Notation 1. *We write*

- $\mathbb{R}_+ = (0, \infty)$ *for the set of positive real numbers,*
- $\mathbb{R}_{\geq 0} = [0, \infty)$ *for the set of non-negative real numbers, and*
- $\mathbb{N}_+ = \{1, 2, \ldots\}$ *for the set of positive natural numbers.*

Let $n \in \mathbb{N}_+$ and let $(e_1, \ldots, e_n) \in \mathbb{R}^n_{\geq 0}$ be a finite sequence describing the elevation levels of a terrain. Assume that each e_i describes a segment with a width of 1 unit.

Assume that it is raining and that each segment receives 1 square unit of water per hour. The task is to find an algorithm that outputs the elevation of each segment after $T \in \mathbb{R}_+$ hours while taking into account the effects of gravity.

Example 1. *The following terrain could be described by the sequence* $10, 30, 50, 70, 90, 0, 70, 50$.

After 16 *hours it would then look as follows with the elevations being* $54, 54, 54, 70, 90, 36, 70, 70$.

Our main result is

Corollary 1. *There is an algorithm that computes the elevation levels in* $\mathcal{O}(n \log(n))$ *time.*

We will prove this at the very end of this paper.

In the following, we will no longer assume that each e_i describes a segment with a width of 1 unit. Instead, let $(w_1, \ldots, w_n) \in \mathbb{R}^n_+$ be a sequence describing the width of the segments. The original problem can then be solved by setting $w_i = 1$.

Definition 1. *Let* $n \in \mathbb{N}_+$. *We say that* $(e_1, \ldots, e_n)$ *is an elevation sequence with widths* $(w_1, \ldots, w_n)$ *if* $(e_1, \ldots, e_n) \in \mathbb{R}^n_{\geq 0}$ *and* $(w_1, \ldots, w_n) \in \mathbb{R}^n_+$.

In our generalized setting, each segment i receives w_i square units of water per hour.

Note that the problem is invariant under splitting of segments and merging of adjacent segments with identical elevation.

By splitting we mean replacing e_i in place by $e_{i,1} = e_{i,2} = e_i$ and w_i by $w_{i,1}, w_{i,2}$ with $w_{i,1} + w_{i,2} = w_i$.

By merging we mean replacing e_i, e_{i+1} with $e_i = e_{i+1}$ in place by $\hat{e}_i = e_i$ and w_i, w_{i+1} by $\hat{w}_i = w_i + w_{i+1}$.

Definition 2. *Let* $n \in \mathbb{N}_+$ *and let* $(e_1, \ldots, e_n)$ *be an elevation sequence.*

1. *We say that* $(e_1, \ldots, e_n)$ *is fully merged if* $e_i \neq e_{i+1}$ *for all* $i \in \{1, \ldots, n-1\}$.
2. *Let* $m \in \mathbb{N}_+$ *and let* $(\tilde{e}_1, \ldots, \tilde{e}_m)$ *be an elevation sequence. We say that* $(e_1, \ldots, e_n)$ *and* $(\tilde{e}_1, \ldots, \tilde{e}_m)$ *are equivalent if the fully merged versions of both sequences are identical.*
3. *We say that* $(e_1, \ldots, e_n)$ *has a single local minimum if there is an* $i \in \{1, \ldots, n\}$ *such that* $e_1, \ldots, e_i$ *is monotonically decreasing and* $e_i, \ldots, e_n$ *is monotonically increasing.*
4. *Assume that* $(e_1, \ldots, e_n)$ *is fully merged. Let* $i \in \{2, \ldots, n-1\}$. *We say that* e_i *is a local maximum if* $e_{i-1} < e_i$ *and* $e_i > e_{i+1}$.

Example 2. *The following terrain has a single local minimum:*

But this one has multiple local minima:

Lemma 1. *Let* $n \in \mathbb{N}_+$ *and let* $(e_1, \ldots, e_n)$ *be an elevation sequence. The fully merged version of* $(e_1, \ldots, e_n)$ *can be computed in* $\mathcal{O}(n)$ *time.*

Proof. Iterate over the e_i in ascending order. If $e_{i+1} = e_i$, merge the two elements. Otherwise move to the next element. □

Lemma 2. *Let* $n, m \in \mathbb{N}_+$, $m < n$, *and let* $(e_1, \ldots, e_n), (\hat{e}_1, \ldots, \hat{e}_m)$ *be equivalent elevation sequences. Let*

$$x \in \left[0, \sum_{i=1}^{n} w_i\right)$$

be a point in the terrain. The elevation level at x *can be determined from* $(e_1, \ldots, e_n)$, $(\hat{e}_1, \ldots, \hat{e}_m)$ *in* $\mathcal{O}(n)$ *time and this process yields the same value for both sequences.*

For $l \in \mathbb{N}_+$ *and* $x_1, \ldots, x_l$ *as above, the elevation levels at these points can be determined in* $\mathcal{O}(\max(l \log(l), n))$ *time.*

Proof. It is sufficient to assume that $(\hat{e}_1, \ldots, \hat{e}_m)$ is the fully merged version of $(e_1, \ldots, e_n)$.

We can determine in $\mathcal{O}(n)$ time the smallest j such that $\sum_{i=1}^{j} w_i > x$. Then the elevation level is e_j.

We can transform $(\hat{e}_1, \ldots, \hat{e}_m)$ into $(e_1, \ldots, e_n)$ by performing a finite number of splitting steps. Assume that it takes a single step such that $(e_1, \ldots, e_k, e_{k+1}, \ldots, e_n) = (\hat{e}_1, \ldots, \hat{e}_k, \hat{e}_k, \ldots, \hat{e}_m)$ for some $k \in 1, \ldots, m$.

If $r < k$ then $\sum_{i=1}^{r} \hat{w}_i = \sum_{i=1}^{r} w_i$. If $r \geq k$ then $\sum_{i=1}^{r} \hat{w}_i = \sum_{i=1}^{r+1} w_i$.

Let $\hat{j}$ be the smallest integer such that $\sum_{i=1}^{\hat{j}} \hat{w}_i > x$.

If $\hat{j} < k$ then $\sum_{i=1}^{\hat{j}} w_i = \sum_{i=1}^{\hat{j}} \hat{w}_i > x$, hence $k > \hat{j} \geq j$ by the definition of j. Furthermore $\sum_{i=1}^{j} \hat{w}_i = \sum_{i=1}^{j} w_i > x$, hence $j \geq \hat{j}$ by the definition of $\hat{j}$. Therefore $j = \hat{j} < k$ and $e_j = e_{\hat{j}} = \hat{e}_{\hat{j}}$.

If $\hat{j} = k$ then $\sum_{i=1}^{k-1} w_i = \sum_{i=1}^{k-1} \hat{w}_i \leq x$. Hence $j \geq k$ by the definition of j. Furthermore: $\sum_{i=1}^{k+1} w_i = \sum_{i=1}^{k} \hat{w}_i > x$. Hence $j \leq k+1$ by the definition of j. However, by our assumption $e_k = e_{k+1} = \hat{e}_k$. Therefore $e_j = \hat{e}_{\hat{j}}$.

If $\hat{j} > k$ then $\sum_{i=1}^{\hat{j}+1} w_i = \sum_{i=1}^{\hat{j}} \hat{w}_i > x$. Hence $j \leq \hat{j}+1$ by the definition of j. Furthermore $\sum_{i=1}^{\hat{j}} w_i = \sum_{i=1}^{\hat{j}-1} \hat{w}_i \leq x$ by the definition of $\hat{j}$. Hence $j > \hat{j}$ by the definition of j. Therefore $j = \hat{j}+1$ and $e_j = e_{\hat{j}+1} = \hat{e}_{\hat{j}}$.

The case where there is more than one splitting step now follows immediately by induction.

For the $x_1, \ldots, x_l$, we can first assume that they are sorted in ascending order by applying a sorting algorithm in $\mathcal{O}(l \log(l))$ time. Then we iterate over the x_i and apply the algorithm above except that, since the x_i are sorted, we do not have to restart the summation for each x_i. Hence, this iteration completes in $\mathcal{O}(\max(l, n))$ time. □

Lemma 3. *Let $n \in \mathbb{N}_+$ and let $(e_1, \ldots, e_n)$ be a fully merged elevation sequence. The following statements are equivalent:*

1. *$(e_1, \ldots, e_n)$ has a single local minimum.*
2. *$(e_1, \ldots, e_n)$ has no local maxima.*

Proof. $\Rightarrow$: Assume that $(e_1, \ldots, e_n)$ has a single local minimum e_i. For $i \leq j < k$, $e_j \leq e_k$, therefore e_j is not a local maximum. For $k < j \leq i$, $e_k \geq e_j$, therefore e_j is not a local maximum.

$\Leftarrow$: Assume that $(e_1, \ldots, e_n)$ has no local maxima. Let i be such that $e_i = \min(e_1, \ldots, e_n)$. We show by induction over $j \in \{i, \ldots, n-1\}$ that $e_j < e_{j+1}$. For $j = i$ it follows by the definition of i and the fact that the sequence is fully merged. Assume that the statement holds for $j-1$, that is, $e_{j-1} < e_j$. If $e_j > e_{j+1}$, then e_j would be a local maximum. Since this is not possible, $e_j < e_{j+1}$. By an identical induction, we see that $e_{j-1} > e_j$ for $j \in \{2, \ldots, i\}$. □

Next we will describe an algorithm that solves the problem for a terrain with a single local minimum. We will call this algorithm the *SLM algorithm.*

Lemma 4 (SLM Algorithm)**.** *Let $n \in \mathbb{N}_+$, $V \in \mathbb{R}_{\geq 0}$, and let $(e_1, \ldots, e_n)$ be a fully merged elevation sequence with a single local minimum. The following algorithm calculates the elevation levels after pouring V square units of water into the terrain. The algorithm runs in $\mathcal{O}(n)$ time.*

Let i be such that e_i is the local minimum of the sequence
For all j, $e_j^0 := e_j$ and $w_j^0 := w_j$
$i^0 := i$
$n^0 := n$
$K := 0$
while true **do**
    $l := \infty$
    if $i^K > 1$ **then**

9: $l := e^K_{i^K-1}$
10: **end if**
11: $r := \infty$
12: **if** $i^K < n^K$ **then**
13: $r := e^K_{i^K+1}$
14: **end if**
15: $m := \min(l, r)$
16: $u := (m - e^K_{i^K})w_{i^K}$
17: **if** $V < u$ **then**
18: $e^K_{i^K} := e^K_{i^K} + \frac{V}{w_{i^K}}$
19: break
20: **end if**
21: $e^K_{i^K} := m$
22: **if** $l = r$ **then**
23: $n^{K+1} := n^K - 2$
24: **else**
25: $n^{K+1} := n^K - 1$
26: **end if**
27: Merge $e^K_{i^K}$ with the surrounding segments if possible
28: Call the resulting sequences $e^{K+1}_1, \ldots, e^{K+1}_{n^{K+1}}$ and $w^{K+1}_1, \ldots, w^{K+1}_{n^{K+1}}$
29: **if** $l \leq r$ **then**
30: $i^{K+1} := i^K - 1$
31: **else**
32: $i^{K+1} := i^K$
33: **end if**
34: $K := K + 1$
35: $V := V - u$
36: **end while**
37: The $e^K_1, \ldots, e^K_{n^K}$ are the elevation levels

Proof. The proof consists of 4 parts. In part 1 we show that the algorithm is well-formed. In part 2 we show that it can be expressed recursively. In part 3 we show that it computes the elevation levels. In part 4 we show that it runs in $\mathcal{O}(n)$ time.

Part 1: By induction over K we show that $i^K \leq n^K$ and that n^{K+1} is in fact the length of the sequence after the merge step in line 27. Note that we only show it for those finitely many K that appear in the algorithm.

For $K = 0$ it is clear. Assume that the condition holds for $K = \tilde{K}$ and that we have started the loop with K set to this value. If $V < u$ holds in line 17, then the loop ends and it is nothing to show. Otherwise assume that $V \geq u$.

If $l = r$ in line 22, then $(l - e^K_{i^K})w_{i^K} = (\min(l, r) - e^K_{i^K})w_{i^K} = (m - e^K_{i^K})w_{i^K} = u \leq V < \infty$. Therefore, since $w_{i^K} > 0$, $l = r < \infty$. By the definitions of l and r, it follows that $i^K > 1$ and $i^K < n^K$ and therefore $e^K_{i^K-1} = l = r = e^K_{i^K+1}$. After setting $e^K_{i^K} := m = l = r$ in line 21, the merge step in line 27 merges $e^K_{i^K-1}, e^K_{i^K}$, and $e^K_{i^K+1}$. Therefore the resulting sequence contains two fewer elements and the assignment $n^{K+1} := n^K - 2$ in line 23 was correct. Furthermore, in line 30, $i^{K+1} := i^K - 1 \leq n^K - 1 - 1 = n^{K+1}$.

Otherwise if $l < r$ then $l < \infty$ and $i^K > 1$ and therefore $e^K_{i^K-1} = l$. After setting $e^K_{i^K} := m = l$ in

line 21, the merge step in line 27 merges $e^K_{i^K-1}$ and $e^K_{i^K}$. Therefore the resulting sequence contains one fewer element and the assignment $n^{K+1} := n^K - 1$ in line 25 was correct. Furthermore, in line 30, $i^{K+1} := i^K - 1 \leq n^K - 1 = n^{K+1}$.

Otherwise if $l > r$ then $r < \infty$ and $i^K < n^K$ and therefore $e^K_{i^K+1} = r$. After setting $e^K_{i^K} := m = r$ in line 21, the merge step in line 27 merges $e^K_{i^K}$ and $e^K_{i^K+1}$. Therefore the resulting sequence contains one fewer element and the assignment $n^{K+1} := n^K - 1$ in line 25 was correct. Furthermore, in line 32, $i^{K+1} := i^K \leq n^K - 1 = n^{K+1}$.

This completes the proof by induction.

Part 2: By induction over K, we show that the sequence $e^K_1, \ldots, e^K_{n^K}$ is fully merged and has a single local minimum, and that $e^K_{i^K}$ is the local minimum of the sequence.

For $K = 0$ it is clear. Assume that the condition holds for $K = \tilde{K}$ and that we have started the loop with K set to this value. If $V < u$ holds in line 17, then $e^K_{i^K} := e^K_{i^K} + \frac{V}{w_{i^K}} < e^K_{i^K} + \frac{u}{w_{i^K}} = m = \min(l, r)$. Hence, $e^K_{i^K} < \min(l, r)$ and $e^K_{i^K}$ remains the strict minimum of the sequence. Otherwise assume that $V \geq u$.

If $l \leq r$ in line 29, then $e^K_{i^K} := m = l$ in line 21 and the merge step in line 27 merges $e^K_{i^K-1}$ and $e^K_{i^K}$. Since we set $i^{K+1} := i^K - 1$ in line 30, $(e^{K+1}_1, \ldots, e^{K+1}_{i^{K+1}}) = (e^K_1, \ldots, e^K_{i^K-1})$ is strictly monotonically decreasing by induction. If $l < r$, then $(e^{K+1}_{i^{K+1}}, \ldots, e^{K+1}_{n^{K+1}}) = (l, r, \ldots, e^K_{n^K})$ is strictly monotonically increasing. Otherwise if $l = r$, then $(e^{K+1}_{i^{K+1}}, \ldots, e^{K+1}_{n^{K+1}}) = (r, e^K_{i^K+2}, \ldots, e^K_{n^K})$ is strictly monotonically increasing. (Note that in the case $l = r$ we merge 3 segments.)

If $l > r$ in line 29, then $e^K_{i^K} := m = r$ in line 21 and the merge step in line 27 merges $e^K_{i^K}$ and $e^K_{i^K+1}$. Since we set $i^{K+1} := i^K$ in line 30, $(e^{K+1}_1, \ldots, e^{K+1}_{i^{K+1}}) = (e^K_1, \ldots, l, r)$ is strictly monotonically decreasing by induction. Furthermore $(e^{K+1}_{i^{K+1}}, \ldots, e^{K+1}_{n^{K+1}}) = (r, \ldots, e^K_{n^K})$ is strictly monotonically increasing.

This completes the proof by induction.

Note that the algorithm is tail recursive. Therefore we can reformulate it as a recursive function. Instead of jumping back to the start of the loop, we can invoke the algorithm again with the sequence set to $e^{K+1}_1, \ldots, e^{K+1}_{n^{K+1}}$ and V set to $V - u$. Recall that we have just proved that this sequence is a valid input for the algorithm and that $e^{K+1}_{i^{K+1}}$ is the minimum that is being computed in line 1.

Part 3: We can now prove by induction over n that the algorithm computes the elevation levels.

Step 1: $n = 1$ or $V < u$ during the first iteration of the loop. Note that $n = 1$ implies $V < \infty = u$.

Since the e_i is the single local minimum of the sequence and since it is a strict minimum, all water will flow to e_i until the negative volume below the adjacent segments is used up. The following image shows the negative volume of an example.

Note that $u := (\min(l, r) - e^K_{i_K})w_i$ is the volume of this space. By our assumption, $V < u$, that is, all of the water fits into this space. In line 18, the algorithm increases the height of e_i by V/w_i which is in fact the height a volume V would take up in this space of width w_i.

Step 2: Assume that the argument holds for all $\tilde{n} < n$. For $V < u$ we have already shown the result in step 1. Otherwise assume $V \geq u$. Let u be as in the first iteration of the loop, $V_1 = u$, and $V_2 = V - V_1 \geq 0$. We have shown above that invoking the algorithm with V is the same as running the iteration once and then invoking the algorithm on the resulting sequence with $V = V_2$. Note that invoking the algorithm with V_1 runs the iteration once and then once more with $V = 0$. Since the sequence is fully merged, we know that $u > 0$; therefore we break out of the loop in the second iteration. However, the second iteration has no effect on the output of the algorithm. Therefore, invoking the algorithm with V is identical to invoking the algorithm once with $V = V_1$ and then with $V = V_2$ on the output of the first invocation. During the first iteration of the invocation with $V = V_1$, we have $V = u \geq u$ in line 17. Therefore the algorithm outputs a new sequence with length $n^1 < n$.

Note that during the invocation of the algorithm with $V = V_1$ we have $V = u$. By an argument identical to that in step 1, this invocation computes the elevation level after pouring V_1 square units of water into the terrain. By our induction assumption (since $n^1 < n$), invoking the algorithm with $V = V_2$ on the output of the first invocation calculates the elevation levels after pouring V_2 square units of water into the terrain, taking into account the changes in elevation caused by the first V_1 square units of water.

Since water in the real world also composes, that is, pouring V square units is the same as pouring V_1 square units and then V_2 square units (ignoring evaporation etc.), this completes the proof by induction.

Part 4: Lastly we show that the algorithm completes in $\mathcal{O}(n)$ time. Computing i in line 1 takes $\mathcal{O}(n)$ time. Within the loop, performing the merge only touches the elements to the immediate left and right of the local minimum; therefore, assuming an implementation via linked lists, this operation takes $\mathcal{O}(1)$ time. All other operations within the loop are clearly $\mathcal{O}(1)$. Each iteration of the loop decreases the length of the sequence by at least 1 and the loop terminates when the length is 1. Therefore the loop iterates at most n times.

This completes the proof. □

Lemma 5. *Applying the algorithm from the previous lemma with $V = V_1 + V_2$ is the same as applying it once with $V = V_1$ and then applying it with $V = V_2$ on the output of the first application.*

Proof. Since the algorithm computes the effects of a physical process and since this physical process decomposes, the algorithm also decomposes. □

We can now solve the general problem using the following ansatz.

Let $(e_1, \ldots, e_n)$ be a fully merged elevation sequence.

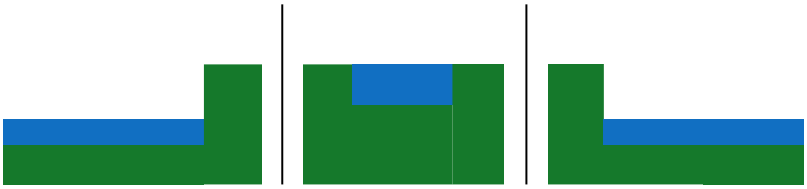
Note that on each local maximum, half of the water flows to the left and half flows to the right. Split each local maximum in half.

We call each of these three components a *sink*. Note that each sink has a single local minimum. Fill each sink uniformly and independently until any of the sinks overflows onto one of the bounding local maxima.

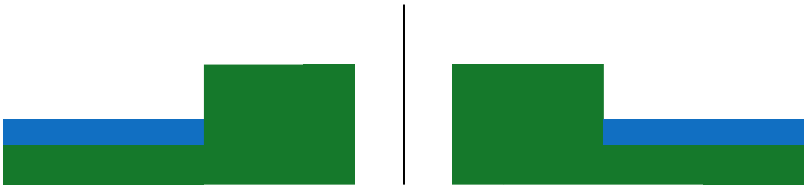
Fill up the overflowing sink and merge it with the sink on the overflowing side.

Note that the merged sink has a single local minimum. Repeat this process until T is reached. Note that if only a single sink remains, that sink will never overflow since it has no bounding local maxima.

One special case occurs if the sink overflows on both sides at the same time.



In this case the overflowing sink is filled up, split in half, and divided equally between its neighbors.



We will now define the sinks of an elevation sequence and their properties.

Let $(e_1, \ldots, e_n)$ be a fully merged elevation sequence and let $k_1, \ldots, k_m \in \{2, \ldots, n-1\}$ be its local maxima.

If the sequence has no local maxima, then it has a single sink S_1 containing all of the segments.

Otherwise, split each e_{k_i} into $e_{k_i,1}$ and $e_{k_i,2}$, each having half of the width of e_{k_i}. Then S_1 consists of $(e_1, \ldots, e_{k_1,1})$. For $i \in \{2, \ldots, m\}$, S_i consists of $(e_{k_{i-1},2}, \ldots, e_{k_i,1})$. S_{m+1} consists of $(e_{k_m,2}, \ldots, e_n)$. Consequently there are $m+1$ sinks.

For each sink S we define

$$\mathcal{B}_l(S) = \begin{cases} \text{the elevation of the first segment in } S, & \text{if there is a sink to the left of } S \\ \infty, & \text{otherwise} \end{cases}$$

$$\mathcal{B}_r(S) = \begin{cases} \text{the elevation of the last segment in } S, & \text{if there is a sink to the right of } S \\ \infty, & \text{otherwise} \end{cases}$$

The *bounding height* $\mathcal{B}(S)$ of S is defined as $\mathcal{B}(S) = \min(\mathcal{B}_l(S), \mathcal{B}_r(S))$.

We write $\mathcal{E}(S)$ for the sequence of segments in S. For example, $\mathcal{E}(S_i) = (e_{k_{i-1}^2}, \ldots, e_{k_i^1})$ for $i \in \{2, \ldots, m\}$. We write $\mathcal{C}(S)$ for the subsequence of $\mathcal{E}(S)$ consisting of those e for which $e \leq \mathcal{B}(S)$. We write $\|\mathcal{E}(S)\|$ for the number of elements in $\mathcal{E}(S)$ and $\|\mathcal{C}(S)\|$ for the number of elements in $\mathcal{C}(S)$.

It is clear that, since $\mathcal{E}(S)$ contains no local maximum, $\mathcal{C}(S)$ is a continuous subsequence of $\mathcal{E}(S)$. That is, if e_a, e_b, and e_c are in $\mathcal{E}(S)$ in that order, and e_a and e_c are in $\mathcal{C}(S)$, then $e_b \in \mathcal{C}(S)$. Let e_l and e_r be the first and last elements in $\mathcal{E}(S)$. By the definition of $\mathcal{B}(S)$, $e_l \leq \mathcal{B}(S)$ or $e_r \leq \mathcal{B}(S)$. Therefore $e_l \in \mathcal{C}(S)$ or $e_r \in \mathcal{C}(S)$.

When discussing the time complexity of our algorithm, we will assume that $\mathcal{E}(S)$ is implemented by storing two pointers. The first pointer points to the first element in $\mathcal{E}(S)$ and the second pointer points to the last element of $\mathcal{E}(S)$. Similarly for $\mathcal{C}(S)$.

For each segment $e \in \mathcal{E}(S)$, let $\mathcal{W}(e)$ be the corresponding width. We define the *capacity* $\text{cap}(S)$ of a sink as follows:

$$\text{cap}(S) = \sum_{e \in \mathcal{C}(S)} \mathcal{W}(e)(\mathcal{B}(S) - e).$$

Note that $\text{cap}(S)$ is the volume of water S can hold before it overflows.

We define the *width* $\mathcal{W}(S)$ of S by

$$\mathcal{W}(S) = \sum_{e \in \mathcal{E}(S)} \mathcal{W}(e).$$

We define the *capacity width* $\mathcal{W}_c(S)$ of S by

$$\mathcal{W}_c(S) = \sum_{e \in \mathcal{C}(S)} \mathcal{W}(e).$$

At certain points in the algorithm, we compute for a sink S how much water is currently in the sink. At these points we store the time since it has started raining and the computed value. We call these values $\text{filltime}(S)$ and $\text{fill}(S)$ respectively. Initially, both values are 0.

For each sink S, we calculate the time (in hours) at which it will overflow, $\text{ot}(S)$, as follows:

$$\text{ot}(S) = \frac{\text{cap}(S) - \text{fill}(S)}{\mathcal{W}(S)} + \text{filltime}(S).$$

If a sink has capacity $\text{cap}(S)$, of which $\text{fill}(S)$ is already used, then the remaining capacity for additional water is $\text{cap}(S) - \text{fill}(S)$. Since the sink receives $\mathcal{W}(S)$ square units of water per hour, this additional capacity is used up after $\frac{\text{cap}(S)-\text{fill}(S)}{\mathcal{W}(S)}$ hours. Since this additional rain occurs after the time $\text{filltime}(S)$, we have to add $\text{filltime}(S)$ to get the absolute time value. We will prove this more formally below.

We define a total order on the sinks S_i by the lexicographic order on the tuples $(\text{ot}(S_i), i)$. That is, sinks that overflow earlier come first. If two sinks overflow at the same time, the sink with the smaller index comes first.

In our algorithm, we will maintain a priority queue containing all sinks. Popping an element from this queue always returns the sink that comes first in the order defined above. That is, it always returns the sink that overflows next.

Some of the properties above require more than $\mathcal{O}(1)$ time to be computed. In these cases, we will assume that the properties are calculated once and then cached. When merging two sinks we will re-calculate the cached values.

In our algorithm, we will maintain doubly-linked lists of elevation segments and sinks. Both of these lists contain their elements from left to right in the terrain. When we say that we remove one or more elevation segments or sinks, we mean removing them from these linked lists.

Theorem 1. *Let $n \in \mathbb{N}_+$, $T \in \mathbb{R}_{\geq 0}$, and let $(e_1, \ldots, e_n)$ be a fully merged elevation sequence. The following algorithm calculates the elevation levels at time T in $\mathcal{O}(n \log(n))$ time.*

Let $(S_i)_i$ be the sinks of $(e_1, \ldots, e_n)$
Let B be a priority queue containing the sinks as described above
while true **do**
    Pop a sink S from B
    if $\text{ot}(S) \geq T$ **then**
        break
    end if
    if $\mathcal{B}_l(S) = \mathcal{B}_r(S)$ **then**
        Let S_l be the sink to the left of S
        Let S_r be the sink to the right of S
        $\text{fill}(S_l) := \text{fill}(S_l) + (\text{ot}(S) - \text{filltime}(S_l))\mathcal{W}(S_l)$
        $\text{fill}(S_r) := \text{fill}(S_r) + (\text{ot}(S) - \text{filltime}(S_r))\mathcal{W}(S_r)$
        $\text{filltime}(S_l) := \text{ot}(S)$
        $\text{filltime}(S_r) := \text{ot}(S)$
        Let e_r be the rightmost segment in S_l
        Let e_l be the leftmost segment in S_r
        Remove all segments between e_r and e_l
        $w := \mathcal{W}(e_r) + \mathcal{W}(e_l) + \mathcal{W}(S)$
        $d_r := w/2 - \mathcal{W}(e_r)$
        $d_l := w/2 - \mathcal{W}(e_l)$
        $\mathcal{W}(e_r) := \mathcal{W}(e_r) + d_r$
        $\mathcal{W}(e_l) := \mathcal{W}(e_l) + d_l$
        $\mathcal{W}(S_l) := \mathcal{W}(S_l) + d_r$
        $\mathcal{W}(S_r) := \mathcal{W}(S_r) + d_l$
        if $\mathcal{C}(S_l)$ ends in e_r **then**
            $\mathcal{W}_c(S_l) := \mathcal{W}_c(S_l) + d_r$

```
        end if
        if C(S_r) ends in e_l then
            W_c(S_r) := W_c(S_r) + d_l
        end if
        Remove S.
        Recalculate ot(S_l)
        Recalculate ot(S_r)
    else
        if B_l(S) < B_r(S) then
            Let S_m be the sink to the left of S
        else
            Let S_m be the sink to the right of S
        end if
        fill(S_m) := fill(S_m) + (ot(S) − filltime(S_m))W(S_m)
        filltime(S_m) := ot(S)
        Remove the elements of C(S)
        Extend E(S_m) to contain E(S)
        W(S_m) := W(S_m) + W(S)
        if B_l(S) < B_r(S) then
            Let e_r be the rightmost segment in S_m
            W(e_r) := W(e_r) + W_c(S)
            if C(S_m) ends in e_r then
                W_c(S_m) := W_c(S_m) + W_c(S)
            end if
            B_r(S_m) := B_r(S)
        else
            Let e_l be the leftmost segment in S_m
            W(e_l) := W(e_l) + W_c(S)
            if C(S_m) ends in e_l then
                W_c(S_m) := W_c(S_m) + W_c(S)
            end if
            B_l(S_m) := B_l(S)
        end if
        b := B(S_m)
        B(S_m) := min(B_l(S_m), B_r(S_m))
        cap(S_m) := cap(S_m) + W_c(S_m)(B(S_m) − b)
        while there is a segment to the left of C(S_m) do
            Let e be the segment immediately to the left of C(S_m)
            if e > B(S_m) or e is outside of E(S_m) then
                break
            end if
            W_c(S_m) := W_c(S_m) + W(e)
            cap(S_m) := cap(S_m) + W(e)(B(S_m) − e)
            Extend C(S_m) to contain e
        end while
        while there is a segment to the right of C(S_m) do
            Let e be the segment immediately to the right of C(S_m)
            if e > B(S_m) or e is outside of E(S_m) then
                break
```

76: **end if**
77: $\mathcal{W}_c(S_m) := \mathcal{W}_c(S_m) + \mathcal{W}(e)$
78: $\text{cap}(S_m) := \text{cap}(S_m) + \mathcal{W}(e)(\mathcal{B}(S_m) - e)$
79: Extend $\mathcal{C}(S_m)$ to contain e
80: **end while**
81: Remove S.
82: Recalculate $\text{ot}(S_m)$
83: **end if**
84: **end while**
85: **for** each sink S in the linked list **do**
86: $\text{fill}(S) := \text{fill}(S) + (T - \text{filltime}(S))\mathcal{W}(S)$
87: Detach $\mathcal{E}(S)$ from the rest of the linked list
88: Apply the SLM algorithm to $\mathcal{E}(S)$ with $V = \text{fill}(S)$
89: Insert the output into the linked list in the place where we just detached $\mathcal{E}(S)$
90: **end for**
91: The linked list of segments are the desired elevation levels.

Proof. The proof consists of three parts. In the first part we will show that the algorithm is well-formed and various useful properties. In the second part we will show that the algorithm computes the correct values. In the third part we will show that it runs in $\mathcal{O}(n \log(n))$ time.

Part 1: We show by induction that the following properties hold for all sinks S in the linked list of sinks whenever we reach line 4.

I The definitions of $\mathcal{B}_l(S)$, $\mathcal{B}_r(S)$, $\mathcal{B}(S)$, $\mathcal{C}(S)$, $\text{cap}(S)$, $\mathcal{W}(S)$, $\mathcal{W}_c(S)$, and $\text{ot}(S)$ hold.

II The $\mathcal{E}(S)$ form a partition of the terrain.

III $\text{filltime}(S) \leq T$ and $\text{filltime}(S) \leq \text{ot}(S_*)$ for all S_* in the linked list.

IV If two sinks in the linked list are next to each other, then the segments at their common boundary have the same elevation and width.

V If the linked list contains more than one sink, then each sink contains at least two segments.

VI The $\mathcal{C}(S)$ are non-empty.

VII $\mathcal{W}(S)$, $\mathcal{W}_c(S)$ are positive and never decrease.

VIII $\mathcal{W}(e)$ is positive for all segments.

IX $\mathcal{B}_l(S)$ and $\mathcal{B}_r(S)$ are positive and never decrease.

X $\text{cap}(S)$ is positive and never decreases.

XI $\mathcal{E}(S)$, when detached from the rest of the linked list of segments, has a single local minimum and is fully merged.

XII The priority queue B contains exactly the sinks in the linked list of sinks and is not empty.

XIII The sequence of values $\text{ot}(S)$ of the sinks S popped in line 4 never decreases.

XIV $\text{fill}(S) \leq \text{cap}(S)$.

XV $\sum_S (\|\mathcal{E}(S)\| - \|\mathcal{C}(S)\|)$ (where S ranges over all S in the linked list) does not increase during this iteration and decreases by 1 for each time to algorithm reaches line 70 or 79.

XVI Applying the SLM algorithm to $\mathcal{E}(S)$ detached from all other sinks with $V = \text{fill}(S)$ computes the elevation levels of the sink at time filltime(S).

Throughout the proof, we will use the prime symbol $'$ to refer to the values of properties as they were at the start of the iteration before any modifications. For example, $\mathcal{W}(S')$.

Note: During the proof we make gratuitous use of the following physical property: Due to **IV** and **XI**, water that falls onto a sink always flows towards the sink and never outside the sink.

Induction start: Assume that we have reached line 4 for the first time. Then none of the S or their properties have been modified after construction. Therefore **I** holds by definition.

Since filltime(S) $= 0$, filltime(S) $\leq T$. Since

$$\text{ot}(S_*) = \frac{\text{cap}(S_*) - \text{fill}(S_*)}{\mathcal{W}(S_*)} + \text{filltime}(S_*) = \frac{\text{cap}(S_*)}{\mathcal{W}(S_*)} \geq 0 = \text{filltime}(S),$$

III holds.

II and **IV** hold by the construction of the sinks. **VI** holds because $\mathcal{C}(S)$ contains one of the end-segments of the sink. **VIII** holds by our definition of elevation sequence.

To show **V**, note that if there is only 1 sink, there is nothing to do. Otherwise each sink is bounded on at least one side by a local maximum of the original elevation sequence. Note that two adjacent elevation segments cannot both be local maxima because one has to be higher than the other. Therefore, if e_l and e_r are segments next to each other and e_r is a local maximum, then the sink to the left of e_r contains e_l and the left half of the split version of e_r. Therefore **V** holds.

$\mathcal{W}(S) > 0$ holds because each sink contains at least one segment and because of **VIII**. $\mathcal{W}_c(S) > 0$ holds because of **VI** and **VIII**. Therefore **VII** holds.

To show **IX**, note that if S is the leftmost sink, $\mathcal{B}_l(S)$ is ∞. Otherwise it is the elevation level of the leftmost segment in S. Since this segment was created by splitting a local maximum of the original elevation segment, and since local maxima are positive by definition, $\mathcal{B}_l(S)$ is positive. Similarly for $\mathcal{B}_r(S)$. Therefore **IX** holds.

To show **X**, recall that $\text{cap}(S) = \sum_{e \in \mathcal{C}(S)} \mathcal{W}(e)(\mathcal{B}(S) - e)$. Since $\mathcal{C}(S)$ is not empty by **VI** and $\mathcal{W}(e) > 0$ by **VIII**, we only have to show that $\mathcal{B}(S) > e$ for some $e \in \mathcal{C}(S)$. If $\mathcal{B}(S) = \infty$, there is nothing to show. Otherwise, if $\mathcal{B}(S) = \min(\mathcal{B}_l(S), \mathcal{B}_r(S)) = \mathcal{B}_l(S)$, the leftmost segment e_l in S was created by splitting a local maximum of the original elevation sequence. $\mathcal{B}_l(S)$ is the elevation of this segment. The segment to the right of e_l, e_*, was not a local maximum of the original sequence and $e_* < e_l = \mathcal{B}(S)$. Therefore e_* is part of $\mathcal{C}(S)$. Similarly for $\mathcal{B}(S) = \mathcal{B}_r(S)$. Therefore **X** holds.

To show **XI**, note that $\mathcal{E}(S)$ does not contain a local maximum by the construction of the sinks. By lemma 3, this is equivalent to $\mathcal{E}(S)$ having a single local minimum. Since the original sequence was fully merged and since $\mathcal{E}(S)$ contains split segments only at its boundary, $\mathcal{E}(S)$ is fully merged. Therefore **XI** holds.

By our construction of B it contains all sinks and by our construction of the sinks, there is at least 1 sink. Therefore **XII** holds.

There is nothing to show for **XIII** and **XV**.

Since $\text{fill}(S) = 0 \leq \text{cap}(S)$, **XIV** holds.

Since $\text{fill}(S) = \text{filltime}(S) = 0$, **XVI** holds.

Induction step: Assume that we have reached line 4 and that the properties hold. We show that the properties continue to hold at the start of the next iteration. Note that, because of **XII**, popping a sink from B is a well-defined operation.

If $\text{ot}(S) \geq T$ in line 5, then we leave the loop and there is nothing to show. Assume $\text{ot}(S) < T$.

Case 1: $\mathcal{B}_l(S) = \mathcal{B}_r(S)$ in line 8.

Since

$$\begin{aligned}\text{ot}(S) &= \frac{\text{cap}(S) - \text{fill}(S)}{\mathcal{W}(S)} + \text{filltime}(S) \\ &= \frac{\sum_{e \in \mathcal{C}(S)} \mathcal{W}(e)(\mathcal{B}(S) - e) - \text{fill}(S)}{\mathcal{W}(S)} + \text{filltime}(S) \\ &< T \\ &< \infty,\end{aligned}$$

by **I**, $\mathcal{C}(S)$ is not empty by **VI**, and $\mathcal{W}(e) > 0$ by **VIII**, we see that $\mathcal{B}_l(S) = \mathcal{B}_r(S) = \min(\mathcal{B}_l(S), \mathcal{B}_r(S)) = \mathcal{B}(S) < \infty$.

Therefore, by **I**, S has neighbors to its left and right and the operations in lines 9 and 10 are well-defined.

Let e_r and e_l be as in lines 15 and 16. By **IV** and **V**, we know that $\mathcal{W}(S) \geq \mathcal{W}(e_l) + \mathcal{W}(e_r)$. Therefore $w/2 \geq \mathcal{W}(e_l), \mathcal{W}(e_r)$ holds for w as in line 18. Therefore $d_r, d_l \geq 0$ in lines 19, 20.

Note that $d_r + d_l = \mathcal{W}(S)$. In line 17 we remove the segments between e_r and e_l but in lines 21 and 22 we distribute the removed width to e_r and e_l. Therefore the total width of the linked list of segments remains the same. Similarly, in line 31, we remove S from the linked list of sinks but in lines 23 and 24 we distribute the removed width to S_l and S_r. Therefore the total width of the linked list of sinks remains the same. Therefore **II** holds. Since we add d_r to both $\mathcal{W}(e_r)$ and $\mathcal{W}(S_l)$, the definition of $\mathcal{W}(S_l)$ continues to hold. Similarly for $\mathcal{W}(S_r)$.

Note that after lines 21 and 22, e_r and e_l have the same length. Because of **IV**, $e_r = \mathcal{B}_l(S) = \mathcal{B}_r(S) = e_l$. Therefore **IV** holds after line 31.

Only the right boundary of S_l and the left boundary of S_r are affected by these modifications. Since S_l and S_r are neighbors after line 31 and since the elevations of e_r and e_l are unchanged, $\mathcal{B}_l(S_l)$, $\mathcal{B}_r(S_l)$, and $\mathcal{B}(S_l)$ remain accurate. Similarly for S_r. Therefore **IX** holds.

Note that $\mathcal{E}(S_l)$ is unchanged by our operations and that $\mathcal{C}(S_l)$, being defined only in terms of $\mathcal{E}(S_l)$ and $\mathcal{B}(S_l)$, remains accurate. If $\mathcal{C}(S_l)$ contains e_r, then $\mathcal{W}_c(S_l)$ becomes accurate after line 26. Similarly for S_r. Therefore **VI** holds.

Since $d_r, d_l \geq 0$, **VII** and **VIII** hold.

If $\mathcal{C}(S_l)$ does not contain e_r, then it is clear that $\text{cap}(S_l)$ is unaffected by the change in $\mathcal{W}(e_r)$. Otherwise, note that $\mathcal{B}(S_l) \leq \mathcal{B}_r(S_l) = e_r \leq \mathcal{B}(S_l)$. Therefore $\mathcal{B}(S_l) = e_r$ and $\text{cap}(S_l)$ is unaffected by the change in $\mathcal{W}(e_r)$. Therefore the definition of $\text{cap}(S_l)$ continues to hold. Similarly for S_r. Hence and **X** holds.

Since we recalculate $\text{ot}(S_l)$ and $\text{ot}(S_r)$ in lines 32 and 33, they are accurate after the iteration.

We have thus shown that **I** holds.

Since S was popped from B and because of **XII**, S comes before S_m in the total order of the sinks, that is $\text{ot}(S) \leq \text{ot}(S'_l)$. Therefore

$$\begin{aligned}
\text{fill}(S_l) &= \text{fill}(S'_l) + (\text{ot}(S) - \text{filltime}(S'_l))\mathcal{W}(S'_l) \\
&\leq \text{fill}(S'_l) + (\text{ot}(S'_l) - \text{filltime}(S'_l))\mathcal{W}(S'_l) \\
&= \text{fill}(S'_l) + \left(\frac{\text{cap}(S'_l) - \text{fill}(S'_l)}{\mathcal{W}(S'_l)}\right)\mathcal{W}(S'_l) \\
&= \text{cap}(S'_l) \\
&= \text{cap}(S_l)
\end{aligned}$$

Similarly for S_r. Therefore **XIV** holds.

We set $\text{filltime}(S_l) = \text{ot}(S) < T$ in line 13. Note that

$$\begin{aligned}
\text{ot}(S_l) &= \frac{\text{cap}(S_l) - \text{fill}(S_l)}{\mathcal{W}(S_l)} + \text{filltime}(S_l) \\
&\geq \text{filltime}(S_l) \\
&= \text{ot}(S).
\end{aligned}$$

and $\text{ot}(S)$ is the minimum of all $\text{ot}(S_*)$. Similarly for S_r. Therefore **III** and **XIII** hold.

Since we do not change the number of segments in S_l and S_r, **V** holds. Since we have not changed the elevations of any of the segments in S_l and S_r, **XI** holds.

Since we have removed exactly S from B and since we have removed exactly S from the linked list in line 31, **XII** holds.

Since we removed S from the linked list and did not modify the number of elements in $\mathcal{E}(S_*)$ or $\mathcal{C}(S_*)$ for any other S_*, **XV** holds.

For each S_* in the list, between $\text{filltime}(S'_*)$ and $\text{ot}(S)$,

$$\begin{aligned}
(\text{ot}(S) - \text{filltime}(S'_*))\mathcal{W}(S'_*) &= (\text{ot}(S) - \text{ot}(S'_*))\mathcal{W}(S'_*) + (\text{ot}(S'_*) - \text{filltime}(S'_*))\mathcal{W}(S'_*) \\
&= (\text{ot}(S) - \text{ot}(S'_*))\mathcal{W}(S'_*) + \text{cap}(S'_*) - \text{fill}(S'_*)
\end{aligned}$$

square units of water fall directly on top of S_*. Recall that

$$\text{cap}(S'_*) = \sum_{e \in \mathcal{C}(S'_*)} \mathcal{W}(e)(\mathcal{B}(S'_*) - e).$$

is the amount of water that S_* can hold before it overflows. Since $\text{fill}(S'_*) \leq \text{cap}(S'_*)$, S_* does not overflow by filling it with $\text{fill}(S'_*)$ square units of water. By **XVI**, applying the SLM algorithm to S_* with $V = \text{fill}(S'_*)$ yields the elevation levels at time $\text{filltime}(S'_*)$. By lemma 5, applying the algorithm with $V = \text{fill}(S'_*)$ and then with $V = (\text{ot}(S) - \text{filltime}(S'_*))\mathcal{W}(S'_*)$ is the same as applying it once with $V = (\text{ot}(S) - \text{ot}(S'_*))\mathcal{W}(S'_*) + \text{cap}(S'_*)$. Since the first summand is non-positive, we see that no S_* overflows between $\text{filltime}(S'_*)$ and $\text{ot}(S)$.

Note that as long as no sink overflows, there is a bijection between the volume of water in a sink and the time at which this volume has fallen into the sink.

For $S_* = S$, we see that at time $\text{ot}(S)$ it is filled with $\text{cap}(S)$ square units of water. Therefore the elevation levels of all $e \in \mathcal{C}(S)$ become $\mathcal{B}(S)$.

We now show that **XVI** continues to hold for S_l and S_r. By the discussion above, we know that the amount of water in S'_l at time filltime$(S) = \text{ot}(S)$ is

$$\begin{aligned}
&(\text{ot}(S) - \text{ot}(S'_l))\mathcal{W}(S'_l) + \text{cap}(S'_l) \\
&= \left(\text{ot}(S) - \frac{\text{cap}(S'_l) - \text{fill}(S'_l)}{\mathcal{W}(S'_l)} - \text{filltime}(S_l)\right)\mathcal{W}(S'_l) + \text{cap}(S'_l) \\
&= (\text{ot}(S) - \text{filltime}(S_l))\,\mathcal{W}(S'_l) + \text{fill}(S'_l) \\
&= \text{fill}(S_l).
\end{aligned}$$

And therefore $\text{fill}(S_l) \leq \text{cap}(S'_l)$. Therefore applying the SLM algorithm to the original S'_l with volume $\text{fill}(S_l)$ yields the elevation levels at time filltime(S_l).

Recall that $e_r \geq \mathcal{B}(S'_l)$. Therefore none of the water poured into S_l stays on top of the segment e_r, that is, the SLM algorithm returns the segment e_r unchanged. Therefore, extending the segment e_r has no effect on the output of the algorithm.

Therefore applying the algorithm to S_l returns the correct levels in the area previously covered by S'_l. Recall that in the area to the right of S_l that was previously covered by S, the elevation levels have risen to be exactly e_r. Therefore the algorithm returns the correct levels everywhere.

A similar discussion applies to S_r. Therefore **XVI** holds.

Case 2: $\mathcal{B}_l(S) < \mathcal{B}_r(S)$ in line 35.

Since $\mathcal{B}_l(S) < \infty$, there is a sink to the left of S and the operation in in line 36 is well-defined.

Let e_r be as in line 46.

In line 42 we remove the elements of $\mathcal{C}(S)$ from the linked list of segments but in line 47 we add the removed width to e_r. Therefore, the total width of the linked list of segments remains the same. Similarly, in line 81, we remove S from the linked list of sinks but in line 44 we add the removed width to S_m. Therefore, the total width of the linked list of sinks remains the same. In line 43 we extend $\mathcal{E}(S_m)$ to contain the remaining elements of $\mathcal{E}(S)$. Therefore all elements in the linked list of segments are contained in a sink. Therefore **II** holds.

Since $\mathcal{W}_c(S) > 0$ in line 47, **VIII** holds. Since we only modify $\mathcal{W}(S_m)$ and $\mathcal{W}_c(S_m)$ by adding positive numbers to them, **VII** holds. Similarly, we only add non-negative numbers to $\text{cap}(S_m)$. Therefore **X** holds.

Note that the segments that we add to $\mathcal{E}(S_m)$ in line 43 have a length of $\mathcal{W}(S) - \mathcal{W}_c(S)$. However, since we extended e_r by $\mathcal{W}_c(S)$ in line 47, the total growth of the width of the segments in $\mathcal{E}(S_m)$ is $\mathcal{W}(S)$ as in line 44. Therefore the definition of $\mathcal{W}(S_m)$ continues to hold.

If there is no sink to the right of S, it is clear that **IV** continues to hold. Otherwise, note that, since $\mathcal{B}_l(S) < \mathcal{B}_r(S)$, the last element of $\mathcal{E}(S)$ is not in $\mathcal{C}(S)$. Therefore, this element was added to $\mathcal{E}(S_m)$ in line 43. Therefore **IV** continues to hold.

Only the right boundary of S_m is affected by these changes. In line 51 we update $\mathcal{B}_r(S_m)$ to have the correct value. In line 61 we update $\mathcal{B}(S_m)$ to have the correct value. Since $\mathcal{B}_r(S'_m) = \mathcal{B}_l(S) < \mathcal{B}_r(S) = \mathcal{B}_r(S_m)$, **IX** holds.

Since we only add elements to $\mathcal{C}(S_m)$ in lines 70 and 79, **VI** holds.

Since we do not remove any segments from $\mathcal{E}(S_m)$, **V** holds.

We compute as in the previous case $\text{fill}(S_m) \leq \text{cap}(S'_m) \leq \text{cap}(S_m)$. Therefore **XIV** holds. We set $\text{filltime}(S_m) = \text{ot}(S) < T$ in line 41. We compute as in the previous case $\text{ot}(S_m) \geq \text{filltime}(S_m) = \text{ot}(S)$. Therefore **III** and **XIII** hold.

Since we have removed exactly S from B and since we have removed exactly S from the linked list in line 81, **XII** holds.

Note that $e > \mathcal{B}_l(S)$ for all $e \in \mathcal{E}(S) \setminus \mathcal{C}(S)$. In line 43 we extend $\mathcal{E}(S_m)$ only by those elements in $\mathcal{E}(S) \setminus \mathcal{C}(S)$. Let e_m be the local minimum in $\mathcal{E}(S'_m)$. Then $e_m \leq e_r = \mathcal{B}_l(S) < e$ for any $e \in \mathcal{E}(S) \setminus \mathcal{C}(S)$. Furthermore, note that the local minimum of S was contained within $\mathcal{C}(S)$. Since $\mathcal{C}(S)$ was a continuous subset of $\mathcal{E}(S)$ that contained the leftmost segment in S, all elements in $\mathcal{E}(S) \setminus \mathcal{C}(S)$ are monotonically increasing. Therefore e_m remains the single local minimum of $\mathcal{E}(S_m)$ and $\mathcal{E}(S_m)$ is fully merged. Therefore **XI** holds.

Note that when we reach line 61, we have moved the elements of $\mathcal{E}(S) \setminus \mathcal{C}(S)$ to $\mathcal{E}(S_m)$. Since we have removed S from the linked list, the value of $\sum_S (\|\mathcal{E}(S)\| - \|\mathcal{C}(S)\|)$ remains the same. Whenever we reach lines 70 or 79, we extend $\mathcal{C}(S_m)$ by one segment. Therefore $\|\mathcal{E}(S_m)\| - \|\mathcal{C}(S_m)\|$ decreases by one. Therefore **XV** holds.

Let $\mathcal{C}_*(S_m)$, $\text{cap}_*(S_m)$, and $\mathcal{W}_{c*}(S_m)$ be the correctly calculated values of these properties based on the other properties that have been calculated up to this point. We will now show that the rest of the code transforms $\mathcal{C}(S'_m)$, $\text{cap}(S'_m)$, and $\mathcal{W}_c(S'_m)$ to match these values.

Note that

$$\begin{aligned}\text{cap}(S'_m) &= \sum_{e \in \mathcal{C}(S'_m)} \mathcal{W}(e')(\mathcal{B}(S'_m) - e) \\ &= \sum_{e \in \mathcal{C}(S'_m)} \mathcal{W}(e)(\mathcal{B}(S'_m) - e)\end{aligned}$$

since $\mathcal{W}(e)$ only changed for e_r for which $e_r \geq \mathcal{B}(S'_m)$. Therefore

$$\begin{aligned}\text{cap}_*(S_m) - \text{cap}(S'_m) &= \sum_{e \in \mathcal{C}_*(S_m)} \mathcal{W}(e)(\mathcal{B}(S_m) - e) - \sum_{e \in \mathcal{C}(S'_m)} \mathcal{W}(e)(\mathcal{B}(S'_m) - e) \\ &= \sum_{e \in C_1 \cup C_2} \mathcal{W}(e)(\mathcal{B}(S_m) - e) + \sum_{e \in \mathcal{C}(S'_m)} \mathcal{W}(e)(\mathcal{B}(S_m) - \mathcal{B}(S'_m)) \\ &= \sum_{e \in C_1 \cup C_2} \mathcal{W}(e)(\mathcal{B}(S_m) - e) + \mathcal{W}_c(S_m)(\mathcal{B}(S_m) - \mathcal{B}(S'_m))\end{aligned}$$

where C_1 are the elements from $\mathcal{C}_*(S_m)$ to the left of $\mathcal{C}(S'_m)$ and C_2 are the elements from $\mathcal{C}_*(S_m)$ to the right of $\mathcal{C}(S'_m)$.

Note that at this point we have already updated $\mathcal{W}_c(S_m)$ to account for the increased width of e_r if $e_r \in \mathcal{C}(S'_m)$.

Since we have already shown that S_m has a single local minimum, we know that C_1 and C_2 are continuous sequences starting immediately to the left and right of $\mathcal{C}(S'_m)$. Therefore we can walk $\mathcal{E}(S_m) \setminus \mathcal{C}(S'_m)$ starting immediately to the left (right) of $\mathcal{C}(S'_m)$ until we find a segment e for which $\mathcal{B}(S_m) \geq e$ no longer holds.

In line 62 we add the second summand from the equation above to $\text{cap}(S_m)$. In lines 63 to 80 we perform the walks described above and update $\mathcal{C}(S_m)$, $\text{cap}(S_m)$, and $\mathcal{W}_c(S_m)$ to the correct values.

In line 82 we set $\text{ot}(S_m)$ to the correct value.

Therefore **I** holds.

We now show that **XVI** continues to hold for S_m. Recall from the previous case that the amount of water in S'_m at time $\text{filltime}(S) = \text{ot}(S)$ is $\text{fill}(S)$ and that $\text{fill}(S) \leq \text{cap}(S'_m)$. Therefore applying the SLM algorithm to the original S'_m with volume $\text{fill}(S)$ yields the elevation levels at time $\text{filltime}(S_m)$.

Recall that $e_r \geq \mathcal{B}(S'_m)$. Therefore none of the water poured into S_l stays on top of the segment e_r, that is the SLM algorithm returns the segment e_r unchanged. Therefore extending the segment e_r has no effect on the output of the algorithm.

Therefore applying the algorithm to S_m returns the correct levels in the area previously covered by S'_m. Recall by the discussion from the previous case that in the area to the right of S_m that was previously covered by $\mathcal{C}(S)$, the elevation levels have risen to be exactly e_r. Therefore the algorithm returns the correct levels in the area previously covered by $\mathcal{C}(S)$.

Note that the elevation level in $\mathcal{E}(S) \setminus \mathcal{C}(S)$ were left unchanged by the rain falling on S and that these levels are all above e_r. Therefore, since S_l contains these segments unchanged, the algorithm returns the correct levels everywhere.

Therefore **XVI** holds.

Case 3: $\mathcal{B}_l(S) > \mathcal{B}_r(S)$ in line 35.

This is identical to the previous case.

This completes the proof by induction.

Part 2: We now show that the algorithm computes the correct elevation levels.

Note that after the first loop ends, **XVI** continues to hold. As in case 1 above, we see that the amount of water in S at time T is

$$\begin{aligned}
&(T - \text{ot}(S))\mathcal{W}(S) + \text{cap}(S) \\
&= \left(T - \frac{\text{cap}(S) - \text{fill}(S)}{\mathcal{W}(S)} - \text{filltime}(S)\right)\mathcal{W}(S) + \text{cap}(S) \\
&= (T - \text{filltime}(S))\,\mathcal{W}(S) + \text{fill}(S).
\end{aligned}$$

Note that this is the value that we assign to $\text{fill}(S)$ in line 86. In particular, $\text{fill}(S) \leq \text{cap}(S)$. By the discussion in case 1 above, applying the SLM algorithm to S with $V = \text{fill}(S)$ yields the elevation levels at time T for the part of the terrain that is covered by $\mathcal{E}(S)$. By **II**, $\mathcal{E}(S)$ are a partition of the terrain. Therefore the algorithm calculates the levels for the entire terrain.

Part 3: We show that the algorithm runs in $\mathcal{O}(n \log(n))$.

Note that during the construction of the sinks, we split each segment at most once. Therefore, after constructing the sinks, there are at most $2n$ segments in the linked list of segments and $2n$ sinks in the linked list of sinks. Constructing the sinks can be performed in $\mathcal{O}(n)$. Hence, B also contains at most $2n$ elements. Constructing B can be performed in $\mathcal{O}(n \log(n))$.

Since we remove one element from B per iteration of the outer loop and since we never add any elements to B, the outer loop runs at most $2n$ times. Furthermore, since $\sum_S (\|\mathcal{E}(S)\| - \|\mathcal{C}(S)\|) \leq 2n$, by **XV**, the inner loops between lines 63 to 80 iterate at most a total of $2n$ times over the entire runtime of the algorithm. Therefore we visit each line in the algorithm at most $2n$ times.

Popping an element from B takes $\mathcal{O}(\log(n))$ time. Modifying the $\mathrm{ot}(S_l)$, $\mathrm{ot}(S_r)$, and $\mathrm{ot}(S_m)$ takes $\mathcal{O}(\log(n))$ time since we have to maintain the heap property of B. Every other operation within the outer loop is $\mathcal{O}(1)$. Therefore the outer loop runs in $\mathcal{O}(n\log(n))$.

In the second loop, applying the SLM algorithm takes $\mathcal{O}(\|\mathcal{E}(S)\|)$ time. Since $\sum_S \|\mathcal{E}(S)\| \leq 2n$, all applications combined run in $\mathcal{O}(n)$ time. □

Proof of corollary 1. Given $(e_1, \ldots, e_n)$, compute the fully merged elevation sequence in $\mathcal{O}(n)$ time. Apply the previous theorem to compute the elevation levels at time T in $\mathcal{O}(n\log(n))$ time. Retrieve the elevation levels at the points $\{0, \ldots, n-1\}$ by applying lemma 2 in $\mathcal{O}(n\log(n))$ time. □