8. 关于这个问题，我们将调查在**稳定匹配问题**且特别在 G-S 算法中的真实性问题. 基本的问题是：一个男人或一个女人通过对他或她的优先表说谎能够得到更好的结果吗？更具体地说，我们假设每个参与者有一个真实的优先顺序. 现在考虑女人 w. 假设 w 更偏爱男人 m 而不爱 m'，但是 m 和 m' 在她的优先表上都是低的. 通过交换 m 和 m' 在她的优先表上的次序（即通过她更偏爱 m' 而不爱 m 的假声明）并且用这个假优先表运行算法，在结束时 w 将拥有比 m 与 m' 真的更偏爱的男人 m''. 这种情况可能发生吗？（我们可以对男人问同样的问题，但是，对于这个问题，我们将集中考虑女人的情况）

通过做下面两件事中的一件重新求解这个问题：

（a）证明对任何一组优先表，交换表中一对人的顺序不可能改善 G-S 算法中一个女人的伴侣；或者

（b）给出一组优先表的例子. 对于它存在一个交换，且它将能够改善交换了优先表的女人的伴侣.

==========

答：不可能

首先给出算法步骤描述：

第一轮，每个男人都选择自己名单上排在首位的女人，并向她表白。这种时候会出现两种情况：（1）该女士还没有被男生追求过，则该女士接受该男生的请求。（2）若该女生已经接受过其他男生的追求，那么该女生会将该男士与她的现任男友进行比较，若更喜欢她的男友，那么拒绝这个人的追求，否则，抛弃其男友……

第一轮结束后，有些男人已经有女朋友了，有些男人仍然是单身。

在第二轮追女行动中，每个单身男都从所有还没拒绝过他的女孩中选出自己最中意的那一个，并向她表白，不管她现在是否是单身。这种时候还是会遇到上面所说的两种情况，还是同样的解决方案。直到所有人都不在是单身。

下面给出程序代码：

```
#include<iostream>
#include <stack>

using namespace std;

#define NUM 4
#define NIL -1

int GetPositionFromLaday(int ladayArray[][NUM], int laday, int man)
{
    for(int i=0; i<NUM; i++)
        if(ladayArray[laday][i] == man)
            return i;
    return NIL;
}
```

```
void ChoosePartener(stack<int>& manStack, int manPos, int manArray[][NUM], int ladayArray[][NUM], int manPerfer[], int manStartPos[], int ladayNow[])
{
    //选择自己名单上排在首位的女人
        int perferLaday = manArray[manPos][manStartPos[manPos]];
        //如果该女孩没有接受过表白，则接受该男孩的表白
        if(ladayNow[perferLaday] == NIL)
        {
            ladayNow[perferLaday] = manPos;
            manPerfer[manPos] = perferLaday;
        }
        //如果已经有人向她表白，则判断其现在拥有的有没有现在追求的好
        else
        {
            int oldPos = GetPositionFromLaday(ladayArray, perferLaday, ladayNow[perferLaday]);
            int newPos = GetPositionFromLaday(ladayArray, perferLaday, manPos);
            if(oldPos < newPos)
            {
                manStartPos[manPos]++;//说明该女生更喜欢现在拥有的，选心目中第二位
                //加入单身行列
                manStack.push(manPos);
            }
            else //换男友
            {
                //被甩的男友恢复自由身份
                manStartPos[ladayNow[perferLaday]]++;
                //加入单身行列
                manStack.push(ladayNow[perferLaday]);
                //将追求的男士改为现任男友
                ladayNow[perferLaday] = manPos;
                manPerfer[manPos] = perferLaday;
            }
        }
}

int main()
{
    int manArray[NUM][NUM] ={{2,3,1,0},{2,1,3,0},{0,2,3,1},{1,3,2,0}};
    int ladayArray[NUM][NUM] = {{0,3,2,1},{0,1,2,3},{0,2,3,1},{1,0,3,2}};

    int manPerfer[NUM] = {0};//每位男生选中的女生
    int manStartPos[NUM] = {0};//记录每位男生选取的是心目中第几位的女生
    int ladayNow[NUM] = {NIL,NIL,NIL,NIL};//女生对应的男生
```

```
    stack<int> manStack; // 还处于单身的男士

    //进行第一轮迭代，每个男生都选择自己名单上排在首位的女生。
    for(int pos=0; pos<NUM; pos++)
    {
        ChoosePartener(manStack, pos, manArray, ladayArray, manPerfer, manStartPos,ladayNow);
    }

    while(manStack.size()!=0)
    {
        int manPos = manStack.top();
        manStack.pop();
        ChoosePartener(manStack, manPos, manArray, ladayArray, manPerfer, manStartPos,ladayNow)
;
    }

    for(int i =0;i<NUM; ++i)
        cout<<"Man NO.: "<<i<<" Laday NO.: "<<manPerfer[i]<<endl;
}
```

我们注意到：

```
    int manArray[NUM][NUM] ={{2,3,1,0},{2,1,3,0},{0,2,3,1},{1,3,2,0}};
    int ladayArray[NUM][NUM] = {{0,3,2,1},{0,1,2,3},{0,2,3,1},{1,0,3,2}};
```

结果如下：

```
[Running] cd "c:\Users\wgl002\Desktop\"
Man NO.: 0 Laday NO.: 2
Man NO.: 1 Laday NO.: 1
Man NO.: 2 Laday NO.: 0
Man NO.: 3 Laday NO.: 3
```

我们任意交换女方的优先表

```
    int manArray[NUM][NUM] ={{2,3,1,0},{2,1,3,0},{0,2,3,1},{1,3,2,0}};
    int ladayArray[NUM][NUM] = {{0,3,1,2},{0,1,2,3},{0,2,3,1},{1,0,3,2}};
```

结果如下：

```
[Running] cd "c:\Users\wgl002\Desktop\"
Man NO.: 0 Laday NO.: 2
Man NO.: 1 Laday NO.: 1
Man NO.: 2 Laday NO.: 0
Man NO.: 3 Laday NO.: 3
```

没啥改变，故不可能~

3. 用下面的函数表，按照增长率上升的顺序排列它们. 即，如果在你的表中，函数 $g(n)$ 紧跟在 $f(n)$ 的后面，那么应该满足 $f(n)$ 是 $O(g(n))$.

$$f_1(n) = n^{2.5}$$
$$f_2(n) = \sqrt{2n}$$
$$f_3(n) = n + 10$$
$$f_4(n) = 10^n$$
$$f_5(n) = 100^n$$
$$f_6(n) = n^2 \log n$$

==========

答：表格上层的函数比下层的函数增长要快

函数名	函数体
F5	100^n
F4	10^n
F1	$n^{2.5}$
F6	$n^2 \log n$
F3	$n+10$
F2	$\sqrt{2n}$

4. 用下面的函数表，按照增长率上升的顺序排列它们. 即，如果在你的表中，函数 $g(n)$ 紧跟在 $f(n)$ 的后面，那么应该满足 $f(n)$ 是 $O(g(n))$.

$$g_1(n) = 2^{\sqrt{\log n}}$$

$$g_2(n) = 2^n$$

算法设计

$$g_3(n) = n^{4/3}$$

$$g_4(n) = n(\log n)^3$$

$$g_5(n) = n^{\log n}$$

$$g_6(n) = 2^{2^n}$$

$$g_7(n) = 2^{n^2}$$

==========

答：表格上层的函数比下层的函数增长要快

函数名	函数体
G6	2^{2^n}
G7	2^{n^2}
G2	2^n
G1	$2^{\sqrt{\log n}}$
G5	$n^{\log n}$
G3	$n^{4/3}$
G4	$n(\log n)^3$

6. 考虑下面的基本问题. 给定由 n 个整数 $A[1],A[2],\cdots,A[n]$ 组成的数组 A. 你想输出一个 2 维的 $n\times n$ 的数组 B，其中 $B[i,j](i<j)$ 包含数组项 $A[i]$ 到 $A[j]$ 的和——即和 $A[i]+A[i+1]+\cdots+A[j]$. (只要 $i\geqslant j$，数组的项 $B[i,j]$ 的值不用指定，因此不管这些值的输出.)

这里是求解这个问题的一个简单的算法.

```
For i=1,2,…,n-1
   For j=i+1,i+2,…,n
     把数组元素从 A[i]加到 A[j]
     把这个结果存储到 B[i,j]
   Endfor
Endfor
```

(a) 针对某些你应该选择的函数 f，对这个算法给出一个形为 $O(f(n))$ 的在规模为 n 的输入上运行时间的界(即这个算法所执行的操作个数的界).

(b) 对同一个函数 f，证明算法在规模为 n 的输入上的运行时间也是 $\Omega(f(n))$(这证明了在运行时间上的一个 $\Theta(f(n))$ 的渐近的紧的界).

(c) 尽管你在(a)和(b)分析的算法是求解这个问题的最自然的方法——毕竟，它仅仅在数组 B 的有关的项上迭代，并对每项填入一个值——但是它包含了某些非常不必要的成分导致了它的效率低下. 给出一个求解这个问题的不同算法，使得它具有渐近的更好的运行时间. 换句话说，你应该设计一个具有 $O(g(n))$ 运行时间的算法，其中 $\lim_{n\to\infty} g(n)/f(n)=0$.

==========

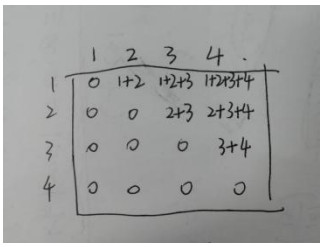
答:

（a）这个算法所执行的操作个数为(n-1) + (n-2) + (n-3)+.....+1 = $\sum_{i=1}^{n-1} i$ = $\frac{n^2}{2}$

算法运行时间的界为 O(n^2)

（b）存在正常量 c1、n0，使得对所有 n>=n0，有 0<=c1g(n)<=f(n)，故算法在规模为 n 的输入上的运行时间也是Ω（f(n)）

（c）



由图其实可以发现

B[1][2] = A[1] + A[2]
B[1][4] = A[1]+A[2]+A[3]+A[4]
.....
B[3][4] = A[3]+A[4]

故提出新算法：O(n)

```
Sum = 0;
For m = i ; m <= j ; m++
      Sum += A[m];
B[i][j] = sum;
```