1. Range Sum Query - Mutable

Given an integer array *nums*, find the sum of the elements between indices *i* and *j* (*i* ≤ *j*), inclusive.

The *update(i, val)* function modifies *nums* by updating the element at index *i* to *val*.

**Example:**

Given nums = [1, 3, 5]  
  
sumRange(0, 2) -> 9  
update(1, 2)  
sumRange(0, 2) -> 8

**Constraints:**

* The array is only modifiable by the *update* function.
* You may assume the number of calls to *update* and *sumRange* function is distributed evenly.
* 0 <= i <= j <= nums.length - 1

**解法1** 将查询区间的数字直接求和

class NumArray {  
public:  
 vector<int>Array;  
 NumArray(vector<int>& nums) {  
 Array = nums;  
 }  
   
 void update(int i, int val) {  
 Array[i] = val;  
 }  
   
 int sumRange(int i, int j) {  
 int res = 0;  
 for(int k = i; k <= j; ++k)res += Array[k];  
 return res;  
 }  
};

**解法2** 求和数组。先将数组的前n项和计算出来，更新的时候将前k项和（k>= i）更新即可

class NumArray {  
public:  
 vector<int>S{0};  
 vector<int>Array;  
 NumArray(vector<int>& nums) {  
 Array = nums;  
 for(int i = 0; i < nums.size(); ++i){  
 S.push\_back(S.back() + nums[i]);  
 }  
 }  
   
 void update(int i, int val) {  
 int d = val - Array[i];  
 Array[i] = val;  
 for(int j = i + 1; j < S.size(); ++j)S[j] += d;  
 }  
   
 int sumRange(int i, int j) {  
 return S[j+1] - S[i];  
 }  
};

**解法3** 分块求和。解法2中update函数花费时间较多，更新的平均时间复杂度为，为了控制更新的范围，将数组划分为多个块，更新控制在对应的块内，将块的尺寸取为，更新的时间复杂度为

class NumArray {  
public:  
 int block\_size;  
 vector<int>Array;  
 vector<int>S;  
 NumArray(vector<int>& nums) {  
 Array = nums;  
 block\_size = int(sqrt(nums.size()));  
 int sum = 0;  
 for(int i = 0; i < nums.size(); ++i){  
 sum += nums[i];  
 if((i+1) % block\_size == 0 || i + 1 == nums.size()){  
 S.push\_back(sum);  
 sum = 0;  
 }  
 }  
 }  
   
 void update(int i, int val) {  
 S[i / block\_size] += val - Array[i];  
 Array[i] = val;  
 }  
   
 int sumRange(int i, int j) {  
 int res = 0;  
 int s\_b = i / block\_size, e\_b = j / block\_size;  
 if(s\_b == e\_b){  
 for(int k = i; k <= j; ++k)res += Array[k];  
 }  
 else{  
 for(int k = i; k < (s\_b+1)\*block\_size; ++k)res += Array[k];  
 for(int b =s\_b + 1; b < e\_b; ++b)res += S[b];  
 for(int k = e\_b\*block\_size; k <= j; ++k)res += Array[k];  
 }  
 return res;  
 }  
};

**解法4** 线段树（不想看了。。。）