#### 1. Merge Sorted Array

class Solution {

public:

void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

int i = m - 1, j = n - 1, k = m + n - 1;

while (i >= 0 && j >= 0) {

nums1[k--] = (nums1[i] > nums2[j]) ? nums1[i--] : nums2[j--];

}

while (j >= 0) {

nums1[k--] = nums2[j--];

}

}

};

#### 2. First Bad Version

class Solution {

public:

int firstBadVersion(int n) {

int left = 1, right = n;

while (left < right) {

int mid = left + (right - left) / 2;

if (isBadVersion(mid)) right = mid;

else left = mid + 1;

}

return left;

}

};

#### 3. Sort Colors

class Solution {

public:

void sortColors(vector<int>& nums) {

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--]);

}

}

};

#### 4. Find Peak Element

class Solution {

public:

int findPeakElement(vector<int>& nums) {

int left = 0, right = nums.size() - 1;

while (left < right) {

int mid = left + (right - left) / 2;

if (nums[mid] > nums[mid + 1]) right = mid;

else left = mid + 1;

}

return left;

}

};

#### 5. Median of Two Sorted Arrays

class Solution {

public:

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

vector<int> merged(nums1.begin(), nums1.end());

merged.insert(merged.end(), nums2.begin(), nums2.end());

sort(merged.begin(), merged.end());

int n = merged.size();

return n % 2 ? merged[n / 2] : (merged[n / 2 - 1] + merged[n / 2]) / 2.0;

}

};