第一章



1-3. 最大公约数为1。快1414倍。

主要考虑循环次数，程序1-2的while循环体做了10次，程序1-3的while循环体做了14141次（14142-2循环）

若考虑其他语句，则没有这么多，可能就601倍。

第二章



2-8.（1）画线语句的执行次数为。。划线语句的执行次数应该理解为一格整体。

（2）画线语句的执行次数为 。。

（3）画线语句的执行次数为 。。

（4）当n为奇数时画线语句的执行次数为 ，

当n为偶数时画线语句的执行次数为 。。

2-10.（1） 当  时，，所以，可选 ，。对于，，所以，。

（2） 当  时，，所以，可选 ，。对于，，所以，。

（3） 由（1）、（2）可知，取，，，当时，有，所以。

2-11. (1) 当时，，所以，。可选 ，。对于，，即。注意：是*f*(*n*)和*g*(*n*)的关系。

（2） 当  时，，所以 ，。可选 ，。对于 ，，即 。

（3）因为 ，。当  时，，。所以，可选 ，，对于，，即 。

第二章

2-17. 证明：设，则 。

























当 时，。所以，。

第五章

5-4. SolutionType DandC1(int left,int right)

{

while(!Small(left,right)&&left<right)

{

int m=Divide(left,right);

if(x<P(m) right=m-1;

else if(x>P[m]) left=m+1;

else return S(P)

}

}

5-7. template <class T>

int SortableList<T>::BSearch(const T&x,int left,int right) const

{

if (left<=right)

{

int m=(right+left)/3;

if (x<l[m]) return BSearch(x,left,m-1);

else if (x>l[m]) return BSearch(x,m+1,right);

else return m;

}

return -1;

}

第五章

9．



证明：因为该算法在成功搜索的情况下，关键字之间的比较次数至少为，至多为。在不成功搜索的情况下，关键字之间的比较次数至少为，至多为。所以，算法的最好、最坏情况的时间复杂度为。

假定查找表中任何一个元素的概率是相等的，为，那么，

不成功搜索的平均时间复杂度为，

成功搜索的平均时间复杂度为。

其中，是二叉判定树的内路径长度，是外路径长度，并且。

11.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 步数 | 0 | 1 | 2 | 3 | 4 | 5 |
| 初始时 | 1 | 1 | 1 | 1 | 1 |  |
| 1 | [1 | 1] | 1 | [1 | 1] | ∞ |
| 2 | [1] | 1 | 1 | [1 | 1] | ∞ |
| 3 | 1 | 1 | 1 | [1 | 1] | ∞ |
| 4 | 1 | 1 | 1 | [1] | 1 | ∞ |
| 排序结果 | 1 | 1 | 1 | 1 | 1 | ∞ |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 步数 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 初始时 | 5 | 5 | 8 | 3 | 4 | 3 | 2 | ∞ |
| 1 | [4 | 2 | 3 | 3] | 5 | [8 | 5] | ∞ |
| 2 | [3 | 2 | 3] | 4 | 5 | [8 | 5] | ∞ |
| 3 | [3 | 2] | 3 | 4 | 5 | [8 | 5] | ∞ |
| 4 | [2] | 3 | 3 | 4 | 5 | [8 | 5] | ∞ |
| 5 | 2 | 3 | 3 | 4 | 5 | [5] | 8 | ∞ |
| 排序结果 | 2 | 3 | 3 | 4 | 5 | 5 | 8 | ∞ |

12.（1）证明：当或或时，程序显然正确。

当n=right-left+1>2时，程序执行下面的语句：

int k=(right-left+1)/3;

StoogeSort(left,right-k);

StoogeSort(left+k,right);

StoogeSort(left,right-k);

①首次递归StoogeSort(left,right-k);时，序列的前2/3的子序列有序。

②当递归执行StoogeSort(left+k,right);时，使序列的后2/3的子序列有序，经过这两次递归排序，使原序列的后1/3的位置上是整个序列中较大的数，即序列后1/3的位置上数均大于前2/3的数，但此时，前2/3的序列并不一定是有序的。

③再次执行StoogeSort(left,right-k);使序列的前2/3有序。

经过三次递归，最终使序列有序。

所以，这一排序算法是正确的。

（2）最坏情况发生在序列按递减次序排列。

，，。

设，则。















冒泡排序最坏时间复杂度为，队排序最坏时间复杂度为，快速排序最坏时间复杂度为。所以，该算法不如冒泡排序，堆排序，快速排序。

13. template <class T>

select (T&x,int k)

{

if(m>n) swap(m,n);

if(m+n<k||k<=0) {cout<<"Out Of Bounds"; return false;}

int \*p=new temp[k];

int mid,left=0,right=n-1,cnt=0,j=0,r=0;

for(int i=0;i<m;i++)

{

while(k>0)

{

do

{

mid=(left+right)/2;

if(a[mid]<b[i]) left=mid;

else if(a[mid]>b[i]) right=mid;

else {cnt=mid; break;}

}while(left<right-1)

if(a[left]<b[i]) cnt=left;

else cnt=left-1;

if(k>cnt)

{

if(cnt>0)

{

for(j=0;j<cnt;j++)

{

temp[j]=a[r];

r++;

}

left=cnt;

k-=cnt;

}

else

{

temp[j]=b[i];

left=0;

k--;

}

}

else

{

for(j=0;j<k;j++)

{

temp[j]=a[r];

r++;

}

left=cnt;

k-=cnt;

return temp[k-1];

}

}

}

}

第六章

1.由题可得：，

所以，最优解为，

最大收益为。

8.



第六章

**6-9**.

普里姆算法。

因为图G是一个无向连通图。

所以n-1<=m<=n (n-1)/2;

O(n)<=m<=O(n2);

克鲁斯卡尔对边数较少的带权图有较高的效率，而，此图边数较多，接近完全图，故选用普里姆算法。

**6-10**.

T仍是新图的最小代价生成树。

证明：假设T不是新图的最小代价生成树，T’是新图的最小代价生成树，那么cost(T’)<cost(T)。有cost(T’)-c(n-1)<cost(t)-c(n-1)，即在原图中存在一颗生成树，其代价小于T的代价，这与题设中T是原图的最小代价生成树矛盾。所以假设不成立。证毕。

第七章

1. Bcost(1,0)=0;

Bcost(2,1)=c(1,1)+Bcost(1.0)=5

Bcost(2,2)=c(1,2)+Bcost(1,0)=2

Bcost(3,3)=min{c(2,3)+Bcost(2,2),c(1,3)+Bcost(2,1)}=min{6+2,3+5}=8

Bcost(3,4)=c(2,4)+Bcost(2,2)=5+2=7

Bcost(3,5)=min{c(1,5)+Bcost(2,1),c(2,5)+Bcost(2,2)}=min{3+5,8+2}=8

Bcost(4,6)=min{c(3,6)+Bcost(3,3),c(4,6)+Bcost(3,4),c(5,6)+Bcost(3,5)}=min{1+8,6+7,6+8}=9

Bcost(4,7)=min{c(3,7)+Bcost(3,3),c(4,7)+Bcost(3,4),c(5,7)+Bcost(3,5)}=min{4+8,2+7,6+8}=9

Bcost(5,8)=min{c(6,8)+Bcost(4,6),c(7,8)+Bcost(4,7)}=min{7+9,3+9}=12

2.向后递推的计算过程如上题所示

向前递推过程如下：

cost(5,8)=0

cost(4,6)=7,cost(4,7)=3

cost(3,3)=min{1+cost(4,6),4+cost(4,7)}=7,

cost(3,4)=min{6+cost(4,6),2+cost(4,7)}=5

cost(3,5)=min{6+cost(4,6),2+cost(4,7)}=5

cost(2,1)=min{3+cost(3,3),3+cost(3,5)}=8

cost(2,2)=min{6+cost(3,3),8+cost(3,5),5+cost(3,4)}=10

cost(1,0)=min{5+cost(2,1),2+cost(2,2)}=12

所以，d(4,6)=d(4,7)=8, d(3,3)=d(3,4)=d(3,5)=7, d(2,1)=5, d(2,2)=4, d(1,0)=2

从s到t的最短路径为 (0, d(1,0)=2, d(2,2)=4, d(3,4)=7, d(4,7)=8),路径长为12。

第七章

9. char A[8]={‘0’,’x’,’z’,’y’,’z’,’z’,’y’,’x’ }

B[8]={‘0’,’z’,’x’,’y’,’y’,’z’,’x’,’z’}

 

(a) c[i][j] （b）s[i][j]

所以，最长公共字串为 (x,y,z,z)。

第七章

11. void LCS::CLCS ( int i , int j )

{

if ( i = = 0 || j = = 0) return;

if (c[i][j] = = c[i-1][j-1]+1)

{

CLCS ( i-1,j-1);

Cout<<a[i];

}

else if ( c[i-1][j]>=c[i][j-1]) CLCS (i-1,j);

else CLCS (i,j-1);

}

12. int LCS::LCSLength()

{

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

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

else if (c[i-1][j]>=c[i][j-1]) c[i][j]=c[i-1][j];

else c[i][j]=c[i][j-1];

return c[m][n];

}

15.  , ,

 , ,

 , ,

 ,



8-1．

状态空间：描述问题的各种可能的情况，一种情况对呀状态空间的一个状态。

显示约束：用于规定每个xi取值的约束条件称为显示约束

隐式约束：用于判定一个候选解是否为可行解的条件

问题状态：在状态空间树中的每个节点称为一个问题状态

解状态：如果从根到树中某个状态的路径代表一个作为候选解的元组，则该状态为解状态

答案状态：如果从根到树中某个状态的路径代表一个作为可行解的元组，则该状态为解状态。

活结点：回溯法从开始结点出发，以深度优先的方式搜索整个解空间，这个开始结点就成为一个活结点。未检测的结点称为活结点

扩展结点：算法从x出发，访问x的摸个后继结点y，则x被称为扩展结点

约束函数：一个约束函数是关于部分向量的函数Bk(x0,x1.....xk),它被定义为：如果可以判定Y的子树上不含任何答案状态，则Bk(x0,x1.....xk)为false,否则为true.

剪枝函数：约束函数和限界函数的目的相同，都是为了剪去不必要搜索的子树，减少问题求解所需实际生成的状态节点数，他们统称为剪枝函数

8-2

bool place(int k,int ,I,int\*x)

{

For(int j=0,j<k,j++)

If((x[j]==i)||(abs(x[j]-j)==abs(j-k)))

Return false;

Return true;

}

Void nqueens(int k,int n,int \*x)

{

For(int i=0;i<n;i++)

If(place(k,I,x))

{

X[k]=I;

If(k= =n-1

{

For(i=0;i<n;i++)cout<<x[i]<<endl;

Return;

}

Else nqueens(k+1,n,x)

}

}

Void nqueens(int n,int \*x)

{

Nqueens(0,n,x);

}