算法效率分析

吉林大学计算机学院
谷方明
fmgu2002@sina.com

学习目标

- 熟练掌握算法时间效率分析的基本方法
  - ✓ 事后估计方法
  - ✓ 事前分析方法
- 掌握算法空间效率分析的基本方法
- 了解时空积分

算法效率（性能）

- 算法效率
  - 时间效率：运行所耗费的时间
  - 空间效率：运行所占用的存储量

- 数据结构分析的实质就是算法分析。
  - 数据结构的优劣是由实现其运算的算法体现的

程序的执行时间

- #include <ctime>
- ……
- time_t start,end;
- start=clock();
- ……
- end=clcok();
- printf(“%d\n”,difftime(end,start));

事后分析

- 事后分析：编写程序，记录算法执行的时间。这是一种统计方法，也称事后统计。
- 优点：直观；
- 缺点
  - ✓ 编写费时。
  - ✓ **依赖计算机硬件和软件等环境因素。**同一算法在不同机器上执行时间不一定相同。时间的测试结果还取决于编写算法的语言、运行的操系统等。
  - ✓ **测试数据设计困难。**测试数据的规模为多少合适？
- 用途：验证

事前分析

- 事前分析：不编程序，直接分析算法。这是一种估算方法，也称事前估算。
- 算法的执行时间就是构成算法的所有语句的执行时间之和
  - ✓ 一个语句的执行时间 = 语句频率 × 一次语句执行的时间
  - ✓ 语句频率是指该语句在一个算法中重复执行的次数；

例：计算1+2+3+……+n的值

- 方法一: 累加法

```
sum=0
for(i=1;i<=n;i++)
    sum+=i;
```

- 累加法 的执行时间：
  - ✓ 赋值=1+ (1 + n) + n = 2n + 2
  - ✓ 比较=1 + n
  - ✓ 加法=n + n =2n

例：计算1+2+3+……+n的值

- 方法二: 高斯公式法

  sum=n*(n+1)/2

- 公式法的执行时间：
  - 赋值= 1
  - 加法= 1
  - 乘法= 1
  - 除法= 1

累加法的执行时间：

赋值 = 2n + 2
比较 = 1 + n
加法 = 2n

度量1:所有语句执行时间之和

- 一个语句的执行时间 ＝

语句频率 × 一次该语句执行的时间

- 语句频率：由算法直接确定，与所用的机器无关，且独立于程序设计语言。
- 语句执行时间：依赖机器、程序设计语言、编译程序

两种解释

- **理想计算模型**
  - 一台通用计算机(理想情况下)
  - 机器指令顺序执行，每次一条指令；
  - 做任一简单的事情都恰好花费1个时间单元
  - 无限内存，存取时间恒定

- **基本运算的时间囿界于常数 u**
  - 估算1个上界

求和两种算法的比较

- 累加法的语句频率之和=5n+3

赋值 = 2n + 2
比较 = 1 + n
加法 = 2n

- 公式法的语句频率之和=4

赋值= 1
加法= 1
乘法= 1
除法= 1

度量2: 所有语句频率之和

- 一个算法的执行时间定义为的所有语句的频率之和，记为t(n)
  - ✓ 其中：n是问题规模（区分输入规模）
    不同的问题表现形式不同。例：矩阵、多项式
- 一般不包括读入数据的时间
  - ✓ 当数据读入的时间比求解问题的时间多，需要考虑
- 特点
  - ✓ 优点：由算法直接确定，不依赖于机器
  - ✓ 缺点：计算依然繁琐

度量3: 时间复杂度(时间复杂性)

- 基本运算：算法中起**主要作用**且**花费时间最多**的运算。

- **时间复杂度:算法执行的基本运算的次数;一般用T(n)表示**，其中 n表示 所研究**问题的规模**。

例：计算1+2+3+……+n的值

算法S(*n* . sum)

S1[初始化]

sum←0.

S2[比较]

FOR *i*=1 TO *n* DO

sum ← sum+i. ▌

算法S的基本运算：加法

时间复杂度 T(n)=n

算法G(*n* . sum)

G1[公式]

sum←n*(n+1)/2. $\blacksquare$

算法G的基本运算：除法

T(n) = 1

例 A是一个含有n个实数的数组，给出求A之最大和最小元素的算法

算法SM (A, n . max,min)

SM1[初始化]

max←min←A[1].

SM2[比较]

FOR i=2 TO n DO

(IF A[i]> max THEN max←A[i].

IF A[i]< min THEN min←A[i]). ▌

算法SM的基本运算:元素的比较运算

时间复杂度为T(n)=2(n-1)。

例 实数数组R由n个元素组成，给定一个实数K，试确定K是否为R的元素。

算法F (R, n, K. i)

F1 [初始化]

$i \leftarrow 1$.

F2 [比较]

WHILE $i \le n$ **DO**

(**IF** R[i]=K **THEN RETURN.**

$i \leftarrow i+1$) . ■

算法F的基本运算是关键字比较，

最少比较次数：1　最大比较次数：n

定义 设一个领域问题的规模为n，D_n是该领域问题的所有输入的集合，任一输入$I \in D_n$，$P(I)$是I出现的概率，且满足$\sum P(I)=1$，$T(I)$是算法在输入I下所执行的基本运算次数。我们定义算法的**期望复杂度**为：

$$E(n)=\sum\{P(I)*T(I)\}$$

该算法的**最坏复杂度为**：

$$W(n)=\max\{T(I)\}$$

该算法的最好复杂度为：

$$B(n)=\min\{T(I)\}$$

上例中，设q(0≤q≤1)为K 在R中的概率

R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]	R[8]
5	20	12	7	30	40	25	16

q

$$\begin{cases} K=R[i] & q/n \\ K!=R[i] & 1-q \end{cases}$$

通过计算我们可以得到算法F的**期望复杂度**为

$$E(n)=\sum\{P(I)*T(I)\}$$

$$=\underbrace{(q/n)*1+(q/n)*2+\ldots+(q/n)*n}_{n项}+\underbrace{(1-q)*n}_{1项}$$

$$=q(n+1)/2+(1-q)n$$

如果已知K在R中，即$q=1$，则有

$$E(n)=(n+1)/2$$

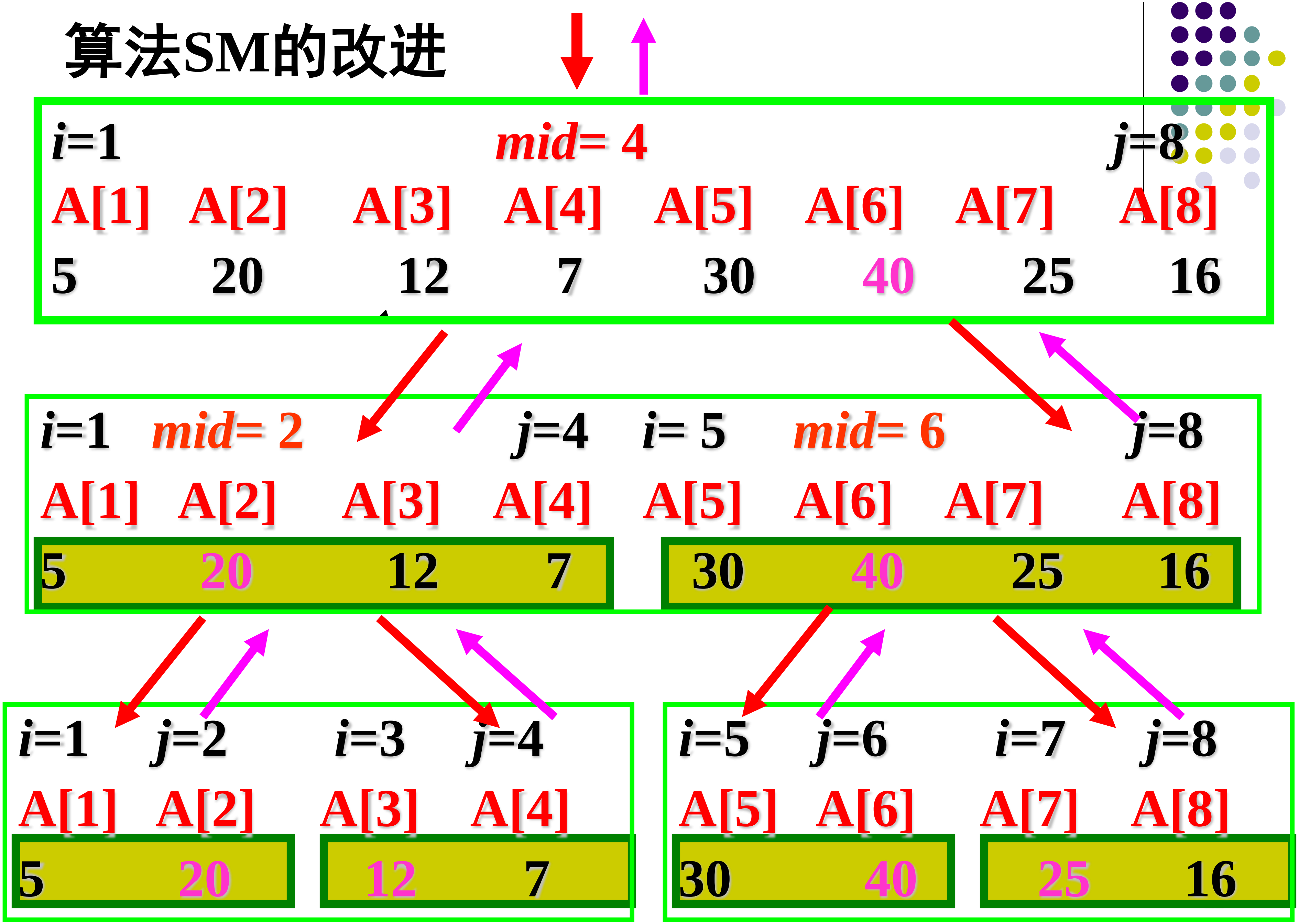
由算法F很容易看出该算法的**最坏复杂度**为

$$W(n)=\max\{T(I)\mid 1\le I\le n+1\}=n$$

小结

- 计算在简化，精度在下降
  - ✓ 所有语句的执行时间之和
  - ✓ 所有语句的频率之和
  - ✓ 基本运算的次数
- 时间复杂度T(n)的缺点
  - ✓ 估算精度：大部分情况下够用
  - ✓ 有时难以确定T(n) 的解析式

算法SM的改进



i=1
mid= 4
j=8
A[1] A[2] A[3] A[4] A[5] A[6] A[7] A[8]
5 20 12 7 30 40 25 16
i=1
mid= 2
j=4
i= 5
mid= 6
j=8
A[1] A[2] A[3] A[4] A[5] A[6] A[7] A[8]
5 20 12 7
30 40 25 16
i=1 j=2 i=3 j=4
A[1] A[2] A[3] A[4]
5 20
12 7
i=5 j=6 i=7 j=8
A[5] A[6] A[7] A[8]
30 40
25 16

算法BS（SM算法的改进）

算法BS（A , i , j . fmax , fmin）

/* 在数组A的第i个元素到第j个元素之间寻找最大和最小元素，已知$i \le j$ */

BS1 [递归出口]

IF $i = j$ THEN （ fmax←fmin←A[i]. RETURN. ）

IF $i = j - 1$ THEN

(IF A[i] < A[j]

THEN（fmax←A[j].fmin←A[i]）.

ELSE（fmax←A[i]. fmin←A[j]）.

RETURN）.

BS2 [取中值]

$$mid \leftarrow \lfloor (i+j)/2 \rfloor$$

BS3 [递归调用]

BS（A，*i*，*mid*. gmax，gmin）.

BS（A，*mid*+1，*j*. hmax，hmin）.

BS4 [合并]

fmax←max{gmax，hmax}.

fmin←min{gmin，hmin}. ▌

如果算法BS的基本运算为元素的比较，则BS对不同的输入A[i]到A[j]都有相同的基本运算次数。设T(n)表示其基本运算次数，则根据算法BS的递归过程，有如下的递归表达式：

$$T(n)=\begin{cases}0 & n=1\\ 1 & n=2\\ T(\lfloor n/2 \rfloor)+T(\lceil n/2 \rceil)+2 & n>2\end{cases}$$

- T(n)的解析式难以获得。
- 特殊情况： $n=2^k$ （ k是正整数 ）

$$
\begin{aligned}
T(n) &= 2\times T(n/2)+2 \\
&= 2\times(2\times T(n/4)+2)+2 \\
&= 4\times T(n/4)+4+2 \\
&\cdots \qquad\qquad \cdots \\
&= 2^{k-1}\times T(2)+\sum_{i=1}^{k-1}2^i \\
&= 2^{k-1}+2^k-2 \\
&= \frac{3}{2}n-2
\end{aligned}
$$

比较算法SM 和算法BS

- 算法SM和BS的时间复杂度均为线性，但因 $\frac{3}{2}n-2<2(n-1)$ ，故就计算时间而言，算法BS优于算法SM。
- 算法BS是递归算法，因此它的实现需要额外的辅助空间栈。
- **Tradeoff:** 空间换时间

函数的比较

- 记f(n)=O(g(n))当且仅当存在正常数C 和 n_0，使得对任意的$n \geq n_0$，有 $f(n) \leq Cg(n)$.
- 记f(n)= Ω(g(n))当且仅当存在正常数C 和 n_0，使得对任意的$n \geq n_0$，有 $f(n) \geq Cg(n)$.
- 记f(n)= Θ ($g(n)$) 当且仅当存在正的常数 C_1， C_2 和 n_0，使得对任意的 $n \geq n_0$，有 $C_1 g(n) \leq f(n) \leq C_2 g(n)$ 。 $f(n)= \Theta(g(n)) \Leftrightarrow f(n)= O(g(n))$ 且$f(n)= \Omega(g(n))$
- 记f(n)= o($g(n)$)当且仅当 f(n)=O(g(n)) 且$f(n) \neq \Theta(g(n))$

大O表示法

- 例如：可以把f(n) = 1000n 记为 O(n^2)
  - ✓ 虽然 n 较小时，1000 n 要比 n^2 大，但 n^2 以更快的速度增长，随着 n 的增大，n^2最终将更大. 在这一情况下，n=1000 是转折点。
- **f(n)=O(g(n))表示f(n)的增长率小于等于g(n)的增长率。这种记法称为大O表示法。一般说“大O...”，有时也说“至多...级（阶）的”**
  - ✓ 写成 1000n < n^2 这样的形式意义不明确；大O表示法更能突出**相对增长率**；

n → ∞的增长率

规模	1	logn	n	n^2
1	1	1	1	1
10	1	4	10	100
100	1	7	100	10000
1000	1	10	1000	1000000
10000	1	13	10000	100000000
100000	1	16	100000	10000000000
1000000	1	19	1000000	1000000000000

- 例 $f(n) = 3n - 2$ 是 $O(n)$.

证明： 由大O的定义，存在$C = 3$，$n_0 = 1$，使得对任意的 $n \geq n_0$ ，有

$$3n - 2 \leq 3n \text{ 即 } f(n) \leq Cg(n) .$$

- 例 $f(n) = 3\log_2 n + \log_2\log_2 n$ 是 $O(\log_2 n)$

证明： 由大O的定义，存在$C = 4$， $n_0 = 2$，使得对任意的 $n \geq n_0$ ，有

$$3\log_2 n + \log_2\log_2 n \leq 4\log_2 n \text{ 即 } f(n) \leq Cg(n) .$$

度量4：渐进时间复杂度

- 将时间复杂度 $T(n)$记为 $O(g(n))$，称为渐进时间复杂度；
- 表明：$T(n)$的一个上界是g(n);
- 渐进时间复杂度O(g(n))表示的是n趋于无穷时的状况。

大O的性质

- 定理1.1：若 $A(n)=a_m n^m+\ldots+a_1 n+a$ 是关于n的m次多项式，则

$$A(n)=O(n^m)$$

  - ✓ 多项式函数的阶取决于阶最高的项，与其系数和其它较低阶项无关。

- **性质：若$f_1(n)=O(g_1(n))$，$f_2(n)=O(g_2(n))$，则**

  (1) $f_1(n)+f_2(n)=O(\max\{g_1(n)，g_2(n)\})$

  (2) $f_1(n)\times f_2(n)=O(g_1(n)\times g_2(n))$

  - ✓ O(g(n))在 =右侧 表示 以g(n)为上界的函数集合

渐进时间复杂度小结

- 利用大O的性质，计算简化，精度下降
- 算法运行时间分析因TAOCP而流行；大O等记号是Knuth首倡；实际尚无统一规定，许多人在使用Θ()更愿用O()；建议O()尽量接近Θ()。

算法的阶

- $O(1)$ 表示算法的时间复杂度为一常数. $O(\log n)$、$O(n)$、$O(n^2)$、$O(n^3)$、$O(n^m)$ 和 $O(2^n)$ 分别表示算法时间复杂度的阶至多为对数、线性、平方、立方、多项式和指数阶的，其中常数 $m \geq 1$.

$$O(1) < O(\log_2 \log_2 n) < O(\log_2 n) < O(n)$$

$$< O(n \log_2 n) < O(n^2) < O(n^3) < O(2^n)$$

P=NP?

度量换算成真实时间

- **CPU性能参数：MIPS(Million Instructions Per Second)。衡量单字长定点指令平均执行速度。常用的CPU>=100MIPS**
- **10^8次CPU基本指令数 换算成 1S**
  - ✓ 基本指令指的是加减乘除比较赋值等指令。
  - ✓ **10^6** 次级别的文件输入输出 换算成 **1S**。
  - ✓ 外设更慢；
  - ✓ 位运算快 **1S** 执行**10^9**条
- 基本运算 **=>** 基本指令 **=> S**

2. 算法的空间效率分析

- 类似时间效率分析

- 事后分析
- 事前分析
  - ✓ 逻辑度量（基本类型）
  - ✓ 空间复杂度S(n)
  - ✓ 渐进空间复杂度
  - ✓ 实际计算：字节

例

- 一段程序如下

```
int n,m;
int head[MAXN];
Int g[MAXN][MAXN];
```

- 空间复杂度是多少？
  - ✓ $S(n) = n^2 + n + 2$
  - ✓ $S(n) = O(n^2)$

3 算法时间与空间分析

- 一个算法在不同的执行时间内，它占用的内存空间量不一定相等，占用空间量y是时间x的函数，即$y=f(x)$。
- 称积分 $\int_0^t f(x)dx$ 为该算法的时空积分，其中t是该算法的执行时间。
- 基于时空积分，可以比较算法优劣，时空积分较小的算法较优。

例如，一个算法执行时间为30秒，前10秒算法占用60个字节，第二个10秒算法占用70个字节，最后10秒算法占用80个字节。该算法的时空分布如图所示。

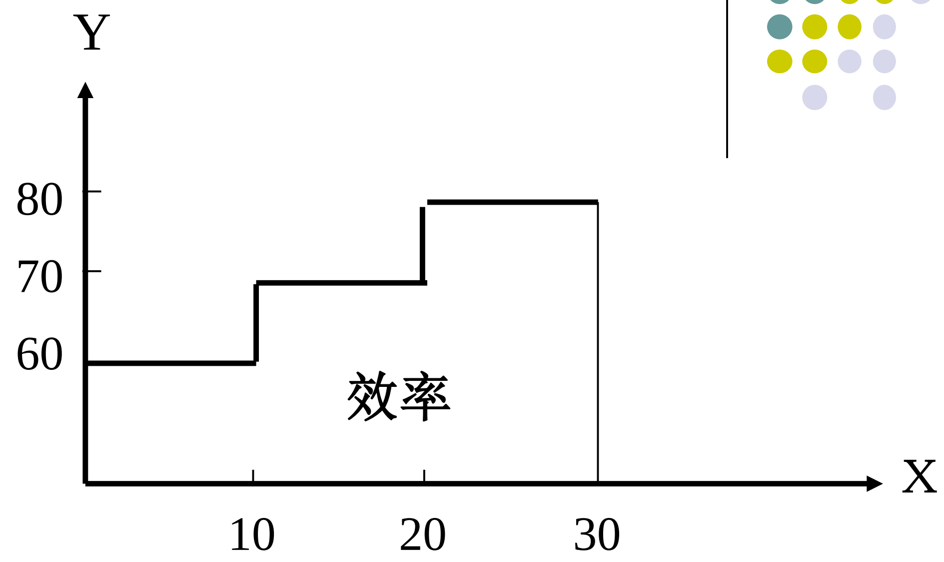
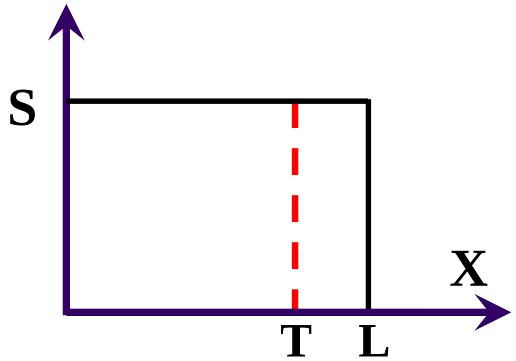


图 时空分布示意图

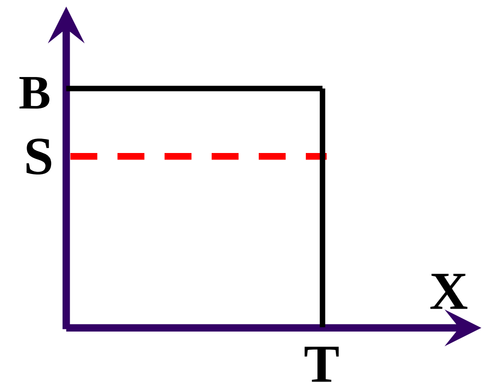
显然算法的时空积分为:

60*10+70*10+80*10=2100（字节秒）

算法SM和算法BS的时空积分比较



(a) SM 的时空图



(b) BS 的时空图

在算法BS运行过程中：

1、需保留递归所需参数，递归最深调用层数正比于 $\log_2 n$，所需存储空间为 $\log_2 n$ 的常数倍；

2、始终需要保留输入文件，占用的空间为 n 的常数倍。

- 算法SM：时间 $L=a\times 2(n-1)$，空间 $S\approx b\times n$;
- 算法 BS：时间 $T\approx a\times(5n/3-2)$, $T<L$,

  空间为 $B\approx c\times\log_2 n+b\times n$, $B>S$,
- $L=T+\Delta L$ ， $B=S+\Delta B$

  $\Delta L\approx a\times[2\times(n-1)-(5n/3-2)]=a\times n/3$， $\Delta B\approx c\times\log_2 n$.
- $W_{SM}=(T+\Delta L)\times S$ ， $W_{BS}=(S+\Delta B)\times T$
- $W_{SM}-W_{BS}=\Delta L\times S-\Delta B\times T$

  $\approx a\times b\times n^2/3-c\log_2 n\times a(5n/3-2)$
- 当 n 较大时， $W_{SM}>W_{BS}$ ， 算法 BS 优于算法 SM

总结

- 时间效率分析的事后估计方法
- 时间效率分析的事前分析方法
  - ✓ 所有语句执行时间之和
  - ✓ 所有语句频率之和
  - ✓ 时间复杂度
  - ✓ 渐进时间复杂度
- 空间效率分析

精度下降

计算简化

课堂测试

课后思考

- 算法的最坏时间复杂度 **VS** 算法的阶?
- 算法的最好时间复杂度 **VS** 输入规模最小 ?
- 好的算法 **VS** 好的计算机 ?