Insertion Sort

void insertionSort(int arr[], int n) {

int i, key, j;

for (i = 1; i < n; i++){

key = arr[i];

j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

Quick Sort

void swap(int \*a, int \*b){

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int Partition(int \*arr, int front, int end){

int pivot = arr[end];

int i = front -1; // int i為所有小於pivot的數 所形成的數列的「最後位置」

for (int j = front; j < end; j++) { // 從front檢查到end-1(因為end是pivot自己)

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]); // 放到比pivot大的數的「前面」

}

}

i++;

swap(&arr[i], &arr[end]); // 比pivot小 pivot 比pivot大

return i;

}

void QuickSort(int \*arr, int front, int end){

if (front < end) {

int pivot = Partition(arr, front, end);

QuickSort(arr, front, pivot - 1);

QuickSort(arr, pivot + 1, end);

}

}

Merge Sort

const int Max = 1000;

void Merge(vector<int> &Array, int front, int mid, int end){

// 把array[front]~array[mid]放進 LeftSub[]

// 把array[mid+1]~array[end]放進 RightSub[]

vector<int> LeftSub(Array.begin()+front, Array.begin()+mid+1);

vector<int> RightSub(Array.begin()+mid+1, Array.begin()+end+1);

LeftSub.insert(LeftSub.end(), Max); // 在LeftSub[]尾端加入值為 Max 的元素

RightSub.insert(RightSub.end(), Max); // 在RightSub[]尾端加入值為 Max 的元素

int idxLeft = 0, idxRight = 0;

for (int i = front; i <= end; i++) {

if (LeftSub[idxLeft] <= RightSub[idxRight] ) {

Array[i] = LeftSub[idxLeft];

idxLeft++;

}

else{

Array[i] = RightSub[idxRight];

idxRight++;

}

}

}

void MergeSort(vector<int> &array, int front, int end){

if (front < end) { // 表示目前的矩陣範圍是有效的

int mid = (front+end)/2; // mid即是將矩陣對半分的index

MergeSort(array, front, mid); // 繼續divide矩陣的前半段subarray

MergeSort(array, mid+1, end); // 繼續divide矩陣的後半段subarray

Merge(array, front, mid, end);// 將兩個subarray做比較並合併出排序後的矩陣

}

}

HW 5

#include <iostream>

#include <math.h>

using namespace std;

long long int A[1000000];

long long int B[1000000];

long long int C[1000000];

long long int merge(long long int tmp[], long long int left, long long int mid, long long int right) {

long long int i, j, k;

long long int inv\_count = 0;

i = left;

j = mid;

k = left;

while((i <= mid - 1) && (j <= right)){

if (A[i] <= A[j]) {

tmp[k++] = A[i++];

}

else {

tmp[k++] = A[j++];

inv\_count = inv\_count + (mid - i);

}

}

while (i <= mid - 1)

tmp[k++] = A[i++];

while (j <= right)

tmp[k++] = A[j++];

for (i = left; i <= right; i++)

A[i] = tmp[i];

return inv\_count;

}

long long int \_mergeSort(long long int tmp[], long long int left, long long int right) {

long long int mid, inv\_count = 0;

if(right>left){

mid = (right + left) / 2;

inv\_count += \_mergeSort(tmp, left, mid);

inv\_count += \_mergeSort(tmp, mid + 1, right);

inv\_count += merge(tmp, left, mid + 1, right);

}

return inv\_count;

}

long long int MergeSort(long long int n)

{

long long int tmp[n];

return \_mergeSort(tmp, 0, n-1);

}

void countSort(long long int n, long long int r, long long int exp) {

long long int output[n]; // output array

long long int i, count[r] = {0};

for (i = 0; i < n; i++)

count[ (B[i]/exp)%r ]++;

for (i = 1; i < r; i++)

count[i] += count[i - 1];

for (i = n - 1; i >= 0; i--){

output[count[ (B[i]/exp)%r ] - 1] = B[i];

count[ (B[i]/exp)%r ]--;

}

for (i = 0; i < n; i++){

B[i] = output[i];

}

cout << B[0] << " " << B[n-1] << endl;

}

void RadixSort(long long int n, long long int r, long long int m)

{

long long int exp;

for (exp = 1; m/exp > 0; exp \*= r){

countSort(n, r, exp);

}

}

int main()

{

long long int n, r;

long long int inversion\_pair = 0;

long long int max = 0;

long long int max\_index = 0;

long long int max2;

long long int count = 0;

while(cin >> n >> r){

inversion\_pair = 0;

max = 0;

max\_index = 0;

max2 = 0;

for(long long int i=0; i<n ;i++){

cin >> A[i];

B[i] = A[i];

if(A[i] > max){

max = A[i];

max\_index = i;

max2 = A[i];//add

}

}

inversion\_pair = MergeSort(n);

cout << inversion\_pair << endl;

RadixSort(n, r, max2);

}

return 0;

}