Exercise Sheet 6
Data Structure Design

Prepared By: Ashik Emon

Detailed Explanation

Goal
We need a data structure that:
- Stores items and pairs of items.
- Supports operations with specific time complexity guarantees.

Let:
- n = number of items
- m = number of pairs

Data Structure Choice
We can use:
- Set or Hash Table for items (fast lookup and insertion).
- Adjacency List using Hash Map for pairs (similar to a graph representation).

Structure:
items: HashSet<Item>
pairs: HashMap<Item, HashSet<Item>>

Methods

initialise() → O(1)

```
function initialise():
    items = empty HashSet
    pairs = empty HashMap
```

Creates two empty data structures without iteration, so runs in constant time O(1).

add_item(x) → amortized O(1), worst O(n)

```
function add_item(x):
    if x in items:
        return "Item already exists"
    items.add(x)
    pairs[x] = empty HashSet
```

Hash insertions are O(1) average; resizing may cost O(n) but amortized remains O(1).

insert_pair(x, y) → O(1)

```
function insert_pair(x, y):
    if x == y:
        return "Error: x and y must be different"
    if x not in items or y not in items:
        return "Error: item missing"
    pairs[x].add(y)
    pairs[y].add(x)
```

Hash lookups and insertions are O(1) average, so constant time.

get_all_pairs(x) → O(1)

```
function get_all_pairs(x):
    if x not in items:
        return "Error: item missing"
    return pairs[x]
```

Direct hash map access returns the set in O(1).

is_pair(x, y) → O(m) (optimized to O(1))

```
function is_pair(x, y):
    if x not in items or y not in items:
        return False
    return y in pairs[x]
```

Membership check in HashSet is O(1) average, improving on O(m).

count(x) → O(1)

```
function count(x):
    if x not in items:
        return "Error: item missing"
    return size(pairs[x])
```

HashSet maintains size internally, so retrieval is O(1).