# C语言排序算法测试实验报告

## 实验环境的搭建

我在浏览器里下载了VMware workstation，又找寻了Ubuntu20.04的安装光驱，用lan链接上网后，在系统自带的终端中，输入了对应的命令，更新了本地软件包后安装装了gcc编译器·sudo apt install build-essential·

##排序算法的实现

我分别编写了快速排序与分并排序的排序代码。对于快速排序，我分为了递归与非递归两个部分，快速排序的关键在于分离，我使用了两个基准来进行分离，其中用中间值代替了习惯中常用的最左边的数字，这样可以避免一些倒序等特殊情况对处理速度的干扰，对于非递归快速排序的实现中，我用额外的低一级的栈来成对的保存已分离数组，这样可以方便去调用和理解。对于归并排序，重点则在于数组的合并，我使用了保守的模型，保证排序的准确性，在归并本身上，我使用了递归的处理方式，便于构思。为了避免临界情况和分区操作的复杂，每种排序的函数中我都接入了简单的插入排序，并设计了一个特定的阈值。

下面是一些片段:···

’’’c

void quicksort1(double arr[],int low ,int high ){

if (low < high){

//插入排序

if (high - low < 10){

intersort(arr,low,high);

return ;

}

int pivot1 , pivot2;

apart(arr,low,high,&pivot1,&pivot2);

quicksort1(arr,low,pivot1-1);

quicksort1(arr,pivot1,pivot2-1);

quicksort1(arr,pivot2,high);

}

’’’

’’’c

···void mergesort(double arr[],int left,int right){

if(left < right){

int mid = left + (right - left)/2;

mergesort(arr,left,mid);

mergesort(arr,mid+1,right);

merge(arr,left,mid,right);

}

}

’’’

对于测试数据，我选择了最简单的rand函数，用min和max来基本地限制数字本身的范围

随后，我分别测试了在不同的优化等级下，三种方案排序100000条数据的用时，手动收集数据后计算了他们的平均值，并用deep seek提供你可视化方案绘制了如下的数据图

！[数据图](Figure\_1.png)

我发现,非递归版本有明显的优势，且对于优化也有较高的相应程度，这也许得益于栈的深度被钉死了。然而递归排序的表现则比较差劲，时间最长且对优化也不敏感，优化提升反而会降速，也许设计简单的代价就是运行开销的增大吧。归并排序则比较中肯，长久来看，它的算法复杂度有数学兜底，而且对优化的反响很好，我相信它存在着很多优化空间。总的来说，这些排序都是在用策略不同的分区进行任务，得益于此，他们可以用更小的代价来完成对数字相对位置的判断。

##实验中遇到的问题

1.非递归快速排序的脆弱：由于栈结构的复杂，我在实验中常常出现不能正常运行它的问题，或是栈结构溢出，或是三分与栈结构的不兼容，最终，我决定将子数组封装为一个更小的栈，这样便于我调试和理解问题，此外我还直接规定了栈的最大范围，牺牲了动态分配来换取结构的正常运行。然而即便如此，在-O3优化下，整数的非递归快速排序也没能顺利完成排序任务。出乎意外的是，调整为浮点数后，它成为了那匹黑马，没有产生报错了。

2.内存测量的失败：由于我对内存知识的空白，我只能查阅资料或者询问AI来拼凑出一个测量的函数，最终它的表现不尽人意，有且仅有0和128kb两种答案，我猜测它只测出了大程序本身的内存占用，最终我只能放弃了这项数据的收集与分析。

3.临界条件的频繁误判：出于个人的原因，我对临界的情况总是很难想象，这囊括了很多种问题包括分区到最后的单数组，包括最后的单栈。最终我通过等效代换的方式用许多途径一一避开了这些临界情况，包括用插入排序，用二元栈，一点点在本子上数数组和指针的位置。最终我成功让它顺利运行。