155-杨凌林-PB17000083-project1

目录

实验内容

排序随机生成的 n 个元素，元素范围为 0 到 $2^{15}-1$。n 的取值为：$2^3, 2^6, 2^9, 2^{12}, 2^{15}, 2^{18}$。需自己编写程序，实现五种排序算法：直接插入排序，堆排序，快速排序，归并排序，计数排序。并将排序结果和运行时长写入文件，并分析不同排序算法的复杂性和性能。

实验设备和环境

版本	Windows 10 专业版 (2004)
操作系统版本	19041.630
计算机型号	Asus U4100U
处理器	Core i7-8550U

表 1: 实验设备

IDE	CLion(2020.2.4)
编译器	g++(9.2.0-2)
构建	CMake

表 2: 实验环境

实验方法和步骤

由于实验要求较多，故在 `main` 函数中，编写了一个简单的菜单：输入 $0, 1, 2$ 三个数，可分别实现退出程序、生成随机数文件、五种排序算法三种功能。

`main` 函数读取用户输入的数字 `0` 后，调用 `random_num.h` 文件中 `random_num_generate` 函数，生成含有 2^{18} 个元素的 0 到 $2^{15}-1$ 之间的随机整数，供排序算法读取。若用户再次输入数字 `1`，便会分别调用头文件中编写好的五种排序算法对六种规模的输入元素排序，排序后的结果和运行时长分别写入对应的文件夹。

src 文件夹的中文件和子函数的组织形式如下：

```
main.cpp
   main()
insertion_sort.h
   insertion_sort(int *A, int n)
heap_sort.h
   heap_sort(int *A, int n)
   build_max_heap(int *A, int n)
   max_heapfy(int *A, int n, int i)
quick_sort.h
   quick_sort(int *A, int n) // quick_sort(A, n) 其实是在调用 quick_so(A, 0, n)
   partition(int *A,int p,int r)
   quick_so(int *A, int p, int r)
   swap(int *A,int i,int j)
merge_sort.h
   merge_sort(int *A, int n) //merge_sort(A, n) 其实是在调用 merge_so(A, 0, n)
   merge(int *A,int p,int q,int r)
   merge_so(int *A,int p,int r)
counting_sort.h
   counting_sort(int *A,int n)
random_num.h
   random_num_generate(FILE *f, int n)
```

注释：

- 由于快速排序和归并排序算法采取分治思想，需要应对任意下标 A[p,...,r]。为了主函数 main 调用方便，添加了函数 quick_so 和 merge_so，处理任意下标的排序，并编写 quick_sort 和 merge_sort 调用 *_so，并赋参数 p=0，使排序从数组第 0 位开始。这样五种排序算法都能通过同一类型的函数指针调用，方便我们编写 for 循环处理。
- 本程序中五种排序算法中的参数 n 是需要排序数组的最后一个元素在数组中的下标，即问题规模减一。例如：对 2^{18} 个元素做快速排序，主函数中调用的是 quick_sort(A, (1<<18)-1)。

实验结果和分析

运行截图

$n = 2^3$ 时排序结果截图如下：

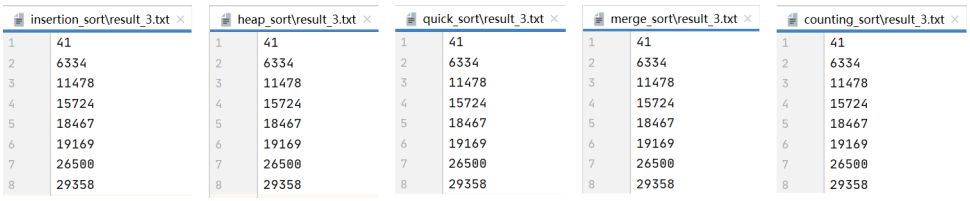


图 1: 五个排序算法 $n = 2^3$ 时排序结果的截图

`quick_sort` 排序时间截图如下：

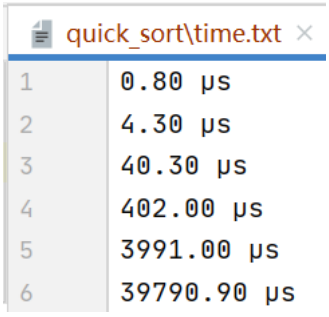


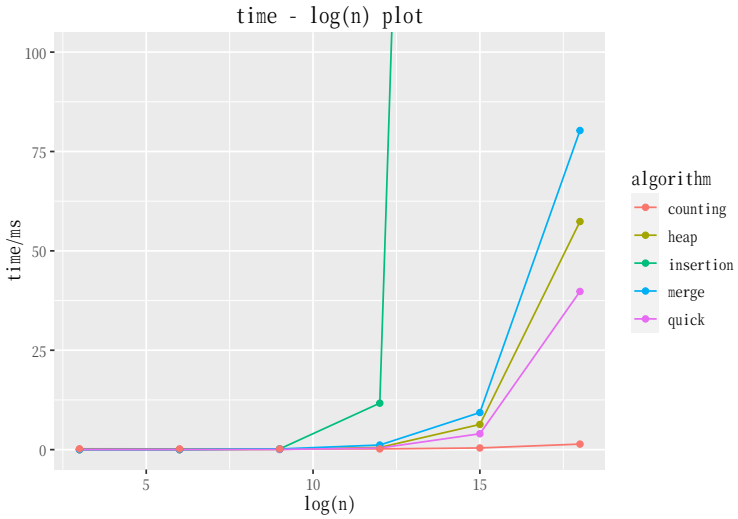
图 2: 快速排序算法六个输入规模运行时间的截图

运行时长图

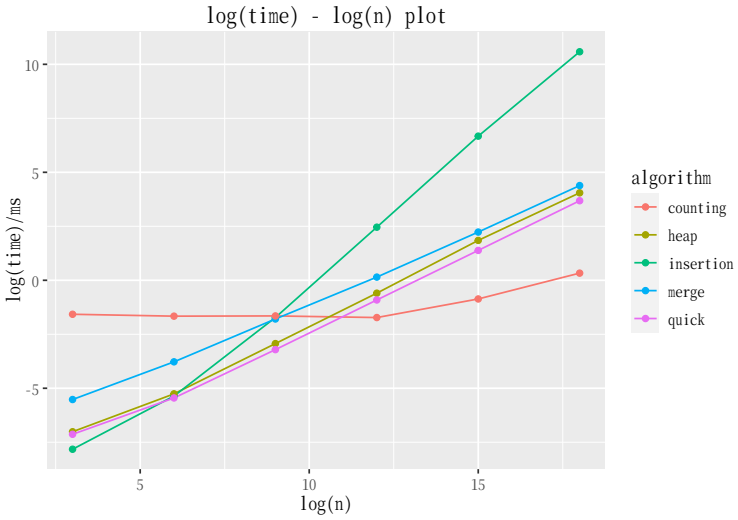
使用 R 对运行时间做图:

```
# 数据读入和预处理
algo=c("insertion","heap","quick","merge","counting")
lgn=as.numeric(c(3,6,9,12,15,18))
path=paste(paste("../ex1/output/",algo[1],sep=''),"_sort/time.txt",sep='')
x=read.table(path)[1]
data=cbind(lgn,time=x,algorithm=rep(algo[1],times=6))
for(i in 2:5){
  path=paste(paste("../ex1/output/",algo[i],sep=''),"_sort/time.txt",sep='')
  x=read.table(path)[1]
  x=cbind(lgn,time=x,algorithm=rep(algo[i],times=6))
  data=rbind(data,x)
}
colnames(data)[-3]=c("lgn","time")
data[,2]=data[,2]/1000  # 将微秒化为毫秒

# 运行时间图
library(ggplot2)
p<-ggplot(data, aes(x=lgn,y=time,color=algorithm))+
  geom_line()+
  geom_point()+
  labs(x="log(n)",y="time/ms")+
  ggtitle("time - log(n) plot")+
  theme(plot.title = element_text(hjust = 0.5))
p+coord_cartesian(ylim = c(0,100))
```




```
q<-ggplot(data, aes(x=lgn,y=log(time),color=algorithm))+
  geom_line()+
  geom_point()+
  labs(x="log(n)",y="log(time)/ms")+
  ggtitle("log(time) - log(n) plot")+
  theme(plot.title = element_text(hjust = 0.5))
q
```



渐进性能分析

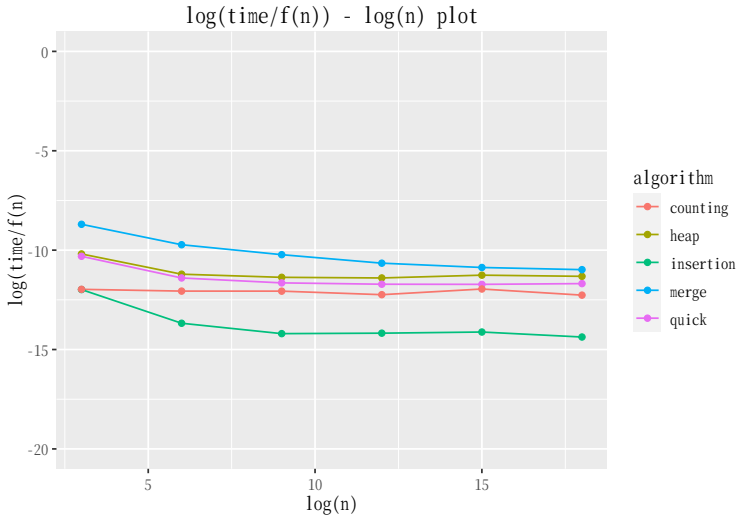
从教材上，我们知道，各排序算法的渐进复杂度如下所示：

排序算法	直接插入排序	堆排序	快速排序	归并排序	计数排序
复杂度 $f(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n\log(n))$	$\mathcal{O}(n\log(n))$	$\mathcal{O}(n\log(n))$	$\mathcal{O}(n+k)$

表 3: 五种排序算法的理论复杂度

为验证渐进性能，我们做 `log(time/f(n)) - log(n)` 图

```
n=2^(lgn)
base=c(n^2, rep(n*lgn, times=3),n+2^15)
r<-ggplot(data, aes(x=lgn,y=log(time/base),color=algorithm))+
  geom_line()+
  geom_point()+
  labs(x="log(n)",y="log(time/f(n)")+
  ggtitle("log(time/f(n)) - log(n) plot")+
  theme(plot.title = element_text(hjust = 0.5))
r+coord_cartesian(ylim = c(-20,0))
```



可以看到，各算法的图形基本渐进处于一条横线上。这便验证了这五种排序算法的理论复杂度是正确的。

五种算法的比较

先看 `time - log(n)` 图和 `log(time) - log(n)` 图，通过观察，可以发现不同排序算法之间有较大差别。

- 插入排序算法在数据规模较小时有较好性能（观察 `log(time)-log(n)` 图），但对于较大规模的问题（$n \geqslant 2^{12}$ 时），运行时间明显比其他算法长。
- 而处理较大规模问题（$n \geqslant 2^{12}$ 时）表现最好的计数排序算法，在简单问题上，运行时间又比其他算法长。

再观察 `log(time/f(n)) - log(n)` 图

- 在三种理论复杂度都为 $\mathcal{O}(n\log(n))$ 的算法（堆排序，快速排序，归并排序）中，由于快速排序的 `log(time/f(n)) - log(n)` 线在其他两种算法的下方，说明快速排序渐进常数更小，性能更好。
- 我们也能通过 `log(time/f(n)) - log(n)` 图发现，直接选择排序的渐进常数最小，这也验证了我们之前的结论：问题规模较小时，插入排序性能好。

实验总结

本次实验测试了五种排序算法在不同问题规模下的运行时间，总结前面的分析，我们可以得到以下结论，为日后排序排序算法的选择提供参考：

- 对于规模较小的问题（$n \leqslant 2^6$），推荐使用插入排序算法；也可选择堆排序，快速排序，归并排序；不推荐计数排序。
- 对于中等规模的问题（$2^6 < n < 2^{12}$），推荐使用快速排序算法。
- 对于较大规模的问题（$n \geqslant 2^{12}$），推荐使用计数排序算法；不推荐插入排序。