Ben Holmes
COMS 311 HW 4

1. On Attached pdf.

2.

```
Node ArrayToBST(arr[], left, right)
{

   if (left > right)
     return;

   int mid = (left + right)/2;
   Node root = arr[mid];

   root.left =  ArrayToBST(arr, left, mid-1);

   root.right = ArrayToBST(arr, mid+1, right);

   return root;
}
```

Recurrence Relation is handwritten on attached pdf.

3. With a set S sorted by its X values

OVERVIEW:

Maintain a global array for indexes and the set S-S'. In the combine method, we compare points, and check whether the condition holds for them to be added to set S-S'. If the condition does hold, we add it to S', and then add the index to the indexes array. In the end we have set S-S', and so we use this to construct the array S'.

PSEUDO CODE:

```
Point[] S-S';
ArrayList<Integer> indexes;

combine(Point[] S, left, mid, right){
        i = left;
        j = mid + 1;
```

```
        while(i <= mid && j <= right)
            if(S[i].x < S[j].x && S[i].y < S[j].y)
                add S[i] to set S-S'
                add i to indexes;
                i++;
            else if(i == mid && j <= right)
                j++;
            else if(i <= mid && j == right)
                i++;
            else
                i++;
                j++;
    }

        public static void merge(int l, int r) {
            if(l < r) {
                int mid = l + (r-l)/2;
                merge(l, mid);
                merge(mid+1, r);
                combine(l, mid, r);
            }
        }
```

OVERVIEW OF CONSTRUCT S':

We loop through the original set S, and construct S', by adding every element from S, that isn't in set S-S'.

```
    constructS'(){
        for(int i =0; i < S.size(); i++) {
            if(!indexes.contains(i)) {
                System.out.println(S.get(i).getX() + " , " + S.get(i).getY());
            }
        }
    }
```

****** Recurrence Relation for merge and combine are on attached sheet.

In addition to that recurrence relation, must also add the time to construct S', which is O(n^2);

4. Do a DFS to get an array of all nodes:
    For each node, get its neighbors from the adjacency list:
    Check if the condition holds:
        i.e. for each neighbor, check if they also have a neighbor
        such that for edge (a,b) | E, (b,c) | E
    if condition holds, add it to G2 adjacency matrix

    Time Complexity O(nm) : For each node, we check all neighbors, thus nm.