User guide:

The program distributes the sorting of the records in input file according to the attribute number and order provided by the user by creating multiple processes. The output data is stored in a file provided by the user where the fields of each record are comma-seperated.

The user will invoke the program using the following invocation and flags:

```
./myhie -i InputFile -k NumOfWorkers -r -a AttributeNumber -o Order -s OutputFile
```

Where -i is the flag for input file, -k is the flag for number of sorting nodes, -r indicates random splitting, -a is the attribute for sorting comparisons, -o is the flag for order, -s it the flag for output file. Of these, only the -r flag is optional and the rest must be provided by the user.

Upon invocation, the program performs the sorting tasks and prints timing statistics for processes and number of signals caught by the root at the completion of task by each node.

The user is also given a message when writing to the output file starts so as to confirm that the process is still running and is not waiting for some interrupt from the user.

Technical implementation:

The program starts from the root node which creates a coord node by using exec command and passing it parameters specified in the comments of the code. The coord then coordinates the worker of k sorters it creates by fork. Each sorting program is then loaded through exec commands and is passed a list of parameters, including the names of pipes it will use for communications. Here, each sorter takes charge of its own range of records and sorts them using the sorting algorithms deployed i.e. bubble sort and insertion sort for each alternate sorter. The coordinator meanwhile creates a merger node through fork where named pipes are used to read timing data and sorted data being sent by each of the k workers through the same named pipes. Through the use of an array, the merger is able to keep track of the name of pipes used by the i^{th} worker and, thus, uses them to communicate with the sorter. The sorted data is stored in the merger node into k arrays of size equal to the range of record sorted by the respective sorter. These partially sorted records are then stored in a single array and sorted using bubble sort algorithm. Thus, the concurrent running of sorters to partially sort arrays before they are merged into a final sorted array reduces the total runtime required. The merger then writes this sorted data to the user provided output file. Upon completion of its tasks, the merger sends SIGUSR2 signal to the root. Side by side, the sorters also send SIGUSR1 signals to the root when they are done. Timing statistics are also recorded and printed for each of the sorters and the merger node. Finally, the total turnaround time is computed as the sum of times for sorting and merging and printed. The user is also provided with the time statistics of

root and the number of SIGUSR1 and SIGUSR2 signals caught by the root using the handlers created to catch these signals.

Design Choices:

The use of structs to store records allows for efficient storing and fetching of records, as well as passing them through pipes. Arrays of size k are created to keep track of information for each of the sorters such as pipe names, the ranges worked on by the sorters etc. This allows for ease of operations, comparisons, as well as use while limiting the number of read and write operations on the pipes without having to pipe this data between process. This helps reduce unnecessary piping which may result in conflicts, errors, and overheads. Furthermore, reading the data from sorters and storing it as partially sorted data into 'sub-arrays' or subsets of the original data reduces the overhead and time needed to sort the data in the merge process. Furthermore, the sorters are created by coord and then loaded through exec by passing them only the relevant arguments that will be needed by them. This is maintained throughout the program in order to maintain integrity of data transfer from the root all the way to the sorters and merger. Root's PID is used to send signals to it by the merger and sorters but since these processes are the grandchildren of root node, the PID is passed as an argument in exec for sorters and is also maintained in the merger node since it is coord created through fork. The fork stores root PID using getppid(). Checks throughout the program for potential failures and waiting for children to finish before continuing with execution of parent allows for effective running of the entire program.

Thus, through the various intricate design choices including the ones described above, the outlined task is completed as instructed. Due to no specifications listed regarding runtime efficiency of sorting algorithms in sorters, insertion sort and bubble sort are deployed. This can lead to a high runtime for large datasets. However, any sorting algorithm could be implemented inside the sorters and the entire program from the root will still perform the task correctly due to no dependency of task completion on the type of sorting used.

Credits:

1. Replace_multiple_spaces_with_one used to normalize spaces to single spaces between fields of each data record. Credits to original author: https://github.com/sangeeths/strings/blob/master/replace-multiple-spaces-with-one.c