**Data Structures Used**

1. **Array**:
   * The hash table is implemented as an array of DataItem objects (hashArray).
   * It stores the data items at indices determined by the hash functions.
2. **DataItem**:
   * A simple class representing a key-value pair for the hash table.
   * In this implementation, it only holds the key (iData).

**Algorithms Used**

1. **Hashing with Double Hashing**:
   * **Primary Hash Function (hashFunc1)**:
     + Determines the initial index based on the key modulo the size of the hash table.
   * **Secondary Hash Function (hashFunc2)**:
     + Computes a step size to resolve collisions by ensuring a different probing sequence.
     + Calculated as 5 - (key % 5).
2. **Probing (Collision Resolution)**:
   * Double hashing is used to resolve collisions by linearly probing with a step size determined by the secondary hash function until an empty slot is found or the desired key is located.
3. **Random Key Generation**:
   * Keys are generated randomly within the range [0, 2 \* size] during the initial table filling.

**Time Complexity Analysis**

**1. Insertion**

* **Best Case**:
  + No collision; the key is inserted directly at the hashed index.
  + Time Complexity: O(1)O(1).
* **Worst Case**:
  + Multiple collisions require probing until an empty slot is found.
  + In the worst-case scenario, it may require O(n)O(n) probes in a nearly full table.
* **Average Case**:
  + Depends on the load factor (α=number of elements/table size\alpha = \text{number of elements} / \text{table size}).
  + For double hashing, expected complexity is O(1/(1−α))O(1 / (1 - \alpha)) for α<1\alpha < 1.

**2. Search**

* **Best Case**:
  + The key is found at the first hashed index.
  + Time Complexity: O(1)O(1).
* **Worst Case**:
  + Multiple collisions require probing through the sequence of possible indices.
  + Time Complexity: O(n)O(n) in the worst-case scenario of a full table.
* **Average Case**:
  + Similar to insertion, the complexity is O(1/(1−α))O(1 / (1 - \alpha)).

**3. Display**

* Iterates through the entire array to display the table.
* Time Complexity: O(n)O(n), where nn is the size of the table.

**4. Space Complexity**

* The hash table uses an array of size nn, so the space complexity is O(n)O(n).
* Additional space is minimal, with a few variables for the DataItem objects.