实验一 排序算法

实验要求

使用各种排序算法对不同规模的字符串组和数组排序。用适当的方法或工具记录排序算法在执行时所消耗的时间。根据不同输入规模时记录的数据，画出算法在不同输入规模下的运行时间曲线图，并作分析。

实验环境

编译环境：Dev C++ 5.11，TDM-GCC 4.9.2 64-bit Debug
机器内存：16GB
时钟频率：2.20GHz

实验过程

首先实现生成实验用随机字符串和数组的单独程序，然后分别用函数实现各排序算法，最后在主函数中调用排序算法，测量运行时间以及实现读写文件操作。

实验关键代码截图

1. 随机字符串（random-char）和数组生成(random-int)

```
char **str;
str = (char**)malloc(sizeof(char*)*m);
for(i =0;i < m;i++)
    str[i] = (char*)malloc(sizeof(char)*33);
srand((unsigned)time(NULL));//以系统时间作为随机数种子
for(i = 0;i < m;i++)
{
    n = 1 + rand() % 32;//字符串长1~32
    for(j = 0;j < n;j++)
    {
        str[i][j] = 'a' + rand() % 26;//生成只有小写的随机字符串
        str[i][j + 1] = '\0';
    }
}
```

以上为生成随机字符串的代码，由于直接定义 2^17 大小的二维字符数组会出错，所以最终改为使用二级指针动态管理。数组生成同理，不再赘述。

2. 字符串比较函数

```
int compare(char a[],char b[])
{
    if(strlen(a) > strlen(b) ||strlen(a) == strlen(b) && strcmp(a,b) > 0)
        return 1;
    return 0;
}//字符串比较函数
```

3. 各种排序函数

基本上由课本伪代码转化而来，只要注意函数传参和数组下标即可。

4. 测量算法运行时间

```
#include "windows.h"
```

```
LARGE_INTEGER nFreq;//时钟频率
LARGE_INTEGER t1;//开始
LARGE_INTEGER t2;//结束
double dt[6]; //时间差
QueryPerformanceFrequency(&nFreq);//获取CPU频率
QueryPerformanceCounter(&t1);//开始时间
INSERTION_SORT(str,m);
QueryPerformanceCounter(&t2);//结束时间
dt[count] = (t2.QuadPart - t1.QuadPart )/ (double)nFreq.QuadPart *1000000;
```

5. 文件操作

读待排序数据：

```
for(count = 0;count < 6;count++)
{
    m = pow(2,arr[count]);
if((fp1 = fopen("../input/data.txt","r"))== NULL)
{
    printf("error");
}
else
{
    for(i = 0;i < m;i++)
    {
        fscanf(fp1,"%s",str[i]);
    }
    printf("read successful!\n");
    fclose(fp1);
}
```

写排序后数据

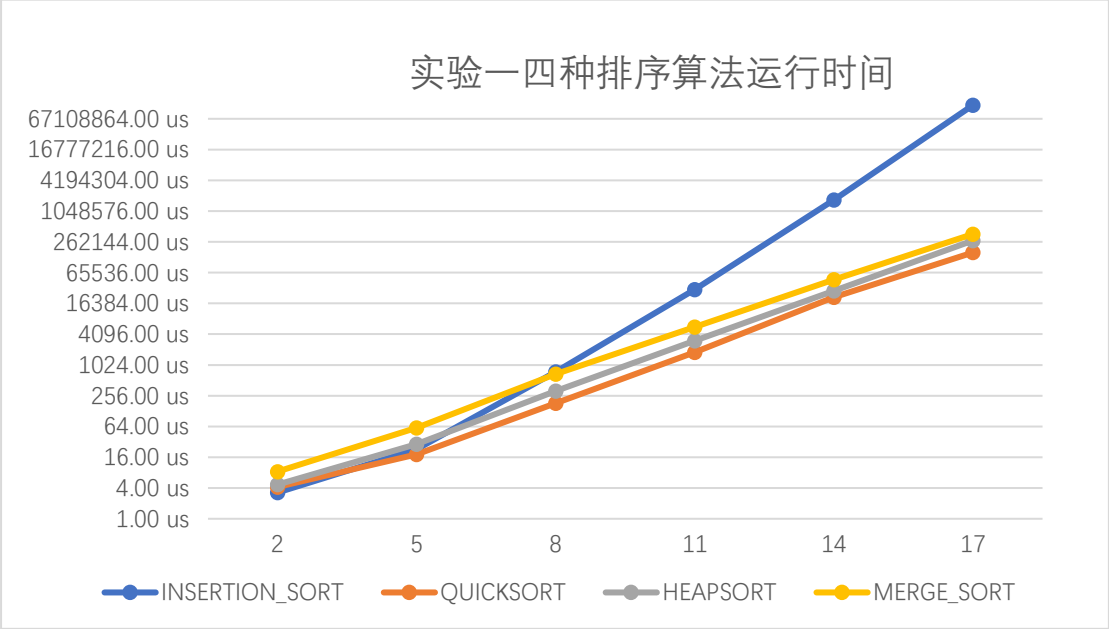
```
switch(count)
    {
        case 0:if((fp2 = fopen("../output/INSERTION_SORT/result_2.txt","w"))== NULL)
        printf("error\n");break;
        case 1:if((fp2 = fopen("../output/INSERTION_SORT/result_5.txt","w"))== NULL)
        printf("error\n");break;
        case 2:if((fp2 = fopen("../output/INSERTION_SORT/result_8.txt","w"))== NULL)
        printf("error\n");break;
        case 3:if((fp2 = fopen("../output/INSERTION_SORT/result_11.txt","w"))== NULL)
        printf("error\n");break;
        case 4:if((fp2 = fopen("../output/INSERTION_SORT/result_14.txt","w"))== NULL)
        printf("error\n");break;
        case 5:if((fp2 = fopen("../output/INSERTION_SORT/result_17.txt","w"))== NULL)
        printf("error\n");break;
    }
for(i = 0;i < m;i++)
    fprintf(fp2,"%s\n",str[i]);
printf("write successful!\n");
dt[count] = (t2.QuadPart - t1.QuadPart )/ (double)nFreq.QuadPart *1000000;  //时间差
fclose(fp2);
```

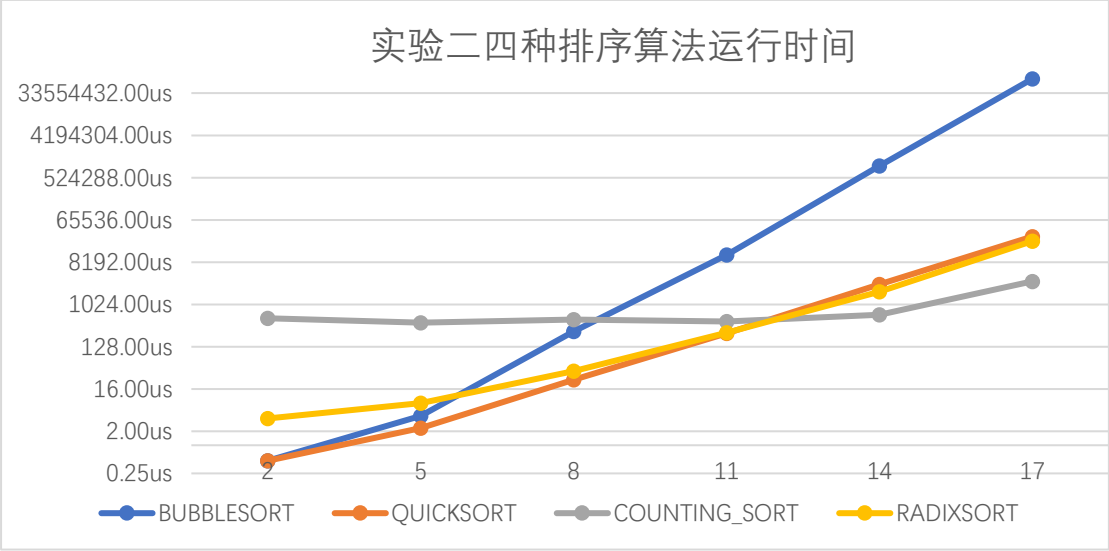
写测量到的运行时间：

```
if((fp3 = fopen("../output/INSERTION_SORT/time.txt","w"))== NULL)
        printf("error\n");
    for(count = 0;count < 6;count++)
    {
        fprintf(fp3,"%f\n",dt[count]);
    }
fclose(fp3);
```

实验结果，分析

实验测得的运行时间整理在 exp1 表格和 exp2 表格中，运行时间图（纵坐标取 lg2 对数）如下：





可以看出，对于不同规模的数据，除了计数排序，其他排序算法运行时间均与数据规模呈正相关。实验一中快速排序，堆排序和归并排序运行时间的增长率相同，与课本中三者平均时间复杂度 O（nlgn）一致；直接插入排序为 O（n^2）与课本

一致。实验二中冒泡排序为 O（n^2），在 2^11 规模以后快排运行时间超过基数排序。计数排序基本保持 O（n）的时间复杂度，但在数规模较小时，用时反而更大，这与计数排序需要额外的辅助数组有关。

对于相同规模不同排序算法的比较，实验一可以看出在 2^2 规模，插入排序运行最快，归并排序最慢，但在规模 2^8 以后，快排最快，堆排序和归并排序其次，插入排序非常慢。实验二可以看出规模较小时计数和基数排序很慢，这与这两个算法需要额外的辅助有关；在规模较大时，计数排序最快，冒泡排序最慢。

实验心得

本次实验让我复习了 c 语言的文件读写操作，认识到了 c 语言的局限性。通过对各种排序算法的分析，充分理解了各算法的基本原理，认识到了各算法的优缺点以及适用条件，对代码的优化有了一定的认识。学习了怎样测量程序运行时间以及图表的制作。