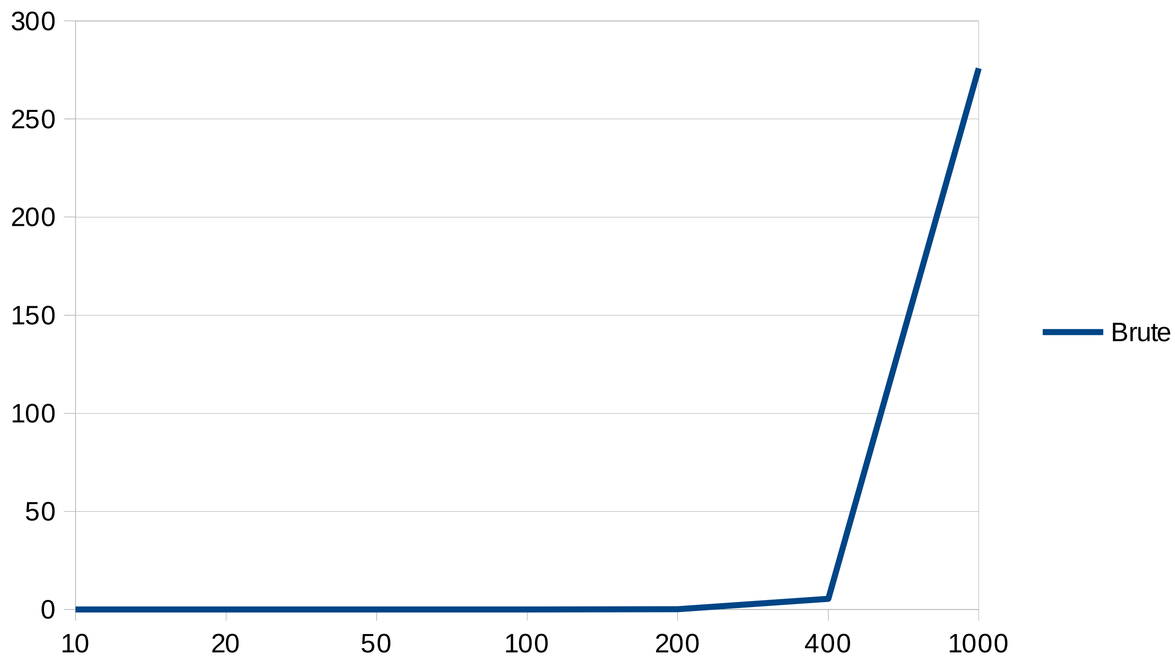
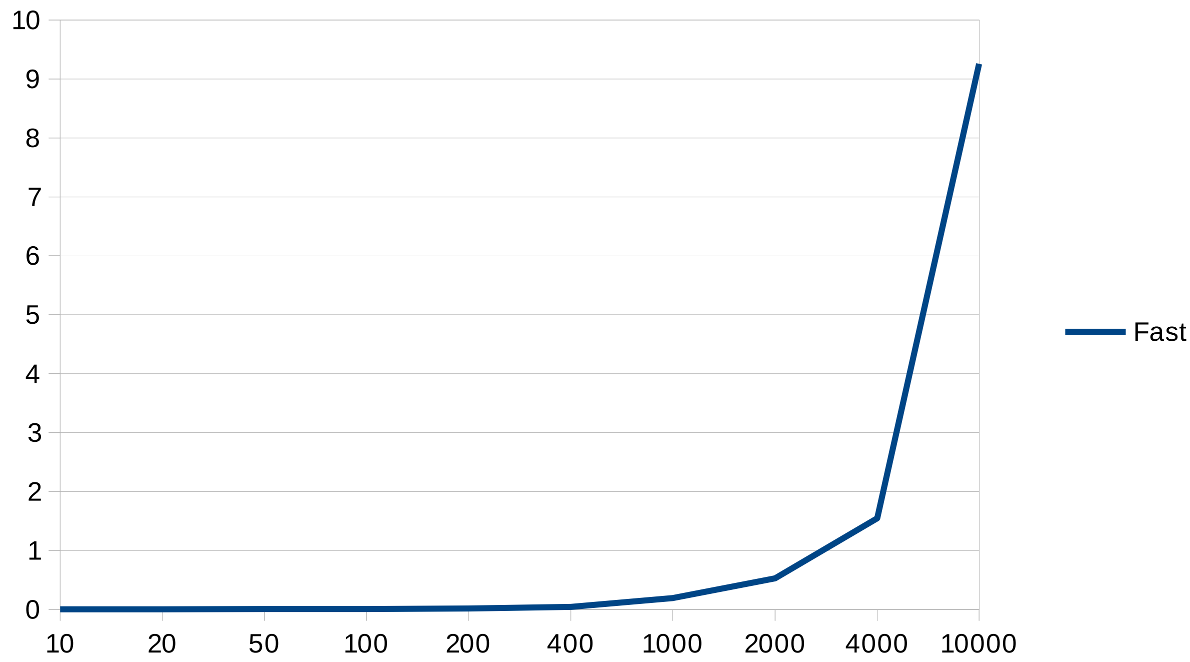
Benjamin Loisch
PSO 2
CS 251
Project 2

Given N random points with x = [0 to 32767] and y = [0 to 32767]:

N (x axis)	Brute execution time (seconds)	Fast execution time (seconds)
10	0.006	0.004
20	0.006	0.005
50	0.013	0.006
100	0.026	0.01
200	0.059	0.018
400	5.416	0.046
1000	275.833	0.196
2000	Very large	0.53
4000	Very large	1.547
10000	Very large	9.255





Analysis of Brute:

In Brute.java I sort the array once before getting started. The array is n elements large. That is n * ln(n) complexity to sort according to java documentation as I use their Arrays.sort() method.

In Brute.java we find all combinations of 4 points in a set of n points.
The generic equation for choosing a set of k combination of points out of n where order doesn't matter is (n choose k) = n(n-1)(n-2)...(n-k+1) / k!

In our case it is (n choose 4) = n(n-1)(n-2)(n-3) / 4! = ((n^4)/12) – ((n^3)/2) + ((11n^2)/12) – (n/2)
The order of growth of (n choose 4) is n^4.

Out total expression for the time complexity of Brute is then our initial sort operation added to our algorithm that finds all combinations of 4 points.

f(n) = (n * ln(n)) + ((n^4)/12) – ((n^3)/2) + ((11n^2)/12) – (n/2)

Tilde notation of this function gives us ~f(n) = (n^4)/12

Let T(n) be the running time of our program.
Using our order of growth, T(n) = a*(n^4)

By studying a log-log plot of the data, we get a line with a slope of about 4
We know a log-log plot equation would give us ln(T(n)) = (4 * ln(n)) + ln(a)

Using a data point from Brute, T(400) = 5.416 = a * (400^4)
a = 2.116 * 10^-10

Let’s confirm this is a good equation. We should get about 275 seconds for n = 1000 and less than a second for n = 200

T(200) = 2.116 * 10^-10 * (200^4) = 0.338
T(1000) = 2.116 * 10^-10 * (1000^4) = 211

Our equation is generally correct.
To estimate **T(1000000)** = 2.116 * 10^-10 * (1000000^4) = 2.116 * 10 ^ 14 seconds.

Analysis of Fast:

In Fast.java I sort the array once before getting started. The array is n elements large. That is n * ln(n) complexity to sort according to java documentation as I use their Arrays.sort() method.

Starting my main for loop, I go through all the elements in the array holding the points.
Main for loop: n

For each element, I sort a secondary copy of the original array. Each time I sort I subtract one element from the secondary array. After sorting I also go through the sorted secondary array and check for any points that in a line.
Sort array based on element and check if in line: (n * ln(n)/2) + n

Total time complexity of algorithm is f(n) = (n * ln(n)) + (n * ((n * ln(n)/2) + n))
f(n) = (n * ln(n)) + (n^2 * ln(n)/2) + (n^2)

Tilde notation of this function gives ~f(n) = n^2 * ln(n) / 2

Let T(n) be our running time for this program.

By studying a log-log plot of the data, we get a line with a slope of about 2
We know a log-log plot equation would give us ln(T(n)) = (2 * ln(n)) + ln(a)

Using a data point from Fast, T(4000) = 1.547 = a * (4000^2)
a = 9.66875 * 10^-8

Let’s confirm this is a good equation. We should get about 9.255 seconds for n = 10000

T(10000) = 9.66875 * 10^-8 * (10000^2) = 9.66

Our equation is generally correct.
To estimate **T(1000000)** = 9.66875 * 10^-8 * (1000000^2) = 96687.5 seconds.