Aadhaar Management System

You are a software engineer working for a government agency responsible for managing Aadhaar numbers, which are unique 12-digit identification numbers assigned to Indian residents. The agency needs an efficient system to store and manage Aadhaar numbers, ensuring quick lookups and minimal collisions.

Your task is to design and implement a hash table of size k that stores these Aadhaar numbers. The hash table should use a **folding technique** to calculate the hash: divide the Aadhaar number into three parts of four digits each (first 4 digits, next 4 digits, and last 4 digits), sum these parts (denoted as x, which can be a five-digit value), and then hash the sum to determine the index using the formula $h(x) = x \mod k$. Additionally, the hash table should use quadratic probing to resolve any collisions and rehash when it becomes 60% full.

Rehashing is the process of reorganizing the hash table when the number of items exceeds a certain capacity threshold. In this system, rehashing occurs when more than 60% of the bins are occupied. The engineer **doubles** the number of bins (from k to $2k$) and reinserts all the existing items (in input order) into the newly expanded hash table according to the new bin size. This process reduces the likelihood of collisions and the need for quadratic probing.

Tasks:

- **Insert an Aadhaar Number** `(HashInsert(T,val)):` The system must allow the insertion of a 12-digit Aadhaar number *val* into the hash table T. The Aadhaar number will be divided into three parts, summed, and hashed to determine the index. If a collision occurs during insertion, the function should resolve it using quadratic probing. Additionally, before insertion.if the hash table becomes 60% full, it should automatically resize by increasing its size by the new size.
- **Displaying the Hash Table** `(DisplayTable(T)):` This function will output the current state of the hash table T, showing the current elements stored in it.

Input Format

- The first line contains an integer k, representing the initial number of bins in the hash table.
- Each subsequent line contains a character from { 'i', 'd', 'e' } followed by zero or one integer value, where the integer value $\in [1, 10^{12}]$.
- The character 'i' is followed by a positive integer 12-digit value 'val' separated by a space. Datatype `long long int` can be used to save *val*, and the format string for the same is `%lld`. Perform `HashInsert(T, val)` operation.
- The character 'd' displays the current state of the hash table$T$.
- The character 'e' terminates the sequence of operations.

Output Format

- The output of each command (if any) should be printed on a separate line. However, no output is printed for the 'i' and 'e' commands.
- For the 'd' command, display the current state of the hash table T, showing the contents of each bin. The resulting table should be displayed with bin values separated by spaces.
- If a bin is empty, it should be represented by -1.

Sample Test Cases

Input 1:

```
9
i 549554030650
i 921669799568
i 876105241277
i 847726180318
d
i 866145760310
i 655801415229
i 716055415139
d
e
```

Output 1:

```
-1 549554030650 847726180318 -1 -1 921669799568 876105241277 -1 -1
-1 847726180318 716055415139 -1 -1 921669799568 -1 -1 -1 -1 549554030650 866145760310
655801415229 -1 876105241277 -1 -1 -1
```

Input 2:

```
7
i 983274651092
i 502938471605
d
i 764320981475
i 412930847561
i 209837461205
i 573829104652
d
e
```

Output 2:

```
983274651092 -1 502938471605 -1 -1 -1 -1
573829104652 -1 764320981475 -1 412930847561 -1 -1 983274651092 209837461205
502938471605 -1 -1 -1 -1
```