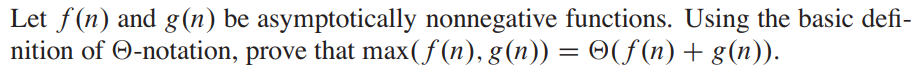
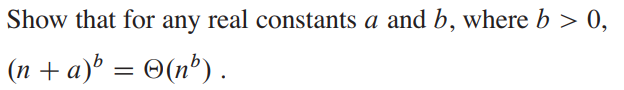
第一次作业

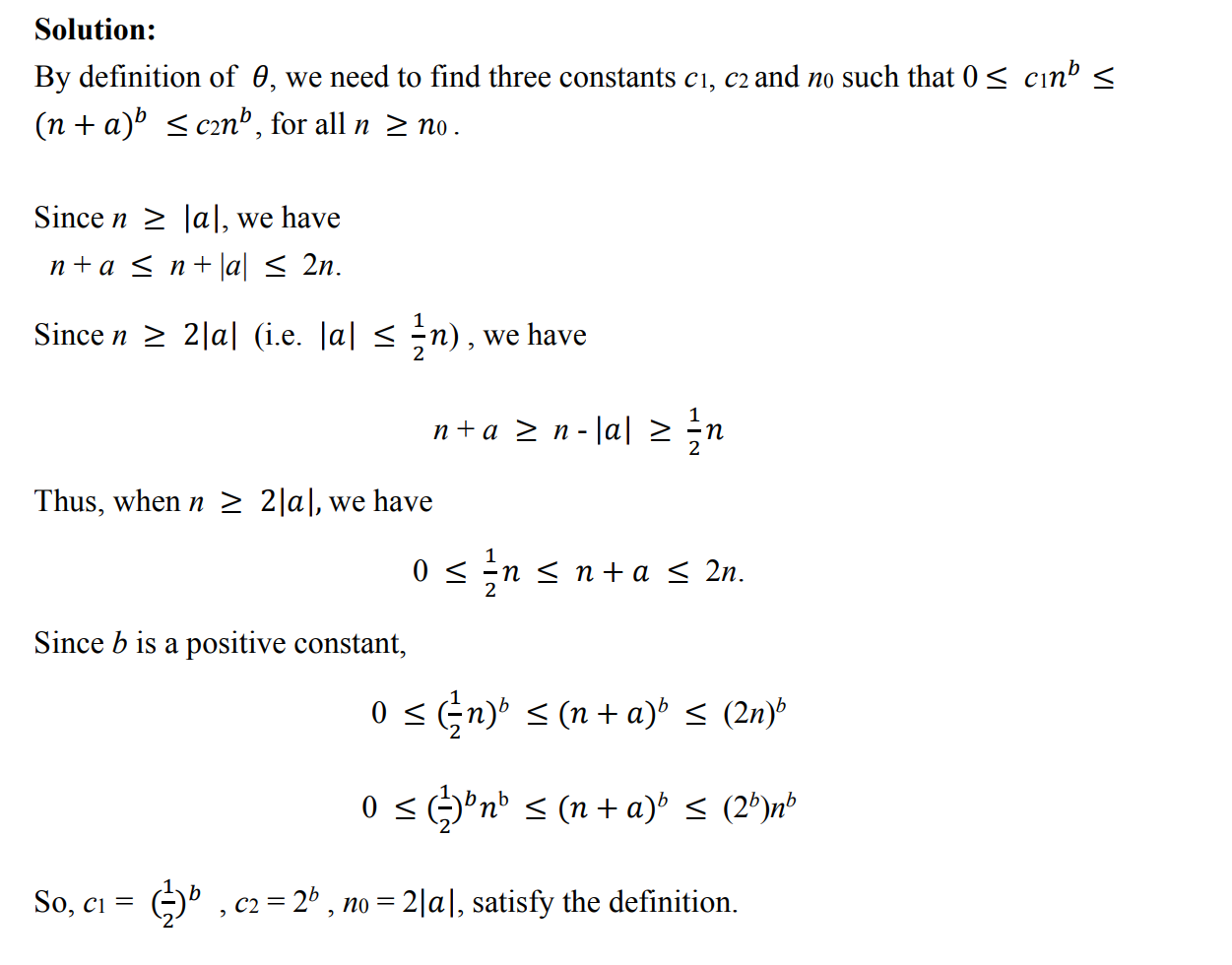
1. 

设f(n)和g(n)是渐近非负函数 ，使用基本定义的Θ符号,证明max (f (n), g (n)) =Θ(f (n) + g (n))。

2.



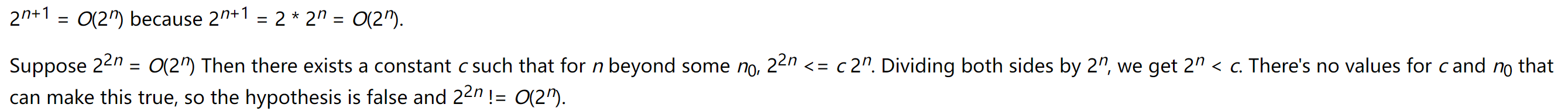
Show that for any real constants a and b, where b > 0,(n +a)b =Θ(n b).



3.



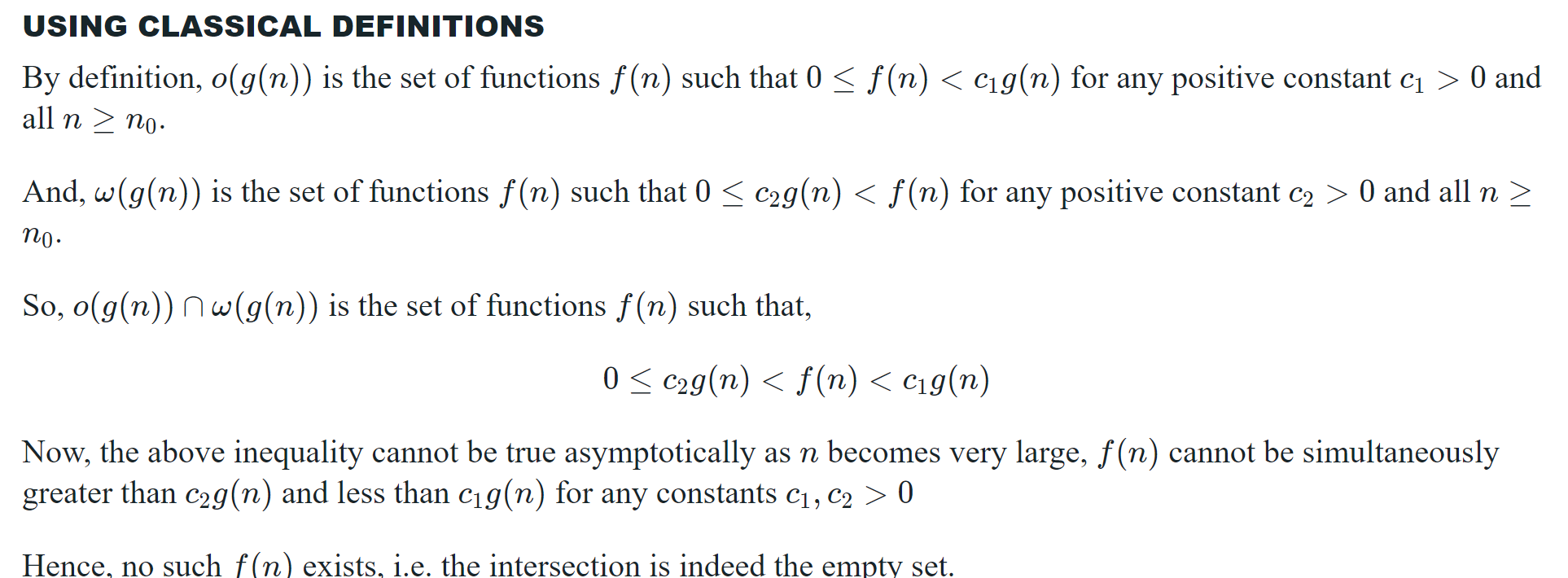
Is 2^(n+1) = O(2^n)? Is 2^(2n) = O(2^n)?

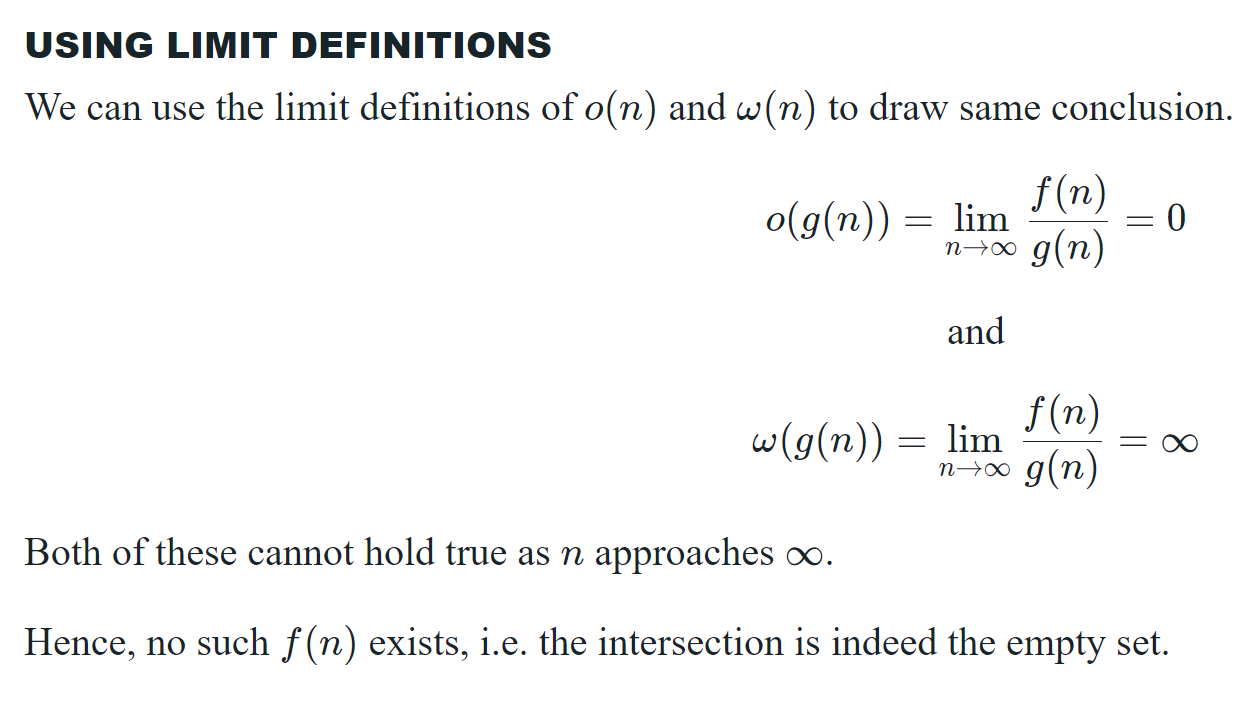


4.



Prove that o(g(n))∩w(g(n)) is the empty set.





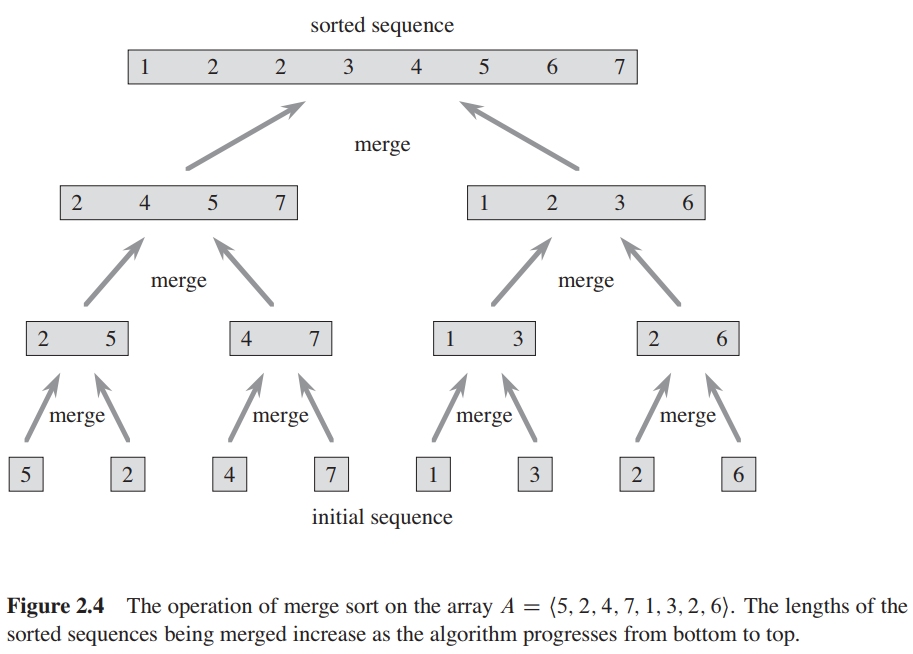
5.

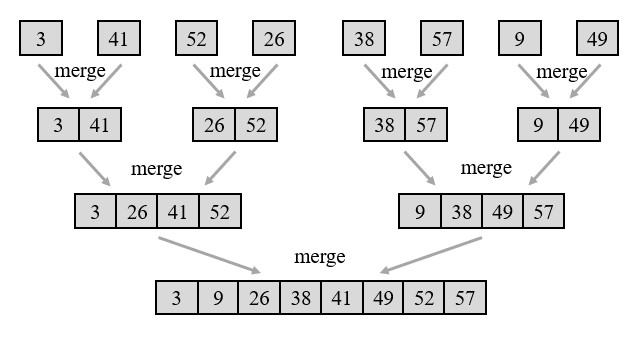


Using Figure 2.4 as a model, illustrate the operation of merge sort on the array

A = <3,41,52,26, 38, 57, 9, 49>.

把图片倒过来就ok了





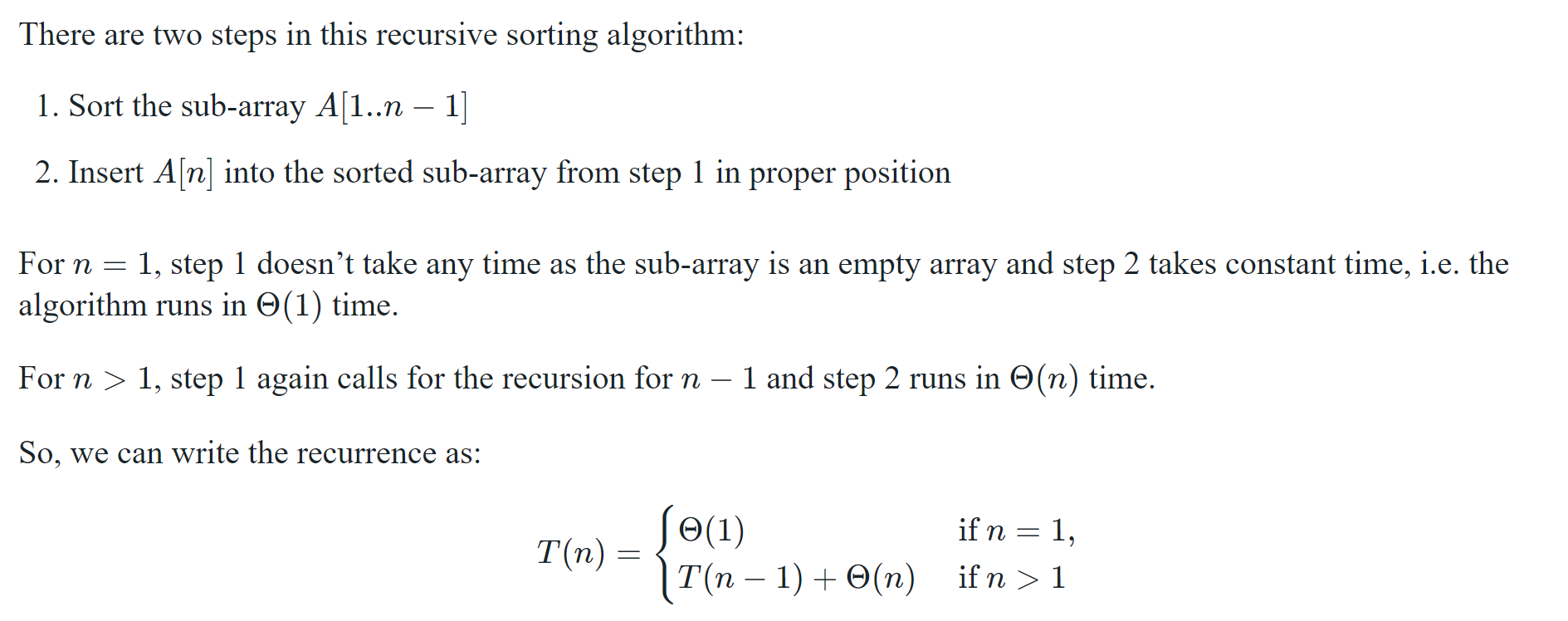
6**.**

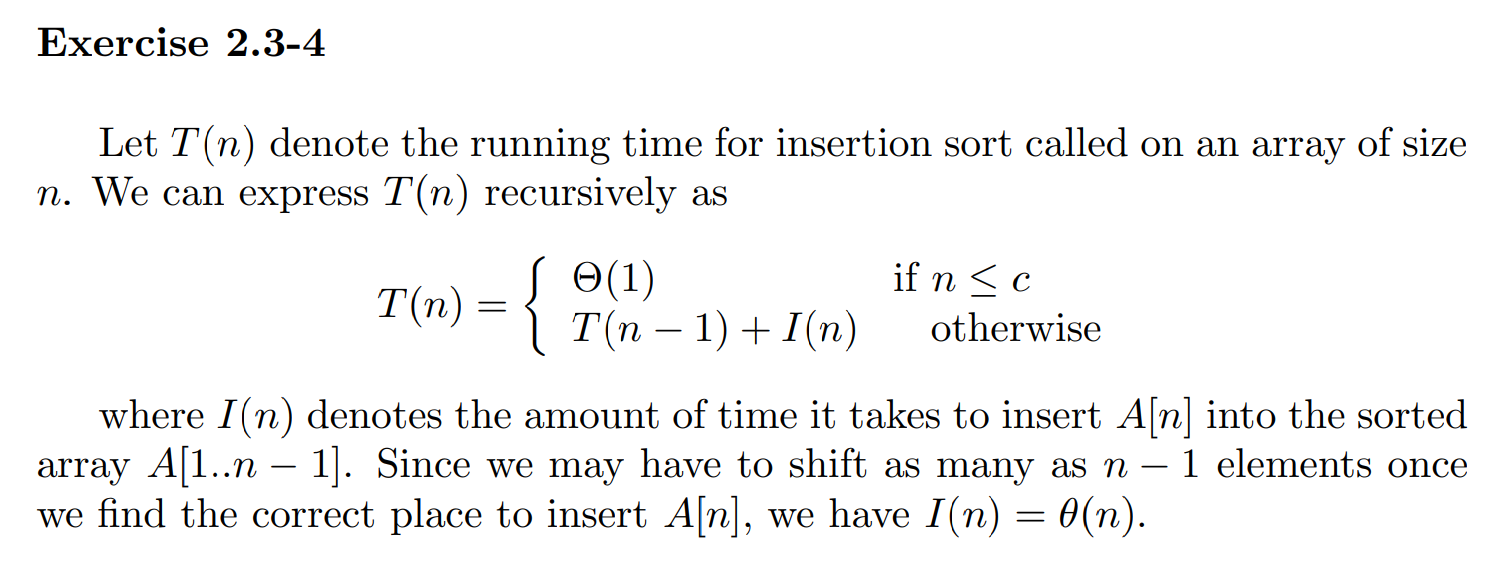
**Express the function n^3/1000-100n^2-100n+3n in terms of Θ-notation** **and then prove it.**

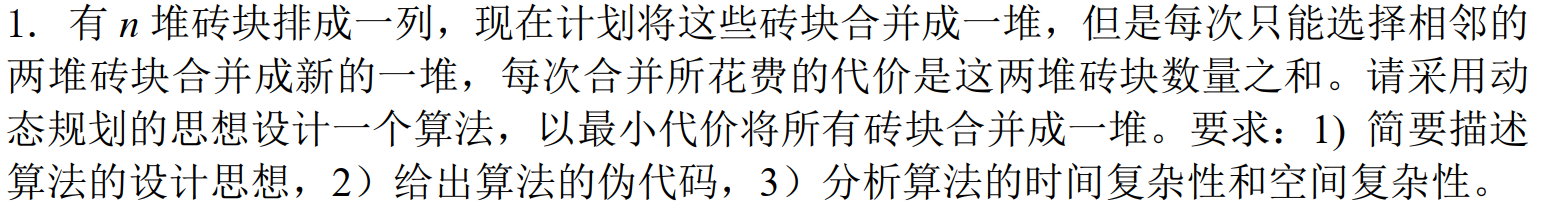
**7.**

**We can express insertion sort as a recursive procedure as follows. In order to sort A [1..n], we recursively sort A[1..n−1] and then insert A[n] into the sorted array A[1..n−1]. Write the recursion formula for the running time of this recursive version of insertion sort.**

我们可以将插入排序表示为一个递归过程，如下所示。为了对A [1..]n]，递归排序A[1..然后将A[n]插入到排序数组A[1..n−1]中。写出这个插入排序的递归版本的运行时间的递归公式。

****

****

8.

1.有n堆砖块排成---列，现在计划将这些砖块合并成-堆，但是每次只能选择相邻的

两堆砖块合并成新的一堆，每次合并所花费的代价是这两堆砖块数量之和。请采用动

态规划的思想设计一个算法，以最小代价将所有砖块合并成-一堆。要求: 1) 简要描述

算法的设计思想，2)给出算法的伪代码，3)分析算法的时间复杂性和空间复杂性。

#include<bits/stdc++.h>

using namespace std;

int n;

//贪心哒咩

int main(){

cin>>n;

int a[n+1];

int sum[n+1];

int w[n+1][n+1];//表示区间i，j的最小价值

sum[0]=0;

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

cin>>a[i];

sum[i]=sum[i-1]+a[i];

w[i][i]=0;

}

//从小到大更新，最外层应该枚举区间长度

for(int l=2;l<=n;l++){

for(int i=1;i<=n-l+1;i++){//枚举左端点

int j=i+l-1;//长度确定，右端点也确定了

w[i][j]=100000;

for(int k=i;k<j;k++){//枚举中间k

w[i][j]=min(w[i][j],w[i][k]+w[k+1][j]+sum[j]-sum[i-1]);

}

}

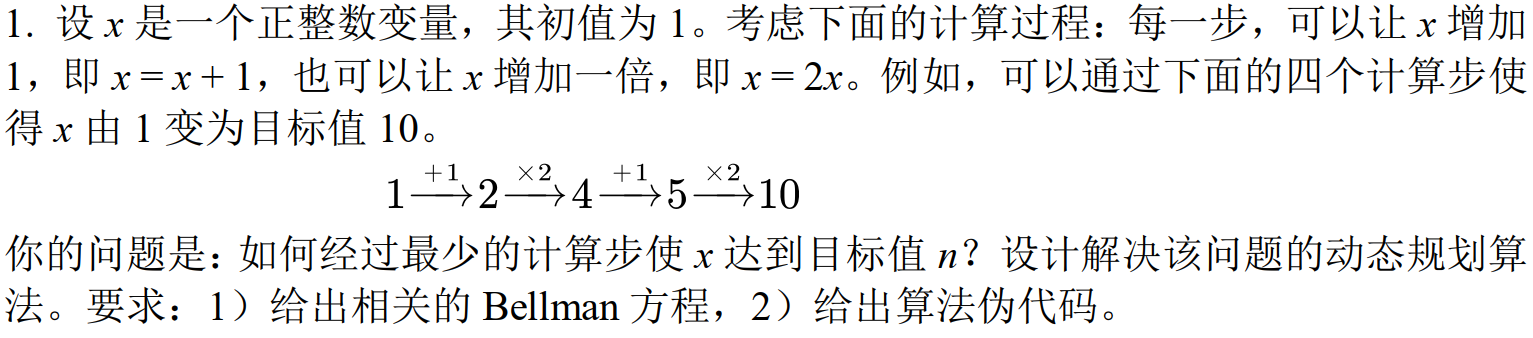
}

cout<<w[1][n];

return 0;

}

9.



**1.设x是一个正整数变量，其初值为1。考虑下面的计算过程:每一步，可以让x增加**

**1，即x=x+ 1，也可以让x增加一倍，即x=2x。例如，可以通过下面的四个计算步使**

**得x由1变为目标值10。**

**1+>2->4+>5->10**

**你的问题是:如何经过最少的计算步使x达到目标值n?设计解决该问题的动态规划算**

**法。要求: 1) 给出相关的Bellman方程，2)给出算法伪代码。**

相关的Bellman方程

令dp[i]表示将x从1变为i所需的最少计算步数，则有：

dp[i] = min(dp[i-1]+1, dp[i/2]+1)

其中dp[i-1]+1表示将x增加1的操作，dp[i/2]+1表示将x乘以2的操作。

算法伪代码

1. 初始化dp[1] = 0

2. for i = 2 to n:

3. if i % 2 == 0:

4. dp[i] = min(dp[i-1]+1, dp[i/2]+1)

5. else:

6. dp[i] = dp[i-1] + 1

7. return dp[n]

时间复杂性和空间复杂性分析

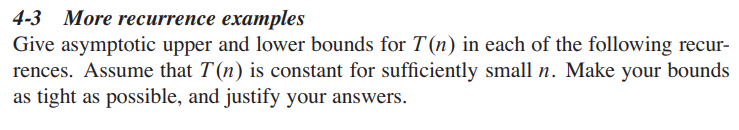
算法中需要遍历从2到n的所有数，并且每个数最多需要进行一次计算操作。因此，算法的时间复杂性为O(n)。此外，需要一个长度为n的数组dp来记录每个数的最少计算步数，因此算法的空间复杂性也为O(n)。

Bellman方程是指将一个最优化问题分解成一系列子问题，并利用各子问题的最优解来推导出整个问题的最优解的方程。这种方法被称为动态规划。Bellman方程的基本形式为：

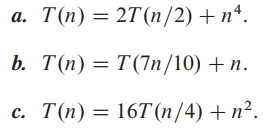
V(i) = max { R(i,j) + βV(j) }

其中，V(i)是状态i的最优值，R(i,j)是从状态i转移到状态j时的即时奖励（或代价），β是折扣因子（用于平衡即时奖励和未来奖励的重要性）。方程的求解通常采用迭代的方式，从某个状态的初始值开始，依次求解出各状态的最优值，直到收敛到最终的最优解。

**10．**







4-3

More recurrence examples

Give asymptotic upper and lower bounds for T(n) in each of the following recur-

rences. Assume that T (n) is constant for sufficiently small n. Make your bounds

as tight as possible, and justify your answers.

a. T(n) = 4T(n/3) + n lgn.

b. T(n)= 2T(n/2) +n^4.

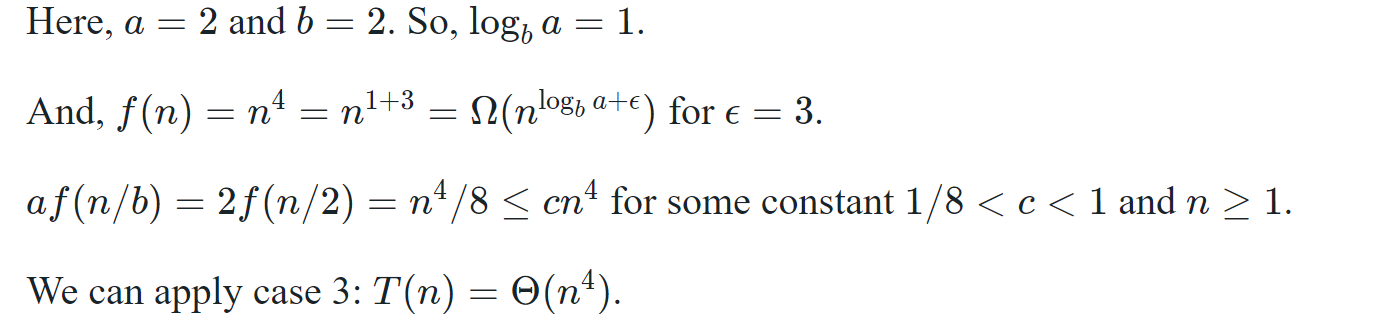
c. T(n) = T(7n/10) + n.

d. T(n)= 16T(n/4) +n^2.

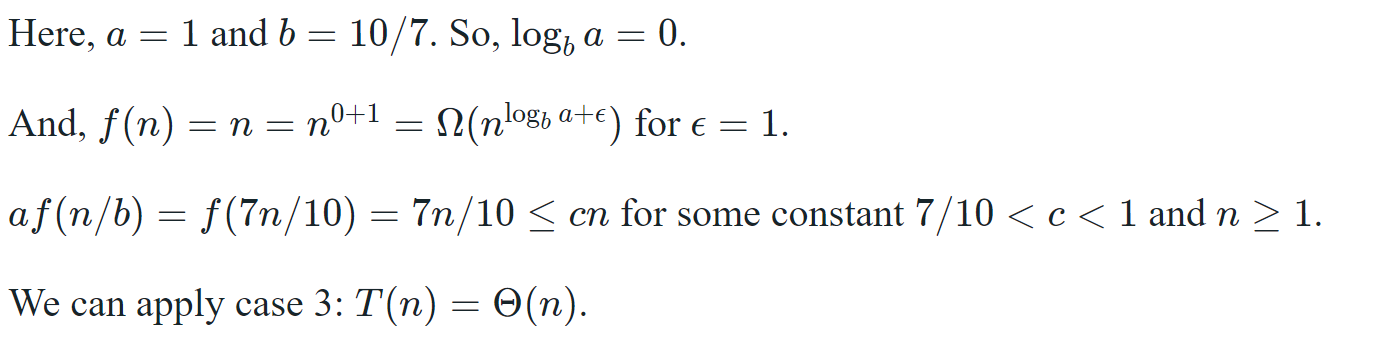
**a.**



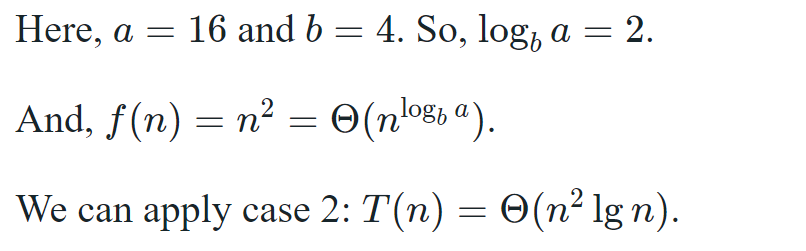
**b.**



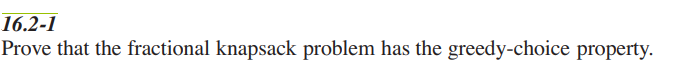
**C**



**D**



11.

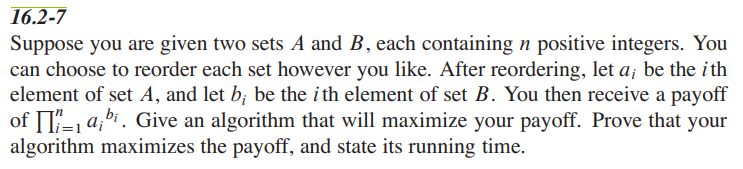


Prove that the fractional knapsack problem has the greedy-choice property.

证明分数背包问题具有贪婪选择性质。

A optimal solution to the fractional knapsack is one that has the highest total value density. Since we are always adding as much of the highest value density we can, we are going to end up with the highest total value density. Suppose that we had some other solution that used some amount of the lower value density object, we could substitute in some of the higher value density object meaning our original solution could not have been optimal.

12.



Suppose you are given two sets A and B, each containing n positive integers. You

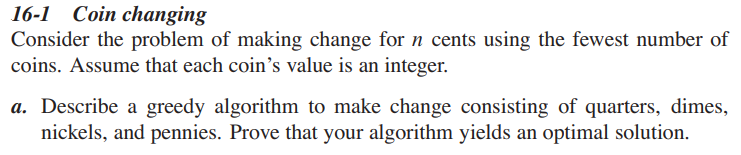
can choose to reorder each set however you like. After reordering, let ai be the *i*th

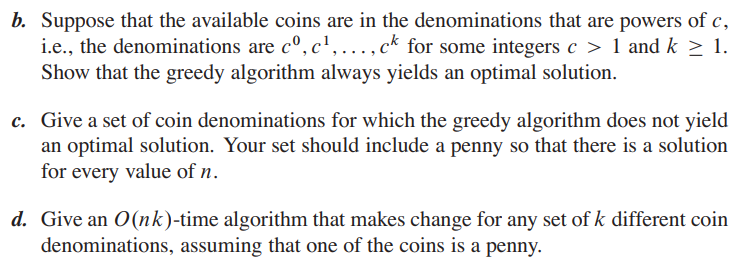
element of set A, and let bi be the ith element of set B. You then receive a payoff

of .Give an algorithm that will maximize your payoff. Prove that your

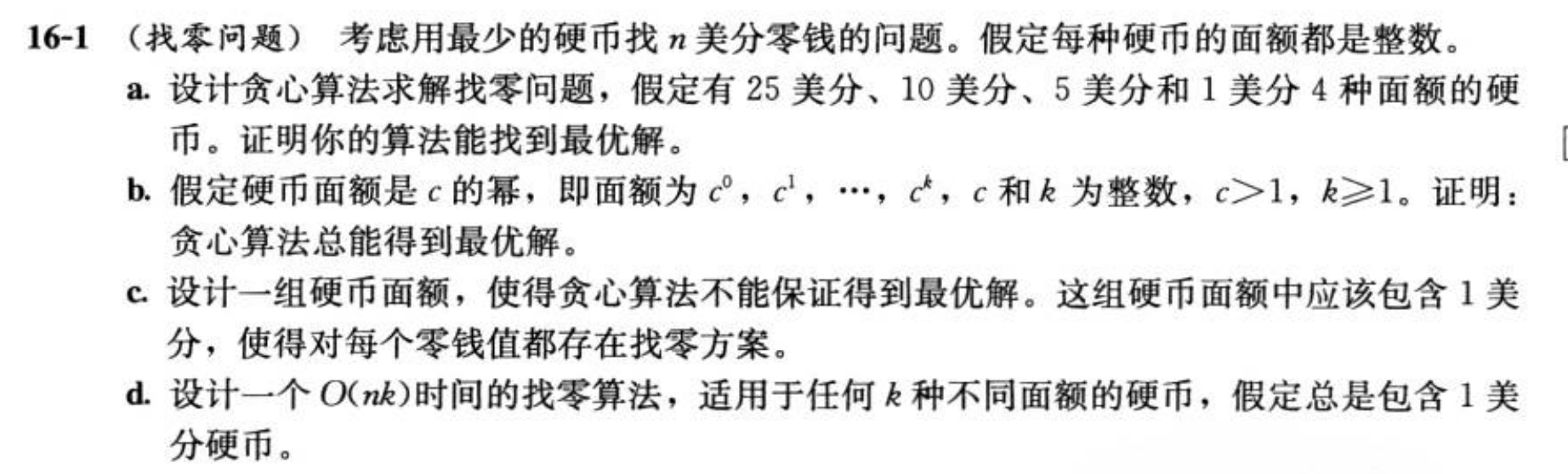
algorithm maximizes the payoff, and state its running time.

13.





（a和d写伪代码，d用自底向上写，时间复杂度不做要求；b给出证明，c举例即可）

****