Coding Assignment 4

ECS 122A Algorithm Design and Analysis

Radio Towers Again

Problem Description

Note: This coding assignment includes a subtask that admits much simpler solutions and is worth 80% of the points. Therefore, even if you cannot solve the full problem, you may still solve the subtask to get partial credit. See the subtasks section for information.

In ByteLand, there are n cities located along the x axis with the i^{th} city located at $(A_i, 0)$.

In ByteLand, there is a telecommunications company that manufactures radio towers. Recently, there was a tsunami that destroyed all radio towers in the region, and this company is now in charge of reconstructing the radio towers.

The company has decided to build exactly k radio towers in ByteLand. Their radio towers demand an ample power source, and therefore **they can only be built in cities**. The towers are designed to have the same coverage range d. However, the actual coverage of a tower may be influenced by nearby buildings and terrain. After investigation, the company finds that a tower will only have p_i percent of its expected coverage when built in the i^{th} city. That is, if a tower is built in the i^{th} city, it can provide service to all cities j such that $|A_i - A_j| \leq d \cdot p_i/100$.

The company is still trying to figure out the best coverage radius d, because a larger coverage radius would imply a higher cost for building the radio towers. Hence, they want to choose the *smallest* possible coverage radius such that it is possible to provide service to all the n cities. More specifically, they want **to find** the smallest d satisfying the following:

- There exist k cities such that if we build a radio tower in each of those k cities, then the service is provided to all the n cities.

Input and Output

Each input file contains several test instances. The first line contains a single integer C, representing the number of test instances. The description of the C instances follows.

Each test instance consists of three lines:

- The first line contains two space-separated integers n, k.
- The second line contains n space-separated integers $p_1, p_2, \ldots p_n$, where $p_i \in \{20, 40, 60, 80, 100\}$ for $i = 1, 2, \ldots, n$. That is, each percentage is a multiple of 20.
- The third line contains n space-separated integers $A_1, A_2, \ldots A_n$.
- It is guaranteed that the city locations are given in increasing order, i.e., $A_i > A_{i-1}$ holds for $1 < i \leq n$.

For each test instance, your program should output the minimum coverage radius d times 60, i.e., it should output $60 \cdot d$. Under the above constraints, it can be proved that $60 \cdot d$ must be an integer. In other words, d can be represented as a fraction $\frac{q}{60}$ for some integer q, and your program should output

$q = 60 \cdot d$.

In this assignment, you are provided with sample submissions that contain code for input and output. You only need to complete a function that, given n, k, the percentages, and locations of the cities, returns $60 \cdot d$.

Constraints

Each input file contains at most 2000 test instances. In a single input file, the total number of cities across all test instances is at most 10^5. The time limit (for all test instances in a single file) is 4 seconds for C/C++ and 10 seconds for Python.

Each test instance satisfies the following additional constraints:

- $1 \leq k \leq n \leq 10^5$.
- For $i = 1, 2, \ldots, n$, $p_i \in \{20, 40, 60, 80, 100\}$.
- $0 \leq A_i \leq 10^9$ for $i = 1, 2, \ldots, n$.

Test Cases

Your program will be evaluated on 10 test files. Passing all the instances within a file will earn you 0.8 points. Note that if your program outputs an incorrect answer for even one of the instances within a file, you will receive 0 points for that file.

Hints

There is a way to solve this problem by using only integer data types, e.g., `int`, `long long` in C++. To achieve this, you *may* want to scale each A_i by 60. (Please be careful about overflow when scaling the numbers.) If you use floating-point numbers, please ensure your program handles precision errors correctly.

Subtasks

Out of the 10 test files, 8 of them satisfy the additional constraint:

- $p_i = 100$ for all $i = 1, 2, \ldots, n$.

That is, in 8 out of 10 test files, all towers will have the same coverage radius d regardless of which cities they are built. With this additional constraint, it can be proven that the minimum coverage distance d must be an integer.

Sample Input 1:

```
5
5 1
100 100 100 100 100
1 3 5 7 9
5 2
100 100 100 100 100
1 3 5 7 9
5 3
100 100 100 100 100
1 3 5 7 9
5 4
100 100 100 100 100
1 3 5 7 9
5 5
```

```
100 100 100 100 100
1 3 5 7 9
```

Sample Output 1:

```
240
120
120
120
0
```

Sample Input 2:

```
2
6 2
60 20 20 20 20 80
10 20 30 110 120 130
2 1
20 20
0 1000000000
```

Sample Output 2:

```
2000
300000000000
```

Sample Explanation

Sample Input 1 contains $C = 4$ test instances. All test instances have the same $n = 5$, the same percentages $(p_1, p_2, \ldots, p_5) = (100, 100, 100, 100, 100)$, and the same city locations: $(A_1, A_2, \ldots, A_5) = (1, 3, 5, 7, 9)$. The only difference is the value of k. The outputs are explained as follows:

- For $k = 1$, the smallest coverage radius is $d = 4$, because we can build the tower at $A_3 = 5$. Note that the output is $60 \cdot d = 240$.
- For $k = 2$, the smallest coverage radius is $d = 2$, because we can build two towers at $A_2 = 3$ and $A_4 = 7$.
- For $k = 3$ or 4, the smallest coverage radii are also $d = 2$.
- For $k = 5$, the smallest coverage radius is $d = 0$ because we can build one tower at each city.

For Sample Input 2, the solution is $d = \frac{100}{3}$ for the first instance. To achieve this, we should build the towers at $A_1 = 10$ and $A_6 = 130$. The coverage ranges of the two towers are $d \cdot \frac{p_1}{100} = 20$ and $d \cdot \frac{p_6}{100} = \frac{80}{3}$, respectively.

Submission Guideline

Write your program in either C, C++, or Python **in a single file**. Submit the file on Gradescope. The time limit on Gradescope is 4 second for C/C++ and 10 seconds for Python. You can make at most 10 submission attempts. You may refer to `sample-cpp.cpp` or `sample-py.py` for sample code that takes input and writes output.