CS 146: Data Structures and Algorithms

B-Trees

- A B-tree is a tree data structure suitable for disk drives.
  - It may take up to 11 ms to access data on disk.
  - Today's modern CPUs can execute billions of instructions per second.
  - Therefore, it's worth spending a few CPU cycles to reduce the number of disk accesses.
- B-trees are often used to implement databases.

Memory/Storage Speed Comparisons

- Suppose your computer ran at human speeds.
  - 1 CPU cycle: 1 second
- Then the time to retrieve one byte from:
  - SRAM
    - 5 seconds
  - DRAM
    - 2 minutes
  - Flash
    - 1 day
  - Hard drive
    - 2 months
  - Tape
    - 1,000 years

B-Trees, *cont'd*

- A B-tree is an m-ary tree.

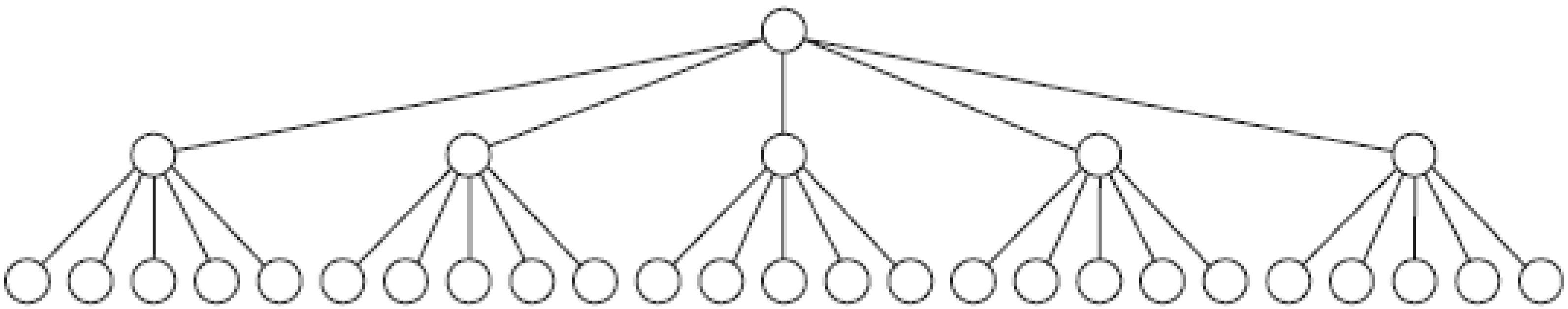

Figure 4.59 5-ary tree of 31 nodes has only three levels

B-Trees, *cont'd*

- A B-tree of order 5 for a disk drive:

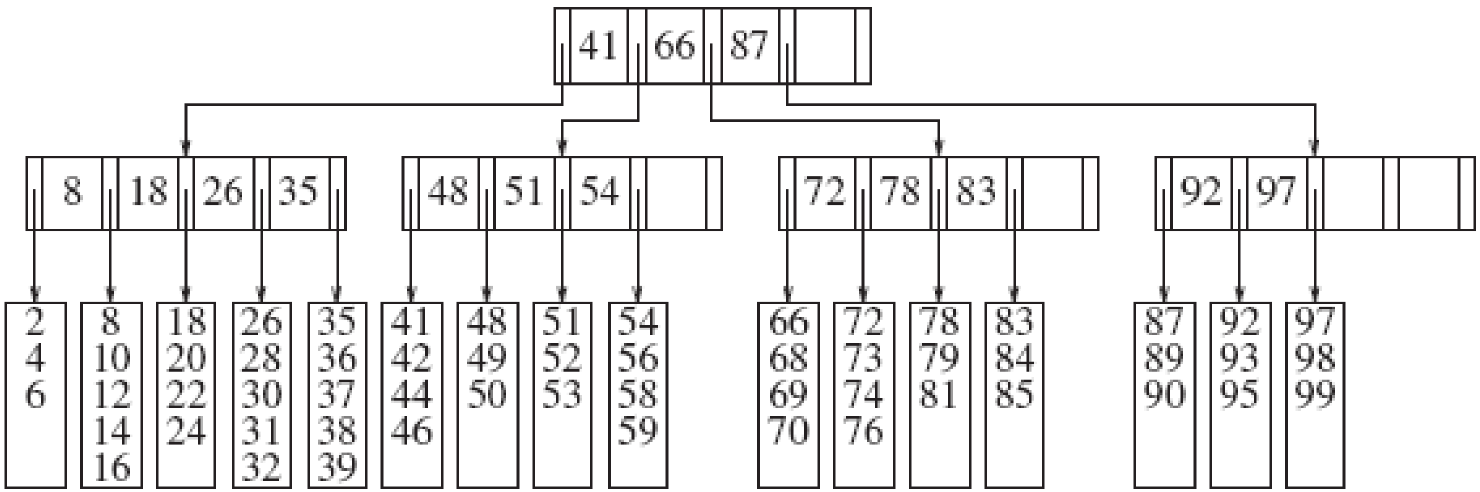

Figure 4.60 B-tree of order 5

B-Tree Insertion of 57

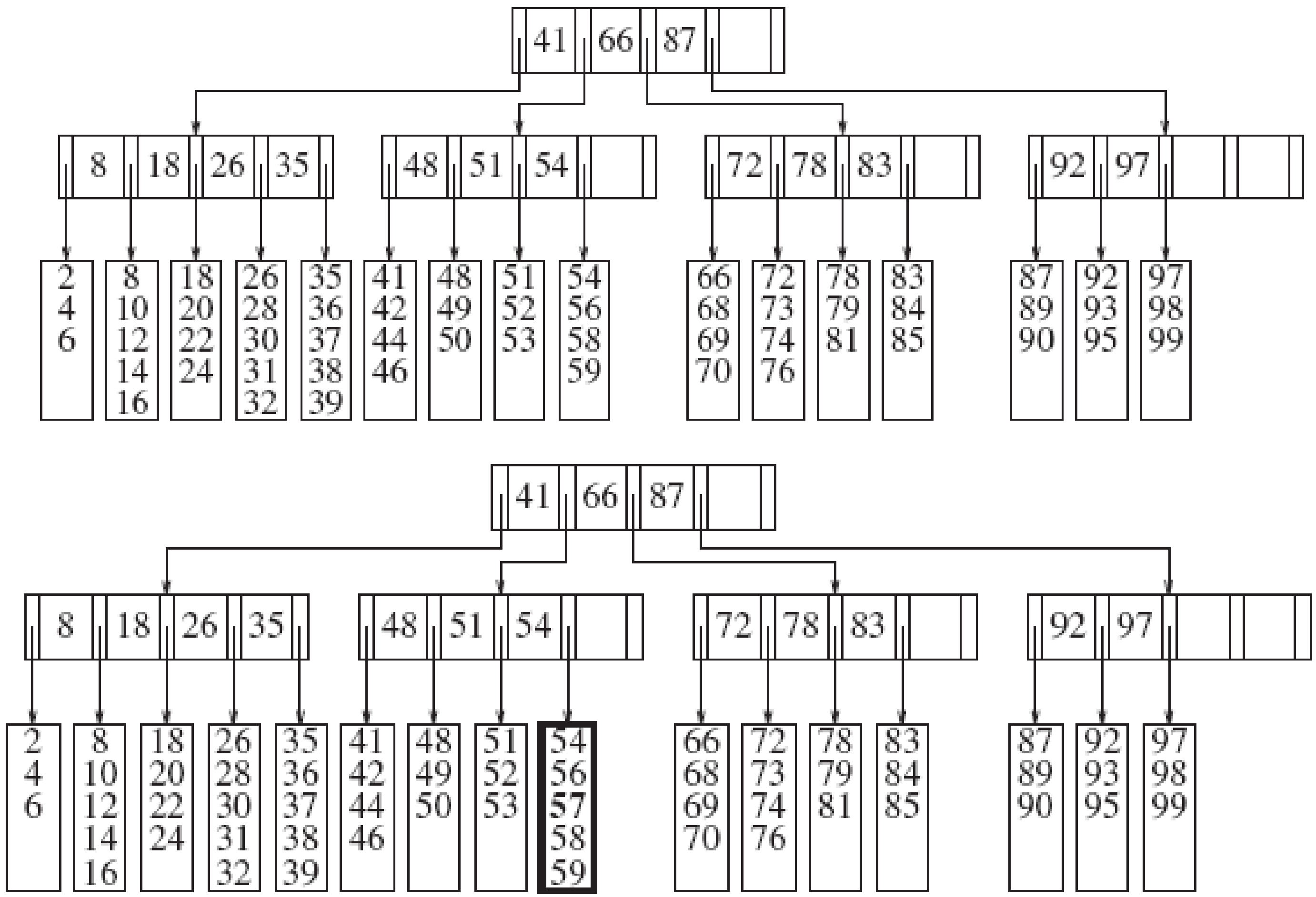

Figure 4.61 B-tree after insertion of 57 into the tree in Figure 4.60

Data Structures and Algorithms in Java, 3rd ed.
by Mark Allen Weiss
Pearson Education, Inc., 2012

B-Tree Insertion of 55

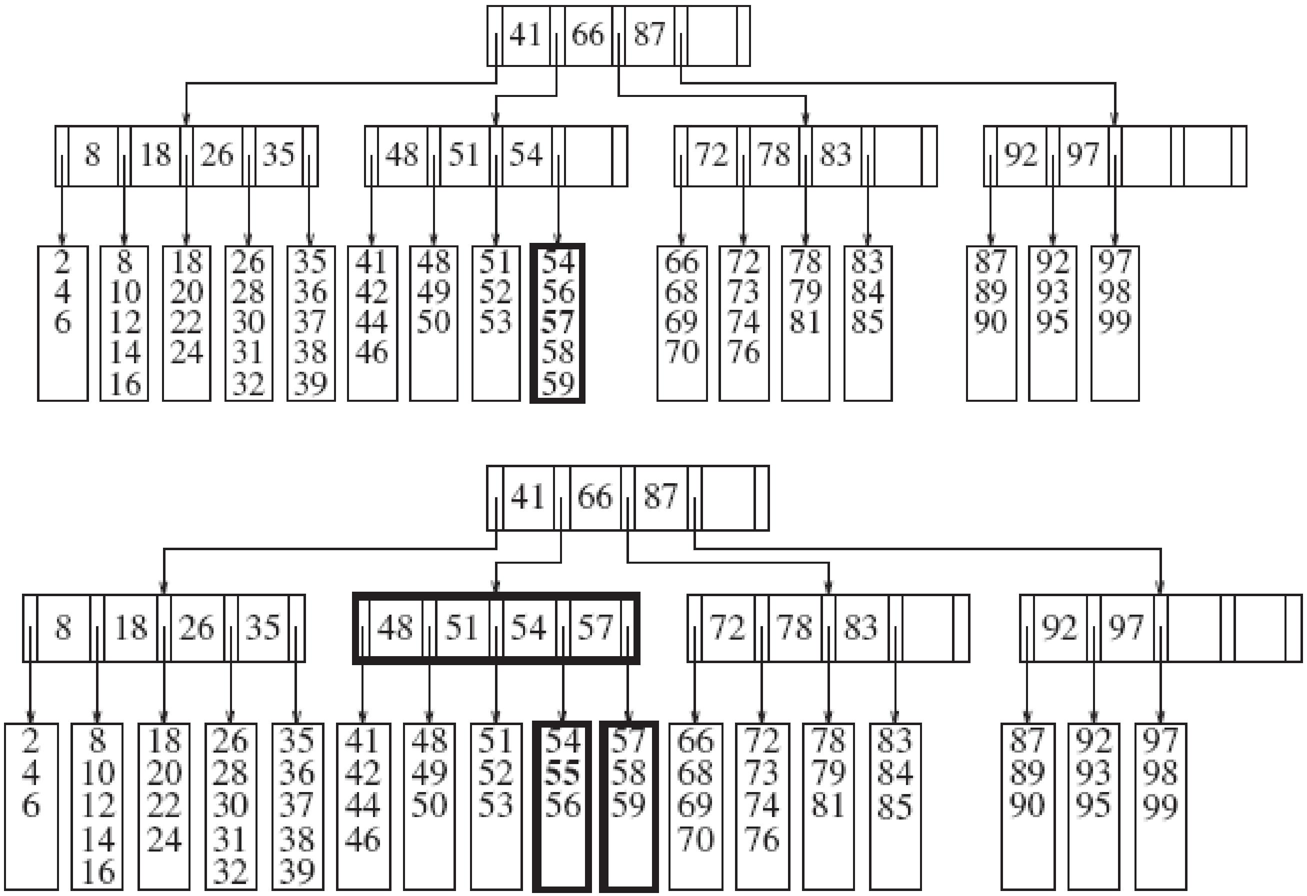

Figure 4.62 Insertion of 55 into the B-tree in Figure 4.61 causes a split into two leaves

B-Tree Insertion of 40

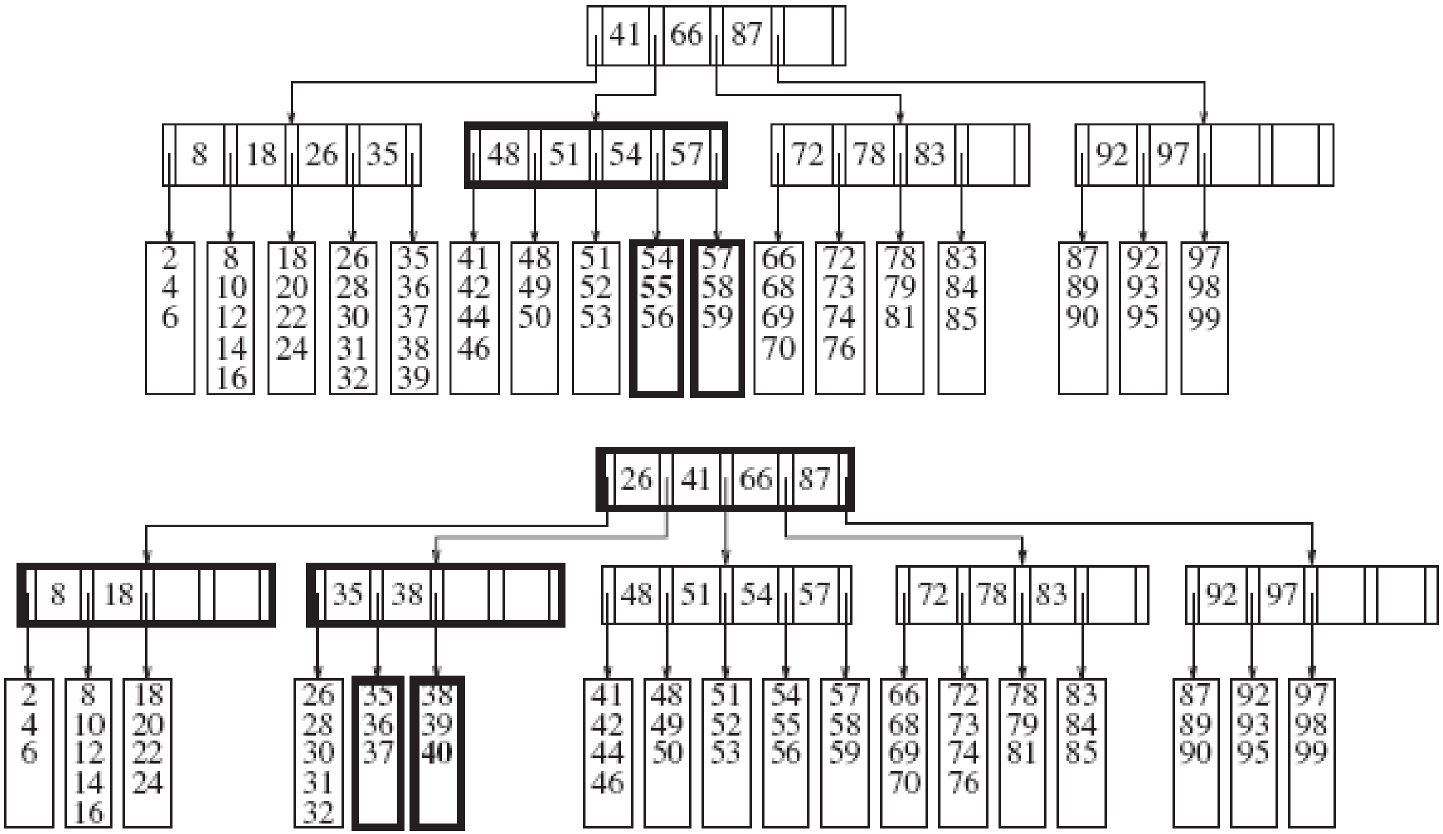

Figure 4.63 Insertion of 40 into the B-tree in Figure 4.62 causes a split into two leaves and then a split of the parent node

Data Structures and Algorithms in Java, 3rd ed.
by Mark Allen Weiss
Pearson Education, Inc., 2012

B-Tree Deletion of 99

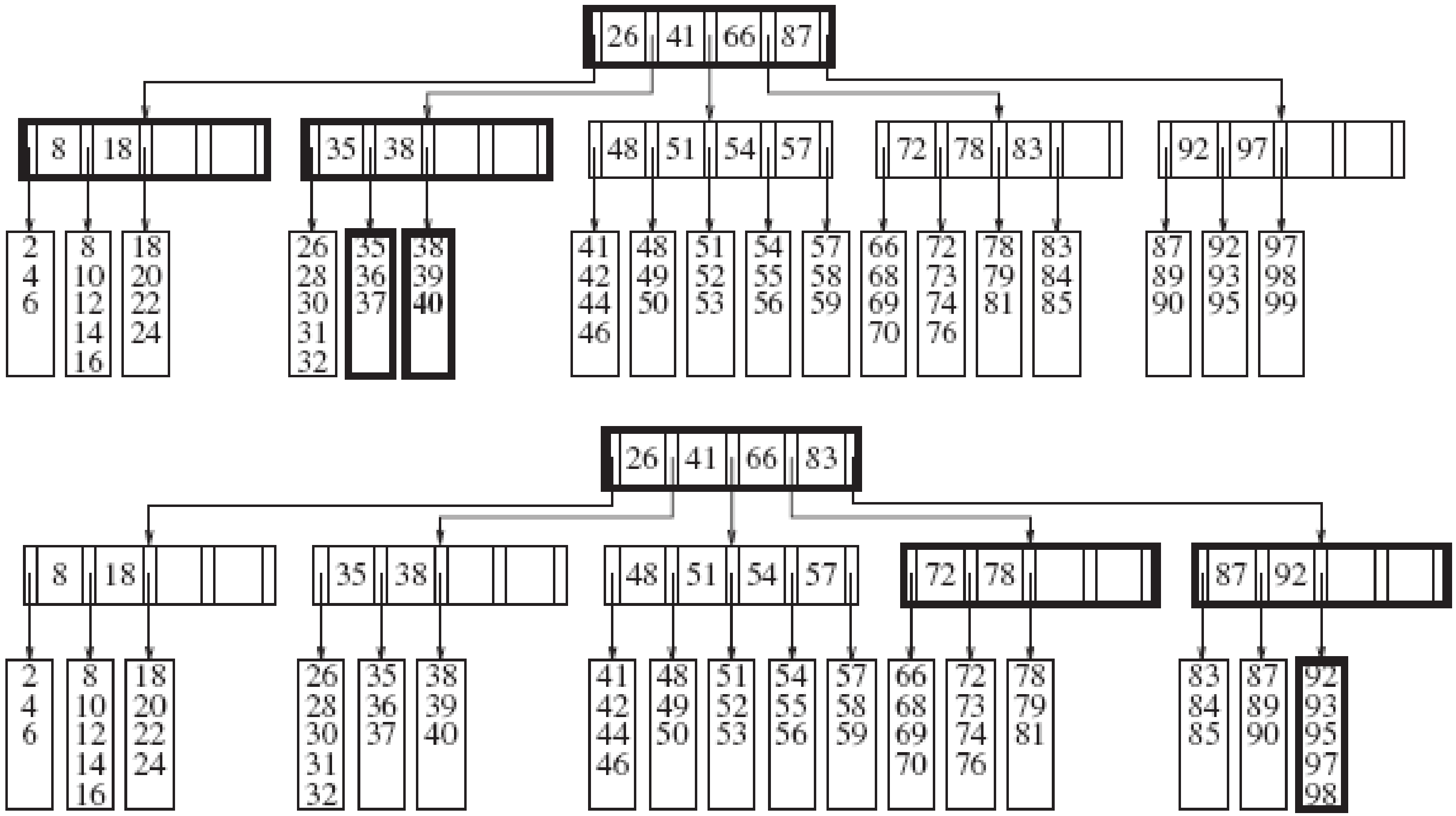

Figure 4.64 B-tree after the deletion of 99 from the B-tree in Figure 4.63

How to Eliminate a Summation

- How to solve $S = \sum_{i=0}^{\infty} A^i$?

$$S = \sum_{i=0}^{\infty} A^i = 1 + A + A^2 + A^3 + A^4 + A^5 + \ldots$$ (a)

$$AS = \sum_{i=1}^{\infty} A^i = A + A^2 + A^3 + A^4 + A^5 + \ldots$$ (b)

Subtract (a) – (b).

$$S - AS = 1$$

$$S(1 - A) = 1$$

$$S = \frac{1}{1 - A}$$

A Harmonic Number

$$\sum_{i=3}^{N+1} \frac{1}{i} \approx \log_e N$$

- See page 5 of the textbook!

OK, But Can You Prove It?

- Intuitively, we know that an operation on a binary search tree (search, insert, delete) of N nodes should take $O(\log N)$ time.

- For a specific node at depth d, each operation should take $O(d)$ time

Remember that logs
in computer science
are base 2 by default.

- Therefore, we can prove that the ave running time of a BST operation is $O(\log N)$ if we can prove that the average depth over all the nodes of a BST is $O(\log N)$.

Proof that the Average Depth is $O(\log N)$

- Internal path length $D(N)$ is the sum of the depths of all the nodes of a tree with N nodes.
  - $D(1) = 0$

- A BST of N nodes has a left subtree containing i nodes and a right subtree of $N - i - 1$ nodes for $0 \leq i < N$.
  - So we have the recurrence relation

$$D(N) = D(i) + D(N - i - 1) + (N - 1)$$

  - We add the $(N - 1)$ to account that each node in the two subtrees is 1 deeper, and there are $N - 1$ such nodes. The root of the tree is at depth 0.

Proof that the Average Depth is $O(\log N)$

$$D(N) = D(i) + D(N - i - 1) + (N - 1)$$

- We can assume that all subtree sizes in a BST are equally likely. Then the average value of both $D(i)$ and $D(N-i-1)$ is each $\frac{1}{N}\left[\sum_{j=0}^{N-1} D(j)\right]$
  - We make this substitution twice, and our recurrence relation becomes:

$$D(N) = \begin{cases} 0 & N = 1 \\ \frac{2}{N}\left[\sum_{j=0}^{N-1} D(j)\right] + (N - 1) & N > 1 \end{cases}$$

Proof that the Average Depth is $O(\log N)$

- To solve: $$D(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} D(j)\right] + (N-1)$$

Drop the insignificant -1 and multiply both sides by N.

$$ND(N) = 2\left[\sum_{j=0}^{N-1} D(j)\right] + N^2$$

How can we eliminate the summation?

Proof that the Average Depth is $O(\log N)$

- To solve: $$D(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} D(j)\right] + (N-1)$$

Drop the insignificant -1 and multiply both sides by N.

$$ND(N) = 2\left[\sum_{j=0}^{N-1} D(j)\right] + N^2 \qquad \text{(a)}$$

First replace N by N-1.

$$(N-1)D(N-1) = 2\left[\sum_{j=0}^{N-2} D(j)\right] + (N-1)^2 \qquad \text{(b)}$$

Then subtract (a) - (b). Remember that $(N\text{-}1)^2 = N^2 - 2N + 1$.

Drop the insignificant +1.

$$ND(N) - (N-1)D(N-1) = 2D(N-1) + 2N$$

Rearrange terms.

$$ND(N) = (N+1)D(N-1) + 2N$$

Proof that the Average Depth is $O(\log N)$

$$ND(N) = (N+1)D(N-1) + 2N$$

Divide through by $N(N+1)$.

$$\frac{D(N)}{N+1} = \frac{D(N-1)}{N} + \frac{2}{N+1}$$

Telescop
e.

$$\frac{D(N-1)}{N} = \frac{D(N-2)}{N-1} + \frac{2}{N}$$

$$\frac{D(N-2)}{N-1} = \frac{D(N-3)}{N-2} + \frac{2}{N-1}$$

$$\vdots$$

$$\frac{D(2)}{3} = \frac{D(1)}{2} + \frac{2}{3}$$

Add together.
Many
convenient
cancellations
of
terms will
occur.

Proof that the Average Depth is $O(\log N)$

$$\frac{D(N)}{N+1} = \frac{D(1)}{2} + 2\sum_{i=3}^{N+1}\frac{1}{i}$$

But $\sum_{i=3}^{N+1}\frac{1}{i} \approx \log_e N$ (a *harmonic number*, see p.5 of the textbook).

$$\frac{D(N)}{N+1} = O(\log N)$$

$$D(N) = O(N\log N)$$

We started by solving:

$$D(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} D(j)\right] + (N-1)$$

And so

$$\frac{D(N)}{N} = O(\log N)$$

Proof that the Average Depth is $O(\log N)$

$$\frac{D(N)}{N} = O(\log N)$$

- Therefore, we've successfully proven that if $D(N)$ is the sum of the depths of all N nodes in a BST, then the average depth of a node is $O(\log N)$.
- And therefore, a BST operation should take on average $O(\log N)$ time.

Break

The Priority Queue ADT

- A priority queue ADT is
  - Similar to a queue, except that
  - Items are removed from the queue in priority order.
- If lower-numbered items have higher priority, then the operations on a priority queue are:
  - **Insert:** Enqueue an item.
  - **Delete minimum:** Find and remove the minimum-valued (highest priority) item from the queue.

Priority Queue Implementation

- Unsorted list
  - Insert: Insert at the end of the list.
  - Delete minimum: Scan the list to find the minimum.
- Sorted list
  - Insert: Insert in the proper position to maintain order.
  - Delete minimum: Delete from the head of the list.
- Binary tree
  - Inserts and deletes take $O(\log N)$ time on average.
- Binary heap
  - Inserts and deletes take $O(\log N)$ worst-case time.
  - No links required!

Binary Heap

- A binary heap (or just heap) is a binary tree that is complete.
  - All levels of the tree are full except possibly for the bottom level which is filled from left to right:

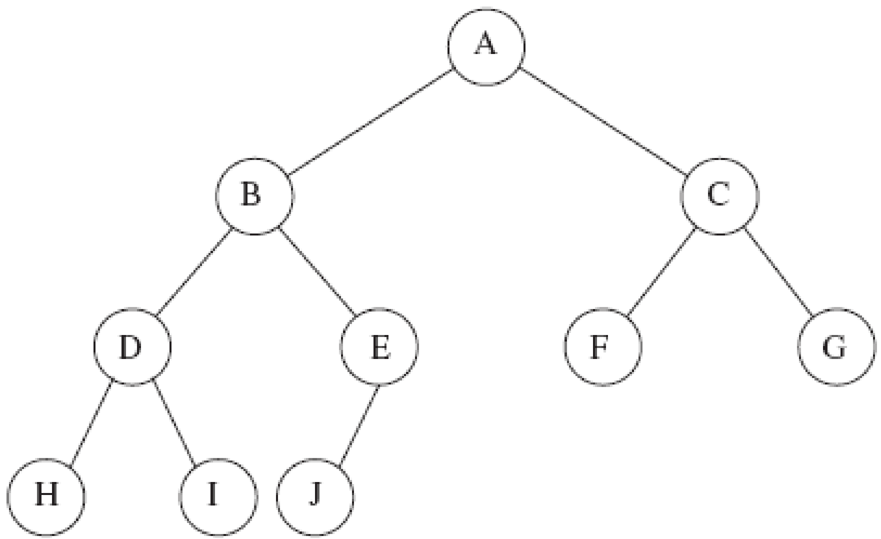

Figure 6.2 A complete binary tree

Data Structures and Algorithms in Java, 3rd ed.
by Mark Allen Weiss
Pearson Education, Inc., 2012

Binary Heap

- Conceptually, a heap is a binary tree.
- But we can implement it as an array.
- For any element in array position i:
  - Left child is at position $2i$
  - Right child is at position $2i + 1$
  - Parent is at position $\lfloor i/2 \rfloor$

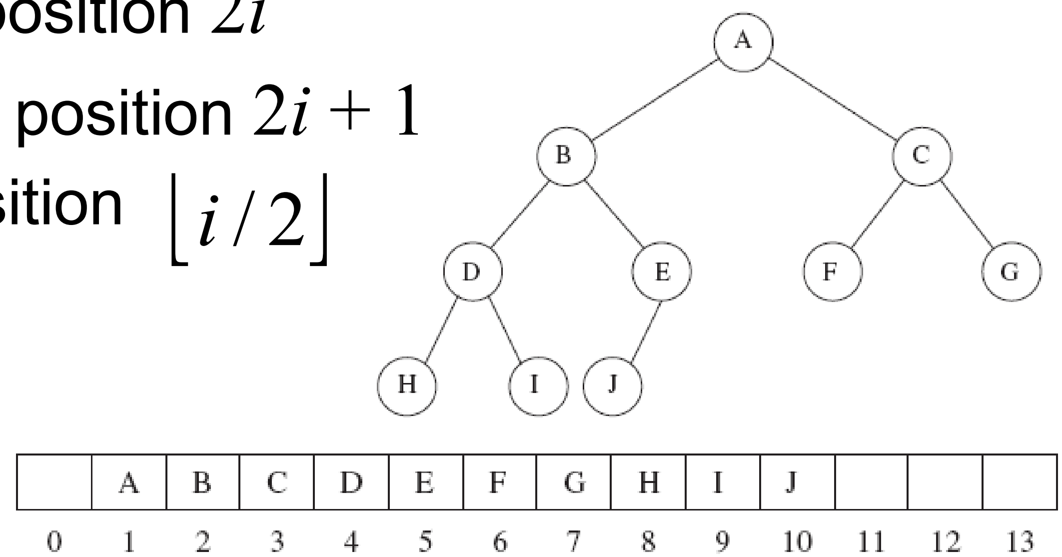

Figure 6.3 Array implementation of complete binary tree

Data Structures and Algorithms in Java, 3rd ed.
by Mark Allen Weiss
Pearson Education, Inc., 2012

Heap-Order Priority

- We want to find the minimum value (highest priority) value quickly.
- Make the minimum value always at the root.
  - Apply this rule also to roots of subtrees.
- Weaker rule than for a binary search tree.
  - Not necessary that values in the left subtree be less than the root value and values in the right subtree be greater than the root value.

Heap-Order Priority

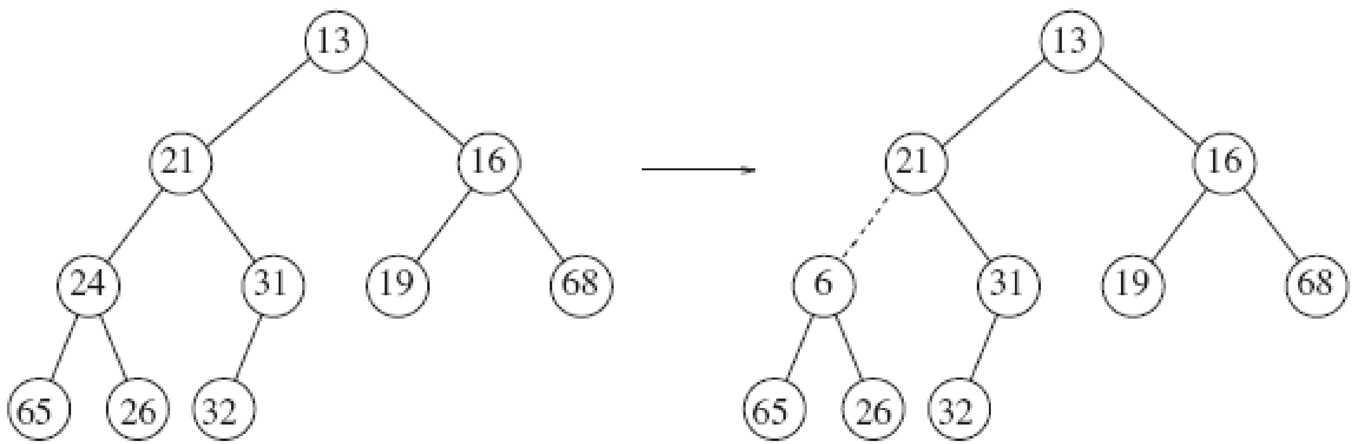

Figure 6.5 Two complete trees (only the left tree is a heap)

Data Structures and Algorithms in Java, 3rd ed.
by Mark Allen Weiss
Pearson Education, Inc., 2012

Heap Insertion

- Create a hole in the next available position at the bottom of the (conceptual) binary tree.
  - The tree must remain complete.
  - The hole is at the end of the implementation array.
- While the heap order is violated:
  - Slide the hole's parent into the hole.
  - "Bubble up" the hole towards the root.
  - The new value percolates up to its correct position.
- Insert the new value into the correct position.

Heap Insertion

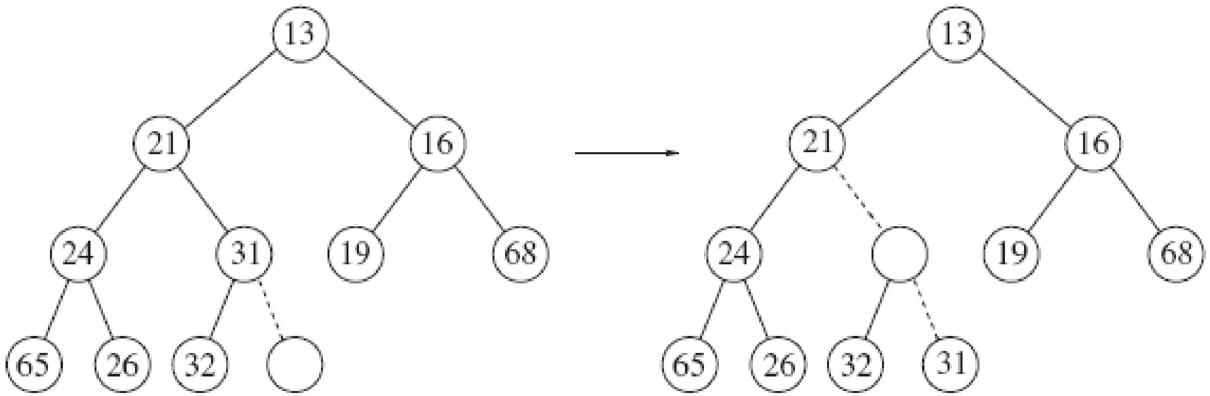

Figure 6.6 Attempt to insert 14: creating the hole and bubbling the hole up

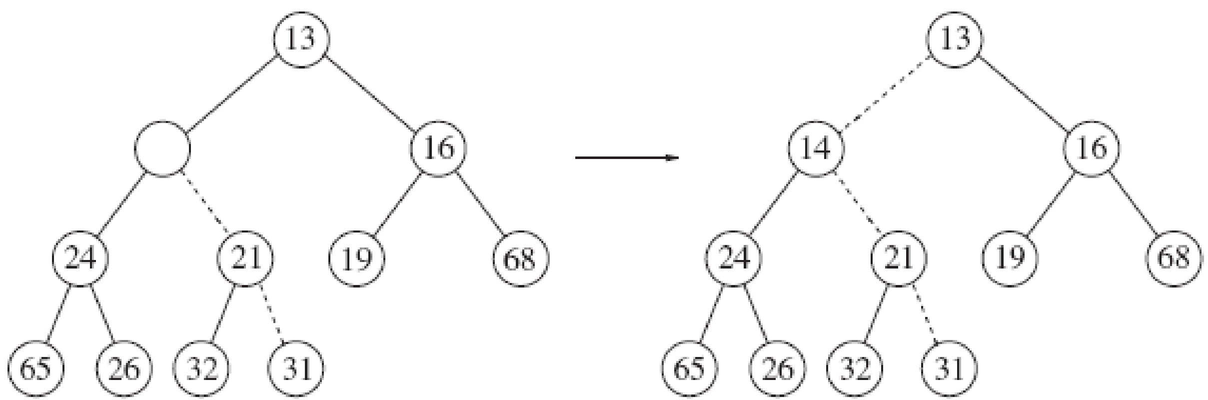

Figure 6.7 The remaining two steps to insert 14 in previous heap

Data Structures and Algorithms in Java, 3rd ed.
by Mark Allen Weiss
Pearson Education, Inc., 2012

Heap Insertion

```
public void insert(AnyType x)
{
    if (currentSize == array.length - 1) {
        enlargeArray(array.length*2 + 1);
    }

    // Percolate up.
    int hole = ++currentSize;
    for (array[0] = x; x.compareTo(array[hole/2]) < 0; hole
/= 2) {
        array[hole] = array[hole/2];
    }
    array[hole] = x;
}
```

Heap Deletion

- Delete the root node of the (conceptual) tree.
  - A hole is created at the root.
  - The tree must remain complete.
  - Put the last node of the heap into the hole.
- While the heap order is violated:
  - The hole percolates down.
  - The last node moves into the hole at the correct position.

Heap Deletion

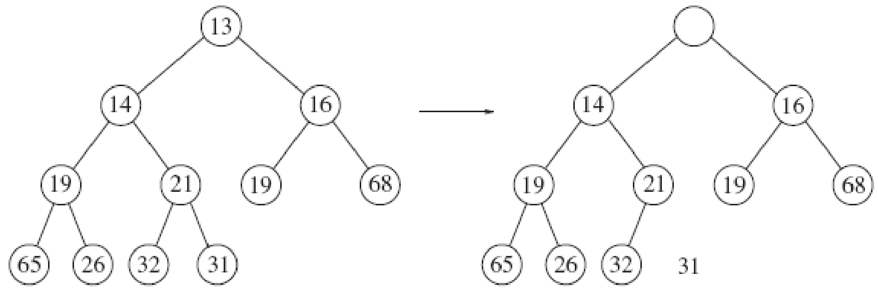

Figure 6.9 Creation of the hole at the root

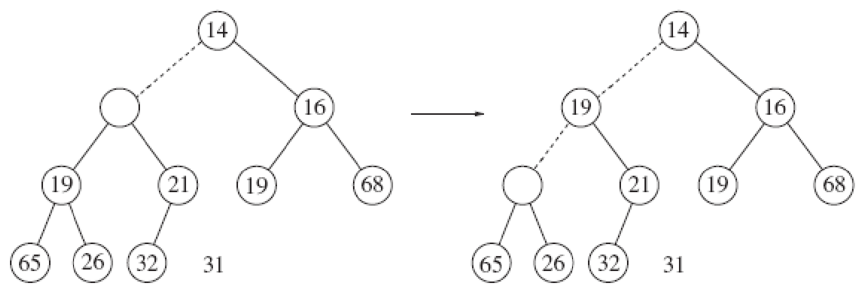

Figure 6.10 Next two steps in `deleteMin`

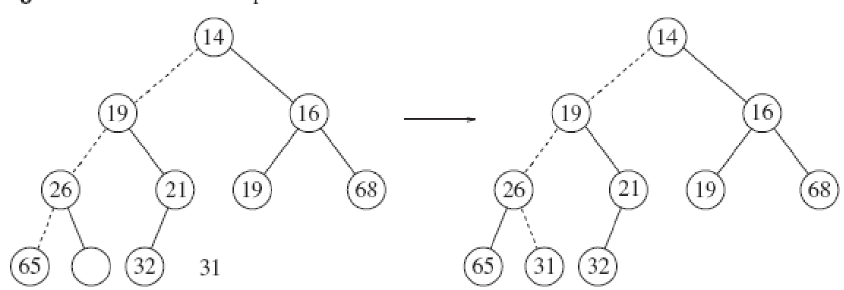

Figure 6.11 Last two steps in `deleteMin`

Data Structures and Algorithms in Java, 3rd ed.
by Mark Allen Weiss
Pearson Education, Inc., 2012

Heap Deletion

```
public AnyType deleteMin() throws Exception
{
    if (isEmpty()) throw new Exception();

    AnyType minItem = findMin();
    array[1] = array[currentSize

    percolateDown(1);
    return minItem;
}
```

It's the root node.

ast value temporarily into the root.

Heap Deletion

```
private void percolateDown(int hole)
{
    int child;
    AnyType tmp = array[hole];

    for (; hole*2 <= currentSize; hole = child) {
        child = hole*2;
        if (    (child != currentSize)
            && (array[child + 1].compareTo(array[child]))
< 0) {
            child++;
        }

        if (array[child].compareTo(tmp) <
            array[hole] = array[child];
        }
        else {
            break;
        }
    }
    array[hole] = tmp;
```

Percolate the root hole down.

Does the last value fit?

Heap Animation

```
appletviewer Chap12/Heap/Heap.html
```