**Sort Colors**

Given an array nums with n objects colored red, white, or blue, sort them **[in-place](https://en.wikipedia.org/wiki/In-place_algorithm)** so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0,100,and 22 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Solution:

We use the sorting based on partition. Taken 3 pointers low, mid and high.

Low can be treated as 0, mid as 1 and high as 2. Mid can be used to move and swap the content based on values of high.

classSolution{

public:

voidsortColors(vector<int>&nums) {

Int low = 0; int high = len-1;int mid = 0;

while(mid<=high)

{

if(nums[mid] == 2)// if the mid content values which should be present at last.

{

swap(nums[mid],nums[high]);

high--;

}

elseif(nums[mid]==0) // mid content the value which shall be low value

{

swap(nums[mid],nums[low]);

low++;mid++;

}

elseif(nums[mid]==1){ //nothing to swap content correct value

mid++;

}

}

}

};

Design HashMap

Design a HashMap without using any built-in hash table libraries.

Implement the MyHashMap class:

MyHashMap() initializes the object with an empty map.

void put(int key, int value) inserts a (key, value) pair into the HashMap. If the key already exists in the map, update the corresponding value.

int get(int key) returns the value to which the specified key is mapped, or -1 if this map contains no mapping for the key.

void remove(key) removes the key and its corresponding value if the map contains the mapping for the key.

classMyHashMap{

public:

intarr[1000001];

MyHashMap() {

for(inti=0; i<1000001; i++){

arr[i]=-1;

}

}

voidput(intkey, intvalue) {

arr[key] = value;

}

intget(intkey) {

returnarr[key];

}

voidremove(intkey) {

arr[key]=-1;

}

};

/\*\*

\* Your MyHashMap object will be instantiated and called as such:

\* MyHashMap\* obj = new MyHashMap();

\* obj->put(key,value);

\* int param\_2 = obj->get(key);

\* obj->remove(key);

\*/

classMyHashMap{public:unordered\_map<int,int>mp;MyHashMap(){}voidput(intkey,intvalue){mp[key]=value;}intget(intkey){if(mp.find(key)!=mp.end())returnmp[key];return-1;}voidremove(intkey){mp.erase(key);}};

class MyHashMap {

public:

unordered\_map<int,int> mp;

MyHashMap() {

}

void put(int key, int value) {

mp[key]= value;

}

int get(int key) {

if (mp.find(key)!= mp.end())return mp[key];

return -1;

}

void remove(int key) {

mp.erase(key);

}

};

Merge Intervals

Given an array of intervals where intervals[i] = [starti, endi],

merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

classSolution{

public:

vector<vector<int>> merge(vector<vector<int>>&intervals) {

vector<vector<int>> res;

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

intfirst = intervals[0][0],

second = intervals[0][intervals[0].size() - 1];

for(inti = 1; i < intervals.size(); i++) {

if(second >= intervals[i][0] || first >= intervals[i][0]) {

second = max(second, intervals[i][intervals[i].size() - 1]);

first = min(first, intervals[i][0]);

continue;

}

res.push\_back({first, second});

first = intervals[i][0];

second = intervals[i][intervals[i].size() - 1];

}

res.push\_back({first, second});

returnres;

}

};