Question 1. Go through the points by decreasing x coordinates, maintaining as you go along the largest y coordinate encountered so far (call it $\hat{y}$). A point p_i is processed by comparing its y coordinate to $\hat{y}$: If $y_i < \hat{y}$ then that point is ignored (and you move on to the next point), but if $\hat{y} < y_i$ then you update $\hat{y}$ to be y_i and output p_i as being in $M(S)$. The $O(n \log n)$ time complexity is because of sorting the points according to their x coordinates (after sorting the rest of the algorithm takes linear time).

Question 2. $1, 5, 7, 10, 12$

Question 3.

1. g, c, a, b, d, h, e, f
2. (a) g: 3; c: 3; d: 2; h: 2
   (b) See the figure on the next page.
   (c) The total number of events ever inserted in the event list is $2n + t$, and similarly for the events deleted from that list. This, and the fact that a manipulation of the event list gives rise to at most two intersection discoveries, together imply that the total number of multiple discoveries is $O(n + t)$.

Question 4. The idea is to partition S into n/p contiguous chunks of size p each. Each of the windows of size p whose s_i we seek to compute either

1. coincides with one of the above-mentioned chunks,
2. overlaps with two adjacent such chunks.

Case 1 occurs for only n/p of the s_i we seek, and we can afford to spend $O(p)$ time on each. The main difficulty is Case 2, which occurs for $O(n)$ of the s_i we seek: We cannot afford to spend more than constant time on each. Note, however, that in Case 2 the overlap is with a suffix of the left chunk, and with a prefix of the second chunk. This observation suggests a pre-processing step in which we compute, for each of the n/p chunks, every prefix-max value (= maximum value in every prefix of that chunk) and every suffix-max value (= maximum value for every suffix of that chunk). This pre-processing takes $O(p)$ time per chunk, hence $O(n)$ total. It makes possible constant-time computation of every s_i: If the window for that s_i overlaps with chunks k and $k + 1$ then s_i is the larger of (i) the suffix-max value of chunk k corresponding to the amount of overlap with that chunk; and (ii) the prefix-max value of chunk $k + 1$ corresponding to the amount of overlap with that chunk. Because these two values are already available (from the pre-processing stage), each s_i is computed using one comparison.

If the above is not clear enough, below is a more formal description.

1. Partition S into n/p contiguous chunks of size p each (the last chunk could be smaller). We call S_k the kth such chunk, i.e., $S_k = x_{(k-1)p+1} x_{(k-1)p+2} \cdots x_{(k-1)p+p}$.

2. Do the following for $k = 1, \ldots, n/p$ in turn:

   (a) Compute in $O(p)$ time the quantities $L_{k,1}, \ldots, L_{k,p}$, where

   $$L_{k,i} = \max\{x_{(k-1)p+1}, \ldots, x_{(k-1)p+i}\}$$

   (i.e., $L_{k,i}$ is the maximum of the leftmost i items of chunk S_k). This can be done by a left to right walk along S_k that keeps track of the maximum encountered so far.

   (b) Compute in $O(p)$ time the quantities $R_{k,1}, \ldots, R_{k,p}$, where

   $$R_{k,i} = \max\{x_{(k-1)p+i}, \ldots, x_{(k-1)p+p}\}$$

   (i.e., $R_{k,i}$ is the maximum of the rightmost $p - i + 1$ items of chunk S_k). This can be done by a right to left walk along S_k that keeps track of the maximum encountered so far.

3. For $i = 1, 2, \ldots, n - p + 1$ compute s_i in constant time as follows:

   - Let $k = \lfloor i/p \rfloor$ and let $j = i \bmod p$ (i.e., $j = i - kp$).
   - If $j = 1$ then $s_i = R_{k,1}$.
   - If $1 < j \leq p - 1$ then $s_i = \max\{R_{k,j}, L_{k+1,j-1}\}$.
   - If $j = 0$ then $s_i = \max\{R_{k,p}, L_{k+1,p-1}\}$.

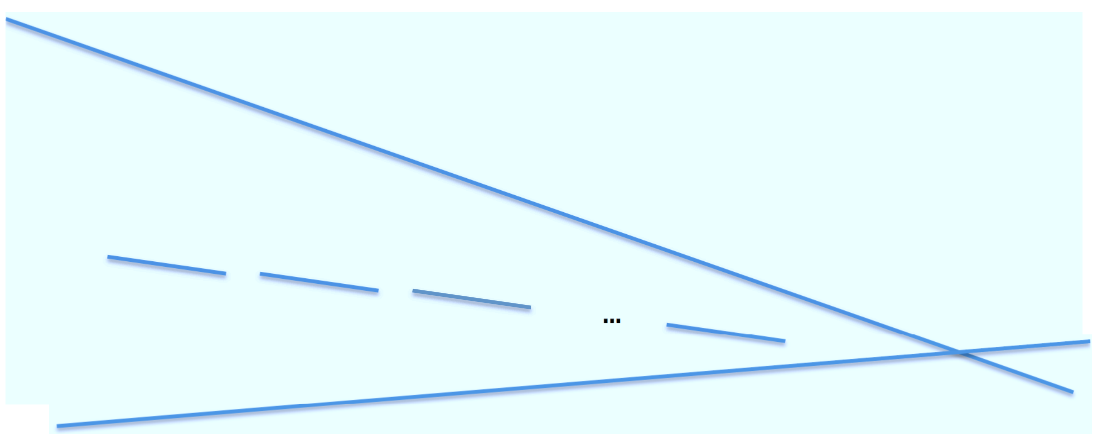

Figure 1: The answer to question 3.2.b.