Problem Set 6, Part I

Problem 1: Counting unique values

1-1)
The worst case occurs when every element in the array is unique.

1-2)
The line comparing arr[j] to arr[i] will execute for (n-1)+(n-2)+...+2+1 = n(n-1)/2 times.

1-3)
The overall time efficiency is O(n^2) times. In the worst case, every element needs to compare with the rest of the elements in the array. The outer loop will execute for n times. The inner loop will execute for n-1 times. The comparison of arr[j] and arr[i] will execute for (n-1)+(n-2)+...+2+1 = n(n-1)/2 = O(n^2) times.

1-4)
The best case occurs when all the elements in the array are the same.

1-5)
The overall time efficiency is O(n) times. The outer loop will execute for n times. For each outer loop, the inner loop will execute for 1 time except in the case i = arr.length-1. Therefore, the overall time efficiency is n-1 = O(n) times.

Problem 2: Improving the efficiency of an algorithm

2-1)

```
public static int numUnique(int[] arr) {
    Sort.mergeSort(arr);
    int count = 1;
    if (arr[0] == arr[arr.length-1]) {
        return count;
    }
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] != arr[i-1]) {
        count++;
        }
    }
    return count;
}
```

2-2)

The worst-case time efficiency is O(nlogn), which is more efficient than O(n^2). The comparison and move in mergesort are both O(nlogn), so the running time is also O(nlogn). In the worst case, every element in the array is unique, and the if statement in the for loop will execute for n-1 times. Therefore, the overall time efficiency is O(nlogn)+O(n) = O(nlogn).

2-3)

The best-case time efficiency is also O(nlogn), which is not more efficient than O(n). The running time of mergesort is still O(nlogn) in the best case. All the elements in the array are the same, so the if statement will execute for 1 times and the for loop will not execute. Therefore, the overall efficiency is O(nlogn).

Problem 3: Practice with references

3-1)

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

3-2)

```
m.next = n.next;
n.next.prev = m;
n.next = m;
m.prev = n;
```

3-3)

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

Problem 4: Printing the odd values in a list of integers

4-1)

```
public static void printOddsRecur(IntNode first) {
    if (first == null) {
        return;
    } else {
        printOddsRecur(first.next);
        if (first.val % 2 != 0 ) {
            System.out.println(first.val);
        }
    }
}
```

4-2)

```
public static void printOddsIter(IntNode first) {
    IntNode trav = first;
    while (trav != null) {
        if (trav.val % 2 != 0) {
            System.out.println(trav.val);
        }
        trav = trav.next;
    }
}
```