CS351 DATA ORGANIZATION AND MANAGEMENT
FALL 2011 – Programming Project #3
External Sorting and Merging in Java
Part 2 – P-Way Merging of Sorted Segments (December 19th 2011)
Due: January 8th 2012 at 11:59 pm

Sorting a file is one of the crucial operations which are employed in many systems. It can be a straightforward task if the whole file can be stored in the main memory at once. However, it is not always possible; the data that is processed can be larger than the main memory size. The external sorting algorithm solves the problem of limited memory for sorting large files.

As you know, the external sorting algorithm has two phases: sorting and merging. In project 2, you implemented the 1st phase, which is sorting. You extracted sorted segments with 100 records and wrote these segments to a file. Now, it is time to finish the implementation of external sorting algorithm by finding the whole sorted file. Therefore, you are to work on the 2nd phase of the algorithm, which is merging. In this phase, you construct the whole sorted file by combining the sorted segments that you create in the project 2.

The basic idea behind merging is as follows: you merge two or more sorted segments to another sorted segment that is constituted by the elements of merged segments. Merging is continued until reaching the whole sorted file. For example, if you merge 2 sorted segments (2-way merge), S1 and S2, you compare the first records of both of the segments to find the record with smaller value. Let's say that you find this record from S2. You should include this record to the newly constructed larger segment. Then, the second record of S2 is compared with the first record of S1. In this way, all of the records in S1 and S2 are compared and new larger sorted segment is constructed. The same logic is applied to all segment pairs in 2-way merge. Figure 1 shows merging steps (passes) of external sorting algorithm.

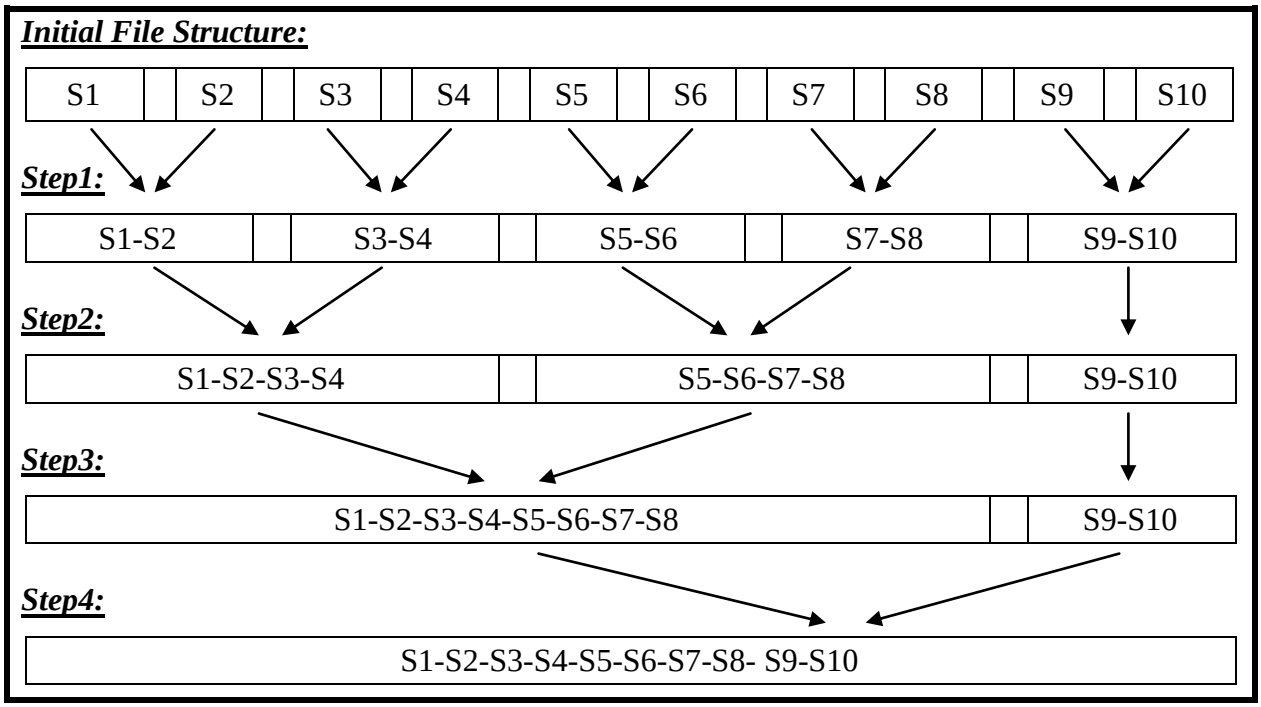

Figure 1. Before merging, we have 10 sorted segments. In the 1st merging step, 5 sorted segments are created. This time, the size of each segment doubles the sizes of initial sorted segments. 2nd step cannot include the last segment, S9-S10, in merging, due to not remaining another segment to be merged. The same situation is seen in the 3rd segment. Lastly, 4th step includes the S9-S10 and constructs whole sorted file.

In figure 1, *p* value is 2. It means that in each step, 2 segments are combined. More generally, in all steps of *p*-way merge, each ***p*** number of segments is unified to construct **one** larger sorted segment.

In this project, you implement the *p*-way merging in Java and perform a set of experiments on your program to examine the effect of *p* and memory size. The console program that you develop should take the memory size, *memsize*, and *p* as input parameters. Here *memsize* indicates the main memory available for merging in terms of number of records. For example, if memsize is equal to 200, it means that memory allocated for merging can hold 200 records (for example Salzberg uses 10MB for this purpose). You should consider that the value of *memsize* is evenly divisible by *p*. In other words, the remainder, which is found from the division of *memsize* to *p*, should be exactly zero. You are expected to check this condition in your program and if needed, ask the user again to enter new parameters until satisfying this condition. Do not assume that *p* is also divisible by the size of segments in any step of merging algorithm. Also, the *p* value should be an integer and greater than 1.

After taking the valid input parameters from the user, you read the initial segments from your output file of the program that you developed for project 2. This file name should be **sortedSegments.txt** as you did in project 2. Following to reading these segments, you constitute new larger sorted segments by combining each *p* of them. For remaining step of merging, the segments, which are constructed in the previous step, are read from file. Then, each *p* of them is combined in one larger segment with sorted order.

You should employ exactly 2 different files in the program. In each step, you read one of them and write the resulting segments to another one. The output file, which is created in the previous step, will be an input file to the current step. You should separate each sorted segment by using an empty line between them for internal steps of merging. You do not wait to construct the whole new sorted segment for writing the file. Whenever you find the record to include the new segment, you can write it to the output file. In other words, do not allocate the memory for newly constructed larger segments.

You have to use *BufferedReader* and *BufferedWriter* classes and their associated Java methods that are mentioned in the section 4 of the 2nd project assignment. The same implementation rules are also applied in this project. While reading the segments, the memory size, *memsize*, should be considered: You cannot exceed memory size. If an excess of memory is detected it will be panalized during grading. However, small number of internal Java variables, some integers like tracking and pointing the segments, storing file names, etc. are acceptable. Also, remember to include the comments to your code.

For testing your program, you can find the file, which includes actual sorted segments of students.txt file, from http://www.bilgekoroglu.com/cs351/sortedSegments.txt. (This file will be available on December 20, 2011 by 11:59 pm.)
After correctly implementing and testing your program, you need to perform a set of experiments. In your submission, you need to include a report, giving your experiment results. You need to provide the following plots and tables in your report. Also answer the questions by analyzing your results:

1. Run your program for the following parameter values. You should run your program 20 times and provide a table for **total time for merging** of each run.
    - *memsize* = 60, 120, 180, 240
    - *p* = 2, 6, 10, 30, 60
2. Plot two graphs for *p* values 10 and 60. The y-axis of each graph is total time and the x-axis is *memsize*. Using the table, which is prepared in Question 1, include total times for all possible *memsize* values (i.e. 60, 120, 180, 240). How does the memory size affect the total merging time?
3. Plot two graphs for *memsize* values 60 and 240. The y-axis of each graph is total time and the x-axis is *p*. Using the table, which is prepared in Question 1, include total times and all possible *p* values (i.e. 2, 6, 10, 30, 60). What observations can you make on the effects of *p* value when the memory size is not changed?

For submission, you must follow the rules given below.

- Name your experiment report as **LastnameFirstname.pdf** (e.g. YilmazAyse.pdf). Prepare your report using a word processor and convert it to a PDF document.
- Put all of your Java files and your report, into a .rar file of which the name should be **LastnameFirstname.rar** (e.g. YilmazAyse.rar). Do not include any input or output files except your Java classes and your report.
- The class, which has main method, should be named as **ExternalSortingPart2.java**. For grading, this class will be run.
- At the end of execution of the program, sorted output file, **sortedStudents.txt** should be generated in the same directory with other Java files.
- Read uploading rules from www.cs.bilkent.edu.tr/~ctoraman/cs351fall11/project3/submission. Also, submit your .rar files from this page.
- You can resubmit your .rar file until the due date. The previous one always replaced with the new one.
- For questions, contact with ctoraman@cs.bilkent.edu.tr.

Note that we may use a supportive tool to detect plagiarism. Do not use any code from internet. Make sure that you write your own code.