1. Adjacency List Runtime

Based on how you implemented your algorithm to fill the adjacency list, we may accept various answers *if you explain how you came to your conclusion*. However, the following two answers are what we would typically expect:

- O(|V| + |E|)

This uses the terminology of vertices (V) and edges (E) in a graph. After all, all of the train stops (the vertices) and their connections (the edges) form a graph.

First, you need to create the adjacency list by going through all of the train stops and creating empty adjacency lists. Since you go through all of the train stops, this is |V| time.

Afterwards, you need to go through the transfers file and add each connection to the adjacency lists. You need to process every adjacency once, so that's |E| time.

Together, that's O(|V|+|E|).

- O(n^2)

This solution is valid if you assume that there are n train stops, and in the worst case, every train stop is connected to every other train stop.

If there are n train stops connected to every other train stop, then there are n-1 connections per train stop.

Thus, if you need to process approximately n adjacencies for n stops, that's n^2 total adjacencies to deal with, or O(n^2).

2. Bad Idea Hash Function

This is not a good idea because you would only be inserting to spots that are multiples of 4, and therefore not hashing to as many possible indices as possible. This will cause many collisions and slow down the operation of your table as more elements are added.

3. Hash Tables

a. Linear Probing

Hash Function: h(x) = x mod M

Initial Table (M = 5):

0: EMPTY
1: EMPTY

2: EMPTY
3: EMPTY
4: EMPTY

Insert 4371 and then 6173:

0: EMPTY
1: 4371
2: EMPTY
3: 6173
4: EMPTY

Remove 6173:

0: EMPTY
1: 4371
2: EMPTY
3: DELETED (notice that this is DELETED and not EMPTY!)
4: EMPTY

Insert 3327 and then 26:

0: EMPTY
1: 4371
2: 3327
3: 26 (notice that 26 would have been inserted to index 1, but both 1 and 2 are filled)
4: EMPTY

Resize to M = 11:
(You cannot just extend the table -- you must re-insert/re-hash all elements.)

0: EMPTY
1: EMPTY
2: EMPTY
3: EMPTY
4: 4371 (or could be 26, if 26 was inserted first)
5: 3327
6: 26 (or could be 4371, if 26 was inserted first)
7: EMPTY
8: EMPTY
9: EMPTY
10: EMPTY

Insert 4199, then 4340, then 9679, then 1323:

0: EMPTY
1: EMPTY
2: EMPTY

3: 1323
4: 4371
5: 3327
6: 26
7: 4340 (would be at index 6, but 26 is there already)
8: 4199
9: EMPTY
10: 9679

b. Separate Chaining

Hash Function: h(x) = x mod M
Remember that you *push_front* elements to each list!

Initial Table (M = 5):

0: nullptr
1: nullptr
2: nullptr
3: nullptr
4: nullptr

Insert 4371, then 1323, then 6173, then 4199, then 4344, then 9679:

0: nullptr
1: 4371
2: nullptr
3: 6173 -> 1323 (order matters!)
4: 9679 -> 4344 -> 4199 (order matters!)

Remove 6173:

0: nullptr
1: 4371
2: nullptr
3: 1323
4: 9679 -> 4344 -> 4199

Insert 3324:

0: nullptr
1: 4371
2: nullptr
3: 1323
4: 3324 -> 9679 -> 4344 -> 4199 (added to the front!)

Resize to M = 11:
(You cannot just extend the table -- you must re-insert/re-hash all elements.)

0: EMPTY
1: EMPTY
2: 3324
3: 1323
4: 4371
5: EMPTY
6: EMPTY
7: EMPTY
8: 4199
9: EMPTY
10: 4344 -> 9679 (order may vary based on which you insert first)

Insert 21:

0: EMPTY
1: EMPTY
2: 3324
3: 1323
4: 4371
5: EMPTY
6: EMPTY
7: EMPTY
8: 4199
9: EMPTY
10: 21 -> 4344 -> 9679 (21 must be at the front)