# Data Structures
# Lesson 13

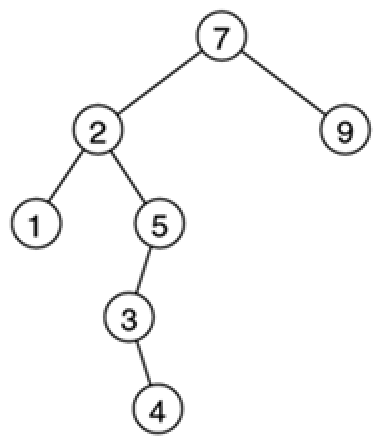
BSc in Computer Science
University of New York, Tirana

Assoc. Prof. Marenglen Biba

# Sample exam

# Question 1

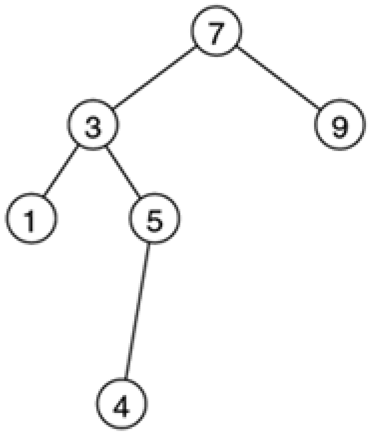
- Consider the binary search tree below:




- Sketch the tree after the node 2 is deleted.

# Solution 1

- Replace the item in node 2 with the smallest item in the right subtree and then remove that node



# Question 2

- Sketch in pseudo-code an algorithm for finding the minimum element in a binary search tree.

# Solution 2

```
protected BinaryNode<AnyType> findMin( BinaryNode<AnyType> t )
{
    if( t != null )
        while( t.left != null )
            t = t.left;

    return t;
}
```

# Question 3

- Sketch in pseudo-code an algorithm for inserting an element in a binary search tree.

# Solution 3

```
protected BinaryNode<AnyType> insert( AnyType x, BinaryNode<AnyType> t )
{
    if( t == null )
        t = new BinaryNode<AnyType>( x );
    else if( x.compareTo( t.element ) < 0 )
        t.left = insert( x, t.left );
    else if( x.compareTo( t.element ) > 0 )
        t.right = insert( x, t.right );
    else
        throw new DuplicateItemException( x.toString( ) );  // Duplicate
    return t;
}
```

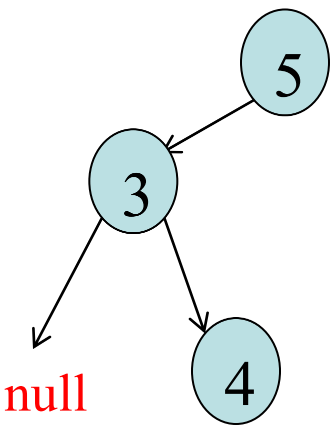
# Question 4

- Sketch in pseudo-code an algorithm for removing the minimum element in a binary search tree.

# Solution 4

**figure 19.11**

The `removeMin` method for the `BinarySearchTree` class




```
/**
 * Internal method to remove minimum item from a subtree.
 * @param t the node that roots the tree.
 * @return the new root.
 * @throws ItemNotFoundException if t is empty.
 */
protected BinaryNode<AnyType> removeMin( BinaryNode<AnyType> t )
{
    if( t == null )
        throw new ItemNotFoundException( );
    else if( t.left != null )
    {
        t.left = removeMin( t.left );
        return t;
    }
    else
        return t.right;
}
```

Recursive call to left

*Delete the minimum.* We go left until finding a node that that has a null left link and then replace the link to that node by its right link. The symmetric method works for delete the maximum.

# Question 5

- Give the definition of an AVL tree. Sketch an AVL tree. Sketch the same tree with a change such that it is not any more an AVL tree.

# Solution 5

- Definition: An AVL tree is a binary search tree with the additional balance property that, for any node in the tree, the height of the left and right subtrees can differ by at most 1. As usual, the height of an empty subtree is -1.

# Solution 5

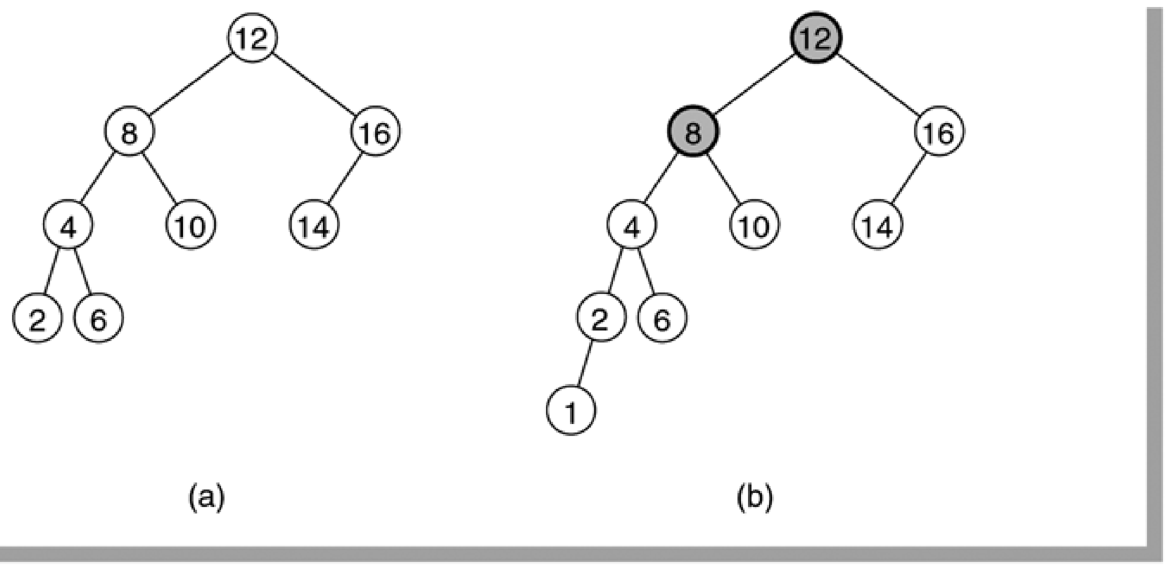



**figure 19.21**

Two binary search trees: (a) an AVL tree; (b) not an AVL tree (unbalanced nodes are darkened)

# Question 6

- Why a binary search tree can degenerate to a linked list? What is the average time required to search in this case?

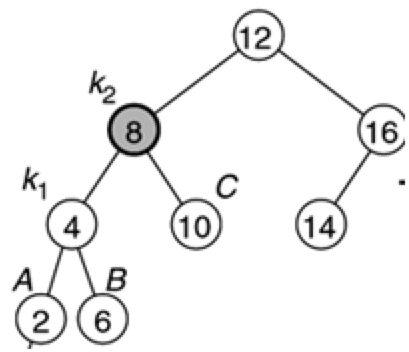
# Solution 6

- Why a binary search tree can degenerate to a linked list? What is the average time required to search in this case?

- Due to an unbalanced tree.
- O(N).

# Question 7

- Suppose we have the following tree:




- We want to keep the tree as an AVL tree. Sketch the tree after the insertion of the element 1.

# Solution 7: rotation

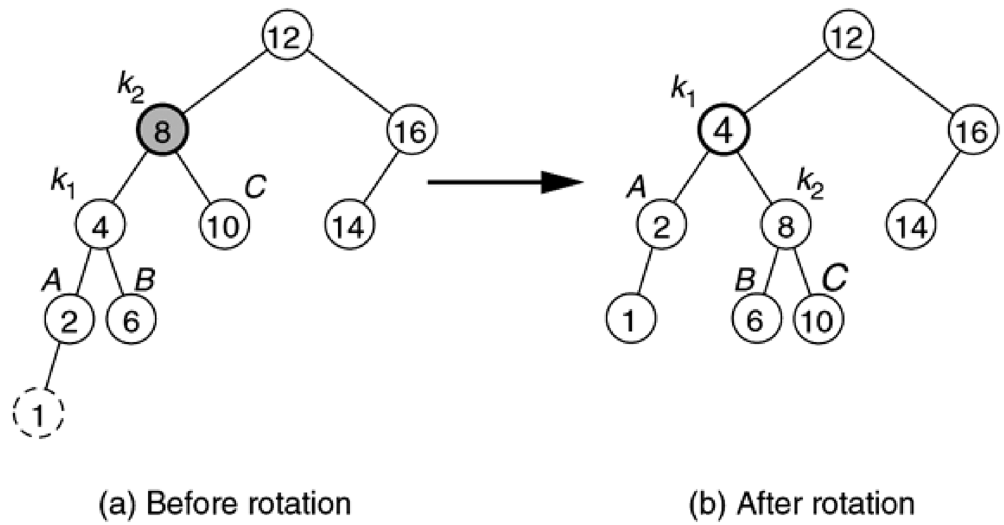
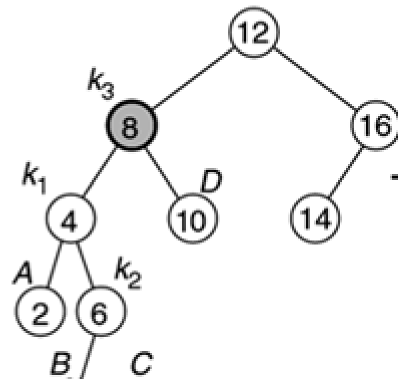



**figure 19.25**

Single rotation fixes an AVL tree after insertion of 1.

# Question 8

- Consider the following tree:




- We want to keep the tree as an AVL tree. Sketch the tree after the insertion of the element 5.

# Solution 8

- Left-right double rotation

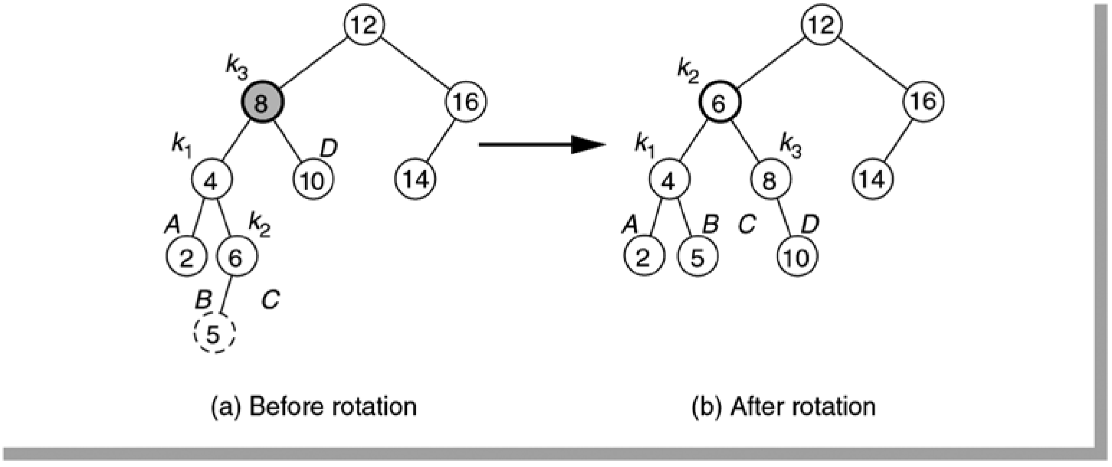



**figure 19.30**

Double rotation fixes AVL tree after the insertion of 5.

# Question 9

Primary clustering occurs in

a. linear probing

b. quadratic probing

c. separate chaining

d. all of the above

e. none of (a), (b), and (c)

# Solution 9

Primary clustering occurs in

a. linear probing

b. quadratic probing

c. separate chaining

d. all of the above

e. none of (a), (b), and (c)

# Question 10

- Given the input {4371, 1323, 6173, 4199, 4344, 9679, 1989}, a fixed table size of 10, and a hash function H(X) = X mod 10, show the resulting
  - a. Linear probing hash table
  - b. Quadratic probing hash table
  - c. Separate chaining hash table

# Solution 10

The hash tables are shown in Figure 20.1.

| | | | | | |
|---|---|---|---|---|---|
| 0 | 9679 | 0 | 9679 | 0 | |
| 1 | 4371 | 1 | 4371 | 1 | 4371 |
| 2 | 1989 | 2 | | 2 | |
| 3 | 1323 | 3 | 1323 | 3 | 6173, 1323 |
| 4 | 6173 | 4 | 6173 | 4 | 4344 |
| 5 | 4344 | 5 | 4344 | 5 | |
| 6 | | 6 | | 6 | |
| 7 | | 7 | | 7 | |
| 8 | | 8 | 1989 | 8 | |
| 9 | 4199 | 9 | 4199 | 9 | 1989, 9679, 4199 |

Figure 20.1, linear probing, quadratic probing, and separate chaining

# Question 11

- Which of the following algorithms solves the unweighted single-source shortest path problem?
  - a. breadth first search
  - b. Dijkstra’s algorithm
  - c. Kruskal’s algorithm
  - d. all of the above
  - e. none of (a), (b), and (c)

# Question 11

- Which of the following algorithms solves the unweighted single-source shortest path problem?
  - a. breadth first search
  - b. Dijkstra's algorithm
  - c. Kruskal's algorithm
  - d. all of the above
  - e. none of (a), (b), and (c)

# Question 12

- If the shortest path algorithm is run and a vertex is not reachable from the starting point, what happens?

  a. a distance of infinity is reported

  b. a distance of –1 is reported

  c. a distance of zero is reported

  d. the algorithm enters an infinite loop

  e. the algorithm's results are undefined

# Solution 12

- If the shortest path algorithm is run and a vertex is not reachable from the starting point, what happens?
  - a. a distance of infinity is reported
  - b. a distance of –1 is reported
  - c. a distance of zero is reported
  - d. the algorithm enters an infinite loop
  - e. the algorithm's results are undefined

# Question 13

Correct the following code:

```
BinaryNode<AnyType> insert( AnyType x, BinaryNode<AnyType> t )
  {
    if( t == null )
      t = new BinaryNode<AnyType>( x );
    else if( x.compareTo( t.element ) > 0 )
      t.left = insert( x, t.left );
    else if( x.compareTo( t.element ) < 0 )
      t.right = insert( x, t.right );
    else
      throw new DuplicateItemException( x.toString( ) );  // Duplicate
    return t;
  }
```

# Solution 13

Correct the following code:

```
BinaryNode<AnyType> insert( AnyType x, BinaryNode<AnyType> t )
  {
     if( t == null )
        t = new BinaryNode<AnyType>( x );
     else if( x.compareTo( t.element ) < 0 )
        t.left = insert( x, t.left );
     else if( x.compareTo( t.element ) > 0 )
        t.right = insert( x, t.right );
     else
        throw new DuplicateItemException( x.toString( ) );  // Duplicate
     return t;
  }
```

# Question 14

Correct the following code:

```
BinaryNode<AnyType> findMin( BinaryNode<AnyType> t )
    {
        if( t != null )
            while( t.right != null )
                t = t.right;

        return t;
    }
```

# Solution 14

Correct the following code:

```
BinaryNode<AnyType> findMin( BinaryNode<AnyType> t )
  {
    if( t != null )
      while( t.left != null )
        t = t.left;

    return t;
  }
```

# Question 15

Correct the following code:

```
BinaryNode<AnyType> find( AnyType x, BinaryNode<AnyType> t )
  {
    while( t != null )
    {
      if( x.compareTo( t.element ) < 0 )
        t = t.right;
      else if( x.compareTo( t.element ) > 0 )
        t = t.left;
      else
        return t;    // Match
    }

    return null;        // Not found
  }
```

# Solution 15

Correct the following code:

```
BinaryNode<AnyType> find( AnyType x, BinaryNode<AnyType> t )
  {
    while( t != null )
    {
      if( x.compareTo( t.element ) < 0 )
        t = t.left;
      else if( x.compareTo( t.element ) > 0 )
        t = t.right;
      else
        return t;    // Match
    }

    return null;        // Not found
  }
```

# Question 16

```
public void unweighted( String startName ) {
    clearAll( );
    Vertex start = vertexMap.get( startName );
    if( start == null )
      throw new NoSuchElementException( "Start vertex not found" );

    Queue<Vertex> q = new LinkedList<Vertex>( );


    while( !q.isEmpty( ) )
    {
      Vertex v = q.remove( );

      for( Edge e : v.adj )
      {
        Vertex w = e.dest;
        if( w.dist == INFINITY )
        {
          w.dist = v.dist + 1;
          w.prev = v;
          q.add( w );
        }
      }
    }
  }
```

# Question 16

```
public void unweighted( String startName ) {
    clearAll( );
    Vertex start = vertexMap.get( startName );
    if( start == null )
      throw new NoSuchElementException( "Start vertex not found" );

    Queue<Vertex> q = new LinkedList<Vertex>( );
    q.add( start ); start.dist = 0;

    while( !q.isEmpty( ) )
    {
      Vertex v = q.remove( );

      for( Edge e : v.adj )
      {
        Vertex w = e.dest;
        if( w.dist == INFINITY )
        {
          w.dist = v.dist + 1;
          w.prev = v;
          q.add( w );
        }
      }
    }
  }
```

# Question 17

```
public void addEdge( String sourceName, String destName,
    double cost )
    {
        Vertex w = getVertex( sourceName );
        Vertex v = getVertex( destName );
        v.adj.add( new Edge( w, cost ) );
    }
```

# Question 17

```
public void addEdge( String sourceName, String destName,
    double cost )
    {
        Vertex v = getVertex( sourceName );
        Vertex w = getVertex( destName );
        v.adj.add( new Edge( w, cost ) );
    }
```

# Question 18

```
public void printPath( String destName )
  {
     Vertex w = vertexMap.get( destName );
     if( w == null )
        throw new NoSuchElementException( "Destination vertex not found" );
     else if( w.dist == -1 )
        System.out.println( destName + " is unreachable" );
     else
     {
        System.out.print( "(Cost is: " + w.dist + ") " );
        printPath( w );
        System.out.println( );
     }
  }
```

# Solution 18

```
public void printPath( String destName )
  {
     Vertex w = vertexMap.get( destName );
     if( w == null )
        throw new NoSuchElementException( "Destination vertex not found" );
     else if( w.dist == INFINITY )
        System.out.println( destName + " is unreachable" );
     else
     {
        System.out.print( "(Cost is: " + w.dist + ") " );
        printPath( w );
        System.out.println( );
     }
  }
```

# Question 19

- Sketch an algorithm for printing the shortest path after the unweighted shortest-path finding algorithm has been ran.

# Solution 19

**figure 14.13**

A routine for printing the shortest path by consulting the graph table (see Figure 14.5)

```
/**
 * Driver routine to handle unreachables and print total cost.
 * It calls recursive routine to print shortest path to
 * destNode after a shortest path algorithm has run.
 */
public void printPath( String destName )
{
    Vertex w = vertexMap.get( destName );
    if( w == null )
        throw new NoSuchElementException( );
    else if( w.dist == INFINITY )
        System.out.println( destName + " is unreachable" );
    else
    {
        System.out.print( "(Cost is: " + w.dist + ") " );
        printPath( w );
        System.out.println( );
    }
}
```

# Exercise 20

- Sketch a liner-time construction algorithm of a graph.

# Exercise 20

```java
/**
 * A main routine that
 * 1. Reads a file (supplied as a command-line parameter)
 *    containing edges.
 * 2. Forms the graph.
 * 3. Repeatedly prompts for two vertices and
 *    runs the shortest path algorithm.
 * The data file is a sequence of lines of the format
 *    source destination.
 */
public static void main( String [ ] args )
{
    Graph g = new Graph( );
    try
    {
        FileReader fin = new FileReader( args[0] );
        BufferedReader graphFile = new BufferedReader( fin );

        // Read the edges and insert
        String line;
        while( ( line = graphFile.readLine( ) ) != null )
        {
            StringTokenizer st = new StringTokenizer( line );

            try
            {
                if( st.countTokens( ) != 3 )
                {
                    System.err.println( "Skipping bad line " + line );
                    continue;
                }
                String source  = st.nextToken( );
                String dest    = st.nextToken( );
                int    cost    = Integer.parseInt( st.nextToken( ) );
                g.addEdge( source, dest, cost );
            }
            catch( NumberFormatException e )
              { System.err.println( "Skipping bad line " + line ); }
        }
    }
    catch( IOException e )
      { System.err.println( e ); }

    System.out.println( "File read..." );
    System.out.println( g.vertexMap.size( ) + " vertices" );

    BufferedReader in = new BufferedReader(
                        new InputStreamReader( System.in ) );
    while( processRequest( in, g ) )
        ;
}
```

**figure 14.14**

A simple `main`

```java
/**
 * Add a new edge to the graph.
 */
public void addEdge( String sourceName, String destName, double cost )
{
    Vertex v = getVertex( sourceName );
    Vertex w = getVertex( destName );
    v.adj.add( new Edge( w, cost ) );
}
```

**figure 14.10**

Add an edge to the graph

# Question 21

- State the heap order property.

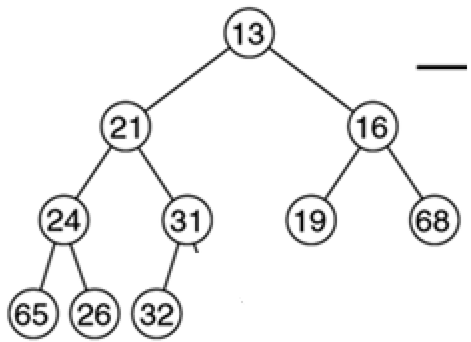
# Solution 21

- Heap-order property

  *In a heap, for every node X with parent P the key in P is smaller than or equal to the key in X.*

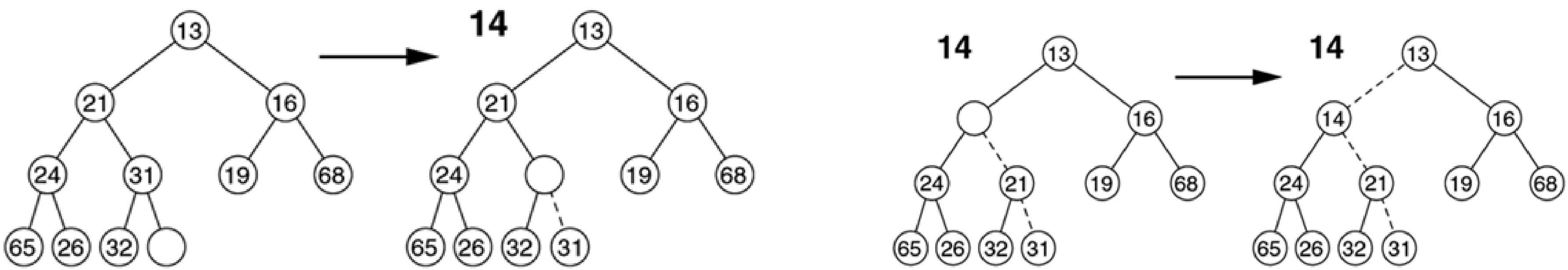
# Exercise 22

- Given the following heap:




- Sketch the operations to insert 14.

# Solution 22: Percolate up



# Exercise 23

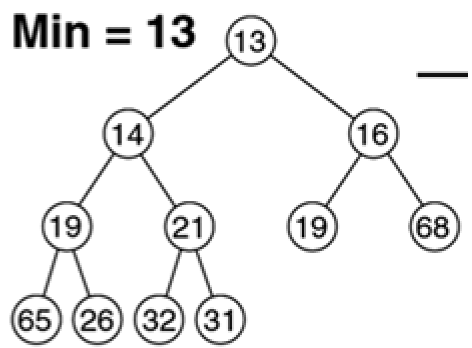
- What is the time complexity of inserting one element in the heap?
- Why?

# Solution 23

- The time required to do the insertion could be as much as O(log N) if the element to be inserted is the new minimum.
- The reason is that it will be percolated up all the way to the root.
- On average the percolation terminates early.

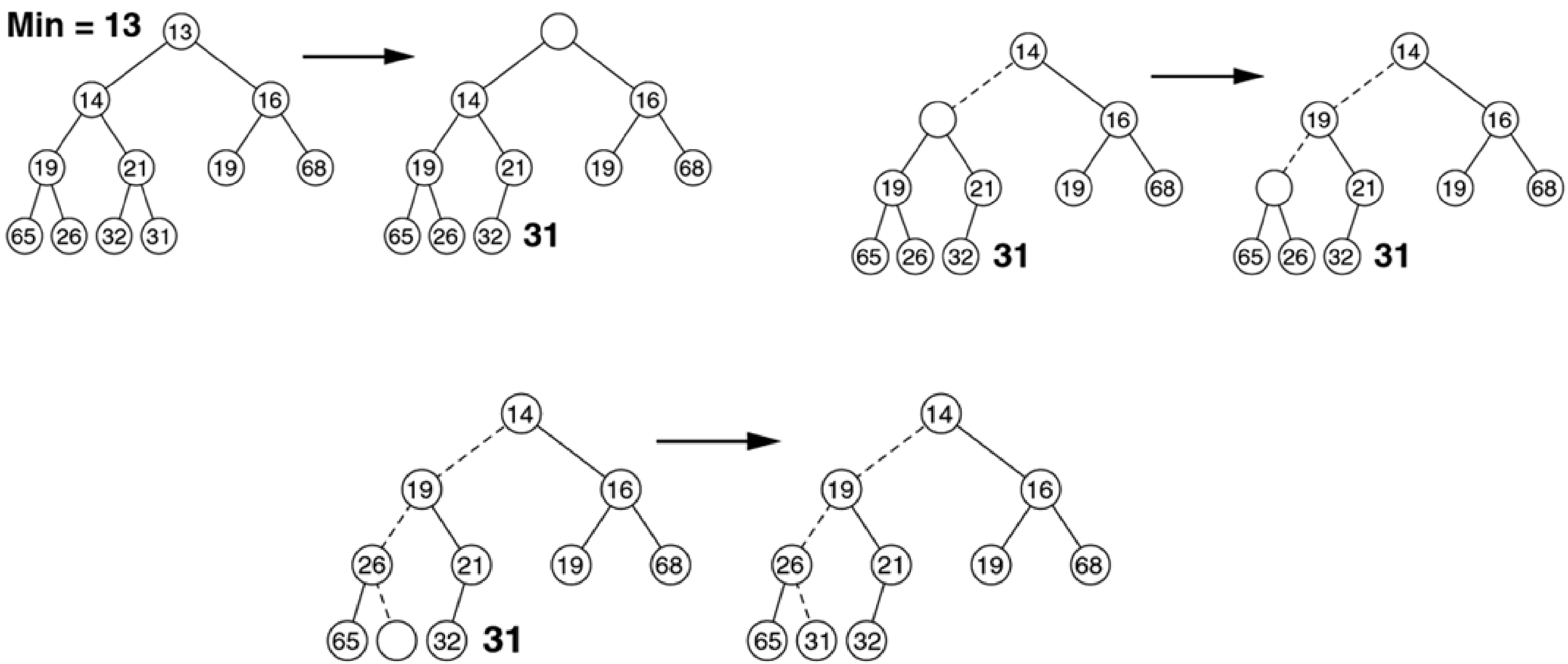
# Exercise 24

- Given the following heap:




- Sketch the operations for deleting the minimum.

# Solution 24: percolate down



# Exercise 25

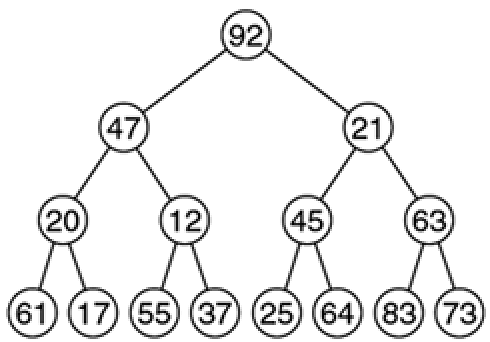
- What is the time complexity of deleting the minimum in a heap?
- Why?

# Solution 25

- Because the tree has logarithmic depth, deleteMin is a logarithmic operation in the worst case.
- Not surprisingly, percolation rarely terminates more than one or two levels early, so deleteMin is logarithmic on average, too.

# Exercise 26

- Given the initial heap:




- Sketch the steps of buildHeap operation.

# Solution 26

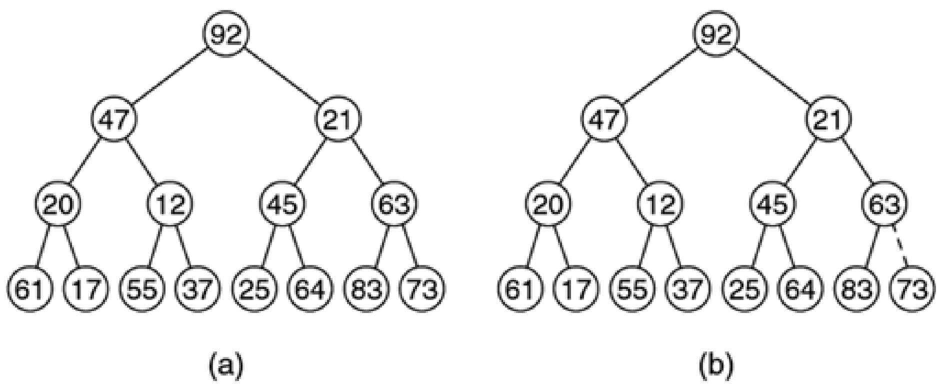
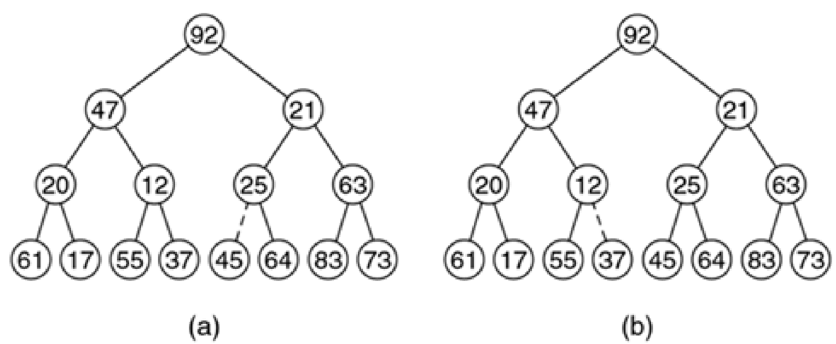



**figure 21.17**

(a) Initial heap;
(b) after
`percolateDown(7)`

**figure 21.18**

(a) After
`percolateDown(6);`
(b) after
`percolateDown(5)`



# Solution 26

**figure 21.19**

(a) After `percolateDown(4);` (b) after `percolateDown(3)`

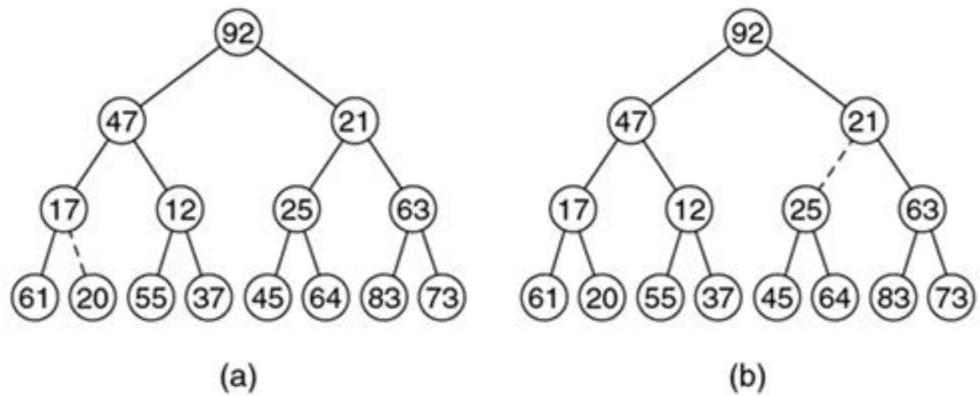
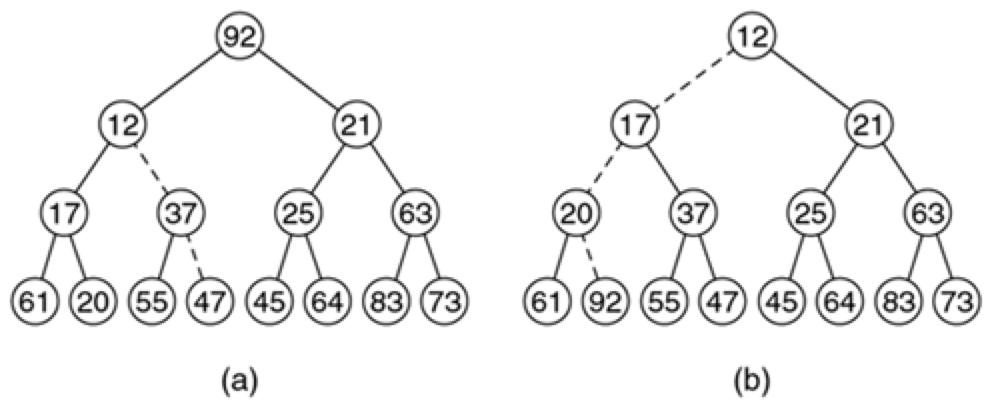



**figure 21.20**

(a) After `percolateDown(2);` (b) after `percolateDown(1)` and `buildHeap` terminates



# Exercise 27

- Sketch an algorithm for the buildHeap operation?
- What is the time complexity of this algorithm?

# Solution 27

```java
/**
 * Establish heap order property from an arbitrary
 * arrangement of items. Runs in linear time.
 */
private void buildHeap( )
{
    for( int i = currentSize / 2; i > 0; i-- )
        percolateDown( i );
}
```

**figure 21.16**

Implementation of the linear-time `buildHeap` method

- Time complexity: linear.

# Exercise 28

- Explain the HeapSort algorithm?

# Solution 28

Algorithm steps

1. Toss each item into the binary heap
2. Apply buildHeap
3. Calling deleteMin N times, with the items exiting the heap in sorted order

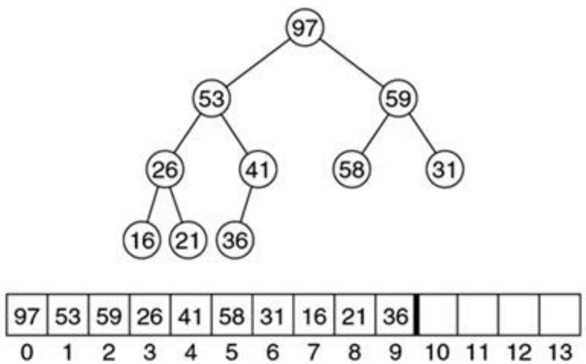
Time Complexity

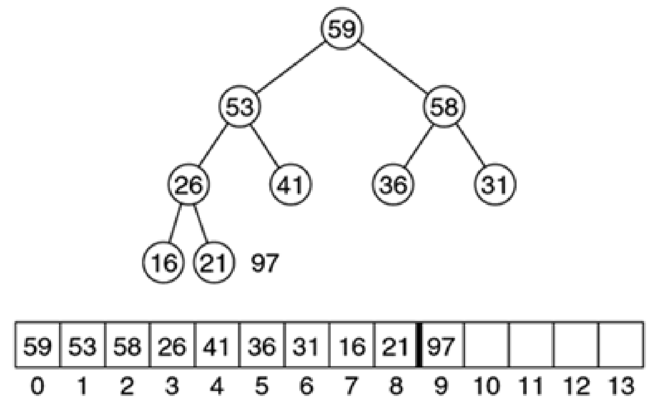
- Step 1 takes linear time total, and step 2 takes linear time.
- In step 3, each call to deleteMin takes logarithmic time, so N calls take O(N log N) time.
- Consequently, we have an O(N log N) worst-case sorting algorithm, called heapsort.

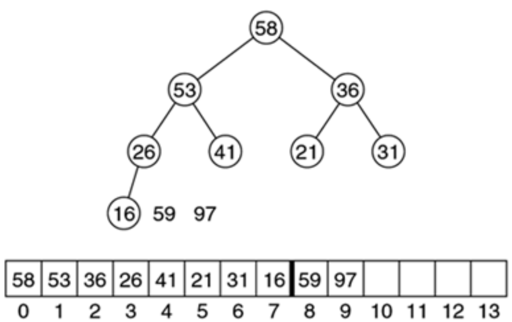
# Exercise 29 and 30

- Modify the HeapSort algorithm to save the double space needed.
- Sketch the algorithm for the following heap.

# Solution 29 and 30




```
// Standard heapsort.
public static <AnyType extends Comparable<? super AnyType>>
void heapsort( AnyType [ ] a )
{
    for( int i = a.length / 2; i >= 0; i-- )  // Build heap
        percDown( a, i, a.length );
    for( int i = a.length - 1; i > 0; i-- )
    {
        swapReferences( a, 0, i );           // deleteMax
        percDown( a, 0, i );
    }
}
```

# Exercise 31

- Which of these algorithms has quadratic time complexity?
  a) BubbleSort
  b) SelectionSort
  c) InsertionSort
  d) HeapSort
  e) MergeSort

# Solution 31

- Which of these algorithms has quadratic time complexity?
  a) BubbleSort
  b) SelectionSort
  c) InsertionSort
  d) HeapSort
  e) MergeSort

# Exercise 32

- Given the following sequence:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 81 | 94 | 11 | 96 | 12 | 35 | 17 | 95 | 28 | 58 | 41 | 75 | 15 |

- apply the increment sequence {1,3,5} for Shellsort.

# Solution 32

**figure 8.5**

Shellsort after each pass if the increment sequence is {1, 3, 5}

| Original | 81 | 94 | 11 | 96 | 12 | 35 | 17 | 95 | 28 | 58 | 41 | 75 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| After 5-sort | 35 | 17 | 11 | 28 | 12 | 41 | 75 | 15 | 96 | 58 | 81 | 94 | 95 |
| After 3-sort | 28 | 12 | 11 | 35 | 15 | 41 | 58 | 17 | 94 | 75 | 81 | 96 | 95 |
| After 1-sort | 11 | 12 | 15 | 17 | 28 | 35 | 41 | 58 | 75 | 81 | 94 | 95 | 96 |

# Exercise 33

- Given the following sequence of numbers:

6 5 3 1 8 7 2 4

- sketch the steps of Mergesort to sort this sequence.

# Solution 33

- Steps
  - Merge-sort.gif

# Exercise 34

- A software house is developing a simple software for a telephone company that wants to run the software on its mobile phones. The software has to solve the following problem (5 points):
  a) The user inserts the contacts in the phone and these are saved with name, surname and number. Choose a data structure to load the contacts from the memory of the phone once the phone is on.
  b) Design a solution so that contacts can be found in the smallest possible time according to their surname. What is the time complexity of your solution?
  c) Suppose the user wants to load all contacts from the SIM card and some of the contacts are repeated (i.e., these are both in the phone and in the SIM card). Design a solution such that the phone has no repeated numbers. What is the time complexity of your solution?

# Solution 34

- Using the list
  a) Order the list by surname, for every name read from the memory, put it in the list and
     - Either keep it sorted with the insertion (if we have N names: time complexity NxN).
     - Or just insert all the contacts and then sort with MergeSort: time complexity NlogN
  b) If sorted, use binary search: time complexity logN.
  c) First check if it exists with binary search, if not insert it: logN.

- Using the hashtable
  a) Insert all contacts, using the hash of: name+surname: time complexity constant.
  b) Find in constant time.
  c) Insert in constant time.

– Solution: No one asked for sorted!!! Therefore hash is preferred.
– If sorted was needed then use the Binary Search Tree
  - Insertion: NlogN
  - Search: logN

# Exercise 35

- A marketing company wants to build software that automatically sends e-mails to customers whose contacts come through banks or other institutions usually in sorted order. The software should work in the following scenario:
  - Initially, the software reads the e-mail addresses from a database provided by an external company, and sends to these addresses the first presentation email.
  - After the first email has been sent successfully, the software listens for replies from the addresses. If a reply comes from a certain address, the software first checks if the address is among those that were sent the first email. If yes, then it sends them the second email with a special offer.
  - Again, if the second email produces a reply, then other emails presenting other products are sent. The software always checks the address is among those got from the external company. All the emails promoting different products can then be customized by the user who wants also to keep track of all the messages sent to an address.

- Design a solution choosing the appropriate data structure that makes the whole process as fast as possible. What is the time complexity of your program?

# Solution 35

We do no need a sorted list:

- Solution using hash tables
  - Every email address is computed the hash (the Key) and saved into the hash table.
  - For every element of the hash table (the Object), we keep a list of messages sent.
- Time complexity:
  - constant time to insert contacts into hash
  - constant time to check first email sent
  - If collisions happen, put messages to different e-mail addresses in the same list.

# Solution 35

- If sorted list is needed as a sorted list of emails, use Binary Search Tree.
  - The key of the node is the email address
  - Insertion: NlogN
  - Search: logN
  - Every element of the tree has a list of emails sent

# Good Luck!