Name:_____Shuqing Ye_____UCI NET ID:_____shuqiny2_____

Test cases (including the edge cases):
input: 1, output: 0
input: 22, output: 0
input: 12, output: 26
input: 48, output: 68
input: 100, output: 455

time complexity: O(log (n)) + O(log (n)) + O(log (n)) = O(log (n)) // 3 helper functions

space complexity: O(log (n)) + O(log (n)) = O(log (n)) // one list, one priority queue

```
int findCoolNumber(int A) {
    // suppose input is greater than 9
    if (A < 10)  return 0;

    List <Integer> primeList = new ArrayList<Integer>();

    if (!getPrimes(A, primeList))
          return 0;

    // build a minHeap
    PriorityQueue<Integer> qp
                = new PriorityQueue<Integer>(
                   (a, b) -> a - b);

   getDigits(primeList , pq);

   return concatenate(pq);
}


// Helper function 1
/* if it is prime number, return fasle;
  * if it is a composite number, divided it up
  * if it has factors greater than 7, return fasle;
  */
boolean getPrimes(int A, List <Integer> list) {
    int primeNum = {2, 3, 5 ,7};
    int i = 0;
    while(A > 1 && i < primeNum.length) {
        if (A % primeNum[i] == 0) {
              list.add(primeNum[i]);
              A /= primeNum[i];
        }
        else i++;
    }

    return A == 1 && i < primeNum.length;
}
```

```
// Helper function 2
/* scan the primeList from start to end, multiply
integers until their product is about to exceed 10
 * put the numbers into priority queue
 */

void getDigits(List <Integer> list ,
PriorityQueue<Integer>pq) {
    int product = list.get(0);
    for (int i = 1; i < list.size(); i++) {
          if (product * list.get(i) < 10) {
               product *= list.get(i);
          }
          else {
                    pq.offer(product);
                    product = list.get(i);
          }
    }

    // put the final number to pq
    pq.offer(product);
}


// Helper function 3
/**
* concatenate numbers we get from getDigits()
* if the result exceeds MAX_VALUE, return 0
*/
int concatenate(PriorityQueue<Integer>pq) {
      int res = 0;
      while(pq.isEmpty()) {
         res = res * 10 + pq.poll();
         if (res < 0) // overflow
             return 0;
      }
      return res;
}
```