Problem Statement

Inspired by the surprising connection between the number of collisions and the number π, we would like to solve a more general problem – only computationally. Consider a set of n point objects located in a gravity-free one-dimensional universe, where for $i \in \{0, \ldots, n-1\}$, the i'th object has mass $m_i \in \mathbb{R}$, and at time $t = 0$, it is located at position $x_i \in \mathbb{R}$ and has velocity $v_i \in \mathbb{R}$. Assume that the objects are numbered from left to right, that is, $x_0 < x_1 < \cdots < x_{n-1}$. Given this initial conditions, the objects undergo several collisions over time and eventually move away from one another. Recall that since there is no external force involved, by Newton's law, the total momentum of the objects is conserved during every collision. Assume that each collision is elastic, and therefore, kinetic energy is conserved too. Also, note that no two objects can ever "cross" each other, and therefore, collisions can happen only between objects i and $i+1$, for some $i \in \{0, \ldots, n-2\}$.

We represent a collision by a tuple of a real number, an integer, and another real number. The tuple (t, i, x) represents a collision happening at time t between objects i and $i+1$ at location x. Given a list of masses, a list of initial positions, and a list of initial velocities, each having the same size n, our goal is to enumerate the resulting collisions in a chronological order. Ties between collisions happening at the same time must be broken from left to right. For example for $i < i'$, if at time t, object i collides with $i+1$ at location x, and i' collides with $i'+1$ at location x', then the collision (t, i, x) must precede (t, i', x'). You may assume that the input is such that no more than 2 objects collide at the same time and the same place.

Your task is to write a **Python** function `listCollisions` that takes the following five arguments:

1. `M`: a list of positive floats, where `M[i]` is the mass of the i'th object,
2. `x`: a sorted list of floats, where `x[i]` is the initial position of the i'th object,
3. `v`: a list of floats, where `v[i]` is the initial velocity of the i'th object,
4. `m`: a non-negative integer,
5. `T`: a non-negative float,

and returns a list of collisions in chronological order that ends as soon as the first m collisions happen or time reaches T (whichever earlier). If the input results in fewer than m collisions and the last collision happens before time T, the list returned must contain all collisions in chronological order. Recall that we are representing each collision by a 3-tuple, as mentioned earlier. Round the t and x values of collisions to 4 decimal digits. (However, note that you should still use the exact values, rather than the rounded ones, in your subsequent calculations to avoid accumulating errors.) You may assume that the lists `M`, `x`, `v` have the same size. Identify all the necessary data structures and implement their methods from scratch. For full credit, your function `listCollisions` must run in time $O(n + m \log n)$.

Here is a helpful reference: the formula to find velocities after an elastic collision, given the masses of two colliding objects and their velocities before collision.

Submission Specifications

Submit a single file named `a2.py`. Your submitted file must contain a function `listCollisions(M,x,v,m,T)` that takes three lists of floats, an integer, and a float as arguments. The function must return a list of 3-tuples, each representing a collision.

Example Test Cases

```
>>> listCollisions([1.0, 5.0], [1.0, 2.0], [3.0, 5.0], 100, 100.0)
[]
>>> listCollisions([1.0, 1.0, 1.0, 1.0], [-2.0, -1.0, 1.0, 2.0], [0.0, -1.0, 1.0, 0.0], 5,
 5.0)
```

```
[(1.0, 0, -2.0), (1.0, 2, 2.0)]
>>> listCollisions([10000.0, 1.0, 100.0], [0.0, 1.0, 2.0], [0.0, 0.0, -1.0], 6, 10.0)
[(1.0, 1, 1.0), (1.505, 0, 0.0), (1.6756, 1, 0.3377), (1.7626, 0, -0.0001), (1.8163, 1,
 0.2080), (1.8533, 0, -0.0002)]
>>> listCollisions([10000.0, 1.0, 100.0], [0.0, 1.0, 2.0], [0.0, 0.0, -1.0], 100, 1.5)
[(1.0, 1, 1.0)]
```