# 1.1冒泡排序

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*冒泡排序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

void bubbleSort(int a[], int n)

{

for(int i =0 ; i< n-1; ++i)

{

for(int j = 0; j < n-i-1; ++j)

{

if(a[j] > a[j+1])

{

int tmp = a[j] ; **//交换**

**//需要有中间值**

a[j] = a[j+1] ;

a[j+1] = tmp;

}

}

}

}

**/\*\*\*\*\*\*\*\*\*\*\*鸡尾酒排序(左右开工)\*\*\*\*\*\*\*\*\*\*要会这个\*\*\*\*\*\*\*\*\*\*\*\*/**

void Cocktailsort(int a[], int n)

{

int left=0;

int right=n-1;

while(left<right){

**//前半轮，将最大元素放到后面**

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

if(a[i]>a[i+1]){

std::swap(a[i],a[i+1]);

}

}

right--;

**//后半轮，将最小元素放到前面**

for(int i=right;i>left;i--){

if(a[i]<a[i-1]){

std::swap(a[i],a[i-1]);

}

}

left++;

}

}

# 1.2选择排序

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*选择排序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**//1找到最小的，取其索引号**

void selectionSort (int a[], int n){

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

{

int minindex=i;**//设当前为最小索引**

for(int j=i+1;j<n;j++)

{

if(a[j]<a[minindex])

{

minindex=j;**//当前值与后面的进行比较，取最小的**

}

}

**//2对其当前值与对最小值交换顺序**

if(minindex==i)

continue;

else

{

//实际的排序位置

//a[i]和a[minindex]交换位置

int temp=a[minindex];

a[minindex]=a[i];

a[i]=temp;

}

}

}

# 1.3插入排序

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*直接插入法(向前插)\*\*\*\*\*\*\*\*\*\*On2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**//没有用网页上的逻辑（向后插）**

void InsertSort (int a[], int n)

{

for(int i=1;i<n;i++)**//从第二个元素开始**

{

int j=i;

int target=a[i]**;//提出来保存准备插入**

**//挪位和前面的元素比**

while (j>=1&&target<a[j-1])**//如果比保存的元素大就向前**

{

a[j]=a[j-1];

j--;

}

a[j]=target;//插入

}

}

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*二分插入（前插）\*\*On2但是比直接法要高一点\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**//分治思想，以后会经常看到**

**//其实就是二分查找用在插入里面**

void Bin\_InsertSort (int a[], int n)

{

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

{

int target=a[i];**//提出来保存准备插入**

int left=0; **//左侧边界值**

int right=i-1;  **//右侧边界值**

while (left <= right)

{

int mid = (left+right) / 2; **//中间值**

if(target< a[mid] )

{

right = mid - 1; **//把右侧边界缩小，在中间值得左边进行寻找**

}

else

{

left = mid + 1;  **//把左侧边界加大，在中间值得右边进行寻找**

}

}

for ( int j = i-1; j >= left; j--)

**//将left到i-1之间的数都往后移动一个位置**

**//注意如果要插入的数比前面排好序的数都小（都大）,则不会进入到该循环内，则 此时将不会有数进行位置的移动**

{

a[j+1] = a[j];

}

a[left] = target;  **//将要插入的数值插入到合适位置**

}

}

# 1.4希尔排序

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*希尔排序是非稳定排序算法。

先将整个待排序的记录序列分割成为若干递减增量子序列分别进行直接插入排序.

\*/

//do while

void ShellSort (int a[], int n)

{

int step=1;

int i,j=0;

**//递减增量序列的划分是关键**

while(step<n/3)

{

step=3\*step+1;//寻找合适的增量序列

**//比n的三分之一小的3的倍数多1 这里是4**

}

**//开始排序**

while(step>=1) **//4->1**

{

**//里面就是一个1-->>step的插入法**

for(i=step;i<n;i++) **//和插值法一样都是从第二项开始比第一项**

{

int j=i;

int target=a[i];

**//挪位和前面的元素比**

while (j>=step&&target<a[j-step]) **//如果比保存的元素大就向前**

{

a[j]=a[j-step];

j-=step;

}

a[j]=target; **//插入**

}

step=step/3;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# 2.1快速排序

/\*\*\*\*\*\*\*\*\*\*\*非随机划分函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static int partition\_norand(int a[],int left,int right)

{

int med=a[left]; **//med默认为第一个，在非随机中**

int mark;

int low=left;

int high=right;

**//两种方法的目的是一致的：将比med小的移动到前面，大的移动到后面**

**//1.----单边循环法**

**/\***

mark=left;

int i;

for(i=left+1;i<=right;i++)

{

if(a[i]<=med)

{

mark++;**//保证始终都是哪一个**

std::swap(a[mark],a[i]);

}

}

**//med和重合部分的交换**

std::swap(a[mark],a[left]);**//最后一次将第一个位置放在他应该在的位置**

**\*/**

**//2.这个复杂度高 但是思路更为清晰 适合死记硬背---双边循环法**

while (low!=high)

{

**//将比med（第一个）小的移动到前面 条件是个跳过条件**

while((low<high)&&(a[high]>med)) {high--;}

//2.1 //if(low<high) a[low++]=a[high];

**//将比med（第一个）大的移动到后面**

while((low<high)&&(a[low]<=med)) {low++;}

//2.1//if(low<high) a[high--]=a[low];

**//对跳过后符合交换条件的进行交换**

//2.2

std::swap(a[low],a[high]);

}

**//med和重合部分的交换**

std::swap(a[low],a[left]); **//最后一次将第一个位置放在他应该在的位置**

//return mark;  **//法1返回值**

return low; **//法2返回值**

}

/\*\*\*\*\*\*\*\*\*\*\*随机划分函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int partition(int a[],int left,int right)

{

**//三数取中**

int randmo=(right-left)+left; **//从left到right-left内的随机数**

std::swap(a[randmo],a[left]);

return partition\_norand(a,left,right);

**//不套用也成**

**/\***

int i=left-1;

int j;

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

{

if(a[j]<=a[right])

{

++i;**//保证始终都是哪一个**

if(i!=j)

{

std::swap(a[i],a[j]);

}

}

}

++i;

std::swap(a[i],a[right]);**//最后一次将第一个位置放在他应该在的位置**

return i;

**\*/**

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*快排函数（递归实现）\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void QuickSort (int a[],int low,int high)

{

if(low<high)

{

int p=partition(a,low,high);

QuickSort(a,low,p-1); //前半递归

QuickSort(a,p+1,high); //后半递归

}

}

# 2.2归并排序

**/归并的思想(分治思想)在单向链表合并中也有用到**

**//归并排序：先分割(MergeSort中递归)+归并(Merge中实现，并且实现部分排序功能)**

**/\*1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Merge\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

void Merge(int a[],int start, int mid, int end)

{

**//申请空间来存放两个有序区归并后的临时区域**

int \*temp = (int \*)malloc((end-start+1) \* sizeof(int));

int i=start;

int j=mid+1;

int k=0;

while(i <= mid && j <= end)

{

if(a[i]<a[j])

{

temp[k++]=a[i++]; **//小的先放入临时数组**

}

else

{

temp[k++]=a[j++];

}

}

while(i<=mid) temp[k++]=a[i++];

while(j<=end) temp[k++]=a[j++];

**//将堆空间排序的数据整合到原来数组中**

for(i=0;i<k;i++) a[start+i]=temp[i];

free(temp);

}

/\*2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MergeSort\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void MergeSort(int a[],int start, int end)

{

int mid = start + (end - start) / 2;

if (start >= end) return;

**//分**

MergeSort(a, start, mid);

MergeSort(a, mid + 1, end);

**//合**

Merge(a, start, mid, end);

}

# 2.3堆排序

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*堆排序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\***

**\* @param array 待调整的堆**

**\* @param parentIndex 要下沉的父节点**

**\* @param length 堆的有效大小**

**\*/**

**//下沉调整**

static void downadjust(int a[],int dad,int length)

{**//temp保存父节点的值**

int temp = a[dad];

int son = dad\*2+1;

while (son<=length)//若子节点在范围内才做比较

{ **//如果有右孩子，且右孩子大于左孩子的值，定位右孩子**

if (son + 1 <=length && a[son]<a[son+1])

son++;

**//如果父节点大于子节点代表调整完成，直接跳出函数**

if (a[dad] > a[son])

break;

else {

**//否则交换父子內容再继续子节点与孙节点比較**

swap(a[dad], a[son]);

dad = son;

son = dad \* 2 + 1;

}

}

a[dad]=temp;

}

static void heapsort(int a [],int length)

{

**//1.将无顺的数组结构构建成二叉堆 复杂度O（n） i从最后一个父节点开始调整**

for (int i=length/2 - 1;i>=0;i--)

downadjust(a,i,length-1);

**//2.循环删除堆顶元素，并将元素移动到结合尾部O((n-1)logn)~>O(nlog(n) 开始堆排序**

**//先将第一个元素和已经排好的元素前一位做交换，再从新调整(刚调整的元素之前的元素)，直到排序完成**

for (int i=length-1;i>0;i--)

{

swap(a[0], a[i]);

downadjust(a,0,i-1);

}

}

# 3.1计数排序

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

//后面的算法就要用Vector了

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*基于数组特性\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*计数排序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**//虽然时间空间复杂度由于一切基于比较的排序算法**

**//但是对于非整数，过于大的数据量的数组，并不适用**

void CountSort(vector<int>&a,vector<int>&b)

{

**// 确保待排序容器非空**

if (a.size() == 0)

return;

**//1.得到数列的最大值和最小值，计算差值**

int min=0;

int max=0;

for(int i=0;i<a.size();i++)

{

if(a[i]>max) max=a[i];

if(a[i]<min) min=a[i];

}

int d=max-min;

**//2.创建统计数组并统计对应元素的个数**

vector<int> countArray(d+1, 0);

for (int i = 0; i<a.size(); i++)

countArray[a[i]-min]++;//这里减min的目的是优化计数数组的空间

**//3.统计数组变形，后面的键值出现的位置为前面所有键值出现的次数之和**

for (int i = 1; i<d+1; i++)

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

**//4.倒序遍历原始数列，通过统计数组的\_\_\_\_\_\_\_将键值放到目标位置**

**//通过计数数组下标逆序遍历，逆序写入sortArray**

for (int i =a.size()-1; i >=0; i--) // 此处逆序是为了保持相同键值的稳定性

{

b[countArray[a[i]-min]-1] = a[i];

countArray[a[i]-min]--;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

vector<int> a={51,3,4,5,15,54,3,6,36,67,1,35,47,45,43};

vector<int> b(a.size(), 0);

CountSort(a, b);

for (int i = 0; i < b.size(); ++i)

cout << b[i] << " ";

cout << endl;

}

# 3.2桶排序

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*桶排序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**//为了应对计数排序的1.范围不能太大 2.元素必须是正整数的缺点**

**//将计数单位变成了一个范围-----一个桶**

void BucketsSort(vector<double>&a,vector<double>&b)

{

/**/ 确保待排序容器非空**

if (a.size() == 0)

return;

**//1.得到数列的最大值和最小值，计算差值**

int min=0;

int max=0;

for(int i=0;i<a.size();i++)

{

if(a[i]>max) max=a[i];

if(a[i]<min) min=a[i];

}

int d=max-min;

**//2.初始化桶**

int bucketSize =a.size();

int bucketCount = floor(d/bucketSize) + 1; //浮点数向下取整

vector<double> buckets [bucketCount];

**//3.利用映射函数将数据分配到各个桶中**

for (int i = 1; i<a.size(); i++)

{

int bi=floor(a[i]-min)/bucketSize;

buckets[bi].push\_back(a[i]);

}

**//4.对每一个桶的元素进行排序，再放入数组**

for (int i=0;i<bucketCount;i++)

{

sort(buckets[i].begin(),buckets[i].end());// 对每个桶进行排序，这里使用了插入排序

for (int j = 0; j < buckets[i].size(); j++)

{

b.push\_back(buckets[i][j]);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# 7.3基数排序（不想学，心好累）