一、算法设计实例

1、　快速排序（分治法）

```
    int partition(float a[], int p, int r)
{
    int i= p, j=r+1;
    float x = a[p];

    while(1)
    {
        while(a[++i] < x);
        while(a[--j] < x);

        if(i >= j)
            break;
        swap (a[i], a[j]);
    }

    a[p] = a[j];
    a[j] = x;
    return j;
}
```

```
void Quicksort(float a[], int p, int r)
{
    //快速排序
    if(p < r)
    {
        int q = partition(a, p, r);
        Quicksort(a, p, q-1);
      Quicksort(a, p+1, r);
    }
}
```

2、　归并排序（分治法）

```
  void mergesort (Type a[], int left, int right)
{
    if (left < rigth)
    {
        int mid = (left + right)/2; //取中点
        mergesort (a, left, mid);
      mergesort (a, mid+1, right);
      mergesort (a, b, left, right);//合并到数组 b
```

```
    mergesort (a, b, left, right);//复制到数组 a
  }
}
```

3、　背包问题（贪心算法）

```
  void knapsack (int n, float m ,float v[], float w[], float x[])
{
  sort(n, v, w) //非递增排序
  int i;
  for (i=1 ; i<=n; i++)
      x[i] = 0;

  float c = m;
  for (i= 1; i<=n; i++)
  {
      if (w[i] > c)
         break;
    x[i] = 1;
    c -= w[i];
  }
```

```
    if (i <= n)
        x[i] = c/w[i] ;
}
```

4、 活动安排问题（贪心算法）

```
    void Greadyselector (int n, Type s[], Type f[], bool A[])
{
    //s[i] 为活动结束时间， f[j]为 j 活动开始时间

    A[i] = true;
    int j= 1;

    for (i=2; i<=n; i++)
    {
        if (s[i] >= f[j])
        {
            A[i] = true; j=i;
        }
        else
            A[i] = false;
    }
```

}

5、 喷水装置问题（贪心算法）

```
void knansack (int w, int d, float r[], int n)
{
//w 为草坪长度  d 为草坪宽度  r[]为喷水装置的喷水半径，
//n 为 n 种喷水装置 ，  喷水装置的喷水半径 >= d/2

sort (r[], n); //降序排序
count = 0; //记录装置数

for (i=1; i<=n; i++)
    x[i] = 0;
//初始时，  所有喷水装置没有安装  x[i]=0

for (i=1; w >= 0; i++)
{
    x[i] = 1;
  count ++;

  w = w - 2*sqart(r[i]*r[i] - 1);
```

```
    }

    count << 装置数: << count << end1;
    for (i=1; i<= n; i++)
        count << 喷水装置半径：<< r[i] << end1;
}
```

6、　最优服务问题（贪心算法）

```
    double greedy (rector<int> x, int s)
{
    rector <int> st(s+1, 0);
    rector <int> su(s+1 ,0);
    int n=x.size();

    //st[] 是服务数组， st[j]为第 j 个队列上的某一个顾客的等待时间
    //su[] 是求和数组， su[j]为第 j 个队列上所有顾客的等待时间

    sort(x.begin(), x.end());
    //每个顾客所需要的服务时间升序排列
```

```
    int i=0, j=0;
    while( i<n )
    {
        st[j] += x[i]; //x[i]= x.begin-x.end
      su[j] += st[j];
      i++;
      j++;
      if ( j==s)
          j=0;
    }
    double t=0;
    for (i=0; i<s; i++)
        t += su[i];
    t /=n;
    return t;
}
```

7、　石子合并问题（贪心算法）

```
float bebig (int A[], int n)
{
m = n;

sort(A, m); //升序

while (m>1)
{
    for (i=3; i<=m; i++)
      if ( p<A[i] )
         break;
      else
         A[i-2]=A[i];


   for (A[i-2] = p;i<= m; i++)
   {
     A[i-1] = A[i];
     m--;
   }
}
count << A[1] << end1
```

}

8、　石子合并问题（动态规划算法）

best [i][j] 表示 i-j 合并化最优值

sum [i][j] 表示第 i 个石子到第 j 个石子的总数量

```
             | 0
f(i,j) =     |
             | min{f(i,k) + f(k+1,j)}+ sum(i,j)
```

```
int sum [maxm]
int best [maxm][maxn];
int n, stme[maxn];

int getbest();
{
    //初始化，没有合并
    for (int i=0; i< n; i++)
        best[i][j] =0;
    //还需要进行 合并
```

```
    for (int r=1; r<n; r++)
     {
          for(i=0; i<n-r; i++)
        {
            int j = i+v;
          best[i][j]= INT- MAX;
          int add = sum[j] -(i>0 ! sum[i-1]: 0);
          //中间断开位置，取最优值
          for (int k=i; k<j; ++k)
          {
              best[i][j]= min(best[i][j], best[i][k]+best[k+1][j])+add;
          }
        }
    }
    return best[0][n-1];
}
```

9、　最小重量机器设计问题（回溯法）

```
typedef struct Qnode
{
    float wei;//重量
    float val;//价格
    int ceng;//层次
    int no;   //供应商

    struct Qnode * Parent;//双亲指针
}Qnode;

float wei[n+1][m+1] = ;
float val[n+1][m+1] = ;

void backstack (Qnode *p)
{
    if (p->ceng == n+1)
    {
        if (bestw > p->wei)
      {
          testw = p->wei;
         best =p;
      }
```

```
    }
    else
    {
        for (i=1; i<=m; i++)
          k=p->ceng;
      vt = p->val + val[k][i];
      wt = p->wei + wei[k][i];

      if (vt <=d && wt <= bestw)
      {
          s = new Qnode;
        s->val = vt;
        s->wei = wt;
        s->ceng = k+1;
        s->no = 1;
        s->parent = p;
        backstrack(S);
      }
    }
}
```

10、　最小重量机器设计问题（分支限界法）

```
typedef struct Qnode
{
    float wei;//重量
    float val;//价格
    int ceng;//层次
    int no;   //供应商

    struct Qnode * Parent;//双亲指针
}Qnode;

float wei[n+1][m+1] = ;
float val[n+1][m+1] = ;

void minloading()
{
    float wt=0;
```

```
float vt=0;
float bestw= Max;//最小重量
Qnode *best;
s = new Qnode;
s->wei = 0;
s->val = 0;
s->ceng = 1;
s->no =0;
s->parent=null;

Iinit_Queue(Q);
EnQueue(Q,S);
do
{
    p=OutQueue(Q);//出队
  if (p->ceng==n+1)
  {
     if(bestw > p->wei)
     {
       bestw = p->wei;
       best =p;
     }
```

```
    }
    else
    {
      for (i=1; i<=m;i++)
     {
         k= p->ceng;
         vt =p->val + val[k][i];
         wt =p->wei + wei[k][i];
         if (vt <=d && wt <=bestw)
         {
             s= new Qnode;
           s->ceng=k+1;
           s->wt=wt;
           s->val=val;
           s->no=i;
           s->parent=p;
           EnQueue (Q, S);
         }
     }
    }
}while(!empty(Q));
```

```
    p=best;
    while(p->parent)
    {
        count << 部件：  << p->ceng-1 << end1;
      count << 供应商：<< p->no << end1;

      p=p->parent;
    }
}
```

11、 快速排序（随机化算法—舍伍德算法）

```
int partion (int a[], int l, int r)
{
    key = a[l];
    int i=l, j=r;
```

```
    while(1)
    {
        while(a[++i]< key && i<=r);
      while(a[--j]> key && j>=l);

      if(i >= j)
          break;
      if (a[i] != a[j])
          swap(a[i] , a[j]);
    }
    if ((j !=l) && a[l] != a[j])
        swap(a[l], a[j]);

    return j;
}

int Ranpartion (int a[], int l, int r)
{
    k=rand()%(r-1 + l)+1;
    swap(a[k], a[l]);
```

```
    int ans = partion(a, l, r);
    return ans;
}

int Quick_sort(int a[], int l, int r, int k)
{
    int p= Randpartion(a, l, r);
    if (p == k)
        return a[k];
    else if (k< p)
        return Quick_sort(a, l, p-1, k);
    else
    {
        int j= 0;
      for (int i= p+1; i<=r; i++)
          b[j++] = a[i]

      return Quick_sort(b, 1, j, k-p);
    }
}
```

12、　线性选择（随机化算法—舍伍德算法）

二、简答题

1．分治法的基本思想

分治法的基本思想是将一个规模为 n 的问题分解为 k 个规模较小的子问题，这些子问题互相独立且与原问题相同。递归地解这些子问题，然后将各个子问题的解合并得到原问题的解。

2．分治法与动态规划法的相同点

将待求解的问题分解成若干子问题，先求解子问题，然后再从这些子问题的解得到原问题的解。

3．分治法与动态规划法的不同点

1、适合于用动态规划法求解的问题，分解得到的各子问题往往不是相互独立的；　而分治法中子问题相互独立。

2、动态规划法用表保存已求解过的子问题的解，再次碰到同样的子问题时不必重新求解，而只需查询答案，故可获得多项式级时间复杂度，效率较高；

而分治法中对于每次出现的子问题均求解，导致同样的子问题被反复求解，故产生指数增长的时间复杂度，效率较低。

动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干个子问题。但是经分解得到的子问题往往不是互相独立的。不同子问题的数目常常只有多项式量级。在用分治法求解时，有些子问题被重复计算了许多次。如果能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，就可以避免大量重复计算，从而得到多项式时间算法。

4. 分支限界法与回溯法的相同点

相同点：二者都是一种在问题的解空间树上搜索问题解的算法。

不同点：1.在一般情况下，分支限界法与回溯法的求解目标不同。回溯法的求解目标是找出解空间中满足约束条件的所有解，而分支限界法的求解目标则是找出满足约束条件的一个解，或是在满足约束条件的解中找出使某一目标函数值达到极大或极小的解，即在某种意义下的最优解。

2.回溯法与分支限界法对解空间的搜索方式不同，回溯法用深度优先搜索，而分支限界法则通常采用广度优先搜索。

3.对节点存储的常用数据结构以及节点存储特性也各不相同，除由搜索方式决定的不同的存储结构外，分支限界法通常需要存储一些额外的信息以利于进一步地展开搜索。

5．分治法所能解决的问题一般具有的特征

（1）该问题的规模缩小到一定的程度就可以容易地解决；

（2）该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质；

（3）利用该问题分解出的子问题的解可以合并为该问题的解；

（4）该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子子问题。

6. 用分支限界法设计算法的步骤

根据所解问题确定解结构；确定解空间树；以广度优先搜索搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。

(1)针对所给问题，定义问题的解空间；(2)确定易于搜索的解空间结构；(3)以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。常用剪枝函数：用约束函数在扩展结点处剪去不满足约束的子树；用限界函数剪去得不到最优解的子树。

7. 回溯法中常见的两类典型的解空间树

回溯法中常见的两类典型的解空间树是子集树和排列树

当所给的问题是从 n 个元素的集合 S 中找出满足某种性质的子集时，相应的解空间树称为子集树。这类子集树通常有2n 个叶结点，遍历子集树需 O(2n)计算时间 。

当所给的问题是确定 n 个元素满足某种性质的排列时，相应的解空间树称为排列树。这类排列树通常有 n!个叶结点。遍历排列树需要 O(n!)计算时间。

8. 分支限界法的搜索策略

分支限界法主要策略是以广度优先或者以最小耗费优先的方式搜索解空间树, 当搜索到扩展节点(即搜索空间树的非叶节点)处, 首先生成其所有的儿子节点以加速搜索进度;在每一扩展节点处计算一个评价函数值, 此时, 对问题的代价进行阶段性评价, 如果到达某节点处的代价超出已经获得的最小代价, 则终止该节点所有分支的继续搜索, 从而有效缩减评价的规模。对于通过评价的扩展节点, 建立其下一层扩展节点表, 从中选择一个最有利的节点作为新的扩展节点, 使搜索向着解空间树上最优解的分支推进, 以便尽快找出一个最优解。