

实验一	统计数字问题
实验二	最大间隙问题
实验三	众数问题
实验四	半数集问题
实验五	集合划分问题
实验六	最少硬币问题
实验七	编辑距离问题
实验八	程序存储问题
实验九	最优效劳次序问题
实验十	汽车加油问题
实验十一	工作分配问题
实验十二	0-1 背包问题
实验十三	最小重量机器设计问题
实验十四	最小权顶点覆盖问题
实验十五	集合相等问题
实验十六	战车问题

实验一 统计数字问题

1、问题描述：

一本书的页码从自然数 1 开始顺序编码直到自然数 n 。书的页码按照通常的习惯编排，每个页码都不含多余的前导数字 0。例如，第 6 页用数字 6 表示，而不是 06 或 006 等。数字计数问题要求对给定书的总页码 n ，计算出书的全部页码中分别用到多少次数字 0，1，2，...，9。

2、题目分析：

考虑由 $0,1,2,\dots,9$ 组成的所有 n 位数。从 n 个 0 到 n 个 9 共有个 n 位数，在这些 n 位数中， $0,1,2,\dots,9$ 每个数字使用次数相同，设为。
满足如下递归式：

由此可知，。

据此，可从低位向高位进行统计，再减去多余的 0 的个数即可。

3、算法设计：

定义数组 $a[10]$ 存放 0 到 9 这 10 个数出现的次数，个位为第 0 位，第 j 位的数字为 r 。采用 `while` 循环从低位向高位统计：

- a. 统计从个位算起前 j 位 $0\sim 9$ 个数；
- b. 如果 $j+1$ 位为 0 ，去掉第 $j+1$ 位补 0 个数；
- c. 统计第 $j+1$ 位出现 $1\sim(r-1)$ 个数；
- d. 统计第 $j+1$ 位出现 r 个数。

4、源程序：

```
#include <iostream.h>
```

```
int main()
```

```
{
```

```
    long int sn[10];
```

```
    int i,n,c,k,s,pown;
```

```
    for(i=0;i<10;i++)
```

```
        sn[i]=0;
```

```

cin>>n;

for(k=s=0,pown=1;n>0;k++,n/=10,pown*=10)
{
    c=n%10;
    for(i=0;i<10;i++)
        sn[i]+=c*k*(pown/10);
    for(i=0;i<c;i++)
        sn[i]+=pown;
    sn[c]+=1+s;
    sn[0]-=pown;
    s+=c*pown;
}
for(i=0;i<10;i++)
    cout<<sn[i]<<'\\n';
}

```

5、算法分析：

函数 count()的复杂度为 $O(1)$ ，主函数调用 count()，故该算法的时间复杂度为 $O(1)$ 。

实验二 最大间隙问题

1、问题描述：

最大间隙问题：给定 n 个实数 $x_1, x_2, \dots, x_n$, 求这 n

个数在实轴上相邻 2 个数之间的最大差值。假设对任何实数的下取整函数耗时 $O(1)$ ，设计解最大间隙问题的线性时间算法。对于给定的 n 个实数 $x_1, x_2, \dots, x_n$ ，编程计算它们的最大间隙。

2、题目分析：

考虑到实数在实轴上按大小顺序排列，先对这 n 个数排序，再用后面的数减去前面的数，即可求出相邻两数的差值，找出差值中最大的即为最大差值。

3、算法设计：

a. 用快速排序算法对这 n 个数排序，快速排序算法是基于分治策略的一个排序算

法。其根本思想是，对于输入的子数组 $a[p:r]$ ，按以下三个步骤进行排序：

①分解：以 $a[p]$ 为基准元素将 $a[p:r]$ 划分为 3 段 $a[p:q-1]$ ， $a[q]$ 和 $a[q+1:r]$ ，使 $a[p:q-1]$ 中任何一个元素小于等于 $a[p]$ ，而 $a[q+1:r]$ 中任何一个元素大于等于 $a[q]$ 。下标 q 在划分过程中确定。

②递归求解：通过递归调用快速排序算法分别对 $a[p:q-1]$ 和 $a[q+1:r]$ 进行排序。

③合并：由于对 $a[p:q-1]$ 和 $a[q+1:r]$ 的排序是就地进行的，所以在 $a[p:q-1]$ 和

$a[q+1:r]$ 都已排好的序后，不需要执行任何计算， $a[p:r]$ 就已排好序。

b. 用函数 `maxtap()` 求出最大差值。

4、源程序：

```
#include<iostream.h>
```

```
#include<stdio.h>
```

```
double a[1000000];
```

```
template<class Type>
```

```
void swap(Type &x,Type &y)
```

```
{
```

```
    Type temp=x;
```

```
    x=y;
```

```
    y=temp;
```

```
}
```

```
template<class Type>
```

```
int Partition(Type *a,int low,int high)
```

```
{
```

```
    Type pivotkey;
```

```
    int  mid=(low+high)/2;
```

```
    if((a[low]<a[mid]&&a[mid]<a[high])||(a[low]>a[mid]&&a[mid]>a[hi  
gh]))
```

```
        swap(a[low],a[mid]);
```

```
    if((a[low]<a[high]&&a[mid]>a[high])||(a[low]>a[high]&&a[mid]<a[high]))
```

```
        swap(a[low],a[high]);
```

```
    pivotkey=a[low];
```

```
    int i=low;
```

```
    int j=high+1;
```

```
    while(true)
```

```
    {
```

```
        while(a[++i]<pivotkey);
```

```
        while(a[--j]>pivotkey);
```

```
        if(i>=j) break;
```

```
        swap(a[i],a[j]);
```

```
    }
```

```
    a[low]=a[j];
```

```
    a[j]=pivotkey;
```

```
    return j;
```

```
}
```

```
template<class Type>
```

```
void Quicksort(Type *a,int low,int high)
```

```
{
```

```
    if(low<high)
```

```
    {
```

```

        int q=Partition(a,low,high);
        Quicksort(a,low,q-1);
        Quicksort(a,q+1,high);
    }
}

```

```

template<class Type>
Type maxtap(Type *a,int n)
{
    Type maxtap=0;
    for(int i=0;i<n-1;i++)
        maxtap=(a[i+1]-a[i])>maxtap?(a[i+1]-a[i]):maxtap;
    return maxtap;
}

```

```

main()
{
    int i,n;
    scanf("%d",&n);
    for(i=0;i<n;i++)
        scanf("%lf",&a[i]);
    Quicksort(a,0,n-1);
    printf("%lf",maxtap(a,n));
}

```

```
return 0;
```

5、算法分析：

快速排序的运行时间与划分是否对称有关，其最坏情况发生在划分过程产生的两个区域分别包含 $n-1$ 个元素和 1 个元素的时候。由于函数 Partition 的计算时间为 $O(n)$ ，所以如果算法 Partition 的每一步都出现这种不对称划分，那么其计算时间复杂性 $T(n)$ 满足：

解此递归方程可得。

在最好情况下，每次划分所取得基准都恰好为中值，即每次划分都产生两个大小为 $n/2$ 的区域，此时，Partition 的计算时间 $T(n)$ 满足：

其解为。

可以证明，快速排序算法在平均情况下的时间复杂性也是。

实验三 众数问题

1、问题描述：

给定含有 n 个元素的多重集合 S ，每个元素在 S 中出现的次数称为该元素的重数。多重集 S 中重数最大的元素称为众数。例如， $S=\{1, 2, 2, 2, 3, 5\}$ 。多重集 S 的众数是 2，其重数为 3。对于给定的由 n 个自然数组成的多重集 S ，编程计算 S 的众数及其重数。

2、题目分析：

题目要求计算集合中重复出现最多的数以及该数出现的次数，假设两个数出现的次数相同，那么取较小的数。先对集合中的元素从小到大排序，再统计重数，找出最大的重数以及它对应的元素。

3、算法设计：

- a. 建立数组 $a[n]$ 存储集合中的元素；
- b. 调用库函数 $\text{sort}(a, a+n)$ 对集合中的元素从小到大排序；
- c. 采用 for 循环依次对排序后的元素进行重数统计：

①用 m 标记元素，用 s 统计该元素的重数，分别用 max 和 t 标记出现的最大重数和该重数对应的元素。

②假设当前元素不同于前一个元素，且前一个元素的重数 $s > \text{max}$ ，更新 max 和 t 。

4、源程序：

```
#include<iostream>

using namespace std;

#include<algorithm>

main()
{
```

```

int?n,i,t,m=0,s,max=0,*a;
scanf("%d",&n);
a=new?int[n];
for(i=0;i<n;i++)
    scanf("%d",&a[i]);
sort(a,a+n);
for(i=0;i<n;i++)
{
    if(m!=a[i])
        s=1;
    else
        s++;
    m=a[i];
    if(s>max)
    {
        t=m;
        max=s;
    }
}
printf("%d\n%d\n",t,max);
return?0;
}

```

5、算法分析：

主函数内调用的库函数 `sort(a,a+n)` 的时间复杂度为，for 循环的时间复杂度为 $O(n)$ ，故该算法的时间复杂度为。

实验四 半数集问题

1、问题描述：

给定一个自然数 n ，由 n 开始可以依次产生半数集 $set(n)$ 中的数如下：(1) $n \in set(n)$ ；(2) 在 n 的左边加上一个自然数，但该自然数不能超过最近添加的数的一半；(3) 按此规则进行处理，直到不能再添加自然数为止。

例如： $set(6) = \{6, 16, 26, 126, 36, 136\}$ ，半数集 $set(6)$ 中有 6 个元素。

注意半数集

是多重集，对于给定的自然数 n ，编程计算半数集 $set(n)$ 中的元素个数。

2、题目分析：

题目要求输入有多行，每行给出一个整数 n ($0 < n < 1000$)，输入以文件结束标志 EOF 结束。半数集 $set(n)$ 中的元素个数即为所有半数集 $set(j)$ ($j \leq n/2$) 的元素个数之和加 1（即将其自身添加到所求元素中）。

3、算法设计：

- a. 用数组 `count[i]` 计算 `set(i)` 中的元素个数, 即计算所有 $j \leq i/2$ 的 `set(j)` 中的元素加其自身的个数之和;
- b. 用数组 `count_sum[i]` 计算所有 $j \leq i$ 的 `set(j)` 中的元素个数之和;
- c. 采用 for 循环分别对 `count[i]`、`count_sum[i]` 进行累加, 即可求出半数集 `set(n)` 中的元素个数。

4、源程序:

```
#include <stdio.h>

int set(int n)
{
    int i, count[1001], count_sum[1001];
    count[1]=1;
    count_sum[1]=1;
    for(i=2; i<=n; i++)
    {
        count[i]=count_sum[i/2]+1;
        count_sum[i]=count_sum[i-1]+count[i];
    }
    return count[n];
}
```

```
}
```

```
main()
```

```
{
```

```
    int n;
```

```
    while(scanf("%d",&n)!=EOF)
```

```
        printf("%d\n",set(n));
```

```
    return 0;
```

```
}
```

5、算法分析：

函数 $\text{set}(n)$ 的时间复杂度为 $O(n)$ ，主函数调用函数 $\text{set}(n)$ ，故该算法的时间复杂度为 $O(n)$ 。

实验五 集合划分问题

2、题目分析：

题目要求计算 n 个元素的集合共有多少个划分（其中每个划分都由不同的非空子集组成）， n 个元素的集合划分为 m 个块的划分数为 $F(n,m)=F(n-1,m-1)+m \cdot F(n-1,m)$ ， m 从 1 到 n 的划分个数相加即可求得总的划分数。

3、算法设计：

- a. 这一题的结果数很大，需要使用 64 位长整型：__int64;
- b. 函数 div()采用递归的方式计算 n 个元素的集合划分为 i 个块的划分数：

$$\textcircled{1} \text{div}(n,1)=1, \text{div}(n,n)=1;$$

$$\textcircled{2} \text{div}(n,i)=\text{div}(n-1,i-1)+i*\text{div}(n-1,i)$$

- c. 主函数采用 for 循环调用函数 div(n,i) ($1 \leq i \leq n$) 对划分数进行累加统计。

4、源程序：

```
#include<stdio.h>
```

```
__int64?div(__int64?n,__int64?i)
{
    if(i==1||i==n)
        return?1;
    else?return?div(n-1,i-1)+i*div(n-1,i);
}
```

```
main()
{
    __int64?i,n,s=0;
    scanf("%I64d",&n);
```

```

for(i=1;i<=n;i++)
    s=s+div(n,i);
printf("%I64d",s);
return 0;
}

```

5、算法分析：

函数 `div()` 的时间复杂度为，主函数内 `for` 循环的时间复杂度为 $O(n)$ ，函数 `div()` 嵌套在 `for` 循环内，故该算法的时间复杂度为。

实验七 编辑距离问题

1、问题描述：

设 A 和 B 是 2 个字符串。要用最少的字符操作将字符串 A 转换为字符串 B 。这

里所说的字符操作包括 (1)删除一个字符； (2)插入一个字符； (3)将一个字符改为另

一个字符。将字符串 A 变换为字符串 B 所用的最少字符操作数称为字符串 A 到 B 的编

辑距离，记为 $d(A,B)$ 。试设计一个有效算法，对任给的 2 个字符串 A 和 B ，计算出它

们的编辑距离 $d(A,B)$ 。

2、题目分析：

题目要求计算两字符串的编辑距离，可以采用动态规划算法求解，由最优子结构性质可建立递归关系如下：

其中数组 $d[i][j]$ 存储长度分别为 i 、 j 的两字符串的编辑距离；用 $edit$ 标记所比拟

的字符是否相同，相同为 0，不同为 1；用 m 、 n 存储字符串 a 、 b 的长度。

3、算法设计：

a. 函数 $min()$ 找出三个数中的最小值；

b. 函数 $d()$ 计算两字符串的编辑距离：

①用 $edit$ 标记所比拟的字符是否相同，相同为 0，不同为 1；

②分别用 m 、 n 存储字符串 a 、 b 的长度，用数组 $d[i][j]$ 存储长度分别为 i 、 j 的两字符串的编辑距离，问题的最优值记录于 $d[n][m]$ 中；

③利用递归式写出计算 $d[i][j]$ 的递归算法。

4、源程序：

```
#include <iostream>
```

```
using namespace std;
```

```
int min(int a, int b, int c)
```

```

{
    int temp = (a < b ? a : b);
    return (temp < c? temp : c);
}

```

```

int d(char* a, char* b)
{
    int m = strlen(a);
    int n = strlen(b);
    int i,j ,temp;
    int **d;
    d=(int  **)malloc(sizeof(int *)*(m+1));
    for(i=0;i<=m;i++)
    {
        d[i]=(int *)malloc(sizeof(int)*(n+1));
    }
    d[0][0]=0;
    for(i=1;i<=m;i++)
        d[i][0]=i;
    for(j=1;j<=n;j++)
        d[0][j]=j;
    for( i=1;i<=m;i++)
    {

```

```

        for(j=1;j<=n;j++)
        {
            int edit=( a[i-1] == b[j-1] ? 0 : 1);
            d[i][j]=min((d[i-1][j-1]+edit),
(d[i][j-1]+1),(d[i-1][j]+1));
        }
    }
    temp = d[m][n];
    free(d);

    return temp;
}

```

```

main()
{
    char a[10000],b[10000];
    scanf("%s\n%s",a,b);
    printf("%d",d(a,b));
    return 0;
}

```

5、算法分析：

函数 $d()$ 采用了双重循环，里层的 for 循环复杂度为 $O(n)$ ，外层的 for 循环复杂度为 $O(m)$ ，主函数调用了函数 $d()$ ，故该算法的时间复杂度为 $O(mn)$ 。

实验八 程序存储问题

1、问题描述：

设有 n 个程序 $\{1, 2, \dots, n\}$ 要存放在长度为 L 的磁带上。程序 i 存放在磁带上的长度是 i_1 ， $1 \leq i \leq n$ 。程序存储问题要求确定这 n 个程序在磁带上的一个存储方案，使得能够在磁带上存储尽可能多的程序。对于给定的 n 个程序存放在磁带上的长度，编程计算磁带上最多可以存储的程序数。

2、题目分析：

题目要求计算给定长度的磁带最多可存储的程序个数，先对程序的长度从小到大排序，再采用贪心算法求解。

3、算法设计：

- 定义数组 $a[n]$ 存储 n 个程序的长度， s 为磁带的长度；
- 调用库函数 $\text{sort}(a, a+n)$ 对程序的长度从小到大排序；
- 函数 $\text{most}()$ 计算磁带最多可存储的程序数，采用 while 循环依次对排序后的程序长度进行累加，用 i 计算程序个数，用 sum 计算程序累加长度（初

始 $i=0, \text{sum}=0$):

① $\text{sum}=\text{sum}+\text{a}[\text{i}]$;

② 假设 $\text{sum}\leq\text{s}$, i 加 1, 否那么 i 为所求;

③ $\text{i}=\text{n}$ 时循环结束;

d. 假设 while 循环结束时仍有 $\text{sum}\leq\text{s}$, 那么 n 为所求。

4、源程序:

```
#include<iostream>
```

```
using namespace std;
```

```
#include<algorithm>
```

```
int a[1000000];
```

```
int most(int *a,int n,int s)
```

```
{
```

```
    int i=0,sum=0;
```

```
    while(i<n)
```

```
    {
```

```
        sum=a[i]+sum;
```

```
        if(sum<=s)
```

```
            i++;
```

```
        else return i;
```

```
    }
```

```
    return n;
```

```
}
```

```
main()
```

```
{
```

```
    int i,n,s;
```

```
    scanf("%d %d\n",&n,&s);
```

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

```
        scanf("%d",&a[i]);
```

```
    sort(a,a+n);
```

```
    printf("%d",most(a,n,s));
```

```
    return 0;
```

```
}
```

5、算法分析：

函数 `most()` 的时间复杂度为 $O(n)$ ，主函数调用的库函数 `sort(a,a+n)` 的时间复杂度为 $O(n^2)$ ，且主函数调用函数 `most()`，故该算法的时间复杂度为 $O(n^2)$ 。

实验九 最优效劳次序问题

1、问题描述：

设有 n 个顾客同时等待一项效劳。顾客 i 需要的效劳时间为 t_i , $1 \leq i \leq n$ 。应如何安排 n 个顾客的效劳次序才能使平均等待时间到达最小？平均等待时间是 n 个顾客等待效劳时间的总和除以 n 。对于给定的 n 个顾客需要的效劳时间，编程计算最优效劳次序。

2、题目分析：

考虑到每个顾客需要的效劳时间已给定，要计算最优效劳次序，可以先对顾客需要的效劳时间由短到长排序，再采用贪心算法求解。

3、算法设计：

- a. 定义数组 $a[n]$ 存储 n 个顾客需要的效劳时间；
- b. 调用库函数 $\text{sort}(a, a+n)$ 对顾客需要的效劳时间由短到长排序；
- c. 函数 $\text{time}()$ 计算最小平均等待时间，采用 while 循环依次对顾客等待效劳时间进行累加，用 $a[i]$ 标记第 $i+1$ 个顾客需要的效劳时间，用 $b[i]$ 计算第 $i+1$ 个顾客的等待效劳时间，用 t 计算前 $i+1$ 个顾客等待效劳时间的总和（初始 $i=1$, $t=a[0], b[0]=a[0]$ ）：

① $b[i]=a[i]+b[i-1]$, $t=b[i]+t$;

② $i++$;

③ $i=n$ 时循环结束；

d. t/n 为所求。

4、源程序：

```
#include<iostream>
using namespace std;
```

```
#include<algorithm>

int a[1000000],b[1000000];

double time(int *a,int n)
{
    int i=1,j=0;
    double t=a[0];
    b[0]=a[0];
    while(i<n)
    {
        b[i]=a[i]+b[i-1];
        t=b[i]+t;
        i++;
    }
    return t/n;
}
```

```
main()
{
    int i,n;
    scanf("%d",&n);
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    sort(a,a+n);
```

```
printf("%0.2lf",time(a,n));  
return 0;  
}
```

5、算法分析：

函数 `time()` 的时间复杂度为 $O(n)$ ，主函数调用的库函数 `sort(a,a+n)` 的时间复杂度为 $O(n^2)$ ，且主函数调用函数 `time()`，故该算法的时间复杂度为 $O(n^2)$ 。

实验十 汽车加油问题

1、问题描述：

有一辆汽车加满油后可行驶 n 公里。旅途中有假设若干个加油站。设计一个有效算法，指出应在哪些加油站停靠加油，使沿途加油次数最少。并证明算法能产生一个最优解。对于给定的 n 和 k 个加油站位置，编程计算最少加油次数。

2、题目分析：

题目要求编程计算最少加油次数，假设无法到达目的地，那么输出“No Solution”。该题 可以采用贪心算法求解，从出发地开始进行判别：油足够那么继续行驶；油不够那么加油，计算加油次数；油满仍不够那么“No Solution”。

3、算法设计：

a. n 表示汽车加满油后可行驶 n 公里， k 表示出发地与目的地之间有 k 个加油站；

b. 定义数组 $a[k+1]$ 存储加油站之间的距离：用 $a[i]$ 标记第 i 个加油站与第 $i+1$ 个加

油站之间的距离（第 0 个加油站为出发地，汽车已加满油，第 $k+1$ 个加油站为目的地）；

c. 用 m 计算加油次数，用 t 标记在未加油的情况下汽车还能行驶 t 公里，采用 for

循环从出发地开始（即 $i=0$ ）依次计算加油次数：

① 假设 $a[i]>n$ ，那么输出“No Solution”；

② 假设 $t<a[i]$ ，那么加油一次： $t=n, m++$ ；

③ 行驶 $a[i]$ 公里后，汽车还能行驶 $t-a[i]$ 公里；

④ $i=k+1$ 时循环结束；

d. m 即为所求。

4、源程序：

```
#include<stdio.h>
```

```
main()
```

```
{
```

```
    int i,n,k,t,m,a[10000];
```

```
    scanf("%d%d",&n,&k);
```

```
    for(i=0;i<=k;i++)
```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：<https://d.book118.com/025221301000011312>