Problem Set 5, Part I

Problem 1: Sorting practice

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

1-2) {3,13,24,27,34,2,50,12]

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

1-4) {12,3,2,13,34,27,50,24}

1-5) {2,3,12,13,34,27,50,24}

1-6) {3,13,24,27,2,34,50,12}

Problem 2: Practice with big-O

2-1)

function	**big-O expression**
a(n) = 5n + 1	a(n) = O(n)
b(n) = 2n + 3n^2 + nlog(n)	b(n)= O(n^2)
c(n) = 5nlog(n) + 10n^3 + n^2	c(n)= O(n^3)
d(n) = 3log(n) + 7n	d(n)=O(n)
e(n) = 8 + 9n + 4nlog(n)	e(n)=O(nlogn)

2-2) O(n): the inner loop is executed O(1) as it is not a function of n. However, the outer loop is executed 3n-2 times which is equivalent to O(n) times. Therefore the count() method would be called O(n) times.

2-3) O(n^3): The outer loop executes at O(n) as it goes from 0 to n. The inner loop alone also executes at O(n) as it goes from 0 to 2n and Big O excludes constants. The second inner loop alone also executes at O(n). Therefore count() will execute O(n^3) times.

Problem 3: Comparing two algorithms

worst-case time efficiency of algorithm A: O(n logn)

Explanation: In the worst case scenario merge sort performs at O(nlogn) efficiency. This is because merge sort divides the array in half over and over therefore giving 2n moves per level and there are approximately log n levels. In big O, this gives O(nlogn). Additionally, worst case scenario the loop in algorithm A traverses through all the way through for a total of arr.length times which adds another O(n). This therefore gives an overall worst-case time efficiency of O(nlogn)

worst-case time efficiency of algorithm B: O(n^2)

Explanation: The worst case scenario in Algorithm B is that there are no duplicates in the array. This would mean both loops would traverse all the way through. The outer loop executes n times and the inner loop would also execute n times with n being arr.length. This therefore gives a total time complexity of n^2 and the big O notation for this is O(n^2);

Problem 4: Practice with references

4-1)

Expression	**Address**	**Value**
x	0x128	0x840
x.ch	0x840	'h'
y.prev	0x326	0x400
y.next.prev	0x666	0x320
y.prev.next	0x402	0x320
y.prev.next.next	0x322	null

4-2)

```
y.prev.next=x;
x.prev=y.prev;
y.prev=x;
x.next=y;
```

4-3)

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