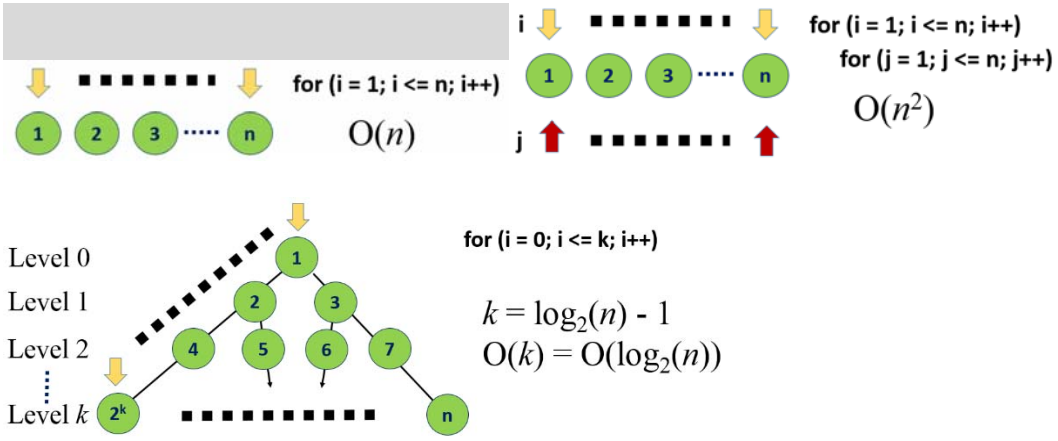
Practice 10 (2015/12/8)

Geometry problem starting from a LINE

1. Time Complexity

Time complexity of an algorithm (method) quantifies the total amount of time needed by the algorithm to run as a function of the problem instance size.



The efficiency ranking for time complexity of algorithms is: O(1), O($\log_2(n)$), O(n), O($n\times\log_2(n)$), O(n^2).

2. Problem

Calculating the overlapping of line segments is the fundamental problem of computational geometry. In this problem, we consider the simplified line overlap problem – how to calculate the overlapping length of horizontal line segments.

In this problem, all horizontal lines have no *y*-coordinate, and they can be regarded to have the same *y*-coordinate. Thus the overlapping check can be achieved by only examining the *x*-coordinates of two endpoints of every line segment. You have to identify all two line pairs with overlap and sum up the total overlapping length. For example, there are three line segments and any two line segments overlap. Thus you have to find three overlapping lengths and then sum up these three overlapping lengths. Two-loop codes can simply solve this problem, but its time complexity is O(n^2), where n is the number of line segments. The runtime will increase substantially as n increases. The time complexity of the optimal solution to this problem is O($n\log_2 n$). All benchmarks for this problem should be able to be solved within 0.156 second. Please design an algorithm to calculate the overlapping length of a horizontal line set and guarantee to solve each benchmark within 2 seconds (including 2 seconds). Any test for a benchmark over 2 seconds is regarded to fail.

Input Format:

There are two integers in a line, where the first integer is the *x*-coordinate of left endpoint and the second integer is the *x*-coordinate of right endpoint. Two integers are separated by space character. Notably, the number of space characters is not fixed,

do not use space-sensitive instruction to read input file.

Input example 1: there are four horizontal line segments and their input content is as follows. *overlap_len*(*i*,*j*) is the overlapping length of line *i* and line *j*. The total overlapping length for these four line segments is: *overlap_len*(1,2)+ *overlap_len*(1,3)+ *overlap_len*(2,3)=45+155+20=220. Notably, the overlapping length between lines 1 and 4 is 0.

75 325
5 120
100 255
325 500

Output format: Please print out the total overlapping length of the line set.

Output example: the output content of the above example is as follows. Since the total overlapping length may exceed the maximum value of an integer, please use a double-type number to output your result.

220

Main features:

(a) Use the following structure to store all lines.

```
typedef struct {
    int left, right;
} Line;
```

(b) Write a brute-force program to calculate the total overlapping length of lines. For each line, you have to compare this line with all the other lines. The program should be run like: line_overlap line_fileName

The time complexity of this program should be O(n^2), where *n* is total number of lines. See how much time you need to finish the program execution. (due tonight)

(c) Sort all lines in increasing order of *x*-coordinate of each line's left endpoint first. Then for each line in the sorted line list, you compare this line (say line *A*) with subsequent lines in the list until the first line (say line *B*) in the list whose *x*-coordinate of left endpoint is larger than or equal to the *x*-coordinate of right endpoint of line *A* occurs. The time complexity has been reduced to O($n \times m$), where *m* is the total number of lines before the occurrence of line *B*. Although it still looks like a time complexity of O(n^2), generally *m* is very small as compared to *n*. Then the execution time can be saved largely. Since you only know the sorting algorithm running in the complexity of O(n^2), you can compute the time (*T*1) for sorting and the time (*T*2) for the remaining operations

individually. Compare the time *T*2 with the time in feature (b). Also compare the time (*T*1 + *T*2) with the time in feature (b). (due in 12/14)

3. Bonus

Suppose you already learn a new sorting algorithm running in the time complexity of O($n \times \log_2 n$), then time *T*1 can be saved largely as problem complexity is huge. But there is still some room for further improvement. Consider the case to sort each endpoint of a line but not only the left endpoint, i.e., you sort left and right endpoints of all lines in a list. You also design a structure, called *End_Pnt* as follows, to store the relationship between endpoints. Pointer *pair* is used to link another endpoint of the same line while pointers *next* and *prev* are used to build a double linked list. This structure is obviously much larger than a simple structure. However sometimes a complex structure can help you to diminish the runtime required for your program. This is so called to save your time at the cost of your memory. Design a method to use or even refine this structure and write a program to check if the performance is improved. By how, how much and why? Write a report and send the report along with your program to ylli@cs.nctu.edu.tw before the midnight 12/11.

```
struct End_Pnt {
  enum {LEF, RIG} l_or_r;
  union {
    Int leftX;
    Int rightX;
  } XPos
  struct End_Pnt *pair;
  struct End_Pnt *next,
*prev;
};
```

```
struct DblLnk {
    ...
    struct *next, *prev;
};
```