1. My programming partner is Nathan Taylor and he submitted the source code of the assignment.

2. We ended up switching about every 45 minutes although we didn't choose that time specifically, it just worked out that way. I've felt good about the way in which we're programming and don't have any complaints.

3. I'll definitely work with Nate again, be's very bright and we have a good time working together. I may have a bit more experience and I'm very dedicated to creating good, working code, but we work well together and he's been really flexible with my schedule.

4. If I'd used a Java List instead of a basic array, I could have just used the methods already defined for that List for most of the methods that we needed to implement. The only thing I would have to do is write a binary search that works with add, contains, and remove. Using an ArrayList would have been similar in run-time efficiency because the Java ArrayList is backed by a basic array, but a LinkedList would make binary search really slow because it would have to iterate through every element before the middle element to compare it with the element we're looking for. It would definitely make program development time shorter to use an ArrayList though.

5. I expect the Big-O behavior of our contains method to be log(N) because it implements a binary search which halves the input each recursion. Because the input is halved each recursion, the method should recurse about log(N) times where the base of log(N) is 2.

6. This is our plot for the time is takes in nanoseconds to check if an element is contained in an input of size N.

Size of N (thousands)	100	200	300	400	500	600	700	800	900	1,000
Time (nanoseconds)	54	63	67	65	69	74	73	76	77	79
Size of N (thousands)	1,100	1,200	1,300	1,400	1,500	1,600	1,700	1,800	1,900	2,000
Time (nanoseconds)	77	80	79	80	83	82	81	84	83	82

Yes, the increases are greater between smaller input sizes than they are between the greater input sizes which means that the running time is increasing slower than linearly, logarithmic time would match up well against our above plot.

7. This is our plot for the time in nano seconds it takes the add an element to an input of size N.

Size of N	10,000	25,000	50,000	70,000	100,000
Time (nanoseconds)	5,739	24,867	53,845	74,658	101,454

See page 2 below

Locating the correct position at which to insert the element should take the same amount of time as normal binary search which is O(log(N)), but actually adding the element into the array will require that we copy everything to the right of the element over a place. I'm not certain if adding an element would be O(N + log(N)) which reduces to O(N) or if it would be O(N*log(N)). If I take the example of N = 1024, then binary search would take about 10 steps or log(N) to find the correct location of the element in its worst case. If the element was added to the first position in the array, which is worst case for adding, then there would be 1024 copies necessary to move all the elements in the array to the right by one, which would work out to be 1034 operations, which is O(N + log(N)), so I believe the worst case complexity of locating where to add an element and adding it should be O(N + log(N)).

8. We probably spent about 23 or 24 hours on this assignment.