一、 简答题

1．答案：

令$a=\frac{8}{3}$，$b=2$，满足条件 a≥1, b＞1，又有$n^{\log_2\frac{8}{3}-\log_2\frac{4}{3}}=n$，$\varepsilon=\log_2\frac{4}{3}>0$，满足主定理

第一种情况，因而，T(n) = $\Theta(n^{\log_2\frac{8}{3}})=\Theta(n^{3-\log_2 3})$

2. 答案：

欧几里德算法

设 r=a mod b，当 r≠0 时，依据欧几里德定理，有

$$\begin{cases} r = a \bmod b \\ \gcd(a,b)=\gcd(b,r) \quad r\neq 0 \end{cases}$$

3. 答案：

（1）结点编码

{AB,AC,AD,BA,BC,BD,DA,DB,DC,EA, EB, EC, ED}

= { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

（2）数据结构--图：邻接矩阵表示法

```
G=(V, E)
V:struct Node{
      char name[10]; //表示道路名称，如 AB
int color;               //表示灯的颜色编号
}v[13];
E: e[13][13];            //1 为相邻，0 表示不相邻
```

（3）输入 e 值，搜索矩阵 e，将不相邻的结点涂成同一种颜色，直到把所有结点全着色，计算颜色数量。

（4）类似问题有：地图着色问题。

4．答案：

```
Divide-and-Conquer(n){
        if (|n| <=n0)       // n0 为邻界点
            return DC(n) ;
  divide P into smaller sub-instances n1, n2, …, nk;
    for (i=1; i <=k; i++)
        yi=Divide-and-Conquer(ni) ;
   return Merge(y1, y2, …, yk) ;}
```

二、算法设计题

1. （1）采用穷举法

设士兵数目为 x，x 为正整数，则由于 x%7==2，因而可设 x 的初值为 9，x 依次递增，步长为 1。由命题可知，第一个满足下式：

$$(x\%3==2)+(x\%5==3)+(x\%7==2)==3$$

即为所求。

（2）用 c 语言表达为：

```
int main()
{
    int x=9;
    while(1)
    {
        if((x%3==2)+(x%5==3)+(x%7==2)==3)
            break;
        x++;
    }
    printf("韩信共有%d 个士兵\n",x);
    return 0;
}
```

（3）由（2）可知时间渐近函数为：T(n) = x – 9，所以，时间渐近复杂度为$\theta(x)$[注：O(x)也对]。

2. （1）计算模型：

因为 $f(x)=3x^2-e^x-1$，所以 $f'(x)=6x-e^x$

得到 $x_2=x_1-(3x_1^2-exp(x_1)-1)/(6x_1- exp(x_1))$

（2）算法描述：

输入：x

输出：x_2

```
equationN(x)
{
    x1=x
    x2=x1-( 3*x1*x1-exp(x1)-1 )/( 6*x1- exp(x1) )
    while(|x1-x2|>0.00001)
    {
        x1= x2
        x2=x1-( 3*x1*x1-exp(x1)-1 )/( 6*x1- exp(x1) )
    }
    return x2;   }
```

3. 参考答案

（1）目标函数：$max\sum_{i=1}^{n} x_i$

s.t. $\sum_{i=1}^{n} x_i\, w_i \leqslant C$

$x_i=0,1 \quad i=1,2,\cdots,n$

（2）

```
void loading(int x[], int w[], int C, int n)
{   int t[]; //存储 w 排序后的下标
    sort(w, t, n); //按 w 从小到大排序
    x[n]={0}; //选中标识位置为未选中
    for(i=1;i<=n&&w[t[i]]<=C;i=i+1){
        x[t[i]]=1;   C=C-w[t[i]];
    }
    output(x);
}
```

三、综合题

1．

（1） 算法设计与描述	（2）算法分析
输入：由 n 个活动组成的数组 A[n]	
输出：最多能够安排的活动数目 S	

void GreedySelect(A[], n){
    int i, j=1;
    A[].f={0}; //将活动选中标识均置为 0
   MergeSort(A);
   A[1].f=1; count=1;
   for(i=2;i<=n;i++)
    if(A[i].s>=A[j].e){
       A[i].f=1;
       j=i;
       count=count+1;
    }
   output(count, A); //S 为 A 中 f=1 活动
}
	(1) 问题的输入规模为 n
	(2) 初始化需要循环 n 次
	(3) 按结束时间排序算法的时间复杂度为 O(nlogn)
	(4) 用贪心算法选择最多安排的活动数目需要循环 n 次
	(5) 依据定理 2.2 可得 T(n)=O(nlog n)

2. **参考答案**

（1）

选择 pivot=a[left]值，j=right 对集合进行二分

1． j-left=k-1，则分界数据就是选择问题的答案

2． j-left>k-1，则选择问题的答案继续在左子集中找，问题规模变小了。
3． j-left<k-1，则选择问题的答案继续在右子集中找，问题变为选择第 k-left-1 小的数，问题的规模也变小。

（2）

```
int select(int a[],int left,int right,int k)
{   int i,j,pivot,t;
    if(left>=right)   return a[left];
    i=left;    j=right+1;
    pivot=a[left];
    while(1)
    {   do{
              i++;
        }while(a[i]<pivot);
        do{
              j--;
        }while(a[j]>pivot);
        if(i>=j)
              break;
        t=a[i],a[i]=a[j],a[j]=t;
    }
    if(j-left+1==k)
        return pivot;
    a[left]=a[j];    a[j]=pivot;
    if(k<(j-left+1))
        select(a,left,j-1,k);
    else
        select(a,j+1,right,k-j-1+left);
}
```

（3）

算法分析

(1)最坏情况下的复杂性是 O(n2)，left 总是为空，第 k 个元素总是位于 right 子集中。
(2)设 n=2k，算法的平均复杂性是 O (n+logn)。若仔细地选择分界元素，则最坏情况下的时间开销也可以变成 O(n)。
(3)它与本章第一个例子非常相似，只对一半子集进行搜索，所不同的时，由于 pivot 点选择的不同，多数情况下它是非等分的。

3. 参考答案

（1） 计算模型

1．计算

由问题分析，可以得到下述递推方程：

$$
\begin{cases}
f_0(w) = 0 & 0 \le w \le W \quad (8-4) \\
f_i(0) = 0 & 0 \le i \le n \quad (8-5) \\
f_i(w) = \max\{ f_{i-1}(w), f_{i-1}(w - w_i) + v_i \} & (8-6) \\
f_i(w) = f_{i-1}(w) & 0 \le w < w_i \quad (8-7)
\end{cases}
$$

其中，式(8-4)表示背包没有装入物品，式(8-5)表示背包承载为 0。

2．存储

i—表示物品编号，j 表示背包当前的限载重量

W—表示背包最大承载重量

w[]—表示物品重量，如第 i 个物品的重量为 w[i]

v[]—表示物品价值，如第 i 个物品的价值为 v[i]

x[]—表示物品的选取状态，如第 i 个物品被选取则 x[i]=1，否则 x[i]=0

f[][]—表示在某限载情况下，背包最优装载的价值，如 f[i][j]指在背包限载重量为 j 的情况下，第 i 阶段(前 i 个物品)最优装载的价值。

（2）

```
int KnapSack(int n,int w[],int v[],int x[],int W)
{
    int i,j;
    int jmax =min(w[1]-1, W);
    //新增第 1 个元素
    for (j = 0; j <= jmax; j++)   //配额小于物品 1 重量
        f[1][j] = 0;              //装不下物品 1
    for (j = w[1]; j <=W ; j++)
        f[1][j] = v[1];
    //进行动态规划
    for ( i = 2; i <= n; i++)
    {
     jmax =min(w[i]-1,W); //配额小于物品 i 重量
     for ( j = 0; j <= jmax; j++) //只能取上一次的最优
        f[i][j] = f[i-1][j];
     for (j = w[i]; j <= W; j++)
        f[i][j] =max(f[i-1][j],(f[i-1][j-w[i]] + v[i]) );
    }
    //判断哪些物品被选中
    for(i=n;i>=1;i--)
    {
        if(f[i][j]>f[i-1][j])
        {
            x[i]=1;
            j=j-w[i];
        }
        else
            x[i]=0;
    }
    printf("选中的物品是:\n");
    for(i=1;i<=n;i++)
        printf("%d ",x[i]);
    return f[n][W];     //返回最优值
}
int min(int a,int b)
{   if(a>=b)
        return b;
    else return a;
}
```

j=W;	

（3）

复杂度分析

1．问题的输入规模是 n 个重量 w[n],n 个价值 v[n]及背包总承载重量 W

2．第 1 阶段动态规划前后两部分合起来的初始化是 W 次

除去第 1 阶段动态规划外，还有 n-1 阶段动态规划，其中均包含 W 次优化过程，可以表示如下：

$\mathrm{T(n)} = \sum_{i=2}^{n}\left(\sum_{j=0}^{W} C\right) = (n-1)(W+1)$

3．回溯推导最优解所需要的计算次数为 n 次。

总综上所述，可得

T(n)=W+(n-1)(W+1)+n=n*W+2*n-1=θ(n*W)

$\log_2 W$，若设 $k=\log_2 W$，则有 $W=2^k$。当背包承载的重量很大时，它的时间复杂度实际上是 $T(n)=\theta(n*2^k)$