For problem 1, 2, submit the paper submission in class, you do not need to submit source code.
For problem 3, submit your source code to blackboard (only .java or .py files).

1. (a) (10) Implement queue using two stacks. You should implement Enqueue and Dequeue operation. Assuming the size of stacks is unlimited. Just give pseudo-code and describe how it works.
   (b) (5) From the queue you implemented in (a), in which the operations Push and Pop each have cost 1 (that is O(1)), what is the worst-case costs of Enqueue and Dequeue?
   (c) (10) What is the average-case costs of Enqueue and Dequeue? Why? (Hint: Amortized Analysis!)

2. (15) In an array of integers, all elements appear even times except one appears odd times. Find that element. Please give an O(n)-time algorithm.
   For example: input array: [1,2,3,3,2,1,1,1,2], should output 2.

3. (60) Hashtable with Quadratic Probing.
   Implement a simple hashtable using quadratic probing for collision resolution. Both the keys and values are integers, assuming greater than 0. The initial table size \( m \) should be 11 (it is too small for a real hashtable, we just use it for the purpose of this homework). Let \( n \) be the number of items in the table. When \( n \geq m/2 \), use the technique of dynamic arrays to enlarge the table. You want to approximately double the table size but keep to the primes. The next table size \( m \) will be 23.
   You should use key\%m as the hash function.
   Let \( b \) be the hash value modulo \( m \). If bucket \( b \) is occupied, you probe \((b + 1^2) \mod m, (b + 2^2) \mod m, (b + 3^2) \mod m, \ldots, (b + (m/2)^2) \mod m\), and stop as soon as you find an empty bucket.
   As long as \( n \) is kept less than \( m/2 \), you will find an empty bucket by the end of the probing.
   You should at least implement the following functions:
   (a) void put(int key, int value): insert key-value pair to the hashtable. Insert key to the key array, value to the value array based on the hash function.
   (b) int get(int key): get the value of the key
   (c) boolean contains(int key): return true if the hashtable contains the key
   (d) void remove(int key): remove the key-value pair
   (e) void rehashing(): this method is called automatically when \( n \geq m/2 \). You should enlarge the table and use findPrime(2 \* m) to get the new table size. You need to compute new hash index for every key stored in the existing hash table.
   (f) int findPrime(int x): find the next (the smallest) prime bigger than \( x \). For example, findPrime(8) = 11, findPrime(22) = 23