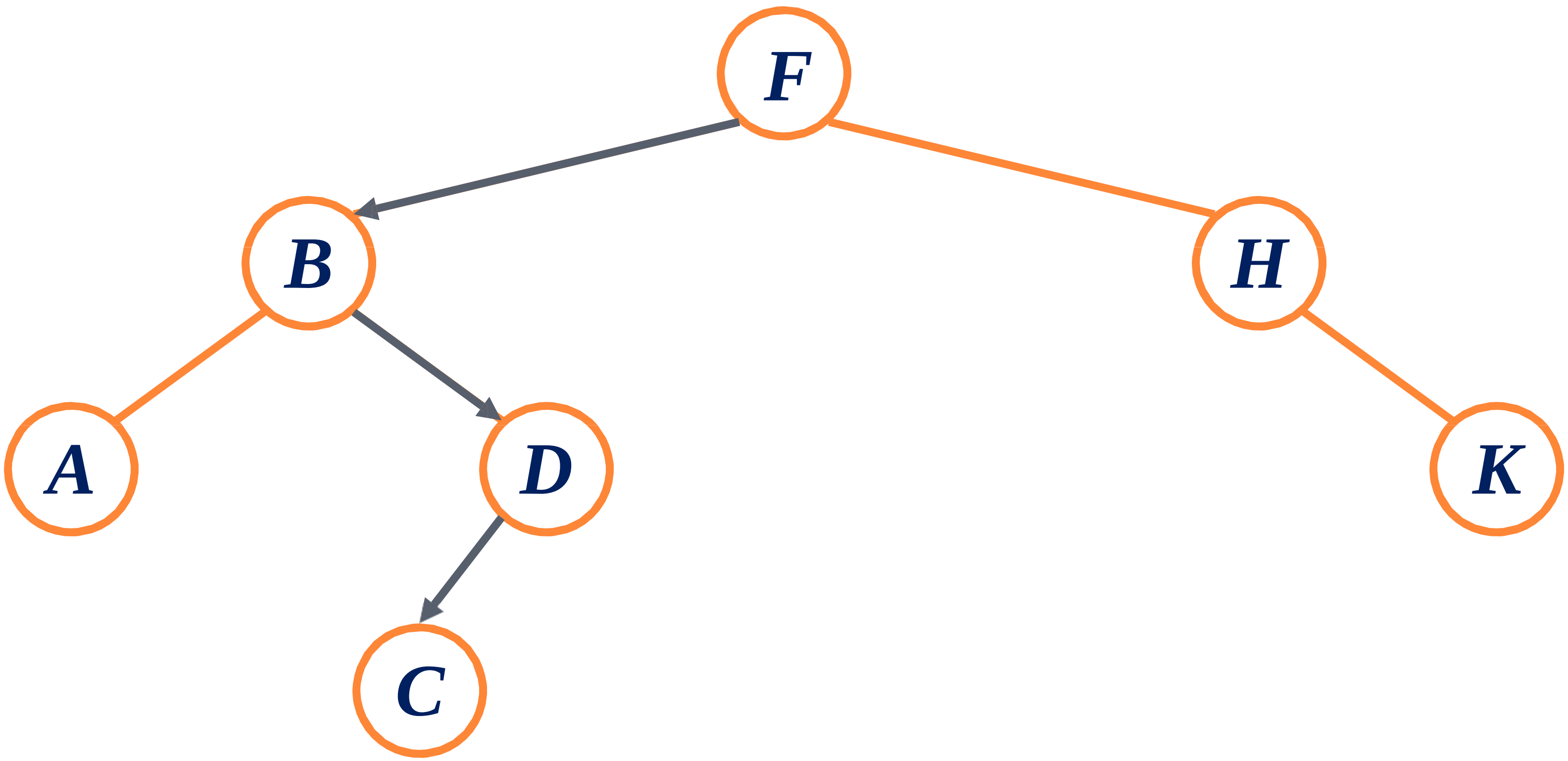
- Adds an element *z* to the tree so that the binary search tree property continues to hold
- The basic algorithm (straightforward)
  - Like the search procedure above
  - Insert *x* in <u>place of NULL</u>

Operations of BSTs: Insert

```
TREE-INSERT( T, z )
y = NIL;
x = T.root;
while x ≠ NIL
      y = x;
      if  z.key < x.key
           x = x.left;
      else x = x.right;
z.p = y;
if  y == NIL
        T.root = z ;
                    // empty tree T
elseif   z.key < y.key
        y.left = z;
else  y.right = z;
```

BST Insert: Example

- Example: Insert *C*



BST Search/Insert: Running Time

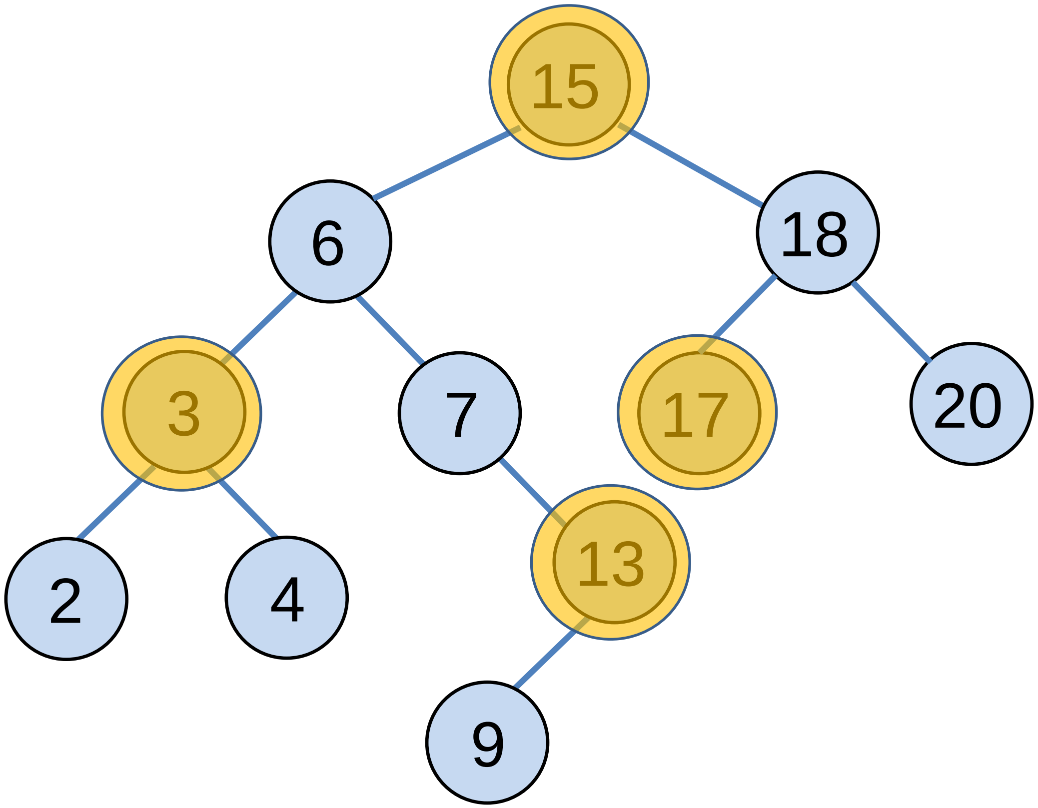
- TREE-INSERT begins at the root of the tree and the pointer x traces a simple path downward *looking for a NIL* to replace with the input item z.
- The height of a binary search tree is h
- What is the running time of *TreeSearch*( ) or *TreeInsert*( )?
  - $O(h)$
- What determines the height of a binary search tree?
  - Worst case: $h = O(n)$ when tree is just a linear string of left or right children

BST OPERATIONS: SUCCESSOR

- The successor of the current node is the one in the in-order tree walk (distinct keys).
- Two cases:
  - ***x* has a right subtree**: successor is <u>minimum node</u> in right subtree (the leftmost node in *x*'s 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.*
    - To find *y*, we simply go up the tree from *x* until we encounter a node that is the *left child* of its parent.

BST Operations: Successor

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



- How about the Predecessor?

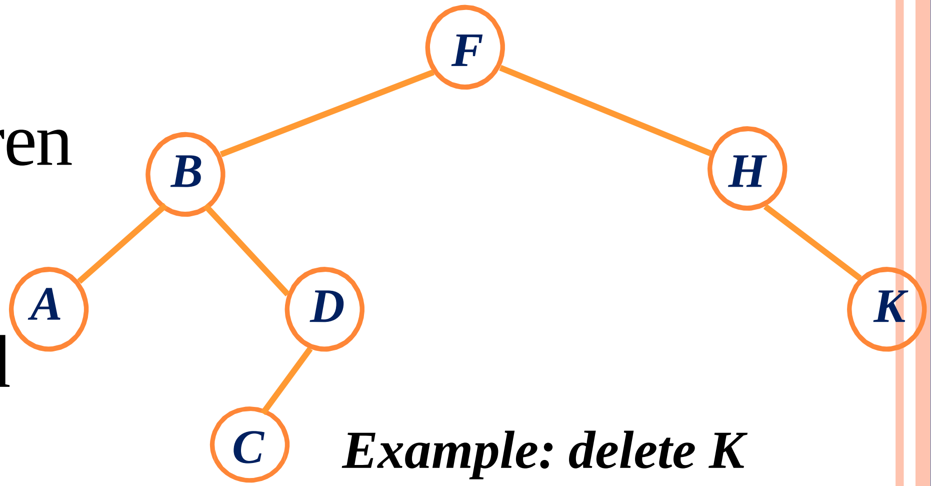
BST Operations: Successor

- **Theorem** ***12.2***

We can implement the dynamic-set operations SEARCH, MINIMUM, MAXIMUM, SUCCESSOR, and PREDECESSOR so that each one runs in $O(h)$ time on a binary search tree of height h.

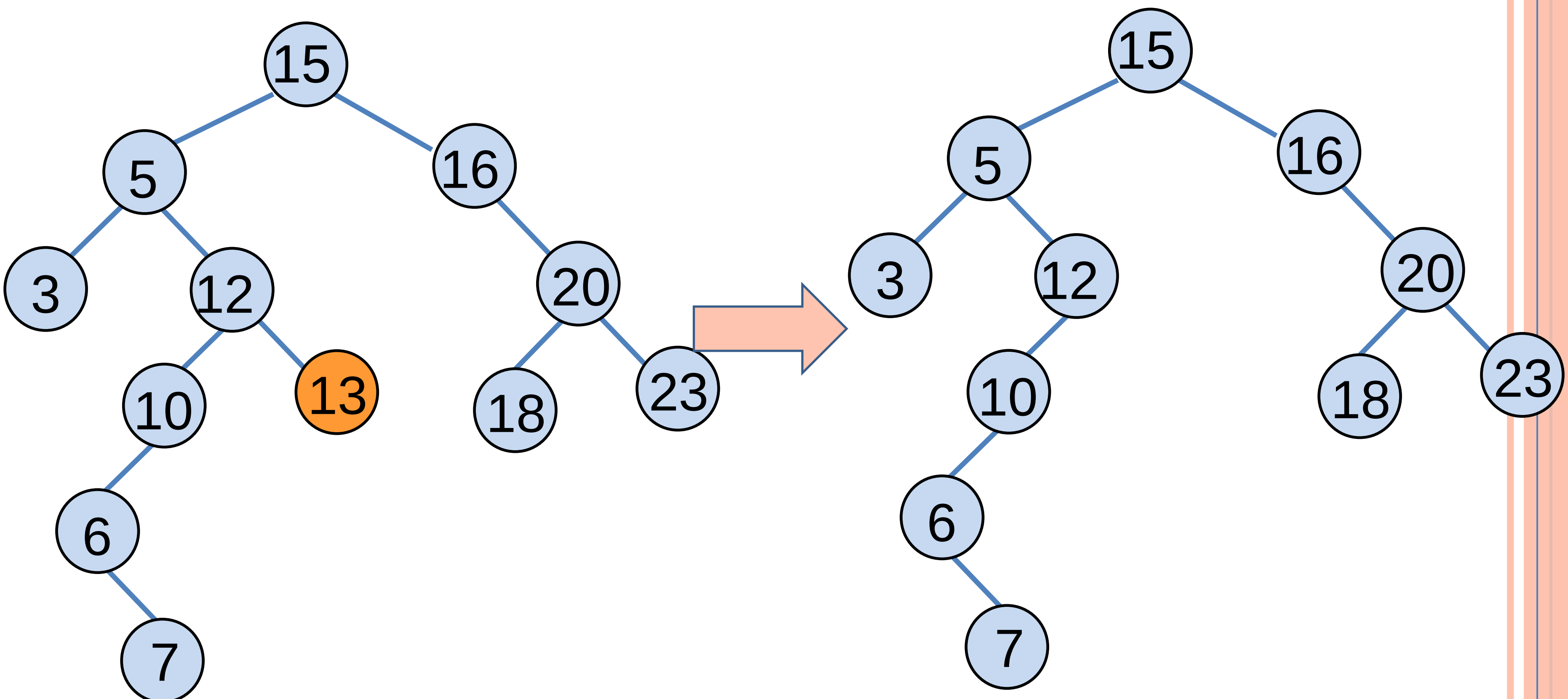
BST OPERATIONS: DELETE

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

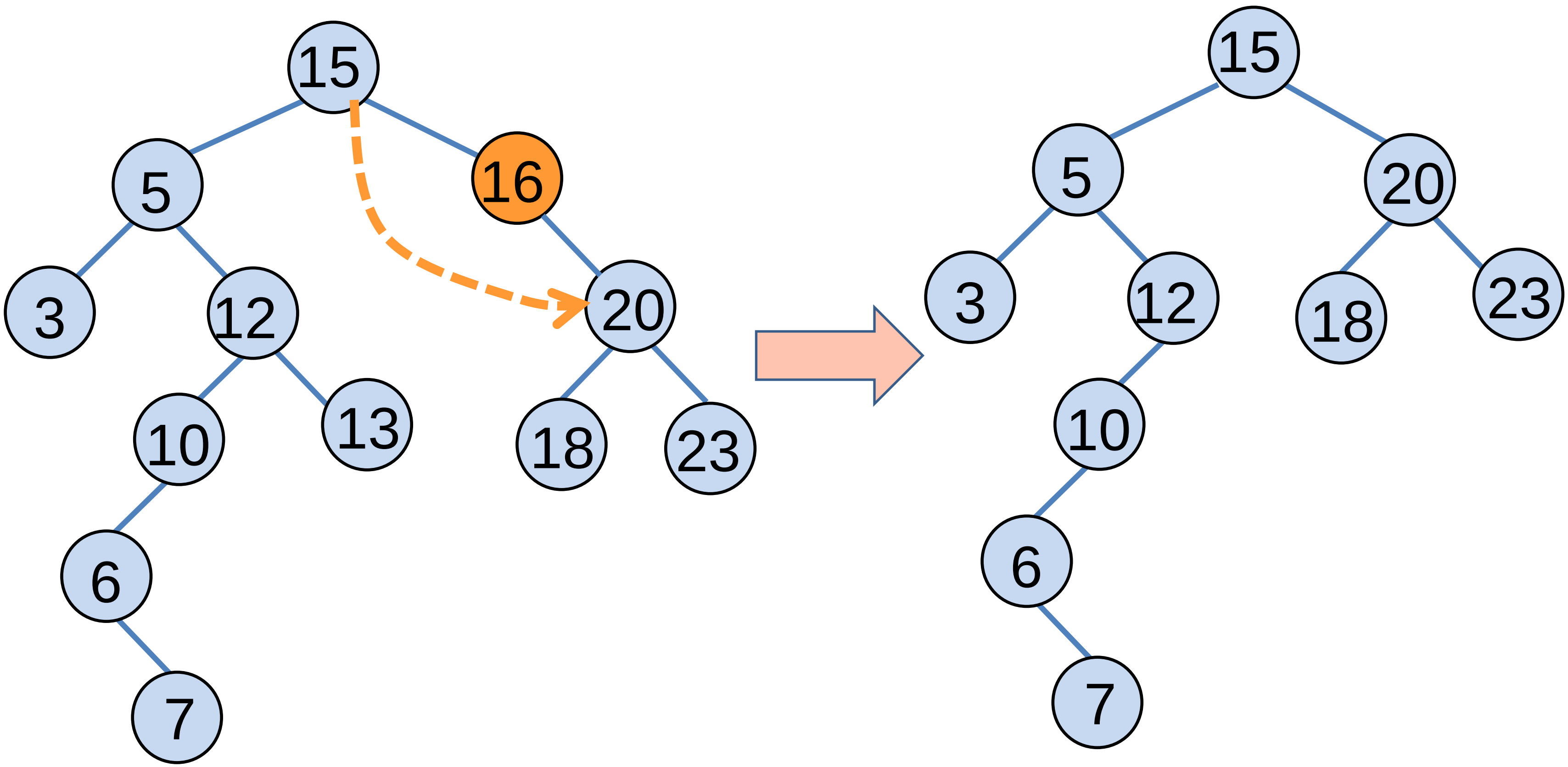


Example: delete K or H or B

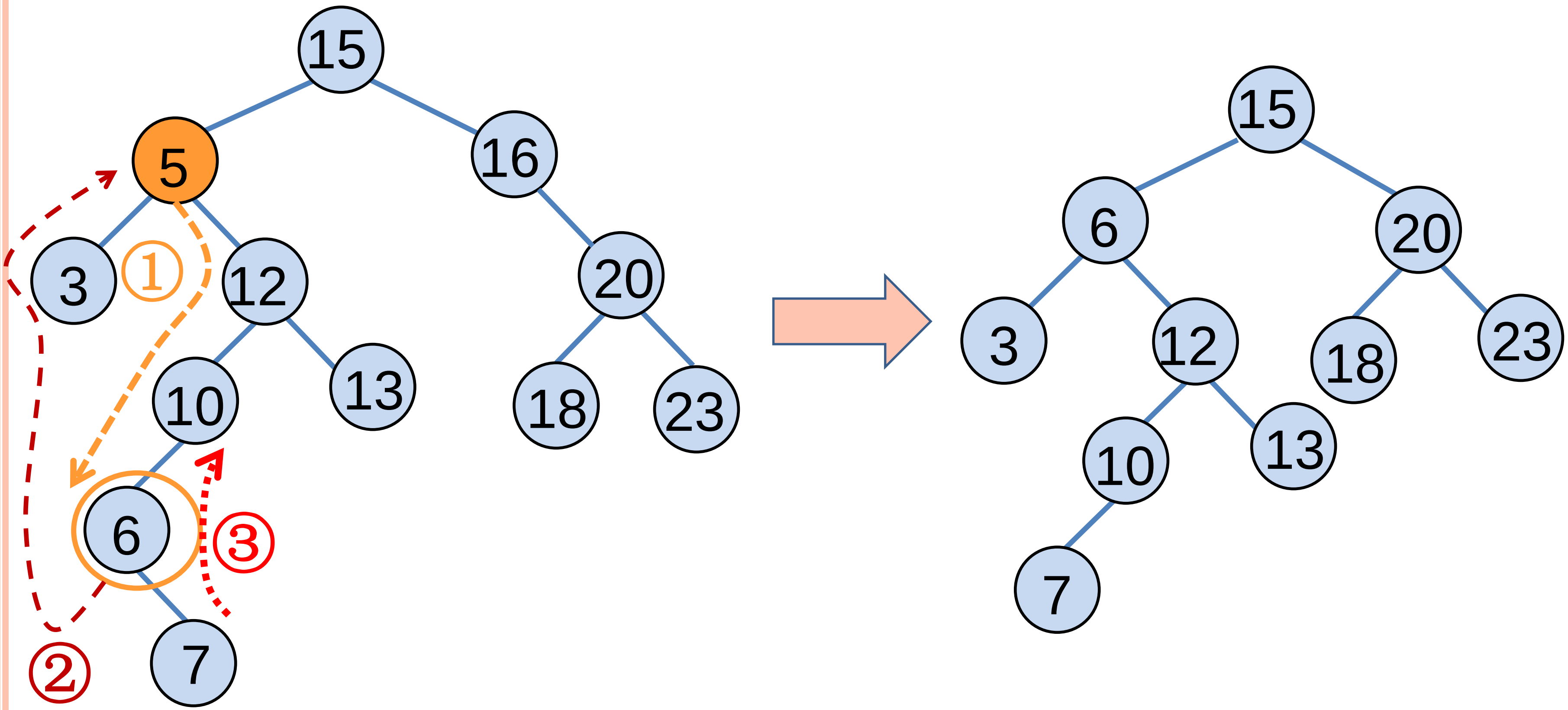
z Has No Children



z Has Only One Child



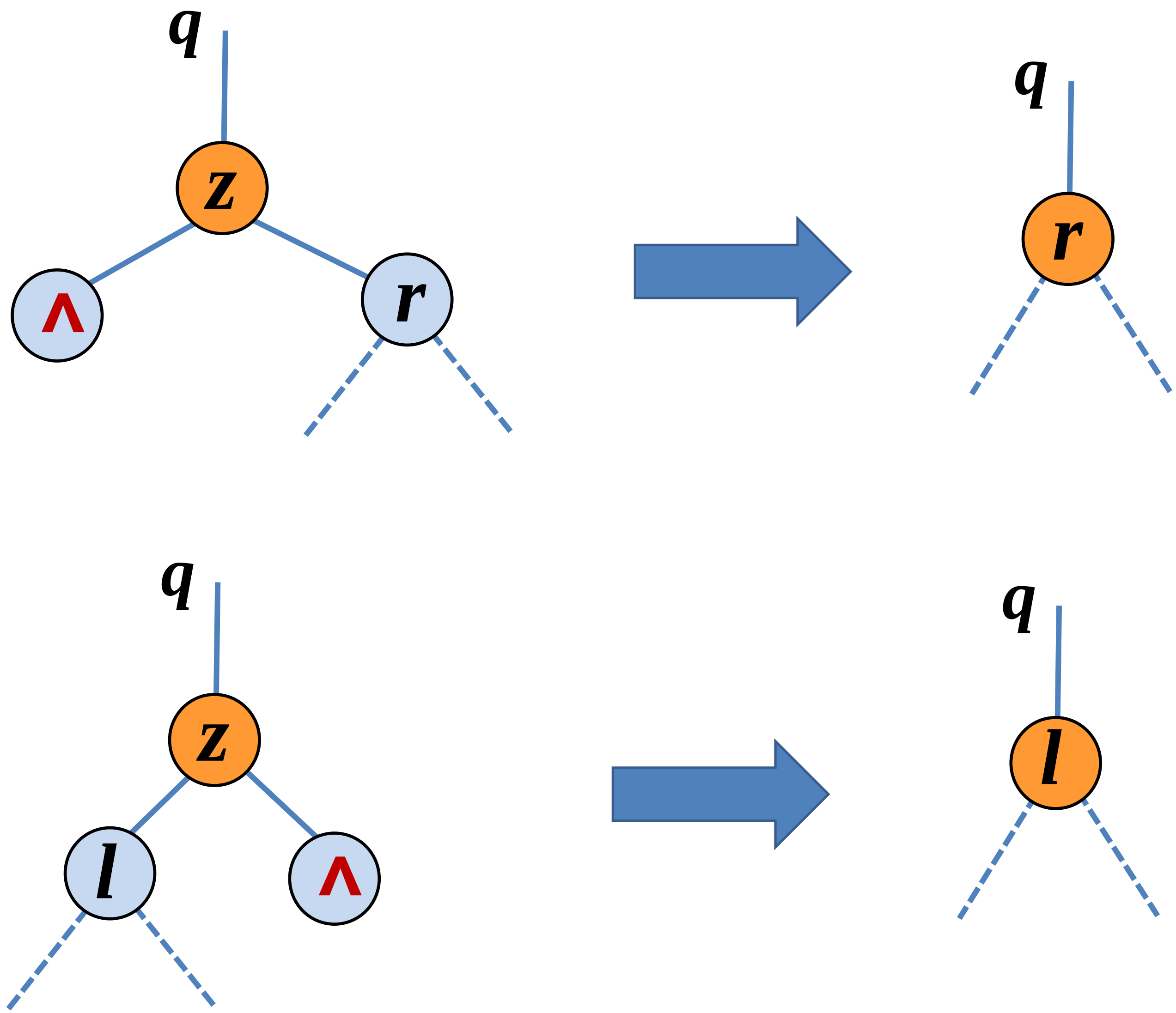
z Has Two Children



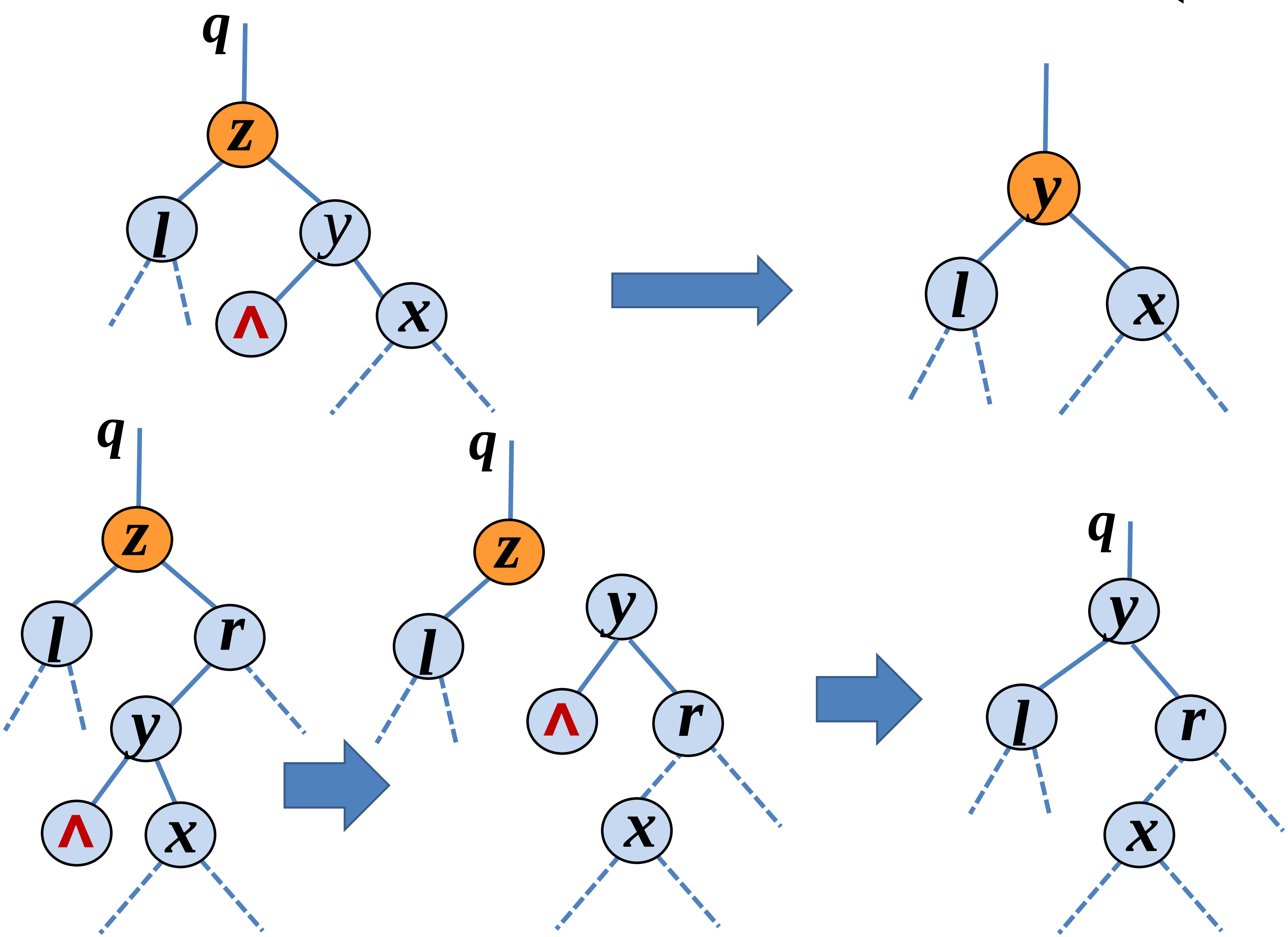
BST Operations: Delete

- *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.

BST Operations: Delete (more)



BST Operations: Delete (more)



Sorting With Binary Search Trees

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

3 1 8 2 6 7 5

1. By inserting nodes to build a BST
2. Inorder tree walk

SORTING WITH BINARY SEARCH TREES

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

BSTSort(*A*)

    for *i* = *1* to *n*

        TREEINSERT(*A*[*i*]);

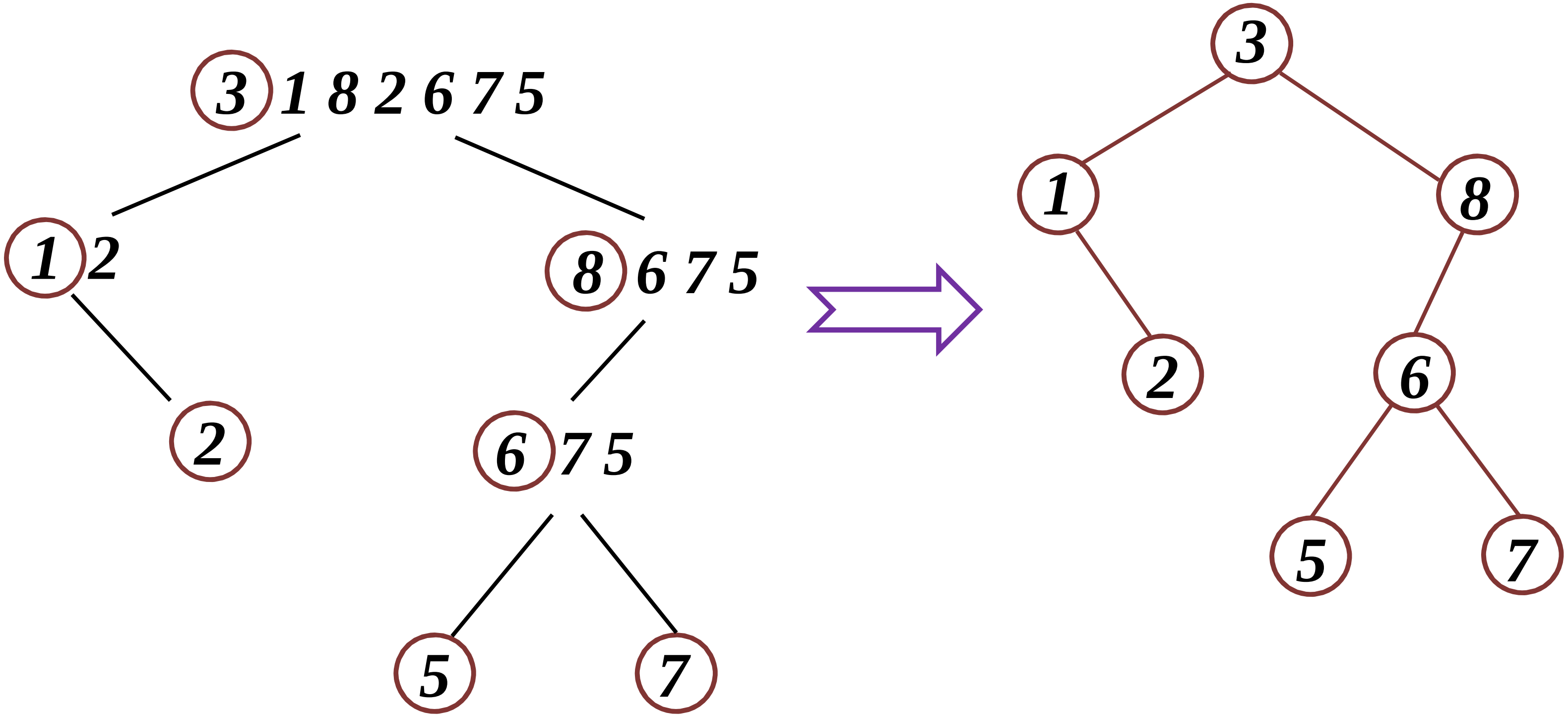
    InorderTreeWalk(root);

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

Sorting With BSTs

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

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



Sorting With BSTs

- Inserted nodes are similar to <u>partition pivot</u> 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?
  - **Quicksort**
  - **Sorts in place (no extra space)**
  - **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 the one that arises from inserting the keys in random order into an initially empty tree.
- 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 like *Successor*, *Predecessor*, *Search*, *Min*, *Max*, *Insert*, and *Delete* all have $O(\lg n)$ average complexity on randomly-built binary search trees.

NODE DEPTH

- The depth of a node = the number of comparisons made during TREE-INSERT. Assuming all input permutations are equally likely, we have
- **Average node depth**

$$= \frac{1}{n}\left[\sum_{i=1}^{n}(\# \textit{comparisons to insert node } i)\right]$$

$$= E\left(\sum_{i=1}^{n} d(x_i)\right) = \sum_{i=1}^{n} E\left(d(x_i)\right)$$

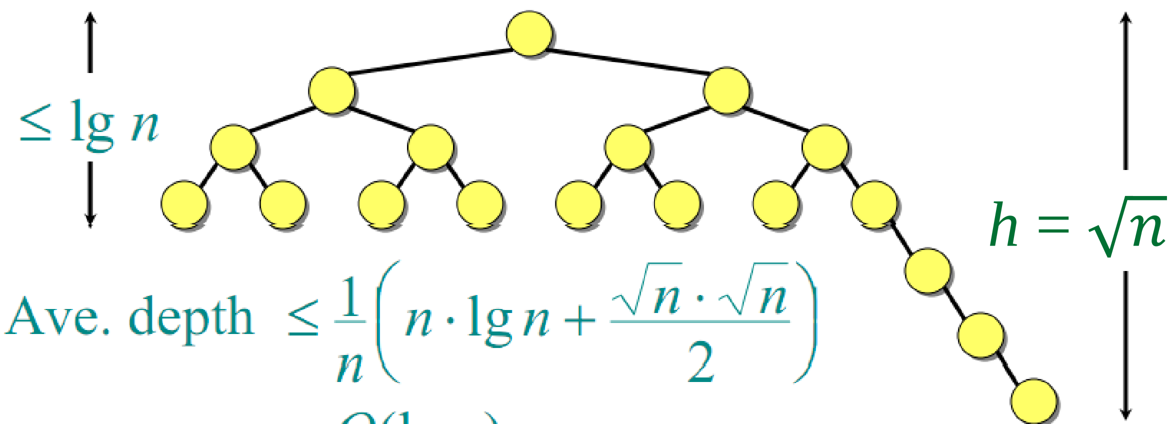
$$= \sum_{i=1}^{n}\frac{1}{n} d(x_i) = \frac{1}{n}\sum_{i=1}^{\log(n+1)} i * 2^{i-1} < \log(n+1) \qquad (\textit{quicksort analysis})$$

$$= O(\log n)$$

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.



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

$$= O(\lg n)$$

$$\sum_{i=1}^{\log n} i * 2^{i-1} - \sum_{i=1}^{\log n} i + \sum_{i=1}^{\sqrt{n}} i$$

$$\leq n \lg n + 1 - n + \frac{\sqrt{n}(\sqrt{n}+1)}{2} \quad \leq n(\lg n + \frac{n}{2})$$

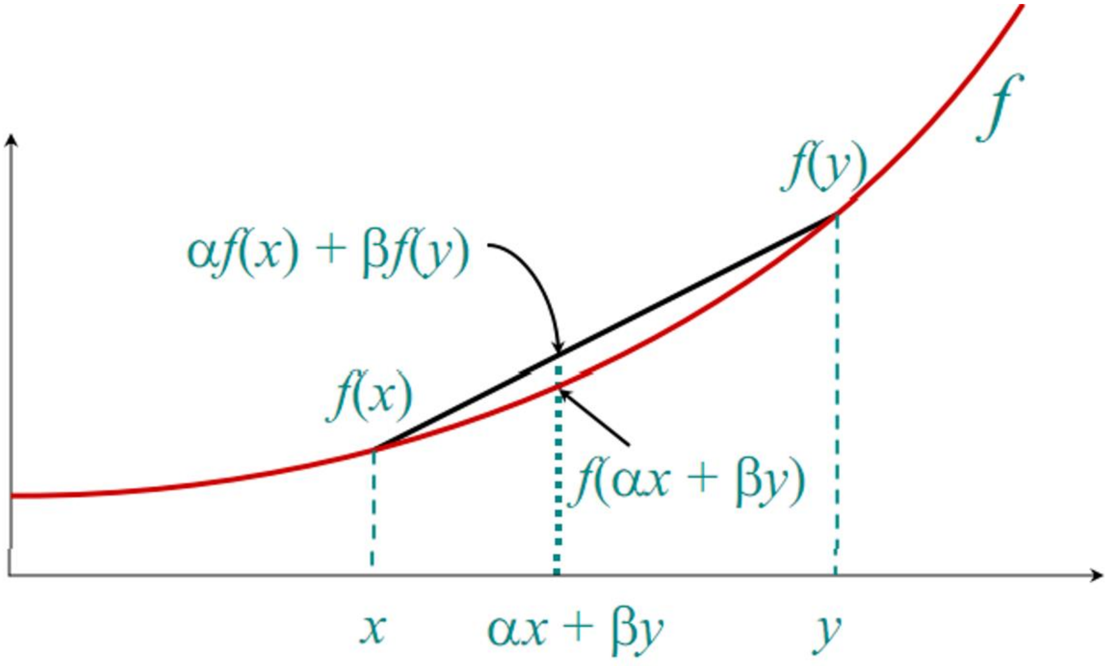
CONVEX FUNCTIONS

Jensen's Inequality:

- A function $f : \mathbb{R} \to \mathbb{R}$ 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 $x, y \in \mathbb{R}$.



Expected Tree Height of a Randomly Built BST

Outline of the analysis:

- Based on the Jensen's inequality, we can say that $f(E[X]) \le E[f(X)]$ for any convex function f and random variable X.

$$
\begin{aligned}
f(E[X]) &= f\left(\textstyle\sum_{k=-\infty}^{+\infty} k * P(x = k)\right) \\
&\le \textstyle\sum_{k=-\infty}^{+\infty} f(k) * P(x = k) \\
&= E[f(X)]
\end{aligned}
$$

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

$$
\text{s.t. } \alpha + \beta = 1
$$

Expected Tree Height of a Randomly Built BST

Outline of the analysis (Three main steps):

- Based on the Jensen's inequality, we can say that $f(E[X]) \leq E[f(X)]$ 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 $2^{E[X_n]} \leq E[2^{X_n}] = E[Y_n] = \mathrm{O}(n^3)$, and hance that $E[X_n] = O(\lg n)$.

CONVEXITY LEMMA

Lemma. Let $f : \mathbb{R} \to \mathbb{R}$ be a convex function $\boldsymbol{f}$, and let $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ be a set of nonnegative constants ([0,1]) such that $\Sigma_k \alpha_k = 1$. Then, for any set $\{x_1, x_2, \ldots, x_n\}$ of real numbers, we have

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

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

$$f(\alpha x + \beta y) \le \alpha f(x) + \beta f(y)$$
s.t. $\alpha + \beta = 1$

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:

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

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

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

$$= \sum_{k=1}^{n}\alpha_k f(x_k). \quad \square$$

Algebra.

Jensen's Inequality

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

Proof.

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

Definition of expectation.

JENSEN'S INEQUALITY

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

Proof.

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

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

Convex lemma (generalized)

JENSEN'S INEQUALITY

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

Proof.
$$f(E[X]) = f\left(\sum_{k=-\infty}^{\infty} k \bullet \Pr\{X = k\}\right)$$
$$\le \sum_{k=-\infty}^{\infty} f(k) \bullet \Pr\{X = k\}$$
$$= E[f(x)]$$

Tricky step, but true –think about it.

Analysis of BST Height

- Let X_n be the random variable denoting <u>the height</u> 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 **<u>rank *k*</u>**, 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

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

Analysis of BST Height (Continued)

- Define the indicator random variable Z_{nk} as

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

$$Z_{nk} = \mathrm{I}(X_n = k)$$

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

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

Exponential Height Recurrence

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

Take expectation of both sides

Exponential Height Recurrence

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

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

Linearity of Expectation.

Exponential Height Recurrence

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

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

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

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

Exponential Height Recurrence

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

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

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

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

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

Exponential Height Recurrence

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

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

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

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

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

Each term appears twice and reindex.

SOLVING THE RECURRENCE

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

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

Solving The Recurrence

- Use substitution to show the $E[Y_n] \leq cn^3$ for some positive constant $\boldsymbol{c}$, which we can pick sufficiently large to handle the initial conditions (**inductive**).

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

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

Substitution.

SOLVING THE RECURRENCE

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

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

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

Integral method.

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

Solving The Recurrence

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

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

Solve the Integral.

Solving The Recurrence

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

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

Algebra.

The Grand Finale

- Putting it all together, and 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, and we have

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

Definition.

THE GRAND FINALE

- Putting it all together, and 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.

- Taking the lg of both sides yields

$$E[Y_n] \leq 3 \lg n + O(1).$$