Problem Set 5, Part I
Problem 1: Sorting practice
1-1)
```
{3, 4, 18, 24, 33, 40, 8, 10, 12}
```

1-2)
```
{4, 10, 18, 24, 33, 40, 8, 3, 12}
```

1-3)
```
{4, 10, 18, 8, 3, 12, 24, 33, 40}
```

1-4)
```
{10, 18, 4, 24, 12, 3, 8, 40, 33}
```

1-5)
```
{10, 8, 4, 3, 12, 24, 18, 40, 33}
```

1-6)
```
{4, 10, 18, 24, 33, 40, 8, 3, 12}
```

Problem 2: Practice with big-O
2-1)

function	**big-O expression**
a(n) = 5n + 1	a(n) = O(n)
b(n) = 5 - 10n - n^2	b(n) = O(n^2)
c(n) = 4n + 2log(n)	c(n) = O(n)
d(n) = 6nlog(n) + n^2	d(n) = O(n^2)
e(n) = 2n^2 + 3n^3 - 7n	e(n) = O(n^3)

2-2)
O(n^2)
The “i” loop iterates 3 times. The “j” loop iterates n times. The “k” loop iterates (n+1)/2 times. All in all, they iterates 3n(n+1)/2 times, which is O(n^2).

2-3)
O(nlog(n))
The outer loop (noted by i) iterates n times, while the iterating times of the inner loop is the power of 2, which means it iterates log(n) times. Combining them, we get n*log(n) times.

Problem 3: Comparing two algorithms
worst-case time efficiency of algorithm A: O(n^2)
explanation: when every element in the arr[] are the same, numDups++ needs to run (n+1)*n/2 times, which is O(n^2)

worst-case time efficiency of algorithm B: O(nlog(n))
explanation: the time efficiency of merge sort is nlong(n), which is more complicated than the following loop whose time efficiency is n.

Problem 4: Practice with references

4-1)

Expression	Address	Value
n	0x100	0x712
n.ch	0x712	‘n’
n.prev	0x718	0x064
n.prev.prev	0x070	0x360
n.prev.next.next	0x714	Null
n.prev.prev.next	0x362	0x064

4-2)

x.next= n.prev.prev.next.next;
x.prev= n.prev.prev.next;
n.prev.next=x;
n.prev=x;

4-3)

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