Algorithm design and problem-solving project

Name: Ruairí Kilgallon

Brief:

## ***Introduction***

An engineering company manufactures aeronautical equipment on 4 different production lines at its factory in Dublin. The production line is sophisticated and audited to the highest quality standards.

Each production line is used for the manufacture of multiple products in a multi-stage production process which ends when the product is packed for dispatch.

Quality Assurance (QA) is important to the company. Each product is manufactured in a single identified batch and faults or issues identified at every stage are logged and tracked by a QA engineer.

Working in the company's IT department, you have been given access to the QA logs of each production line. You have been asked to assist the management team in analysing the QA data from the manufacturing process. The QA logs contain the following data and are newly created for each month:-

* 1. Line Code – Numeric
  2. Batch code - Numeric
  3. Batch date & time – numbered for day of month, hour of day, minute of hour.
  4. Product Id - numeric
  5. Issue Code & description - numeric + text
  6. Resolution code & description – numeric + text
  7. Reporting employee id – numeric

Task 1:

* The production line logs are either ordered by date & time or may be in a random order for each day. Prepare a report for each line in Product id, Issue code, date & time order.
* There are huge amounts of data stored, the running time of this algorithm should be O(NLog(N)) or better.
* I was tasked with sorting the logs in a way that the log IDs are grouped together and within the ID the logs are sorted by issue code then date and time. To solve this in O(NLog(N)) or better I implemented a form of quick sort.

Psuedocode

LINE\_SIZE=4;

LOG\_SIZE=5;

For(i=0; i<LINE\_SIZE; i++) {

Count=0;

for(j=0; i<LINE\_SIZE; i++) {

if(logs[j][0].line\_code == i+1) {

for(k=0; k<LOG\_SIZE; k++) {

sorted\_logs[i][Count++] = logs[j][k]

}

}

}

Qsort(sortedlogs[i], LOG\_SIZE, sizeof(struct QA\_logs), compare)

Key points:

The compare function that is seen within the qsort function is used to compare the logs by first checking the ID then issue code then date and time the pseudocode is below

If(logA->product\_ID != logB->product\_ID) {

Return logA->product\_ID – logB->product\_ID;

}

If(logA->issue\_code != logB->issue\_code) {

Return logA ->issue\_code – logB->issue\_code;

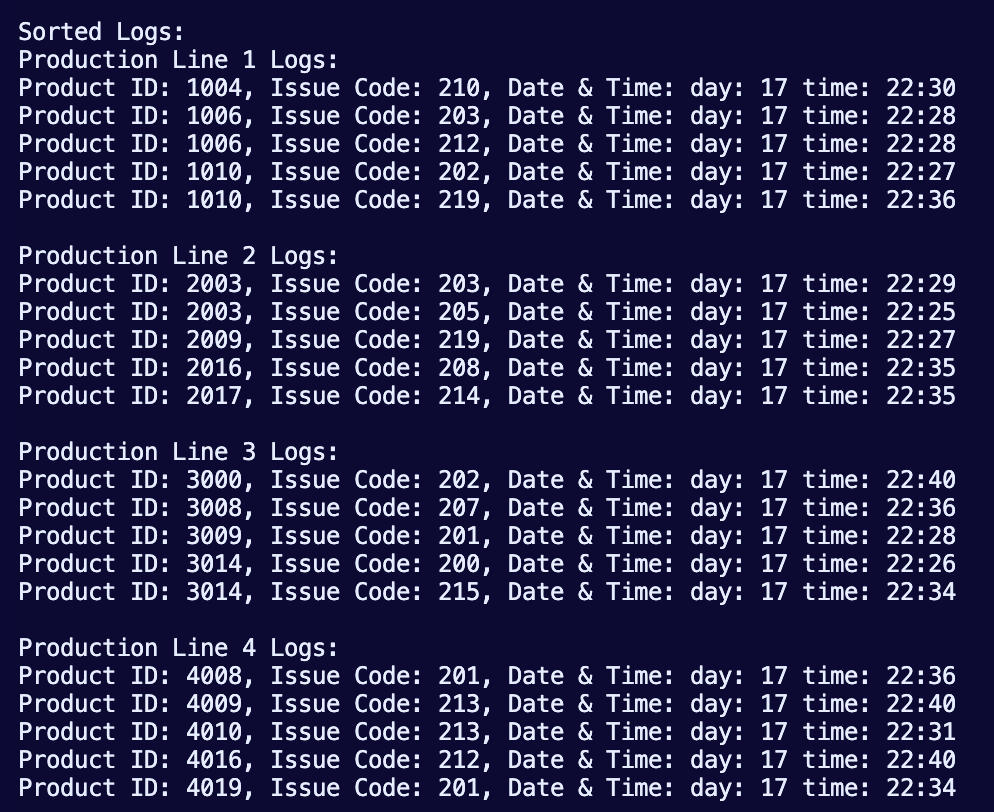
}

Return logA->batch\_date\_time – logB-> batch\_date\_time

I also used the library function “Qsort” instead of writing the whole algorithm from scratch. The quicksort works by first choosing a pivot element (typically the last element in the array) which is then compared to each element in the array then all the elements less than the pivot are placed to the left of it. This cycle continues until the array is sorted.

Results

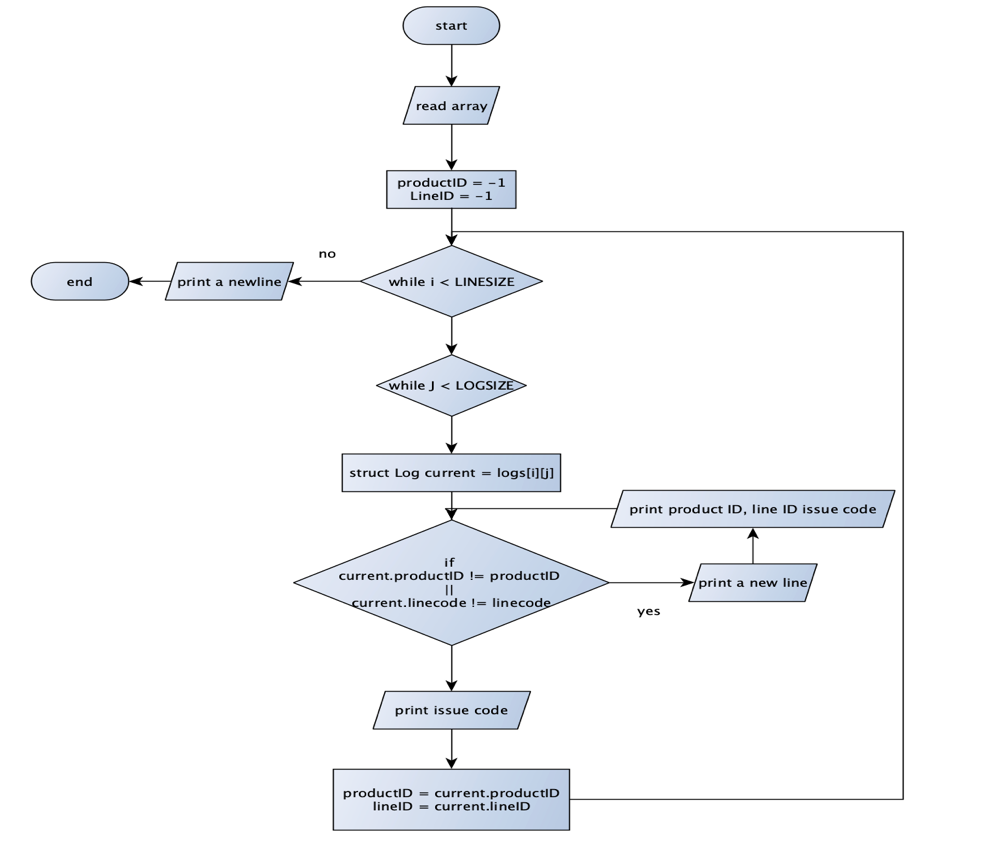
For this I randomly inputted data and as you can see below the data was sorted to match the task requirements of product ID -> issue code -> date & time with an efficiency of O(NLog(N)) or better.



Task 2:

* Due to changes in the manufacturing process, the same product can be manufactured on different lines.
* Prepare a report which uses a *single list* to report issue codes by product Id and line Id for all production lines.
* There are huge amounts of data stored, the running time of this algorithm should be O(N) or better.

For task 2 I was tasked with merging the QA data into a single list and show the issue code for the product ID and line ID this should be done in o(N) or better. To do this I created a function that loops through the logs while comparing the current product ID and line ID with their previous ones, if they are different they are printed on the next line if they aren't they are printed on the same line thus grouping them together. The flowchart and pseudocode below demonstrate this.



Pseudocode

ProductID, LineID = -1;

For(i=0; i<LINESIZE; i++) {

For(j=0; j<LOGSIZE; j++) {

struct Log current = logs[i][j];

if(current.productID != productID || current.Linecode != lineID) {

if(currentproductID != -1) {

printf(“\n”)

}

printf(“current.productID, current.linecode, current.issuecode);

}

Else {

Printf(currentissuecode);

ProductID = current.productID;

LineID = current.linecode;

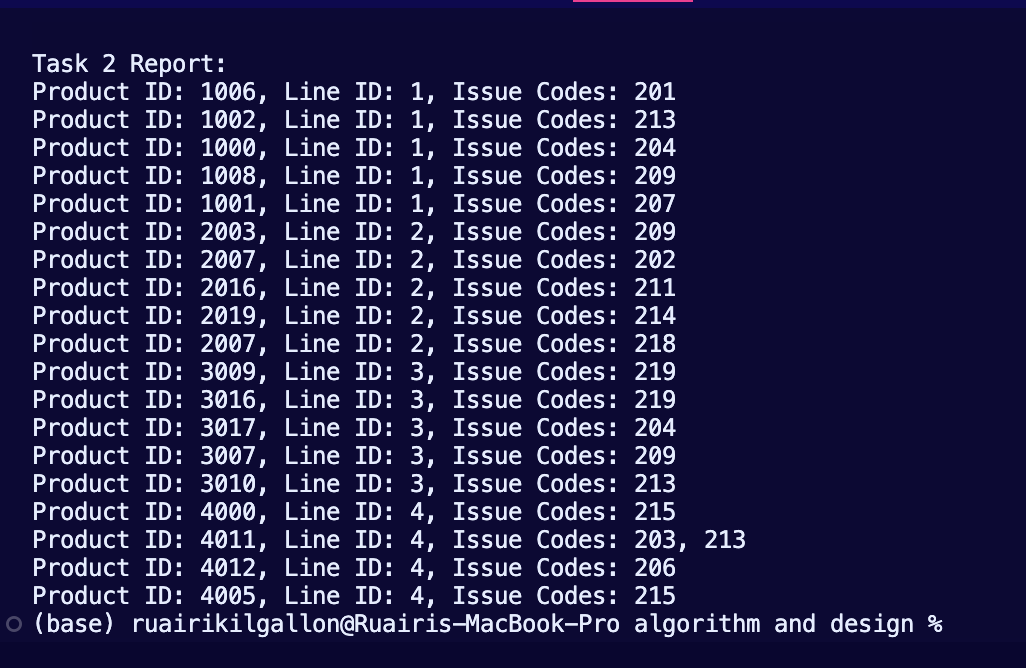
}

}

Printf(“\n”)

Results

This function takes the sorted data from the function developed from task 1 and passes it into this one, it groups the issue codes by product ID and line ID with a time complexity of O(N) as the logs were already sorted in task 1 therefore meeting the specifications of task 2. the screenshot bellow shows how the data is outputted



Task 3:

* Provide a facility to search for the earliest occurrence of an issue code for a given product id across all production lines.
* There are huge amounts of data stored, the running time of this algorithm should be O(Log(N)) or better.

To solve task 3 I decided to implement a binary search to find the earliest occurrence of an issue code for a given product ID. The function i used finds the earliest occurrence of a product ID across all production lines. It initializes variables to track the earliest occurrence and loops through each production line, and performs binary search within sorted logs. Once a matching product IDis found, it updates the earliest occurrence.it outputs the earliest occurrence found or indicates if the product ID was not found, achieving a time complexity of O(log N) as required for Task 3.

int earliestLine = LINESIZE;

int earliestIndex = LOGSIZE;

for (int i = 0; i < LINESIZE; i++) {

int left = 0;

int right = LOGSIZE - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (logs[i][mid].product\_id == productID) {

if (i < earliestLine || (i == earliestLine && mid < earliestIndex)) { earliestLine = i;

earliestIndex = mid;

}

right = mid - 1;

} else if (logs[i][mid].product\_id < productID) {

left = mid + 1;

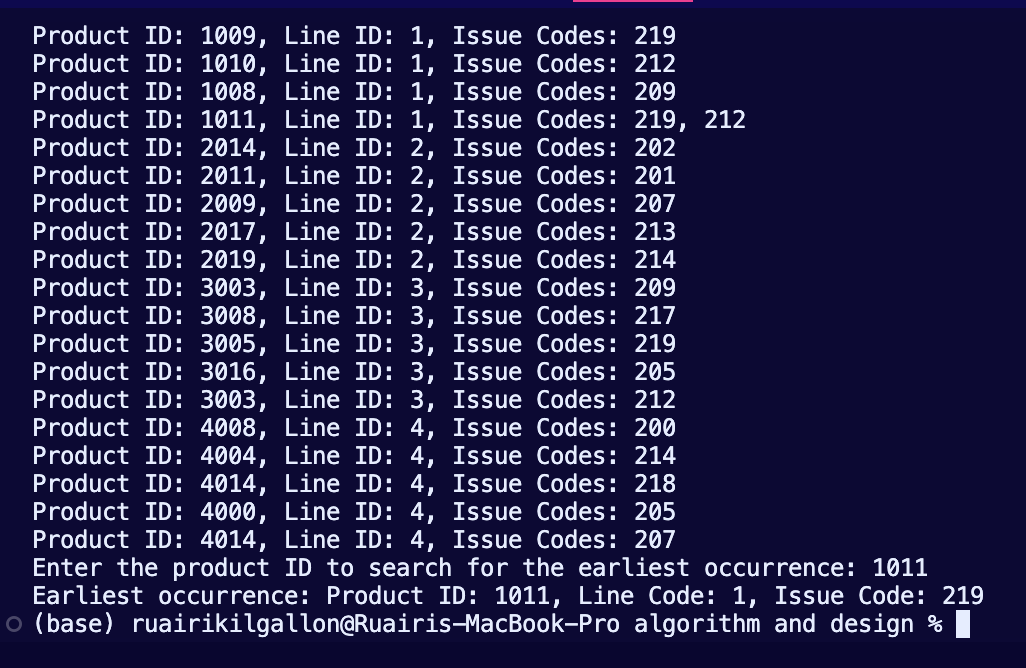
} else { right = mid - 1;

}

}

Results

The function created searches for the earliest occurrence of the product ID with a time complexity of O(Log(N)) as N is the number of log entries and log for the production lines. This is seen in the picture below



Task 4:

* Provide a report which summarises the number of issues reported for a product across all production lines.
* There are huge amounts of data stored, the running time of this algorithm should be O(N) or better.

To solve task 4 i created a function that loops through the logs and production line by using nested loops while counting each product ID with its corresponding index in a separate array called “issuecount”. This is conveyed in the pseudocode below.

Int issuecount[10000] = {0};

For(i=0; i<LINESIZE; i++) {

For(j=0; j<LOGSIZE; j++) {

int productID = logs[i][j].productID;

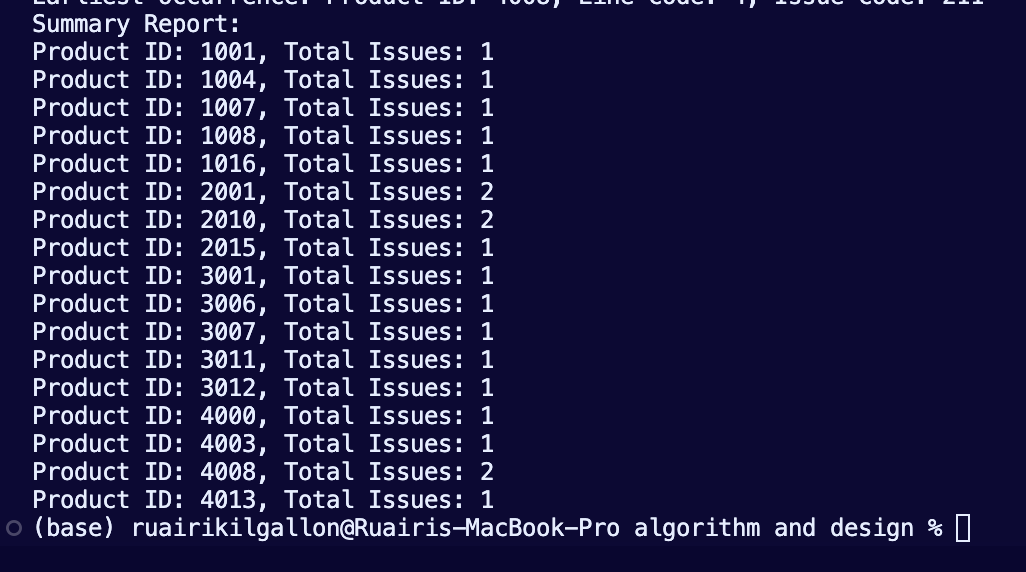
issuecount++;

}

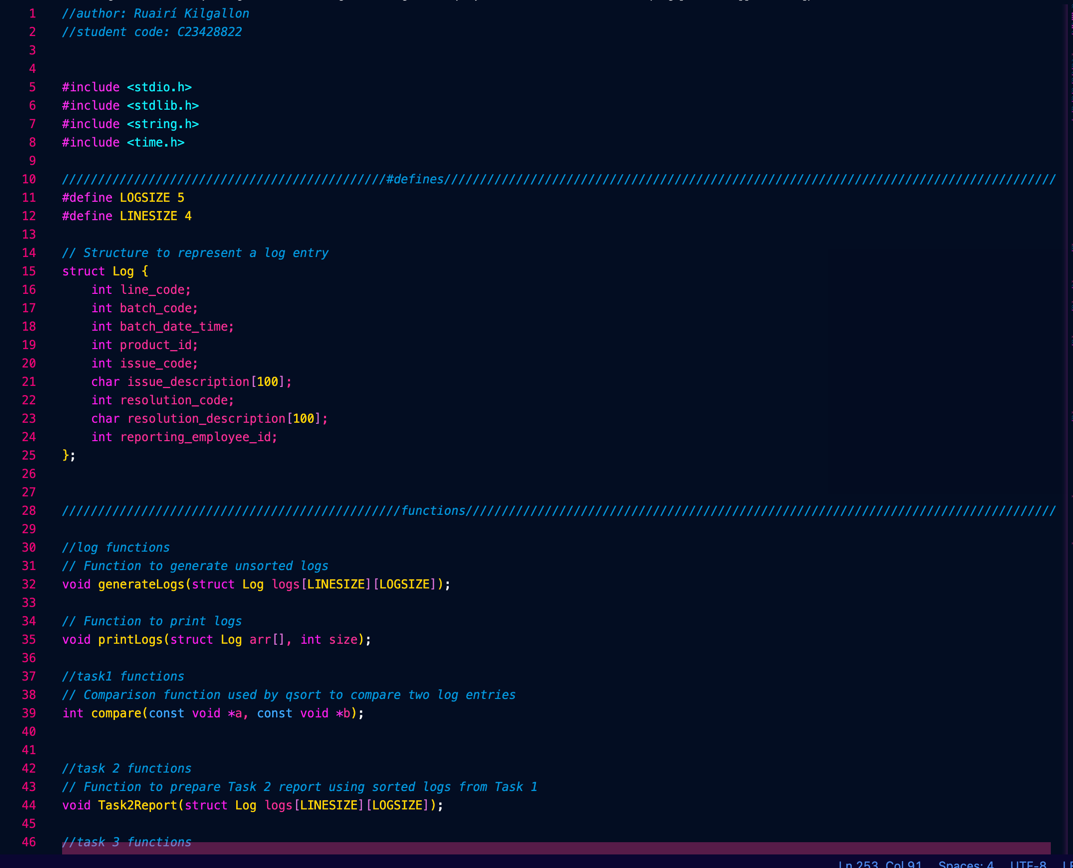
}

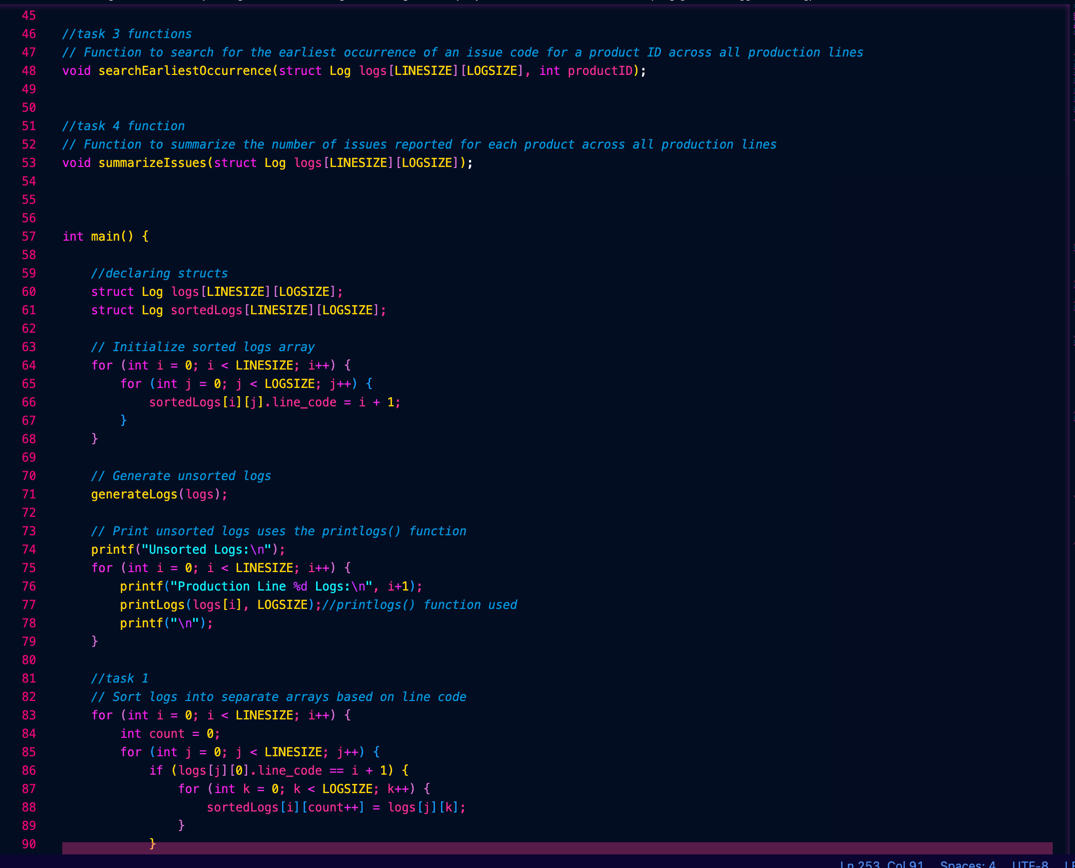
results

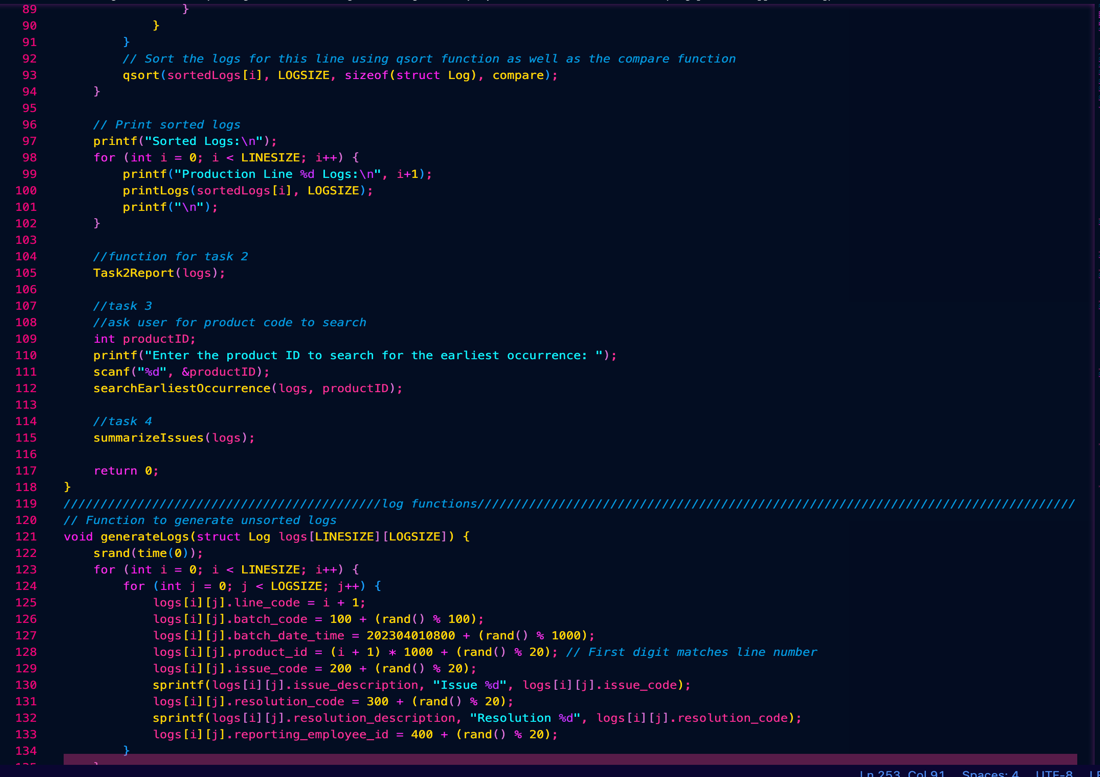
The function created achieves a time complexity of O(N) as it loops through the Logs ie “N” while updating the count which has the same time complexity. The results are in the screenshot below.

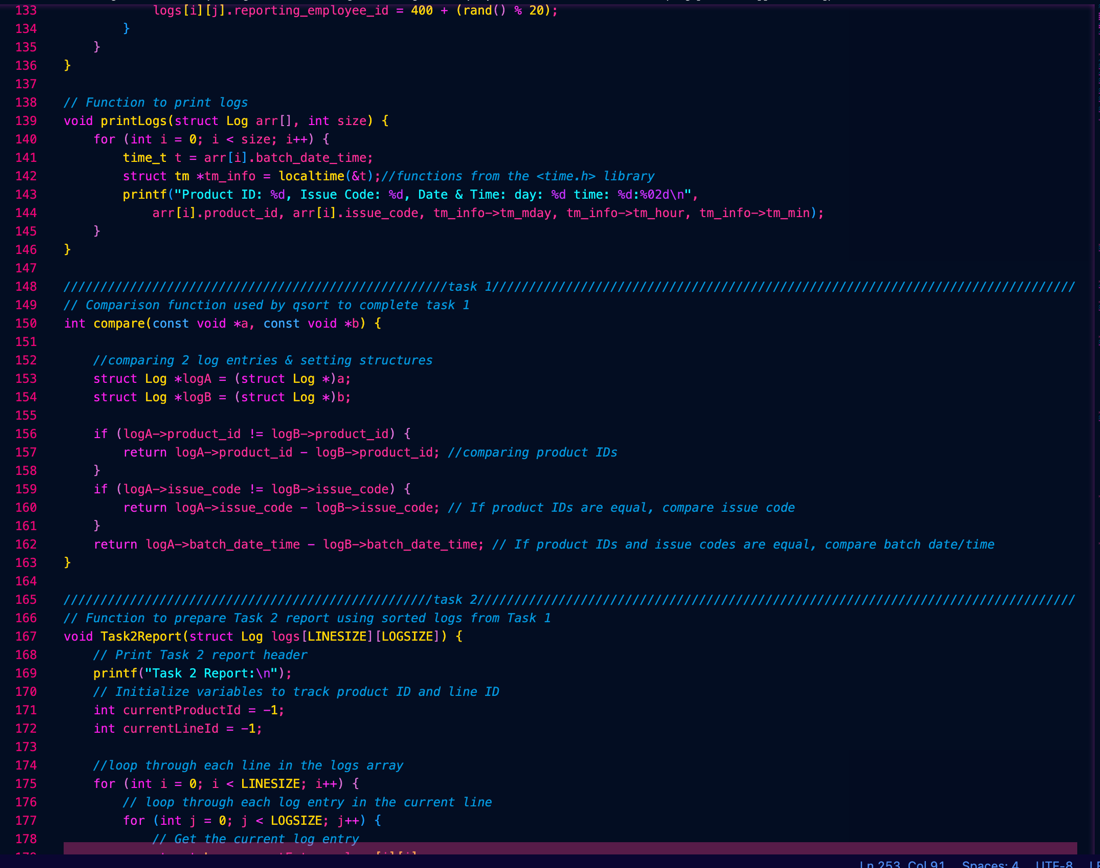


Full code









}