Problem Set 4, Part I

Problem 1: Sorting practice

1-1)
{7, 10, 13, 27, 24, 20, 14, 33}
1-2)
{7, 13, 14, 24, 27, 20, 10, 33}
1-3)
{7, 13, 14, 20, 10, 24, 27, 33}
1-4)
{10, 7, 13, 27, 24, 20, 14, 33}
1-5)
{10, 7, 13, 27, 24, 20, 14, 33}
1-6)
{7, 13, 14, 27, 24, 20, 10, 33}

Problem 2: Practice with big-O

2-1)

function	**big-O expression**
a(n) = 5n + 1	a(n) = O(n)
b(n) = 2n^3 + 3n^2 + nlog(n)	b(n) = O(n^3)
c(n) = 10 + 5nlog(n) + 10n	c(n) = O(nlog(n))
d(n) = 4log(n) + 7	d(n) = O(log(n))
e(n) = 8 + n + 3n^2	e(n) = O(n^2)

2-2)
Since the outer loop runs 2n times and inter loop runs n-2 times,
2n * (n-2) = 2n^2 -4n.
Therefore O(n^2).

2-3)
Multiplying the First loop 5times, second loop (n^2-2)/2 times, and third loop log(n) times gives 5 * (n^2-2)/2 * log(n).
Therefore O(n^2log(n)).

Problem 3: Comparing two algorithms

worst-case time efficiency of algorithm A: O(nlog(n))
Explanation: After the separation in half, when the left and right numbers store alternate elements, it will cause worst time efficiency.

worst-case time efficiency of algorithm B: O(n)
Explanation: Since the loop is repeats relevant to the length of the array, the worst efficiency will be caused when the length of the array is the longest.

Problem 4: Counting unique values

4-1)The worst case of this algorithm is when there are no same numbers.

4-2)n * n(i + 1) = n^2 + in + n

4-3)O(n^2). Because there are no same numbers, the inner loop have to run all the way through.

4-4) The best case of this algorithm is when all the numbers are same.

4-5)O(n) If all the numbers are same, the inner loop have to run 1 times each.

Problem 5: Improving the efficiency of an algorithm

5-1)

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

5-2) Since the worst to best case for mereSort is nlogn, the big-O notation for the new method is O(nlogn).

5-3)No. Since the best case for the original method was O(n), it is more efficient then the new case.

Problem 6: Practice with references

6-1)

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

6-2)

```
public static void main(String[] args) {
n.next.prev = m;
m = n.next;
m.prev = n;
n.next = m;
}
```

6-3)

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

problem 7: Printing the odd values in a list of integers

7-1)

```
public static void printOddsRecur(IntNode first) {
      if (first == null) {
            return;
      } else {
            if (first.val % 2 == 1) {
                  System.out.println(first.val)
            } else {
                  System.out.println(first.next)
            }
      }
}
```

7-2)

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