An integer `n` is **strictly palindromic** if, for **every** base `b` between `2` and `n - 2` (**inclusive**), the string representation of the integer `n` in base `b` is **palindromic**.

Given an integer `n`, return `true` *if* `n` *is* ***strictly palindromic*** *and* `false` *otherwise*.

A string is **palindromic** if it reads the same forward and backward.

Example 1:

```
Input: n = 9
Output: false
Explanation: In base 2: 9 = 1001 (base 2), which is palindromic.
In base 3: 9 = 100 (base 3), which is not palindromic.
Therefore, 9 is not strictly palindromic so we return false.
Note that in bases 4, 5, 6, and 7, n = 9 is also not palindromic.
```

Example 2:

```
Input: n = 4
Output: false
Explanation: We only consider base 2: 4 = 100 (base 2), which is not palindromic.
Therefore, we return false.
```

Constraints:

- $4 <= n <= 10^5$

You are given a **0-indexed** array of strings `garbage` where `garbage[i]` represents the assortment of garbage at the i^{th} house. `garbage[i]` consists only of the characters `'M'`, `'P'` and `'G'` representing one unit of metal, paper and glass garbage respectively. Picking up **one** unit of any type of garbage takes `1` minute.

You are also given a **0-indexed** integer array `travel` where `travel[i]` is the number of minutes needed to go from house `i` to house `i + 1`.

There are three garbage trucks in the city, each responsible for picking up one type of garbage. Each garbage truck starts at house `0` and must visit each house **in order**; however, they do **not** need to visit every house.

Only **one** garbage truck may be used at any given moment. While one truck is driving or picking up garbage, the other two trucks **cannot** do anything.

Return *the **minimum** number of minutes needed to pick up all the garbage.*

Example 1:

```
Input: garbage = ["G","P","GP","GG"], travel = [2,4,3]
Output: 21
Explanation:
The paper garbage truck:
1. Travels from house 0 to house 1
2. Collects the paper garbage at house 1
3. Travels from house 1 to house 2
4. Collects the paper garbage at house 2
Altogether, it takes 8 minutes to pick up all the paper garbage.
The glass garbage truck:
1. Collects the glass garbage at house 0
2. Travels from house 0 to house 1
3. Travels from house 1 to house 2
4. Collects the glass garbage at house 2
5. Travels from house 2 to house 3
6. Collects the glass garbage at house 3
Altogether, it takes 13 minutes to pick up all the glass garbage.
Since there is no metal garbage, we do not need to consider the metal garbage truck.
Therefore, it takes a total of 8 + 13 = 21 minutes to collect all the garbage.
```

Example 2:

```
Input: garbage = ["MMM","PGM","GP"], travel = [3,10]
Output: 37
Explanation:
The metal garbage truck takes 7 minutes to pick up all the metal garbage.
The paper garbage truck takes 15 minutes to pick up all the paper garbage.
The glass garbage truck takes 15 minutes to pick up all the glass garbage.
It takes a total of 7 + 15 + 15 = 37 minutes to collect all the garbage.
```

Constraints:

- `2 <= garbage.length <= 10^5`
- `garbage[i]` consists of only the letters `'M'`, `'P'`, and `'G'`.
- `1 <= garbage[i].length <= 10`
- `travel.length == garbage.length - 1`
- `1 <= travel[i] <= 100`

You are given a **0-indexed** integer array `nums` of **even** length consisting of an **equal** number of positive and negative integers.

You should **rearrange** the elements of `nums` such that the modified array follows the given conditions:

1. Every **consecutive pair** of integers have **opposite signs**.
2. For all integers with the same sign, the **order** in which they were present in `nums` is **preserved**.
3. The rearranged array begins with a positive integer.

Return *the modified array after rearranging the elements to satisfy the aforementioned conditions*.

Example 1:

```
Input: nums = [3,1,-2,-5,2,-4]
Output: [3,-2,1,-5,2,-4]
Explanation:
The positive integers in nums are [3,1,2]. The negative integers are [-2,-5,-4].
The only possible way to rearrange them such that they satisfy all conditions is [3,-2,1,-5,2,-4].
Other ways such as [1,-2,2,-5,3,-4], [3,1,2,-2,-5,-4], [-2,3,-5,1,-4,2] are incorrect because they do not satisfy one or more conditions.
```

Example 2:

```
Input: nums = [-1,1]
Output: [1,-1]
Explanation:
1 is the only positive integer and -1 the only negative integer in nums.
So nums is rearranged to [1,-1].
```

Constraints:

- `2 <= nums.length <= 2 * 10^5`
- `nums.length` is **even**
- `1 <= |nums[i]| <= 10^5`
- `nums` consists of **equal** number of positive and negative integers.

You want to water `n` plants in your garden with a watering can. The plants are arranged in a row and are labeled from `0` to `n - 1` from left to right where the i^{th} plant is located at `x = i`. There is a river at `x = -1` that you can refill your watering can at.

Each plant needs a specific amount of water. You will water the plants in the following way:

- Water the plants in order from left to right.
- After watering the current plant, if you do not have enough water to **completely** water the next plant, return to the river to fully refill the watering can.
- You **cannot** refill the watering can early.

You are initially at the river (i.e., `x = -1`). It takes **one step** to move **one unit** on the x-axis.

Given a **0-indexed** integer array `plants` of `n` integers, where `plants[i]` is the amount of water the i^{th} plant needs, and an integer `capacity` representing the watering can capacity, return *the **number of steps** needed to water all the plants*.

Example 1:

```
Input: plants = [2,2,3,3], capacity = 5
Output: 14
Explanation: Start at the river with a full watering can:
- Walk to plant 0 (1 step) and water it. Watering can has 3 units of water.
- Walk to plant 1 (1 step) and water it. Watering can has 1 unit of water.
- Since you cannot completely water plant 2, walk back to the river to refill (2 steps).
- Walk to plant 2 (3 steps) and water it. Watering can has 2 units of water.
- Since you cannot completely water plant 3, walk back to the river to refill (3 steps).
- Walk to plant 3 (4 steps) and water it.
Steps needed = 1 + 1 + 2 + 3 + 3 + 4 = 14.
```

Example 2:

```
Input: plants = [1,1,1,4,2,3], capacity = 4
Output: 30
Explanation: Start at the river with a full watering can:
- Water plants 0, 1, and 2 (3 steps). Return to river (3 steps).
- Water plant 3 (4 steps). Return to river (4 steps).
- Water plant 4 (5 steps). Return to river (5 steps).
- Water plant 5 (6 steps).
Steps needed = 3 + 3 + 4 + 4 + 5 + 5 + 6 = 30.
```

Example 3:

```
Input: plants = [7,7,7,7,7,7,7], capacity = 8
Output: 49
Explanation: You have to refill before watering each plant.
Steps needed = 1 + 1 + 2 + 2 + 3 + 3 + 4 + 4 + 5 + 5 + 6 + 6 + 7 = 49.
```

Constraints:

- `n == plants.length`
- `1 <= n <= 1000`
- `1 <= plants[i] <= 10^6`
- `max(plants[i]) <= capacity <= 10^9`