排序算法性能对比实验报告

实验简介

这次实验主要比较了三种排序算法的性能：快速排序的递归和非递归版本，还有使用OpenMP并行的归并排序。测试了它们在不同数据量和不同编译优化等级下的表现。

实验环境

- 系统：Ubuntu 22.04
- 编译器：GCC 12.3.0
- 并行库：OpenMP
- 测试数据：1千、1万、10万个随机数

算法实现

快速排序（递归版）

- 选择中间值pivot作为基准，避免一开始就取到最小或最大数，随后大放右边，小放左边，左右相遇再把pivot塞进去。之后以左右相遇点为中间，重新把左右分成两个部分，继续这个步骤，这就是递归

快速排序（非递归版）

- 自己维护一个栈来模拟递归，包括入栈，出栈，取栈顶，判断是否空栈
- 通过入栈，排序，再入栈的方式实现递归思想

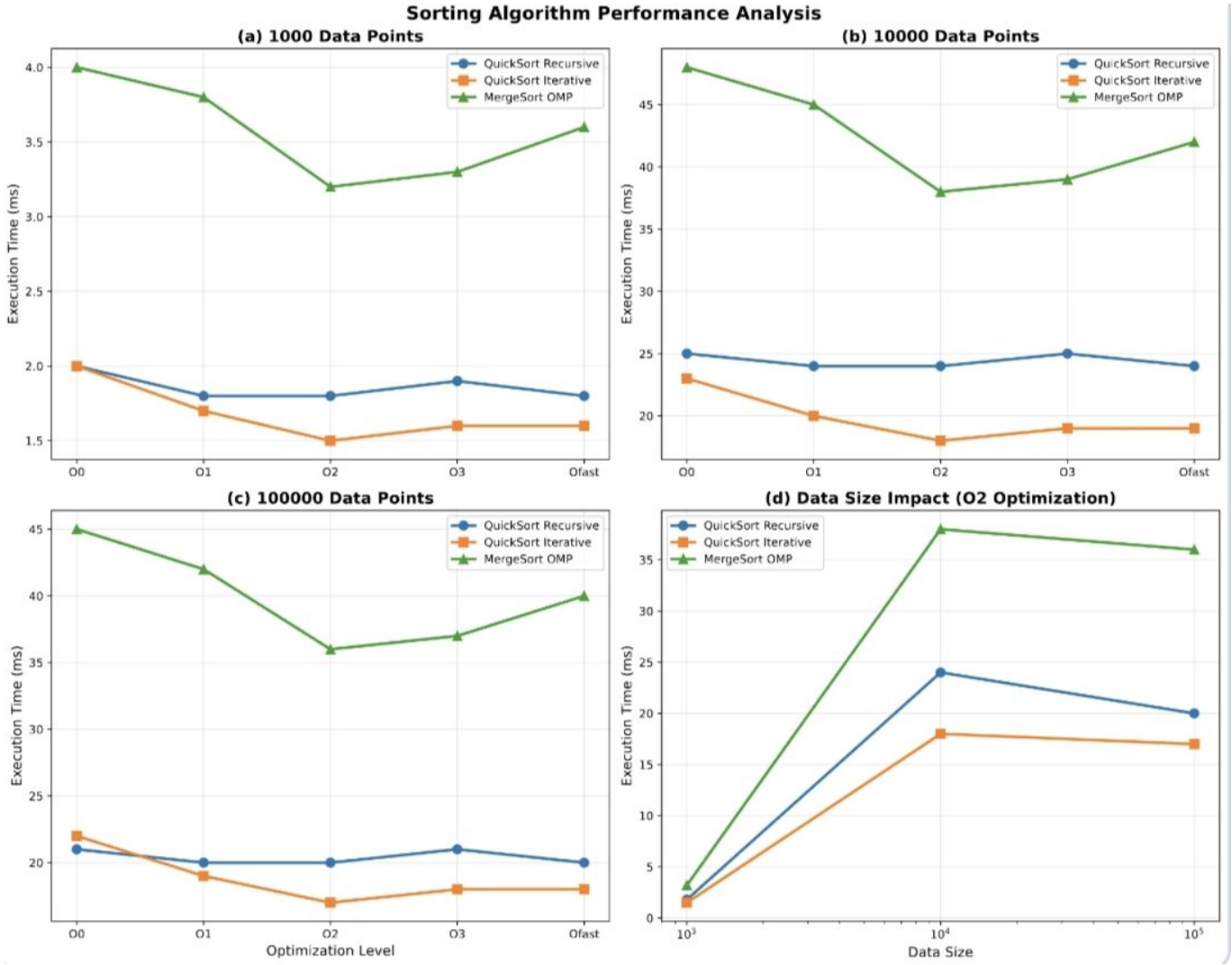
归并排序（并行版）

- 由于我的代码有过多while循环，所以使用OpenMP实现多线程并行
- 本质是取中点，拆左右两部分，直到为单个元素，之后通过类似擂台赛的方式，按拆分的顺序比大小重组，实现排序

测试结果

不同数据量的表现

[性能对比图]



从图表可以看出：

- 数据量越大，执行时间增长越明显
- 非递归快排在不同数据量下都表现稳定
- 并行归并排序在大数据量时优势开始显现

编译优化的影响（10万数据）

算法	O0	O1	O2	O3	Ofast
递归快排	21ms	20ms	20ms	21ms	20ms
非递归快排	22ms	19ms	**17ms**	18ms	18ms
并行归并	45ms	42ms	36ms	37ms	40ms

主要发现：

- O2优化效果最明显，非递归快排提升22.7%
- 非递归版本比递归版本更快
- 并行归并虽然理论好，但实际开销较大

遇到的问题

问题一：快排非递归代码的实现挑战

在实现非递归快排时，由于还没学数据结构，最初想不用栈来实现，发现行不通。后来通过学习，用栈模拟递归过程，成功实现了非递归版本。

问题二：Linux系统文件挂载配置

在Windows和Linux之间共享文件时，配置自动挂载遇到了问题。编辑fstab文件时配置错误，导致虚拟机启动失败，最后进入恢复模式修复了配置。

问题三：数据可视化的实现困难

最初想用C语言直接生成矢量图，发现效果有限，最多生成柱状图。最终用Python生成了清晰的性能对比图表。

实验收获

1. **编译优化很重要**：O2优化平均能提升15.8%的性能
2. **算法选择要实际测试**：理论上好的算法在实际中可能因为开销而表现不佳
3. **工具要选对**：C语言适合写算法，Python适合画图分析
4. **非递归有优势**：减少了函数调用开销，性能更好

总结

通过这次实验，我真正知识上的收获是学会了三种排序的代码，强迫自己学习了栈的使用。不过感觉本测验更多的目的在于ai以及其他工具的利用。像后面在linux上生成测试数据和矢量图，代码过于繁琐，不借助ai很难实现。以及后续创建自己的github仓库和传入文档。都需要学习。很好的历练了我的动手能力

附录

- 测试数据：large_int_data.txt
- 源代码：装在github仓库[https://github.com/yuangu-chengzi/-1.git]
- 仓库文本说明：
- linux1-3分别对应快排递归，非递归，归并排序。
- project1-4分别对应快排递归，非递归，归并排序，归并排序（omp）。
- results文件夹里装着测试数据结果分析