sort()函数是 C++中的排序函数其头文件为：#include<algorithm>头文件；
qsort()是 C 中的排序函数，其头文件为：#include<stdlib.h>

1、qsort()----六类 qsort 排序方法

qsort 函数很好用，但有时不太会用比如按结构体一级排序、二级排序、字符串排序等。
函数原型：
void qsort(void *base, size_t nelem, size_t width, int (*fcmp)(const void*,const void *))
输入参数：
Base：待排序的数组
nelem：数组元数的个数（长度）
width：每一个元素所占存储空间的大小
fcmp：用于对数组元素进行比较的函数的指针（该函数是要自己写的），返回值为 1 或-1（p1>p2 则返回-1，p1<p2 则返回 1，p1==p2 则返回 0），size_t 是 int
输出参数：base 以升序排列
以下是其具体分类及用法(若无具体说明是以降序排列)：
（1）对一维数组排序：
（Element_type 是一位数组中存放的数据类型，可以是 char,int,float,double,ect）

```
int comp(const void *p1,const void *p2)
{
        return  *((Element_type*)p2)>*((Element_type*)p1)?1:-1;
}
int main()
{
        Element_type list[MAX];
        initial(list);//这是对数组 list[max]初始化
        qsort(list, sizeof(list),sizeof(Element_type),Comp);//调用函数 qsort
        return 0;
}
```

（2）对字符串排序：

```
int Comp(const void *p1,const void *p2)
{
        return strcmp((char *)p2,(char *)p1);
}
int main()
{
        char a[MAX1][MAX2];
        initial(a);
        qsort(a,lenth,sizeof(a[0]),Comp);
        //lenth 为数组 a 的长度
}
```

（3）按结构体中某个关键字排序（对结构体一级排序）：

```
struct Node
{
        double data;
```

```
	int other;
}s[100];
int Comp(const void *p1,const void *p2)
{
	return (*(Node *)p2)->data > (*（Node *）p1)->data ? 1 : -1;
}
qsort(s,100,sizeof(s[0]),Comp);
```

（4）按结构体中多个关键字排序（对结构体多级排序）[以二级为例]：

```
struct Node
{
	int x;
	int y;
}s[100];
//按照 x 从小到大排序，当 x 相等时按 y 从大到小排序（这是 3 跟 4 的区别）
int Comp(const void *p1,const void *p2)
{
	struct Node *c=(Node *)p1;
	struct Node *d=(Node *)p2;
	if(c->x!=d->x)
		return c->x-d->x;
	else
		return d->y - c->y;
}
```

（5）对结构体中字符串进行排序：

```
struct Node
{
	int data;
	char str[100];
}s[100];
//按照结构体中字符串 str 的字典序排序
int Comp(const void *p1,const void *p2)
{
	return strcmp((*(Node *)p1).str,(*(Node *)p2).str);
}
qsort(s,100,sizeof(s[0],Comp);
```

（6）计算几何中求凸包的 Comp

```
int Comp(const void *p1,const void *p2)//重点 Comp 函数，把除了 1 点外的所有的点旋转角度排序
{
	struct point *c=(point *)p1;
	struct point *d=(point *)p2;
	if( cacl(*c, *d,p[1])<0)
		return 1;
	else		if(!cacl(*c,		*d,		p[1])		&&
```

```
dis(c->x,c->y,p[1].x,p[1].y)<dis(d->x,d->y,p[1].x,p[1].y ) )
        //如果在一条直线上，则把远的放在前面
                return 1;
        else
                return -1;
}
```

2、sort()

sort 对给定区间所有元素进行排序
stable_sort 对给定区间所有元素进行稳定排序
partial_sort 对给定区间所有元素部分排序
partial_sort_copy 对给定区间复制并排序
nth_element 找出给定区间的某个位置对应的元素
is_sorted 判断一个区间是否已经排好序
partition 使得符合某个条件的元素放在前面
stable_partition 相对稳定的使得符合某个条件的元素放在前面
语法描述为：
（1）sort(begin,end)，表示一个范围，例如：

```
int _tmain(int argc, _TCHAR* argv[])
{
 int a[20]={2,4,1,23,5,76,0,43,24,65},i;
 for(i=0;i<20;i++)
  cout<<a[i]<<endl;
 sort(a,a+20);
 for(i=0;i<20;i++)
 cout<<a[i]<<endl;
 return 0;
}
```

输出结果将是把数组 a 按升序排序，说到这里可能就有人会问怎么样用它降序排列呢？这就是下一个讨论的内容。
（2）sort(begin,end,compare)
一种是自己编写一个比较函数来实现，接着调用三个参数的 sort：sort(begin,end,compare)就成了。对于 list 容器，这个方法也适用，把 compare 作为 sort 的参数就可以了，即：sort(compare)。
1）自己编写 compare 函数：

```
bool compare(int a,int b)
{
   return a<b; //升序排列，如果改为 return a>b，则为降序
}
int _tmain(int argc, _TCHAR* argv[])
{
   int a[20]={2,4,1,23,5,76,0,43,24,65},i;
   for(i=0;i<20;i++)
   cout<<a[i]<<endl;
   sort(a,a+20,compare);
```

```
    for(i=0;i<20;i++)
    cout<<a[i]<<endl;
    return 0;
}
```

2）更进一步，让这种操作更加能适应变化。也就是说，能给比较函数一个参数，用来指示是按升序还是按降序排,这回轮到函数对象出场了。

为了描述方便，我先定义一个枚举类型 EnumComp 用来表示升序和降序。很简单：

```
enum Enumcomp{ASC,DESC};
```

然后开始用一个类来描述这个函数对象。它会根据它的参数来决定是采用“<”还是“>”。

```
class compare
{
    private:
    Enumcomp comp;
    public:
    compare(Enumcomp c):comp(c) {};
    bool operator () (int num1,int num2)
    {
    switch(comp)
    {
    case ASC:
    return num1<num2;
    case DESC:
    return num1>num2;
    }
    }
};
```

接下来使用 sort(begin,end,compare(ASC))实现升序，sort(begin,end,compare(DESC))实现降序。

主函数为：

```
int main()
{
    int a[20]={2,4,1,23,5,76,0,43,24,65},i;
    for(i=0;i<20;i++)
    cout<<a[i]<<endl;
    sort(a,a+20,compare(DESC));
    for(i=0;i<20;i++)
    cout<<a[i]<<endl;
    return 0;
}
```

3)其实对于这么简单的任务（类型支持“<”、“>”等比较运算符），完全没必要自己写一个类出来。标准库里已经有现成的了，就在 functional 里，include 进来就行了。functional 提供了一堆基于模板的比较函数对象。它们是（看名字就知道意思了）：equal_to<Type>、not_equal_to<Type>、greater<Type>、greater_equal<Type>、less<Type>、less_equal<Type>。

对于这个问题来说，greater 和 less 就足够了，直接拿过来用：

升序：sort(begin,end,less<data-type>());
降序：sort(begin,end,greater<data-type>()).

```
int _tmain(int argc, _TCHAR* argv[])
{
   int a[20]={2,4,1,23,5,76,0,43,24,65},i;
   for(i=0;i<20;i++)
   cout<<a[i]<<endl;
   sort(a,a+20,greater<int>());
   for(i=0;i<20;i++)
   cout<<a[i]<<endl;
   return 0;
}
```

4)既然有迭代器，如果是 string 就可以使用反向迭代器来完成逆序排列，程序如下：

```
int main()
{
   string str("cvicses");
   string s(str.rbegin(),str.rend());
   cout << s <<endl;
   return 0;
}
```

这是我在百度上找到的 1011 题的答案，我觉得用它来说明 sort()函数最具有代表性

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <functional>
using namespace std;
int stick[100], n;
bool used[100];

//unused:没有使用的棍子的数目
//left:剩下的长度
//len:当前认为的计算的长度
bool dfs(int unused, int left, int len)
{
        // 所有的棍子已经用了，且没有剩余的长度,符合搜索条件
        if (unused == 0 && left == 0)
        return true;
        int i;
        //没有剩下的.则新开一条棍子
```

```
        if (left == 0)
        left = len;
        //寻找没有使用过的棍子
        for (i=0; i<n; ++i)
        {
                //找到没有用过的,而且长度比 left 值要小(能够填进去)
                if (!used && stick<=left)
                {
                        //使用当前棍子
                        used = true;
                        //若在当前情况下能够扩展出正确答案,则返回
                        if (dfs(unused-1, left-stick, len))
                                //成功搜索,返回
                                return true;
                        //否则不使用当前的棍子
                        used = false;
                        //若使用 stick 不能扩展出正确结果,那么如果 stick 与 left 等长,则
证明 len 不可能是正确答案
                        //若 left 与 len 等长,就是没有办法扩展
                        if (stick == left || left == len)
                                break;
                }
        }
        //经过一轮搜索仍得不到正确答案,则返回 false
        return false;
}
int main()
{
        int i, sum;
        while (scanf("%d", &n) != EOF && n)
        {
                sum = 0;
                for (i=0; i<n; ++i)
                {
                        scanf("%d", &stick);
                        used = false;
                        sum += stick;
                }
                //先进行从大到小排序
                sort(stick, stick+n, greater<int>());
                //根据题目条件，从小向大寻找
                for (i=stick[0]; i<=sum; ++i)
                {
                        //棍子总长被 i 整除才进行搜索,否则没用
```

```
                        if (sum % i == 0)
                        {
                                if (dfs(n, 0, i))
                                {
                                        printf("%d\n", i);
                                        break;
                                }
                        }
                }
        }
        return 0;
}
```