Problem Set 6, Part I

Problem 1: Quicksort
1-1) {7,10,13,27,24,20,14,33}

1-2) {7,10,13,14,20,24,27,33}

Problem 2: Mergesort
2-1) {3, 13, 24, 27, 34, 2, 50, 12}

2-2) {3, 13, 24, 27, 2, 12, 34, 50}

2-3) {3, 13, 24, 27, 34, 2, 50, 12}

Problem 3: Counting unique values

3-1) The worst case for this algorithm is that if each element has to check for every value “i”, it checks the elements of the array in front of it every single time. In the big O notation, it would be O(n^2). This would occur if every value is unique.

3-2) n * (n-1)/2

3-3) In the big O notation, it would be O(n^2) since the exact formula has n^2 in it and getting rid of all the constants would give you n^2. This would occur if every value is unique.

3-4) The best case would be if every number in the array were the unique. If this where to happen, the inner for loop would not run its contents.

3-5) This would give us O(n) since it has to go through every element in the array once.

Problem 4: Improving the efficiency of an algorithm

4-1)

```
public static int numUnique(int[] arr) {
      Sort.mergeSort(arr);

      int count = 1;
      for (int i = 1; i < arr.length; i++) {
            if (arr[i] != arr[i-1]) {
                  count++;
            }
      }
      return count;
}
```

4-2) The worst case time efficiency for my method is O(nlogn). Since the mergeSort algorithm is O(nlogn), that would be the overall time efficiency of the method. The exact equation would be nlogn + n.

4-3) It would not have a better best-case efficiency because the best case would also be O(nlogn) and the best case for the other one is O(n) which is faster.

Problem 5: Practice with references

5-1)

Expression	**Address**	**Value**
n	0x128	0x800
n.ch	0x800	‘e’
n.next	0x802	0x240
n.prev.next	0x182	0x800
n.next.prev	0x246	0x800
n.next.prev.prev	0x806	0x180

5-2)

```
public static void main(String[] args) {
      // makes m’s next node ‘t’
      m.next = n.next;

      // sets the ‘t’ nodes prev to be ‘a’ node
      n.next.prev = m;

      // sets n.next to ‘a’ node
      n.next = m;

      // sets m.prev as to ‘e’ node or n
      m.prev = n;
}
```

5-3)

```
public static void addNexts(DNode last) {
      last.next = null;
      Dnode trav = last;
      trav = trav.prev;
      trav.next = last;


      while (trav.prev != null) {
            trav.prev.next = trav;
            trav = trav.prev;
      }
```

```
}
```

Problem 6: Printing the odd values in a list of integers

6-1)

```
public static void printOddsRecur(IntNode first) {
      if (first == null) {
            return;
      }

      if (first.val % 2 == 1) {
            System.out.println(first.val);
      }

      printOddsRecur(first.next);
}
```

6-2)

```
public static void printOddsIter(IntNode first) {
      while (first != null) {
            if (first.val % 2 == 1) {
                  System.out.println(first.val);
            }
            first = first.next;
      }
}
```