**一、冒泡排序：**

（1）思想是：  
从第1个开始，1和2比，2和3比，3和4比，如果前面比后面大，则互相交换之，一直到n-1和n进行比。这是第一轮。

然后第二轮再从第1个开始，2和3比，3和4比，再一直比到n-1和n，比的时候符合条件（前大后小）则交换。

然后一直到从n-1个开始，最后比较一次n-1和n。

因此，时间复杂度是O(n2)；

代码：

int n = 100;

int m[100];

for (int i = 0; i < n; i++)//赋值，从m[0]到m[99]，值是100->1

m[i] = 100 - i;

for (int i = 0; i < n; i++) //显示当前排序

cout << m[i] << " ";

cout << "————分割线————" << endl;

for (int i = 0; i<n; i++) //排序

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

{

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

{

int temp;

temp = m[j];

m[j] = m[j + 1];

m[j + 1] = temp;

}

}

for (int i = 0; i < n; i++)//显示当前排序

cout << m[i] << " ";

**二、插入排序：**

（1）其思想是：

①首先，给最左边留一个空（用于临时存储要被移动的数字），例如a[0]

②然后从右往左开始比；例如从a[1]开始

③假如当前值a[3]比其左边的大，进入判断，否则继续下一个数字；

④假如当前数值比左边的小，于是，把当前值给最左边预留的空位（a[0]）；

⑤然后进入循环，问，现在这个位置（第j位）值，是不是是比左边的小，如果小，将左边的值赋给他（而他的值目前在a【0】处），然后当前位置（j）往左移动一位（j--）并且再一次判断，移动后的这个位置，其值是不是比左边的值小，如果小，执行相同的指令；

⑥假如当前位置的值比左边的大了，于是终止循环，由于记录了终止循环时的位置（此时的位置的值，要么是完全没有交换，要么是把当前位置的值赋给了他右边的位置），因此，把之前存储在a【0】位置的值，赋给他（无论是哪种情况都不影响）。

⑦因此，只要被排序过的，一定是小的在左边，大的在右边（大的都被移动到右边去了），等排序完了，整体数组一定是小的在左边，大的在右边了。

代码：

void charu() //插入排序

{

using namespace std;

int n = 100;

int m[100];

for (int i = 0; i < n; i++)//赋值，从m[0]到m[99]，值是100->1

m[i] = 100 - i;

for (int i = 0; i < n; i++) //显示当前排序

cout << m[i] << " ";

cout << "————分割线————" << endl;

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

{

int temp = m[i]; //临时存放这个m[i]（插入的数字）

int j = i; //记录j=i

if (j>0 && m[j] < m[j - 1]) //首先j要>0（防止出界），然后插入这个位置比左边位置的小

while (temp < m[j - 1] && j>0) //开始循环，将左边放到右边，直到左边的比右边小

m[j--] = m[j - 1]; //每次交换后，都要往左移动一位（即第一次是交换n-1和n，第二次就是n-2和n-1）

m[j] = temp; //此时j的位置被赋值（即插入的位置，右边都比他大，左边也比他小）

}

for (int i = 0; i < n; i++) //显示当前排序

cout << m[i] << " ";

}

**三、希尔排序：**

相当于连续多次的插入排序（但比插入排序优化）

时间复杂度是f(n)=n log n;（比n2要小）

代码：

//希尔排序

for (gap = length / 2; gap > 0; gap /= 2) //每次步长减半

{

for (i = 0; i < gap; i++) //步长有多少，就移动多少次

{

for (j = i + gap; j < length; j += gap) //以步长为间距进行交换，注意，初始位置是i位置加步长的位置（即第二步）

{

if (m[j] < m[j - gap]) //当前位置和其步长位置之前的进行比较（以步长为间距）。第一次是第二步和第一步进行比较

{

int temp = m[j];

int k = j - gap; //两个数字的位置差，k是第一步的位置

while (k >= 0 && m[k]>temp) //k>0说明在右边，要交换的位置在右边。第二个指第一步比临时存储第二步大（所以需要交换，否则第一步比第二步小则不用交换）

{

m[k + gap] = m[k]; //当前位置和右边的交换

k -= gap; //然后往左移动一个步长（同组的左边那个）

}

m[k + gap] = temp; //将存储在临时的，赋值给移动后最左边的位置

}

}

}

}

**四、简单选择排序**

（1）思想：

从第1个开始，然后先看第2个是否比第一个小，是则交换然后继续，不是则继续。然后再看第3个是否比第一个小，判断是否交换，再看第4个，以此类推。一直到第n个。——因此，可能比较了n-1次，但是0次交换（说明排序前，该位置是没问题的），也可能是若干次交换，但无论如何，当比较n-1次后，该位置的数字就是它应该的数字。

然后从第2个开始，依次比较3、4、5...一直到第n个。

总比较次数是（n-1+1）\*(n-1)/2次

（2）时间复杂度：  
 f(n)=O(n2);

（3）代码：

int n = 100;

int m[100];

for (int i = 0; i < n; i++)//赋值，从m[0]到m[99]，值是100->1

m[i] = 100 - i;

for (int i = 0; i < n; i++) //显示当前排序

cout << m[i] << " ";

cout << "————分割线————" << endl;

for (int i = 0; i < n-1; i++) //第i轮，总计n-1轮

for (int j = i; j < n; j++) //第i轮中的循环

{

if (m[j] < m[i]) //后面比前面小则交换

{

int temp;

temp = m[j];

m[j] = m[i];

m[i] = temp;

}

}

for (int i = 0; i < n; i++) //显示当前排序

cout << m[i] << " ";

**五、归并排序**

（1）思路：

首先，将数组对半分拆，放入1个新数组之中（分为前后两部分）；

使用递归，分别对分拆后的数组（前后某个部分）继续分拆，放入一个更新的数组之中（分为前后两部分），此时，这个更新的数组，长度和旧的是一样的（但由于递归，每次占用的相对于总长度而说越来越少）；

一直到对半分后为1个的情况下，将其放入相对较旧的那个数组对应的位置之中。

然后递归返回，开始排序。

此时，是有一较旧数组和一较新数组，有较新数组使用的初始下标、中间下标和结束下标（以中间为分割，分为前后两部分）。

然后对较新数组的前后两部分进行排序，放入较旧的数组之中。

继续递归返回，此时较旧数组作为较新数组，和另外一个较新数组一起属于一个较新数组的前后两部分，然后通过排序，放入较旧数组之中。

一直递归到初始数组为止。

然后新数组就是排序好的。

（2）代码：

int n = 100;

int m[100];

for (int i = 0; i < n; i++)//赋值，从m[0]到m[99]，值是100->1

m[i] = 100 - i;

for (int i = 0; i < n; i++) //显示当前排序

cout << m[i] << " ";

cout << "————分割线————" << endl;

int\*p = new int[n];

guibing(m, p, 0, n-1);

for (int i = 0; i < n; i++) //显示当前排序，这里要输出新数组p的

cout << p[i] << " ";

delete[]p; //删除（如果需要保留则不删除）

void guibing(int \*old, int \*ne, int first, int last) //旧数组old（没排序的），新数组ne（排序好的），first（数组第一个元素），数组的最后一个元素的下标

{

int middle;

int \*ne2 = new int[last + 1]; //ne2的长度是数组总长

if (first == last) //如果相等，说明数组第一个元素就是最后一元素

ne[first] = old[first]; //将没排序的放到排序好的那个数组对应的位置

else

{

middle = (first + last) / 2; //寻找中间的坐标

guibing(old, ne2, first, middle); //把

guibing(old, ne2, middle + 1, last);

caozuo(ne2, ne, first, middle, last);

}

delete[]ne2;

}

void caozuo(int\*ne2, int\*ne, int i, int m, int n)

{

int j, k, l;

for (j = m + 1, k = i; i <= m&&j <= n; k++) //j是后半部分数组的起始下标，k是前半部分数组的起始下标，当前半部分下标i比m大，或者后半部分下标比n大则结束循环

{

if (ne2[i] < ne2[j]) //如果较新数组的后半部分位置j比前半部分对应数组位置k大

ne[k] = ne2[i++]; //把小的那个放入较旧的数组的k位置（之后k+1），并且小的那个移动到下一个位置

else

ne[k] = ne2[j++]; //同上

}

//此时，前后部分必然有一部分复制完，另外一部分剩1个或者更多

if (i <= m) //如果前面的没有复制完，则复制完

{

for (l = 0; l <= m - i; l++)

ne[k + l] = ne2[i + l];

}

if (j <= n)

{

for (l = 0; l <= n - j; l++)

ne[k + l] = ne2[j + l];

}

}

**六、快速排序**

（1）思想：

①先用第一个元素，作为比对值key。

②然后从后面开始找，如果有比key小的，交换之（key到后面）；

③然后再从前面找，有比key大的，和key交换，（key又到前面）；然后再从后面找，②③循环，直到前后相遇

④于是前后相遇的地方为中间值，返回其下标。

此时，这个下标前面的必然比他小，后面的必然比他大。原因是数组中每个数他都比较过了，并且把比他小的通过交换放他前面了，比他大的放他后面了。

⑤以下标为中心，分为前后两部分（不包括下标所在数字），因为下标所在数字是其正确的位置。

⑥前后两部分分别进行②③循环，形成递归（并且递归的时候，每次至少将一个数字移动到其正确的位置）。直到每部分剩了一个数字为止（剩2个数字的时候依然在交换，并且交换后两部分各一个数字然后停止）。

代码：

int n = 100;

int m[100];

for (int i = 0; i < n; i++)//赋值，从m[0]到m[99]，值是100->1

m[i] = 100 - i;

for (int i = 0; i < n; i++) //显示当前排序

cout << m[i] << " ";

cout << "————分割线————" << endl;

QuickSort(m, 0, 99);

for (int i = 0; i < n; i++) //显示当前排序

cout << m[i] << " ";

void swap(int&a, int&b) //交换2个值

{

int temp = a;

a = b;

b = temp;

}

void QuickSort(int\*m, int f, int l) //参数为数组、第一个下标和最后一个下标

{

int mid;

if (f < l) //如果第一个比最后一个小

{

mid = getMid(m, f, l); //得到中间值，并将中间值放到最中间位置

QuickSort(m, f, mid - 1); //从开始到中间前一个

QuickSort(m, mid + 1, l); //从中间后一个到最后

}

}

int getMid(int\*m, int f, int l) //参数为，数组，第一个下标，最后一个下标

{

int key = m[f]; //key是第一个数字的下标

while (f < l)

{

while (f < l&&m[l] >= key) //第一个下标比最后一个下标小，并且最后一个下标的值大于等于key（第一个下标的值）

l--; //最后一个往里面靠一位，直到找到一个后面比前面小标小的

swap(m[f], m[l]); //因为后面比前面那个小，所以交换之

while (f < l&&m[f] <= key) //最前面那个下标比key小，则最前面的下标往后移动一位

f++;

swap(m[f], m[l]); //这是前面的m[f]比后面的大了，所以交换

}

return f; //返回该数字的下标（由于前后重合了，所以f=l

}