1296. Shuffle

给定一个长度为 n 的排列，元素标号为 $1 \ldots n$。你需要若干次操作将这个排列排序为 $1, 2, \ldots, n$，每次一操作是一系列“并行的”交换：你可以选择若干个**不相交**的元素对，将他们分别交换。
请构造一种方案，使得操作的次数尽可能少。

Key Idea:找环。找到环之后，如果是 2 层小环，直接结束。如果超过此规模，2 轮置换必然能够结束（显然）

Chapter: 分治算法

1299.Closest Pair 最近点对问题

1300. k-th Smallest Number 第 k 小的数

这一题采用的思想是 mid-of-mid，将一组数分成 A,B,C(A<k,B=k,C>k)三组后直接去找。

1301. Bubbling bubbles 计算冒泡排序的交换次数

这一题本质上是魔改归并排序，需要注意冒泡排序每次交换都会使得逆序对恰好减 1，因此就是求逆序对，只是贡献在元素上记录。

1302. Optimal sort 侧重在理解归并排序的“大小比较”只需要知道两个元素的大小关系

1304. Probabilistic killer 是想通过一种暴力的随机方法过了 1299 的 closest pair 问题

Chapter: 图论相关

1342. Shortest path 无向图带权值的单源最短路

给一个 $n(1 \le n \le 2500)$ 个点 $m(1 \le m \le 6200)$ 条边的无向图，求
数据保证，s 出发可以到达 t。

跑 Dijkstra 算法直截了当，而且满足复杂度。

1343. 0-1 Shortest path 零一最短路，最直接的方式就是 BFS。

BFS 找最短路使用双端队列，其中距离 0 的都放前面距离 1 的都放后面然后跑一遍 BFS 即可。核心代码：

```
void bfs(){
    vis[s] = true;
    for(int i=0;i<graph[s].size();i++)
        if(graph[s][i].value == 0)
            deq.push_front(graph[s][i]);
        else
            deq.push_back(graph[s][i]);
    while(!deq.empty()){
        node temp = deq.front();
        deq.pop_front();
        if(vis[temp.to])
            continue;
        vis[temp.to] = true;
        dis[temp.to] = min(dis[temp.to], dis[temp.from] + temp.value);
        for(int i=0;i<graph[temp.to].size();i++)
            if(graph[temp.to][i].value == 0)
                deq.push_front(graph[temp.to][i]);
            else
                deq.push_back(graph[temp.to][i]);
    }
}
```

1344. Negative cycle 有向图找负环并且输出的经典问题

给一个 $n(1 \le n \le 2500)$ 个点 $m(1 \le m \le 6200)$ 条边的**有向图**，边权
你需要判断图中是否存在负环。如果有，请输出任意一个负环。

负环的查找用 Bellman-Ford，输出用记录前驱的方式即可。第一层循环直接更新距离。第二层循环是第 n 次，如果还能更新说明有问题。第三层循环找所有环的前驱节点并准备输出。

```
bool BellmanFord(){
        memset(dis, 0x3f3f3f3f, sizeof(dis));
        for(int i=1;i<n;i++)
                for(int j=0;j<m;j++)
                        if(dis[edge[j].to] > dis[edge[j].from] + edge[j].value){
                                dis[edge[j].to] = dis[edge[j].from] + edge[j].value;
                                pre[edge[j].to] = edge[j].from;
                        }

        bool flag = false;
        for(int j=0;j<m;j++)
                if(dis[edge[j].to] > dis[edge[j].from] + edge[j].value){
                        flag = true;
                        dis[edge[j].to] = dis[edge[j].from] + edge[j].value;
                        pre[edge[j].to] = edge[j].from;
                }
        for(int j=0;j<m;j++)
                if(dis[edge[j].to] > dis[edge[j].from] + edge[j].value){
                        int tmp = edge[j].to;
                        for(int i=1;i<=n;i++){
                                ans.push_back(tmp);
                                tmp = pre[tmp];
                        }
                        break;
                }
        return flag;
}
```

1341. 2-SAT 这题是 SAT 的弱化版本

对于一系列形如 $(x_i = a) \vee (x_j = b)$ 的约束，其中 $a, b \in 0, 1$，你需要判断是否存在一组 x_i 使其同时满足。如果有，请输出一组解。

本质上就是对所有的约束条件建边并且找强连通分量的存在。事实上拓扑序越大的应该越容易被满足，因此在 $x_i = 0$ or 1中找个拓扑序大的输出即可。关键在于有向图怎么找强连通分量：DFS 找到 finish time 后在逆图中逆序输出结果。

1345. Strict k-th shortest path 严格 k 短路

给一个 $n(1 \le n \le 10^5)$ 个点 $m(1 \le m \le 2 \times 10^5)$ 条边的**有向图**，求 s 到 t 的严格 $1, 2, \ldots, k(k = 10)$ 短路的长度。
严格 k 短路定义为：

- $k = 1$：最短路。
- $k > 1$：严格长于 $k - 1$ 短路的路径中最短的一条。

在这里，路径上的点可重，这种路径在一些地方被称为途径 (walk)。
数据保证，s 出发可以到达 t。

A*算法或者魔改后的 Dijkstra 也可以过。

Chapter: 贪心与 DP

1428. Minimum Spanning Tree

Prim 或者 Kruskal 都能过

1429. Holiday scheduling: episode 1 游玩代价太大

只需要专心工作，因此是贪心的按照最早开始的来做就行。

1430. Holiday scheduling: episode 2 可以游玩或打工

// f[i][0] 出去工作了 i 天的最大收益
那么会由啥也不干、玩一天、工作一天 3 个状态转移

1431. Knapsack 3 in 1 背包数量比较特殊

需要将背包容量转化成二进制后变成 01 背包
01 背包容量必须从大到小枚举函数 f(容量从 w 到 0)

1432. Colorful inversion

为什么就是最长下降子序列长度：因为每个下降子序列是一个团，颜色必须互不相同，因此染为最长下降子序列长度是必要的。同时因为最长下降子序列长度满足条件，所以亦是充分的。
最长下降子序列长度就是 LIS 问题，结合单调队列能很快。

1433. Bo

本题解决的核心思路就是利用可靠的撤销操作，利用贪心完成本题。这种套路有时被称为反悔贪心。
具体实现时，我们每当完成一次交易时，将交易卖出的价

格作为一个可买入的价格放入集合里，而以后购买这个价格代表的是：撤销这一次交易，并将卖出价格更新为当前的。

而带反悔的贪心是这样的：

```
for i = 1 to n:
    if prices.min() < a[i]:
        ans += a[i] - prices.min()
        prices.erase_min()
        prices.add(a[i])
    prices.add(a[i])
```

可以注意到的是，如果某次卖出成功了，会有两个 a[i] 放进集合里，含义其实分别是撤销一次交易和买入撤销后没有操作的 a[i]。更抽象地来说，把本题写成整数规划的形式，相当于把 $x_i = 1$ 变为了 0 和 -1，从堆里能够取出一次则值减一。

1434. Fair division

你——伟大的赏金猎人，经过了史诗般的战斗后，夺取了属于你的战利品——一个重量为 w 克的金块。

你的 n 个盾从功不可没。根据每个人自己汇报的功劳大小，你决定给第 i 个盾从 a_i 重量的金块，其中 $\sum_i a_i = w$。

在以枪火为筹码谈生意的比尔吉沃特，没有人做慈善。如果你请比灌商行老板帮你分金块，你需要足额支付手续费。如果一个金块在切割前重量为 x，你需要为这次分割付出 $x \times p\%$ 枚的银蛇币，之后金块可以以任意比例被切为两块。

你需要在满足所有盾从的要求的情况下，支付尽可能少的银蛇币。由于银蛇币不可分割，你只需要在最后一次性支付所有切割代价的和**向上取整**枚银蛇币即可。

反方向的合并果子过程

1449. Maximum flow

套用 Dinic's 或者 Ford-Fulkerson 都能过

```
int fordFulkerson(){
    int ans = 0;
    int u;
    while(BFS() == true){
        int capacity = INT_MAX;
        for(int i=t; i!=s; i=parent[i]){
            u = parent[i];
            capacity = min(capacity, rgraph[u][i]);
        }
        for(int i=t; i!= s; i=parent[i]){
            u = parent[i];
            rgraph[u][i] -= capacity;
            rgraph[i][u] += capacity;
        }
        ans += capacity;
    }
    return ans;
}
```

```
bool make_level(){
    for(int i=1;i<=n;i++)
        level[i] = INF;
    queue<int>q; q.push(source); level[source] = 0; cur[source] = list_head[source];
    while(!q.empty()){
        int x = q.front();q.pop();
        for(int i = list_head[x];i;i = e[i].next){
            int y = e[i].to;
            if(e[i].flow > 0 && level[y] == INF){
                q.push(y);
                cur[y] = list_head[y];
                level[y] = level[x] + 1;
                if(y == sink) return true;
            }
        }
    }
    return false;
}
int dfs(int x, int flow) {
    if (x == sink) return flow;
    int used = 0;
    for (int &i = cur[x]; i != 0; i = e[i].next) // i : reference
        if (e[i].flow > 0 && level[x] + 1 == level[e[i].to]) {
            int ret = dfs(e[i].to, min(e[i].flow, flow - used));
            used += ret;
            e[i].flow -= ret;
            e[i ^ 1].flow += ret;
            if (used == flow) break;
        }
    return used;
}
int dinic() {
    int ans = 0;
    while (make_level()) {
        for (int i = 1; i <= n; i++) // initialize cur[]
            cur[i] = list_head[i];
        ans += dfs(source, INF);
    }
    return ans;
}
```

1450. Underdetermined

给定一个 $n \times n$ 的 01 矩阵 A，现在你需要判断：是否能够矩阵中某些位置上的 1 变成 0，使得这个矩阵的行列式不为 0。

关键需要建立一个网络图的关系，每行仅一个每列仅一个

1451. Life on a tree

交大有 n 幢宿舍楼。由于大家生活在树上，因此 n 幢宿舍楼之间的通路形成一棵树，每条边的边权都是 1。

为了能及时挽救被半期考试折磨的可怜人，现在需要选择一些宿舍楼建立医疗点存放足量的速效救心丸。选择的要求是：

- 每幢宿舍楼都能到达一个距离不超过 k 的医疗点。
- 在这个前提下，建立的医疗点数量尽可能少。

请输出一种可行的方案。

朴素方法：从底层开始染色，染色过程中对受影响的节点去 DFS 染色，总复杂度大概在 $n^{3/2}$

1452. If only

有一个玩家在一张无向图上玩游戏，玩家现在在 s 号点，他希望在 k 秒后恰好移动到 t 号点。玩家每秒必须移动到相邻的结点上。

你需要判断：如果玩家采取最优策略，是否能达成目标。

如果玩家某时刻无法移动，亦视为任务失败。

把一个点拆成奇数点和偶数点后跑 BFS 即可。

1453. Only if

有 p 个玩家在一张无向图上玩游戏，i 号玩家现在在 s_i 号点。每个玩家希望在 k 秒后恰好移动到 $t_1, t_2, \cdots, t_p$ 中的一个点上。每个玩家每秒开始时必须移动到相邻的点上，并且两个玩家不能在移动后处在同一个点上。

你需要判断：如果玩家们合作并且采取最优策略，是否能让所有玩家达成目标。

如果存在玩家某时刻无法移动，亦视为任务失败。

把这个问题拆成 t 层的网络流问题即可。为了保证每个节点只能被一个节点独占，加入一条单向的边就可以了。

1454. Xingqiu at your service

> 蛟龙易斩，雨线难画。

行秋有两个技能，小技能「画雨笼山」与大招「裁雨留虹」。其中，小技能可以给大招充能，并且两次小技能恰好能充满一个大招所需的元素能量。释放大招后，元素能量清空。

行秋拥有一把叫做祭礼剑的武器，这把武器在释放小技能时有 p 的概率立刻刷新小技能的冷却时间，但被刷新的小技能不能再次触发刷新。我们把有概率刷新的小技能称为主动释放的小技能，被刷新的小技能称为免费释放的小技能。

行秋的大招能量现在是空的，并且他想释放 n 次大招。现在请问，期望下他要主动释放多少次小技能？

我们假定忽略掉冷却时间、施法时间、技能持续时间等游戏中的限制。这意味着，元素能量充满时大招就可以释放，并且可以在主动释放与免费释放的两次小技能间释放大招。

有数学方法

```
int main() {
    double p;
    int n;
    cin >> n >> p;
    n *= 2;
    double e = n/(1+p) + p/(1+p)/(1+p)*(1-pow(p, n));
    printf("%.9lf", e);
}
```

当然如果想不到数学方法可以用矩阵快速幂去计算递推过程（类似斐波那契数列的计算）

```
scanf("%lld%lf",&nn,&pp);
__float128 p = pp;
if(nn==1){
    cout<<setprecision(20)<<(long double)(2-p)<<endl;
    return 0;
}
f[0] = 0; f[1] = 1; f[2] = (1-p) * f[1] + 1;
x.mat[0][0] = 1-p;  x.mat[0][1] = p;    x.mat[0][2] = 1;    x.mat[1][0] = 1;
res.mat[0][0] = 1;  res.mat[1][1] = 1;  res.mat[2][2] = 1;
nn*=2;  nn-=2;
while(nn){
    if(nn&1)    res=res*x;
    nn>>=1;
    x=x*x;
}
goal.mat[0][0] = f[2];  goal.mat[1][0] = 1; goal.mat[2][0] = 1;
goal = res*goal;
cout<<setprecision(20)<<(long double)goal.mat[0][0]<<endl;
```