Intersection of Line Segments Algorithm

Figure 1:

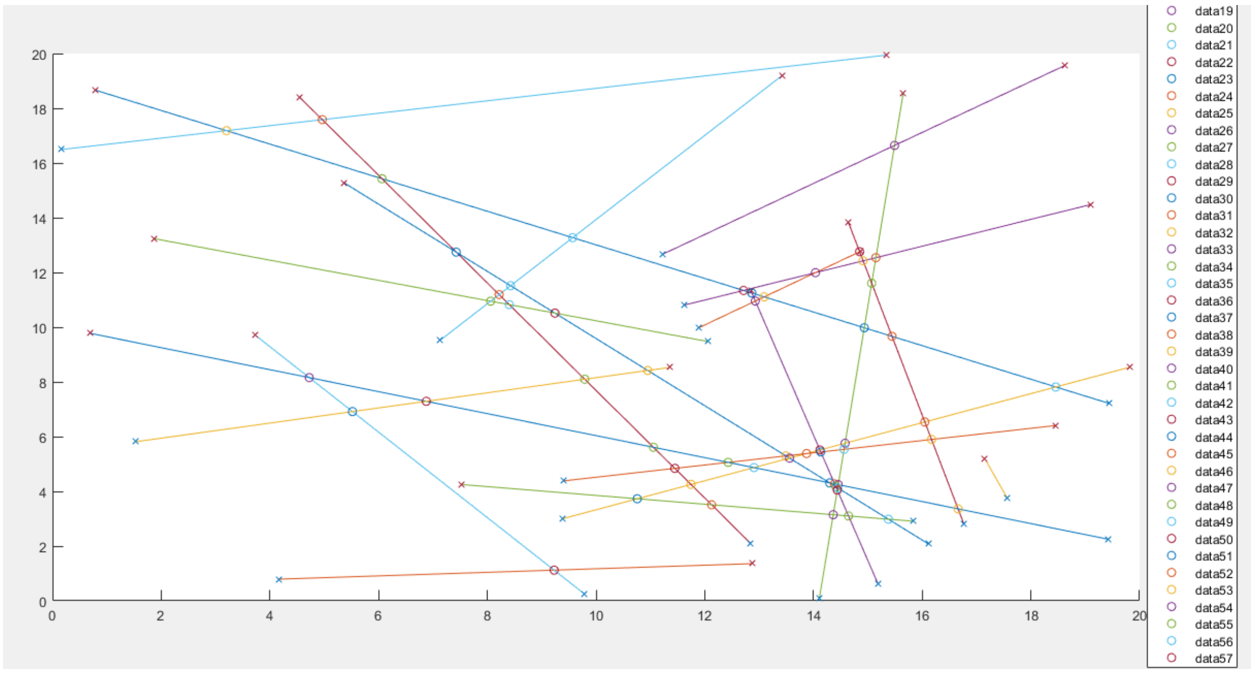
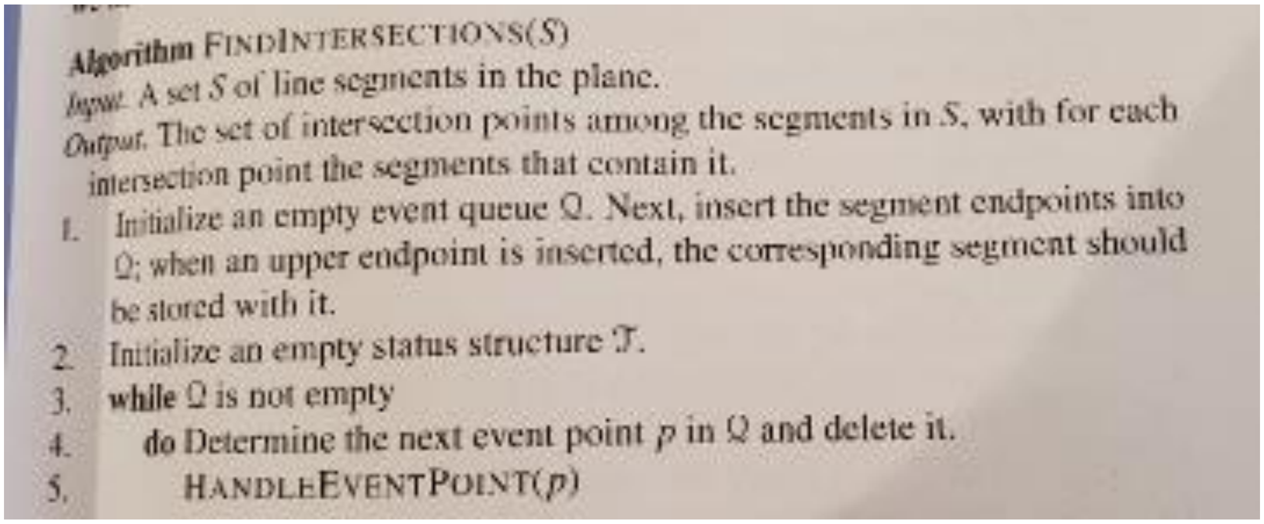


Figure 1 shows the output of the algorithm displaying the randomly generated line segments and all intersection points between them.

Figure 2:



```
Algorithm FINDINTERSECTIONS(S)
Input. A set S of line segments in the plane.
Output. The set of intersection points among the segments in S, with for each
    intersection point the segments that contain it.
1.  Initialize an empty event queue Q. Next, insert the segment endpoints into
    Q; when an upper endpoint is inserted, the corresponding segment should
    be stored with it.
2.  Initialize an empty status structure T.
3.  while Q is not empty
4.      do Determine the next event point p in Q and delete it.
5.          HANDLEEVENTPOINT(p)
```

Figure 2 shower the pseudo code for the algorithm taken from Computation Geometry Algorithms and Applications 3rd Edition written by: Mark de Berg, Otfried Cheong, Marc van Kreveld and Mark Overmars.

Event Point Handling:

For a intersection to occur they will become neighbors in the tree at some point during the sweep through the discrete event points. When Handling Event Points the first thing to do is determine they type of the event point. We either have an upper endpoint, intersection point, or lower endpoint. For upper endpoints we insert the point into a tree that is either empty or containing the segments from the last iteration. Then we determine what segments intersect the sweep line of the current iteration. Once the tree is built for the current iteration, we find the left and right neighbors of the segment corresponding the sweep line. After the left and right neighbors are found they are checked for intersection with the new line that was inserted. If an intersection is found, then it is plotted and added to the priority queue. The second type of event point Intersection points are handled by creating a new tree where the sweep line is the height of the intersection point. Only one of the lines that created the intersection are added to this tree the other is kept as well. This is because at the intersection point both lines intersect at the same point. Just below this points the left and right neighbors' swap. These lines are then checked for a new Intersection point if an intersection is found then it is added to the queue. Lastly to handle the lower endpoints. All Points intersecting the sweep line are added to a tree. Just after the lower endpoint that segment is no longer part of the tree. When this occurs the left and right neighbors become adjacent. Due to this they need to be checked for intersection. Event Points may be found more then once. In order to handle this the best way is to insert them into the priority queue when found then remove any duplicates at the start of the next iteration. This is because checking to see if the priority queue contains a point already would require looking at each element for every new event point found. Removing duplicates only requires log(n) time for each duplicate event.

Data Structures:

Red-Black Tree:

The data structure used for the tree in this algorithm is a slightly augmented red-black tree. This tree sorts the segments based of the x value of their intersection with a sweep line. The method for insert maintains the red-black property. The method for delete does not. This was done because during the algorithm the root of the tree is always deleted, and the entire tree is always deleted thus maintaining the red- black property does not increase the speed of deleting elements. Conversely insertions are always done with new elements and the red-black property must be maintained so that insertions remain O log(n) as well as the get left neighbor and get right neighbor methods.

Priority Queue:

The priority queue is made from a heap that maintains a max heap property. This data structure is used to sort the values based on their Y value. It is used to determine the next sweep line.

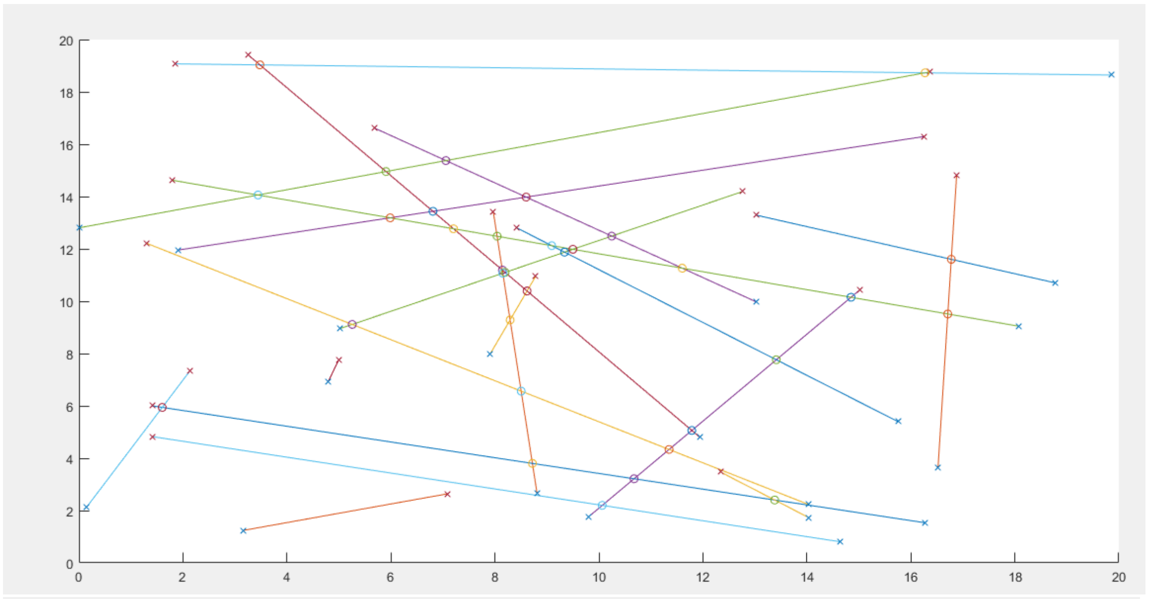
Additional Methods:

Find New Event Point:

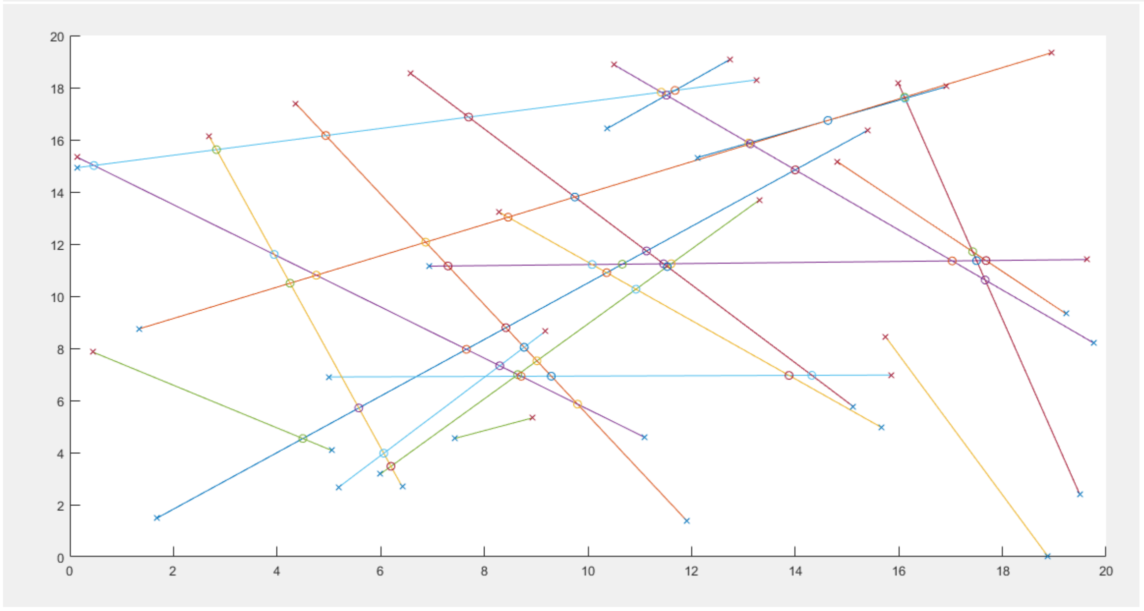
Find new Event Point takes two segments as input and outputs an intersection point and the lines that were used to find that intersection point.

Get Left/Right Neighbor:

These methods search the tree for the closet segments to the current Event Point and return the nodes that contain them.



0
2
4
6
8
10
12
14
16
18
20



0
2
4
6
8
10
12
14
16
18
20