第一节：数据结构基本知识

1.1 什么是数据结构

例：写程序实现一个函数 PrintN，使得传入一个正整数位 N 的参数后，能顺序打印从 1 道 N 的全部正整数。

代码 1（循环实现）：

```
void PrintN(int N){
	int i;
	for(i =1; i<=N; i++)
	{
		printf("%d\n",i);
	}
	return;
}
```

代码 2（递归实现）：

```
void PrintN(int N){
	if(N){
		PrintN(N-1);
		printf("%d\n",N);
	}
	return;
}
```

结论：解决问题方法的效率，跟空间的利用率有关。

例 3：写程序计算给定多项式在给定点 x 处的值

代码：

```
double f(int n, double a[], double x)
{
	int i;
	double p = a[n];
	for(i=n;i>0;i--)
	{
		p = a[i-1]+x*p;
	}
	return p;
}
```

C 语言中提供了一个函数为 clock()，它捕捉从程序开始运行到 clock()被调用时所耗费的时间。这个时钟单位是 clock tick。常熟 CLK_TCK 为机器时钟每秒所周的 clock tick。例子如下：

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

```
clock_t start,stop;
/*clock_t 是 clock()函数返回的变量类型*/
double duration;

int main()
{
	start = clock();
	MyFunction();//所需要测量运行时间的函数
	stop = clock();
	duration = ((double)(stop-start)/CLK_TCK;
	return 0;
}
```

结论：解决问题方法的效率，跟算法的巧妙程度有关。

什么是数据结构?

数据对象在计算机中的组织方式。数据对象必定与一系列加在其上面的操作相关联，完成这项操作所用的方法就是算法。

抽象数据类型

抽象是指：描述数据类型的方法不依赖与具体事项，它：

①与存放数据的机器无关；②与数据存储的物理结构无关；③与实现操作的算法和编程语言均无关。

它只描述数据对象集和相关操作集“是什么”，并不涉及“如何做”的问题。

1.2 算法

1.什么是算法?

算法是：

（1）一个有限指令集；

（2）接受一些输入（有时候也没有输入）；

（3）产生输出；

（4）一定在**有限步骤**后终止；

（5）每一条指令必须：有充分明确的目标，不可以有歧义；计算机能够处理的范围之内；描述应**不依赖于任何一种计算机语言以及具体的实现**。

2.什么是好的算法?

在分析一般算法的效率时，我们经常关注一下两种复杂度：

（1）最坏情况复杂度 $T_{worst}(n)$；

（2）平均复杂度 $T_{avg}(n)$。

3 复杂度的渐进表示法

$T(n)=O(f(n))$表示存在常数 $C>0$，$n_0>0$ 使得当 $n\geqslant n_0$ 时有 $T(n)\leqslant Cf(n)$；

$T(n)=\Omega(f(n))$表示存在常数 $C>0$，$n_0>0$ 使得当 $n\geqslant n_0$ 时有 $T(n)\geqslant Cf(n)$；

$T(n)=\theta(h(n))$表示同时又 $T(n)= O(f(n))$和 $T(n)=\Omega(f(n))$。

不同复杂度函数的感性理解，如表 1.1 所示。

1.1 不同复杂度函数的规模

函数	1	2	4	8	16	32

1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n^2	*1*	*4*	*16*	*64*	*256*	*1024*
n^3	*1*	*8*	*64*	*512*	*4096*	*32768*
2^n	*1*	*4*	*16*	*256*	*65536*	*4294967296*
n!	*1*	*2*	*24*	*40326*	*2092278988000*	*26313×10^{33}*

如果复杂度为灰色斜体的部份，要想办法尽可能降到黑色部分。

复杂度分析小窍门:

（1）若两段算法分别有复杂度 $T_1(n)=O(f_1(n))$和 $T_2(n)=O(f_2(n))$，则

①两段代码的和 $T_1(n)+T_2(n)=max(O(f_1(n)), T_2(n)=O(f_2(n)))$

②两段代码嵌套起来 $T_1(n)\times T_2(n)=O(f_1(n)\times f_2(n))$。

（2）若 T(n)是关于 n 的 k 阶多项式，那么 $T(n)= \theta(n^k)$。

（3）一个 for 循环的时间复杂度等于循环次数乘循环体内代码的复杂度。

（4）if-else 结构的复杂度取决于 if 的条件判断复杂度和两个分枝部分的复杂度，总体复杂度取三者中最大。

1.3 应用实例

1.例 1

图 1

复杂度为 **$O(N^1)$**

算法 2：

```
int MaxSubseqSum2(int A[], int N)
{
	int ThisSum, MaxSum=0;
	int i ,j;
	for(i=0;i<N;i++)/*i 是子列左端的位置*/
	{
		ThisSum =0;/*ThisSum 是从 A[i]dao A[j]的子列和*/
		for(j=i; j<N;j++)
		{
			ThisSum+=A[j];
			/*对于相同的 i，不同的 j，只要把 j-1 此循环的基础上累加 1 项即可*/
			if(ThisSum>MaxSum)
			MaxSum=ThisSum;
		}
	}
	return MaxSum;
}
```

复杂度为 **$O(N^2)$**

算法 3：分而治之

先将这个数组分为两半，分别递归的解决两边的问题。在左边我们会得到一个左边的最大值，右边会得到一个右边的最大值，然后再求一个跨越边界的最大子列和。然后对其进行比较，寻找到最大值。

图 2

算法 4：在线处理

代码：

```
int MaxSubseqSum4(int A[], int N)
{
    int ThisSum, MaxSum;
    int i;
    ThisSum=MaxSum=0;
    for(i=0;i<N;i++){
        ThisSum +=A[i];/*向右累加*/
        if(ThisSum>MaxSum)
            MaxSum=ThisSum;/*发现更大和则更新当前结果*/
        else if(ThisSum<0)/*如果当前子列和为负*/
            ThisSum=0;/*则不可能使后面的部分和增大，抛弃之*/
    }
    return MaxSum;
}
```

复杂度为 **O(N)**

为了理解这个算法，举个例子来走一遍过程。假设一组数字为：

序号	0	1	2	3	4	5	6	7
取值	-1	3	-2	4	-6	1	6	-1

刚开始，我们就令 ThisSum 和 MaxSum 为 0。进入 for 循环。

i 取值	步骤	ThisSum	MaxSum
0	ThisSum 为-1，-1<0，故抛弃这个值	0	0
1	然后 ThisSum 加上 A[1]，由于之前 ThisSum 令为 0 了，所以现在值为3。然后ThisSum>MaxSum，所以把ThisSum赋值给 MaxSum，后者等于 3。	3	3
2	然后 ThisSum 加上 A[2]，当前和为+1，然后 MaxSum>ThisSum，所以不赋值，由于 ThisSum 为+1，故不抛弃。	+1	3
3	ThisSum 加上 A[3]，结果为 5，然后 ThisSum>MaxSum，所以把 ThisSum 赋值给 MaxSum，后者等于 5	+5	5
4	ThisSum 加上 A[4]，结果为-1。此时 ThisSum 小于 MaxSum，所以不改变 Max 的值。又由于 This＜0，故赋值为 0。	0	5
5	ThisSum 加上 A[5]，值为 1，此时 ThisSum 小于 MaxSum，所以不改变 Max 的值。	1	5
6	ThisSum 加上 A[6]，值为 7，此时 ThisSum 大于 MaxSum，所以把 ThisSum 赋值给 MaxSum，后者等于 7	7	7
7	ThisSum 加上 A[7]，值为 6，此时 ThisSum 小于 MaxSum，	6	7

	所以不改变 Max 的值。		

故最大值为 7。