

## Exercises 2 - Triangulation of Polygons & Linear Programming

**Discussion:** Wednesday, 11. November 2015

### Lecture 3 – 02.11.2015

**Exercise 1 – Monotone Polygons.** In the lecture we have seen the algorithm MAKEMONOTONE, which partitions a given polygon  $P$  into  $y$ -monotone polygons.

1. Prove the correctness of the sub-routine HANDLEMERGEVERTEX; see slides of lecture.
2. Assume that the polygon has  $O(1)$  turn vertices. Modify MAKEMONOTONE such that it partitions  $P$  into  $y$ -monotone polygons using  $O(n)$  running time – instead of  $O(n \log n)$  running time.
3. The sub-routines of MAKEMONOTONE add edges to a doubly-connected edge list. In the lecture it has been claimed that each insertion takes  $O(1)$  running time. Prove that this claim holds. Further, argue that the insertion of an edge into a doubly-connected edge list in general may need more than  $O(1)$  time.

**Exercise 2 – Art-Gallery.** Prove or falsify the following statement.

Let  $\mathcal{P}$  be a simple polygon and consider a set of cameras that together observe the complete border of  $P$ , then they also observe the complete interior of  $P$ .

**Exercise 3 – Polygon-Splitting.** Let  $P$  be a simple polygon with  $n$  vertices. Describe an algorithm that splits  $P$  into two simple polygons such that each of them has  $\lfloor 2n/3 \rfloor + 2$  vertices.

*Hint:* Triangulate the polygon and consider the dual graph of that triangulation.

### Lecture 4 – 09.11.2015

**Exercise 4 – Correctness.** In the lecture we have seen the algorithm RANDOMPERMUTATION( $A$ ).

1. Prove the correctness of RANDOMPERMUTATION by showing that all permutations of  $A$  have the same probability.
2. Show, that the permutations of  $A$  do not have the same probability, when the expression  $r \leftarrow \text{Random}(k)$  is replaced by  $r \leftarrow \text{Random}(n)$ .

**Exercise 5 – Trains.** We are given  $n$  trains that run on parallel tracks. Each train  $z_i$  ( $i = 1, \dots, n$ ) has constant speed  $v_i$  and starts at position  $k_i$  on its track at time  $t = 0$ . Describe an algorithm that computes in  $O(n \log n)$  time which trains are at least once in leading position until a given time  $t_{stop} > 0$ .