关于快排：

1. 每次一个下标从 left 到 right 的待排序序列，随机找一个 left 和 right 之间的位置 x，a[x]=key。
2. i 从 left 开始向右一直找，直到找到一个比 key 大的值，位置为 i; j 从 right 开始向左一直找，直到找到一个比 key 小的值，位置为 j。
3. 如果 i<=j，交换 a[i]和 a[j]，然后继续按此规律进行，直到 i>.j。
4. 如此，j 左边的数都小于 key，i 右边的数都大于 key，再分治法，分别排序(left, j)和(i, right)即可。
5. 由于采用了分治法，快排平均树高 logN, 每层平均比较 N 次，所以平均时间复杂度为 O(NlogN)。

关于 stable：

为了让快排 stable，核心算法不变，只有细节有一点变化。说白了就是双关键字排序，第一关键字是数的大小，第二关键字是相同数的序号。但是不能分两次排序先排大小，再排序号。而是应该在一次排序里改变边界条件，即找 i 和 j 的位置的时候，加一个当数的值一样时，让序号也满足值得排序条件，如 while (a[i].data<key.data || (a[i].data==key.data && a[i].index<key.index))。如此让排大小的时候同时保持相同大小的数的相对位置，就满足了快排的 stable。

关于快排和插排的合并优化算法：

和 lab1 类似，找到了理论的 K 值为 350。再通过实际实验，得出优化算法的 K 值为 300。K 值比预计的大很多，是由于快排中用了随机化，Java 的产生随机数方法比较慢，导致快排稍微变慢，在数据规模为 300 时，插排依旧比快排快，所以如果不用随机数，每次比较取中间数，得出 K 值大约为 50，在正常范围之内。

以上 K 值均通过实际实验验证，这里具体过程数据和图表不再赘述，类似 lab1，通过手动改变 MySort 类中的构造方法里 n 的值，再运行 TestSort.java 可以查证。