1.分治法

（1）二分搜索

代码（递归）:

int BinarySearch\_Rec(int [] a, int x, int left, int right)

{

while (left <= right){

int middle = left + (right - left) / 2;

if (x==a[middle])

return middle;

if (x > a[middle])

return BinarySearch\_Rec(a, x, mid + 1, right);

else

return BinarySearch\_Rec(a, x, left, middle-1);

}

return -1; // 未找到x

}

时间复杂度：O(logn）

（2）合并排序

代码（递归）:

void MergeSort(Type a[], int left, int right)

{

if (left<right) {

//至少有2个元素

int i=(left+right)/2; //取中点

MergeSort(a, left, i);

MergeSort(a, i+1, right);

//二路归并合并到数组b

merge(a, left, i, right); }

时间复杂度：O(nlogn)

（3）快速排序

代码（递归）:

void qSort(Type a[], int p, int r)

{

if (p<r) {

int q=partition(p,r);

qSort (p,q-1); //对左半段排序

qSort (q+1,r); //对右半段排序

}

}

int partition (type a[], int p, int r){

int i = p, j=r+1;

Type x = a[p];

while(true){

while (a[++i] < x&&i<r);

while (a[--j] > x);

if (i >= j) break;

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

}

a[p] = a[j]; //给起点a[p]赋值

a[j] = x;//将循环到j作为q位置

return j;

}

最好时间复杂度：O(nlogn)

2.动态规划

（1）最长公共子序列

动态规划代码(P55 LCSLength函数):

void LOOKUP\_CHAIN (int m,int n ,char []x,char []y,int[][]c,int [][]b)

{

int i,j;

for (i=1;i<=m;i++) c[i][0]=0;

for(i=1;i<=n;i++) c[0][i]=0;

for(i=1;i<=m;i++)

for(j=1;j<=n;j++){

if(x[i]==y[j]){c[i][j]=c[i-1][j-1]+1; b[i][j]=1;}

else if (c[i-1][j]>=c[i][j-1]){c[i][j]=c[i-1][j];b[i][j]=2;}

else{ c[i][j]=c[i][j-1]; b[i][j]=3; }

}}}

时间复杂度：O(mn）

（2）最大子段和问题



0-1背包问题代码

int Knapsack(int n, int c, int w[], int v[],int x[]){

int m[n+1][c+1];

for(int j=0;j<=c;j++)

m[0][j]=0;

for(int i=1;i<=n;i++){

for(int j=0;j<=c;j++){

if(w[i] <= j){

m[i][j] = max(m[i - 1][j], m[i-1][j - w[i]] + v[i]);

}

else {

m[i][j] = m[i-1][j];

}

}

}

int j=c;

for(int i=n;i>1;i--){

if(m[i][j] == m[i-1][j]){

x[i] = 0;

}

else {

x[i] = 1;

j -= w[i];

}

}

return m[n][c];

}

简答：每一种算法解决的场景、思路、适用问题？

分治法：

解决的问题满足以下特征：

① 问题的规模缩小到一定的程度就可以容易地解决；

② 问题可以分解为若干个规模较小的相同问题;

③ 子问题的解可以合并为该问题的解；

④ 各个子问题是相互独立的；

动态规划：

解决的问题满足以下特征：

① 子问题不独立；

② 存在迭代递归关系；

③ 存在最优子结构；（如果总问题是最优解，则所有子问题都是最优解）

④ 利用分治求解，部分子问题被重复计算。



回溯法：

主要思想：类似穷举的搜索尝试：在解空间树中搜索尝试过程中寻找问题的解，当发现已不满足求解条件时，就“回溯”（即回退），尝试别的路径。类似于深度优先的思路搜素整个解空间，同时在求解过程中，降低无用搜索，加入剪枝函数。

问题的解空间：

【解空间】一个复杂问题的解决方案是由若干个小的决策步骤组成的决策

序列，解决一个问题的所有可能的决策序列构成该问题的解空间。

【可行解】解空间中满足约束条件的决策序列。

【最优解】在约束条件下使目标达到最优的可行解。

分支限界法：

主要思想：分支限界法类似于回溯法，也是一种在问题的解空间树上搜索问题解的算法。以广度优先或以最小耗费优先的方式搜索解空间树。

【求解步骤】

① 活结点一旦成为扩展结点，就一次性产生其所有儿子结点。当前点从活动节点队列中

删去；

② 儿子结点中，不可行解或非最优解结点删去，其余儿子结点被加入活结点列表中；

③ 从活结点表中取下一结点成为当前扩展结点；

④ 重复上述结点扩展过程。直至活动结点列表为空；

