CMPT125, Spring 2022

Homework Assignment 3
Due date: Friday, March 11, 2022, 23:59

You need to implement the functions in ***assignment3.c***.
Submit only the ***assignment3.c*** file to CourSys.

Solve all 3 problems in the assignment.

The assignment will be graded automatically.
Make sure that your code compiles without warnings/errors, and returns the required output.

Your code MUST compile in CSIL with the Makefile provided.
If the code does not compile in CSIL, the grade on the assignment is 0 (zero).
Even if you can't solve a problem, make sure the file compiles properly.

Warning during compilation will reduce points.
More importantly, they indicate that something is probably wrong with the code.

Memory leak during execution of your code will reduce points.
Check that all memory used for intermediate calculations are freed properly.

Your code must be readable, and have reasonable documentation, but not too much.
No need to explain i+=2 with // increase i by 2.

An example of a test file is included.
Your code will be tested using the provided tests as well as additional tests.
Do not hard-code any results produced by the functions as we will have additional tests.
You are strongly encouraged to write more tests to check your solution is correct, but you don't have to submit them.

1. You need to implement the functions in ***assignment3.c***.
2. If necessary, you may add helper functions to the assignment3.c file.
3. You should not add main() to assignment3.c, because it will interfere with main() in the test file.
4. Submit only the ***assignment3.c*** file to CourSys.

Question 1 [30 points]

Write a function that gets an array of 2d-points (see definition in assignment3.h) of length n, and sorts the array using qsort().
Given two points $a=(a_x,a_y)$ and $b=(b_x,b_y)$ we compare them as follows:

1) *if $|a_x|+|a_y| < |b_x|+|b_y|$, then a should come before b in the sorted array.*
2) *if $|a_x|+|a_y| = |b_x|+|b_y|$, then the point with the lower x coordinate should come first.*
3) *if $|a_x|+|a_y| = |b_x|+|b_y|$, and $a_x=b_x$, then the point with the lower y coordinate comes first.*

Remark: For a point $a=(a_x,a_y)$ the quantity $|a_x|+|a_y|$ is sometimes called the "L_1-norm of a" or the "Manhattan distance" of the point (a_x,a_y) from (0,0).

That is, we sort the points according to their Manhattan distance from (0,0) in the increasing order, and for points at the same distance, then we sort them according to the x-coordinate, and if also the x-coordinates are equal, we sort according to the y-coordinates.

You need to implement the comparison function, and apply qsort() on the array with this comparison function.
For example:

- *Input: [(3,2), (7,1), (1,1), (-3,4), (-5,0), (-6,2), (-3,4)]*
- *Expected output: [(1,1), (-5,0), (3,2), (-3,4), (-3,4), (-6,2), (7,1)]*

Explanation:
(1,1) has the smallest norm, because 1+1=2
Then we have (3,2) and (-5,0). Both have the same L_1-norm, so we compare them by their x-coordinate: (-5,0) needs to come before (3,2).
Next comes (-3,4) because its L_1-norm is 7.
(-6,2) and (7,1) have both L_1-norm equal to 8, and -6<7.

***Note that we do allow some points in the array to be equal.*

```
void sort_points(point2d* A, int n);
```

Question 2 [30 points]

Write a generic implementation of insertion sort.

```
// the function gets an array of length n of objects of given size
// and a compare function
// The function applies Insertion Sort on the array
// using the compare function
// The function returns the number of swaps made by the algorithm
// if compare (a,b)<0, then a must come before b in the sorted array
// if compare (a,b)>0, then a must come after b in the sorted array
int gen_insertion_sort(void* array, int n, size_t size,
                       int compare(const void*, const void*));
```

***Note that the order of the elements in the sorted array depends on the compare function.*

Question 3 [40 points: 10 points each]

Write the following three functions:

*a) The function g*ets an array A, two indices of the array, and a boolean predicate pred.
It returns the smallest index i between start and end (including start and end themselves), such that pred(A[i])==true.
If there is no index i∈[start,end] with pred(A[i])==true, the function returns -1.
You may assume start<= end

```
int find(int* A, int start, int end, bool (*pred)(int));
```

*b) The function g*ets an array A, two indices of the array, and a boolean predicate pred.
It returns the number of indices i between start and end (including start and end themselves), such that pred(A[i])==true.
You may assume start<= end

```
int count(int* A, int start, int end, bool (*pred)(int));
```

*c) The function g*ets an array A, two indices of the array, and a function f.
It applies f to each element of A with indices between start and end (including start and end).
You may assume start<= end

```
void map(int* A, int start, int end, int (*f)(int));
```

d) *The function g*ets an array A, two indices of the array, and a function f. The function f gets 2 ints and works as follows:

1. Start with accumulator = A[start]
2. For i=start+1...end
   accumulator=f(accumulator, A[i])
3. Return accumulator

For example, if f computes the sum of the two inputs, then reduce() will compute the sum A[start] + A[start+1] +...+A[end].
You may assume start<= end

```
int reduce(int* A, int start, int end, int (*f)(int,int));
```

**If start==end, the loop is not executed, and the function just returns accumulator=A[start].