JACKIE MCANINCH
CS 430

FINAL PROJECT PSEUDOCODE

* Let L be a collection of lines added to the graph to separate points, either horizontal or vertical. In actual code this was done with a dictionary, with key k signifying a line with an x or y value of k, and value 0, 1, or 2:

    0: a horizontal line exists at y = k
    1: both a horizontal <u>and</u> vertical line exist at x = k and y = k
    2: a vertical line exists at x = k

This is done to achieve quick lookup and improve runtime

```
def LOCAL_OPTIMIZATION(P, n): # P is set of pts, n is number of pts
    for i in range(1, n):
        ADD_LINE(i+.5, hor.)          # O(1) operation
    notDone = True
    while notDone:
        notDone = False
        for line1 ∈ L:
            success = False
            REMOVE_LINE(line1)     # O(1)
            for line2 ∈ L:
                REMOVE_LINE(line2)     # O(1)
                for j in range(1, n):
                    if TRY_LINE(j+.5, vert.) or TRY_LINE(j+.5, hor.):
                        success = notDone = True
                        break
                if success == False:
                    ADD_LINE(line2)
                else: break
            if success == False:
                ADD_LINE(line1)
```

DESCRIPTION:

- Lines 1 and 2 are the initialization — the algorithm starts with a horizontal line in between each point (feasible solution)
- Lines 3-5: boolean notDone keeps track of whether any permanent changes were made to L. If there is, the optimization loop must run again in case a line is added to a region that has already been checked
- Lines 6-10: 2 lines are permuted and chosen to be removed
- Lines 11-14: first checks if adding a vertical line at j+.5 will result in another feasible solution, and if not, check if adding a horizontal line at the same value results in a feasible solution. If one is successful, the loop breaks and looks for a new combination. notDone and success both set to True
- Lines 15-19: if all combinations for a line have been exhausted and no success has been found, that line is restored and a new one will be removed.

METHODS:

- ADD_LINE(): adds or updates dictionary entry. $\underline{O(1)}$
- REMOVE_LINE(): deletes or updates (eg. $1 \to 0$). $\underline{O(1)}$
- TRY_LINE(): calls ADD_LINE ($O(1)$), then evaluates if all pts are isolated by dividing the graph into the regions created by the lines, and checking every x and y coordinate to count the number of points inside. If successful, return True, else REMOVE_LINE. Thus total runtime = $O(1) + \underline{O(n^2)} + O(1)$

every coordinate must be checked

note:
*this description only describes the logical structure of the program. Source code contains different methods but follows the structure

JACKIE MCANINCH
CS 430

FINAL PROJECT RUNTIME ANALYSIS

LINE BY LINE ANALYSIS

- 1-2: runs (n-1) times to init. $O(n)$.
- 4-19: worst case, the size of L is decreased by 1 every iteration. The lowest size L can be and still separate all pts is $2(\sqrt{n}-1)$, so $O(n)$.
- 6-19: this for loop runs approximately 1 iteration for every line in L (approximate because some are added during). $O(n)$
- 9-19: same as above. $O(n)$
- 11-14: runs (n-1) times at most. $O(n)$
- 12: as mentioned on previous page, the cost to test if all points are isolated is $O(n^2)$
- all other operations are $O(1)$

TOTAL THEORETICAL RUNTIME: $\cancel{O(n)+}\left(O(n)\cdot O(n)\cdot O(n)\cdot O(n)\cdot O(n^2)\right)$
$= O(n^6)$

EXPERIMENTAL RUNTIME

This estimate is much higher than the algorithm actually takes. Due to the random distribution of points, probability states it is highly unlikely that the while loop at 4 will run more than 2 or 3 times. Therefore the actual average case will be closer to $O(n^5)$, at least for small/medium n values.
Actual runtimes are listed below:

n	time (s)
5	0.0019
10	0.0268
20	0.4804
27	1.2072
30	1.4898
100	133.0074
150	1021.2179

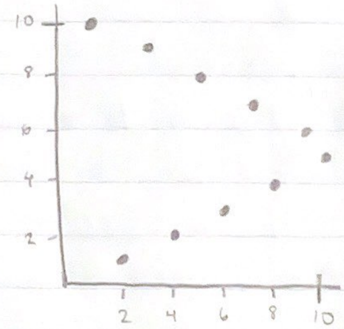
results follow the model mx^b, where $m = 1.755\cdot10^{-8}$ and $b = 5.03$ with an R^2 value of 1.

This confirms the hypothesis that runtime is $O(n^5)$ for small/medium n, but remains $O(n^6)$ for $n \to \infty$.

JACKIE MCANINCH
CS 420
3

FINAL PROJECT SUBOPTIMAL SOLUTION

INSTANCE:



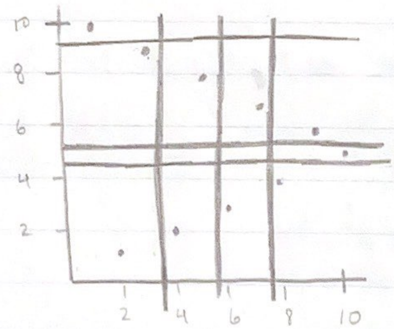
labeled as "instance00.txt" in project

ALGORITHM RUN:

1. start with all horizontal lines dividing between pts.
2. lines $y = 1.5$ and $y = 8.5$ are replaced with $x = 3.5$
3. lines $y = 2.5$ and $y = 7.5$ are replaced with $x = 5.5$
4. lines $y = 3.5$ and $y = 6.5$ are replaced with $x = 7.5$

Result:

6 lines total



OPTIMAL SOLUTION:

The intuitive solution would be to place a vertical divider after every 2 points, so that each region has 1 high point and 1 low point. Then, 1 horizontal line can be added to separate each of these pairs for a total of only 5 lines:

5 lines total

min possible, because 4 lines only creates 9 regions for 10 pts.