Let $A \in \mathcal{V}$ source, $B \in \mathcal{V}$ sink. For the following, we suppose that $Turn_{j-1}$ has just finished and $A = Player(j)$ is currently deciding $Turn_j$. We use the following notation:

$$c_{Av} = DTr_{A \to v, j-1}$$
$$c'_{Av} = DTr_{A \to v, j}$$

Moreover, X and X' will be the flows returned by some execution of $MaxFlow_{\mathcal{G}_{j-1}}(A, B)$ and $MaxFlow_{\mathcal{G}_j}(A, B)$ respectively.

Furthermore, we suppose an arbitrary ordering of the members of $N^+(A)$. We set $n = |N^+(A)|$. Thus

$$N^+(A) = \{v_1, ..., v_n\}$$

We use these subscripts to refer to the respective capacities (a.k.a. direct trusts) and flows. Thus

$$\begin{array}{l} x_i = x_{Av_i} \ , \\ c_i = c_{Av_i} \ , \end{array} \text{where } i \in [n]$$

Definition 1 (Trust Reduction).

Trust Reduction on neighbour i is defined as $\delta_i = c_i - c'_i$.
Flow Reduction on neigbour i is defined as $\Delta_i = x_i - c'_i$.
We will also use the standard notation for 1-norm and ∞-norm:

$$||\delta_i||_1 = \sum_{i=1}^{n} \delta_i$$
$$||\delta_i||_\infty = \max_{1 \le i \le n} \delta_i$$

Definition 2 (Restricted Flow).

Let $i \in [n]$. Let $F_{A_i \to B}$ be x'_i when:

$$c'_i = c_i \ \textit{and}$$
$$\forall k \in [n] \setminus \{i\}, c'_k = 0 \ .$$

This definition can be rephrased equivalently as follows:
Let $v \in N^+(A)$. Let $F_{A_v \to B}$ be x'_{Av} when:

$$c'_{Av} = c_{Av} \ \textit{and}$$
$$\forall w \in N^+(A) \setminus \{v\}, c'_{Aw} = 0 \ .$$

Let $L \subset [n]$. Let $F_{A_L \to B}$ be $\sum_{i \in L} x'_i$ when:

$$\forall i \in L, c'_i = c_i \text{ and}$$
$$\forall i \in [n] \setminus L, c'_i = 0 \ \ .$$

The latter definition can be rephrased equivalently as follows:
Let $S \subset N^+(A)$. Let $F_{A_S \to B}$ be $\sum_{v \in S} x'_{Av}$ when:

$$\forall v \in S, c'_{Av} = c_{Av} \text{ and}$$
$$\forall v \in N^+(A) \setminus S, c'_{Av} = 0 \ \ .$$

The choice of the definition will depend on whether K in $F_{A_K \to B}$ is a node, an index or a set of nodes or indices.

Theorem 1 (Saturation theorem).

$$(\forall i \in [n], c'_i \leq x_i) \Rightarrow (\forall i \in [n], x'_i = c'_i)$$

Proof. From the flow definition we know that

$$\forall i \in [n], x'_i \leq c'_i \ \ . \tag{1}$$

In turn $j - 1$, there exists some valid flow Y such that

$$\forall i \in [n], y_i = c'_i$$

with a flow value $\sum_{i=1}^{n} y_i$, which can be created as follows: We start from X and for each (A, v_i) edge we reduce the flow along paths starting from this edge for a total reduction of $x_i - c'_i$ on all those paths. Y is also obviously valid for turn j and, since all capacities c'_i are saturated, there can be no more outgoing flow from the source, thus Y is a maximum flow in $\mathcal{G}_j$. □

Theorem 2 (Trust transfer theorem (flow terminology)).
Let A source, B sink. We create a new graph where

$$\forall i \in [n], c'_i \leq x_i \text{ and}$$
$$\sum_{i=1}^{n} c'_i = F - V \ \ .$$

It then holds that $maxFlow_{\mathcal{G}_j}(A, B) = F' = F - V$.

Proof. From theorem 1 we can see that $x'_i = c'_i$. It holds that

$$F' = \sum_{i=1}^{n} x'_i = \sum_{i=1}^{n} c'_i = F - V \quad .$$

□

Lemma 1 (Flow limit lemma).

$$\forall i \in [n], x_i \leq F_{A_i \to B}$$

Proof. Suppose a flow where $\exists i \in [n] : x_i > F_{A_i \to B}$. If for any $k \neq i$ we choose $c'_k < c_k$, then $x'_i \geq x_i$. We set the new capacities as follows:

$$\begin{aligned} \forall k \neq i, c'_k &= 0 \text{ and} \\ c'_i &= c_i \quad . \end{aligned}$$

Then for X' we will have

$$\begin{aligned} \forall k \neq i, x'_k &= 0 \text{ and} \\ x'_i &= x_i \quad , \end{aligned}$$

which is also a valid flow for $\mathcal{G}_{j-1}$ and thus by definition

$$F_{A_i \to B} = x'_i = x_i > F_{A_i \to B} \quad ,$$

which is a contradiction. Thus the proposition holds. □

Theorem 3 (Trust Saving Theorem).
Suppose some $i \in [n]$ and two alternative capacities configurations, say C'_1 and C'_2 such that

$$\begin{aligned} c'_{1,i} &= F_{A_i \to B} \quad , \\ c'_{2,i} &= c_i \quad , \\ \forall k \in [n] \setminus \{i\}, c'_{1,k} &= c'_{2,k} \quad . \end{aligned}$$

Then $maxFlow_1 = maxFlow_2$.

Proof. From the Flow Limit lemma (1) we know that $x_i \leq F_{A_i \to B}$, thus we can see that any increase in c'_i beyond $F_{A_i \to B}$ will not influence x_i and subsequently will not incur any change on the rest of the flows. □

Theorem 4 (Invariable trust reduction with naive algorithms).
If $\forall i \in [n], c'_i \leq x_i$, then $||\delta_i||_1$ and $||\Delta_i||_1$ are independent of x'_i, c'_i.

Proof. Since $\forall i \in [n], c'_i \leq x_i$, by applying the Saturation theorem (1) we see that $x'_i = c'_i$, thus $\delta_i = c_i - x'_i$ and $\Delta_i = x_i - x'_i$. We know that $\sum_{i=1}^{n} x'_i = F - V$, so we have

$$||\delta_i||_1 = \sum_{i=1}^{n} \delta_i = \sum_{i=1}^{n} (c_i - x'_i) = \sum_{i=1}^{n} c_i - F + V \text{ and}$$
$$||\Delta_i||_1 = \sum_{i=1}^{n} \Delta_i = \sum_{i=1}^{n} (x_i - x'_i) = \sum_{i=1}^{n} x_i - F + V \quad .$$

thus $||\delta_i||_1, ||\Delta_i||_1$ are independent from x'_i and c'_i. □

Until now *MaxFlow* has been viewed purely as an algorithm. This algorithm is not guaranteed to always return the same flow when executed muliple times on the same graph. However, the corresponding flow value, *maxFlow*, is always the same. Thus *maxFlow* can be also viewed as a function from a matrix of capacities to a positive real number. Under this perspective, we prove the following theorem. Let $\mathcal{C}$ be the family of all capacity matrices $C = [c_{vw}]_{V(\mathcal{G}) \times V(\mathcal{G})}$.

Theorem 5 (maxFlow continuity).
Let $p \in \mathbb{N} \cup \{\infty\}$. The function $maxFlow : \mathcal{C} \to \mathbb{R}^+$ is continuous with respect to the $|| \cdot ||_p$ norm.

Proof. Let $C_0 \in \mathcal{C}$. We want to prove that

$$\forall \epsilon > 0, \exists \delta > 0 : 0 < ||C - C_0||_p < \delta \Rightarrow |maxFlow\,(C) - maxFlow\,(C_0)\,| < \epsilon \quad .$$

We will prove it by contradiction. Suppose that

$$\exists \epsilon > 0 : \forall \delta > 0, 0 < ||C - C_0||_p < \delta \Rightarrow |maxFlow\,(C) - maxFlow\,(C_0)\,| \geq \epsilon \quad .$$

Let $v_1, u_1 \in V\,(\mathcal{G})$. Let C such that

$$c_{v_1 u_1} = c_{0,v_1 u_1} + \frac{\epsilon}{2}$$
$$\forall\,(v,u) \in E\,(\mathcal{G}) \setminus \{(v_1, u_1)\}, c_{vu} = c_{0,vu} \quad .$$

Due to the construction, for $\delta = \epsilon$ we have

$$0 < ||C - C_0||_p < \delta \quad . \tag{2}$$

Any valid flow for C_0 is also valid for C, thus

$$maxFlow\,(C_0) \leq maxFlow\,(C) \quad . \tag{3}$$

Also, it is obvious by the way that C was constructed that

$$maxFlow\,(C) \leq maxFlow\,(C_0) + \frac{\epsilon}{2} \tag{4}$$

From (3) we have $maxFlow\,(C_0) \leq maxFlow\,(C) + \frac{\epsilon}{2}$, which, in combination with (4), gives

$$|maxFlow\,(C) - maxFlow\,(C_0)\,| \leq \frac{\epsilon}{2} < \epsilon \ ,$$

which, together with (2) contradicts our supposition. Thus $maxFlow$ is continuous on C_0. Since C_0 is arbitrary, the result holds for all $C_0 \in \mathcal{C}$, thus $maxFlow$ is continuous with respect to $||\cdot||_p$ for any $p \in \mathbb{N} \cup \{\infty\}$. $\square$

Here we show three naive algorithms for calculating new direct trusts so as to maintain invariable risk when paying a trusted party. Let $F = \sum\limits_{i=1}^{n} x_i$. To prove the correctness of the algorithms, it suffices to prove that

$$\forall i \in [n], c'_i \leq x_i \text{ and} \tag{5}$$

$$\sum_{i=1}^{n} c'_i = F - V \ . \tag{6}$$

```
First Come First Served Trust Transfer
Input : old flows x_i, value V
Output : new capacities c'_i
fcfs((x_i), V) :
  n = length(x_i)
  F_cur = F = Σ_{i=1}^{n} x_i
  if (F < V)
    return(⊥)
  for (i = 1 to n)
    c'_i = x_i
  i = 1
  while (F_cur > F - V)
    reduce = min(x_i, F_cur - (F - V))
    F_cur = F_cur - reduce
    c'_i = x_i - reduce
    i += 1
  return(⋃_{i=1}^{n} {c'_i})
```

Proof of correctness for fcfs.

We will first show that at the end of the execution, $i \leq n+1$. Suppose that $i > n+1$ on line 14. This means that $F_{cur,n}$ exists and $F_{cur,n} = F - \sum_{i=1}^{n} x_i = 0 \leq F - V$ since, according to the condition on line 4, $F - V \geq 0$. This means however that the `while` loop on line 9 will break, thus $F_{cur,n+1}$ cannot exist and $i = n+1$ on line 14, which is a contradiction, thus the first proposition holds. We can also note that, even if $i = n+1$ at the end of the execution, the `while` loop will break right after the last incrementation, thus the algorithm will never try to read or write the nonexistent objects x_{n+1}, c'_{n+1}.

We will now show that $\forall i \in [n], c'_i \leq x_i$, as per the requirement (5). Let $i \in [n]$. In line 7 we can see that $c'_i = x_i$ and the only other occurence of c'_i is in line 12 where it is never increased ($reduce \geq 0$), thus we see that the requirement (5) is satisfied.

We will finally show that $\sum_{i=1}^{n} c'_i = F - V$. From line 3 we see that $F_{cur,0} = F$. Let $i \in [n]$ such that $F_{cur,i}$ exists. If $F_{cur,i} \leq F - V$, then $F_{cur,i+1}$ does not exist because the `while` loop (line 9) breaks after calculating $F_{cur,i}$. Else

$$F_{cur,i+1} = F_{cur,i} - \min\left(x_{i+1}, F_{cur,i} - F + V\right) \ . \text{ (lines 10- 11)}$$

If $\nexists i \in [n] : \min\left(x_i, F_{cur,i-1} - (F - V)\right) = F_{cur,i-1} - (F - V)$, then $\forall i \in [n], \min\left(x_i, F_{cur,i} - (F - V)\right) = x_i$, thus from line 12 it will be $\forall i \in [n], c'_i = 0$ and from line 11, $F_{cur,n} = 0$. However, we have

$$\left.\begin{array}{l} min\left(x_n, F_{cur,n-1} - (F - V)\right) \neq F_{cur,n-1} - (F - V) \\ F_{cur,n-1} = x_n \end{array}\right\} \Rightarrow$$
$$\Rightarrow x_n < x_n - (F - V) \Rightarrow F < V$$

which is a contradiction, since if this were the case the algorithm would have failed on lines 4 - 5. Thus

$$\exists i \in [n] : \min\left(x_{i+1}, F_{cur,i} - (F - V)\right) = F_{cur,i} - (F - V)$$

That is the only $i \in [n]$ such that $F_{cur,i+1} = F - V$, so

$$\forall 0 < k < i, F_{cur,k} = F_{cur,k-1} - x_k \Rightarrow F_{cur,i} = F - \sum_{k=1}^{i-1} x_k \ .$$

Furthermore, since $F_{cur,i+1} = F - V$, it is

$$c'_{i+1} = x_{i+1} - F_{cur,i} + F - V = x_i - F + \sum_{k=1}^{i-1} x_k + F - V \Rightarrow$$

$$\Rightarrow c'_{i+1} = \sum_{k=1}^{i} x_k - V \ ,$$

$$\forall k \leq i, c'_k = 0 \text{ and}$$

$$\forall k > i+1, c'_k = x_k \ .$$

In total, we have

$$\sum_{k=1}^{n} c'_k = \sum_{k=1}^{i} x_k - V + \sum_{k=i+1}^{n} x_k = \sum_{k=1}^{n} x_k - V \Rightarrow \sum_{k=1}^{n} c'_k = F - V \ .$$

Thus the requirement (6) is satisfied. □

Complexity of algorithm fcfs. Since i is incremented by 1 on every iteration of the `while` loop (line 13) and $i < n+1$ at the end of the execution, the complexity of the `while` loop is $O(n)$ in the worst case. The complexity of lines 4 - 5 and 8 is $O(1)$ and the complexity of lines 3, 6 - 7 and 14 is $O(n)$, thus the total complexity of algorithm is $O(n)$. □

```
Absolute Equality Trust Transfer ( ||Δ_i||_∞ minimizer)
Input : old flows x_i, value V
Output : new capacities c'_i
abs((x_i), V) :
  n = length(x_i)
  F_cur = F = Σ_{i=1}^{n} x_i
  if (F < V)
    return(⊥)
  X = preprocess(x_i)
  empty = 0
  reduction = 0
  while (F_cur > F - V)
    (i, X) = popMin(X)
    F_prov = F_cur - (n - empty)*(x_i - reduction)
    if (F_prov > F - V)
      reduction = x_i
      empty += 1
```

```
15        F_cur = F_prov
16      else
17        aux = reduction
18        reduction += (F_cur - (F - V)) / (n - empty)
19        F_cur -= (n - empty)*(reduction - aux)
20        #lines 17 & 19 can be replaced by break. In this
21        #case, the loop (line 9) can become while (TRUE).
22    for (i = 1 to n)
23      c'_i = max(0, x_i - reduction)
24    return(⋃_{i=1}^{n} {c'_i})
```

The function $\texttt{preprocess}(x_i)$ returns a data structure $\texttt{X}$ containing the set of flows (x_i), such that the corresponding function $\texttt{popMin(X)}$ is able to repeatedly return the index of a tuple consisiting of the index of the minimum element and a new data structure missing exactly the minimum element. Examples of such pairs of functions are:

$$\begin{cases}\texttt{preprocess = quickSort}\\ \texttt{popMin = (}x_1\texttt{, X}\setminus x_1\texttt{)}\end{cases} \quad \text{and}$$

$$\begin{cases}\texttt{preprocess = FibonacciHeap}\\ \texttt{popMin = (find-min(X),delete-min(X))}\end{cases} .$$

Proof of correctness of abs. First of all, we can note that, if $F_{prov} \leq F - V$ in line 12, then the execution will enter the `else` clause of line 16. Therefore, in line 19, F_{cur} will get the value $F - V$, as we can see by executing the lines 17 - 19 by hand. This in turn means that the loop in line 9 will break right after the `else` clause is executed. Furthermore, the assignment in line 15 in combination with the truth of the statement $F_{prov} > F - V$ in line 12 shows that, if the execution enters the `if` clause of line 12, then the loop of line 9 will be executed at least once more. These two observations amount to the fact that the `else` clause will be executed exactly one time and afterwards the `while` loop will break.

We use the notation $F_{cur,0}$, $reduction_0$ and $empty_0$ to refer to the initial values of the corresponding variables, as set in lines 3, 8 and 7 respectively. Furthermore, the notation $empty_j$, $reduction_j$, $F_{cur,j}$ and i_j is used to refer to the values of the corresponding variables after the j-th iteration of the `while` loop. i_j is chosen in line 10. From lines 11, 13, 15, 12 and 17 to 19 we see that

$$F_{cur,j} = \begin{cases} F_{prov,j}, & F_{prov,j} > F - V \\ F - V, & F_{prov,j} \leq F - V \end{cases} \text{, where}$$

$$F_{prov,j} = F_{cur,j-1} - (n - empty_{j-1})(x_{i_j} - x_{i_{j-1}}), j \geq 1 \text{ and}$$
$$x_{i_0} = 0 \ .$$

It is worth noting that the maximum number of iterations is n, or else $j \leq n$. This holds because, if we suppose that $F_{cur,n+1}$ exists, it is

$$F_{cur,n} > F - V \geq 0 \tag{7}$$

However, we can easily see that in this case

$$F_{cur,n} = F_{cur,0} - \sum_{j=1}^{n}(n-(j-1))(x_{i_j} - x_{i_{j-1}}) =$$
$$= \sum_{j=1}^{n} x_{i_j} - \sum_{j=1}^{n}(n-(j-1))x_{i_j} + \sum_{j=1}^{n-1}(n-j)x_{i_j} =$$
$$= \sum_{j=1}^{n} x_{i_j} - \sum_{j=1}^{n-1}[(n-(j-1)) - (n-j)]x_{i_j} - (n-(n-1))x_{i_n} =$$
$$= \sum_{j=1}^{n} x_{i_j} - \sum_{j=1}^{n-1} x_{i_j} - x_{i_n} = 0 \ ,$$

which is a contradiction to (7), thus $F_{cur,n+1}$ does not exist and $j \leq n$. This means that `popMin()` will never fail.

We will now show that $\forall j \in [n], empty_j < n$. At line 7, it is $empty_0 = 0 < n$. *empty* is again modified in line 14, where it is incremented by at most 1 at each iteration of the `while` loop (line 9). As we saw above, the iterations cannot exceed n and *empty* is not incremented in the last iteration which consists of the `else` clause, thus $\forall j \in [n], empty_j < n$.

Next, we will show that $\forall i \in [n], c'_i \leq x_i$, as per the requirement (5). From line 23, we see that it suffices to prove that $reduction \geq 0$. In line 8, *reduction* is initialized to 0. In line 13, *reduction* is set to x_i, which is always a non-negative value. The last line where *reduction* is modified is 18. In this line, it is $F_{cur} > F - V$ or else the `while` loop would have broken before beginning this iteration and $n > empty$ as we previously saw. Thus the non-negative variable *reduction* is increased and the resulting value is always positive.

We will finally show that $\sum\limits_{i=1}^{n} c'_i = F - V$, which satisfies the requirement (6). Let $k, 0 \leq k \leq n$ be such that at the end of the execution

$$\forall j \leq k, c_{i_j} = 0 \wedge \forall j > k, c_{i_j} > 0 \ .$$

The following holds:

$$\sum_{j=1}^{n} c'_{i_j} = \sum_{j=k+1}^{n} c'_{i_j} = \sum_{j=k+1}^{n} (x_{i_j} - reduction_{k+1}) =$$

$$= \sum_{j=k+1}^{n} (x_{i_j} - (x_{i_k} + \frac{F_{cur,k} - (F - V)}{n-k})) =$$

$$= \sum_{j=k+1}^{n} (x_{i_j} - (x_{i_k} + \frac{F_{cur,0} - \sum\limits_{l=1}^{k} (n-(l-1))(x_{i_l} - x_{i_{l-1}}) - F + V}{n-k})) =$$

$$= \sum_{j=k+1}^{n} (x_{i_j} - (x_{i_k} + \frac{F - \sum\limits_{l=1}^{k} (n-l+1) x_{i_l} + \sum\limits_{l=1}^{k-1} (n-l) x_{i_l} - F + V}{n-k})) =$$

$$= \sum_{j=k+1}^{n} x_{i_j} - (n-k) x_{i_k} - \frac{n-k}{n-k}(-\sum_{l=1}^{k-1} x_{i_l} - (n-k+1) x_{i_k} + V) =$$

$$= \sum_{j=k+1}^{n} x_{i_j} - (n-k) x_{i_k} + \sum_{j=1}^{k-1} x_{i_j} + (n-k+1) x_{i_k} - V =$$

$$= \sum_{j=1}^{n} x_{i_j} - V = F - V \ ,$$

thus the desired property holds. □

Complexity of algorithm abs. Lines 4 - 5, 7 - 8 and 11 - 19 have a complexity of $O(1)$. Lines 2 - 3 and 22 - 24 have a complexity of $O(n)$. The `while` loop of line 9 is repeated at most n times, as we saw in the proof of correctness. Thus the total complexity is

$$O(preprocess) + O(n) O(popMin) \ .$$

If the flows are first sorted, it would be

$$O(preprocess) = O(quicksort) = O(nlogn) \text{ and}$$
$$O(popMin) = O(1) \ ,$$

amounting to a total complexity of $O(nlogn)$. In the case a Fibonacci heap is used, it is

$$O(preprocess) = O(FibonacciHeap) = O(n) \text{ and}$$
$$O(popMin) = O(find\text{-}min) + O(delete\text{-}min) = O(logn) \ ,$$

thus the total complexity is again $O(nlogn)$. □

Proof that abs minimizes $||\Delta_i||_\infty$. Let *reduction* be the final value of the corresponding variable. It holds that

$$\forall i \in [n] : c'_i > 0, x_i - c'_i = reduction \ ,$$
$$\forall i \in [n] : c'_i = 0, x_i - c'_i = x_i \text{ and}$$
$$\forall i \in [n] : c'_i = 0, reduction \geq x_i \ ,$$

thus we deduce that

$$||\Delta_i||_\infty = \max_{1 \leq i \leq n} x_i - c'_i = reduction \ .$$

□

Algorithm 1: Proportional equality trust transfer

Input : x_i flows, $n = |N^+(s)|$, V value
Output: u'_i capacities

1 $F \leftarrow \sum_{i=1}^{n} x_i$
2 **if** $F < V$ **then**
3 | **return** $\perp$
4 **for** $i \leftarrow 1$ *to* n **do**
5 | $u'_i \leftarrow x_i - \frac{V}{F} x_i$
6 **return** $U' = \bigcup_{k=1}^{n} \{u'_k\}$

Proof of correctness for algorithm 1.

– We will show that $\forall i \in [n] \ u'_i \leq x_i$.
According to line 5, which is the only line where u'_i is changed, $u'_i = x_i - \frac{V}{F} x_i \leq x_i$ since $x_i, V, F > 0$ and $V \leq F$.

– We will show that $\sum_{i=1}^{n} u'_i = F - V$.
With $F = \sum_{i=1}^{n} x_i$, on line 6 it holds that $\sum_{i=1}^{n} u'_i = \sum_{i=1}^{n} (x_i - \frac{V}{F} x_i) = \sum_{i=1}^{n} x_i - \frac{V}{F} \sum_{i=1}^{n} x_i = F - V$.

□

Complexity of algorithm 1. The complexity of lines 1, 4 - 5 and 6 is $O(n)$ and the complexity of lines 2 - 3 is $O(1)$, thus the total complexity of algorithm 1 is $O(n)$. □

Naive algorithms result in $u'_i \leq x_i$, thus according to 4, $||\delta_i||_1$ is invariable for any of the possible solutions U', which is not necessarily the minimum (usually it will be the maximum). The following algorithms concentrate on minimizing two δ_i norms, $||\delta_i||_\infty$ and $||\delta_i||_1$.

Algorithm 2: $||\delta_i||_\infty$ minimizer

Input : $X = \{x_i\}$ flows, $n = |N^+(s)|$, V value, ϵ_1, ϵ_2
Output: u'_i capacities

if $\epsilon_1 < 0 \vee \epsilon_2 < 0$ **then**
  return $\perp$
$F \leftarrow \sum_{i=1}^{n} x_i$
if $F < V$ **then**
  return $\perp$
$\delta_{max} \leftarrow \max_{i \in [n]} \{u_i\}$
$\delta^* \leftarrow$ `BinSearch(`$0,\delta_{max},F\text{-}V,n,X,\epsilon_1,\epsilon_2$`)`
for $i \leftarrow 1$ *to* n **do**
  $u'_i \leftarrow \max(u_i - \delta^*, 0)$
return $U' = \bigcup_{k=1}^{n} \{u'_k\}$

Since trust should be considered as a continuous unit and binary search dissects the possible interval for the solution on each recursive call, inclusion of the ϵ-parameters in `BinSearch` is necessary for the algorithm to complete in a finite number of steps.

Algorithm 3: *

Input : bot, top, F', n, X, ϵ_1, ϵ_2
Output: δ^*

function `BinSearch` **if** $bot = top$ **then**
  return bot
else
  for $i \leftarrow 1$ *to* n **do**
    $u'_i \leftarrow \max(0, u_i - \frac{top+bot}{2})$
  if $maxFlow < F' - \epsilon_1$ **then**
    return `BinSearch(`bot, $\frac{top+bot}{2}$,F',n,X,ϵ_1,ϵ_2`)`
  else if $maxFlow > F' + \epsilon_2$ **then**
    return `BinSearch(`$\frac{top+bot}{2}$, top,F',n,X.ϵ_1,ϵ_2`)`
  else
    return $\frac{top+bot}{2}$

Proof that $maxFlow(\delta)$ is strictly decreasing for $\delta : maxflow(\delta) < F$.
Let $maxFlow(\delta)$ be the $maxFlow$ with $\forall i \in [n], u'_i = max(0, u_i - \delta)$. We will prove that the function $maxFlow(\delta)$ is strictly decreasing for all $\delta \leq \max\limits_{i\in[n]}\{u_i\}$ such that $maxFlow(\delta) < F$.
Supppose that $\exists \delta_1, \delta_2 : \delta_1 < \delta_2 \wedge maxFlow(\delta_1) \leq maxFlow(\delta_2) < F$. We will work with configurations of $x'_{i,j}$ such that $x'_{i,j} \leq x_i, j \in \{1, 2\}$.
Let $S_j = \{i \in N^+(s) : i \in MinCut_j\}$. It holds that $S_1 \neq \emptyset$ because otherwise $MinCut_1 = MinCut_{\delta=0}$ which is a contradiction because then $maxFlow(\delta_1) = F$. Moreover, it holds that $S_1 \subseteq S_2$, since $\forall u'_{i,2} > 0, u'_{i,2} < u'_{i,1}$. Every node in the $MinCut_j$ is saturated, thus $\forall i \in S_1, x'_{i,j} = u'_{i,j}$. Thus $\sum\limits_{i\in S_1} x_{i,2} < \sum\limits_{i\in S_1} x_{i,1}$ and, since $maxFlow(\delta_1) \leq maxFlow(\delta_2)$, we conclude that for the same configurations, $\sum\limits_{i\in N^+(s)\setminus S_1} x_{i,2} > \sum\limits_{i\in N^+(s)\setminus S_1} x_{i,1}$.
However, since $x'_{i,j} \leq x_i, j \in \{1, 2\}$, the configuration $[x''_{i,1} = x'_{i,2}, i \in N^+(s) \setminus S_1], [x''_{i,1} = x'_{i,1}, i \in S_1]$ is valid for $\delta = \delta_1$ and then $\sum\limits_{i\in S_1} x''_{i,1} + \sum\limits_{i\in N^+(s)\setminus S_1} x''_{i,1} = \sum\limits_{i\in S_1} x'_{i,1} + \sum\limits_{i\in N^+(s)\setminus S_1} x'_{i,2} > maxFlow(\delta_1)$, contradiction.
Thus $maxFlow(\delta)$ is strictly decreasing. $\square$

We can see that if $V > 0, F' = F - V < F$ thus if $\delta \in (0, \max\limits_{i\in[n]}\{u_i\}] : maxFlow(\delta) = F' \Rightarrow \delta = \min ||\delta_i||_\infty : maxFlow(||\delta_i||_\infty) = F'$.

Proof of correctness for function 3. Supposing that $[F' - \epsilon_1, F' + \epsilon_2] \subset [maxFlow(top), maxFlow(bot)]$, or equivalently $maxFlow(top) \leq F' - \epsilon_1 \wedge maxFlow(bot) \geq F' + \epsilon_2$, we will prove that $maxFlow(\delta^*) \in [F' - \epsilon_1, F' + \epsilon_2]$.
First of all, we should note that if an invocation of `BinSearch` returns without calling `BinSearch` again (line 2 or 11), its return value will be equal to the return value of the initial invocation of `BinSearch`, as we can see on lines 7 and 9, where the return value of the invoked `BinSearch` is returned without any modification. The case where `BinSearch` is called again is analyzed next:

- If $maxFlow(\frac{top+bot}{2}) < F' - \epsilon_1 < F'$ (line 6) then, since $maxFlow(\delta)$ is strictly decreasing, $\delta^* \in [bot, \frac{top+bot}{2})$. As we see on line 7, the interval $(\frac{top+bot}{2}, top]$ is discarded when the next `BinSearch` is called. Since $F' + \epsilon_2 \leq maxFlow(bot)$, we have $[F' - \epsilon_1, F' + \epsilon_2] \subset [maxFlow(\frac{top+bot}{2}), maxFlow(bot)]$ and the length of the available interval is divided by 2.
- Similarly, if $maxFlow(\frac{top+bot}{2}) > F' + \epsilon_2 > F'$ (line 8) then $\delta^* \in (\frac{top+bot}{2}, top]$. According to line 9, the interval $[bot, \frac{top+bot}{2})$ is discarded when the next `BinSearch` is called. Since $F' - \epsilon_1 \geq maxFlow(top)$, we have $[F' - \epsilon_1, F' + \epsilon_2] \subset (maxFlow(top), maxFlow(\frac{top+bot}{2})]$ and the length of the available interval is divided by 2.

As we saw, $[F' - \epsilon_1, F' + \epsilon_2] \subset [maxFlow(top), maxFlow(bot)]$ in every recursive call and $top - bot$ is divided by 2 in every call. From topology we know that $A \subset B \Rightarrow |A| < |B|$, so the recursive calls cannot continue infinitely. $|[F' - \epsilon_1, F' + \epsilon_2]| = \epsilon_1 + \epsilon_2$. Let bot_0, top_0 the input values given to the initial invocation of `BinSearch`, bot_j, top_j the input values given to the j-th recursive call of `BinSearch` and $len_j = |[bot_j, top_j]| = top_j - bot_j$. We have $\forall j > 0, len_j = top_j - bot_j = \frac{top_{j-1} - bot_{j-1}}{2} \Rightarrow \forall j > 0, len_j = \frac{top_0 - bot_0}{2^j}$. We understand that in the worst case $len_j = \epsilon_1 + \epsilon_2 \Rightarrow 2^j = \frac{top_0 - bot_0}{\epsilon_1 + \epsilon_2} \Rightarrow j = \log_2(\frac{top_0 - bot_0}{\epsilon_1 + \epsilon_2})$. Also, as we saw earlier, δ^* is always in the available interval, thus $maxFlow(\delta^*) \in [F' - \epsilon_1, F' + \epsilon_2]$. □

Complexity of function 3. Lines 1 - 2 have complexity $O(1)$, lines 4 - 5 have complexity $O(n)$, lines 6 - 11 have complexity $O(maxFlow) + O(BinSearch)$. As we saw in the proof of correctness for function 3, we need at most $\log_2(\frac{top - bot}{\epsilon_1 + \epsilon_2})$ recursive calls of `BinSearch`. Thus the function 3 has worst-case complexity $O((maxFlow + n)\log_2(\frac{top - bot}{\epsilon_1 + \epsilon_2}))$. □

Proof of correctness for algorithm 2. We will show that $maxFlow \in [F - V - \epsilon_1, F - V + \epsilon_2]$, with u_i' decided by algorithm 2.

Obviously $maxFlow(0) = F, maxFlow(\max_{i\in[n]}\{u_i\}) = 0$, thus $\delta^* \in \max_{i\in[n]}\{u_i\}$. According to the proof of correctness for function 3, we can directly see that $maxFlow(\delta^*) \in [F - V - \epsilon_1, F - V + \epsilon_2]$, given that ϵ_1, ϵ_2 are chosen so that $F - V - \epsilon_1 \geq 0, F - V + \epsilon_2 \leq F$, so as to satisfy the condition $[F' - \epsilon_1, F' + \epsilon_2] \subset [maxFlow(top), maxFlow(bot)]$. □

Complexity of algorithm 2. The complexity of lines 1 - 2 and 4 - 5 is $O(1)$, the complexity of lines 3, 6, 8 - 9 and 10 is $O(n)$ and the complexity of line 7 is $O(BinSearch) = O((maxFlow + n)\log_2(\frac{\delta_{max}}{\epsilon_1+\epsilon_2}))$, thus the total complexity of algorithm 2 is $O((maxFlow + n)\log_2(\frac{\delta_{max}}{\epsilon_1+\epsilon_2}))$. □

However, we need to minimize $\sum_{i=1}^{n}(u_i - u_i') = ||\delta_i||_1$.