单选题

使用分治法不需要满足的条件（B）

A、子问题不能够重复

B、子问题必须具有相同的性质

C、子问题的解可以合并

D、原问题和子问题使用相同的方法求解

分治法解决问题分为三步走，即分、治、合。下面列出了几种操作，请按分、治、合顺序选择正确的表述(C)

（1）将子问题的解合并为大问题的解
（2）将问题分解为子问题
（3）将子问题合并为大问题
（4）求子问题的解
（5）将问题分解为可重复的子问题

A、2，1，3

B、5，1，3

C、2，4，1

D、5，4，1

快速排序算法思想，运用分治算法对 n个元素进行划分，如何选择划分基准？下面（D）答案解释最合理

A、随机选择一个元素作为划分基准

B、取子序列的第一个元素作为划分基准

C、用中位数的中位数方法寻找划分基准

D、以上皆可行。但不同方法，算法复杂度上界可能不同

解决问题的基本步骤是（C）。（1）算法设计（2）算法实现（3）数学建模（4）算法分析（5）正确性证明

A、(3)(1)(4)(5)(2)

B、(3)(4)(1)(5)(2)

C、(3)(1)(5)(4)(2)

D、(1)(2)(3)(4)(5)

问题:分治法的时间复杂性分析，通常是通过分析得到一个关于时间复杂性T(n)的一个递归方程， 然后解此方程可得T(n)的结果。T(n)的递归定义如下：关于该定义中k，n/m，f(n)的解释准确的是（B）

A:k是常系数，n/m是规模为n的问题分为m个子问题，f(n)是分解为子问题的时间复杂性与合并子问题的解的时间复杂性之和。

B:k是子问题个数，n/m是子问题的规模，f(n)是分解为子问题的时间复杂性与合并子问题的解的时间复杂性之和

C:k是子问题个数，n/m是子问题的规模，f(n)是规模为n的问题分解为子问题的时间复杂性

D:k是常系数，n/m是规模为n的问题分为m个子问题，f(n)是将子问题的解合并为问题的解的时间复杂性。

问题:快速排序和归并排序是常用的排序算法，也都是采用分治法解决的问题。快速排序的时间复杂性为O()，而归并排序的时间复杂性为O(nlogn)，究其原因，下面的解释你认为哪个正确？（C）

A:这是因为归并排序把问题划分为子问题时的时间复杂性低，而快速排序划分为子问题是使用partition()函数，划分为子问题的时间复杂性高。

B:因为归并排序把问题划分为两个子问题时其规模大致相等，是原来规模的n/2，而快速排序划分为子问题是使用partition()函数，划分为子问题时不能保证二个子问题的规模大致相同，在极端状况下，每次都只划分为1个子问题，其规模为原问题规模n-1，因此快速排序在极端状况下的时间复杂性的递归定义为T(n)=T(n-1)+O(n)

C:归并排序的分和合的时间复杂性之和低于快速排序的分和合的时间复杂性之和

D:以上都不正确

简答题一

给定数组a[0:n-1] 试设计一个算法，在最坏情况下用

$$n+\lceil logn\rceil-2$$

试比较找出a[0:n-1]中元素的最大值和次大值

```
#include <iostream.h>
#define m 8
void max1max2(int a[m][2], int n, int k)
{

    int s[8][2], i, max1, max2;
    if (n == 1)
        cout << "最大值：" << s[m - 1][0];
    else
    {
        for (int i = 0; i < m - 1;)
        {
            if (a[i][0] < a[i + 1][0])
            {
                max1 = a[i + 1][0];
                max2 = a[i][0];
            }
            else
            {
                max1 = a[i][0];
                max2 = a[i + 1][0];
            }
            s[k][0] = max1;
            s[k][1] = max2;
            k++;
            i = i + 2;
        }
        n = n / 2;
        for (int x = 0; x < n; x++)
            for (int j = 0; j < 2; j++)
                a[i][j] = s[i][j];
        n = n / 2;
    }
    max1max2(s, n, k);
}

void main()
{
    int a[m][2];
    for (int i = 0; i < 8; i++)
    {
        cin >> a[i][0];
        a[i][1] = 0;
    }
    max1max2(a, 8, 0);
}
```

简答题二

设$A=\{a_1,a_2,...,a_n\}$是正整数的集合，且

$$\sum_{l=1}^{n} a_i = N$$

设计一个算法判断是否能够把A划分成两个子集A_1和A_2，使得A_1中的数之和与A_2中的数之和相等？说明算法的设计思想，估计算法最坏情况下的时间复杂度

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;
int main()
{
    int n;
    int sum=0;
    int a[310];
    int ans[155];
    scanf("%d",&n);
    for(int i=1; i<=n; i++)
    {
        scanf("%d",&a[i]);
        sum+=a[i];
    }
    if(sum%2!=0)
    {
        printf("no\n");
        return 0;
    }
    int m=sum/2;
    int dp[n+1][m+1];
    memset(dp,0,sizeof(dp));
    dp[1][0]=1;
    dp[1][a[1]]=1;
    for(int i=2; i<=n; i++) //前i个元素
    {
        for(int j=0; j<=m; j++) //可以构成j
        {
            if(dp[i-1][j]||dp[i-1][j-a[i]])
            {
                dp[i][j]=1;
            }
        }
    }
    if(!dp[n][m])
    {

        printf("no\n");
        return 0;
    }
    int cnt=0;
    for(int j=m; j>=0; j--)
    {
        for(int i=n; i>=1; i--)
        {
            if(dp[i][j]&&!dp[i-1][j])//二维数组中每一列最顶部的那个T
```

```
            {
                ans[cnt]=a[i];
                cnt++;
                j=j-a[i];
                if(j==0)//找完结束
                {
                    break;
                }
            }
        }
    }
    for(int i=0; i<cnt-1; i++)
    {
        printf("%d ",ans[i]);
    }
    printf("%d\n",ans[cnt-1]);
    return 0;
}
```