Implementation of data structures and algorithms
Short Project 8: Hashing

Due: 11:59 PM, Mon, March 30.

Submission procedure: same as usual.

Team task:

1. Implement Cuckoo hashing algorithm. Use a secondary hash table when the threshold is reached during insertion. You can even use hash table from java library for the secondary hash table.
2. Compare its performance with Java's HashTable/HashMap/HashSet on millions of operations: add, contains, and remove. Compare for load factors of 0.5, and 0.75. Also, investigate the performance of Cuckoo algorithm with the load factor of 0.9. Write a short report on your studies.

Practice task (optional):

1. Given an array A of integers, and an integer X, find how many pairs of elements of A sum to X:

```
static int howMany(int[] A, int X) { // RT = O(n), expected.
// How many indexes i,j (with i != j) are there with A[i] + A[j] = X?
// A is not sorted, and may contain duplicate elements
// If A = {3,3,4,5,3,5} then howMany(A,8) returns 6
}
```

2. Generate an array of random integers, and calculate how many distinct numbers it has: static<T> int distinctElements(T[ ] arr) { ... }
   Compare running times of HashSet and your hashing implementation, for large n.

3. Given an array A, return an array B that has those elements of A that occur exactly once, in the same order in which they appear in A:

```
static<T extends Comparable<? super T>> T[] exactlyOnce(T[] A) {
// RT = O(n),  expected.
// Ex: A = {6,3,4,5,3,5}.  exactlyOnce(A) returns {6,4}
}
```

4. Given an array A of integers, find the length of a longest streak of consecutive integers that occur in A (not necessarily contiguously):

```
static int longestStreak(int[] A) { // RT = O(n), expected.
// Ex: A = {1,7,9,4,1,7,4,8,7,1}.  longestStreak(A) return 3,
//    corresponding to the streak {7,8,9} of consecutive integers
//    that occur somewhere in A.
```

```
}
```