Divide and Conquer

Below is an implementation of a divide and conquer solution to the maxima set problem. It will be discussed in lab.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef struct {
   int x;
   int y;
   int ismax;
} Point;

typedef struct {
     struct Point *points;
     int length;
} PointHolder;


int lexicompare(Point, Point);
Point findSmallest(PointHolder);
PointHolder maximaSet(PointHolder);



int lexicompare(Point u, Point v){
     int diff = u.x - v.x;
     if(diff != 0) {
          return diff;
     }
     return u.y - v.y;

}

Point findSmallest(PointHolder ph){
     int length = ph.length;
     Point *points = (Point *) ph.points;
     int i;
     int xmin = 65;
     int ymin = 65;
     Point min;
     for(i=0; i < length; i++){
          if (points[i].ismax ==0)
               continue;
          if(points[i].x < xmin){
               xmin = points[i].x;
               ymin = points[i].y;
          } else if (points[i].x == xmin && points[i].y < ymin){
               ymin = points[i].y;
          }
     }
     min.x = xmin;
```

```
    min.y = ymin;
    return min;

}

PointHolder maximaSet(PointHolder partition){
    Point l[64];
    Point g[64];
    Point *part = partition.points;
    Point pivot = part[0];
    int diff, i;
    int count, lcount = 0, gcount = 0, pcount=0;

    if (partition.length <= 1){
        return partition;
    }

    for(pcount = 1; pcount < partition.length; pcount ++) {
        if (part[pcount].ismax ==0)
            continue;
        diff = lexicompare(pivot, part[pcount]);
        if (diff >= 0) {
            l[lcount] = part[pcount];
            lcount++;
        }else{
            g[gcount] = part[pcount];
            gcount++;
        }
    }
    g[gcount] = pivot;
    gcount++;
    PointHolder lholder, gholder;
    lholder.points = l;
    lholder.length = lcount;
    gholder.points = g;
    gholder.length = gcount;


    PointHolder lmax = maximaSet(lholder);
    PointHolder gmax = maximaSet(gholder);
    Point min = findSmallest(gmax);


    Point * lmaxpoints = lmax.points;
    Point * gmaxpoints = gmax.points;

    for(i=0; i < lmax.length; i++) {
        if (lmaxpoints[i].x <= min.x && lmaxpoints[i].y <= min.y){
            lmaxpoints[i].ismax = 0;
        }
    }

    Point *points = (Point *)malloc((gmax.length+lmax.length)*sizeof(Point));
```

```
	count=0;
	for(i=0; i < lmax.length; i++){
		if(lmaxpoints[i].ismax){
			points[count] = lmaxpoints[i];
			count++;
		}
	}

	for(i=0; i < gmax.length; i++) {
		if(gmaxpoints[i].ismax){
			points[count] = gmaxpoints[i];
			count++;
		}
	}

	PointHolder u;
	u.length = count;
	u.points = points;
	if (lmax.length > 1)
		free(lmaxpoints);
	if (gmax.length > 1)
		free(gmaxpoints);
	return u;

}


int main() {
	Point mypoints[64];
	PointHolder maxima;
	PointHolder initial;
	int i;
	srand(time(NULL));

	for(i=0; i < 64; i++) {
		mypoints[i].x = (rand() % 64);
		mypoints[i].y = (rand() % 64);
		mypoints[i].ismax = 1;
		printf("%d,%d;\n", mypoints[i].x, mypoints[i].y);
	}

	initial.points = mypoints;
	initial.length = 64;

	maxima = maximaSet(initial);
	printf("\n---------------\n\n");
	Point *maximapoints  = maxima.points;
	for(i=0; i < maxima.length; i++) {
		printf("%d,%d;\n",(maximapoints[i]).x, (maximapoints[i]).y);

	}

	if (maxima.length > 1)
```

```
        free(maxima.points);

    return 0;
}
```