Version 1:

Q1) Solution:

```
int get_deletion_probes_count(vector<int>& hash_table, int key) {

    int table_size = hash_table.size();
    int index = hashFunc1(key);
    int step = hashFunc2(key);

    int probes = 1;

    // slot occupied
    while (hash_table[index] != key)
    {
        index = (index + step) % table_size;
        probes++;
        if (hash_table[index] == -1)
            return -1;
    }

    return probes;

}
```

Partial grading criteria:

- 1 pt for computing the updated index
- 1 pt for handling the case where the key to be deleted does not exist in the graph. Grant credit if the student has specified an assumption that the key to be deleted exists in the table
- 2 points for the core logic

Q2) Solution: O(1)

Version 2:

Q1) Solution:

```
int get_insertion_probes_count(vector<int>& hash_table, int key) {

    int table_size = hash_table.size();
    int index = hashFunc1(key);
    int step = hashFunc2(key);

    int probes = 1;

    // slot occupied
    while (hash_table[index] != -1)
    {
        index = (index + step) % table_size;
        probes++;

        if (probes > table_size) //no infinite loop if table full
            return -1;
    }

    return probes;

}
```

Partial grading criteria:

- 1 pt for computing the updated index
- 1 pt for handling the case where the hash table is full (probes > table_size)
- 2 points for the core logic

Q2) Solution: O(n)