James Rogers
605.621 Algorithms

Project 1

This looks way better in the provided .md file if you have a markdown viewer! Also, it looks better on the git repo!

Introduction

This project implements an algorithm to calculate the closest pair of points in a set of points. Additionally, this project also implements a deterministic Turing machine (DTM) class to implement DTMs.

A git repo of the project is at: https://github.com/jmr172/algorithms/tree/master/Project_1

This project was made with Java 8. It can be run from the command line using the following from within the src directory:

```
$ javac Project1.java && java Project1 ../inputs/input1.txt ../inputs/input2.txt ../output/output1.txt ../output/output2.txt
```

* input1.txt is the input for the closest pairs.
  * input files with 100, 150, 200, and 1000 data points are provided.
* input2.txt is the input for the DTMs.
  * The first line is the input for the provided DTM
  * The second line is the input for the addition DTM
  * The third line is the input for the subtraction DTM
  * The fourth line is the input for the multiplication DTM
* output1.txt contains the output of the closest pairs calculation if necessary
* output2.txt contains the output of the DTM. This will also print to the terminal if the input size is less than 30 characters.

James Rogers
605.621 Algorithms

Closest Pairs

For part 1, I implemented a brute force solution to finding the closest set of points in a set. All my data was generated by the python script I wrote called random_point_generator.py.

To run this program, use:

```

python3 random_point_generator.py [number_of_points]

```

The number of comparisons required to calculate the distance between every one of n points is given by `(n^2 - n) / 2` which is derived from the sum of 1 -> n-1.

```

Proof by induction:

For n=3, the number of comparisons is 3.

For n=4, the number of comparisons is 6.

For n=5, the number of comparisons is 10.

For n=6, the number of comparisons is 16.

If comparisons for n are calculated by `(n^2 - n) / 2`, then the number of

comparisons for n+1 is given by `((n+1)^2 - (n+1) / 2)`. Plugging in 5 for n to

both equations shows that the formula holds for both n and n+1.

```

James Rogers
605.621 Algorithms

A sample output from running my program on 100 data points is shown below:

```

The two closest points are Point 171 and Point 172 with a distance of 427.369

Total number of comparisons: 19900

```

I devised and implemented a more efficient way of calculating the closest pairs without having to calculate the distance between every point. My implementation assumes the list of points is sorted from lowest x value to greatest x value.

1. I calculate the midpoint between the x coordinate of the first point and the x coordinate of the last point. All points to the left of the midpoint get put in group A. All points to the right of the midpoint get put in group B.

2. We now have two subgroups, each of roughly the size of `n/2`. We have to calculate the distance between each point within each subgroup. Using the equation from the above proof, each subgroup will require `(n^2 - 2n) / 8` comparisons. Since there are two subgroups, there will be `(n^2 - 2n) / 4` comparisons. Return the shortest distance from each subgroup.

3. Compare the two returned distances and retain the shortest one as calculated shortest distance.

4. Next, we need to compare points from subgroup A to subgroup B. We do not

need to calculate the distance between every point. If a point is greater than

the current shortest distance away from the midpoint, it cannot be one of the

closest pair of points.

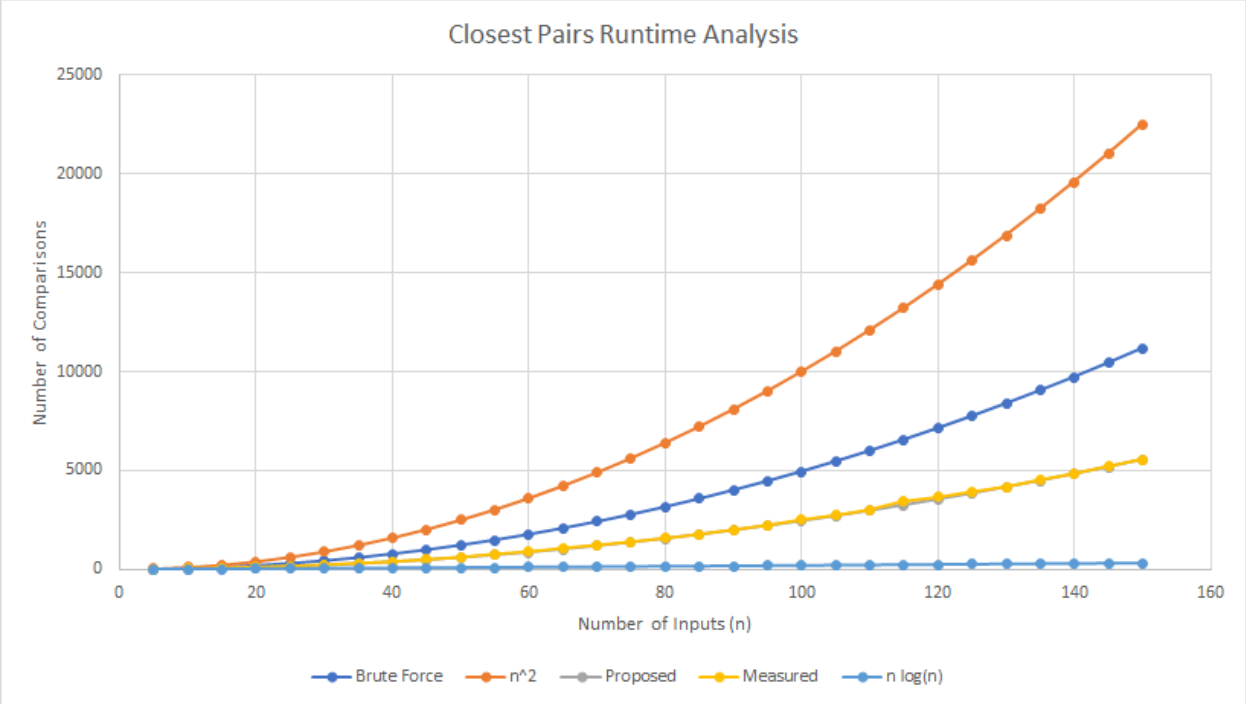
```

Proof:

If point 1 is in A and point 2 is in B, then the midpoint must lie between them. If point 1 cannot reach the midpoint by drawing a straight line that is equal to or shorter than the shortest calculated distance, then it also cannot reach any of the points beyond the midpoint in a distance that is shorter than the shortest distance. Therefore, we can skip their comparisons.
```

James Rogers
605.621 Algorithms

```

5. Calculate the distance between every point of the original input within the calculated shortest distance of the midpoint. The shortest distance is either the previously calculated shortest distance, or one of the new distances.

I measured the number of comparisons from 30 runs ranging from input size of 5-150.

The data is shown below.

| n | Brute Force | n^2 | Proposed | Measured | n log(n) |
|---|---|---|---|---|---|
| 5 | 10 | 25 | 4 | 7 | 4 |
| 10 | 45 | 100 | 20 | 24 | 10 |
| 15 | 105 | 225 | 49 | 56 | 18 |
| 20 | 190 | 400 | 90 | 100 | 27 |
| 25 | 300 | 625 | 144 | 159 | 35 |
| 30 | 435 | 900 | 210 | 214 | 45 |
| 35 | 595 | 1225 | 289 | 307 | 55 |
| 40 | 780 | 1600 | 380 | 381 | 65 |
| 45 | 990 | 2025 | 484 | 494 | 75 |
| 50 | 1225 | 2500 | 600 | 606 | 85 |
| 55 | 1485 | 3025 | 729 | 772 | 96 |
| 60 | 1770 | 3600 | 870 | 885 | 107 |
| 65 | 2080 | 4225 | 1024 | 1069 | 118 |
| 70 | 2415 | 4900 | 1190 | 1218 | 130 |
| 75 | 2775 | 5625 | 1369 | 1390 | 141 |
| 80 | 3160 | 6400 | 1560 | 1570 | 153 |
| 85 | 3570 | 7225 | 1764 | 1771 | 165 |
| 90 | 4005 | 8100 | 1980 | 2002 | 176 |
| 95 | 4465 | 9025 | 2209 | 2230 | 188 |
| 100 | 4950 | 10000 | 2450 | 2502 | 200 |
| 105 | 5460 | 11025 | 2704 | 2735 | 213 |
| 110 | 5995 | 12100 | 2970 | 3020 | 225 |
| 115 | 6555 | 13225 | 3249 | 3434 | 237 |
| 120 | 7140 | 14400 | 3540 | 3646 | 250 |
| 125 | 7750 | 15625 | 3844 | 3922 | 263 |
| 130 | 8385 | 16900 | 4160 | 4179 | 275 |
| 135 | 9045 | 18225 | 4489 | 4520 | 288 |
| 140 | 9730 | 19600 | 4830 | 4844 | 301 |
| 145 | 10440 | 21025 | 5184 | 5215 | 314 |
| 150 | 11175 | 22500 | 5550 | 5565 | 327 |
```

James Rogers
605.621 Algorithms

I then graphed the brute force approach vs my proposed implementation bounded by n^2 and n log(n), shown below.



While unable to get down to O(n log(n)), my implementation greatly outperforms the brute force solution while still returning the correct answer in all test cases. Additionally, my theoretical number of comparisons vs my measured values maps nearly perfectly, and is difficult to distinguish between the two on the graph.

Here is a sample output run on the same input as the brute force solution:

```

The final closest points are Point 171 and Point 172 with a distance of 427.369

Total number of comparisons: 10024

```

James Rogers
605.621 Algorithms

Deterministic Turing Machine

The instructor provided DTM accepts a tape consisting of 0s, 1s, and bs and returns halts in a YES state if the two digits before the first b are equal to 0.

Here is a successful run with input and each transition:

```
Running provided DTM...
1100b
1100b
1100b
1100b
1100b
1100b
110bb
11bbb
DTM ended in a YES state.
```

Here is an unsuccessful run with input and each transition:

```
Running provided DTM...
101b
101b
101b
101b
101b
10bb
1bbb
DTM ended in a No state.
```

James Rogers
605.621 Algorithms

```

Addition Table

| Q - {qY, qN} | 1 | b | x |
|---|---|---|---|
| q0 | q1, b, 1 | q0, b, 1 | q5, x, 1 |
| q1 | q1, 1, 1 | qN, b, 0 | q2, x, 1 |
| q2 | q2, 1, 1 | q3, 1, -1 | qN, x, 0 |
| q3 | q3, 1, -1 | qN, b, 0 | q4, x, -1 |
| q4 | q4, 1, -1 | q0, b, 1 | qN, x, 0 |
| q5 | qY, 1, 0 | qN, b, 0 | qN, x, 0 |

Subtraction Table

| Q - {qY, qN} | 1 | b | x |
|---|---|---|---|
| q0 | q1, b, 1 | q0, b, 1 | q7, x, 1 |
| q1 | q1, 1, 1 | qN, b, 0 | q2, x, 1 |
| q2 | q2, 1, 1 | q3, b, -1 | qN, x, 0 |
| q3 | q4, b, -1 | qN, b, 0 | q6,x,-1 |
| q4 | q4, 1, -1 | qN, b, 0 | q5, x, -1 |
| q5 | q5, 1, -1 | q0, b, 1 | qN, x, 0 |
| q6 | q6, 1, -1 | qY, 1, 0 | qN, x, 0 |
| q7 | qY, 1, 0 | qN, b, 0 | qN, x, 0 |

Multiplication Table

| Q - {qY, qN} | 1 | b | x | j |
|---|---|---|---|---|
| q0 | q2, b, 1 | q1, b, 1 | qN, x, 0 | qN, j, 0 |
| q1 | q2, b, 1 | q1, b, 1 | qY, x, 0 | qN, j, 0 |
| q2 | q2, 1, 1 | qN, b, 0 | q3, x, 1 | qN, j, 0 |
| q3 | q4, j, 1 | q7, b, -1 | qN, x, 0 | q3, j, 1 |
| q4 | q4, 1, 1 | q5, b, 1 | qN, x, 0 | qN, j, 0 |
| q5 | q5, 1, 1 | q6, 1, -1 | qN, x, 0 | qN, j, 0 |
| q6 | q6, 1, -1 | q6, b, -1 | q3, x, 1 | q6, j, -1 |
| q7 | qN, 1, 0 | qN, b, 0 | q8, x, -1 | q7, 1, -1 |
| q8 | q8, 1, -1 | q1, b, 1 | qN, x, 0 | qN, j, 0 |
```

James Rogers
605.621 Algorithms

Next, I designed and implemented a DTM that would accept and perform addition on a tape containing two unary numbers separated by an x. The addition table in the above figure displays my transitions.

1. I find a 1 and convert it to a blank.

2. Next, I find the x.

3. Then, I find the next b, and convert it to a 1.

4. Next, I return to the first b. Repeat steps 1-4 until there are no more 1's

left of the x.

Here is a successful run with padded input and each transition:

```
Running unary addition DTM...
11x111bbbbbbbb
b1x111bbbbbbbb
b1x111bbbbbbbb
b1x111bbbbbbbb
b1x111bbbbbbbb
b1x111bbbbbbbb
b1x111bbbbbbbb
b1x1111bbbbbbb
b1x1111bbbbbbb
b1x1111bbbbbbb
b1x1111bbbbbbb
b1x1111bbbbbbb
b1x1111bbbbbbb
b1x1111bbbbbbb
bbx1111bbbbbbb
bbx1111bbbbbbb
bbx1111bbbbbbb
```

James Rogers
605.621 Algorithms

bbx1111bbbbbbb

bbx1111bbbbbbb

bbx1111bbbbbbb

bbx11111bbbbbb

bbx11111bbbbbb

bbx11111bbbbbb

bbx11111bbbbbb

bbx11111bbbbbb

bbx11111bbbbbb

bbx11111bbbbbb

bbx11111bbbbbb

bbx11111bbbbbb

DTM ended in a YES state.

```

Here is an unsuccessful run with padded input and each transition:

```

Running unary addition DTM...

1x1x111bbbbbbbbb

bx1x111bbbbbbbbb

bx1x111bbbbbbbbb

bx1x111bbbbbbbbb

bx1x111bbbbbbbbb

DTM ended in a No state.

```
```

James Rogers
605.621 Algorithms

Next, I designed and implemented a DTM that would accept and perform subtraction on a tape containing two unary numbers separated by an x. The subtraction table in the above figure displays my transitions.

1. I find a 1 and convert it to a blank.

2. Next, I find the x.

3. Then, I find the first b to the right of the 1's. I go back one space and convert that 1 to a b.

4. Next, I return to the first b. Repeat steps 1-4 until there are no more 1's left of the x.

Here is a successful run with input and each transition:

```
Running unary subtraction DTM...
11x111b
b1x111b
b1x111b
b1x111b
b1x111b
b1x111b
b1x111b
b1x111b
b1x11bb
b1x11bb
b1x11bb
b1x11bb
b1x11bb
b1x11bb
bbx11bb
bbx11bb
```

James Rogers
605.621 Algorithms

bbx11bb

bbx11bb

bbx11bb

bbx1bbb

bbx1bbb

bbx1bbb

bbx1bbb

bbx1bbb

bbx1bbb

DTM ended in a YES state.

```

Here is an unsuccessful run with input and each transition:

```

Running unary subtraction DTM...

11xx111b

b1xx111b

b1xx111b

b1xx111b

b1xx111b

DTM ended in a No state.

```
```

James Rogers
605.621 Algorithms

Next, I designed and implemented a DTM that would accept and perform multiplication on a tape containing two unary numbers separated by an x. The multiplication table in the above figure displays my transitions.

1. I find a 1 and convert it to a blank.
2. Next, I find the x.
3. Then, I find the first 1 to the right of the x. I convert it to a temporary character 'j' for James.
4. Then, I find the first b to the right of the 1's.
5. Next, I find the next b and convert it to a 1.
6. I return to the x.
7. Return to 3. If no 1's are found, go to 8.
8. Convert all j's back to 1's and return to the first b on the left.
9. Repeat 1-8 until no more 1's are left of the x. The answer will be the bits appended to the input.

Here is a successful run with input and each transition:

```
Running unary multiplication DTM...
11x11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b1x11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b1x11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b1x11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b1xj1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b1xj1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b1xj1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b1xj1b1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b1xj1b1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b1xj1b1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b1xj1b1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b1xj1b1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
```

James Rogers
605.621 Algorithms

b1xj1b1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjjb1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjjb1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjjb1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjjb11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjjb11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjjb11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjjb11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjjb11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjjb11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjjb11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjjb11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xj1b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1x11b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1x11b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1x11b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1x11b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbx11b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbx11b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

James Rogers
605.621 Algorithms

bbxj1b111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb111bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxjjb1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbxj1b1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbx11b1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbx11b1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbx11b1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

bbx11b1111bbbbbbbbbbbbbbbbbbbbbbbbbbbbb

DTM ended in a YES state.

```

Here is an unsuccessful run with input and each transition:
```

James Rogers
605.621 Algorithms

```

Running unary multiplication DTM...

11x1x1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1x1x1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1x1x1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1x1x1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjx1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

b1xjx1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

DTM ended in a No state.

```