算法导论实验1、2

肖文宗 JL16110020

1.实验要求

实验1：排序 n 个元素，元素为随机生成的长为1..32的字符串（字符串均为英文小写字母），n的取值为：2^2，2^5，2^8，2^11，2^14，2^17；

算法：直接插入排序，堆排序，归并排序，快速排序。

实验2：排序n个元素，元素为随机生成的1到65535之间的整数，n的取值为： 2^2，2^5，2^8，2^11，2^14，2^17 ；

算法：冒泡排序，快速排序， 基数排序，计数排序。

2.实验环境

- 编辑器：Notepad++
- 编译器：codeblocks
- 解释器：windows10 企业版下的命令行
- 机器内存：8G
- 时钟主频：2.40GHz

3.实验过程

1. 创建所需的文件夹，按照ppt上的要求即可。
2. 编写生成实验一所需的**随机字符串**的程序。
3. 编写实验一所需的算法。
4. 编写生成实验二所需的**随机数**的程序。
5. 编写实验二所需的算法。
6. 用适当的方法对所得数据进行分析。

4.实验关键代码截图

1. 生成**随机字符串**。

   在codeblocks下执行produce_strings,生成了131072（2^17)个随机字符串，生成的文件如下：

input_strings.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
131032. knnlindvkpsnzrwndydzgyxlhra
131033. xdq
131034. qdsbqywcnrsoioclohkvfkinbi
131035. alzjecefuc
131036. wruofe
131037. jptqcqwm
131038. lp
131039. ssvrbzvfgpjjnagcikjraeiexgsyrh
131040. lzcedb
131041. khoutdwzfgjqylxiruqj
131042. hrz
131043. scyxfxipnekqclyjwqhinrbhhlb
131044. edkyhiwmfedskpkma
131045. vxzehoxiyetzuxeewlmua
131046. afnkvyyviusjazyyiwedxbdchznw
131047. ppvvdhqdfllbexefventitslwutxkpsb
131048. pwlwmodacfgzjlwpb
131049. xujlqgepsvhtdwspuedxghemipye
131050. zxevhpnptch
131051. jcthuidboahxhugwumb
131052. kfqtudiphrhsvabqletbgaan
131053. gmqvnlfudaq
131054. kfpdohyqiqbojevmfkinwbijopfsr
131055. abcseqbfhkvkyhrvtygdlcgwovlt
131056. dppgtjlmlqtxnfxuwtpubvidbetypnnu
131057. fbxqojbgeytmwqk
131058. hfse
131059. xi
131060. rgke
131061. fkdijtwvvczvxntpykzfcdu
131062. lntuxxsejvosoknlophoccdan
131063. hribtvompolhuucjntv
131064. gmrpaws
131065. yfkgwuhspyefoqokmrtbmres
131066. xifvckuhwcfqlxglenzwbmiu
131067. mgyqbbplinnvjwrldkydqk
131068. fldgxnbfzhdr
131069. hhvpukmfieqmqopgawnmb
131070. jouoqeuhqbqosmyfrh
131071. jvbakjaiwmjrwzmmrsgzijrwegxlxgyj
131072. zjlkjviwezufnixfryqa
```

2. 实验一各个算法的执行。

   - insertion_sort

```
E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\insertion_sort>insertion_sort.exe 4
../../output/insertion_sort/result_4.txt
0.000012s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\insertion_sort>insertion_sort.exe 32
../../output/insertion_sort/result_32.txt
0.000015s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\insertion_sort>insertion_sort.exe 256
../../output/insertion_sort/result_256.txt
0.000582s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\insertion_sort>insertion_sort.exe 2048
../../output/insertion_sort/result_2048.txt
0.025743s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\insertion_sort>insertion_sort.exe 16384
../../output/insertion_sort/result_16384.txt
1.577331s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\insertion_sort>insertion_sort.exe 131072
../../output/insertion_sort/result_131072.txt
97.954277s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\insertion_sort>
```

   - heap_sort

```
E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\heap_sort>heap_sort.exe 4
../../output/heap_sort/result_4.txt
0.000003s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\heap_sort>heap_sort.exe 32
../../output/heap_sort/result_32.txt
0.000013s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\heap_sort>heap_sort.exe 256
../../output/heap_sort/result_256.txt
0.000138s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\heap_sort>heap_sort.exe 2048
../../output/heap_sort/result_2048.txt
0.001482s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\heap_sort>heap_sort.exe 16384
../../output/heap_sort/result_16384.txt
0.013691s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\heap_sort>heap_sort.exe 131072
../../output/heap_sort/result_131072.txt
0.146516s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\heap_sort>
```

- merge_sort

```
E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\merge_sort>merge_sort.exe 4
../../output/merge_sort/result_4.txt
0.000006s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\merge_sort>merge_sort.exe 32
../../output/merge_sort/result_32.txt
0.000013s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\merge_sort>merge_sort.exe 256
../../output/merge_sort/result_256.txt
0.000131s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\merge_sort>merge_sort.exe 2048
../../output/merge_sort/result_2048.txt
0.001284s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\merge_sort>merge_sort.exe 16384
../../output/merge_sort/result_16384.txt
0.014095s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\merge_sort>merge_sort.exe 131072
../../output/merge_sort/result_131072.txt
0.125962s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\merge_sort>
```

- quick_sort

```
E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\quick_sort>quick_sort.exe 4
../../output/quick_sort/result_4.txt
0.000001s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\quick_sort>quick_sort.exe 32
../../output/quick_sort/result_32.txt
0.000009s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\quick_sort>quick_sort.exe 256
../../output/quick_sort/result_256.txt
0.000115s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\quick_sort>quick_sort.exe 2048
../../output/quick_sort/result_2048.txt
0.001071s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\quick_sort>quick_sort.exe 16384
../../output/quick_sort/result_16384.txt
0.010941s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\quick_sort>quick_sort.exe 131072
../../output/quick_sort/result_131072.txt
0.121167s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex1\source\quick_sort>
```

3. 生成**随机数**。

   在codeblocks下执行produce_numbers,生成了131072（2^17)个随机数，生成的文件如下：

input_numbers.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
131032.12520
131033.45356
131034.28064
131035.16208
131036.15281
131037.30900
131038.30589
131039.39583
131040.46169
131041.40134
131042.48993
131043.35554
131044.34257
131045.56775
131046.30425
131047.44028
131048.30972
131049.35283
131050.2788
131051.28932
131052.42939
131053.34751
131054.34560
131055.40779
131056.20453
131057.16920
131058.55327
131059.43354
131060.34375
131061.29218
131062.28587
131063.27122
131064.51625
131065.19162
131066.18055
131067.48909
131068.30154
131069.33782
131070.16398
131071.34009
131072.38496
```

4. 实验二各个算法的执行。

   - bubble_sort

```
E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\bubble_sort>bubble_sort.exe 4
../../output/bubble_sort/result_4.txt
0.000000s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\bubble_sort>bubble_sort.exe 32
../../output/bubble_sort/result_32.txt
0.000003s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\bubble_sort>bubble_sort.exe 256
../../output/bubble_sort/result_256.txt
0.000145s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\bubble_sort>bubble_sort.exe 2048
../../output/bubble_sort/result_2048.txt
0.008711s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\bubble_sort>bubble_sort.exe 16384
../../output/bubble_sort/result_16384.txt
0.743413s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\bubble_sort>bubble_sort.exe 131072
../../output/bubble_sort/result_131072.txt
53.364723s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\bubble_sort>
```

   - quick_sort

```
E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\quick_sort>quick_sort.exe 4
../../output/quick_sort/result_4.txt
0.000000s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\quick_sort>quick_sort.exe 32
../../output/quick_sort/result_32.txt
0.000002s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\quick_sort>quick_sort.exe 256
../../output/quick_sort/result_256.txt
0.000022s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\quick_sort>quick_sort.exe 2048
../../output/quick_sort/result_2048.txt
0.000187s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\quick_sort>quick_sort.exe 16384
../../output/quick_sort/result_16384.txt
0.001749s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\quick_sort>quick_sort.exe 131072
../../output/quick_sort/result_131072.txt
0.017391s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\quick_sort>
```

- radix_sort

```
E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\radix_sort>radix_sort.exe 4
../../output/radix_sort/result_4.txt
0.000022s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\radix_sort>radix_sort.exe 32
../../output/radix_sort/result_32.txt
0.000007s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\radix_sort>radix_sort.exe 256
../../output/radix_sort/result_256.txt
0.000028s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\radix_sort>radix_sort.exe 2048
../../output/radix_sort/result_2048.txt
0.000201s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\radix_sort>radix_sort.exe 16384
../../output/radix_sort/result_16384.txt
0.001678s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\radix_sort>radix_sort.exe 131072
../../output/radix_sort/result_131072.txt
0.015383s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\radix_sort>
```

- counting_sort

```
E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\counting_sort>counting_sort.exe 4
../../output/counting_sort/result_4.txt
0.000447s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\counting_sort>counting_sort.exe 32
../../output/counting_sort/result_32.txt
0.000428s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\counting_sort>counting_sort.exe 256
../../output/counting_sort/result_256.txt
0.000457s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\counting_sort>counting_sort.exe 2048
../../output/counting_sort/result_2048.txt
0.000469s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\counting_sort>counting_sort.exe 16384
../../output/counting_sort/result_16384.txt
0.000752s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\counting_sort>counting_sort.exe 131072
../../output/counting_sort/result_131072.txt
0.002846s

E:\桌面\document\算法导论2017秋\实验\JL16110020-project1\ex2\source\counting_sort>
```

5. 关键代码。

    1. 实验一关键代码。
        - insertion_sort

```
for(i = 1; i < n; i++)
{
    strcpy(temp,string[i].str);
    temp_len = string[i].len;
    j = i - 1;
    while(CompareString1(temp,j) == -1)    //string temp < string sting[j].str
    {
        stpcpy(string[j+1].str,string[j].str);
        string[j+1].len = string[j].len;
        j--;
    }
    strcpy(string[j+1].str,temp);
    string[j+1].len = temp_len;
}
```

- heap_sort

```
void Heap_Sort(int n)
{
    int i;
    char temp[33]; //be used to exchange two strings
    int temp_len;  //be used to exchange two integers
    int heap_size = n;
    Build_Max_Heap(heap_size);
    for(i = n; i >=2; i--)
    {
        strcpy(temp,string[1].str);
        strcpy(string[1].str,string[i].str);
        strcpy(string[i].str,temp);
        temp_len = string[1].len;
        string[1].len = string[i].len;
        string[i].len = temp_len;
        heap_size--;
        Max_Heapify(heap_size,1);
    }
}
void Build_Max_Heap(int n)
{
    int i;
    for(i = n/2; i >= 1 ; i--)
    {
        Max_Heapify(n,i);
    }
}
void Max_Heapify(int n,int i)
{
    char temp[33]; //be used to exchange two strings
    int temp_len;  //be used to exchange two integers
    int l = 2*i;  //left child
    int r = 2*i + 1;  //right child
    int largest;
    if((l <= n) && (CompareString1(l,i) == 1))
        largest = l;
    else largest = i;
    if((r <= n) && (CompareString1(r,largest) == 1))
        largest = r;
    if(largest != i)
    {
        strcpy(temp,string[i].str);
        strcpy(string[i].str,string[largest].str);
        strcpy(string[largest].str,temp);
        temp_len = string[i].len;
        string[i].len = string[largest].len;
        string[largest].len = temp_len;
        Max_Heapify(n,largest);
    }
}
```

- merge_sort

```
void MergeSort(int p,int r)
{
     int q;
     if(p < r)
     {
          q = (p + r)/2;
          MergeSort(p,q);
          MergeSort(q + 1,r);
          Merge(p,q,r);
     }
}
```

```
void Merge(int p,int q,int r)
{
    int i,j,k;
    int n1 = q - p + 1;
    int n2 = r - q;
    struct String left[n1+1];
    struct String right[n2+1];
    for(i = 0; i < n1 ; i++)
        left[i] = string[p+i];
    for(j = 0; j < n2; j++)
        right[j] = string[q+1+j];
    strcpy(left[n1].str,"zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz"); //32 'z' , the biggest
    left[n1].len = 32;
    stpcpy(right[n2].str,"zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz");  //32
    right[n2].len = 32;

    i = 0,j = 0;
    for(k = p; k <= r; k++)
    {
        if(CompareString1(left[i].str,right[j].str) == -1)  //l[i].str < r[j].str
        {
            string[k] = left[i];
            i++;
        }
        else
        {
            string[k] = right[j];
            j++;
        }
    }
}
```

- quick_sort

```
void QuickSort(int p,int r)
{
     int q;
     if(p < r)
     {
          q = Partition(p,r);
          QuickSort(p,q-1);
          QuickSort(q+1,r);
     }
}
```

```
int Partition(int p,int r)
{
    int i,j;
    struct String x = string[r];
    struct String temp;
    i = p - 1;
    for(j = p; j < r; j++)
    {
        if(CompareString1(string[j].str,x.str) != 1)
        {
            i++;
            temp = string[i];
            string[i] = string[j];
            string[j] = temp;
        }
    }
    temp = string[i+1];
    string[i+1] = string[r];
    string[r] = temp;
    return i+1;
}
```

2. 实验二关键代码。
   - bubble_sort

```
void BubbluSort(int n)
{
    int i,j;
    int temp;
    for(i = 0 ; i < n - 1; i++)
    {
        for(j = 0; j < n - i - 1 ; j++)
        {
            if(number[j] > number[j+1])
            {
                temp = number[j];
                number[j] = number[j+1];
                number[j+1] = temp;
            }
        }
    }
}
```

   - quick_sort

```
void QuickSort(int p,int r)
{
    int q;
    if(p < r)
    {
        q = Partition(p,r);
        QuickSort(p,q-1);
        QuickSort(q+1,r);
    }
}
```

```
int Partition(int p, int r)
{
    int i,j,temp;
    int x = number[r];
    i = p - 1;
    for(j = p; j <r;j++)
    {
        if(number[j] <= x)
        {
            i++;
            temp = number[i];
            number[i] = number[j];
            number[j] = temp;
        }
    }
    temp = number[i+1];
    number[i+1] = number[r];
    number[r] = temp;
    return i+1;
}
```

- radix_sort

```
void CountingSort(int n,int t)
{
    int i,j;
    int c[10];  //'0'~'9'
    for(i = 0; i < 10 ; i++)
        c[i] = 0;
    for(j = 0; j < n ; j++)  //now c[i] contains the number of elements equal to i
        c[number[j].num[t]-48]++;
    for(i = 1; i < 10 ; i++)
        c[i] += c[i-1];
    for(j = n-1; j>=0 ; j--)
    {
        strcpy(sorted_number[c[number[j].num[t]-48] - 1].num,number[j].num);
        c[number[j].num[t]-48]--;
    }
    for(j = 0; j < n ; j++)
    {
        strcpy(number[j].num,sorted_number[j].num);
    }
}

void RadixSort(int n,int d)
{
    int i;
    for(i = 0; i < d; i++)
        CountingSort(n,d-i-1);  //use counting_sort to sort the (d-i)th digit of n numbers
}
```

- counting_sort

```
void CountingSort(int n)
{
    int i,j;
    int c[65536];  //0~65535
    for(i = 0; i < 65536 ; i++)
        c[i] = 0;
    for(j = 0; j < n ; j++)  //now c[i] contains the number of elements equal to i
        c[number[j]]++;
    for(i = 1; i < 65536 ; i++)
        c[i] += c[i-1];
    for(j = n-1; j>=0 ; j--)
    {
        sorted_number[c[number[j]] - 1] = number[j];
        c[number[j]]--;
    }
}
```

实验结果、分析

1. 如何记录排序算法所消耗的时间

   之前考虑过到底是计算整个程序运行的时间，还是只计算排序算法运行的时间，最后觉得应该只计算算法运行的时间更为合理，所以本次实验我都是只计算排序算法消耗的时间，举一个例子吧：

```
LARGE_INTEGER  timeStart;  //the start time
LARGE_INTEGER  timeEnd;  //the end time
LARGE_INTEGER  frequency; //the frequency of CPU
QueryPerformanceFrequency(&frequency);
double quadpart = (double)frequency.QuadPart;

QueryPerformanceCounter(&timeStart);

Heap_Sort(n);

QueryPerformanceCounter(&timeEnd);
float elapsed = (timeEnd.QuadPart - timeStart.QuadPart) / quadpart;
```

   如图所示，我计算时间都是在排序算法调用前记录一次，排序算法调用后再记录一次。

2. 使用什么工具来记录算法消耗的时间

   首先考虑本次实验需要什么精度的时间测量工具，最开始尝试过clock函数，但是由于其精度是ms，由于现在的计算机能力都比较强，所以对于所有的8个算法，当规模稍小时运行时间都是0，所以专门去查了一下精度更高的函数，得到一张表，如下：

核心函数	头文件	函数库	精度	准确度
QueryPerformanceCounter	windows.h	API	us	非常准确
GetTickTount	windows.h	API	ms	准确
clock	time.h	C函数	ms	较准确
time	time.h	C函数	s	很准确
ftime	sys/timeb.h	C函数	ms	较准确

   所以我使用精度为微秒的QueryPerformanceCounter函数。

3. 实验所得数据

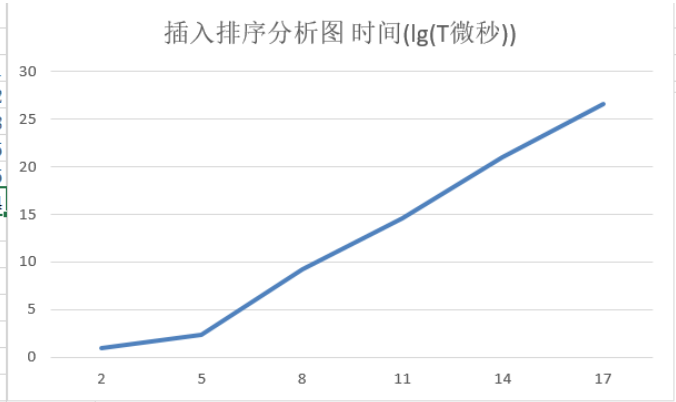
规模 / 时间(微秒) 算法	4	32	256	2048	16384	131072
insertion_sort	2	15	582	25743	1577331	97954277
heap_sort	3	13	138	1482	13691	146516
merge_sort	1	13	131	1284	14095	125962
quick_sort	1	9	115	1071	10941	121167
bubble_sort	0	3	145	8711	743413	53364723
quick_sort	0	2	22	187	1749	17391
radix_sort	22	7	28	201	1678	15383
counting_sort	447	428	457	469	752	2846

4. 图表分析

   1. 实验一算法分析
      - insertion_sort

插入排序分析图

规模(lgN)	时间(lg(T微秒))
2	1
5	2.32
8	9.18
11	14.65
14	20.95
17	26.54



注:因为理论上插入排序是n^2阶的，所以我对输入规模和运行时间都取了对数，分别作为x轴和y轴，这样就变成了线性关系了，观察曲线，大概是线性的。

- heap_sort

堆排序分析图

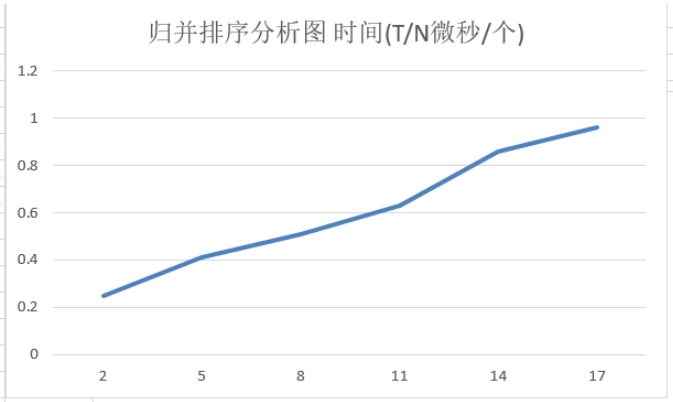
规模(lgN)	时间(T/N微秒/个)
2	0.75
5	0.41
8	0.54
11	0.72
14	0.84
17	1.12

注:因为理论上堆排序是nlgn阶的，所以我对输入规模取对数，对运行时间除以n，分别作为x轴和y轴，这样就变成了线性关系了，观察曲线，除了第一个数据反常，大概是线性的。

- merge_sort

归并排序分析图

规模(lgN)	时间(T/N微秒/个)
2	0.25
5	0.41
8	0.51
11	0.63
14	0.86
17	0.96



注:因为理论上归并排序是nlgn阶的，所以我对输入规模取对数，对运行时间除以n，分别作为x轴和y轴，这样就变成了线性关系了，观察曲线，大概是线性的。

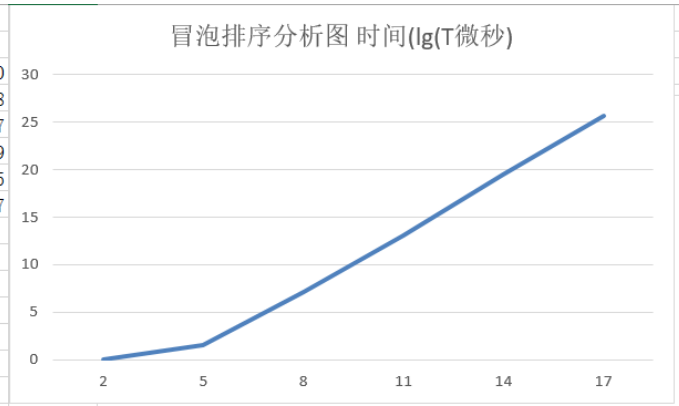
- quick_sort

快速排序分析图	
规模(1gN)	时间(T/N微秒/个)
2	0.25
5	0.28
8	0.44
11	0.52
14	0.67
17	0.92

注:因为理论上快速排序是nlgn阶的，所以我对输入规模取对数，对运行时间除以n，分别作为x轴和y轴，这样就变成了线性关系了，观察曲线，大概是线性的。

2. 实验二算法分析
   - bubble_sort

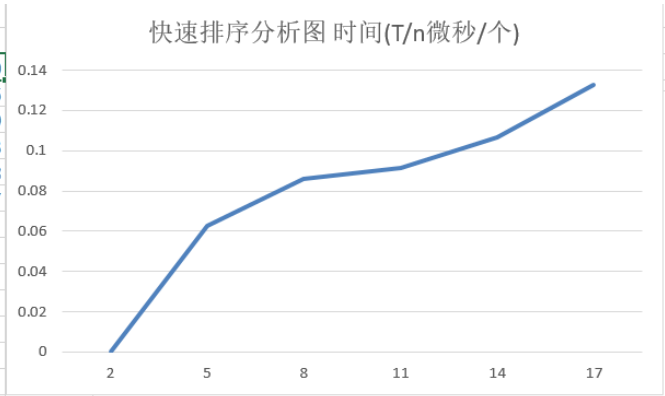
冒泡排序分析图	
规模(1gN)	时间(1g(T微秒))
2	0
5	1.58
8	7.17
11	13.09
14	19.5
17	25.67



注:因为理论上冒泡排序是n^2阶的，所以我对输入规模和运行时间都取了对数，分别作为x轴和y轴，这样就变成了线性关系了，观察曲线，大概是线性的。

   - quick_sort

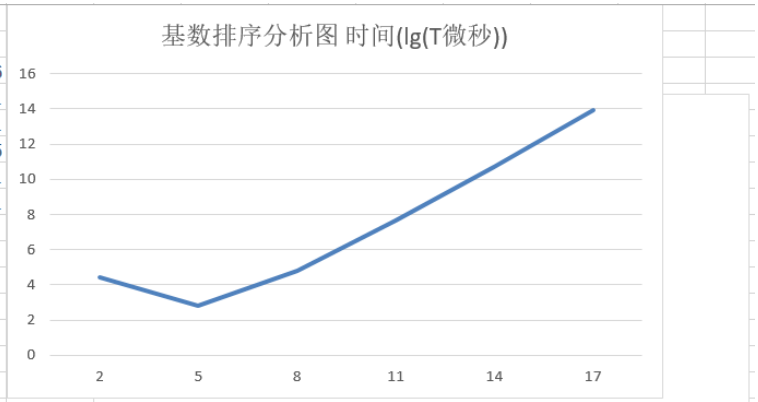
快速排序分析图	
规模(1gN)	时间(T/n微秒/个)
2	0
5	0.0625
8	0.0859
11	0.0913
14	0.1068
17	0.1327



注:因为理论上快速排序是nlgn阶的，所以我对输入规模取对数，对运行时间除以n，分别作为x轴和y轴，这样就变成了线性关系了，观察曲线，大概是线性的。

   - radix_sort

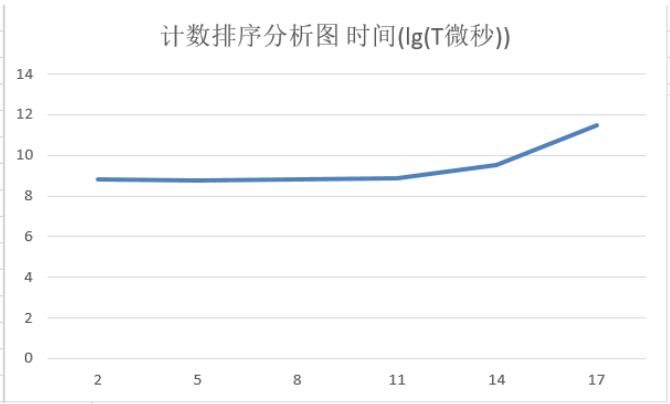
基数排序分析图	
规模(lgN)	时间(lg(T微秒))
2	4.46
5	2.81
8	4.81
11	7.65
14	10.71
17	13.91



注:因为理论上基数排序是n阶的，所以我对输入规模和运行时间都取了对数，分别作为x轴和y轴，这样就变成了线性关系了，观察曲线，除了第一个数据反常，大概是线性的。

- counting_sort

计数排序分析图	
规模(lgN)	时间(lg(T微秒))
2	8.8
5	8.74
8	8.84
11	8.87
14	9.55
17	11.47



注:因为理论上计数排序是n阶的，所以我对输入规模和运行时间都取了对数，分别作为x轴和y轴，这样就变成了线性关系了，观察曲线，前面的四个数据都很反常，观察其核心代码：

```
void CountingSort(int n)
{
    int i,j;
    int c[65536];  //0~65535
    for(i = 0; i < 65536 ; i++)
        c[i] = 0;
    for(j = 0; j < n ; j++)  //now c[i] contains the number of elements equal to i
        c[number[j]]++;
    for(i = 1; i < 65536 ; i++)
        c[i] += c[i-1];
    for(j = n-1; j>=0 ; j--)
    {
        sorted_number[c[number[j]] - 1] = number[j];
        c[number[j]]--;
    }
}
```

由代码可以看出，只有第二个for循环和第四个for循环是在对待排序数据进行处理，第一个for循环和第三个for循环都是在对一个固定大小（65536）的辅助数组操作，所以这里会花费较多的时间，如果输入数据少则影响更加明显，所以就造成了4~2048规模时，得到的运行时间都差不多。

5. 不同的算法间的比较

对于实验一，从图表中的得出的结论是：insertion_sort < heap_sort < merge_sort < quick_sort,不过在规模为2的时候，heap_sort的运行时间是大于insertion，其余数据都是按照以上的顺序来的。

对于实验二，从图表中得出的结论是：n <=32 时，counting_sort < radix_sort < bubble_sort < quick_sort。32 < n <= 256 时，counting_sort < bubble_sort < radix_sort < quick_sort。256 < n < 2048 时，bubble_sort < counting_sort < radix_sort < quick_sort。n > 2048 时，bubble_sort < quick_sort < radix_sort < counting_sort。

实验心得

1. 以前对文件操作比较陌生，经过这次实验对文件的操作熟练的多了。
2. 对调试程序更加熟悉了，本次实验因为一个BUG专门又重新学习了gdb调试。
3. 对于求算法运行时间，以前只会用精度为ms的clock,本次实验又了解了更加精确的QueryPerformanceCounter函数,其精度可达微秒级，适合本次实验使用。

注：助教若需要验证代码的正确性，只需要打开windows命令行执行对应的exe即可。