哈爾濱工業大學

数据库实验报告

题　　目　数据库查询算法的实现

专　　业　信息安全

学　　号　1180300829

学　　生　余　涛

指 导 教 师　史 建 焘

1 实验目的

掌握多路归并排序算法，并用高级语言实现

2 实验环境

JAVA 语言

3 实验内容及要求

3.1 随机生成具有 1000000 条记录的二进制文件，每条记录的长度为 16 字节

创建一个 Data 类储存数据：

```
public class Data {
    private int A;  // 4字节int
    private String B;  // 12位String

    public Data(int a, String b) {
        A = a;
        B = b;
    }
```

创建数据代码如下：

```
// 产生数据
public void create_data(String filename) {
    try {
        FileOutputStream os = new FileOutputStream(filename);
        Random rm = new Random();
        for (int i = 0; i < data_num; i++) {
            int A = rm.nextInt(Integer.MAX_VALUE);  // 生成4字节int
              System.out.println(i + "    " + A);
            os.write(sumHex(A));
            String B = getRandomStr( bytes: 12);  // 生成12位String
              System.out.println(i + "...." + B);
            os.write(B.getBytes());
        }
        os.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

3.2 分成小文件：

1MB 内存 = 1024*1024 B，而待排序文件大小=16B * 1000000，若分为 16 个子集合，则每个子集合大小为 1000*1000B < 1MB。本机的内存页大小为 4KB，1MB

内存共有 256 个内存页，16<256，因此分为 16 个子集合满足两趟扫描算法的要求。

第一趟分成小文件如下：

```
1.  // 划分小文件
2.      public void split_file(String filename) {
3.          File myfile = new File(filename);
4.          if (myfile.isFile() && myfile.exists()) {
5.              try {
6.                  FileInputStream fi = new FileInputStream(myfile);
7.                  for (int i = 0; i < block_number; i++) {  // 16 个子文件
8.                      List<Data> templist = new ArrayList<>();
9.                      for (int j = 0; j < part_file_size / 16; j++) {  // 每个文件的记录
    总数
10.                         Data tempdata = get_data(fi);
11.                         if (tempdata != null) {
12.                             templist.add(tempdata);
13.                         }
14.                     }
15.                     Collections.sort(templist, new Comparator<Data>() {  // 对子
    文件内按照 A 进行排序
16.                         @Override
17.                         public int compare(Data o1, Data o2) {
18.                             return o1.getA() - o2.getA();
19.                         }
20.                     });
21.                     FileOutputStream os = new FileOutputStream(prefix + i);
22.                     for (Data tempdata : templist) {  // 将 templist 中的所有记录读入
    一个小文件
23.                         os.write(sumHex(tempdata.getA()));
24. //                          System.out.println(tempdata.getA());
25.                         os.write(tempdata.getB().getBytes());
26. //                          System.out.println(tempdata.getB());
27.                     }
28.                     os.close();
29. //                      System.out.println("————————————————————————
    ————————————————————————————————————————————")
    ;
30.                 }
31.                 fi.close();
32.             } catch (IOException e) {
33.                 e.printStackTrace();
34.             }
35.         }
```

```
36.     }
```

3.3 归并排序：

按照 ppt 的算法生成代码即可，代码内容如下：

```
1.  public void merge_sort() {
2.          try {
3.              int num = 1;
4.  //              int all = 1;
5.              Data[][] M = new Data[block_number][256];
6.              Data[] M_compare = new Data[block_number];  // 比较缓冲区
7.              Data[] M_output = new Data[block_number];  // 输出缓冲区
8.              int[] Mi_next = new int[block_number];  // 记录 Mi 中元素移动的指针,
    即 Mi 中移到第几个元素了
9.              int[] Si_next = new int[block_number];  // 记录 Si 中读到第几块了
10.             int[] flag = new int[block_number];  // 记录某一块是否读完
11.             Data finished = new Data(-1, "finished");  // 记录已经读完的 Data
12.             FileOutputStream os = new FileOutputStream("src/merge_sort_resul
    t/result");  // 输出文件
13.
14.             int i;
15.             for (i = 0; i < block_number; i++) {
16.                 FileInputStream fi = new FileInputStream(prefix + i);
17.                 for (int j = 0; j < page_size / 16; j++) {  // 读 Si 的第一块存到 Mi
    中
18.                     M[i][j] = readData(fi);
19.                 }
20.                 M_compare[i] = M[i][0];  // 第一个元素
21.                 fi.close();
22.             }
23.             int P_output = 0;  // 输出指针起始位置
24.             while (getmin(M_compare) != -1) {  // 如果 M_compare 有最小值
25.                 i = getmin(M_compare);  // M_compare 最小值的位置
26.                 Data min_i = M_compare[i];  // M_compare 最小值
27.                 M_output[P_output] = min_i;  // 将第 i 个元素存入 M_output 的
    P_output 位置
28.                 P_output++;  // P_output 指向下一个位置
29.                 Mi_next[i] = Mi_next[i] + 1;  // 记录 Mi 中的位置的指针指向下一个
30.                 if (min_i.getA() == -1) {  // 为 finished 跳出循环
31.                     break;
32.                 }
```

```
33.             if (P_output == M_output.length) {  // 如果 P_output 指向结束位置，输出 M_output 的内容
34.                 for (Data data : M_output) {
35.                     System.out.println(num + ":" + data.getA());
36. //                      System.out.println(data.getA());
37.                     os.write(CreateData.intToBytes(data.getA()));
38.                     System.out.println(num + ":" + data.getB());
39. //                      System.out.println(data.getA());
40.                     os.write(data.getB().getBytes());
41.                     num++;
42.                 }
43.                 P_output = 0;  // P_output 置于输出缓冲区起始位置
44.             }
45.             if (Mi_next[i] < M[i].length) {  // 如果 Mi 有下一个元素，将 Mi 的下一个元素放入 M_compare 的第 i 位置
46.                 M_compare[i] = M[i][Mi_next[i]];
47.             } else {
48.                 FileInputStream fi = new FileInputStream(prefix + i);
49.                 fi.read(new byte[page_size * (Si_next[i] + 1)]);  // 先读入已经读过的块，跳过
50.                 Si_next[i] = Si_next[i] + 1;
51.                 if (flag[i] == 0) {  // 如果还有下一块
52.                     for (int j = 0; j < page_size / 16; j++) {  // 重新读入新的一块中所有内容到 M
53.                         Data tempdata = readData(fi);
54.                         if (tempdata == null) {  // 没有下一块
55.                             flag[i] = 1;
56.                             M[i][j] = finished;
57.                             break;
58.                         }
59.                         M[i][j] = tempdata;
60.                     }
61.                     Mi_next[i] = 0;
62.                     M_compare[i] = M[i][0];
63.                 } else {
64.                     M_compare[i] = finished;
65.                 }
66.                 fi.close();
67.             }
68.         } // M_compare 中所有值均为 Finished，没有最小值，算法结束
69.     } catch (IOException e) {
70.         e.printStackTrace();
71.     }
72. }
```

3.4 实验结果

（1）分成的 16 个小文件：

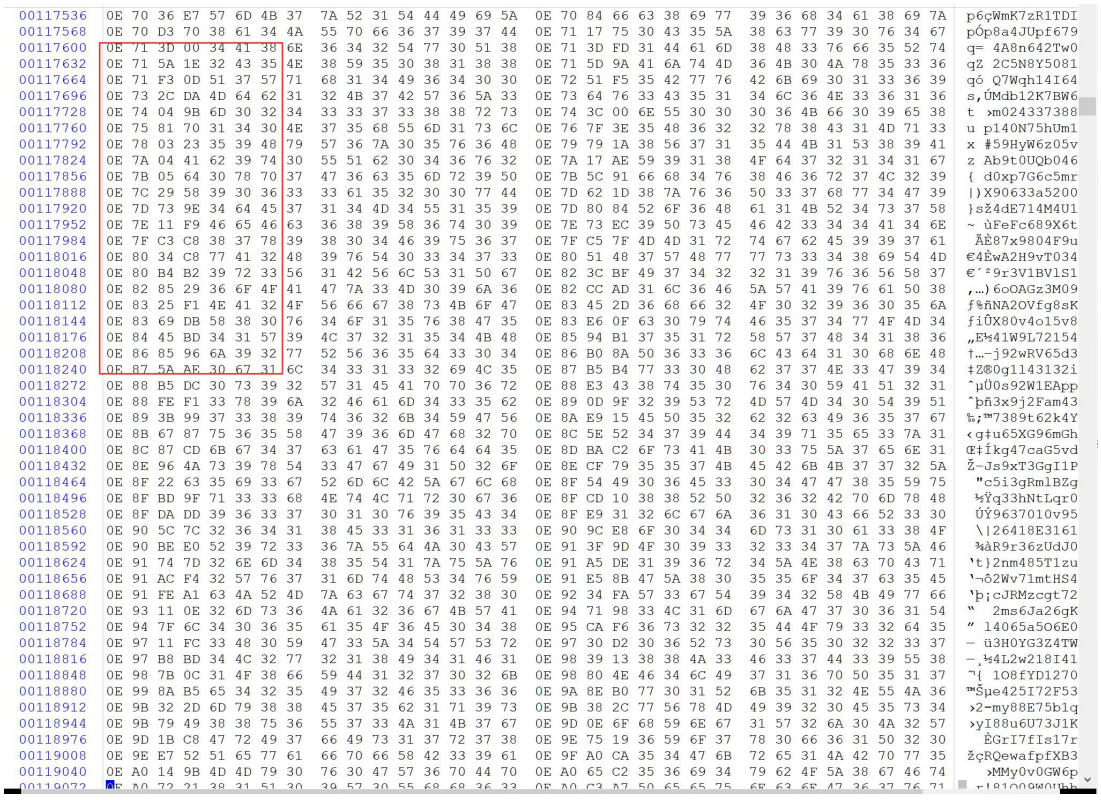
1000000 个数据总大小为 1000000 * 16 B，分成 16 个小文件，则每个小文件大小为 1000000B，为 977KB，小文件大小满足要求。

文件大小: 977 KB
1,000,000 字节

part_0	2021/5/14 18:08	文件	977 KB
part_1	2021/5/14 18:08	文件	977 KB
part_2	2021/5/14 18:08	文件	977 KB
part_3	2021/5/14 18:08	文件	977 KB
part_4	2021/5/14 18:08	文件	977 KB
part_5	2021/5/14 18:08	文件	977 KB
part_6	2021/5/14 18:08	文件	977 KB
part_7	2021/5/14 18:08	文件	977 KB
part_8	2021/5/14 18:08	文件	977 KB
part_9	2021/5/14 18:08	文件	977 KB
part_10	2021/5/14 18:08	文件	977 KB
part_11	2021/5/14 18:08	文件	977 KB
part_12	2021/5/14 18:08	文件	977 KB
part_13	2021/5/14 18:08	文件	977 KB
part_14	2021/5/14 18:08	文件	977 KB
part_15	2021/5/14 18:08	文件	977 KB

用 winhex 随便打开一个小文件查看：

按照 A 属性在小文件内部完成了排序：

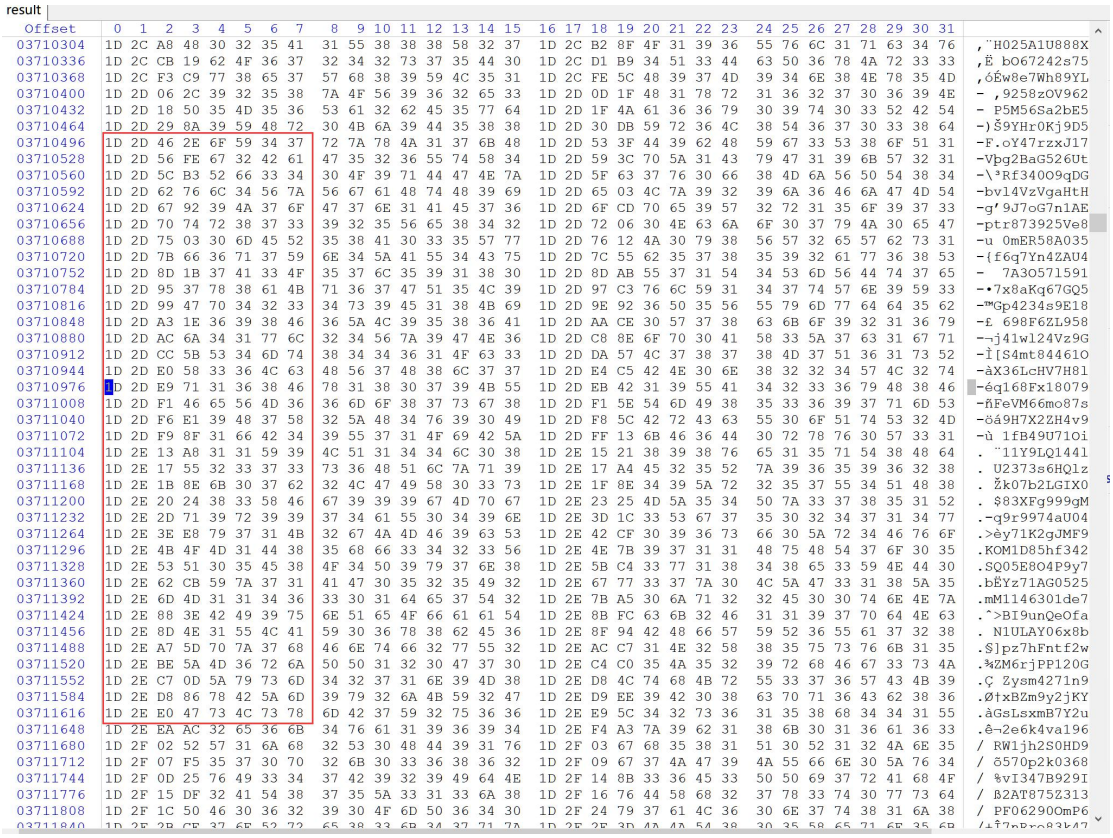


（2）最后的输出文件：

源文件大小如下：

输出文件大小如下：

刚好 1000000 个数据，文件大小合适，然后查看内部 A 属性的顺序：


result

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
03710304	1D	2C	A8	48	30	32	35	41	31	55	38	38	58	32	37	1D	2C	B2	8F	4F	31	39	36	55	76	6C	31	71	63	34	76	,¨H025A1U888X
03710336	1D	2C	CB	19	62	4F	36	37	32	34	32	73	37	35	44	30	1D	2C	D1	B9	34	51	33	44	63	50	36	78	4A	72	33	33	,Ë bO67242s75
03710368	1D	2C	F3	C9	77	38	65	37	57	68	38	39	59	4C	35	31	1D	2C	FE	5C	48	39	37	4D	39	34	6E	38	4E	78	35	4D	,óÉw8e7Wh89YL
03710400	1D	2D	06	2C	39	32	35	38	7A	4F	56	39	36	32	65	33	1D	2D	0D	1F	48	31	78	72	31	36	32	37	30	36	39	4E	- ,9258zOV962
03710432	1D	2D	18	50	35	4D	35	36	53	61	32	62	45	35	77	64	1D	2D	1F	4A	61	36	36	79	30	39	74	30	33	52	42	54	- P5M56Sa2bE5
03710464	1D	2D	29	8A	39	59	48	72	30	4B	6A	39	44	35	38	38	1D	2D	30	DB	59	72	36	4C	38	54	36	37	30	33	38	64	-)Š9YHr0Kj9D5
03710496	1D	2D	46	2E	6F	59	34	37	72	7A	78	4A	31	37	6B	48	1D	2D	53	3F	44	39	62	48	59	67	33	53	38	6F	51	31	-F.oY47rzxJ17
03710528	1D	2D	56	FE	67	32	42	61	47	35	32	36	55	74	58	34	1D	2D	59	3C	70	5A	31	43	79	47	31	39	6B	57	32	31	-Vþg2BaG526Ut
03710560	1D	2D	5C	B3	52	66	33	34	30	4F	39	71	44	47	4E	7A	1D	2D	5F	63	37	76	30	66	38	4D	6A	56	50	54	38	34	-\³Rf340O9qDG
03710592	1D	2D	62	76	6C	34	56	7A	56	67	61	48	74	48	39	69	1D	2D	65	03	4C	7A	39	32	39	6A	36	46	6A	47	4D	54	-bvl4VzVgaHtH
03710624	1D	2D	67	92	39	4A	37	6F	47	37	6E	31	41	45	37	36	1D	2D	6F	CD	70	65	39	57	32	72	31	35	6F	39	37	33	-g’9J7oG7n1AE
03710656	1D	2D	70	74	72	38	37	33	39	32	35	56	65	38	34	32	1D	2D	72	06	30	4E	63	6A	6F	30	37	79	4A	30	65	47	-ptr873925Ve8
03710688	1D	2D	75	03	30	6D	45	52	35	38	41	30	33	35	57	77	1D	2D	76	12	4A	30	79	38	56	57	32	65	57	62	73	31	-u 0mER58A035
03710720	1D	2D	7B	66	36	71	37	59	6E	34	5A	41	55	34	43	75	1D	2D	7C	55	62	35	37	38	35	39	32	61	77	36	38	53	-{f6q7Yn4ZAU4
03710752	1D	2D	8D	1B	37	41	33	4F	35	37	6C	35	39	31	38	30	1D	2D	8D	AB	55	37	31	54	34	53	6D	56	44	74	37	65	- 7A3O571591
03710784	1D	2D	95	37	78	38	61	4B	71	36	37	47	51	35	4C	39	1D	2D	97	C3	76	6C	59	31	34	37	74	57	6E	39	59	33	-•7x8aKq67GQ5
03710816	1D	2D	99	47	70	34	32	33	34	73	39	45	31	38	4B	69	1D	2D	9E	92	36	50	35	56	55	79	6D	77	64	64	35	62	-™Gp4234s9E18
03710848	1D	2D	A3	1E	36	39	38	46	36	5A	4C	39	35	38	36	41	1D	2D	AA	CE	30	57	37	38	63	6B	6F	39	32	31	36	79	-£ 698F6ZL958
03710880	1D	2D	AC	6A	34	31	77	6C	32	34	56	7A	39	47	4E	36	1D	2D	C8	8E	6F	70	30	41	58	33	5A	37	63	31	67	71	-¬j41wl24Vz9G
03710912	1D	2D	CC	5B	53	34	6D	74	38	34	34	36	31	4F	63	33	1D	2D	DA	57	4C	37	38	37	38	4D	37	51	36	31	73	52	-Ì[S4mt84461O
03710944	1D	2D	E0	58	33	36	4C	63	48	56	37	48	38	6C	37	37	1D	2D	E4	C5	42	4E	30	6E	38	32	32	34	57	4C	32	74	-àX36LcHV7H81
03710976	1D	2D	E9	71	31	36	38	46	78	31	38	30	37	39	4B	55	1D	2D	EB	42	31	39	55	41	34	32	33	36	79	48	38	46	-éq168Fx18079
03711008	1D	2D	F1	46	65	56	4D	36	36	6D	6F	38	37	73	67	38	1D	2D	F1	5E	54	6D	49	38	35	33	36	39	37	71	6D	53	-ñFeVM66mo87s
03711040	1D	2D	F6	E1	39	48	37	58	32	5A	48	34	76	39	30	49	1D	2D	F8	5C	42	72	43	63	55	30	6F	51	74	53	32	4D	-öá9H7X2ZH4v9
03711072	1D	2D	F9	8F	31	66	42	34	39	55	37	31	4F	69	42	5A	1D	2D	FF	13	6B	46	36	44	30	72	78	76	30	57	33	31	-ù 1fB49U71Oi
03711104	1D	2E	13	A8	31	31	59	39	4C	51	31	34	34	6C	30	38	1D	2E	15	21	38	39	38	76	65	31	35	71	54	38	48	64	. ¨11Y9LQ1441
03711136	1D	2E	17	55	32	33	37	33	73	36	48	51	6C	7A	71	39	1D	2E	17	A4	45	32	35	52	7A	39	36	35	39	36	32	38	. U2373s6HQlz
03711168	1D	2E	1B	8E	6B	30	37	62	32	4C	47	49	58	30	33	73	1D	2E	1F	8E	34	39	5A	72	32	35	37	55	34	51	48	38	. Žk07b2LGIX0
03711200	1D	2E	20	24	38	33	58	46	67	39	39	39	67	4D	70	67	1D	2E	23	25	4D	5A	35	34	50	7A	33	37	38	35	31	52	. $83XFg999gM
03711232	1D	2E	2D	71	39	72	39	39	37	34	61	55	30	34	39	6E	1D	2E	3D	1C	33	53	67	37	35	30	32	34	37	31	34	77	.-q9r9974aU04
03711264	1D	2E	3E	E8	79	37	31	4B	32	67	4A	4D	46	39	63	53	1D	2E	42	CF	30	39	36	73	66	30	5A	72	34	46	76	6F	.>èy71K2gJMF9
03711296	1D	2E	4B	4F	4D	31	44	38	35	68	66	33	34	32	33	56	1D	2E	4E	7B	39	37	31	31	48	75	48	54	37	6F	30	35	.KOM1D85hf342
03711328	1D	2E	53	51	30	35	45	38	4F	34	50	39	79	37	6E	38	1D	2E	5B	C4	33	77	31	38	34	38	65	33	59	4E	44	30	.SQ05E8O4P9y7
03711360	1D	2E	62	CB	59	7A	37	31	41	47	30	35	32	35	49	32	1D	2E	67	77	33	37	7A	30	4C	5A	47	33	31	38	5A	35	.bËYz71AG0525
03711392	1D	2E	6D	4D	31	31	34	36	33	30	31	64	65	37	54	32	1D	2E	7B	A5	30	6A	71	32	32	45	30	30	74	6E	4E	7A	.mM1146301de7
03711424	1D	2E	88	3E	42	49	39	75	6E	51	65	4F	66	61	61	54	1D	2E	8B	FC	63	6B	32	46	31	31	39	37	70	64	4E	63	.^>BI9unQeOfa
03711456	1D	2E	8D	4E	31	55	4C	41	59	30	36	78	38	62	45	36	1D	2E	8F	94	42	48	66	57	59	52	36	55	61	37	32	38	. N1ULAY06x8b
03711488	1D	2E	A7	5D	70	7A	37	68	46	6E	74	66	32	77	55	32	1D	2E	AC	C7	31	4E	32	58	38	35	75	73	76	6B	31	35	.§]pz7hFntf2w
03711520	1D	2E	BE	5A	4D	36	72	6A	50	50	31	32	30	47	37	30	1D	2E	C4	C0	35	4A	35	32	39	72	68	46	67	33	73	4A	.¾ZM6rjPP120G
03711552	1D	2E	C7	0D	5A	79	73	6D	34	32	37	31	6E	39	4D	38	1D	2E	D8	4C	74	68	4B	72	55	33	37	36	57	43	4B	39	.Ç Zysm4271n9
03711584	1D	2E	D8	86	78	42	5A	6D	39	79	32	6A	4B	59	32	47	1D	2E	D9	EE	39	42	30	38	63	70	71	36	43	62	38	36	.ØtxBZm9y2jKY
03711616	1D	2E	E0	47	73	4C	73	78	6D	42	37	59	32	75	36	36	1D	2E	E9	5C	34	32	73	36	31	35	38	68	34	34	31	55	.àGsLsxmB7Y2u
03711648	1D	2E	EA	AC	32	65	36	6B	34	76	61	31	39	36	39	34	1D	2E	F4	A3	7A	39	62	31	38	6B	30	31	36	61	36	33	.ê¬2e6k4va196
03711680	1D	2F	02	52	57	31	6A	68	32	53	30	48	44	39	31	76	1D	2F	03	67	68	35	38	31	51	30	52	31	32	4A	6E	35	/ RW1jh2S0HD9
03711712	1D	2F	07	F5	35	37	30	70	32	6B	30	33	36	38	36	32	1D	2F	09	67	37	4A	47	39	4A	55	66	6E	30	5A	76	34	/ õ570p2k0368
03711744	1D	2F	0D	25	76	49	33	34	37	42	39	32	39	49	64	4E	1D	2F	14	8B	33	36	45	33	50	50	69	37	72	41	68	4F	/ %vI347B929I
03711776	1D	2F	15	DF	32	41	54	38	37	35	5A	33	31	33	6A	38	1D	2F	16	76	44	58	68	32	37	78	33	74	30	77	73	64	/ ß2AT875Z313
03711808	1D	2F	1C	50	46	30	36	32	39	30	4F	6D	50	36	34	30	1D	2F	24	79	37	61	4C	36	30	6E	37	74	38	31	6A	38	/ PF06290OmP6
03711840	1D	2F	2B	CE	37	6F	52	72	65	38	33	6B	34	37	71	7A	1D	2F	2F	3D	4A	4A	54	38	30	35	58	65	71	6F	35	6B	/+Î7oRre83k47

可见输出文件完成了排序。

（3）运行时间：

第一趟运行时间：**11.172**秒
第二趟运行时间：**10.823**秒

第一趟：使用 java 自带的 comparator 对子文件按照 A 属性排序，速度较快。

第二趟：由于每次读入文件的时候是一次性读入 16 个字节作为一个新的数据，相对于一次性读一个字节，效率较高。