Fall 2023 COP 3502 Section 4 Final Exam - Part A (25 pts) 12/7/2023 Solution

For this portion of the exam, you'll complete a solution to the following problem:

There are **n** points of interest on the Cartesian grid for this problem. You are an Uber driver starting at the first pointed listed amongst these **n** points of interest. You will make **n**-1 trips, visiting each of the other locations exactly once. The amount of money you make for a single trip is the square of the distance between the two points you are traveling. Since you want to maximize profit, the location you visit from your current location will always be the farthest location that has yet to be visited, from your current location. If there is a tie between multiple locations, go to the one with the minimum x coordinate. If there is still a tie, go to the one with the smallest y coordinate. Here is the input/output format:

First line will contain a single positive integer, **n** ($2 \leq \mathbf{n} \leq 1000$), the number of points of interest.

The following **n** lines will contain two space separate integers each: $\mathbf{x_i}$ ($0 \leq \mathbf{x_i} \leq 10^4$) and $\mathbf{y_i}$ ($0 \leq \mathbf{y_i} \leq 10^4$), respectively, representing the x and y coordinate of the i^{th} point ($1 \leq i \leq \mathbf{n}$). The output will be a single integer, the money you'll make following the directions stated.

Here is a skeleton of the solution, including function prototypes and some parts of main:

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

long long distsq(long long* pt1, long long* pt2);
int farthest(int curI, int* used, long long** pts, int n);
int beat(long long** pts, int curI, int i, int j);

int main() {

    int n;
    scanf("%d", &n);
    long long** pts = calloc(n, sizeof(long long*));
    for (int i=0; i<n; i++) pts[i] = calloc(2, sizeof (long long));

    for (int i=0; i<n; i++)
        scanf("%lld%lld", &pts[i][0], &pts[i][1]);

    /* Allocate and initialize used array. (Question 2) */
    int curI = 0;
    long long res = 0;

    for (int i=0; i<n-1; i++) {
        /* Fill in code for traveling each subsequent path. (Question 4) */
    }

    printf("%lld\n", res);
    /* Free dynamically allocated memory. (Question 5) */
    return 0;
}
```

Thus, the (x, y) points are stored in a 2 dimensional array of size n by 2. The coordinates are stored as long long to avoid overflow that might occur with int. The logic for the beat function is complicated, so that is given to you below:

```
int beat(long long** pts, int curI, int i, int j) {

    long long d1 = distsq(pts[curI], pts[i]);
    long long d2 = distsq(pts[curI], pts[j]);

    if (d1 > d2) return 1;
    if (d1 == d2 && pts[curI][0] < pts[i][0]) return 1;
    if (d1 == d2 && pts[curI][0] == pts[i][0] && pts[curI][1] < pts[i][1])
        return 1;

    return 0;
}
```

Basically, this function returns 1, if, when we are currently at point curI (the point stored in the 1D array pts[curI]), traveling to point i (pts[i]) is "better than" traveling to point j (pts[j]), according to the criteria we are using to select the next point to travel to.

For this question, you'll complete the distsq function, a portion of the farthest function, and the missing portions of main.

1) (5 pts) Complete the distsq function. This function should return the square of the distance between the two Cartesian points pointed to by pt1 and pt2. (Note: pt1[0] and pt2[0] store x coordinates for the 2 points and pt1[1] and pt2[1] store the y coordinates for the two points.) **No credit will be given if any floating point operations are used at all.**

```
long long distsq(long long* pt1, long long* pt2) {
    return (pt1[0]-pt2[0])*(pt1[0]-pt2[0]) +
           (pt1[1]-pt2[1])*(pt1[1]-pt2[1]);
}
```

Grading: 1 pt diff x, 1 pt square it without pow,
1 pt diff y, 1 pt square it without pow, 1 pt for add those and return

2) (5 pts) In main, the used array will keep track of which points out of the n will be visited. It should be an integer array. Dynamically allocate space for this array and set all elements to 0 except for the item at index 0, which should be set to 1, since this is the starting location. (This should be 2 lines of code.)

```
    int* used = calloc(n, sizeof(int));
    used[0] = 1;
```

Grading: array name can be anything as it's not referenced at all in given code.
4 pts first line - 1 pt dec, 1 pt malloc/calloc, 1 pt params, 1 pt fill all 0
1 pt second line (1 pt to set used 0 to 1.)

3) (6 pts) Now you will complete the farthest function. This takes in the current index of the point you are at, the used array indicating which points have already been visited, the points array itself, and the length of the points array. The goal of this function is to return the **index** of the farthest point from pts[curI] that hasn't yet been used/visited, applying the previously stated tie-breakers. Your code **must call** the beat function to earn any credit. Initially, we will set the result to -1 and update it as necessary. Please place all of your code in the for loop.

```
int farthest(int curI, int* used, long long** pts, int n) {

    int res = -1;
    for (int i=0; i<n; i++) {

        if (used[i]) continue;

        if (res == -1) res = i;
        else if (beat(pts, curI, i, res))
            res = i;
    }

    return res;
}
```

Grading: 1 pt skipping used, 2 pts setting res to I if res was -1,
3 pts for last case (note cases 2, 3 can be combined via short circuiting)

4) (5 pts) In main, fill in the body of the for loop that is missing. Four things need to be done: (1) calculate the index of the next point to travel to via function call. (2) Mark the appropriate point as used, (3) Update the total result of money made, (4) Update the current index location. Please write these four lines of code below:

```
    int nextI = farthest(curI, used, pts, n);
    used[nextI] = 1;
    res += distsq(pts[curI], pts[nextI]);
    curI = nextI;
```

Grading: 1 pt for lines 1, 2, 4, 2 pts for line 3, several orders of these lines work.

5) (4 pts) Please free the memory dynamically allocated for the array pts and the used array which you allocated memory for in question 2. You should have 4 lines of code.

```
    for (int i=0; i<n; i++)
        free(pts[i]);
    free(pts);
    free(used);
```

Grading: 1 pt for each line, pts must be freed after pts[i]…