1. Find one non-repeating element in an array
    - Method 1: (Using sorting)
        - Firstly sort the array using nLog(n) time complexity and compare each element with its adjacent elements. O(nLog(n))
    - Method 2: (Using XOR) (Only works for even number of duplicates)
        - xor all numbers, the remaining number will be the non-duplicate number
        -
```
int findUnique(int* array,int size)
{
        int mask = array[0];
        for (int i = 1;i < size;i++)
                mask ^= array[i];
        return mask;
}
```
    - This question can be extended to check if an array has unique number. We just check if the result is zero (only works for numbers duplicated for even number of times).
2. Find two unique numbers in an array
    - Firstly we xor all numbers in the array, the result indicates the xor result of two unique numbers,. Next we try to find the characteristics of these two unique numbers and partitioning the remaining numbers into two sets. One set will have only one unique number. We look at the position of the bit sets where the bit is set which represents the differences between two numbers. In this way, we know that we can utilize this characteristic. This characteristic can be calculated as mask = mask & ~(mask - 1);After this we use this mask to 'and' the numbers in the array, if equal to zero, we xor the mask with the number and assign it to x, otherwise, we xor the mask with the number and assign it to y.
    -
```
void findTwoUniques(int* array,int size)
{
        int mask = array[0];
        int x = 0,y = 0;
        for (int i = 1;i < size;i++)
                mask ^= array[i];
        mask = mask & ~(mask - 1);
        for (int i = 0;i < size;i++)
        {
                if (mask & array[i])
                        x ^= array[i];
                else
                        y ^= array[i];
        }
        cout << "unique1: " << x << " unique2: " << y << endl;
}
```
3. Check if an array has size N ranging from 1 ~ N has duplicates
    - Method1 : use equation (sum all elements in the array compared to sum of 1 ~ N) if they are equal -> no duplicates
    - Method2 : use xor, xoring 1 ~ N and xoring all elements in the array, if result is zero -> no duplicates

```
bool hasDuplicates(int* array,int size)
{
        int mask = 1;
        for (int i = 2;i <= size;i++)
                mask ^= i;
        for (int i = 0;i < size;i++)
                mask ^= array[i];
        if (mask == 0)
                return false;
        else
                return true;
}
```

4. Find duplicates in an array (array range (1 ~ N))
    - Find two duplicates in an array
        - Use XOR (firstly get the result of xoring 1 ~ N), then use this result to xor every element in the array, so we can get the result of x xor y (x and y are repeating elements). Find the first different bit in this result, based on this characteristic, we can separate the original elements into two parts, one part where the xoring results will have the bit which is set in the characteristic, one part will have the bit not set. Which means those two repeated numbers will be separated. So we can use the xoring results from these two parts and xor the same elements in the 1 ~ N… the result from these two parts will be the final results (O(n), O(1))

```
void findTwoRepeatingNumbers(int* array,int size)
{
        int x = 0,y = 0;
        int mask = 1;
        for (int i = 2;i <= size - 2;i++)
                mask ^= i;
        for (int i = 0;i < size;i++)
                mask ^= array[i];
        cout << "mask: " << mask << endl;
        mask = mask & ~(mask - 1);
        for (int i = 0;i < size;i++)
        {
                if (mask & array[i])
                        x ^= array[i];
                else
                        y ^= array[i];
        }
        for (int i = 1;i <= size - 2;i++)
        {
                if (mask & i)
                        x ^= i;
                else
                        y ^= i;
        }
        cout << "repeat 1: " << x << " repeat 2: " << y << endl;
}
```

- Use index (will modify the original array. Firstly once we encounter an element, we use the absolute value as index, to see if the corresponding slot is a positive number, if so, we set that positive number to negative. If the slot is negative, we have found a repeated number. (O(n), O(1), but will modify the original array).

```
void findTwoRepeatingNumbers1(int* array,int size)
{
        for (int i = 0;i < size;i++)
        {
                if (array[abs(array[i])] > 0)
                        array[abs(array[i])] = -array[abs(array[i])];
                else
                        cout << "found: " << abs(array[i]) << endl;
        }
}
```

5. Find prime numbers
    - Method 1: for a number n, if this number is a prime number, then all numbers between 2 ~ sort (n) divided by n will not yield 0. If the divided result is zero, then this number is a composite number. This algorithm will give us O(nsqrt(n)) time complexity and O(1) space complexity

```
void findPrimeNumbers(int n)
{
        int i,j;
        for (i = 2;i < n;i++)
        {
                for (j = 2;j * j <= i;j++)
                        if (i % j == 0)
                                break;
                if (j * j > i)
                        cout << i << endl;
        }
}
```

- Method 2: firstly we reserve a space of n, assign every number to be 1, 1 represents the index is a prime number otherwise it is a composite number. We know that a composite number can be represented by several prime numbers, e.g 4 (2 + 2), 6 (3 + 3), 8 (2,2,2,2). …Thus, for each index whose value is 1, we start checking the integer multiples of this value, and set those values to 0. Finally all values that are 1 and their indexes represent the prime values.

```
void findPrimeNumbers1(int n)
{
        int* vec = new int[n];
        for (int i = 0;i < n;i++) vec[i] = 1;
        for (int i = 2;i < n;i++)
                if (vec[i])
                        for (int j = i + i;j < n;j += i)
                                vec[j] = 0;
        for (int i = 2;i <= n;i++)
                if (vec[i])
                        cout << i << endl;
}
```