1.

折半查找的平均时间复杂度为 O(logn),顺序查找的平均时间复杂度为 O(n)，而该有序数组含 1000000 个元素，则折半查找平均比顺序查找快 $\frac{1000000}{\log_2(1000000)} = 50171.666$ 倍

2.

因为每次渡河最多只能有一位士兵，所以可利用减治法，N 位士兵过河的问题划分为 1 位士兵先过河，后 N-1 位士兵再过河。

假设两个小男孩称为 A 和 B，士兵和小男孩初始都在河的左岸

① A，B 从左岸出发，划到右岸，B 下船（去程）

② A 单独返回左岸（返程）

③ A 下船，一位士兵上船到右岸（去程）

④ B 从右岸上船，到左岸与 A 汇合（返程）

⑤ 循环上述 1-4 步，将剩下 N-1 位士兵渡过河

要使得 N 位士兵全部渡河，并且 A 和 B 最后共同操纵船只，则上述 1-4 步需要循环 N 次

即：往返 2N 次（岸与岸之间横渡 4N 次）

3.

a. 前序法：abdecf

b. 中序法：dbeacf

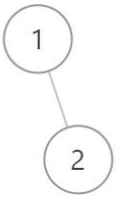
c. 后序法：debfca

4.

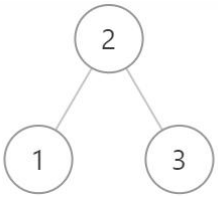
a.

① 插入 1：

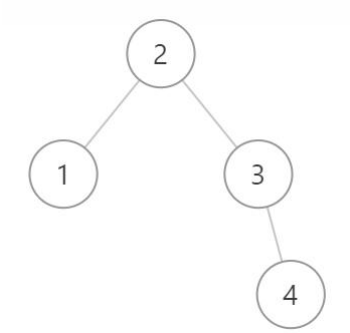
② 插入 2：



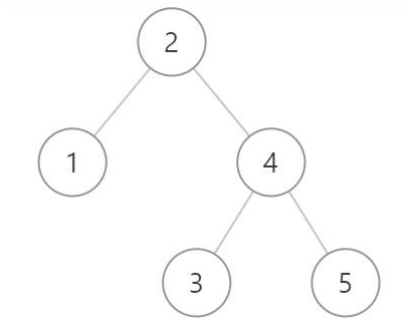
③ 插入 3：



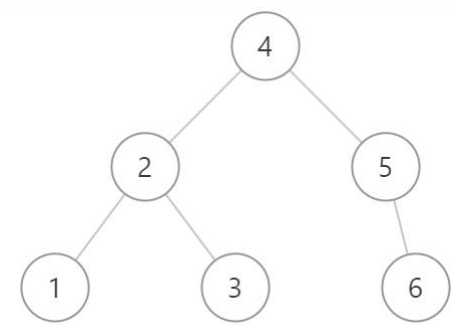
④ 插入 4:



⑤ 插入 5:



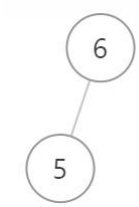
⑥ 插入 6:



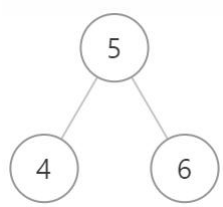
b.

① 插入 6:

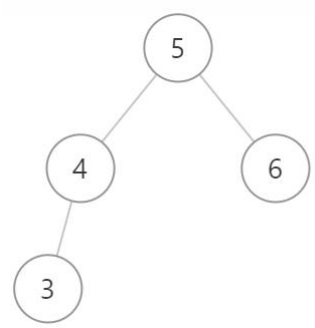
② 插入 5:



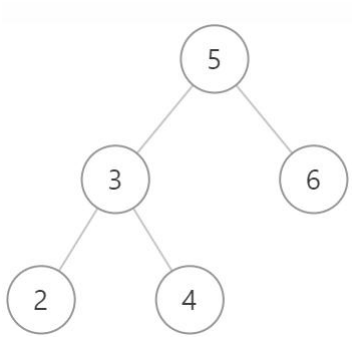
③ 插入 4:



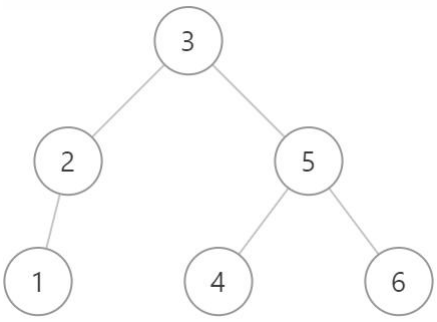
④ 插入 3:



⑤ 插入 2:



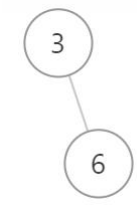
⑥ 插入 1:



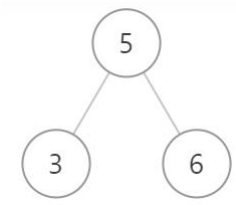
c.

① 插入 3:

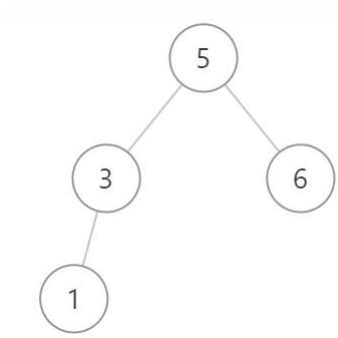
② 插入 6:



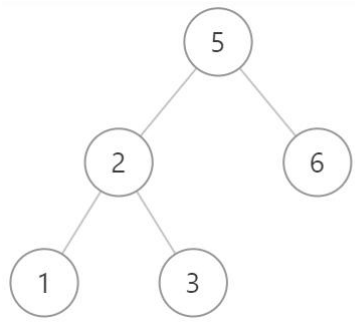
③ 插入 5:



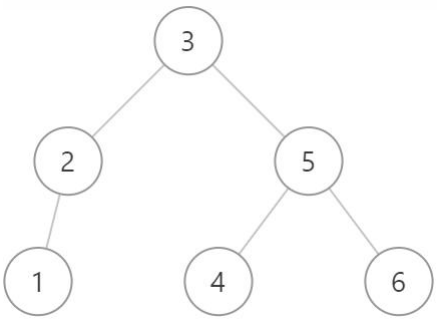
④ 插入 1:



⑤ 插入 2:

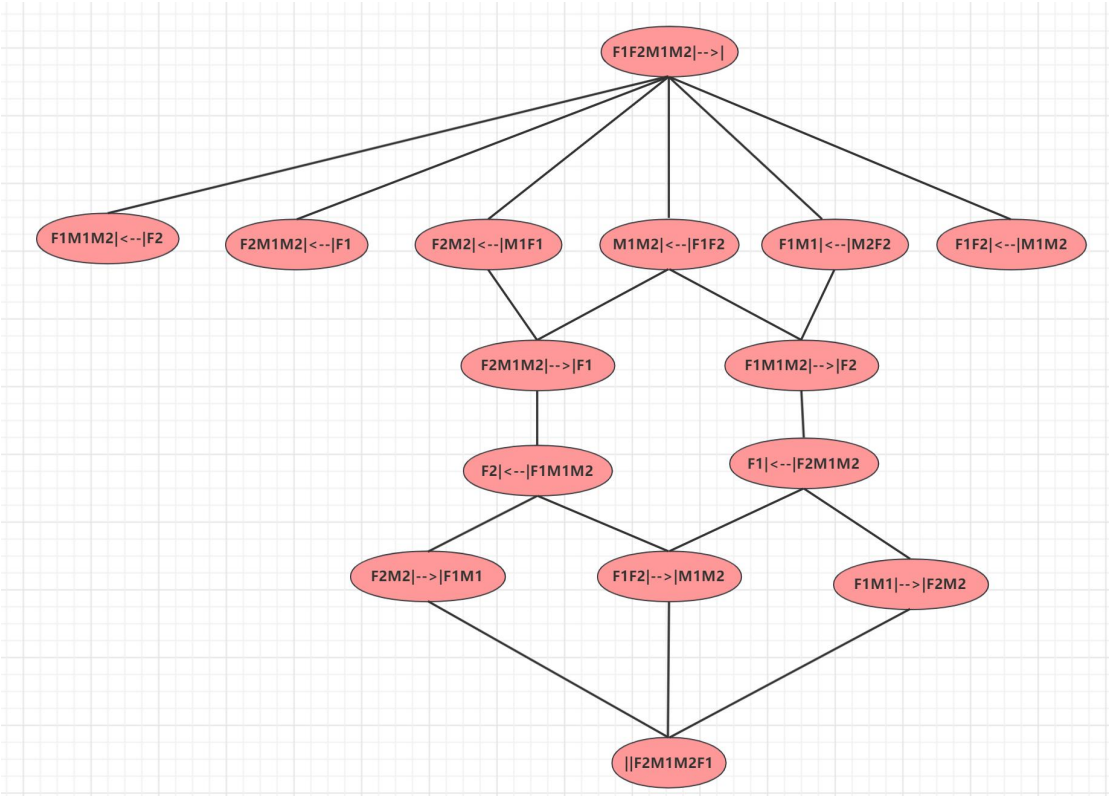


⑥ 插入 4:



5.

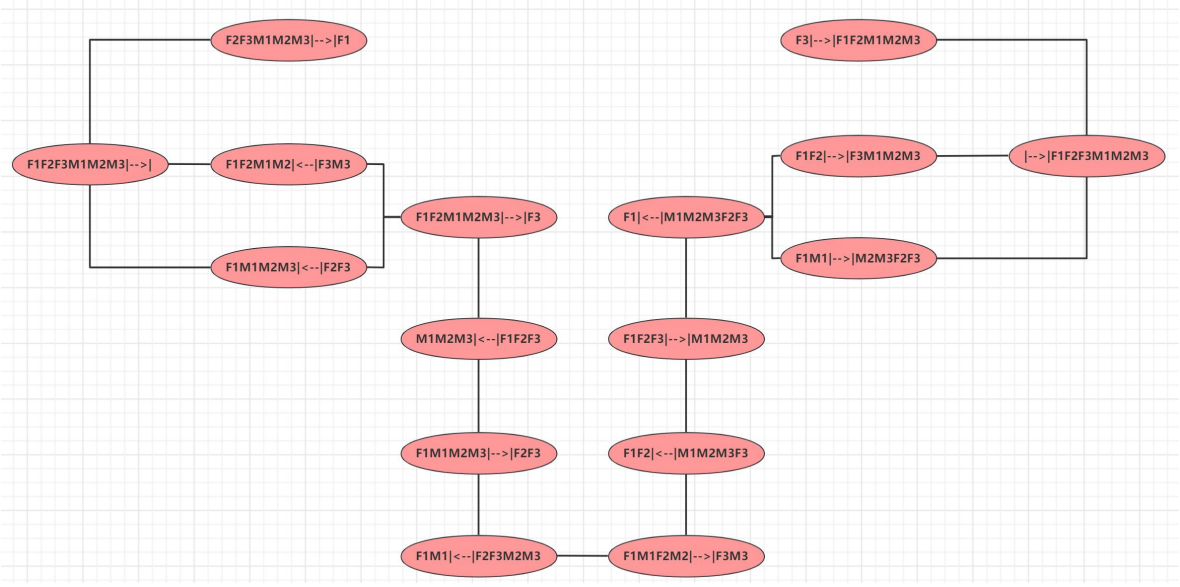
设每对夫妇的下标相同，妻子为 F，丈夫为 M（例如，第一对夫妇中妻子为 F1，丈夫为 M1）
a. 当 n=2 时
状态时空图如下:



由图可知，从左岸到右岸的可行路径有 8 条，即对应 8 种可能情况，每种情况过河次数为 5 次

b. 当 n=3 时

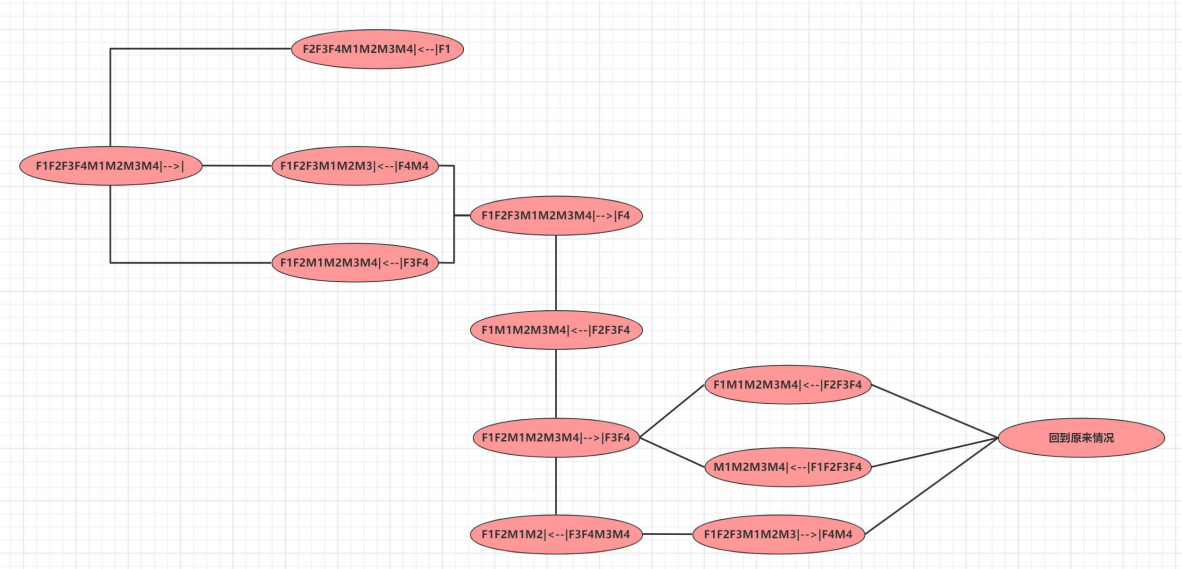
部分情况已省去，核心状态时空图如下：



由图可知，从河左岸到右岸的过河次数为 11 次

c. 当 n>=4 时，无解

n=4 时的时空图如下：



由图可知，当 n>=4 时，在经过 6 次过河后只要在丈夫不吃醋的情况下，第七次过河都会回到原来已有的情况，此问题无解。

编程题：

算法思路：

可以观察到，一个矩阵右上角的元素是该行最大的元素，又是该列最小的元素。

不断将右上角元素与目标值比较：

① 若由上角元素和目标值相等，则退出返回 true

② 若右上角元素大于目标值，则可以去掉最后一列，在子矩阵中继续寻找

③ 若右上角元素小于目标值，则可以去掉最前面一行，在子矩阵中继续寻找

若最后矩阵减小为空，则说明找不到目标值，返回 false

时间和空间复杂度：

时间复杂度：O(n+m)，在每次寻找目标值的过程中，若没有找到目标值，则要么减少一行，要么减少一列，最多减少 m+n 次行或列。

空间复杂度：O(1)

补充函数代码截图：

```cpp
bool searchMatrix(vector<vector<int> >& matrix, int target){
    //从右上角来看，是一颗二叉搜索树
    //对于右上角的元素来说，该行往左依次减小，该列往下依次增大
    //若targrt<右上角元素，则去掉第n-1列
    //若target>右上角元素，则去掉第0行
    //直到target==右上角元素

    //先判断矩阵是否为空
    int m = matrix.size();
    if (m == 0)//一行都没有
        return false;
    int n = matrix[0].size();
    //m行n列
    int i = 0, j = n - 1;//从第0行第n-1个数开始找
    while (i < m && j >= 0) {
        if (matrix[i][j] == target)
            return true;//找到目标值
        else if (matrix[i][j] < target)//右上角元素比目标值小
            i++;//去掉矩阵最前面一行
        else if (matrix[i][j] > target)//右上角元素比目标值大
            j--;//去掉矩阵最后一列
    }
    return false;//退出循环，说明没找到
}
```

运行结果：