```
basicFiles.sort((f1, f2) -> { // L0 文件实质上内部根据 flush 时间分成了多层
    int majorCampare = f1.getMajorId() - f2.getMajorId();
    if (majorCampare != 0) {
        return majorCampare; // 先按照 flush 时间排序
    } else {
        return (int) (f1.getFileId() - f2.getFileId()); // 再按照主键顺序排序
    }
});

if (level1Files.size() == 0) { // 当 L1 为空，则可以将所有 L0 文件同时当作输入源
    upperLevelFiles = basicFiles;
    lowerLevelFiles = null;
} else { // 否则就根据 L1 的文件主键范围，拟定从 L0 中选取的文件
    double ra = Double.MAX_VALUE;

    for (int viceCount = 0; viceCount <= level1Files.size(); viceCount++) {
        for (int ind = 0; ind <= level1Files.size() - viceCount; ind++) {
            int leftInd = ind, rightInd = ind + viceCount - 1;
            byte[] minRowKey = …; // 根据选取的 L1 文件，计算出 L0 文件主键最小值
            byte[] maxRowKey = …; // 根据选取的 L1 文件，计算出 L0 文件主键最大值

            long viceSize = 0L;

            // calcPlan 实现了“L0 文件选择算法”
            List<FileMeta> thisPlan = calcPlan(minRowKey, maxRowKey, basicFiles);
            if (thisPlan.size() > 0) {
                long currentLevelSize = 0L;
                for (FileMeta l0Pick : thisPlan) {
                    currentLevelSize += l0Pick.getFileSize();
                }
                // 每次计算选取计划的读扩大率，比较以找到能最小化读扩大率的选取方式
                double thisRA = viceSize * 1.0 / currentLevelSize;
                if (lowerLevelFiles == null || thisRA < ra) {
                    lowerLevelFiles = new ArrayList<>();
                    for (int j = leftInd; j <= rightInd; j++) {
                        lowerLevelFiles.add(level1Files.get(j));
                    }
                    upperLevelFiles = thisPlan;
                    ra = thisRA;
                }
            }
        }
    }
}
```