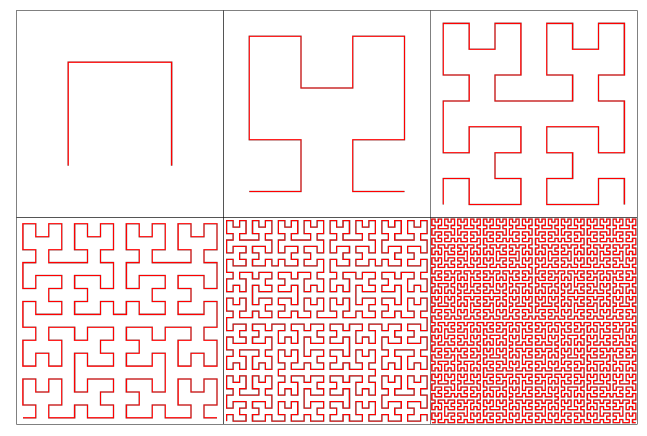
**2D Data Sorting Algorithm by Hilbert Sort**

**Problem Statement:**

Sorting is a fundamental building block for computation. For example, database systems use sorting extensively. Therefore, developing more efficient sorting algorithm is important as computer architectures evolve. (Satish, Harris, & Garland, 2009)

Applications as in GPS guidance system requires representing and store data points in a 2D grid. Efficiently query those 2D geospatial data is important. Also as computer networks become more distributed, the cost of communication through physical space needs to be addressed and the cost of communication is proportional to physical distance. (Orhai & Teuscher) Therefore, it is important to arrange spatial data that are contiguous in memory to be have similar keys.

One way of sorting 2D data points is along a continuous Hilbert Curve which better preserves distance between 2D points as shown in Fig1.



**Fig.1 Six levels of Hilbert curve**

**Background:**

A basic Hilbert curve is shown in top left picture. The square is divided into four quadrants with origin at the center. The curve starts at the lower left quadrant, move to quadrant in the top left, then to top right and at last to lower right quadrant. Hilbert curve is a recursive in nature which divides the square into four quadrants and recursively fill each quadrant with a rotated copy of a basic Hilbert curve. Given the locations of interest, they will be sorted based on the order when Hilbert curve visits them.

**Algorithm:**

My algorithm will use recursion by dividing square into quadrants to sort the data points. The base case is when there is less than 2 points in the current quadrant. The recurve case will be continue dividing the current quadrant into four quadrants and then place the points into appropriate smaller quadrant.

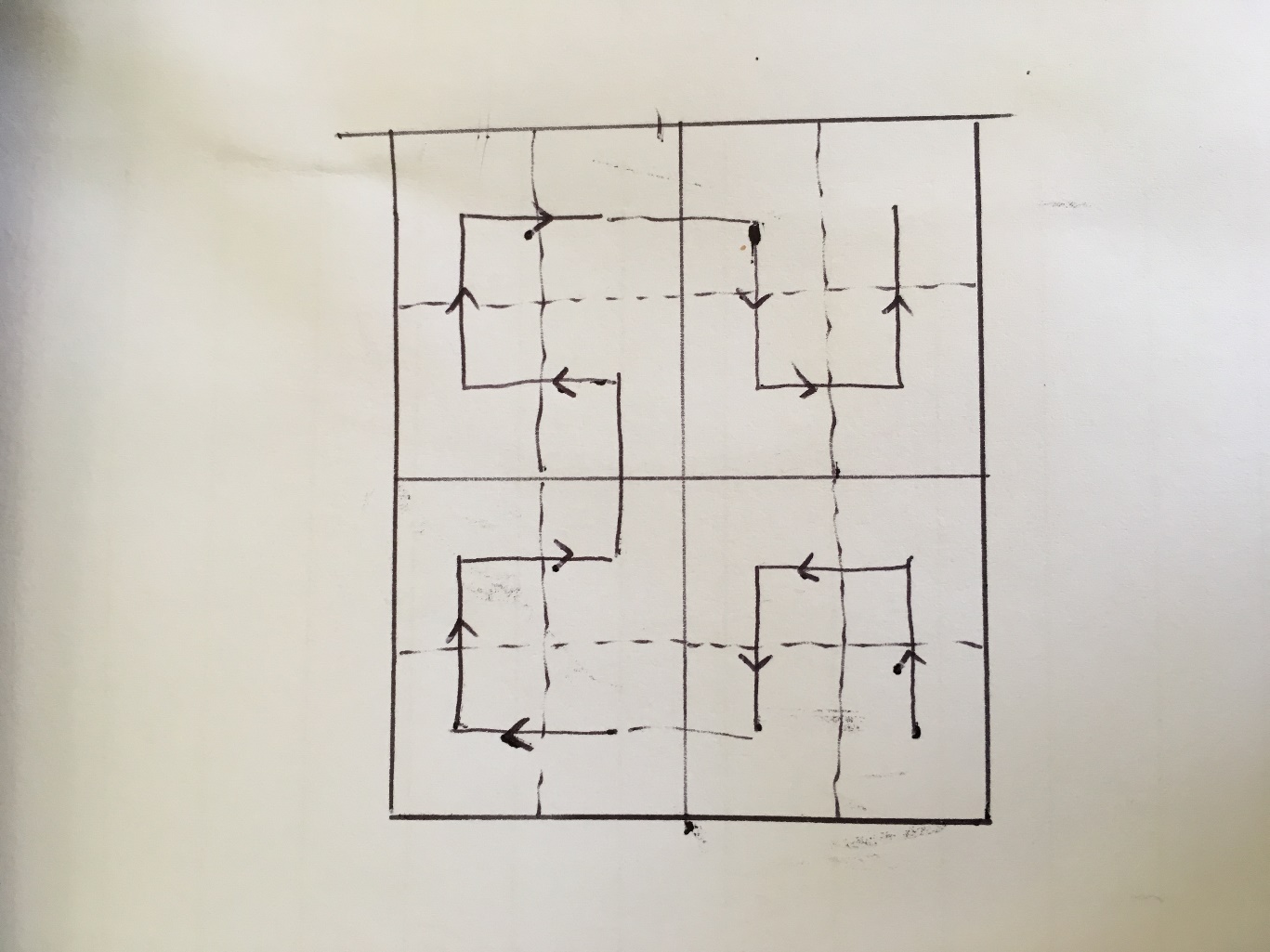
Detailed Java implementation:

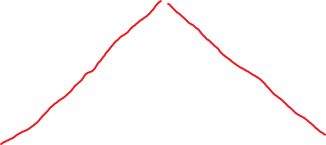
I created Location object to store each point’s x, y coordinates and identifier string. I used a scanner method to store each location of interest into a queue to be sorted. I had another method to read output using scanner. I have a class called HilbertSort which has a recursive method to sort a Queue of locations. The class has two queues as data fields. One is a queue to store unsorted locations, the other queue to store sorted items.

In the Hilbert sort recursive method, location objects are put into corresponding quadrants based on x, y values. Each quadrant is a Queue. Next, the program will visit each quadrant starting **from lower left, to top left, then to top right, lastly to lower right** as the standard order. When visiting each quadrant, if there is only one location in that quadrant, the location will be added to sorted queue. If there are more location objects in that quadrant, it makes a recursive method call while decreasing the square’s length by half.

Before making the recursive call, we need adjust the x, y coordinates of each location in that quadrant by translation or reflection along an axis. We need to translate each quadrant’s points to lower left corner so the coordinates always range from 0 to S (size of the square) and maintain their spatial relationships. For example, points in top left quadrant need to reduce y coordinates by S/2 while points in top right quadrant need to reduce both x and y coordinates by S/2 to move to lower left corner.

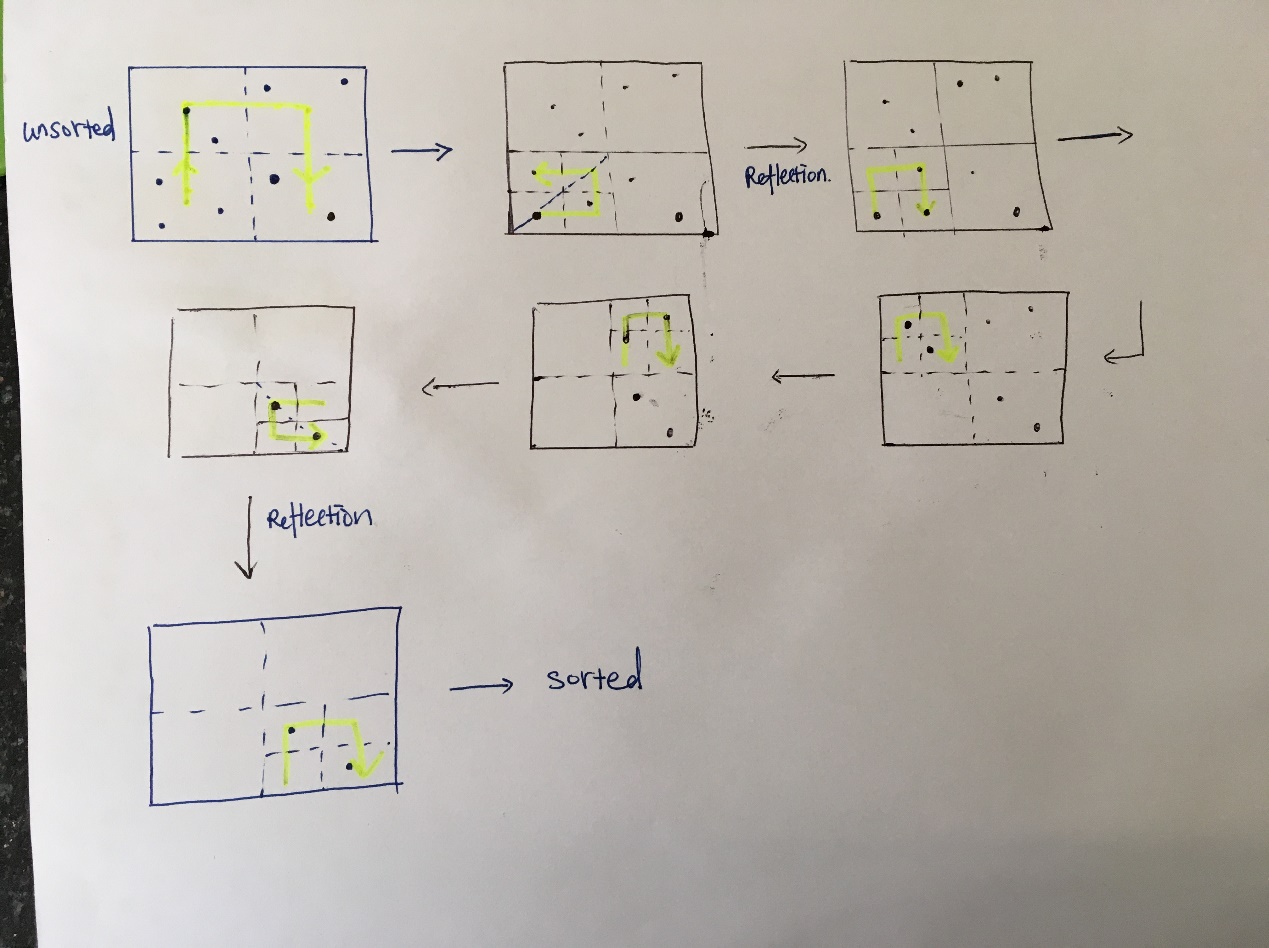
For points in lower left quadrant and lower right quadrant, we need to do reflection along an axis. As shown in Fig 2, the lower two quadrants have different orders visiting their smaller quadrants than the standard order shown in the top two quadrants. Performing a reflection for the points in lower two quadrants allows the lower two quadrants to maintain its original visiting order while the program visits in the order as in the top two quadrants. Even though, the points’ coordinates are changed due to reflection, their relative order remains the same. For example, in lower left quadrant, all the points need to perform reflection along the 45 degrees axis as shown in red line. And in lower right quadrant, the points need to first perform a left translation by S/2 distance and then a reflection along the 45 degree axis.





**Fig. 2 Diagram of Hilbert Sort order in each quadrant**

The diagram below provides an example how the algorithm sorts 2D data through recursion.



**Fig. 3 Diagram of Hilbert Sort recursion algorithm procedure**

Based on results from all test cases, the location order from Hilbert sort matches that in output file. One test case has input as shown here.

32 7

0 0 P0

0 1 P1

0 2 P2

0 3 P3

*0 4 P4*

*0 5 P5*

*0 6 P6*

*0 7 P7*

*1 7 P8*

*2 7 P9*

*3 7 P10*

4 7 P11

5 7 P12

6 7 P13

7 7 P14

7 6 P15

7 5 P16

7 4 P17

7 3 P18

7 2 P19

7 1 P20

7 0 P21

6 0 P22

5 0 P23

4 0 P24

3 0 P25

2 0 P26

1 0 P27

3 3 00Q00

3 4 00q00

4 3 00r00

4 4 00R00

The console output for test case 2000 is shown below.

*Sorted locations using Hilbert sort are [P0, P1, P27, P26, P25, 00Q00, P2, P3, P4, P5, P6, P7, P8, P9, P10, 00q00, 00R00, P11, P12, P13, P14, P15, P16, P17, P18, P19, 00r00, P24, P23, P22, P20, P21]*

*test output is [P0, P1, P27, P26, P25, 00Q00, P2, P3, P4, P5, P6, P7, P8, P9, P10, 00q00, 00R00, P11, P12, P13, P14, P15, P16, P17, P18, P19, 00r00, P24, P23, P22, P20, P21]*

*Hilbert sort list matches output file: true*

For the test case involving actual location, I used Google map to verify the Hilbert sort order and find that places that are close to each other on the map tend to sit closer to each other in Hilbert sorted queue. It confirms that Hilbert curve sort items by their relative distance from origin.

The Hilbert sort problem taught me the basic principles of how to develop a sorting algorithm and recursion algorithm. To develop a sorting algorithm, one needs to identify the ordering criteria. In our case, the order criterion is based on the position of the quadrant. Recursion needs to divide bigger problem into small steps that can be called many times till meeting base case criteria. Another important lesson is to reduce redundancy and increase efficiency by rethinking the algorithm. For example, instead of using nested if else case and each quadrant having their individual visiting order, we can transform the coordinates to use only one standard order.

# References

Orhai, M., & Teuscher, C. (n.d.). *Spatial Sorting Algorithms for Parallel Computing.*

Satish, N., Harris, M., & Garland, M. (2009). *Designing efficient sorting algorithms for manycore GPUs.* Rome: Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium.