1. 算法在计算中的作用
   1. 算法
   2. 作为一种技术的算法
2. 算法基础
   1. 插入排序
   2. 分析算法

2.2-2: selection sort(A)

for i=1 to A.legth-1

  key=A[i]

  j=i+1

while j< A.legth+1

if A[j]<A[i]

A[i]=A[j]

j++

A[j]=key

* 1. 设计算法
     1. 分治法
     2. 分析分治算法

1. 函数的增长
   1. 渐进记号
   2. 标准记号和常用函数
2. 分治策略
   1. 最大子数组问题

4.1-5

//已知的A数组中的前j个数中的最大子数组

MAX-SIZE-j

//待求的含前j+1个数的最大子数组,初始为NULL

MAX-SIZE-j+1

for i=1 to MAX-SIZE-j

max+= MAX-SIZE-j[i]

for i=j+1 downto 1

sum+=A[i]

if(sum>=max)

MAX-SIZE-j+1=A[i] to A[j+1]

if(MAX-SIZE-j+1==NULL)

MAX-SIZE-j+1= MAX-SIZE-j

return MAX-SIZE-j+1

* 1. 矩阵乘法的Strassen算法

template<typename DerivedA,typename DerivedB, typename DerivedC>

void StrassenAlgorithm(const MatrixBase< DerivedA>& matrixA, const MatrixBase< DerivedB>& matrixB, MatrixBase< DerivedC>& matrixC) {

int matrix\_half = 0;

matrix\_half = matrixB.rows()/2;

//在使用迭代的时候一定要注意设置停止的手段

if (matrix\_half < 1) {

matrixC = matrixA \* matrixB;

return;

}

MatrixXd A11(matrix\_half, matrix\_half);

MatrixXd A12(matrix\_half, matrix\_half);

MatrixXd A21(matrix\_half, matrix\_half);

MatrixXd A22(matrix\_half, matrix\_half);

MatrixXd B11(matrix\_half, matrix\_half);

MatrixXd B12(matrix\_half, matrix\_half);

MatrixXd B21(matrix\_half, matrix\_half);

MatrixXd B22(matrix\_half, matrix\_half);

MatrixXd C11(matrix\_half, matrix\_half);

MatrixXd C12(matrix\_half, matrix\_half);

MatrixXd C21(matrix\_half, matrix\_half);

MatrixXd C22(matrix\_half, matrix\_half);

A11 = matrixA.block(0, 0, matrix\_half, matrix\_half);

A12 = matrixA.block(0, matrix\_half, matrix\_half, matrix\_half);

A21 = matrixA.block(matrix\_half, 0, matrix\_half, matrix\_half);

A22 = matrixA.block(matrix\_half, matrix\_half, matrix\_half, matrix\_half);

B11 = matrixB.block(0, 0, matrix\_half, matrix\_half);

B12 = matrixB.block(0, matrix\_half, matrix\_half, matrix\_half);

B21 = matrixB.block(matrix\_half, 0, matrix\_half, matrix\_half);

B22 = matrixB.block(matrix\_half, matrix\_half, matrix\_half, matrix\_half);

C11 = matrixC.block(0, 0, matrix\_half, matrix\_half);

C12 = matrixC.block(0, matrix\_half, matrix\_half, matrix\_half);

C21 = matrixC.block(matrix\_half, 0, matrix\_half, matrix\_half);

C22 = matrixC.block(matrix\_half, matrix\_half, matrix\_half, matrix\_half);

MatrixXd S1(matrix\_half, matrix\_half);

MatrixXd S2(matrix\_half, matrix\_half);

MatrixXd S3(matrix\_half, matrix\_half);

MatrixXd S4(matrix\_half, matrix\_half);

MatrixXd S5(matrix\_half, matrix\_half);

MatrixXd S6(matrix\_half, matrix\_half);

MatrixXd S7(matrix\_half, matrix\_half);

MatrixXd S8(matrix\_half, matrix\_half);

MatrixXd S9(matrix\_half, matrix\_half);

MatrixXd S10(matrix\_half, matrix\_half);

S1 = B12 - B22;

S2 = A11 + A12;

S3 = A21 + A22;

S4 = B21 - B11;

S5 = A11 + A22;

S6 = B11 + B22;

S7 = A12 - A22;

S8 = B21 + B22;

S9 = A11 - A21;

S10 = B11 + B12;

MatrixXd P1(matrix\_half, matrix\_half);

MatrixXd P2(matrix\_half, matrix\_half);

MatrixXd P3(matrix\_half, matrix\_half);

MatrixXd P4(matrix\_half, matrix\_half);

MatrixXd P5(matrix\_half, matrix\_half);

MatrixXd P6(matrix\_half, matrix\_half);

MatrixXd P7(matrix\_half, matrix\_half);

StrassenAlgorithm(A11, S1, P1);

StrassenAlgorithm(S2, B22, P2);

StrassenAlgorithm(S3, B11, P3);

StrassenAlgorithm(A22, S4, P4);

StrassenAlgorithm(S5, S6, P5);

StrassenAlgorithm(S7, S8, P6);

StrassenAlgorithm(S9, S10, P7);

C11 = P5 + P4 - P2 + P6;

C12 = P1 + P2;

C21 = P3 + P4;

C22 = P5 + P1 - P3 - P7;

matrixC.block(0, 0, matrix\_half, matrix\_half) = C11;

matrixC.block(0, matrix\_half, matrix\_half, matrix\_half) = C12;

matrixC.block(matrix\_half, 0, matrix\_half, matrix\_half) = C21;

matrixC.block(matrix\_half, matrix\_half, matrix\_half, matrix\_half) = C22;

}

* 1. 用代入法求解递归式

1. 概率分析和随机算法
   1. 雇用问题
   2. 指示器随机变量

**5.2-4帽子核对问题：**

Xi=I{顾客拿到自己的帽子}=

E[Xi]=1/(n-i+1);

E[X]=E[]

=

=

* 1. …
  2. …

**思考题：**

5-2（查找一个无序数组）

RANDOM-SEARCH(A):

n=A.length

for i=1 to n

swap A[i] with A[RANDOM(i,n)]

for i=1 to n

if(A[i]==x)

return i

return “x not in A”