Detecting Molecules

Petr is working for a company that has built a machine for detecting molecules. Each molecule has a positive integer weight. The machine has a *detection range* $[l, u]$, where l and u are positive integers. The machine can detect a set of molecules if and only if this set contains a subset of the molecules with total weight belonging to the machine's detection range.

Formally, consider n molecules with weights $w_0, \ldots, w_{n-1}$. The detection is successful if there is a set of distinct indices $I = \{i_1, \ldots, i_m\}$ such that $l \leq w_{i_1} + \ldots + w_{i_m} \leq u$.

Due to specifics of the machine, the gap between l and u is guaranteed to be greater than or equal to the weight gap between the heaviest and the lightest molecule. Formally, $u - l \geq w_{max} - w_{min}$, where $w_{max} = \max(w_0, \ldots, w_{n-1})$ and $w_{min} = \min(w_0, \ldots, w_{n-1})$.

Your task is to write a program which either finds any one subset of molecules with total weight within the detection range, or determines that there is no such subset.

Implementation details

You should implement one function (method):

- `int[] find_subset(int l, int u, int[] w)`
  - `l` and `u`: the endpoints of the detection range,
  - `w`: weights of the molecules.
  - if the required subset exists, the function should return an array of indices of molecules that form any one such subset. If there are several correct answers, return any of them.
  - if the required subset does not exist, the function should return an empty array.

For the C language the function signature is slightly different:

- `int find_subset(int l, int u, int[] w, int n, int[] result)`
  - `n`: the number of elements in `w` (i.e., the number of molecules),
  - the other parameters are the same as above.
  - instead of returning an array of m indices (as above), the function should write the indices to the first m cells of array `result` and then return m.
  - if the required subset does not exist, the function should not write anything to the `result` array and it should return 0.

Your program may write the indices into the returned array (or to the `result` array in C) in any order.

Please use the provided template files for details of implementation in your programming language.

Examples

Example 1

`find_subset(15, 17, [6, 8, 8, 7])`
In this example we have four molecules with weights 6, 8, 8 and 7. The machine can detect subsets of molecules with total weight between 15 and 17, inclusive. Note, that $17 - 15 \geq 8 - 6$. The total weight of molecules 1 and 3 is $w_1 + w_3 = 8 + 7 = 15$, so the function can return `[1, 3]`. Other possible correct answers are `[1, 2]` ($w_1 + w_2 = 8 + 8 = 16$) and `[2, 3]` ($w_2 + w_3 = 8 + 7 = 15$).

Example 2

`find_subset(14, 15, [5, 5, 6, 6])`
In this example we have four molecules with weights 5, 5, 6 and 6, and we are looking for a subset of them with total weight between 14 and 15, inclusive. Again, note that $15 - 14 \geq 6 - 5$. There is no subset of molecules with total weight between 14 and 15 so the function should return an empty array.

Example 3

`find_subset(10, 20, [15, 17, 16, 18])`
In this example we have four molecules with weights 15, 17, 16 and 18, and we are looking for a subset of them with total weight between 10 and 20, inclusive. Again, note that $20 - 10 \geq 18 - 15$. Any subset consisting of exactly one element has total weight between 10 and 20, so the possible correct answers are: `[0]`, `[1]`, `[2]` and `[3]`.

Subtasks

1. (9 points): $1 \leq n \leq 100$, $1 \leq w_i \leq 100$, $1 \leq u, l \leq 1000$, all w_i are equal.
2. (10 points): $1 \leq n \leq 100$, $1 \leq w_i, u, l \leq 1000$ and $\max(w_0, \ldots, w_{n-1}) - \min(w_0, \ldots, w_{n-1}) \leq 1$.
3. (12 points): $1 \leq n \leq 100$ and $1 \leq w_i, u, l \leq 1000$.
4. (15 points): $1 \leq n \leq 10\,000$ and $1 \leq w_i, u, l \leq 10\,000$.
5. (23 points): $1 \leq n \leq 10\,000$ and $1 \leq w_i, u, l \leq 500\,000$.
6. (31 points): $1 \leq n \leq 200\,000$ and $1 \leq w_i, u, l < 2^{31}$.

Sample grader

The sample grader reads the input in the following format:

- line 1: integers n, l, u.
- line 2: n integers: $w_0, \ldots, w_{n-1}$.

Roller Coaster Railroad

Anna is working in an amusement park and she is in charge of building the railroad for a new roller coaster. She has already designed n special sections (conveniently numbered from 0 to $n-1$) that affect the speed of a roller coaster train. She now has to put them together and propose a final design of the roller coaster. For the purpose of this problem you may assume that the length of the train is zero.

For each i between 0 and $n-1$, inclusive, the special section i has two properties:

- when entering the section, there is a speed limit: the speed of the train must be **less or equal to** s_i km/h (kilometers per hour),
- when leaving the section, the speed of the train is **exactly** t_i km/h, regardless of the speed at which the train entered the section.

The finished roller coaster is a single railroad line that contains the n special sections in some order. Each of the n sections should be used exactly once. Consecutive sections are connected with tracks. Anna should choose the order of the n sections and then decide the lengths of the tracks. The length of a track is measured in meters and may be equal to any non-negative integer (possibly zero).

Each meter of the track between two special sections slows the train down by 1 km/h. At the beginning of the ride, the train enters the first special section in the order selected by Anna, going at 1 km/h.

The final design must satisfy the following requirements:

- the train does not violate any speed limit when entering the special sections;
- the speed of the train is positive at any moment.

In all subtasks except subtask 3, your task is to find the minimum possible total length of tracks between sections. In subtask 3 you only need to check whether there exists a valid roller coaster design, such that each track has zero length.

Implementation details

You should implement the following function (method):

- `int64 plan_roller_coaster(int[] s, int[] t)`.
  - `s`: array of length n, maximum allowed entry speeds.
  - `t`: array of length n, exit speeds.
  - In all subtasks except subtask 3, the function should return the minimum possible total length of all tracks. In subtask 3 the function should return 0 if there exists a valid roller coaster design such that each track has zero length, and any positive integer if it does not exist.

For the C language the function signature is slightly different:

- `int64 plan_roller_coaster(int n, int[] s, int[] t)`.
  - `n`: the number of elements in `s` and `t` (i.e., the number of special sections),
  - the other parameters are the same as above.

Example

```
plan_roller_coaster([1, 4, 5, 6], [7, 3, 8, 6])
```

In this example there are four special sections. The best solution is to build them in the order $0, 3, 1, 2$, and to connect them by tracks of lengths $1, 2, 0$ respectively. This is how a train travels along this railroad track:

- Initially the speed of the train is 1 km/h.
- The train starts the ride by entering special section 0.
- The train leaves section 0 going at 7 km/h.
- Then there is a track of length 1 m. When the train reaches the end of the track, its speed is 6 km/h.
- The train enters special section 3 going at 6 km/h and leaves it at the same speed.
- After leaving section 3, the train travels along a 2 m long track. Its speed decreases to 4 km/h.
- The train enters special section 1 going at 4 km/h and leaves it going at 3 km/h.
- Immediately after special section 1 the train enters special section 2.
- The train leaves section 2. Its final speed is 8 km/h.

The function should return the total length of tracks between the special sections: $1 + 2 + 0 = 3$.

Subtasks

In all subtasks $1 \leq s_i \leq 10^9$ and $1 \leq t_i \leq 10^9$.

1. (11 points): $2 \leq n \leq 8$,
2. (23 points): $2 \leq n \leq 16$,
3. (30 points): $2 \leq n \leq 200\,000$. In this subtask your program only needs to check whether the answer is zero or not. If the answer is not zero, any positive integer answer is considered correct.
4. (36 points): $2 \leq n \leq 200\,000$.

Sample grader

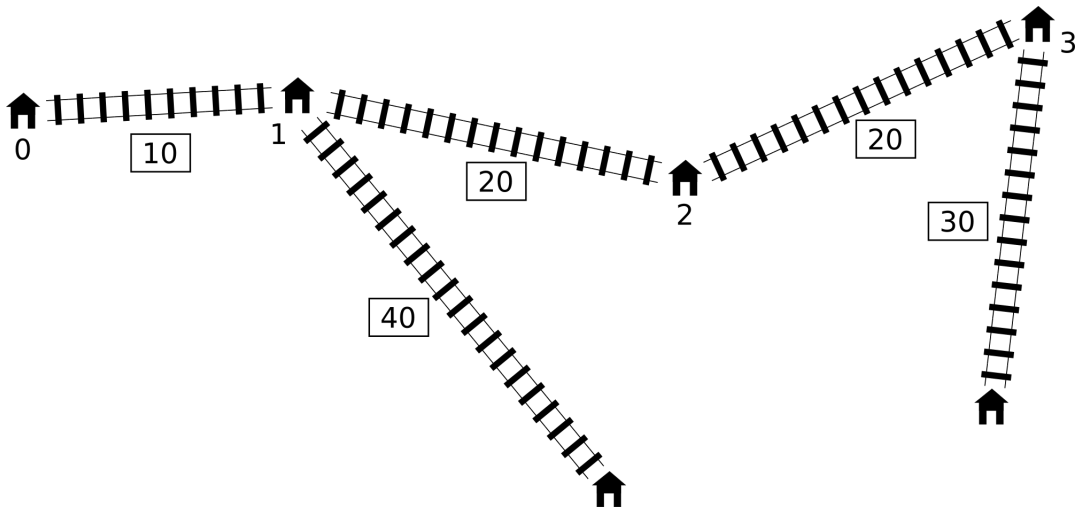
The sample grader reads the input in the following format:

- line 1: integer n.
- line 2 + i, for i between 0 and $n - 1$: integers s_i and t_i.

Shortcut

Pavel has a toy railway. It is very simple. There is a single main line consisting of n stations. These stations are numbered from 0 to $n-1$ in order along the line. The distance between the stations i and $i+1$ is l_i centimeters ($0 \leq i < n-1$).

Apart from the main line there may be some secondary lines. Each secondary line is a railway line between a station on the main line and a new station that does not lie on the main line. (These new stations are not numbered.) At most one secondary line can start in each station of the main line. The length of the secondary line starting at station i is d_i centimeters. We use $d_i = 0$ to denote that there is no secondary line starting at station i.



Pavel is now planning to build one shortcut: an express line between two different (possibly neighbouring) stations of **the main line**. Express line will have length of exactly c centimeters, regardless of what two stations it will connect.

Each segment of the railway, including the new express line, can be used in both directions. The *distance* between two stations is the smallest length of a route that goes from one station to the other along the railways. The *diameter* of the whole railway network is the maximum distance among all pairs of stations. In other words, this is the smallest number t, such that the distance between every pair of stations is at most t.

Pavel wants to build the express line in such a way that the diameter of the resulting network is minimized.

Implementation details

You should implement the function
`int64 find_shortcut(int n, int[] l, int[] d, int c)`

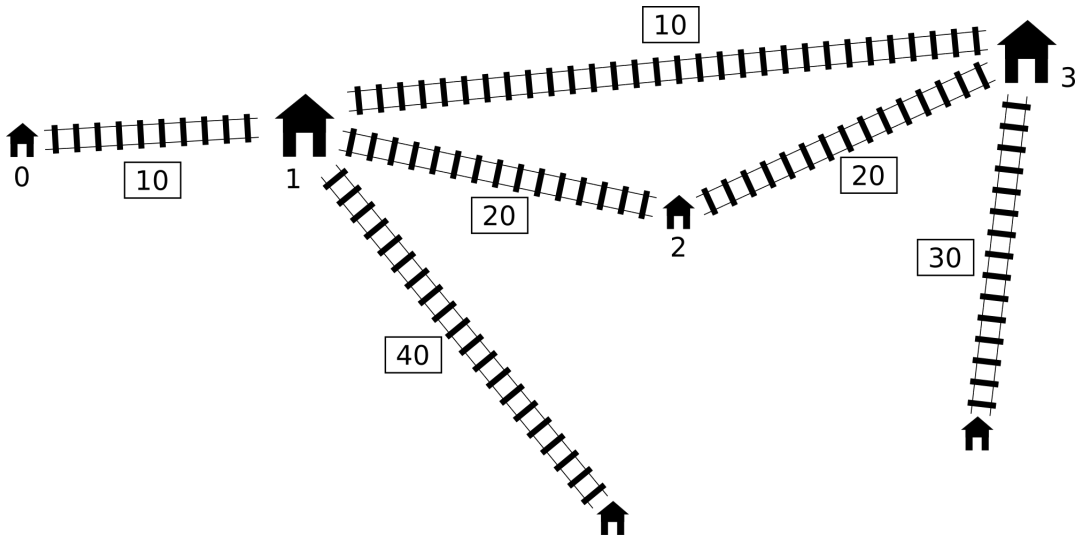
- `n`: number of stations on the main line,
- `l`: distances between stations on the main line (array of length $n-1$),
- `d`: lengths of secondary lines (array of length n),
- `c`: length of the new express line.
- the function should return the smallest possible diameter of the railway network after adding the express line.

Please use the provided template files for details of implementation in your programming language.

Examples

Example 1

For the railway network shown above, the grader would make the following function call:
`find_shortcut(4, [10, 20, 20], [0, 40, 0, 30], 10)`
The optimal solution is to build the express line between stations 1 and 3, as shown below.



The diameter of the new railway network is 80 centimeters, so the function should return 80.

Example 2

The grader makes the following function call:

```
find_shortcut(9, [10, 10, 10, 10, 10, 10, 10, 10],
    [20, 0, 30, 0, 0, 40, 0, 40, 0], 30)
```

The optimal solution is to connect stations 2 and 7, in which case the diameter is 110.

Example 3

The grader makes the following function call:

```
find_shortcut(4, [2, 2, 2],
    [1, 10, 10, 1], 1)
```

The optimal solution is to connect stations 1 and 2, reducing the diameter to 21.

Example 4

The grader makes the following function call:

```
find_shortcut(3, [1, 1],
    [1, 1, 1], 3)
```

Connecting any two stations with the express line of length 3 does not improve the initial diameter of the railway network which is 4.

Subtasks

In all Subtasks $2 \leq n \leq 1\,000\,000$, $1 \leq l_i \leq 10^9$, $0 \leq d_i \leq 10^9$, $1 \leq c \leq 10^9$.

1. (9 points) $2 \leq n \leq 10$,
2. (14 points) $2 \leq n \leq 100$,
3. (8 points) $2 \leq n \leq 250$,
4. (7 points) $2 \leq n \leq 500$,
5. (33 points) $2 \leq n \leq 3000$,
6. (22 points) $2 \leq n \leq 100\,000$,
7. (4 points) $2 \leq n \leq 300\,000$,
8. (3 points) $2 \leq n \leq 1\,000\,000$.

Sample grader

The sample grader reads the input in the following format:

- line 1: integers n and c,
- line 2: integers $l_0, l_1, \ldots, l_{n-2}$,
- line 3: integers $d_0, d_1, \ldots, d_{n-1}$.