

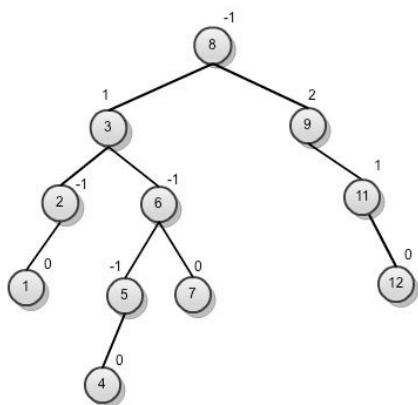
**CSE 373, Spring 2011**  
**Final Key**

**1. Sorting (12 Points)**

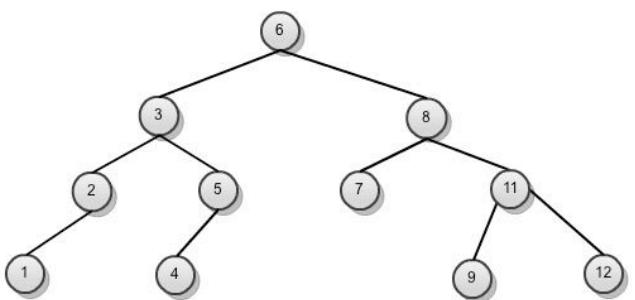
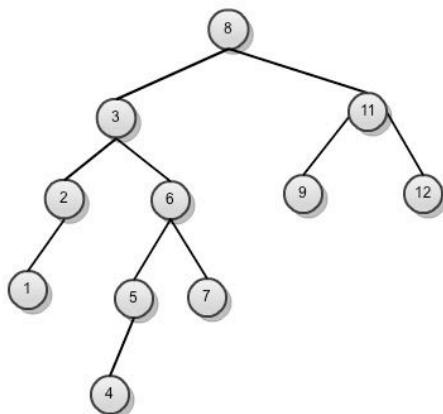
Part	Conditions	Answer	Expected Runtime
a	array size 700000, ascending order	insertion sort	$O(n)$
b	array size 350000, random order, no extra memory may be allocated	quick sort	$O(n \log n)$
c	array size 1000000, descending order	merge sort	$O(n \log n)$
d	array size 2500000 containing zip codes (i.e. values between 0 - 99999), random order	bucket sort	$O(n)$

## 2. AVL Trees (10 Points)

a.



b.



### 3. Heap Implementation (12 Points)

Part	Answer
a	<pre>public void delete(int p) {     if (p &lt;= 0    p &gt; size) {         throw new NoSuchElementException();     }     array[p] = Integer.MIN_VALUE;     bubbleUp(p);     this.remove(); }</pre>
b	<p>Checking that the position is valid and throwing the exception is O(1). Setting the element to be deleted to <code>Integer.MIN_VALUE</code> is O(1). Bubbling up the value to be deleted is worst case O(log n). Performing a regular remove from the top of the heap is O(log n). Therefore, we have <math>O(1 + 1 + \log n + \log n) = O(\log n)</math>.</p>

#### 4. Hashing (12 Points)

Part	Answer
a	<pre>      value +-----+ 0       10     +-----+ 1       22     +-----+ 2       82     +-----+ 3       53     +-----+ 4               +-----+ 5       55     +-----+ 6       92     +-----+ 7       R      +-----+ 8               +-----+ 9       75     +-----+</pre>
b	Yes, 86 failed to be inserted because a bucket couldn't be found after trying half of the entries. The second 55 failed to be inserted because it was already in the set.
c	7
d	10
e	.7

## 5. Topological Sort (10 points)

A B D E C F H G I J

## 6. Minimum Spanning Trees (12 points)

Part	Answer
a	A, B, C, D, E, G, H, I, J, F edges: $AB = 2$ $BC = 3$ $BD = 5$ $DE = 4$ $DG = 6$ $EH = 7$ $GI = 8$ $IJ = 1$ $EF = 13$ Total: 49
b	$IJ = 1$ $AB = 2$ $BC = 3$ $DE = 4$ $BD = 5$ $DG = 6$ $EH = 7$ $GI = 8$ $EF = 13$ Total: 49

## 7. Graph Implementation (12 points)

```
// BFS-based Solution 1: looking at previous/source node to determine node's set
public void friendsAndEnemies(V v1, Set<V> friends, Set<V> enemies) {
    this.clearVertexInfo();
    Queue<V> queue = new LinkedList<V>();
    queue.add(v1);
    friends.add(v1);
    this.vertexInfo.get(v1).visited = true;

    while(!queue.isEmpty()) {
        V v = queue.remove();

        for (V n : this.neighbors(v)) {
            VertexInfo<V> vi = this.vertexInfo.get(n);
            if (!vi.visited) {
                vi.visited = true;
                queue.add(n);

                if (friends.contains(v)) {
                    enemies.add(n);
                } else {
                    friends.add(n);
                }
            }
        }
    }
}

// BFS-based Solution 2: looking at distance to determine node's set
public void friendsAndEnemiesDistance(V v1, Set<V> friends, Set<V> enemies) {
    this.clearVertexInfo();
    vertexInfo.get(v1).distance = 0;

    Queue<V> queue = new LinkedList<V>();
    queue.offer(v1);

    friends.add(v1);

    while(!queue.isEmpty()) {
        V v = queue.poll();

        for (V n : this.neighbors(v)) {
            VertexInfo<V> vi = this.vertexInfo.get(n);
            if (vi.distance == Integer.MAX_VALUE) {
                vi.distance = vertexInfo.get(v).distance + 1;

                queue.offer(n);

                if (vi.distance % 2 == 0) {
                    friends.add(n);
                } else {
                    enemies.add(n);
                }
            }
        }
    }
}
```

```

// DFS-based Solution 1: looking at distance to determine node's set
public void friendsAndEnemies(V v1, Set<V> friends, Set<V> enemies) {
    this.clearVertexInfo();
    friends.add(v1);
    vertexInfo.get(v1).distance = 0;

    for (V neighbor : neighbors(v1)) {
        friendsAndEnemies(neighbor, friends, enemies, 0, false);
    }
}

public void friendsAndEnemies(V v1, Set<V> friends, Set<V> enemies, int distance,
boolean isFriend) {
    if (distance < vertexInfo.get(v1).distance) {
        vertexInfo.get(v1).distance = distance;
        if (isFriend) {
            friends.add(v1);
            enemies.remove(v1);
        } else {
            enemies.add(v1);
            friends.remove(v1);
        }
    }

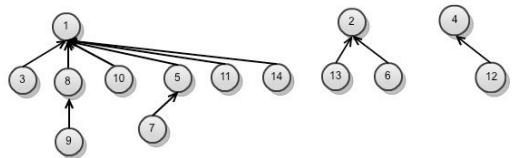
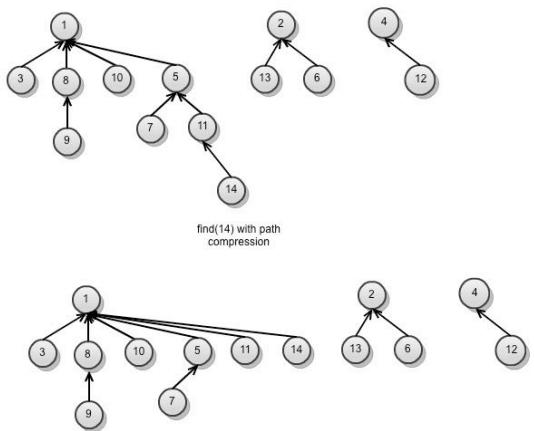
    for (V neighbor : neighbors(v1)) {
        friendsAndEnemies(neighbor, friends, enemies, distance + 1, !isFriend);
    }
}
}

```

## 8. Disjoint Sets (10 points)

Part	Answer
a	<p>The diagram illustrates the construction of a disjoint set union structure for 14 elements. It starts with 14 individual nodes labeled 1 through 14. Union operations are performed sequentially:</p> <ul style="list-style-type: none"> <li>union(1, 3)</li> <li>union(5, 7)</li> <li>union(8, 9)</li> <li>union(1, 8)</li> <li>union(2, 13)</li> <li>union(1, 10)</li> <li>union(11, 14)</li> <li>union(5, 11)</li> <li>union(2, 6)</li> <li>union(4, 12)</li> <li>union(x, y) = union(1, 5)</li> </ul> <p>After these operations, the structure consists of several components:</p> <ul style="list-style-type: none"> <li>Component 1: Rooted at node 1, containing nodes 1, 3, 5, 8, 9, 10, 11, 12, 13, 14.</li> <li>Component 2: Rooted at node 2, containing nodes 2, 13.</li> <li>Component 3: Rooted at node 4, containing nodes 4, 5, 6, 7, 11, 12.</li> <li>Component 4: Rooted at node 6, containing nodes 6, 10.</li> <li>Component 5: Rooted at node 7, containing nodes 7, 11.</li> <li>Component 6: Rooted at node 8, containing nodes 8, 10.</li> <li>Component 7: Rooted at node 9, containing nodes 9, 11.</li> <li>Component 8: Rooted at node 10, containing nodes 10.</li> <li>Component 9: Rooted at node 11, containing nodes 11.</li> <li>Component 10: Rooted at node 12, containing nodes 12.</li> <li>Component 11: Rooted at node 13, containing nodes 13.</li> <li>Component 12: Rooted at node 14, containing nodes 14.</li> </ul>

b



## 9. BTrees (10 points)

Part	Answer
a	<pre>graph TD; Root[17] --- Node1[6, 11]; Root --- Node2[25, 50]; Node1 --- Leaf1[1, 3, 5]; Node1 --- Leaf2[6, 7, 9]; Node1 --- Leaf3[11, 13, 15]; Node2 --- Leaf4[17, 19, 21, 23]; Node2 --- Leaf5[25, 31, 41]; Node2 --- Leaf6[50, 52, 53, 54];</pre>
b	<p>The point of B-trees is to store data on disk. For them to work efficiently, nodes should all fit within one disk block, which is a fixed size. Internal nodes hold keys and links. External nodes hold keys and data. Therefore, a different number of keys/links than keys/data may fit into a single disk block. This means that different values of M and L may be required for internal and external nodes.</p>