**1、冒泡排序（Bubble Sort）‌：通过比较相邻元素并交换它们的位置，将较大的元素“浮”到序列的末尾，直到整个序列排序完成。时间复杂度为O(n^2)。**

#include <iostream>

using namespace std;

const int MAXN = 10001;

float temp, a[MAXN] = {0, 49, 38, 65, 97, 76, 13, 27, 49};

int n, k;

void insert\_sort() {

int k;

for (int i = 1; i <= n; i++) { //i控制当前序列中最小值存放的数据位置

k = i;

for (int j = i + 1; j <= n; j++) //在当前无序区a[i..n]中选最小的元素a[k]

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

k = j;

if (k != i) { //交换a[i]和a[k]，将当前最小值放到a[i]位置

temp = a[i];

a[i] = a[k];

a[k] = temp;

}

}

}

void pop\_sort() {b

bool ok;

for (int i = n; i > 1; i--) { //进行n-1轮冒泡

ok=true;

for (int j = 1; j < i; j++) { //每轮进行i次的比较

if (a[j] > a[j + 1]){ //相邻两个元素比较，若逆序就交换

swap(a[j], a[j + 1]); //交换

ok=false;

}

}

if (ok==true) break;

}

}

int main() {

n = 8;

pop\_sort();

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

cout << a[i] << " ";

return 0;

}

**‌‌2、选择排序（Selection Sort）‌：每次从未排序的元素中选择最小（或最大）的元素，将其放置到已排序序列的末尾，直到所有元素排序完成。时间复杂度为O(n^2)。**

**‌‌**初 始 关键字 [49 38 65 97 76 13 27 49]  
第一趟排序后 13［38 65 97 76 49 27 49］  
第二趟排序后 13 27［65 97 76 49 38 49］  
第三趟排序后 13 27 38 [97 76 49 65 49]  
第四趟排序后 13 27 38 49 [76 97 65 49]  
第五趟排序后 13 27 38 49 49 [97 65 76]  
第六趟排序后 13 27 38 49 49 65 [97 76]  
第七趟排序后 13 27 38 49 49 65 76 [97]  
最后排序结果 13 27 38 49 49 65 76 97

排序过程，具体实现步骤如下：

①读入数据存放在a数组中。

②在a[1]～a[n]中选择值最小的元素，与第1位置元素交换，则把最小值元素放入a[1]中。

③在a[2]～a[n]中选择值最小的元素，与第2位置元素交换，则把最小值元素放入a[2]中，……

④直到第n-1个元素与第n个元素比较排序为止。

程序实现方法：用两层循环完成算法，外层循环i控制当前序列最小值存放的数组位置，内层循环j控制从i+1到n序列中选择最小的元素所在位置k。

#include <iostream>

using namespace std;

const int MAXN = 10001;

float temp, a[MAXN] = {0, 49, 38, 65, 97, 76, 13, 27, 49};

int n, k;

void select\_sort() {

int k;

for (int i = 1; i <= n; i++) { //i控制当前序列中最小值存放的数据位置

k = i;

for (int j = i + 1; j <= n; j++) //在当前无序区a[i..n]中选最小的元素a[k]

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

k = j;

if (k != i) { //交换a[i]和a[k]，将当前最小值放到a[i]位置

temp = a[i];

a[i] = a[k];

a[k] = temp;

}

}

}

int main() {

n = 8;

select\_sort();

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

cout << a[i] << " ";

return 0;

}

**3、插入排序（Insertion Sort）‌：将元素逐个插入到已排序的序列中，通过比较和移动元素来找到正确的位置。时间复杂度为O(n^2)。**

插入排序思想：回忆一下打牌时抓牌的情景，为了方便打牌，抓牌时一般一边抓牌一边按花色和大小插入恰当的位置，当抓完所有的牌时，手中的牌便是有序的，这排序方法即插入排序。

当读入一个元素时，在已经排序好的序列中，搜寻它正确的位置，再放入读入的元素。但不该忽略一个重要的问题：在插入这个元素前，应当先将将它后面的所有元素后移一位，以保证插入位置的原元素不被覆盖。

例如:设n=8，数组a中8个元素是: 36,25,48,12,65,43,20,58，执行插入排序程序后，其数据变动情况：

第0步:[36] 25 48 12 65 43 20 58  
第1步:[25 36] 48 12 65 43 20 58  
第2步:[25 36 48] 12 65 43 20 58  
第3步:[12 25 36 48] 65 43 20 58  
第4步:[12 25 36 48 65] 43 20 58  
第5步:[12 25 36 43 48 65] 20 58  
第6步:[12 20 25 36 43 48 65] 58  
第7步:[12 20 25 36 43 48 58 65]

#include <iostream>

using namespace std;

const int MAXN = 10001;

float temp, a[MAXN] = {0, 49, 38, 65, 97, 76, 13, 27, 49};

int n, k;

void insert\_sort() {

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

int j;

for (j = i - 1; j >= 1; j--) //在前面有序区间中为a[i]找合适的插入位置

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

break; //找到比a[i]小的位置就退出,插入其后

if ( j != i - 1) {

temp = a[i]; //将比a[i]大的数据向后移

for (k = i - 1; k > j; k--)

a[k + 1] = a[k]; //将a[i]放在正确位置上

a[k + 1] = temp;

}

}

}

int main() {

n = 8;

insert\_sort();

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

cout << a[i] << " ";

return 0;

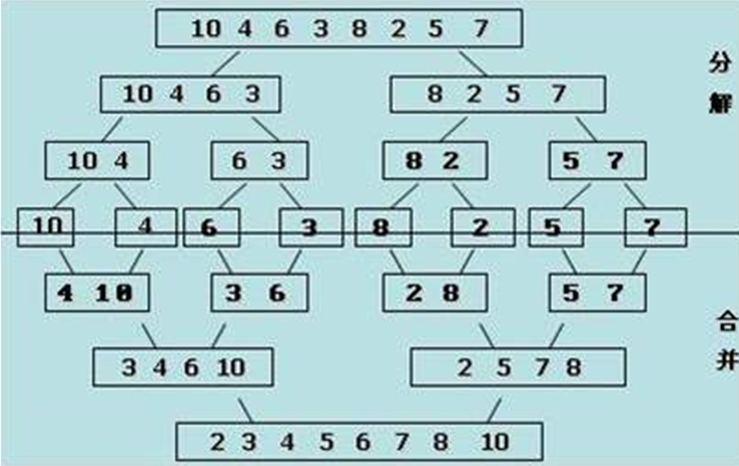
}

**‌‌‌‌4、归并排序（Merge Sort）‌：将序列分成两半，分别排序后再合并。时间复杂度为O(nlogn)。**

归并排序是建立在归并操作上的一种有效的排序算法,该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为二路归并。

例如有8个数据需要排序：10 4 6 3 8 2 5 7

归并排序主要分两大步：分解、合并。



合并过程为：比较a[i]和a[j]的大小，若a[i]≤a[j]，则将第一个有序表中的元素a[i]复制到r[k]中，并令i和k分别加上1；否则将第二个有序表中的元素a[j]复制到r[k]中，并令j和k分别加上1，如此循环下去，直到其中一个有序表取完，然后再将另一个有序表中剩余的元素复制到r中从下标k到下标t的单元。归并排序的算法我们通常用递归实现，先把待排序区间[s,t]以中点二分，接着把左边子区间排序，再把右边子区间排序，最后把左区间和右区间用一次归并操作合并成有序的区间[s,t]。

#include <iostream>

using namespace std;

const int MAXN = 10001;

float temp, r[MAXN], a[MAXN] = {0, 49, 38, 65, 97, 76, 13, 27, 49};

int n, k;

void msort(int s, int t) {

if (s == t)

return; //如果只有一个数字则返回，无须排序

int mid = (s + t) / 2;

msort(s, mid); //分解左序列

msort(mid + 1, t); //分解右序列

int i = s, j = mid + 1, k = s; //接下来合并

while (i <= mid && j <= t) {

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

r[k] = a[i];

k++;

i++;

} else {

r[k] = a[j];

k++;

j++;

}

}

while (i <= mid) { //复制左边子序列剩余

r[k] = a[i];

k++;

i++;

}

while (j <= t) { //复制右边子序列剩余

r[k] = a[j];

k++;

j++;

}

for (int i = s; i <= t; i++)

a[i] = r[i];

return ;

}

int main() {

n = 8;

msort(1, n);

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

cout << a[i] << " ";

return 0;

}

归并排序的时间复杂度是O(nlogn)，速度快。同时，归并排序是稳定的排序。即相等的元素的顺序不会改变。如输入记录 1(1) 3(2) 2(3) 2(4) 5(5) (括号中是记录的关键字)时输出的 1(1) 2(3) 2(4) 3(2) 5(5) 中的2 和 2 是按输入的顺序。这对要排序数据包含多个信息而要按其中的某一个信息排序,要求其它信息尽量按输入的顺序排列时很重要，这也是它比快速排序优势的地方。

**‌‌5、快速排序（Quick Sort）‌：通过选择一个“基准”元素，将序列分成两部分，一部分包含比基准小的元素，另一部分包含比基准大的元素，然后递归地对这两部分进行排序。时间复杂度为O(nlogn)。**

#include <iostream>

using namespace std;

const int MAXN = 10001;

float temp, a[MAXN] = {0, 49, 38, 65, 97, 76, 13, 27, 49};

int n, k;

void quick\_sort(int l, int r) {

int i, j, mid, p;

i = l;

j = r;

mid = a[(l + r) / 2]; //将当前序列在中间位置的数定义为分隔数

do {

while (a[i] < mid)

i++; //在左半部分寻找比中间数大的数

while (a[j] > mid)

j--; //在右半部分寻找比中间数小的数

if (i <= j) {

//若找到一组与排序目标不一致的数对则交换它们

p = a[i];

a[i] = a[j];

a[j] = p;

i++;

j--; //继续找

}

} while (i <= j); //注意这里不能少了等号

if (l < j)

quick\_sort(l, j); //若未到两个数的边界，则递归搜索左右区间

if (i < r)

quick\_sort(i, r);

}

int main() {

n = 8;

quick\_sort(1, n);

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

cout << a[i] << " ";

return 0;

}