268. Missing Number

```cpp
class Solution {
public:
    // Brute force: O(N^2) time, O(1) space
    int missingNumberBruteForce(vector<int>& nums) {
        int n = nums.size();
        for(int i = 0; i <= n; i++) { // o(N)
            bool found = false;
            for(int x : nums) { // o(N) => NET: O(N*N) = O(N^2)
                if(x == i) {
                    found = true;
                    break;
                }
            }
            if(found == false) return i;
        }
        return -1; // don't expect this return to ever hit.
    }

    // Sort-based: O(NlogN) time, O(1) space
    int missingNumberBetter_1(vector<int>& nums) {
        int n = nums.size();
        sort(nums.begin(), nums.end()); // o(NlogN)
        for(int i = 0; i < n; i++) { // o(N)
            if(nums[i] != i) return i;
        }
        return n;
    }

    // Seen-array: O(N) time, O(N) space
    int missingNumberBetter_2(vector<int>& nums) {
        int n = nums.size();
        vector<bool> seen(n+1, false);
        for(int x : nums) seen[x] = true; // you're present -> o(N)
        for(int i = 0; i <= n; i++){ // o(N) == o(2N), O(N)
            if(!seen[i]) return i; // not present case
        }
        return -1; // not expected to hit
    }

    // Sum-difference: O(N) time, O(1) space
    int missingNumberBest_1(vector<int>& nums) {
        int n = nums.size();
        int expectedSum = (n*(n+1))/2;
        for(int k : nums) expectedSum -= k;// Time: O(N), Space: o(1)
```

```
        return expectedSum;
    }

    // XOR-based optimal: O(N) time, O(1) space
    int missingNumberBest_2(vector<int>& nums) {
        // xor better, never fails for high N, such as N = 10^5
        int xor1 = 0, xor2 = 0;
        for(int i = 0; i < nums.size(); i++) { // O(N) & O(1)
            xor2 ^= nums[i]; // to xor the values of nums array
            xor1 ^= (i + 1); // to track the initial expected array
        }
        return xor1 ^ xor2;
    }

    int missingNumber(vector<int>& nums) {
        return missingNumberBest_2(nums);
    }
};
```

2149. Rearrange Array Elements by Sign

```
class Solution {
public:
    // Bruteforce approach: O(N) time, O(N) extra space
    vector<int> rearrangeArrayBruteForce(vector<int>& nums) {
        int n = nums.size();
        vector<int> pos, neg;
        pos.reserve(n/2);
        neg.reserve(n/2);
        for (int x : nums) { // o(n)
            if (x > 0) pos.push_back(x);
            else       neg.push_back(x);
        }

        vector<int> result(n);
        for (int i = 0; i < n/2; i++) { // O(n/2)  = o(n) + o(n/2), space: O(N)+O(N)
            result[2*i]   = pos[i];
            result[2*i+1] = neg[i];
        }
        return result;
    }

    // Optimal approach: O(N) time, O(N) extra space
    vector<int> rearrangeArrayOptimal(vector<int>& nums) {
        int n = nums.size();
```

```
        vector<int> result(n);
        int posIdx = 0, negIdx = 1;
        for (int x : nums) {
            if (x > 0) {
                result[posIdx] = x;
                posIdx += 2;
            } else {
                result[negIdx] = x;
                negIdx += 2;
            }
        }
        return result; // space: o(N), time: O(N)
    }

    vector<int> rearrangeArray(vector<int>& nums) {
        return rearrangeArrayOptimal(nums);
    }
};
```

75. Sort Colors:

```
class Solution {
public:
    // Brute-force sort: O(N log N) time, O(1) space
    void sortColorsBruteForce(vector<int>& nums) {
        // sort(nums.begin(), nums.end()); // bruteforce, o(nlogn)
        sort(nums.begin(), nums.end());
    }

    // Counting approach: O(N) time, O(1) space
    void sortColorsBetter(vector<int>& nums) {
        // Better
        int count0 = 0, count1 = 0, count2 = 0;
        for(int x : nums) {
            if(x == 0) count0++;
            else if(x == 1) count1++;
            else count2++;
        }
        int i = 0;
        while(count0--) nums[i++] = 0;
        while(count1--) nums[i++] = 1;
        while(count2--) nums[i++] = 2; // O(n)+3*o(n/3)=o(2N), space: O(1)
    }

    // Best: Dutch National Flag algorithm: O(N) time, O(1) space
    void sortColorsOptimal(vector<int>& nums) {
```

```
    // **DUTCH NATIONAL FLAG ALGORITHM**
    int low = 0, mid = 0, high = nums.size() - 1;
    while(mid <= high) {
        if(nums[mid] == 0) swap(nums[low++], nums[mid++]);
        else if(nums[mid] == 1) mid++;
        else swap(nums[mid], nums[high--]); // Space: O(1), Time: O(N)
    }
  }

  void sortColors(vector<int>& nums) {
    sortColorsOptimal(nums);
  }
};
```