Answer for the follow up question of problem1: factors\_and\_caching

1.I will use a directed graph to cache the calculation. Implement the directed graph through a n\*n matrix, (n is the number of elements in the input array). Which is like this:

boolean[][] directedGraph=new boolean[n][n];

If directedGraph[i][j]==true, it indicates that element in position j is a factor of element in position i, else we set directedGraph[i][j] false.

At the beginning set all of the element in directedGraph to false.

During the calculation process, if we find element in position j is a factor of element in position i, we can change directedGraph[i][j] to true.

2. The time complexity of getting a previous calculated result is constant time( O(1) ), because if we want to know whether element in position j is a factor of element of position i, we simply need to check whether directedGraph[i][j] is true, it costs constant time.

The space complexity is O(n\*n) because we use a n by n matrix.

One of the ways of improvement is that we can use n linkedlist instead of a matrix. We can create a linkedlist for each element in the input array and put its factors into the linkedlist accordingly. By this way we can save space but may cost more time when checking whether element in position i is a factor of element in position j, since we need to linear scan the linkedlist of element j to see whether element i exists.

3. Reversing the functionality will not change my cache algorithm, this time we can set directedGraph[i][j] to true when the element in position i is a factor of element in position j.