Assignment1 comp6521

Zhiyang Xu, Salar Azadani

# Introduction

In this report, we are going to introduce the two-phase multiway merge sort and its implementation. As well as our approach to make the algorithm more efficient.

**The structure of our report:**

1. Two-phase multiway merge sort introduction and its implementation.
2. The method chosen for implementation and its reason: use replacement selection sort to sort the sub list, and use Loser Tree to make it more efficient.
3. Conclusion

**The structure of our code and how to run it:**

Inside the source file of the IDEA project, there is a package called twoPMMS in which there are three classes:

1. generateData:

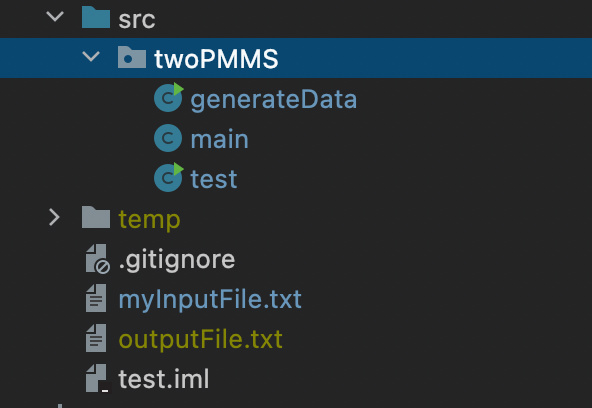
Code to generate n ints between a minimum value and a maximum one randomly. And store them inside myInputFile.txt in the project folder.

1. main

All code related to 2PMMS algorithm.

1. test

Initialize the main class and call on the method of it. And then generate a outputFile.txt contains the result of the algorithm. And a temp folder, inside of which, there are Intermediate file of the result of cutting the data into small chunks and Intermediate file after merge.



How to run:

1. run the main function inside generateData class.
2. run the main function inside test class.
3. check the result in the outputFile.txt.

# Two-phase multiway merge sort

To handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually RAM) and instead, they must reside in the slower external memory (usually a hard drive). In this class we are introduced the Two-phase multiway merge sort which requires two phases. In the sorting phase, first we cut the huge data into chunks which are small enough to fit in main memory and then they are read, sorted, and written out to a temporary file. In the merge phase, the sorted sub-files are combined into a single larger file.

So, we first divide the file into runs such that the size of a run is small enough to fit into main memory. Then sort each run in main memory using merge sort sorting algorithm. Finally merge the resulting runs together into successively bigger runs, until the file is sorted.

# The method chosen for implementation and its reason

Our code uses replacement selection sort to sort a sub list, multi-way merge to merge sub lists, loser tree to achieve optimal merge

Why use replacement selection sort to sort a sub list: When considering the issue of efficiency, we assume that there are 8 parallel lines waiting to be merged. If 2 are merged at a time, 4+2+1 times need to be merged, a total of 3 times. Each data is also operated by io 3 times, if 8 ways are merged together, each data will only be operated by io once. Therefore, reducing the number of merge passes can greatly reduce the overhead of system io. In order to reduce the number of merges, we can start from two aspects:

1. Generate the largest possible sequence: Assuming that the memory can only sort m pieces of data at a time, replacement selection sort can generate more than m and less than 2m pieces of ordered data each time.
2. Use multi-way merge

Pseudocode of the algorithm:

Make Values an array of n elements.

Make Active an array of n booleans, all initially true.

Read n values from memory into Values.

Until no values are left to process:

Find the smallest value that is still active.

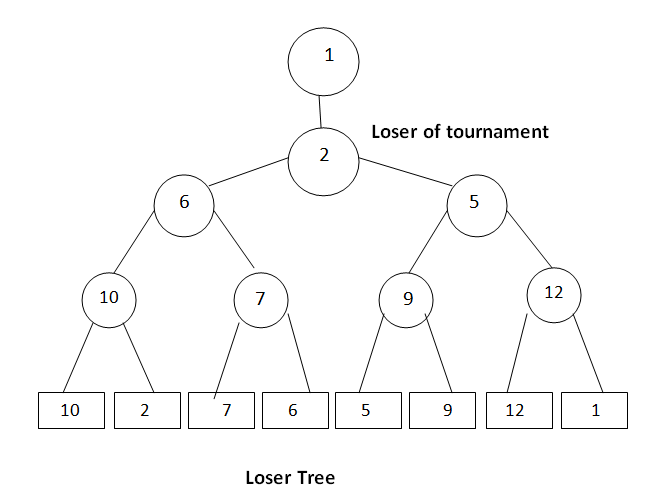
Write it to the output device.

Read from the input device into the slot where the old element was.

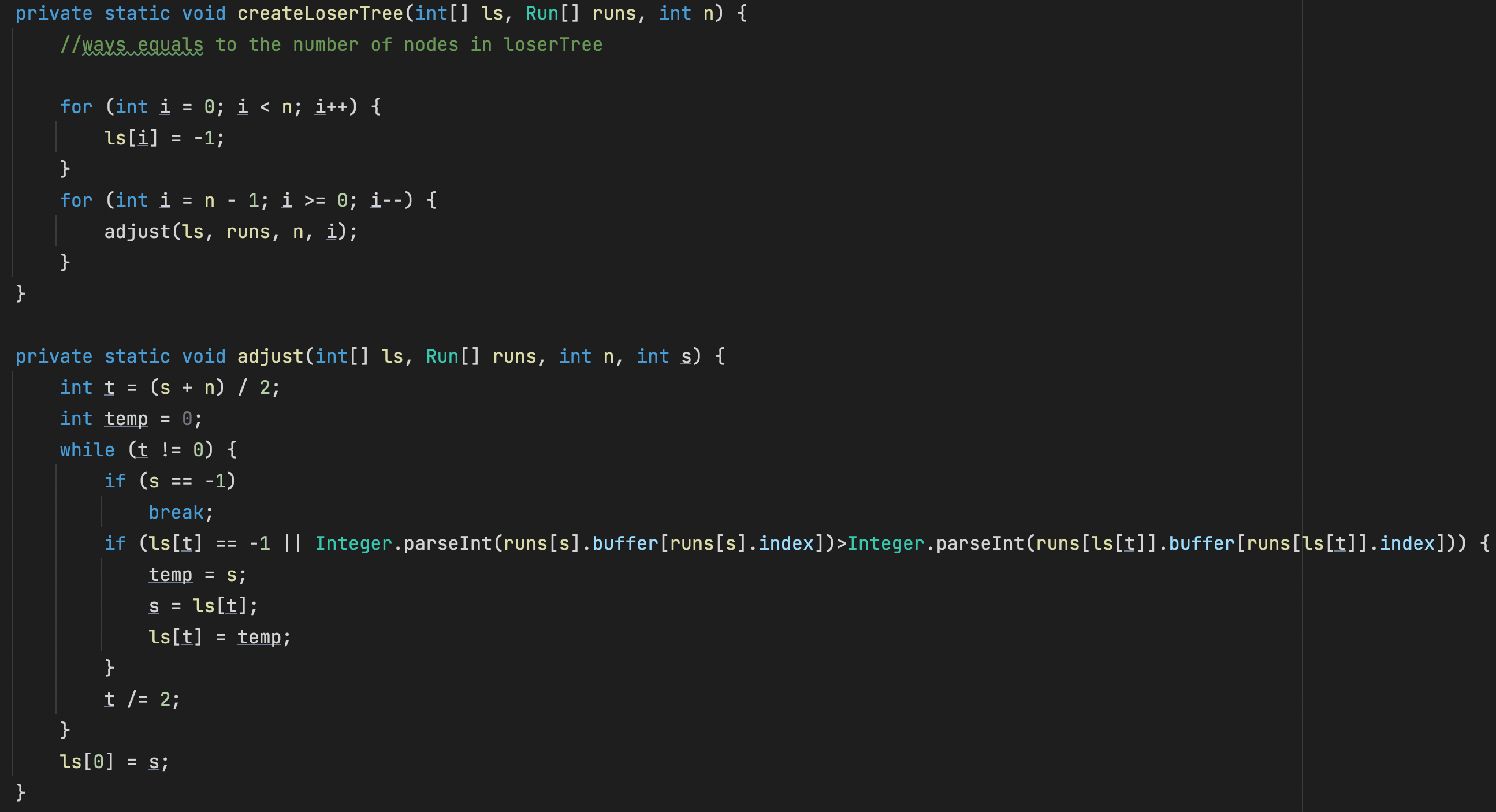
If it was smaller than the old element, mark the old slot inactive.

If all slots are inactive, mark them all active.

Why use the loser tree? If there are currently k paths and m sequences need to be merged, each output data needs to be compared for k-1 times, and the time complexity is O(n). The use of the loser tree can only be done after initialization When it needs to compare k-1 times, then only logkM times are needed each time, and the time complexity is O(logk). The principle is like a group game, each person does not have to compete with everyone else once, but is divided into two groups, and the winner is only compared with the winner of the other group.



My array implementation of loser tree:



# Conclusion