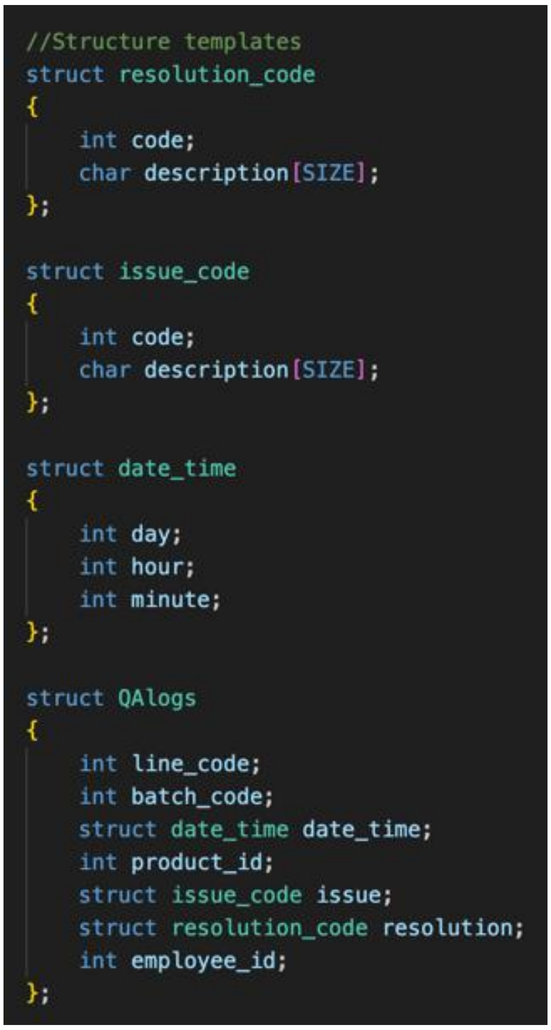
[redacted]

Brief:

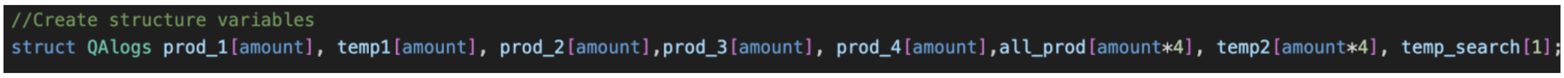
I have been given access to the QA logs for an engineering companies' 4 different production lines that all manufacture multiple products. All the faults and issues with each product in a single identified batch is logged by the QA engineer. The QA logs contain the line code, batch code, batch date and time, product id, issue code and description and the reporting employee id for the product. I must assist the management team in analysing the QA data from the manufacturing process. The design requirement's are firstly sorting the data in product id, issue code and date and time order with an algorithm that should be O(nlog(n)) or better, secondly combine all the data from all production lines into a single list and sort the issue codes by product id and line id with an algorithm that should be O(n) or better, thirdly create a facility that provides you with when a certain issue code happened first for a certain product id with an algorithm that should be O(log(n)) or better and finally summarise the amount of issues each product has had across all production lines with an algorithm that is O(n) or better.

Data Structure:

The data structure I designed was a structure that I could then use to create arrays of structures for each production line and for different use cases.



```c
//Structure templates
struct resolution_code
{
    int code;
    char description[SIZE];
};

struct issue_code
{
    int code;
    char description[SIZE];
};

struct date_time
{
    int day;
    int hour;
    int minute;
};

struct QAlogs
{
    int line_code;
    int batch_code;
    struct date_time date_time;
    int product_id;
    struct issue_code issue;
    struct resolution_code resolution;
    int employee_id;
};
```

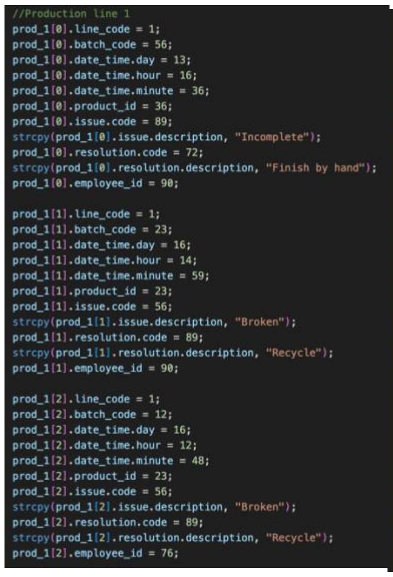


```c
//Create structure variables
struct QAlogs prod_1[amount], temp1[amount], prod_2[amount],prod_3[amount], prod_4[amount],all_prod[amount*4], temp2[amount*4], temp_search[1];
```

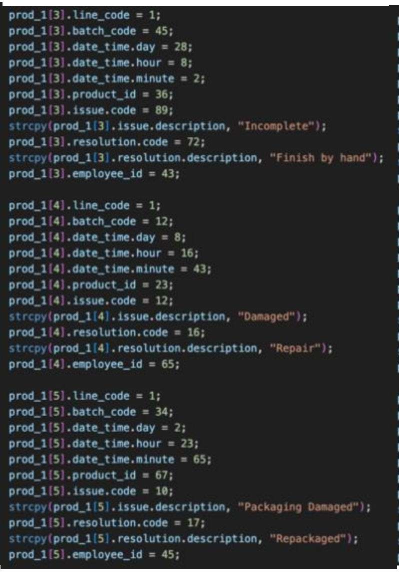
Test Data:

I created 13 test cases(13 structures) for each production line using the structure template and put them all in inside their own array of structures. Using this test data, I can ensure that my implementation for each task will meet the design requirements. This is as a result of some test cases having some of the same data while some also having different data. For example, production line issue 2(prod_1[1]) has the same product_id, issue code and description, resolution code and description and finally day, as issue 3 (prod_1[2]. This allows each design I create for each task to be tested in their own way and make sure the design works with all possibilities.

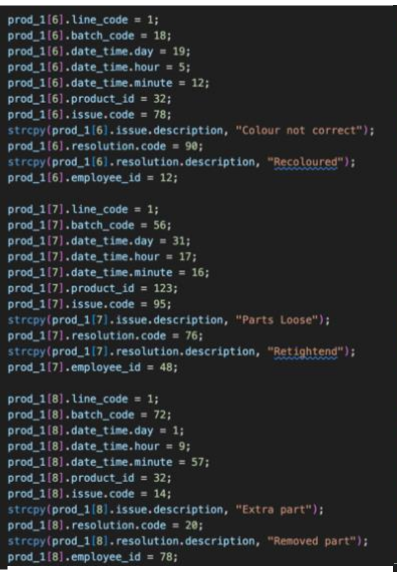
Production line 1



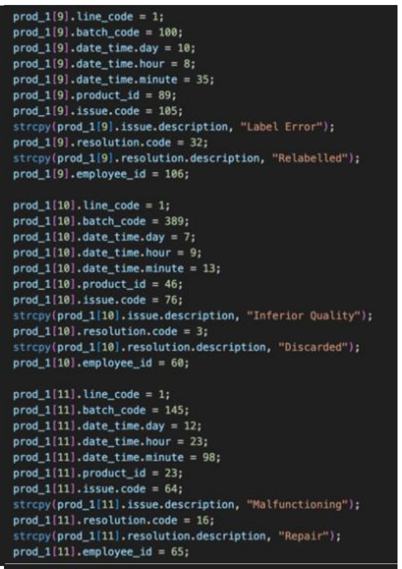
```c
//Production line 1
prod_1[0].line_code = 1;
prod_1[0].batch_code = 56;
prod_1[0].date_time.day = 13;
prod_1[0].date_time.hour = 16;
prod_1[0].date_time.minute = 36;
prod_1[0].product_id = 36;
prod_1[0].issue.code = 89;
strcpy(prod_1[0].issue.description, "Incomplete");
prod_1[0].resolution.code = 72;
strcpy(prod_1[0].resolution.description, "Finish by hand");
prod_1[0].employee_id = 90;

prod_1[1].line_code = 1;
prod_1[1].batch_code = 23;
prod_1[1].date_time.day = 16;
prod_1[1].date_time.hour = 14;
prod_1[1].date_time.minute = 59;
prod_1[1].product_id = 23;
prod_1[1].issue.code = 56;
strcpy(prod_1[1].issue.description, "Broken");
prod_1[1].resolution.code = 89;
strcpy(prod_1[1].resolution.description, "Recycle");
prod_1[1].employee_id = 90;

prod_1[2].line_code = 1;
prod_1[2].batch_code = 12;
prod_1[2].date_time.day = 16;
prod_1[2].date_time.hour = 12;
prod_1[2].date_time.minute = 48;
prod_1[2].product_id = 23;
prod_1[2].issue.code = 56;
strcpy(prod_1[2].issue.description, "Broken");
prod_1[2].resolution.code = 89;
strcpy(prod_1[2].resolution.description, "Recycle");
prod_1[2].employee_id = 76;
```



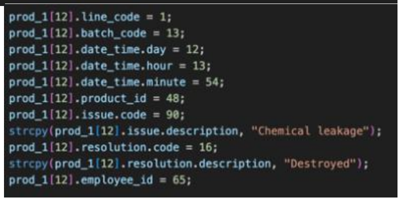
```c
prod_1[3].line_code = 1;
prod_1[3].batch_code = 45;
prod_1[3].date_time.day = 28;
prod_1[3].date_time.hour = 8;
prod_1[3].date_time.minute = 2;
prod_1[3].product_id = 36;
prod_1[3].issue.code = 89;
strcpy(prod_1[3].issue.description, "Incomplete");
prod_1[3].resolution.code = 72;
strcpy(prod_1[3].resolution.description, "Finish by hand");
prod_1[3].employee_id = 43;

prod_1[4].line_code = 1;
prod_1[4].batch_code = 12;
prod_1[4].date_time.day = 8;
prod_1[4].date_time.hour = 16;
prod_1[4].date_time.minute = 43;
prod_1[4].product_id = 23;
prod_1[4].issue.code = 12;
strcpy(prod_1[4].issue.description, "Damaged");
prod_1[4].resolution.code = 16;
strcpy(prod_1[4].resolution.description, "Repair");
prod_1[4].employee_id = 65;

prod_1[5].line_code = 1;
prod_1[5].batch_code = 34;
prod_1[5].date_time.day = 2;
prod_1[5].date_time.hour = 23;
prod_1[5].date_time.minute = 65;
prod_1[5].product_id = 67;
prod_1[5].issue.code = 10;
strcpy(prod_1[5].issue.description, "Packaging Damaged");
prod_1[5].resolution.code = 17;
strcpy(prod_1[5].resolution.description, "Repackaged");
prod_1[5].employee_id = 45;
```



```c
prod_1[6].line_code = 1;
prod_1[6].batch_code = 18;
prod_1[6].date_time.day = 19;
prod_1[6].date_time.hour = 5;
prod_1[6].date_time.minute = 12;
prod_1[6].product_id = 32;
prod_1[6].issue.code = 78;
strcpy(prod_1[6].issue.description, "Colour not correct");
prod_1[6].resolution.code = 90;
strcpy(prod_1[6].resolution.description, "Recoloured");
prod_1[6].employee_id = 12;

prod_1[7].line_code = 1;
prod_1[7].batch_code = 56;
prod_1[7].date_time.day = 31;
prod_1[7].date_time.hour = 17;
prod_1[7].date_time.minute = 16;
prod_1[7].product_id = 123;
prod_1[7].issue.code = 95;
strcpy(prod_1[7].issue.description, "Parts Loose");
prod_1[7].resolution.code = 76;
strcpy(prod_1[7].resolution.description, "Retightend");
prod_1[7].employee_id = 48;

prod_1[8].line_code = 1;
prod_1[8].batch_code = 72;
prod_1[8].date_time.day = 1;
prod_1[8].date_time.hour = 9;
prod_1[8].date_time.minute = 57;
prod_1[8].product_id = 32;
prod_1[8].issue.code = 14;
strcpy(prod_1[8].issue.description, "Extra part");
prod_1[8].resolution.code = 20;
strcpy(prod_1[8].resolution.description, "Removed part");
prod_1[8].employee_id = 78;
```

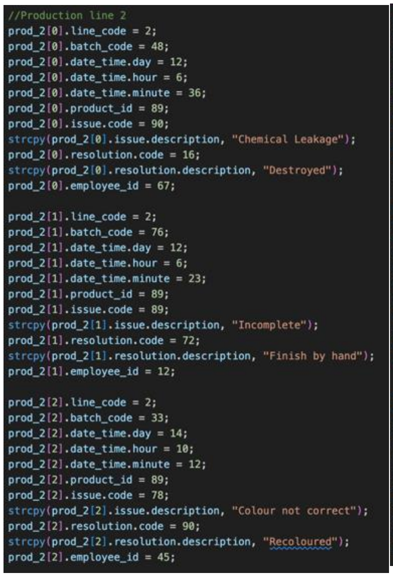


```c
prod_1[9].line_code = 1;
prod_1[9].batch_code = 100;
prod_1[9].date_time.day = 10;
prod_1[9].date_time.hour = 8;
prod_1[9].date_time.minute = 35;
prod_1[9].product_id = 89;
prod_1[9].issue.code = 105;
strcpy(prod_1[9].issue.description, "Label Error");
prod_1[9].resolution.code = 32;
strcpy(prod_1[9].resolution.description, "Relabelled");
prod_1[9].employee_id = 106;

prod_1[10].line_code = 1;
prod_1[10].batch_code = 389;
prod_1[10].date_time.day = 7;
prod_1[10].date_time.hour = 9;
prod_1[10].date_time.minute = 13;
prod_1[10].product_id = 46;
prod_1[10].issue.code = 76;
strcpy(prod_1[10].issue.description, "Inferior Quality");
prod_1[10].resolution.code = 3;
strcpy(prod_1[10].resolution.description, "Discarded");
prod_1[10].employee_id = 60;

prod_1[11].line_code = 1;
prod_1[11].batch_code = 145;
prod_1[11].date_time.day = 12;
prod_1[11].date_time.hour = 23;
prod_1[11].date_time.minute = 98;
prod_1[11].product_id = 23;
prod_1[11].issue.code = 64;
strcpy(prod_1[11].issue.description, "Malfunctioning");
prod_1[11].resolution.code = 16;
strcpy(prod_1[11].resolution.description, "Repair");
prod_1[11].employee_id = 65;
```

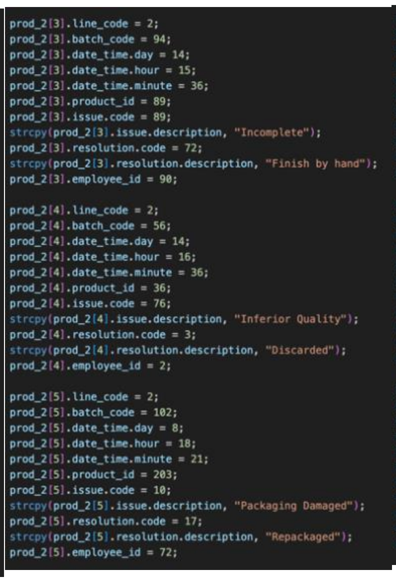


```c
prod_1[12].line_code = 1;
prod_1[12].batch_code = 13;
prod_1[12].date_time.day = 12;
prod_1[12].date_time.hour = 13;
prod_1[12].date_time.minute = 54;
prod_1[12].product_id = 48;
prod_1[12].issue.code = 90;
strcpy(prod_1[12].issue.description, "Chemical leakage");
prod_1[12].resolution.code = 16;
strcpy(prod_1[12].resolution.description, "Destroyed");
prod_1[12].employee_id = 65;
```

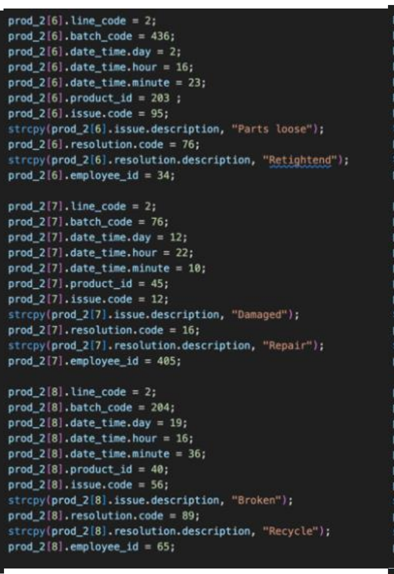
Production line 2



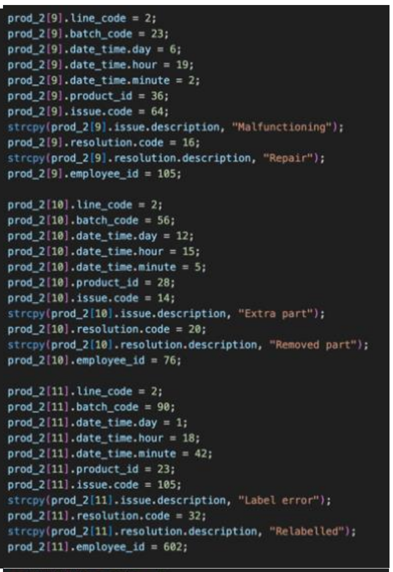
```c
//Production line 2
prod_2[0].line_code = 2;
prod_2[0].batch_code = 48;
prod_2[0].date_time.day = 12;
prod_2[0].date_time.hour = 6;
prod_2[0].date_time.minute = 36;
prod_2[0].product_id = 89;
prod_2[0].issue.code = 90;
strcpy(prod_2[0].issue.description, "Chemical Leakage");
prod_2[0].resolution.code = 16;
strcpy(prod_2[0].resolution.description, "Destroyed");
prod_2[0].employee_id = 67;

prod_2[1].line_code = 2;
prod_2[1].batch_code = 76;
prod_2[1].date_time.day = 12;
prod_2[1].date_time.hour = 6;
prod_2[1].date_time.minute = 23;
prod_2[1].product_id = 89;
prod_2[1].issue.code = 89;
strcpy(prod_2[1].issue.description, "Incomplete");
prod_2[1].resolution.code = 72;
strcpy(prod_2[1].resolution.description, "Finish by hand");
prod_2[1].employee_id = 12;

prod_2[2].line_code = 2;
prod_2[2].batch_code = 33;
prod_2[2].date_time.day = 14;
prod_2[2].date_time.hour = 10;
prod_2[2].date_time.minute = 12;
prod_2[2].product_id = 89;
prod_2[2].issue.code = 78;
strcpy(prod_2[2].issue.description, "Colour not correct");
prod_2[2].resolution.code = 90;
strcpy(prod_2[2].resolution.description, "Recoloured");
prod_2[2].employee_id = 45;
```



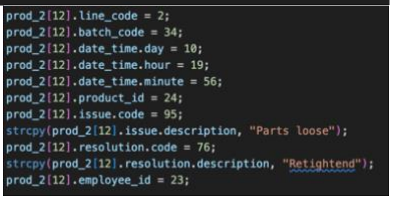
```c
prod_2[3].line_code = 2;
prod_2[3].batch_code = 94;
prod_2[3].date_time.day = 14;
prod_2[3].date_time.hour = 15;
prod_2[3].date_time.minute = 36;
prod_2[3].product_id = 89;
prod_2[3].issue.code = 89;
strcpy(prod_2[3].issue.description, "Incomplete");
prod_2[3].resolution.code = 72;
strcpy(prod_2[3].resolution.description, "Finish by hand");
prod_2[3].employee_id = 90;

prod_2[4].line_code = 2;
prod_2[4].batch_code = 56;
prod_2[4].date_time.day = 14;
prod_2[4].date_time.hour = 16;
prod_2[4].date_time.minute = 36;
prod_2[4].product_id = 36;
prod_2[4].issue.code = 76;
strcpy(prod_2[4].issue.description, "Inferior Quality");
prod_2[4].resolution.code = 3;
strcpy(prod_2[4].resolution.description, "Discarded");
prod_2[4].employee_id = 2;

prod_2[5].line_code = 2;
prod_2[5].batch_code = 102;
prod_2[5].date_time.day = 8;
prod_2[5].date_time.hour = 18;
prod_2[5].date_time.minute = 21;
prod_2[5].product_id = 203;
prod_2[5].issue.code = 10;
strcpy(prod_2[5].issue.description, "Packaging Damaged");
prod_2[5].resolution.code = 17;
strcpy(prod_2[5].resolution.description, "Repackaged");
prod_2[5].employee_id = 72;
```



```c
prod_2[6].line_code = 2;
prod_2[6].batch_code = 436;
prod_2[6].date_time.day = 2;
prod_2[6].date_time.hour = 16;
prod_2[6].date_time.minute = 23;
prod_2[6].product_id = 203 ;
prod_2[6].issue.code = 95;
strcpy(prod_2[6].issue.description, "Parts loose");
prod_2[6].resolution.code = 76;
strcpy(prod_2[6].resolution.description, "Retightend");
prod_2[6].employee_id = 34;

prod_2[7].line_code = 2;
prod_2[7].batch_code = 76;
prod_2[7].date_time.day = 12;
prod_2[7].date_time.hour = 22;
prod_2[7].date_time.minute = 10;
prod_2[7].product_id = 45;
prod_2[7].issue.code = 12;
strcpy(prod_2[7].issue.description, "Damaged");
prod_2[7].resolution.code = 16;
strcpy(prod_2[7].resolution.description, "Repair");
prod_2[7].employee_id = 405;

prod_2[8].line_code = 2;
prod_2[8].batch_code = 204;
prod_2[8].date_time.day = 19;
prod_2[8].date_time.hour = 16;
prod_2[8].date_time.minute = 36;
prod_2[8].product_id = 40;
prod_2[8].issue.code = 56;
strcpy(prod_2[8].issue.description, "Broken");
prod_2[8].resolution.code = 89;
strcpy(prod_2[8].resolution.description, "Recycle");
prod_2[8].employee_id = 65;
```

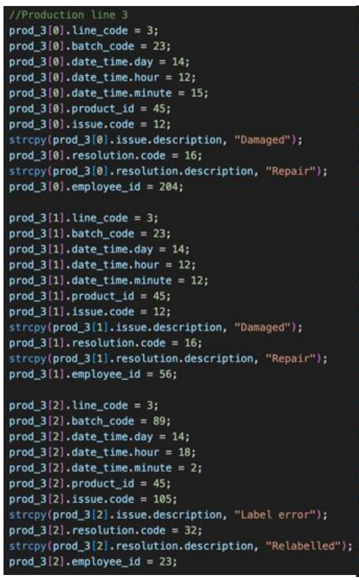


```c
prod_2[9].line_code = 2;
prod_2[9].batch_code = 23;
prod_2[9].date_time.day = 6;
prod_2[9].date_time.hour = 19;
prod_2[9].date_time.minute = 2;
prod_2[9].product_id = 36;
prod_2[9].issue.code = 64;
strcpy(prod_2[9].issue.description, "Malfunctioning");
prod_2[9].resolution.code = 16;
strcpy(prod_2[9].resolution.description, "Repair");
prod_2[9].employee_id = 105;

prod_2[10].line_code = 2;
prod_2[10].batch_code = 56;
prod_2[10].date_time.day = 12;
prod_2[10].date_time.hour = 15;
prod_2[10].date_time.minute = 5;
prod_2[10].product_id = 28;
prod_2[10].issue.code = 14;
strcpy(prod_2[10].issue.description, "Extra part");
prod_2[10].resolution.code = 20;
strcpy(prod_2[10].resolution.description, "Removed part");
prod_2[10].employee_id = 76;

prod_2[11].line_code = 2;
prod_2[11].batch_code = 90;
prod_2[11].date_time.day = 1;
prod_2[11].date_time.hour = 18;
prod_2[11].date_time.minute = 42;
prod_2[11].product_id = 23;
prod_2[11].issue.code = 105;
strcpy(prod_2[11].issue.description, "Label error");
prod_2[11].resolution.code = 32;
strcpy(prod_2[11].resolution.description, "Relabelled");
prod_2[11].employee_id = 602;
```

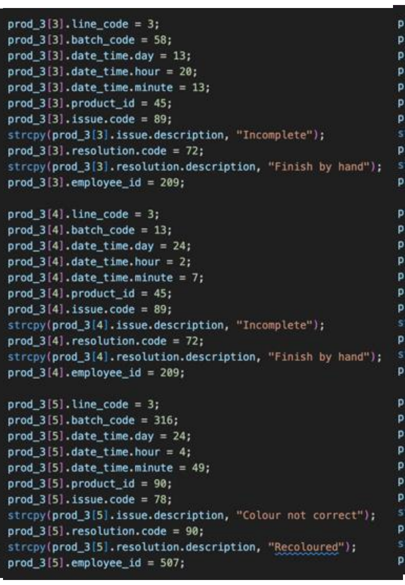


```c
prod_2[12].line_code = 2;
prod_2[12].batch_code = 34;
prod_2[12].date_time.day = 10;
prod_2[12].date_time.hour = 19;
prod_2[12].date_time.minute = 56;
prod_2[12].product_id = 24;
prod_2[12].issue.code = 95;
strcpy(prod_2[12].issue.description, "Parts loose");
prod_2[12].resolution.code = 76;
strcpy(prod_2[12].resolution.description, "Retightend");
prod_2[12].employee_id = 23;
```

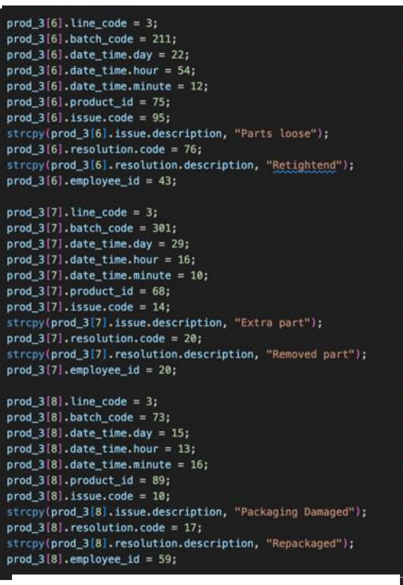
Production Line 3



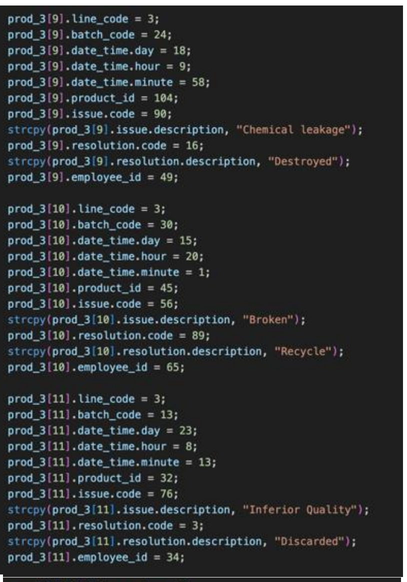
```c
//Production line 3
prod_3[0].line_code = 3;
prod_3[0].batch_code = 23;
prod_3[0].date_time.day = 14;
prod_3[0].date_time.hour = 12;
prod_3[0].date_time.minute = 15;
prod_3[0].product_id = 45;
prod_3[0].issue.code = 12;
strcpy(prod_3[0].issue.description, "Damaged");
prod_3[0].resolution.code = 16;
strcpy(prod_3[0].resolution.description, "Repair");
prod_3[0].employee_id = 204;

prod_3[1].line_code = 3;
prod_3[1].batch_code = 23;
prod_3[1].date_time.day = 14;
prod_3[1].date_time.hour = 12;
prod_3[1].date_time.minute = 12;
prod_3[1].product_id = 45;
prod_3[1].issue.code = 12;
strcpy(prod_3[1].issue.description, "Damaged");
prod_3[1].resolution.code = 16;
strcpy(prod_3[1].resolution.description, "Repair");
prod_3[1].employee_id = 56;

prod_3[2].line_code = 3;
prod_3[2].batch_code = 89;
prod_3[2].date_time.day = 14;
prod_3[2].date_time.hour = 18;
prod_3[2].date_time.minute = 2;
prod_3[2].product_id = 45;
prod_3[2].issue.code = 105;
strcpy(prod_3[2].issue.description, "Label error");
prod_3[2].resolution.code = 32;
strcpy(prod_3[2].resolution.description, "Relabelled");
prod_3[2].employee_id = 23;
```



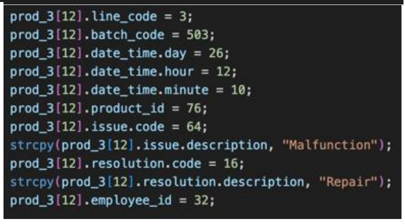
```c
prod_3[3].line_code = 3;
prod_3[3].batch_code = 58;
prod_3[3].date_time.day = 13;
prod_3[3].date_time.hour = 20;
prod_3[3].date_time.minute = 13;
prod_3[3].product_id = 45;
prod_3[3].issue.code = 89;
strcpy(prod_3[3].issue.description, "Incomplete");
prod_3[3].resolution.code = 72;
strcpy(prod_3[3].resolution.description, "Finish by hand");
prod_3[3].employee_id = 209;

prod_3[4].line_code = 3;
prod_3[4].batch_code = 13;
prod_3[4].date_time.day = 24;
prod_3[4].date_time.hour = 2;
prod_3[4].date_time.minute = 7;
prod_3[4].product_id = 45;
prod_3[4].issue.code = 89;
strcpy(prod_3[4].issue.description, "Incomplete");
prod_3[4].resolution.code = 72;
strcpy(prod_3[4].resolution.description, "Finish by hand");
prod_3[4].employee_id = 209;

prod_3[5].line_code = 3;
prod_3[5].batch_code = 316;
prod_3[5].date_time.day = 24;
prod_3[5].date_time.hour = 4;
prod_3[5].date_time.minute = 49;
prod_3[5].product_id = 90;
prod_3[5].issue.code = 78;
strcpy(prod_3[5].issue.description, "Colour not correct");
prod_3[5].resolution.code = 90;
strcpy(prod_3[5].resolution.description, "Recoloured");
prod_3[5].employee_id = 507;
```



```c
prod_3[6].line_code = 3;
prod_3[6].batch_code = 211;
prod_3[6].date_time.day = 22;
prod_3[6].date_time.hour = 54;
prod_3[6].date_time.minute = 12;
prod_3[6].product_id = 75;
prod_3[6].issue.code = 95;
strcpy(prod_3[6].issue.description, "Parts loose");
prod_3[6].resolution.code = 76;
strcpy(prod_3[6].resolution.description, "Retightend");
prod_3[6].employee_id = 43;

prod_3[7].line_code = 3;
prod_3[7].batch_code = 301;
prod_3[7].date_time.day = 29;
prod_3[7].date_time.hour = 16;
prod_3[7].date_time.minute = 10;
prod_3[7].product_id = 68;
prod_3[7].issue.code = 14;
strcpy(prod_3[7].issue.description, "Extra part");
prod_3[7].resolution.code = 20;
strcpy(prod_3[7].resolution.description, "Removed part");
prod_3[7].employee_id = 20;

prod_3[8].line_code = 3;
prod_3[8].batch_code = 73;
prod_3[8].date_time.day = 15;
prod_3[8].date_time.hour = 13;
prod_3[8].date_time.minute = 16;
prod_3[8].product_id = 89;
prod_3[8].issue.code = 10;
strcpy(prod_3[8].issue.description, "Packaging Damaged");
prod_3[8].resolution.code = 17;
strcpy(prod_3[8].resolution.description, "Repackaged");
prod_3[8].employee_id = 59;
```

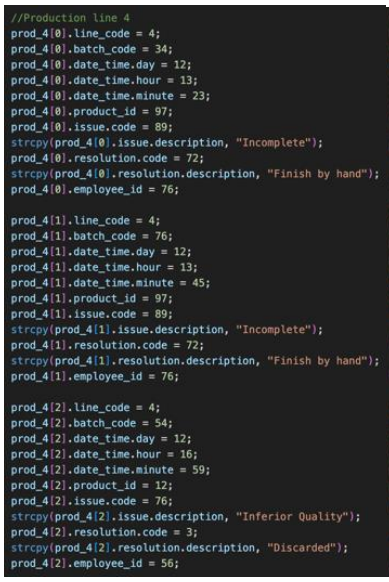


```c
prod_3[9].line_code = 3;
prod_3[9].batch_code = 24;
prod_3[9].date_time.day = 18;
prod_3[9].date_time.hour = 9;
prod_3[9].date_time.minute = 58;
prod_3[9].product_id = 104;
prod_3[9].issue.code = 90;
strcpy(prod_3[9].issue.description, "Chemical leakage");
prod_3[9].resolution.code = 16;
strcpy(prod_3[9].resolution.description, "Destroyed");
prod_3[9].employee_id = 49;

prod_3[10].line_code = 3;
prod_3[10].batch_code = 30;
prod_3[10].date_time.day = 15;
prod_3[10].date_time.hour = 20;
prod_3[10].date_time.minute = 1;
prod_3[10].product_id = 45;
prod_3[10].issue.code = 56;
strcpy(prod_3[10].issue.description, "Broken");
prod_3[10].resolution.code = 89;
strcpy(prod_3[10].resolution.description, "Recycle");
prod_3[10].employee_id = 65;

prod_3[11].line_code = 3;
prod_3[11].batch_code = 13;
prod_3[11].date_time.day = 23;
prod_3[11].date_time.hour = 8;
prod_3[11].date_time.minute = 13;
prod_3[11].product_id = 32;
prod_3[11].issue.code = 76;
strcpy(prod_3[11].issue.description, "Inferior Quality");
prod_3[11].resolution.code = 3;
strcpy(prod_3[11].resolution.description, "Discarded");
prod_3[11].employee_id = 34;
```

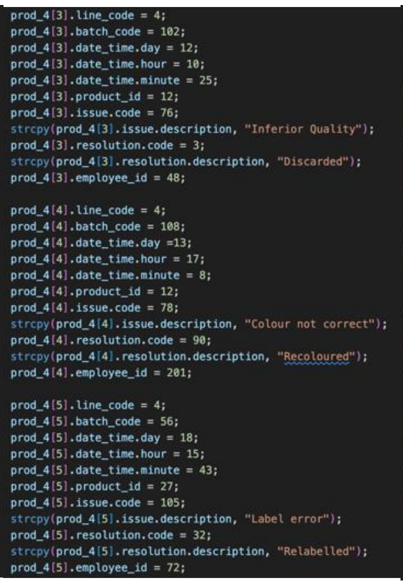


```c
prod_3[12].line_code = 3;
prod_3[12].batch_code = 503;
prod_3[12].date_time.day = 26;
prod_3[12].date_time.hour = 12;
prod_3[12].date_time.minute = 10;
prod_3[12].product_id = 76;
prod_3[12].issue.code = 64;
strcpy(prod_3[12].issue.description, "Malfunction");
prod_3[12].resolution.code = 16;
strcpy(prod_3[12].resolution.description, "Repair");
prod_3[12].employee_id = 32;
```

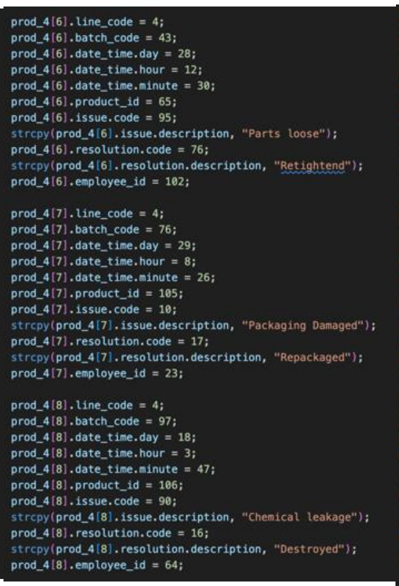
Production Line 4



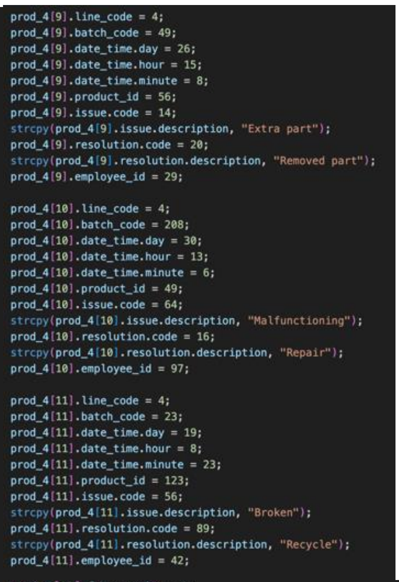
```c
//Production line 4
prod_4[0].line_code = 4;
prod_4[0].batch_code = 34;
prod_4[0].date_time.day = 12;
prod_4[0].date_time.hour = 13;
prod_4[0].date_time.minute = 23;
prod_4[0].product_id = 97;
prod_4[0].issue.code = 89;
strcpy(prod_4[0].issue.description, "Incomplete");
prod_4[0].resolution.code = 72;
strcpy(prod_4[0].resolution.description, "Finish by hand");
prod_4[0].employee_id = 76;

prod_4[1].line_code = 4;
prod_4[1].batch_code = 76;
prod_4[1].date_time.day = 12;
prod_4[1].date_time.hour = 13;
prod_4[1].date_time.minute = 45;
prod_4[1].product_id = 97;
prod_4[1].issue.code = 89;
strcpy(prod_4[1].issue.description, "Incomplete");
prod_4[1].resolution.code = 72;
strcpy(prod_4[1].resolution.description, "Finish by hand");
prod_4[1].employee_id = 76;

prod_4[2].line_code = 4;
prod_4[2].batch_code = 54;
prod_4[2].date_time.day = 12;
prod_4[2].date_time.hour = 16;
prod_4[2].date_time.minute = 59;
prod_4[2].product_id = 12;
prod_4[2].issue.code = 76;
strcpy(prod_4[2].issue.description, "Inferior Quality");
prod_4[2].resolution.code = 3;
strcpy(prod_4[2].resolution.description, "Discarded");
prod_4[2].employee_id = 56;
```



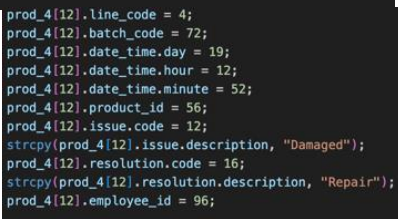
```c
prod_4[3].line_code = 4;
prod_4[3].batch_code = 102;
prod_4[3].date_time.day = 12;
prod_4[3].date_time.hour = 10;
prod_4[3].date_time.minute = 25;
prod_4[3].product_id = 12;
prod_4[3].issue.code = 76;
strcpy(prod_4[3].issue.description, "Inferior Quality");
prod_4[3].resolution.code = 3;
strcpy(prod_4[3].resolution.description, "Discarded");
prod_4[3].employee_id = 48;

prod_4[4].line_code = 4;
prod_4[4].batch_code = 108;
prod_4[4].date_time.day =13;
prod_4[4].date_time.hour = 17;
prod_4[4].date_time.minute = 8;
prod_4[4].product_id = 12;
prod_4[4].issue.code = 78;
strcpy(prod_4[4].issue.description, "Colour not correct");
prod_4[4].resolution.code = 90;
strcpy(prod_4[4].resolution.description, "Recoloured");
prod_4[4].employee_id = 201;

prod_4[5].line_code = 4;
prod_4[5].batch_code = 56;
prod_4[5].date_time.day = 18;
prod_4[5].date_time.hour = 15;
prod_4[5].date_time.minute = 43;
prod_4[5].product_id = 27;
prod_4[5].issue.code = 105;
strcpy(prod_4[5].issue.description, "Label error");
prod_4[5].resolution.code = 32;
strcpy(prod_4[5].resolution.description, "Relabelled");
prod_4[5].employee_id = 72;
```



```c
prod_4[6].line_code = 4;
prod_4[6].batch_code = 43;
prod_4[6].date_time.day = 28;
prod_4[6].date_time.hour = 12;
prod_4[6].date_time.minute = 30;
prod_4[6].product_id = 65;
prod_4[6].issue.code = 95;
strcpy(prod_4[6].issue.description, "Parts loose");
prod_4[6].resolution.code = 76;
strcpy(prod_4[6].resolution.description, "Retightend");
prod_4[6].employee_id = 102;

prod_4[7].line_code = 4;
prod_4[7].batch_code = 76;
prod_4[7].date_time.day = 29;
prod_4[7].date_time.hour = 8;
prod_4[7].date_time.minute = 26;
prod_4[7].product_id = 105;
prod_4[7].issue.code = 10;
strcpy(prod_4[7].issue.description, "Packaging Damaged");
prod_4[7].resolution.code = 17;
strcpy(prod_4[7].resolution.description, "Repackaged");
prod_4[7].employee_id = 23;

prod_4[8].line_code = 4;
prod_4[8].batch_code = 97;
prod_4[8].date_time.day = 18;
prod_4[8].date_time.hour = 3;
prod_4[8].date_time.minute = 47;
prod_4[8].product_id = 106;
prod_4[8].issue.code = 90;
strcpy(prod_4[8].issue.description, "Chemical leakage");
prod_4[8].resolution.code = 16;
strcpy(prod_4[8].resolution.description, "Destroyed");
prod_4[8].employee_id = 64;
```



```c
prod_4[9].line_code = 4;
prod_4[9].batch_code = 49;
prod_4[9].date_time.day = 26;
prod_4[9].date_time.hour = 15;
prod_4[9].date_time.minute = 8;
prod_4[9].product_id = 56;
prod_4[9].issue.code = 14;
strcpy(prod_4[9].issue.description, "Extra part");
prod_4[9].resolution.code = 20;
strcpy(prod_4[9].resolution.description, "Removed part");
prod_4[9].employee_id = 29;

prod_4[10].line_code = 4;
prod_4[10].batch_code = 208;
prod_4[10].date_time.day = 30;
prod_4[10].date_time.hour = 13;
prod_4[10].date_time.minute = 6;
prod_4[10].product_id = 49;
prod_4[10].issue.code = 64;
strcpy(prod_4[10].issue.description, "Malfunctioning");
prod_4[10].resolution.code = 16;
strcpy(prod_4[10].resolution.description, "Repair");
prod_4[10].employee_id = 97;

prod_4[11].line_code = 4;
prod_4[11].batch_code = 23;
prod_4[11].date_time.day = 19;
prod_4[11].date_time.hour = 8;
prod_4[11].date_time.minute = 23;
prod_4[11].product_id = 123;
prod_4[11].issue.code = 56;
strcpy(prod_4[11].issue.description, "Broken");
prod_4[11].resolution.code = 89;
strcpy(prod_4[11].resolution.description, "Recycle");
prod_4[11].employee_id = 42;
```



```c
prod_4[12].line_code = 4;
prod_4[12].batch_code = 72;
prod_4[12].date_time.day = 19;
prod_4[12].date_time.hour = 12;
prod_4[12].date_time.minute = 52;
prod_4[12].product_id = 56;
prod_4[12].issue.code = 12;
strcpy(prod_4[12].issue.description, "Damaged");
prod_4[12].resolution.code = 16;
strcpy(prod_4[12].resolution.description, "Repair");
prod_4[12].employee_id = 96;
```

Task 1

Design:

My design for this task is using the merge sort function to sort each production line's data in product id, issue code and finally date & time order. This is done by running the merge sort function 4 times for each production line. Each time it is run it takes in a different production line's array and sorts that production line's data. It works by firstly splitting the array in half until it is only left with one structure in the array and returns the product id of that structure. It will then go back to the old function and continue until there is again only one structure left and return the product id. Once two product id's are returned, it will run the merge function. The merge function combines them together and puts them into the correct order of product id, issue code and date and time order. It will keep doing this until it is left with two sorted halves of the array and it will do its final merge. The merge function will take each half and start with the first two structures in each half of the array. It will then put the correct structure into the temporary array of structures (temp) depending on which structures' product id, issue code and date and time is less then, equal to or greater than the other. It will keep running until one half of the array becomes empty and will put the rest of the other half of the array into the temp. Once it has sorted all the numbers, the final for loop puts them all back into the original array of structures in the correct order. Once this is done the algorithm will stop, it will return back to main, the original array of structures for the production line that was passed in is outputted and the process repeats itself for the next production line.

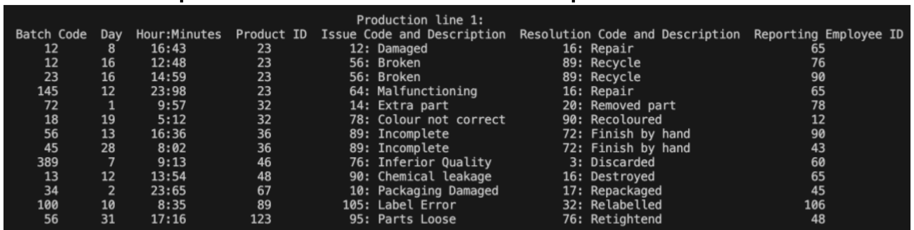
Running time:

The running time of the algorithm is O(nlog(n)) which meets the running time requirements of O(nlog(n)) or better. This is because mid is being divided by 2 every time the merge sort function is run and the for loop runs n times which gives you O(nlog(n)).

Test Plan:

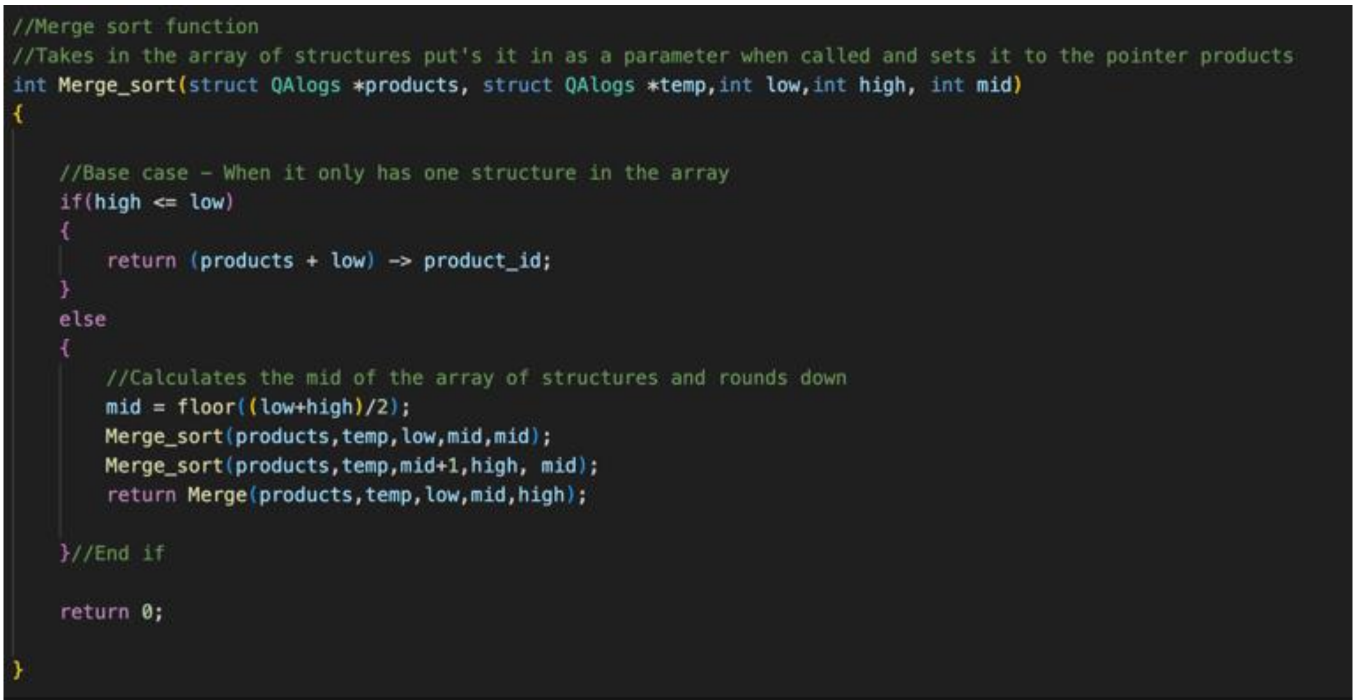
My test plan for this task was quite simple. As my test data contains product ids, issue codes and dates and time that are the same between some of the structures, I was able to see that the design still works and puts the data in the correct order which allowed me to see that the design does check if the other requirement's are met and put them in the order of product id, issue code and finally date and time order.

Output from the test data of production line 1

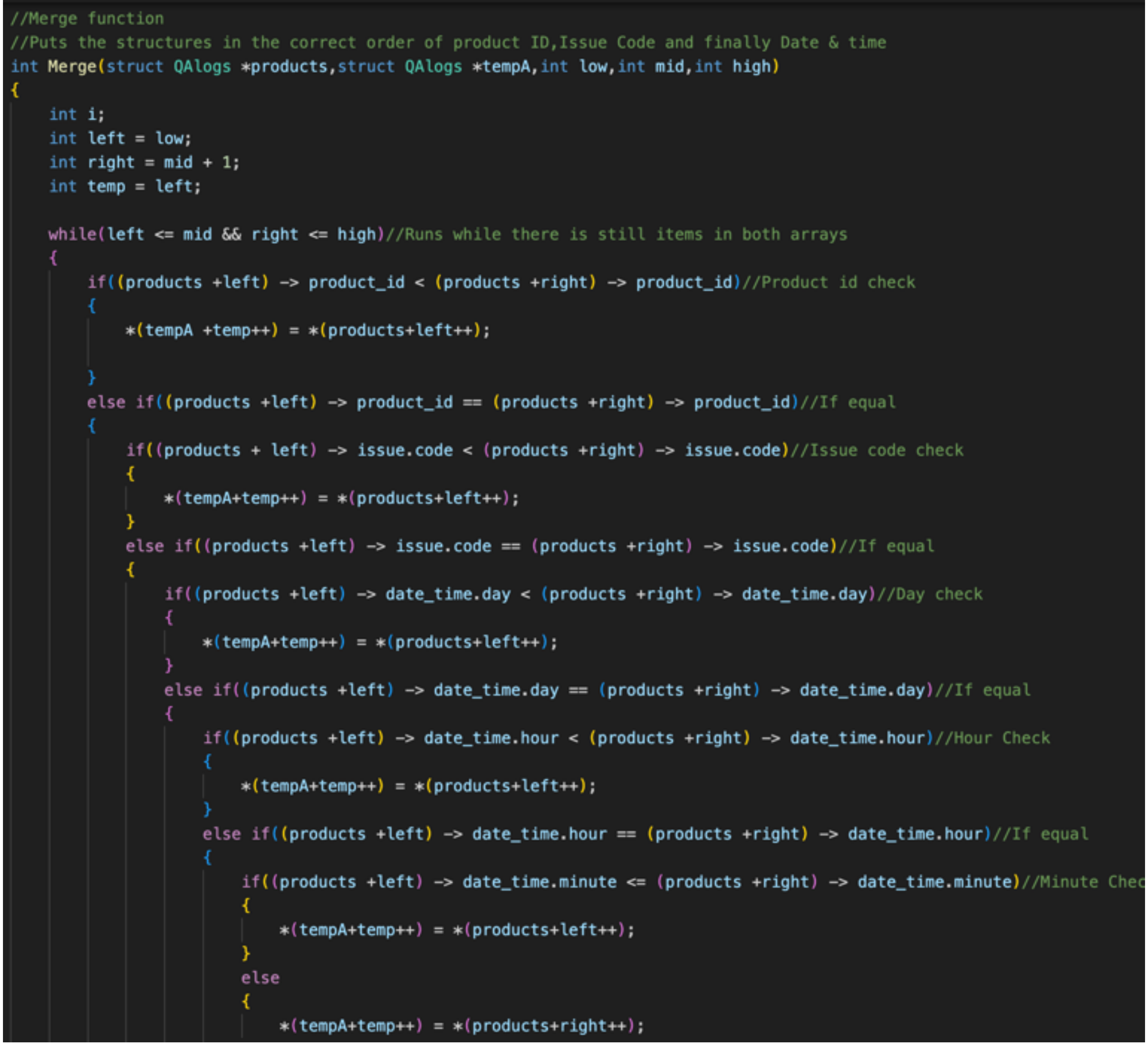


```
                                        Production line 1:
 Batch Code  Day  Hour:Minutes  Product ID  Issue Code and Description  Resolution Code and Description  Reporting Employee ID
     12        8     16:43          23            12: Damaged                   16: Repair                          65
     12       16     12:48          23            56: Broken                    89: Recycle                         76
     23       16     14:59          23            56: Broken                    89: Recycle                         90
    145       12     23:98          23            64: Malfunctioning            16: Repair                          65
     72        1      9:57          32            14: Extra part                20: Removed part                    78
     18       19      5:12          32            78: Colour not correct        90: Recoloured                      12
     56       13     16:36          36            89: Incomplete                72: Finish by hand                  90
     45       28      8:02          36            89: Incomplete                72: Finish by hand                  43
    389        7      9:13          46            76: Inferior Quality           3: Discarded                       60
     13       12     13:54          48            90: Chemical leakage          16: Destroyed                       65
     34        2     23:65          67            10: Packaging Damaged         17: Repackaged                      45
    100       10      8:35          89           105: Label Error               32: Relabelled                     106
     56       31     17:16         123            95: Parts Loose               76: Retightend                      48
```

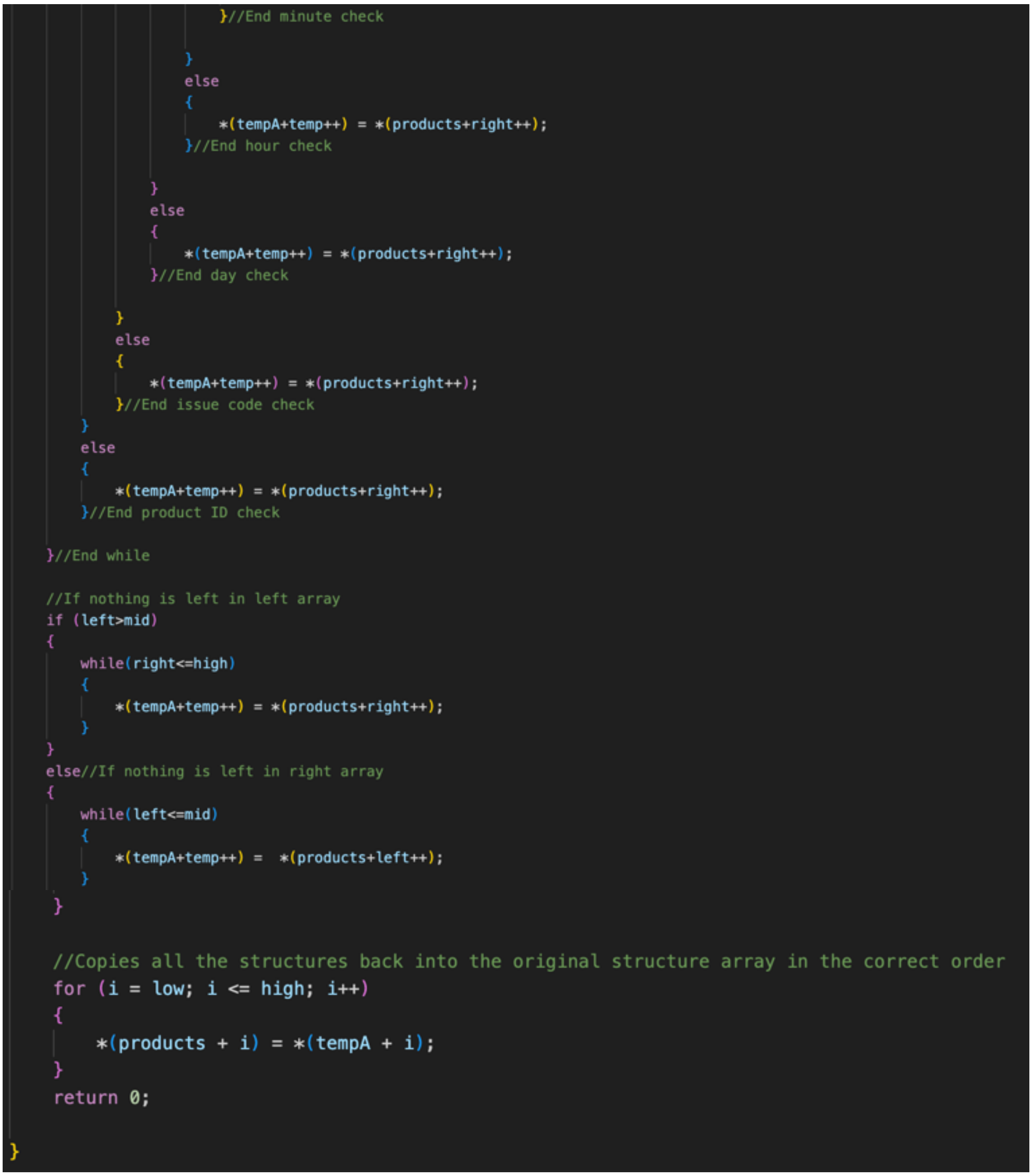
Working C Code



```c
//Merge sort function
//Takes in the array of structures put's it in as a parameter when called and sets it to the pointer products
int Merge_sort(struct QAlogs *products, struct QAlogs *temp,int low,int high, int mid)
{

    //Base case - When it only has one structure in the array
    if(high <= low)
    {
        return (products + low) -> product_id;
    }
    else
    {
        //Calculates the mid of the array of structures and rounds down
        mid = floor((low+high)/2);
        Merge_sort(products,temp,low,mid,mid);
        Merge_sort(products,temp,mid+1,high, mid);
        return Merge(products,temp,low,mid,high);

    }//End if

    return 0;

}
```



```c
//Merge function
//Puts the structures in the correct order of product ID,Issue Code and finally Date & time
int Merge(struct QAlogs *products,struct QAlogs *tempA,int low,int mid,int high)
{
    int i;
    int left = low;
    int right = mid + 1;
    int temp = left;

    while(left <= mid && right <= high)//Runs while there is still items in both arrays
    {
        if((products +left) -> product_id < (products +right) -> product_id)//Product id check
        {
            *(tempA +temp++) = *(products+left++);

        }
        else if((products +left) -> product_id == (products +right) -> product_id)//If equal
        {
            if((products + left) -> issue.code < (products +right) -> issue.code)//Issue code check
            {
                *(tempA+temp++) = *(products+left++);
            }
            else if((products +left) -> issue.code == (products +right) -> issue.code)//If equal
            {
                if((products +left) -> date_time.day < (products +right) -> date_time.day)//Day check
                {
                    *(tempA+temp++) = *(products+left++);
                }
                else if((products +left) -> date_time.day == (products +right) -> date_time.day)//If equal
                {
                    if((products +left) -> date_time.hour < (products +right) -> date_time.hour)//Hour Check
                    {
                        *(tempA+temp++) = *(products+left++);
                    }
                    else if((products +left) -> date_time.hour == (products +right) -> date_time.hour)//If equal
                    {
                        if((products +left) -> date_time.minute <= (products +right) -> date_time.minute)//Minute Chec
                        {
                            *(tempA+temp++) = *(products+left++);
                        }
                        else
                        {
                            *(tempA+temp++) = *(products+right++);
```



```
                        }//End minute check

                    }
                    else
                    {
                        *(tempA+temp++) = *(products+right++);
                    }//End hour check

                }
                else
                {
                    *(tempA+temp++) = *(products+right++);
                }//End day check

            }
            else
            {
                *(tempA+temp++) = *(products+right++);
            }//End issue code check
        }
        else
        {
            *(tempA+temp++) = *(products+right++);
        }//End product ID check

    }//End while

    //If nothing is left in left array
    if (left>mid)
    {
        while(right<=high)
        {
            *(tempA+temp++) = *(products+right++);
        }
    }
    else//If nothing is left in right array
    {
        while(left<=mid)
        {
            *(tempA+temp++) =  *(products+left++);
        }
    }

    //Copies all the structures back into the original structure array in the correct order
    for (i = low; i <= high; i++)
    {
        *(products + i) = *(tempA + i);
    }
    return 0;

}
```

Pseudocode

```
Merge_sort(products,low,high)
if high <= low
   return
else
   mid = (low+high)/2
   Merge_sort(products,low,mid)
   Merge_sort(products,mid+1 ,high)
   Merge(products,low,mid,high)

Merge(products,low,mid,high)
left = low
right = mid + 1
temp = left
while left <= mid AND right <= high
   if products[left].product_id < products[right].product_id
      tempA[temp++] = products[left++]
   else if products[left].product_id == products[right].product_id
      if products[left].issue.code < products[right].issue.code
         tempA[temp++] = products[left++]
      else if products[left].issue.code == products[right].issue.code
         if products[left].date_time.day < products[right].date_time.day
            tempA[temp++] = products[left++]
         else if products[left].date_time.day == products[right].date_time.day
            if products[left].date_time.hour < products[right].date_time.hour
               tempA[temp++] = products[left++]
            else if products[left].date_time.hour == products[right].date_time.hour
               if products[left].date_time.minute <= products[right].date_time.minute
                  tempA[temp++] = products[left++]
               else
                  tempA[temp++] = products[right++]
            else
               tempA[temp++] = products[right++]
         else
            tempA[temp++] = products[right++]
      else
         tempA[temp++] = products[right++]
   else
      tempA[temp++] = products[right++]
Endwhile
if left > mid
   while right <= high
      tempA[temp++] = products[right++]
else
   while left <= mid
      tempA[temp+=] = products[left++]

for i = low to i <= high
   products[i] = tempA[i]
Endfor
```

Task 2

Design

My design for task 2 is very similar to task 1. It again uses the merge sort algorithm. I created an array of structures that can contain all the test data called all_prod. I then put all the data inside this array and passed the array into the merge sort function. The merge sort function will then run the same way I have outlined in task 1 but the merge function will be slightly different as it will put the correct structure into the temp array of structures depending on which structures' product id and line id is less then, equal to or greater than. Like task 1, once it has sorted all the numbers, the final for loop puts them all back into the original array of structures in the correct order. Once this is done the algorithm will stop, it will return back to main and the original array of structures is outputted.

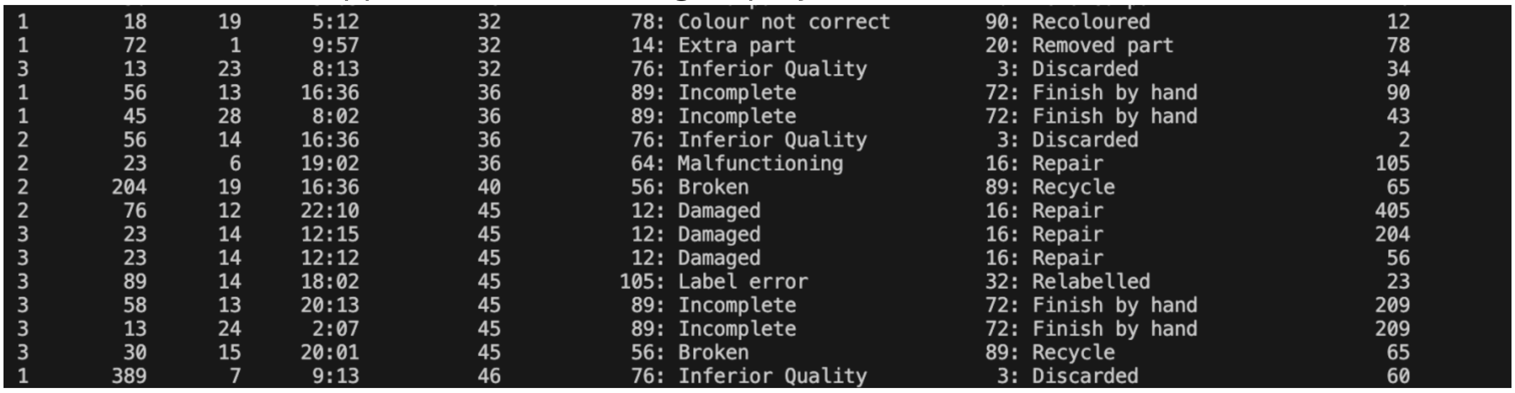
Running Time

The running time of this algorithm is O(nlog(n)) which meets the running time requirements of O(N) or better. This is because mid is being divided by 2 every time the merge sort function is run and the for loop runs n times which gives you O(nlog(n)) .

Test Plan

My test plan for this task was the same as task 1. The test was to check if some of the data was the same would it still be sorted in the correct order. This was done the same way as task 1 by having some of the test data contain the same product id's.

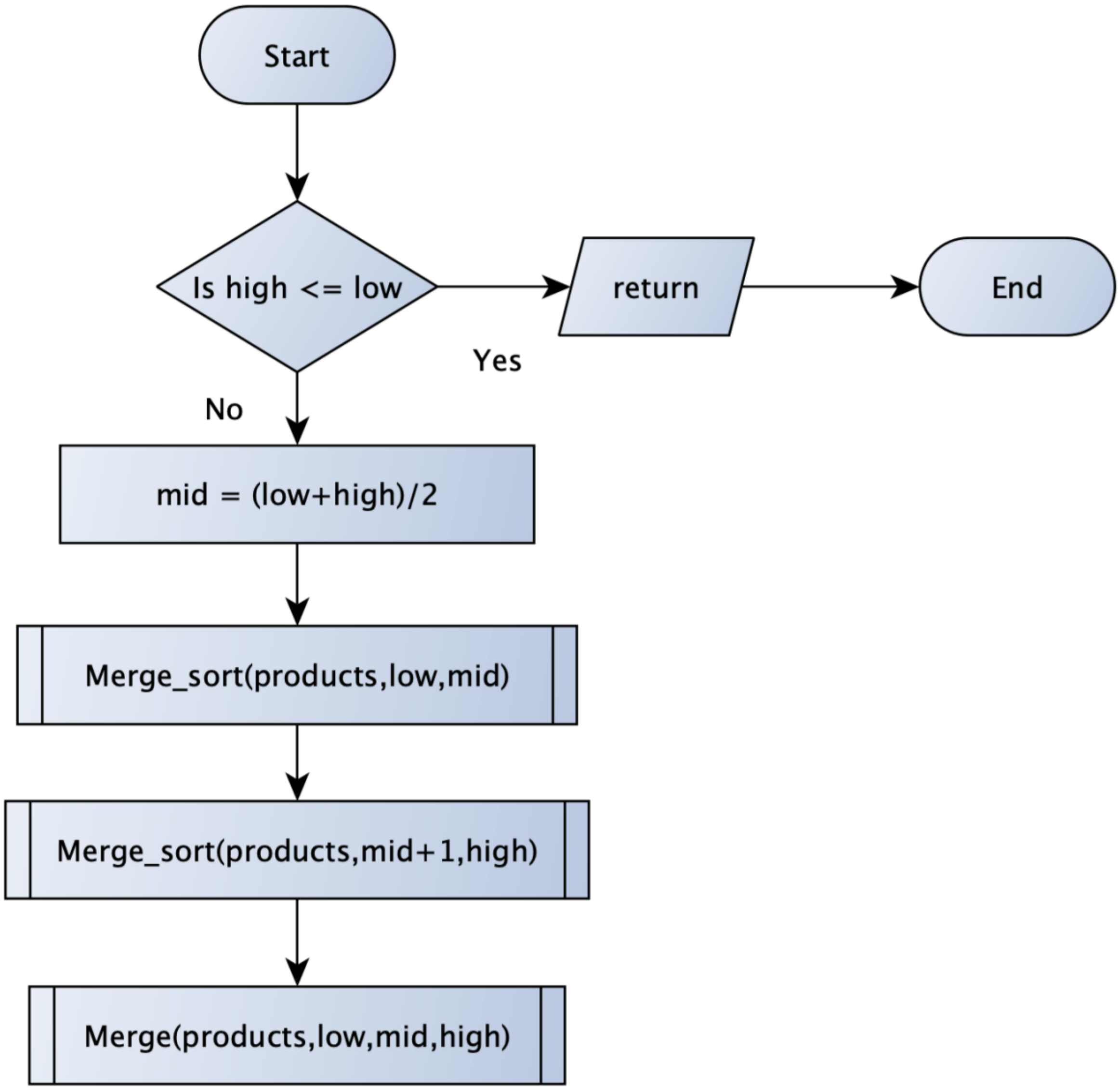
This is a snippet of the data being displayed after the test



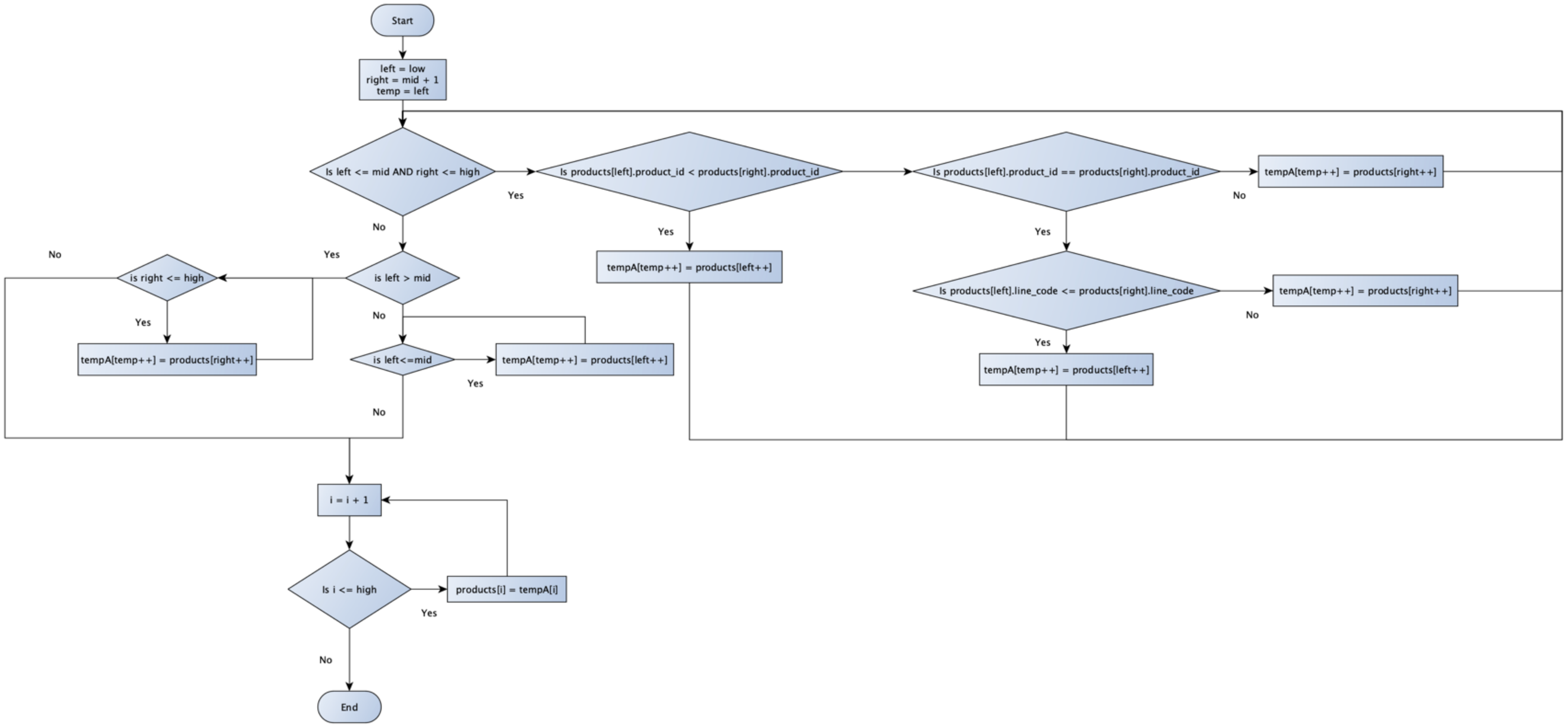
```
1      18    19    5:12      32        78: Colour not correct      90: Recoloured          12
1      72     1    9:57      32        14: Extra part              20: Removed part        78
3      13    23    8:13      32        76: Inferior Quality         3: Discarded           34
1      56    13   16:36      36        89: Incomplete              72: Finish by hand      90
1      45    28    8:02      36        89: Incomplete              72: Finish by hand      43
2      56    14   16:36      36        76: Inferior Quality         3: Discarded            2
2      23     6   19:02      36        64: Malfunctioning          16: Repair             105
2     204    19   16:36      40        56: Broken                  89: Recycle             65
2      76    12   22:10      45        12: Damaged                 16: Repair             405
3      23    14   12:15      45        12: Damaged                 16: Repair             204
3      23    14   12:12      45        12: Damaged                 16: Repair              56
3      89    14   18:02      45       105: Label error             32: Relabelled          23
3      58    13   20:13      45        89: Incomplete              72: Finish by hand     209
3      13    24    2:07      45        89: Incomplete              72: Finish by hand     209
3      30    15   20:01      45        56: Broken                  89: Recycle             65
1     389     7    9:13      46        76: Inferior Quality         3: Discarded           60
```

Flowchart

Merge_sort(products,low,high)


Start
Is high <= low
Yes
No
return
End
mid = (low+high)/2
Merge_sort(products,low,mid)
Merge_sort(products,mid+1,high)
Merge(products,low,mid,high)

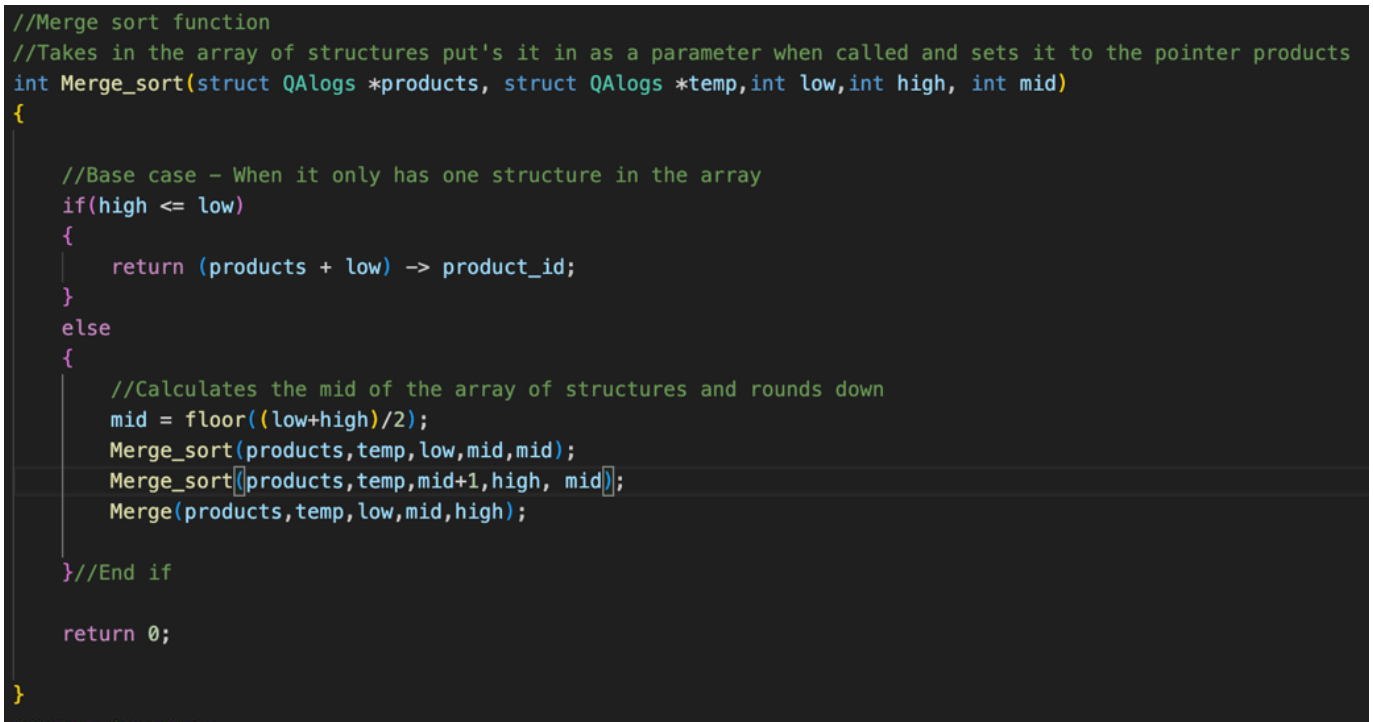
Merge(products,low,mid,high)


Start
left = low
right = mid + 1
temp = left
Is left <= mid AND right <= high
Yes
No
Is products[left].product_id < products[right].product_id
Yes
tempA[temp++] = products[left++]
Is products[left].product_id == products[right].product_id
No
tempA[temp++] = products[right++]
Yes
Is products[left].line_code <= products[right].line_code
No
tempA[temp++] = products[right++]
Yes
tempA[temp++] = products[left++]
is left > mid
Yes
is right <= high
No
Yes
tempA[temp++] = products[right++]
No
is left<=mid
Yes
tempA[temp++] = products[left++]
No
i = i + 1
Is i <= high
Yes
products[i] = tempA[i]
No
End

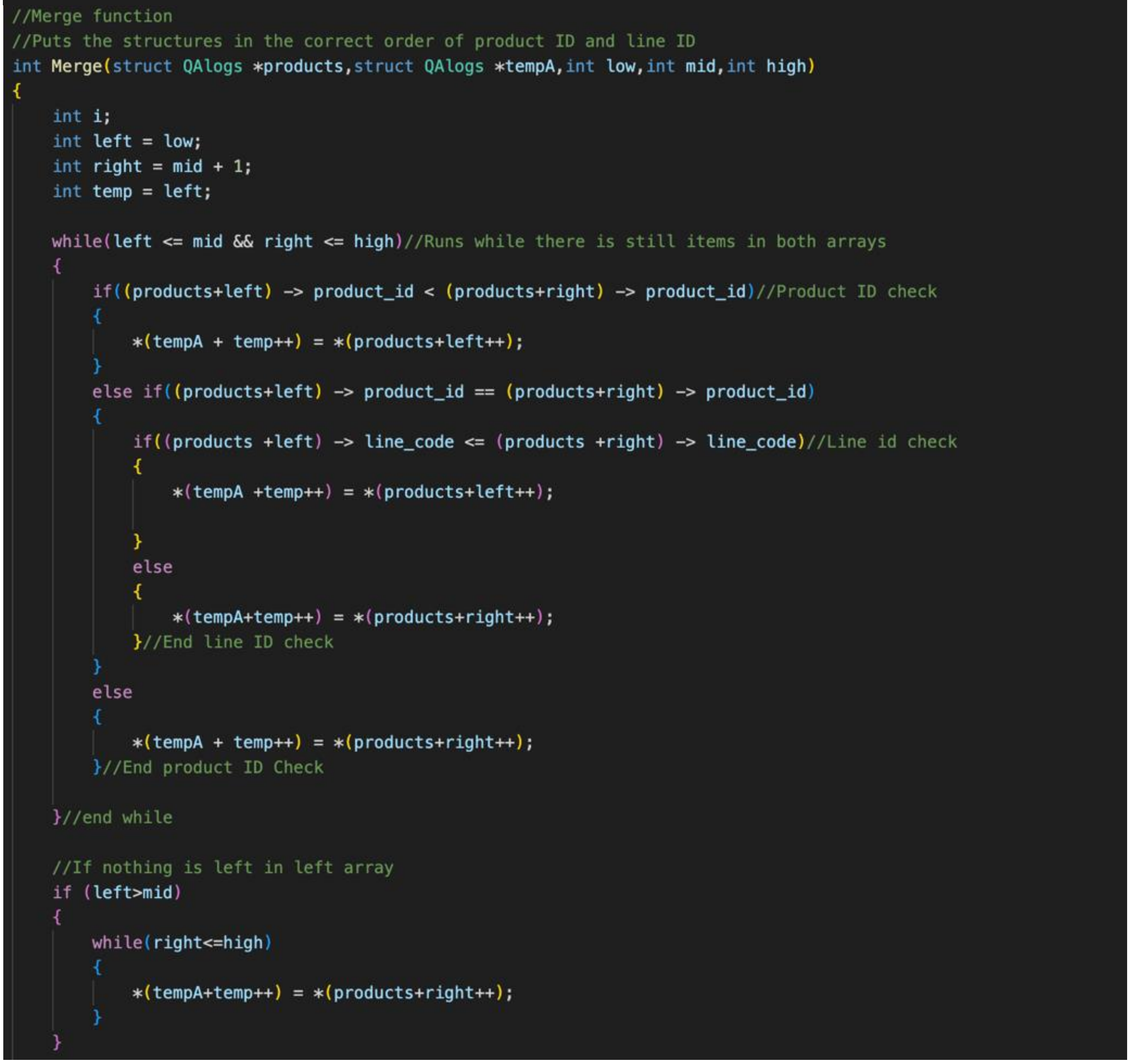
Pseudocode

```
Merge_sort(products,low,high)
if high <= low
    return
else
    mid = (low+high)/2
    Merge_sort(products,low,mid)
    Merge_sort(products,mid+1 ,high)
    Merge(products,low,mid,high)

Merge(products,low,mid,high)
left = low
right = mid + 1
temp = left
while left <= mid AND right <= high
    if products[left].product_id < products[right].product_id
      tempA[temp++] = products[left++]
   else if products[left].product_id == products[right].product_id
      if products[left].line_code <= products[right].line_code
         tempA[temp++] = products[left++]
      else
         tempA[temp++] = products[right++]
   else
      tempA[temp++] = products[right++]
Endwhile
if left > mid
   while right <= high
      tempA[temp++] = products[right++]
else
   while left <= mid
      tempA[temp+=] = products[left++]

for i = low to i <= high do
   products[i] = tempA[i]
Endfor
```

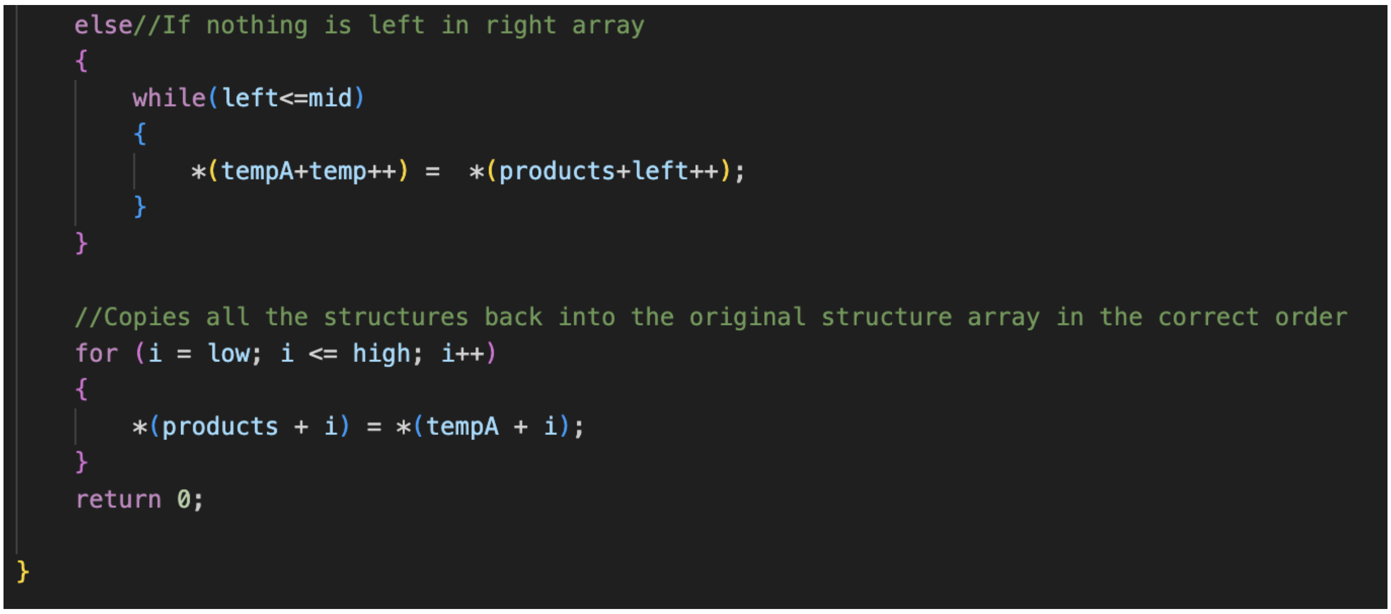
Working C Code



```c
//Merge sort function
//Takes in the array of structures put's it in as a parameter when called and sets it to the pointer products
int Merge_sort(struct QAlogs *products, struct QAlogs *temp,int low,int high, int mid)
{

    //Base case - When it only has one structure in the array
    if(high <= low)
    {
        return (products + low) -> product_id;
    }
    else
    {
        //Calculates the mid of the array of structures and rounds down
        mid = floor((low+high)/2);
        Merge_sort(products,temp,low,mid,mid);
        Merge_sort(products,temp,mid+1,high, mid);
        Merge(products,temp,low,mid,high);

    }//End if

    return 0;

}
```



```c
//Merge function
//Puts the structures in the correct order of product ID and line ID
int Merge(struct QAlogs *products,struct QAlogs *tempA,int low,int mid,int high)
{
    int i;
    int left = low;
    int right = mid + 1;
    int temp = left;

    while(left <= mid && right <= high)//Runs while there is still items in both arrays
    {
        if((products+left) -> product_id < (products+right) -> product_id)//Product ID check
        {
            *(tempA + temp++) = *(products+left++);
        }
        else if((products+left) -> product_id == (products+right) -> product_id)
        {
            if((products +left) -> line_code <= (products +right) -> line_code)//Line id check
            {
                *(tempA +temp++) = *(products+left++);

            }
            else
            {
                *(tempA+temp++) = *(products+right++);
            }//End line ID check
        }
        else
        {
            *(tempA + temp++) = *(products+right++);
        }//End product ID Check

    }//end while

    //If nothing is left in left array
    if (left>mid)
    {
        while(right<=high)
        {
            *(tempA+temp++) = *(products+right++);
        }
    }
```



```
    else//If nothing is left in right array
    {
        while(left<=mid)
        {
            *(tempA+temp++) =  *(products+left++);
        }
    }

    //Copies all the structures back into the original structure array in the correct order
    for (i = low; i <= high; i++)
    {
        *(products + i) = *(tempA + i);
    }
    return 0;

}
```

Task 3

Design:

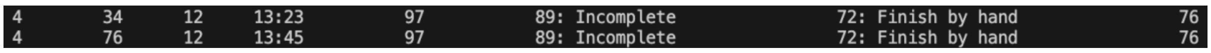
My design for task 3 uses binary search to find the earliest occurrence of an issue code of a given product id across all production lines. The binary search algorithm takes in a sorted array of all production lines sorted by product id and issue code (products), a temporary array of structures (search), the variables low, high and mid and finally the product id (product_key) and issue id (issue_key) you are looking for. Once it is sorted by product id and issue code and all these parameters it will be able to find the earliest occurrence. The first if statement will keep the search running once low is not bigger then high as if so it means that the product id and issue code do not exist. Inside the if, it starts by calculating the mid. Once it has that, it checks if the product id and issue code in that structure at the mid of the array of structures is equal to the one we are looking for if not it checks if the product id is greater than or the product_id is equal and the issue code is greater than and if so runs the binary search function again with low being mid + 1and if not runs the binary search function again with high being mid – 1. If it finds that the product id and issue code in that structure at the mid of the array is the same, it will run the inner if and will mean it is in the right area of the array. It will then check if the product id and issue code of the structure before is the same and if so will mean it is not the earliest occurrence and will run the binary search algorithm again with high and low as mid – 1. If not it will put that structure into the temporary array of structures. Once this is done the algorithm will stop, it will return back to main and the temporary array of structures is outputted.

Running time:

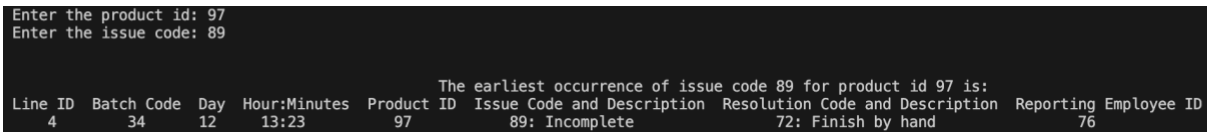
The running time of this algorithm is O(log(n)) which meets the running time requirements of O(log(n)) or better. This is because the mid is always being calculated which halves the amount of data in the array every time which is a logarithmic calculation giving the algorithm a running time of O(log(n)).

Test plan:

For this task my test plan had four tests. The first test being it could find a unique product id and issue code. The second test being if it could find the issue code that occurred the first in that product id if it had multiple of that same issue code. The third test was to check if it could find the largest product id and the smallest product id. The final test was if the product id and issue code didn't exist. Once it can pass all these it is able to find the earliest occurrence of any issue code of any product id.

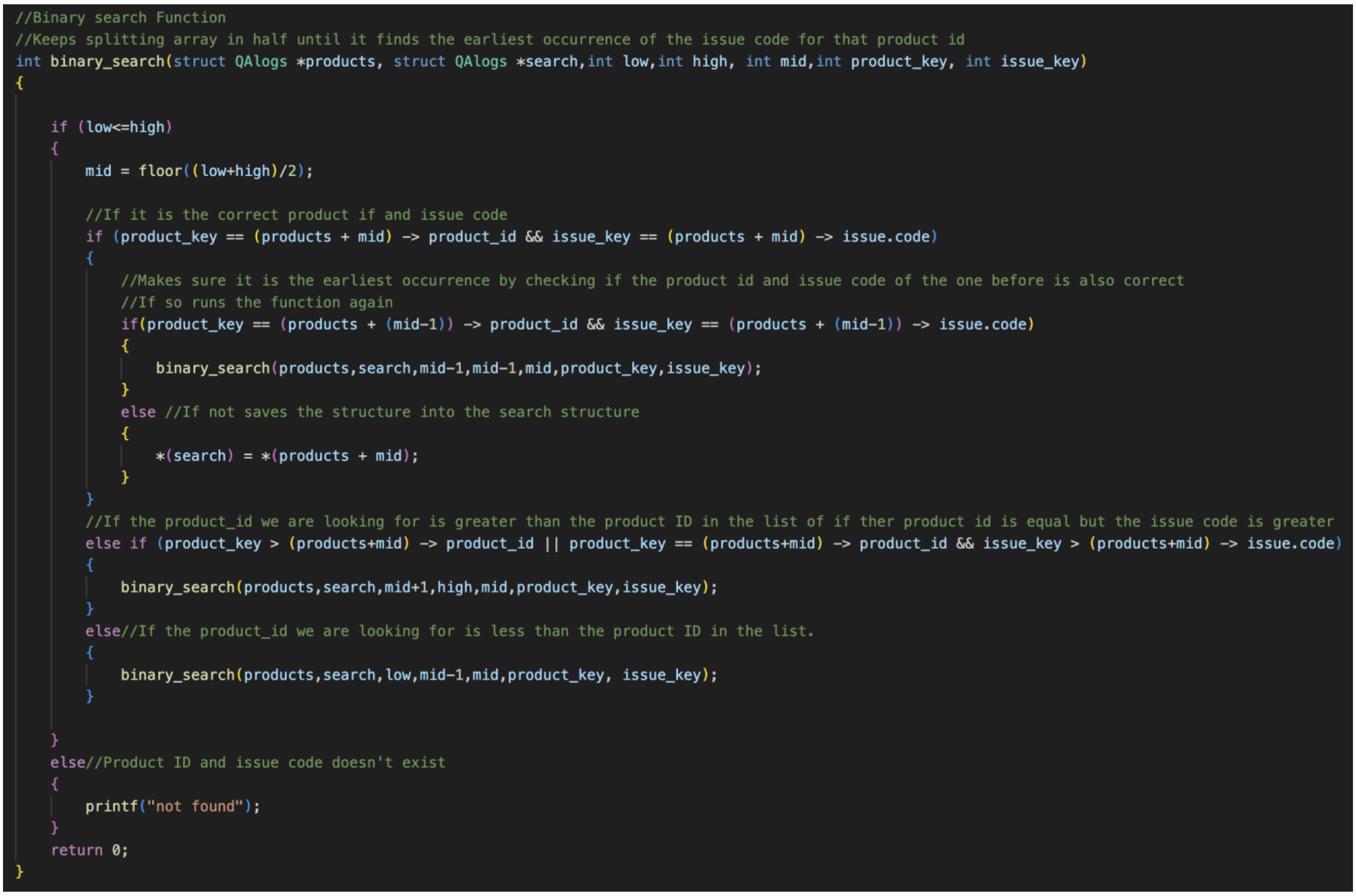


```
4       34      12      13:23           97          89: Incomplete                  72: Finish by hand                      76
4       76      12      13:45           97          89: Incomplete                  72: Finish by hand                      76
```



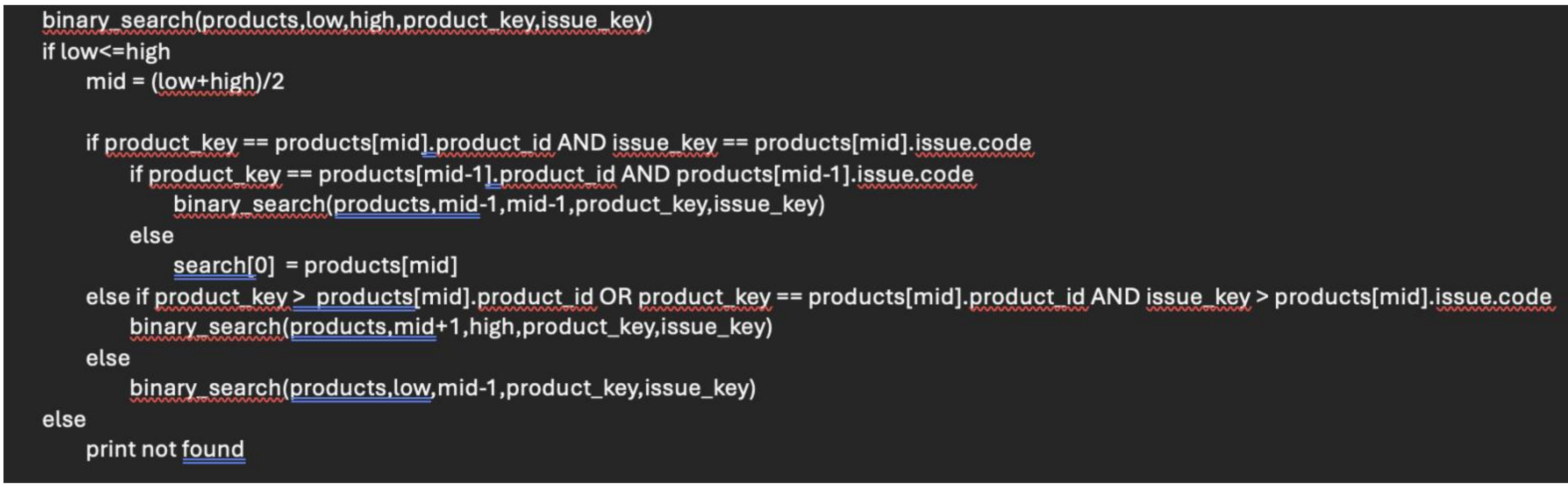
```
Enter the product id: 97
Enter the issue code: 89


                                      The earliest occurrence of issue code 89 for product id 97 is:
Line ID  Batch Code  Day  Hour:Minutes  Product ID  Issue Code and Description  Resolution Code and Description  Reporting Employee ID
   4         34      12      13:23          97           89: Incomplete                 72: Finish by hand                 76
```

Working C Code



```c
//Binary search Function
//Keeps splitting array in half until it finds the earliest occurrence of the issue code for that product id
int binary_search(struct QAlogs *products, struct QAlogs *search,int low,int high, int mid,int product_key, int issue_key)
{

    if (low<=high)
    {
        mid = floor((low+high)/2);

        //If it is the correct product if and issue code
        if (product_key == (products + mid) -> product_id && issue_key == (products + mid) -> issue.code)
        {
            //Makes sure it is the earliest occurrence by checking if the product id and issue code of the one before is also correct
            //If so runs the function again
            if(product_key == (products + (mid-1)) -> product_id && issue_key == (products + (mid-1)) -> issue.code)
            {
                binary_search(products,search,mid-1,mid-1,mid,product_key,issue_key);
            }
            else //If not saves the structure into the search structure
            {
                *(search) = *(products + mid);
            }
        }
        //If the product_id we are looking for is greater than the product ID in the list of if ther product id is equal but the issue code is greater
        else if (product_key > (products+mid) -> product_id || product_key == (products+mid) -> product_id && issue_key > (products+mid) -> issue.code)
        {
            binary_search(products,search,mid+1,high,mid,product_key,issue_key);
        }
        else//If the product_id we are looking for is less than the product ID in the list.
        {
            binary_search(products,search,low,mid-1,mid,product_key, issue_key);
        }

    }
    else//Product ID and issue code doesn't exist
    {
        printf("not found");
    }
    return 0;
}
```

Pseudocode



```
binary_search(products,low,high,product_key,issue_key)
if low<=high
    mid = (low+high)/2

    if product_key == products[mid].product_id AND issue_key == products[mid].issue.code
        if product_key == products[mid-1].product_id AND products[mid-1].issue.code
            binary_search(products,mid-1,mid-1,product_key,issue_key)
        else
            search[0]  = products[mid]
    else if product_key > products[mid].product_id OR product_key == products[mid].product_id AND issue_key > products[mid].issue.code
        binary_search(products,mid+1,high,product_key,issue_key)
    else
        binary_search(products,low,mid-1,product_key,issue_key)
else
    print not found
```

Task 4

Design:

My design for this task is called the summary function. It takes in a sorted array by product id and high which is the location of the last item in the array which is 51 in this case. For my design to work the array must be sorted by product id which can be done using the algorithm from task 2. The variable j keeps track of what product id we are counting. The for loop is used to go through every item in the array. Inside the for loop, the if statement checks if the product id in the structure located in the array at i is the same as the product id in the structure at location j. If so it adds one to count. If it is not equal it means we are on a new product id which causes it to output the product id and the number of issues it had by outputting count, it resets the count back to 1 as we have found one issue for the new product id and sets j to i as this is the new product id we are now going to count. It keeps running this if statement until i gets to 53. Once the for loop ends, it outputs the final product id and count.

Running Time:

The running time of this algorithm is O(n) which meets the running time requirements of O(n) or better. It has this running time due to the for loop running n times.

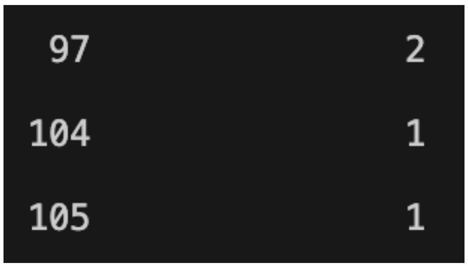
Test plan:

My test plan for this algorithm was to change one of the product id's to the same as another and see if it adds one to the value it had outputted before. This allows me to see that it will work even if the data was different.

Example

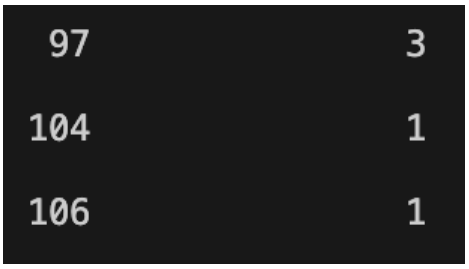
Changed one structures product id from 105 to 97

Before



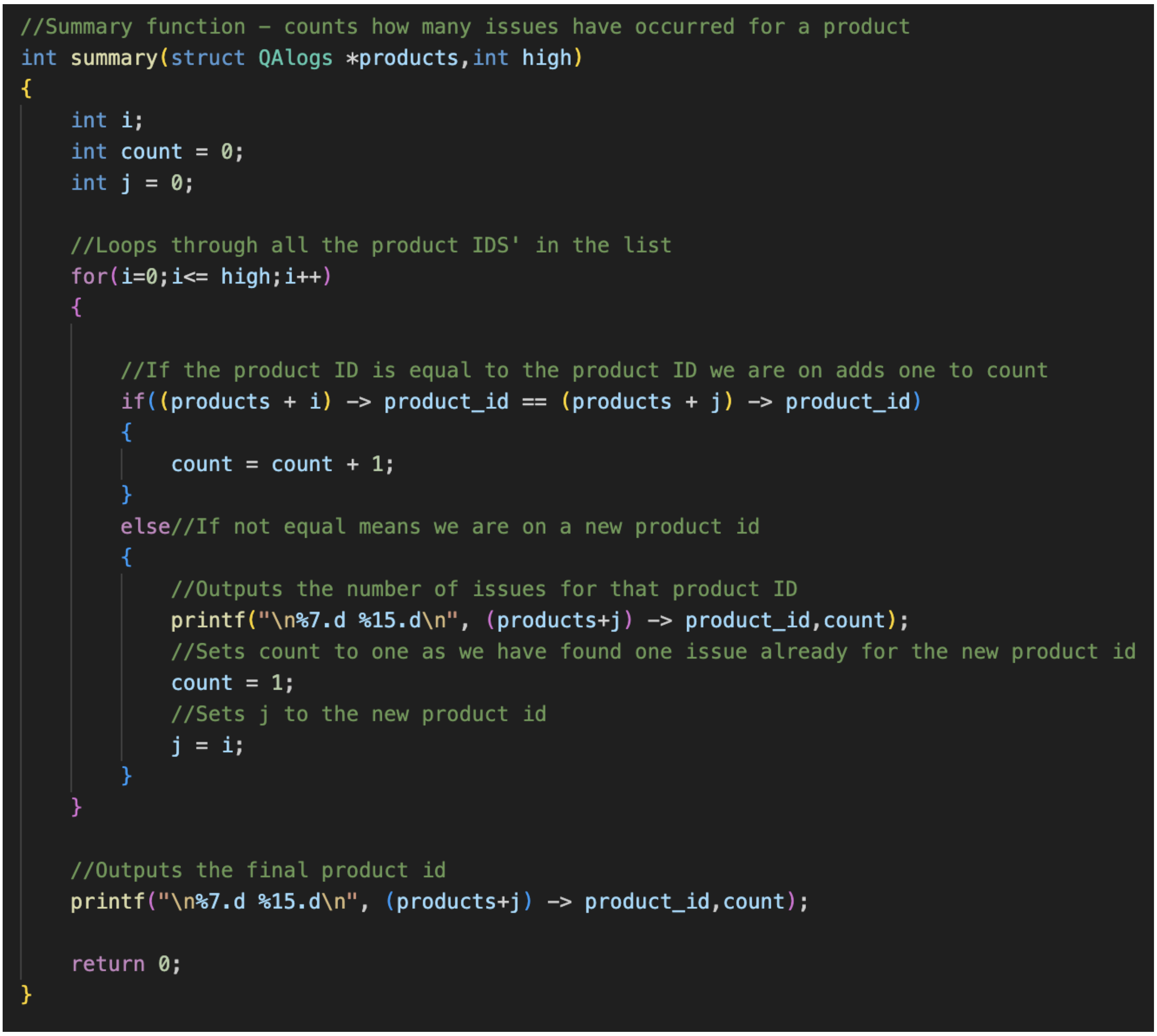
```
 97                    2
104                    1
105                    1
```

After



```
 97                    3
104                    1
106                    1
```

Working C Code



```c
//Summary function - counts how many issues have occurred for a product
int summary(struct QAlogs *products,int high)
{
    int i;
    int count = 0;
    int j = 0;

    //Loops through all the product IDS' in the list
    for(i=0;i<= high;i++)
    {

        //If the product ID is equal to the product ID we are on adds one to count
        if((products + i) -> product_id == (products + j) -> product_id)
        {
            count = count + 1;
        }
        else//If not equal means we are on a new product id
        {
            //Outputs the number of issues for that product ID
            printf("\n%7.d %15.d\n", (products+j) -> product_id,count);
            //Sets count to one as we have found one issue already for the new product id
            count = 1;
            //Sets j to the new product id
            j = i;
        }
    }

    //Outputs the final product id
    printf("\n%7.d %15.d\n", (products+j) -> product_id,count);

    return 0;
}
```

Pseudocode

```
summary(products,high)
count = 0
j = 0
for i=0 to i <= high do
   if products[i].product_id == products[j].product_id
      count = count + 1
   else
      print product[j].product_id, count
      count = 1
      j = i
Endfor
Print product[j].product_id, count
```