**Report**

The main functions in Brute.java and Fast.java each contain a call to Arrays.sort(). This will not be taken into consideration in the following report as it's purpose is merely for aesthetics and testing. Additionally, the sort is not utilized when obtaining results for this report.

**Estimations:**

**Brute**

Proposition A
The brute-force method has an approximate time complexity of N^4.

Proof
The Brute.java class contains a function run() which takes an array of N Point objects and finds any unique lines of length connecting 4 Points. The algorithm is composed of four nested for loops, optimized to avoid repetitions. The first loop performs (N – 3) iterations. The remaining loops perform linearly decreasing number of loops with a maximum of (N – 3) iterations, and a minimum of 1 iteration. Because the number of iterations for the remaining three loops is linear, we can take the average number of iterations and consider them constant.

From the above reasoning we obtain the following formula for the total number of references to the if statement wrapped in the for loops:

$$(N - 3) * 1/2*(N - 3) * 1/2*(N - 3) * 1/2*(N - 3) = 1/8*(N - 3)^4$$

which is approximately $N^4$.

**Fast**

Proposition B
The fast-sorting method has an approximate time complexity of $N^2 \log N$.

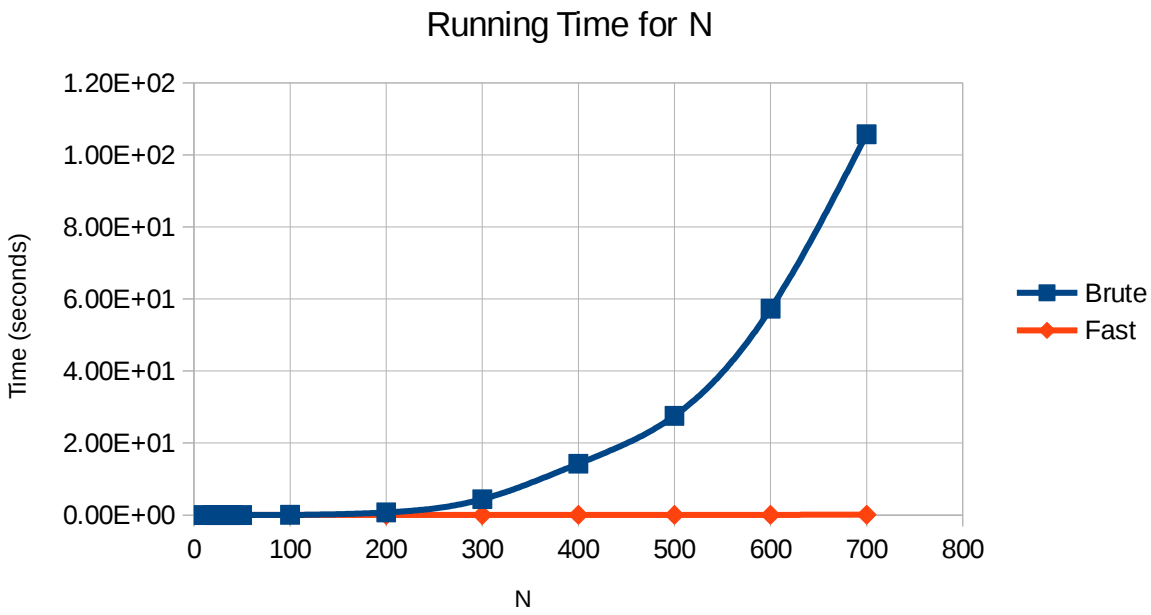
Proof
The Fast.java class contains a function run() which takes an array of N Point objects and finds any unique lines of length greater than 3 Points. The algorithm is composed as follows:

- A for loop iterates over the first (N – 3) Points.
  ~ (N)
- Inside this for loop, the selected Point object (p) is defined as fixed.
- The remaining Points are sorted by their slope relative to p.
  ~ ½(NlogN)
- The sorted remaining Points are iterated over once, with any groups of more than 2 Points with the equal slopes being saved as a line.
  ~ ½(N)
- Point p is then removed from the Points that will be sorted in the next iteration. (This causes the above two to be halved)

This results in approximately $N(\frac{1}{2}(N\log N) + \frac{1}{2}N) \sim N^2(\log N) + N^2 \sim N^2 \log N$

**Empirical Evidence:**

| N | Brute | Fast |
|---|---|---|
| 10 | 1.37859 E-4 | 5.7699 E-4 |
| 20 | 6.37345 E-4 | 2.38679 E-4 |
| 30 | 0.002716790 | 4.6805 E-4 |
| 40 | 0.003440671 | 5.53795 E-4 |
| 50 | 0.005292140 | 8.98189 E-4 |
| 100 | 0.048888147 | 0.002485466 |
| 200 | 0.726735100 | 0.008691335 |
| 300 | 4.411956000 | 0.016937291 |
| 400 | 14.24622800 | 0.048029173 |
| 500 | 27.54054000 | 0.048852670 |
| 600 | 57.27717000 | 0.071190080 |
| 700 | 105.6785600 | 0.094614826 |




**Estimation:**

From an estimation from the observed values I postulate the following running times for N = 1,000,000:

Brute: 4.5 E+14 s Fast: 3.8 E+6 s

Appendix of Code:

```
public class Brute {
	public ArrayList<Point[]> run(Point[] points) {
		ArrayList<Point[]> lines = new ArrayList<>();

		for (int a = 0; a < points.length - 3; a++) {
			for (int b = a + 1; b < points.length - 2; b++) {
				for (int c = b + 1; c < points.length - 1; c++) {
					for (int d = c + 1; d < points.length; d++) {
						if (Point.areCollinear(points[a], points[b],
						    points[c], points[d]))
						{
							lines.add(new Point[]{points[a],
							    points[b], points[c], points[d]});
						}
					}
				}
		}
		}
	return lines;
	}
}

public class Fast {
	public ArrayList<Point[]> run(Point[] points) {
		ArrayList<Point[]> lines = new ArrayList<>();

		for (int i = 0; i < points.length - 3; i++) {
			Point[] sortable = new Point[points.length - (i + 1)];
			for (int j = 0; j < sortable.length; j++) sortable[j] = points[j+1+i];

			Arrays.sort(sortable, points[i].BY_SLOPE_ORDER);

			int count;
			for (int j = 0; j < sortable.length; j += count) {

				double s = points[i].slope(sortable[j]);
				count = 1;

				while (j+count < sortable.length && s ==
					points[i].slope(sortable[j + count])) count++;

				if (count > 2) {
					Point[] line = new Point[count + 1];

					line[0] = points[i];

					for (int c = 0; c < count; c++)
						line[c + 1] = sortable[j + c];
					lines.add(line);
				}
			}
		}
	}
}
```