复杂度

2019年6月25日 9:20

例2：输出n个正整数

代码：

```
#include<iostream>
#include<time.h>
using namespace std;

void PrintN1(int n)
{
    cout << "循环输出:";
    for (int i = 0; i <= n; i++)
        cout <<i;
    return;
}
void PrintN2(int n)
{
    cout << "递归输出:";
    if (n)
        PrintN2(n - 1);
        cout << n;
}
void main()
{
    int n;
    cin >> n;
    clock_t t;
    t = clock();
    PrintN1(n);
    t = clock() - t;
    cout <<"循环耗费的时间为："<< ((float)t) / CLOCKS_PER_SEC
    <<endl;
    t = clock();
    PrintN2(n);
    t = clock() - t;
    cout << "递归耗费的时间为：" << ((float)t) / CLOCKS_PER_SEC
    << endl;

}
```

结果：

5

循环输出:012345循环耗费的时间为：0.001

递归输出：012345递归耗费的时间为：0.002

10000

循环能成功，递归stakoverflow

例3：多项式计算

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

```
using namespace std;
#define MAXN 10

double f1(int n,double a[], double x)
{
	double p = a[0];
	for (int i = 1; i <= n; i++)
		p += a[i] * pow(x,i);
	return p;
}
double f2(int n, double a[],double x)//秦九韶算法提取x
{
	double p = a[n];
	for (int i = n; i >0; i--)
		p = a[i-1] + x * p;
	return p;
}

void main()
{
	int n,x;
	clock_t t1, t2;
	double duration;
	double a[MAXN];//设有9次方，x为1.1
	for (int i = 0; i < MAXN; i++)
		a[i] = (double)i;
	t1=clock();
	f1(MAXN - 1, a, 1.1);
	t1 = clock() - t1;
	duration=t1 / CLOCKS_PER_SEC;
	cout << "f1时长" << duration<<endl;
	t2 = clock();
	f2(MAXN - 1, a, 1.1);
	t2 = clock() - t2;
	duration = t2 / CLOCKS_PER_SEC;
	cout << "f2时长" << duration << endl;
	return;
}
```

结果:

f1时长0

f2时长0

怎么解决：重复多次调用函数

代码:

```
#include<iostream>
#include<time.h>
using namespace std;
#define MAXN 10
#define MAXK 1e7

double f1(int n,double a[], double x)
{
	double p = a[0];
	for (int i = 1; i <= n; i++)
		p += a[i] * pow(x,i);
```

```
    return p;
}
double f2(int n, double a[],double x)//秦九韶算法提取x
{
    double p = a[n];
    for (int i = n; i >0; i--)
        p = a[i-1] + x * p;
    return p;
}

void main()
{
    int n,x;
    clock_t t1, t2;
    double duration1,duration2;
    double a[MAXN];//设有9次方，x为1.1
    for (int i = 0; i < MAXN; i++)
        a[i] = (double)i;
    t1=clock();
    for(int i=0;i<MAXK;i++)
    {
        f1(MAXN - 1, a, 1.1);
    }
    t1 = clock() - t1;
    duration1=(double)t1 / CLOCKS_PER_SEC/MAXK;
    cout << "f1时长" << duration1<<endl;
    t2 = clock();
    for (int i = 0; i < MAXK; i++)
    {
        f2(MAXN - 1, a, 1.1);
    }
    t2 = clock() - t2;
    duration2 = (double)t2 / CLOCKS_PER_SEC / MAXK;
    cout << "f2时长" << duration2 << endl;
    return;
}
```

结果：

f1时长6.698e-07

f2时长5.93e-08

什么是算法

例1：

选择排序：从未排序部分找出最小元插入已排序的最后位置

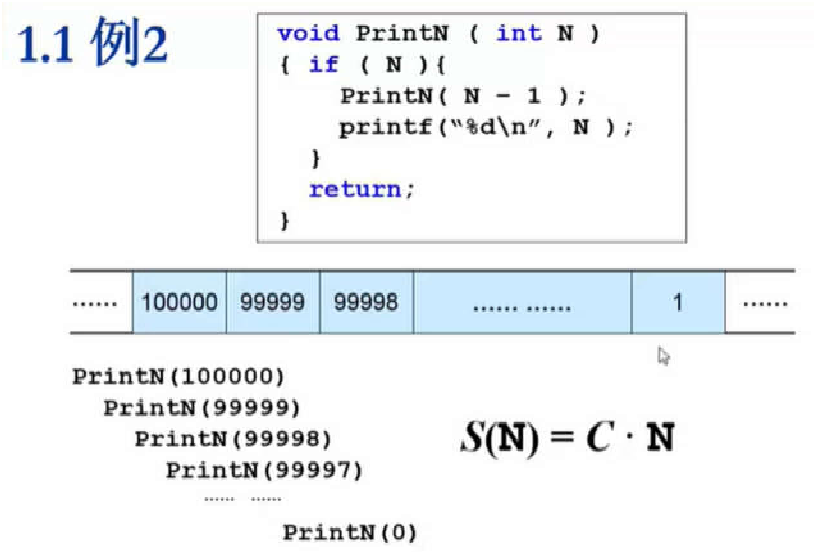
```
void selectionSort(int List[],int N)//递减排序
{
  for(i=0;i<N;i++)
    MinPosition=ScanForMin(List,I,N-1);//找出[i]到[N-1]中最小元，将其赋值给MinPosition
    Swap(List[i],List[MinPosition]);//将未排序部分的最小元换到有序部分的最后位置
}
```

衡量算法的好坏：

1. 空间复杂度S(n)--占用储存空间的长度
2. 时间复杂度T(n)--耗费时间
3. 最坏复杂度Tworst（n）
4. 平均复杂度Tavg（n）

例2：递归输出N

```
void PirntN(int n)
{
  if(n)
      PrintN(n-1);
      cout<<n;
  return;
}
```



例3：

f1 T(n)=C1*n^2+C2*n

乘法总次数(1+2+...+N)=(n^2+n)/2

```
for(i=1;i<=n;i++)
  p+=(a[i]*pow(x*i));
```

f2 T(n)=C*n

```
for(i=n;i>0;i--)
  p=a[n-1]+x*p;
```

复杂度的渐进表示法

- 上界：T(n) = O(f(n)) 表示存在常数C >0, n 0>0 使得当 n · n 0 时有 T(n) · C·f(n)
- 下界：T(n) = Ω (g (n)) 表示存在常数C >0, n 0>0 使得当 n · n 0 时有 T(n) · C·g (n)
- T(n) = Θ (h (n)) 表示同时有 T(n) = O(h (n)) 和 T(n) = Ω (h (n))

复杂度分析小窍门 ·

若两段算法分别有复杂度 $T1(n) = O(f1(n))$ 和 $T2(n) = O(f2(n))$，则

$T1(n) + T2(n) = \max(O(f1(n)), O(f2(n)))$

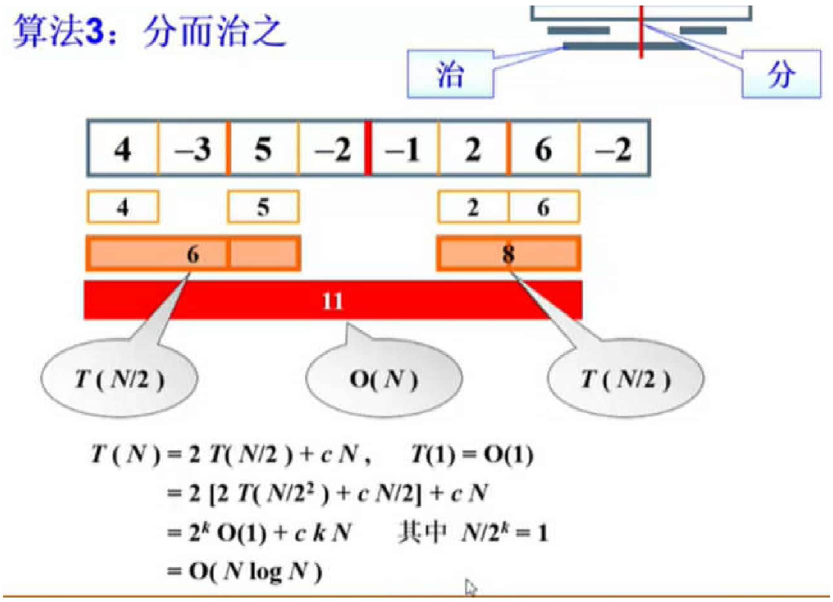
$T1(n) \cdot T2(n) = O(f1(n) \cdot f2(n))$

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

一个for循环的时间复杂度等于循环次数乘以循环体 代码的复杂度

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

算法改进：分而治之



代码：

```
int Max3(int A, int B, int C)
{ /* 返回3个整数中的最大值 */
    return A > B ? A > C ? A : C : B > C ? B : C;
}

int DivideAndConquer(int List[], int left, int right)
{ /* 分治法求List[left]到List[right]的最大子列和 */
    int MaxLeftSum, MaxRightSum; /* 存放左右子问题的解 */
    int MaxLeftBorderSum, MaxRightBorderSum; /*存放跨分界线的结果*/

    int LeftBorderSum, RightBorderSum;
    int center, i;

    if (left == right) { /* 递归的终止条件，子列只有1个数字 */
        if (List[left] > 0)  return List[left];
        else return 0;
    }

    /* 下面是"分"的过程 */
    center = (left + right) / 2; /* 找到中分点 */
    /* 递归求得两边子列的最大和 */
    MaxLeftSum = DivideAndConquer(List, left, center);
    MaxRightSum = DivideAndConquer(List, center + 1, right);
```

```
    /* 下面求跨分界线的最大子列和 */
    MaxLeftBorderSum = 0; LeftBorderSum = 0;
    for (i = center; i >= left; i--) { /* 从中线向左扫描 */
        LeftBorderSum += List[i];
        if (LeftBorderSum > MaxLeftBorderSum)
            MaxLeftBorderSum = LeftBorderSum;
    } /* 左边扫描结束 */

    MaxRightBorderSum = 0; RightBorderSum = 0;
    for (i = center + 1; i <= right; i++) { /* 从中线向右扫描 */
        RightBorderSum += List[i];
        if (RightBorderSum > MaxRightBorderSum)
            MaxRightBorderSum = RightBorderSum;
    } /* 右边扫描结束 */

    /* 下面返回"治"的结果 */
    return Max3(MaxLeftSum, MaxRightSum, MaxLeftBorderSum +
    MaxRightBorderSum);
}

int MaxSubseqSum3(int List[], int N)
{ /* 保持与前2种算法相同的函数接口 */
    return DivideAndConquer(List, 0, N - 1);
}
```

算法改进：在线处理