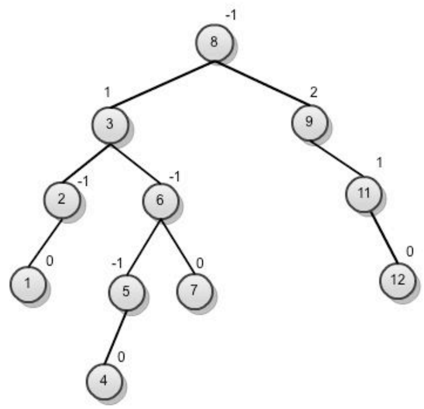
CSE 373, Spring 2011
Final Key

1. **Sorting** (12 Points)

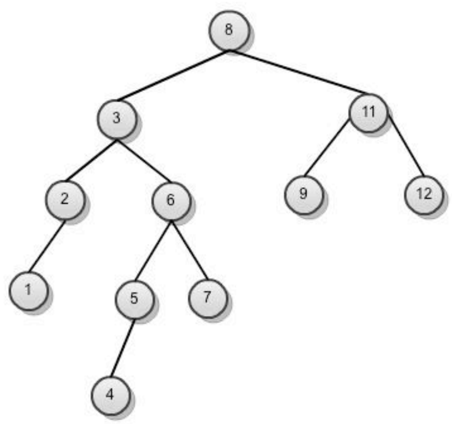
Part	Conditions	Answer	Expected Runtime
a	array size 700000, ascending order	insertion sort	O(n)
b	array size 350000, random order, no extra memory may be allocated	quick sort	O(n log n)
c	array size 1000000, descending order	merge sort	O(n log n)
d	array size 2500000 containing zip codes (i.e. values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

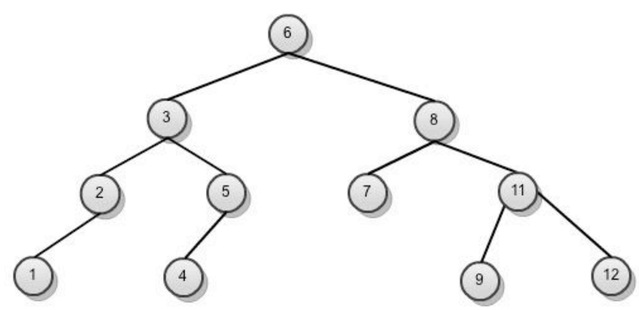
2. AVL Trees (10 Points)

a.



b.





3. Heap Implementation (12 Points)

<table>
<tr><th>Part</th><th>Answer</th></tr>
<tr><td>a</td><td><pre>public void delete(int p) {
    if (p <= 0 || p > size) {
        throw new NoSuchElementException();
    }
    array[p] = Integer.MIN_VALUE;
    bubbleUp(p);
    this.remove();
}</pre></td></tr>
<tr><td>b</td><td>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 O(1 + 1 + log n + log n) = O(log n).</td></tr>
</table>

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)

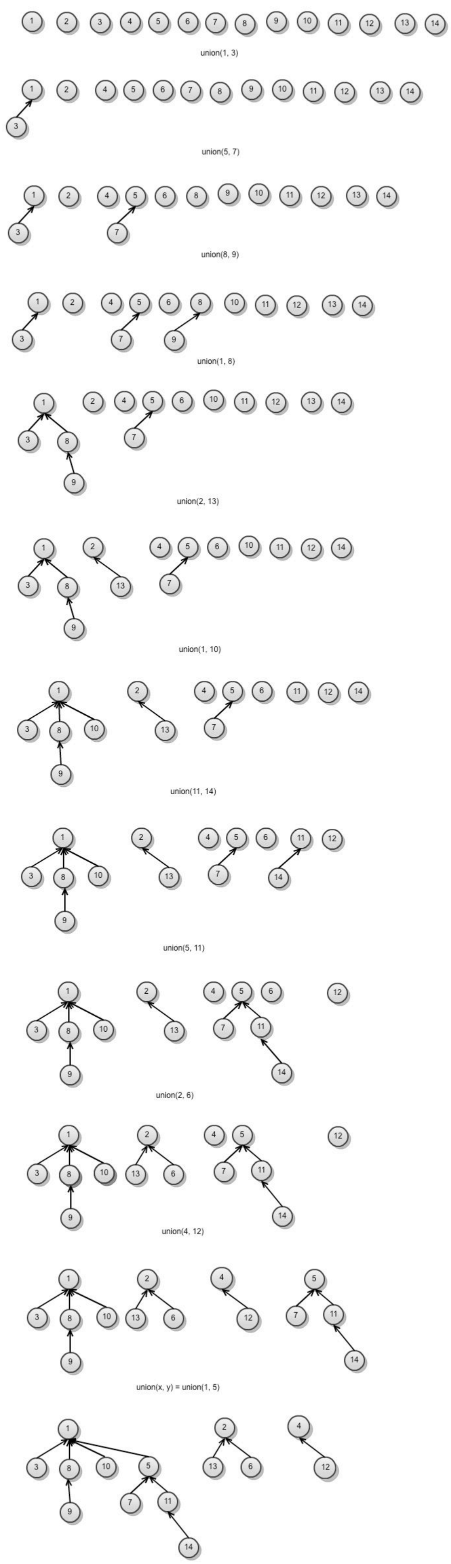
Part	Answer
a	A, B, C, D, E, G, H, I, J, F edges: AB = 2 BC = 3 BD = 5 DE = 4 DG = 6 EH = 7 GI = 8 IJ = 1 EF = 13 Total: 49
b	IJ = 1 AB = 2 BC = 3 DE = 4 BD = 5 DG = 6 EH = 7 GI = 8 EF = 13 Total: 49

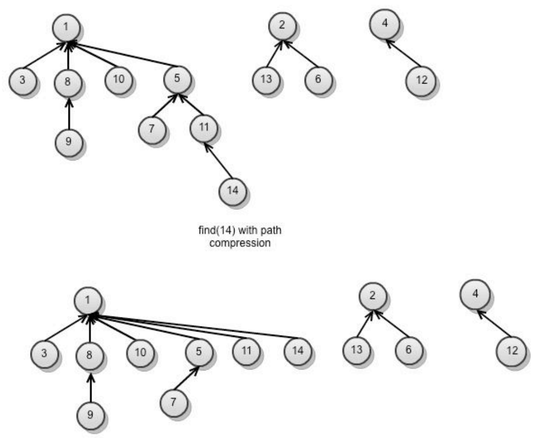
7. Graph Implementation (12 points)

```java
// 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);
                     }
               }
         }
   }
}
```

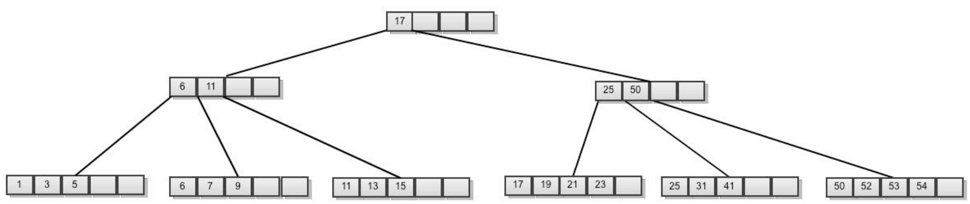
```
// DFS-based Solution 1: looking at distance to determine node's set
public void friendsAndEnemies(V v1, Set<V> friends, Set<V> enemies) {
   this.clearVertexInfo();
   friends.add(v1);
   vertexInfo.get(v1).distance = 0;

   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)

<table>
<tr><th>Part</th><th>Answer</th></tr>
<tr><td>a</td><td>
</td></tr>
</table>

b	

9. BTrees (10 points)

Part	Answer
a	
b	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.