Due: 24 May 2024 at 11:59pm

COMP2823 – Assignment 5

All submitted work must be done individually without consulting someone else's solutions in accordance with the University's "Academic Dishonesty and Plagiarism" policies.

Questions

Question 1 [8 pts] Solve the following recursions. Explain your answers. You can use any technique that we have discussed in the lectures (unrolling, master theorem, ...).

Note: You may assume that $T(1) = 1$, log refers to log base 2 and $n = 2^m$.

1. $T(n) = 3T(n/2) + n^2$
2. $T(n) = 4T(n/2) + n$
3. $T(n) = T(n-1) + n$
4. $T(n) = 2T(n/2) + n \log n$
5. $T(n) = \sqrt{2}T(n/2) + \log n$
6. $T(n) = 3T(n/3) + \sqrt{n}$
7. $T(n) = 7T(n/3) + n^2$
8. $T(n) = T(\sqrt{n}) + \log n$

Question 2 [32 pts]

The next two questions (2.1 and 2.2) in this assignment will look at the top skeleton problem. The *top skeleton* of a set S of n line segments in the plane is an important concept for many applications, including visibility and motion planning. The segments are regarded as opaque obstacles, and their top skeleton consists of the portion of the segments visible from the point $(0, \infty)$. We will study the special case where all the segments in S are horizontal segments with y-coordinates greater than or equal to 0.

An example of the input is given in Fig. 1(left); the corresponding output is given in Fig. 1(right).

A horizontal segment s_i in S is represented by the triple (l_i, r_i, h_i) where l_i and r_i denote the left and right x-coordinates of the segment respectively, and

h_i denotes the y-coordinate of s_i. The input is a list of triples; one per segment. The output is the top skeleton specified as a list of horizontal segments arranged in order by their left x-coordinates. For the example shown in Fig. 1, the input and output are:

Input			Output		
0	3	3	0	2	3
2	4	4	2	4	4
1	5	2	4	5	2
6	7	4	5	6	3
5	8	3	6	7	4
9	11	3	7	8	3
9	12	1	9	11	3
			11	12	1

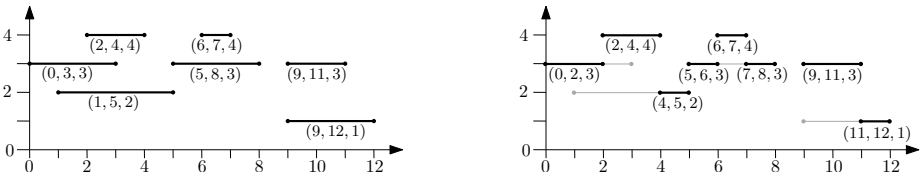

Figure 1: (left) Example of input. (right) Illustrating the output.

1. We will solve the top skeleton problem using a divide-and-conquer approach. Recall from the lecture that a divide-and-conquer algorithm works by recursively breaking down a problem into smaller subproblems of the same type, until these become simple enough to be solved directly. The solutions to the subproblems are then combined into a solution to the original problem. Your task for Question 2.1 is to handle the combine step. That is, as a subroutine to the divide-and-conquer algorithm (Question 2.2) you will need `CombineSkeletons`(H_1, H_2) that takes two top skeletons as input and produces a single top skeleton H of them. An example is shown in Fig. 2, where two top skeletons are given as input and "combined" into a single top skeleton.

   You may assume that the two top skeletons passed as arguments to `CombineSkeletons` are given as a list of horizontal segments ordered from left to right.

   (a) Description of how your algorithm works ("in plain English"). For full points the algorithm should run in $O(n)$ time where n is the total number of segments in H_1 and H_2. [7 points]

   (b) Argue why your algorithm is correct. [5 points]

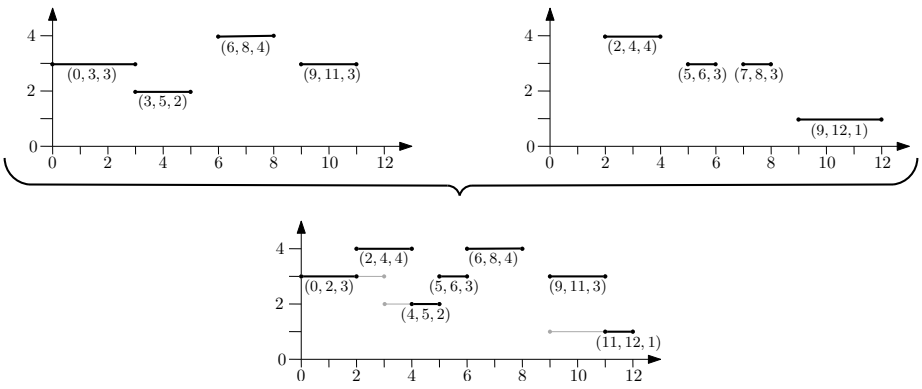

Figure 2: Illustrating `CombineSkeletons` that takes two top skeletons of horizontal segments as input and "combine" them into a single top skeleton.

(c) Prove an upper bound on the time complexity of your algorithm. [4 points]

2. Using the function `CombineSkeletons`(H_1, H_2) from Question 2.1 your task is to design a Divide-and-Conquer algorithm that computes the top skeleton of a set S of n horizontal line segments in the plane. Each segment $s_i \in S$ is represented (as above) as a triple (l_i, r_i, h_i). An example of the input instance is given in Fig. 1(left) and the top skeleton is highlighted in Fig. 1(right).

   The output must follow the input format to `CombineSkeletons` stated in Question 2.1, that is, list the horizontal segments in the top skeleton from left to right. For example, the top skeleton shown Fig. 1(right) must be listed as:

   ```
   0 2 3
   2 4 4
   4 5 2
   5 6 3
   6 7 4
   7 8 3
   9 11 3
   11 12 1
   ```

   (a) Description of how your algorithm works ("in plain English"). For full points (and assuming that `CombineSkeletons` runs in $O(n)$ time) the algorithm should run in $O(n \log n)$ time where n is the number of segments in S. [7 points]

(b) Argue why your algorithm is correct. [7 points]

(c) Prove an upper bound on the time complexity of your algorithm. [2 points]

Question 3 [20 pts]

As input you are given a *directed* graph $G = (V, E)$ embedded in the Euclidean plane, with n vertices. That is, every vertex has a position in the plane (x and y-coordinates) and every directed edge is a straight-line directed segment between two vertices. The weight of an edge is a positive integer.

No two edges of the graph intersect, so it is said to be a non-crossing graph.

Your task is to develop a divide-and-conquer algorithm for computing the length of the shortest directed cycle in the graph (sum of the edge weights in the cycle).

To help you develop an efficient algorithm you may use the following facts:

- The number of edges in a non-crossing graph $G = (V, E)$ is $O(|V|)$.
- Any subgraph of a non-crossing graph is also a non-crossing graph.
- There exists a partition of the vertices of every non-crossing graph G into three sets A, S and B, such that each of A and B has at most $2n/3$ vertices, S has $O(\sqrt{n})$ vertices, and there are no edges with one endpoint in A and one endpoint in B. The set S is called a *balanced splitter* of G. Three examples are shown in Fig. 3.

  A balanced splitter of a non-crossing graph can be computed in $O(n)$ time.

  The intuition for the balanced splitter is that it is a relatively small set of vertices ($O(\sqrt{n})$), which (when removed) separates the graph into relatively balanced pieces, see Fig. 3(left).

  For example, in a balanced tree the root would be a balanced splitter (see Fig. 3(middle)) and in a square grid graph (see Fig. 3(right)) a column or a row near the centre would be a balanced splitter.

Advice:

- Start by thinking about how to compute a shortest directed cycle from a given vertex $v \in V$.
- The "combine" step: How do you compute the shortest directed cycle in G if you know the shortest cycle in A and the shortest cycle in B?
- A naive approach runs in roughly $O(n^2 \log n)$ time. Your aim should be to develop an algorithm with $O(n^{3/2} \operatorname{polylog} n)$ running time. However, any subquadratic algorithm will be given marks.

(a) Description of how your algorithm works ("in plain English"). [9 points]

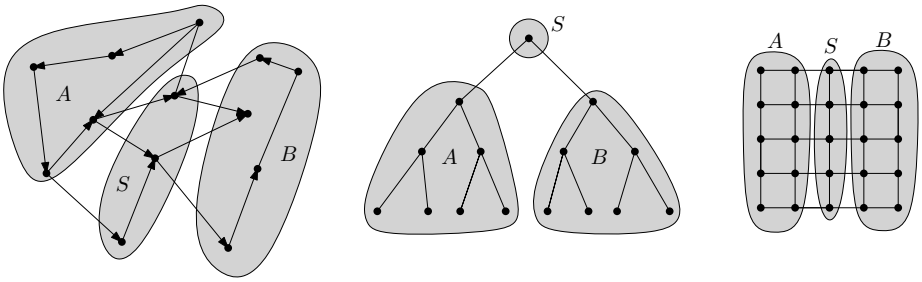

Figure 3: Illustrating *balanced splittors* for three non-crossing graphs.

(b) Argue why your algorithm is correct. [7 points]

(c) Prove an upper bound on the time complexity of your algorithm. [4 points]