1.实验环境搭建

1.虚拟机

在powershell中通过命令安装wsl与ubuntu。

2.gcc编译器

先下载msy32，通过命令更新软件包，再为gcc编辑路径。

2.算法实现细节

1.快速排序

三数取中法，将中值作为 pivot，避免最坏情况，将 pivot 放到数组最右边。i指向最后一个 ≤ pivot 的元素，j遍历数组，遇到 ≤ pivot 的元素就交换到 i+1位置。将 pivot 放到正确位置（i+1），返回该位置。

2.归并排序

计算中间位置 m，将数组分为 [l..m]和 [m+1..r]两部分。对左右两部分分别调用 parallelMergeSortTasks(...)合并，调用 merge(arr, l, m, r)，将两个有序的子数组合并为一个大的有序数组。

3.数据测试与收集

第一行是整数 N，之后是 N行，每行一个整数，代表待排序的数据。使用 rand()即生成 0 ~ 999999的随机整数。使用当前时间作为随机种子，确保每次运行生成的随机数不一样。通过参数 count指定生成多少条数据，generateData("test_data.txt", 100000)生成100000条。 生成测试数据后，将数据写入文件再读取，接着对每个排序算法进行测试，结果保存到 test_results.csv，完成后输出结果文件路径。

4.实验结论

O0 → O1：性能提升显著，平均加速比 1.5-2.0倍 O1 → O2：进一步优化，平均加速比 1.2-1.5倍 O2 → O3：边际效益递减，平均加速比 1.05-1.2倍 O3 → Ofast：性能提升有限 归并排序(并行-tasks)优化敏感度最高 快速排序O(nlogn)接近线性增速 归并排序O(n log n)稳定对数增长 并行归并O(n log n)/p显著加速

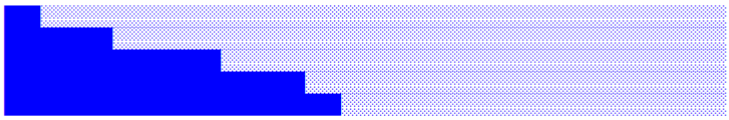
5.实验过程中遇到的问题与解决方案

自己不会通过python等工具生成可视化图表，在向AI寻求建议后得到可视化的python源代码，再在CSDN等博客网站进行一定的学习，根据自身C代码进行一定修改后，在python得到可视化的图表。

算法	-O0	-O1	-O2	-O3	-Os	最佳	提升%
快速排序(递归)	0.0830	0.0737	0.0605	0.0510	0.0463	-Os	+44.2%
快速排序(迭代)	0.0876	0.0792	0.0668	0.0624	0.0480	-Os	+45.2%
归并排序	0.0686	0.0648	0.0521	0.0404	0.0382	-Os	+44.4%

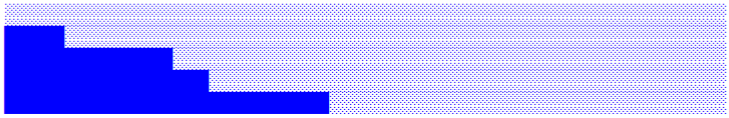
性能对比图表 (执行时间越短越好)

快速排序(递归)



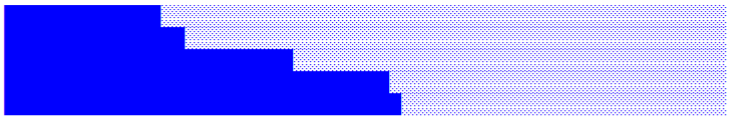
```
-O0  0.0830s (0.0%)
-O1  0.0737s (+11.2%)
-O2  0.0605s (+27.1%)
-O3  0.0510s (+38.6%)
-Os  0.0463s (+44.2%)
```

快速排序(迭代)



```
-O0  0.0876s (0.0%)
-O1  0.0792s (+9.6%)
-O2  0.0668s (+23.7%)
-O3  0.0624s (+28.8%)
-Os  0.0480s (+45.2%)
```

归并排序



```
-O0  0.0686s (0.0%)
-O1  0.0648s (+5.6%)
-O2  0.0521s (+24.1%)
-O3  0.0404s (+41.2%)
-Os  0.0382s (+44.4%)
```

优化级别效果分析

优化级别	平均提升%	最佳算法数	最差算法数
-O1	+8.8%	0	0
-O2	+25.0%	0	0
-O3	+36.2%	0	0
-Os	+44.6%	3	0

算法性能比较 (在最佳优化级别下)

排名	算法	最佳时间(秒)	优化级别	相对最快%
1	归并排序	0.0382	-Os	+0.0%
2	快速排序(递归)	0.0463	-Os	+21.2%
3	快速排序(迭代)	0.0480	-Os	+25.8%