**Project Report**

Table of Contents

[**Introduction** 1](#_Toc162973152)

[**Design** 1](#_Toc162973153)

[**Test Data** 1](#_Toc162973154)

[**Task 1 Walkthrough** 2](#_Toc162973155)

[**Task 2 Walkthrough** 2](#_Toc162973156)

[**Task 3 Walkthrough** 2](#_Toc162973157)

[**Task 4 Walkthrough** 2](#_Toc162973158)

[**Pseudocode + Code** 3](#_Toc162973159)

[**Pseudocode** 3](#_Toc162973160)

[**Code** 11](#_Toc162973161)

[**Flowchart** 19](#_Toc162973162)

# **Introduction**

In this project we were given a prompt stating that “a pharmaceutical company manufactures 4 product lines from its factory in Dublin”. In this report I will be preparing informaton on the manufacturing process, creating test data based on the given tasks, and implementing functions based on the tasks given to complete.

# **Design**

In this project, the main data structure was to use the struct command in c to store different data types for the different variables.

The sorting algorithm used was the merge sort algorithm because of its efficiency. Merge sort uses recursion to divide, conquer and combine a given array in order.

# **Test Data**

The data below is the test data used in the program to get results and to test each task.

struct department employees[SIZE] = {

        {2, 796, {4, 12, 15.05}, 23, {202, "Issue Desc..."}, {23, "Resolution Desc..."}, 101},

        {4, 648, {6, 2, 20.55}, 84, {365, "Issue Desc..."}, {69, "Resolution Desc..."}, 102},

        {6, 408, {28, 4, 16.45}, 39, {724, "Issue Desc..."}, {23, "Resolution Desc..."}, 103},

        {8, 384, {23, 11, 9.38}, 27, {592, "Issue Desc..."}, {18, "Resolution Desc..."}, 104},

        {3, 497, {13, 9, 18.27}, 51, {365, "Issue Desc..."}, {38, "Resolution Desc..."}, 105},

        {5, 297, {2, 3, 11.15}, 18, {104, "Issue Desc..."}, {49, "Resolution Desc..."}, 106},

        {7, 743, {16, 3, 12.45}, 23, {145, "Issue Desc..."}, {96, "Resolution Desc..."}, 109},

        {9, 392, {5, 9, 21.25}, 39, {115, "Issue Desc..."}, {18, "Resolution Desc..."}, 108},

        {1, 487, {18, 12, 8.15}, 21, {165, "Issue Desc..."}, {29, "Resolution Desc..."}, 107},

        {10, 276, {6, 4, 4.55}, 51, {120, "Issue Desc..."}, {18, "Resolution Desc..."}, 110}

};

# **Task 1 Walkthrough**

This task was asking to organise the line logs by product ID, then issue code, then date and time. In this task, merge sort was used to organise the array to be in the order needed. Then another function was created to display the new sorted data, only displaying product id, issue code, and date & time.

# **Task 2 Walkthrough**

This task was stating that there was a change in the manufacturing process and that the same product could be manufactured on different lines. It’s similar to task 1 as merge sort was also used on this. The main difference is that malloc() was used to temporarily allocate memory to the issueReport array, then freed right when the program is finished. Another difference is that instead of sorting by date and time, it was instead sorted by product id, then line id for all of the lines. Another function was created to display the sorted data, only showing the product id, line code and issue code.

# **Task 3 Walkthrough**

This task is where the scanf() function was used. It was asking to use code to find the earliest occurrence of an issue code. Using a for loop as well as an if statement to find when the first occurrence of the given issue code was the easiest method. In the same function, the sorted data was displayed, only showing the product id, line code and issue code if the given issue code was found in the program. If not found the program would display an appropriate error message for the user.

# **Task 4 Walkthrough**

This task was asking to provide a report summary of the number of issues reported per product. Basically, meaning that for each product, to count the number of issues reported, then to display it out onto the terminal. A for loop with a nested for loop was made to count the number of issues found for every product id. The displayed variables was the product id and the number of issues counted.

# **Pseudocode + Code**

**Pseudocode:**

START

DEFINE SIZE 10

DEFINE LENGTH 20

Typedef char STRING

STRUCT dateTime:

Int day

Int month

Float hourandMin

STRUCT desc:

Int code

STRING description[LENGTH]

STRUCT department:

Int lineCode

Int batchCode

dateTime batchDateTime

int productid

desc issue

desc resolution

int employeeID

VOID displayIssueReport(struct department \*\*emp)

VOID displaySortedDepartment(struct department \*emp)

VOID mergeSort(struct department \*emp, low, high)

VOID merge(struct department \*emp, low, mid, high)

VOID issueSort(struct department \*\*emp, low, high)

VOID issueMerge(struct department \*\*emp, low, mid, high)

VOID searching(struct department \*emp)

VOID noOFIssues(struct department \*emp)

ALGORITHM main():

Int i

STRUCT department \*issueReport[SIZE]

STRUCT department employees[SIZE] // initialised array

// TASK 1

mergeSort(employees, 0, SIZE-1)

displaySortedDepartment(employees)

// TASK 2

FOR I = 0 to SIZE-1:

// allocates memory for issueReport array

issueReport[i] = (struct department \*)malloc(Sizeof(struct department))

IF issueReport[i] != NULL:

IssueReport[i]->productID = employees[i].productID

issueReport[i]->issue.code = employees[i].issue.code

issueReport[i]->lineCode = employees[i].lineCode

END IF

END FOR

issueSort(issueReport, 0, SIZE-1)

PRINT ”issue Report Code”

displayIssueReport(issueReport)

FOR I = 0 to SIZE-1:

IF issueReport[i] != NULL:

FREE issueReport[i]

END IF

END FOR

// TASK 3

PRINT “Search”

searching(employees)

// TASK 4

PRINT ”Summary Report of Issues”

noOFIssues(employees)

RETURN 0

END MAIN ALGORITHM

ALGORITHM mergeSort(emp, low, high);

IF low < high:

mid = (low + high) / 2

mergeSort(emp, low, mid)

mergeSort(emp, mid+1, high)

merge(emp, low, mid, high)

END IF

END mergeSort ALGORITHM

ALGORITHM merge(emp, low, mid, high):

temp[high – low + 1]

int I, j

int k = low

int ptrL = mid – low + 1

int ptrR = high – mid

STRUCT department LEFT[LENGTH], RIGHT[LENGTH]

FOR I = 0 to ptrL:

LEFT[i] = emp[low + i]

END FOR

FOR j = 0 to ptrR:

RIGHT[j] = emp[mid + 1 + j]

END FOR

i = 0, j = 0

WHILE i < ptrL AND j < ptrR:

IF LEFT[i].productID < RIGHT[j].productID OR

LEFT[i].productID == RIGHT[i].productID AND LEFT[i].issue.code < RIGHT[i].issue.code OR

LEFT[i].productID == RIGHT[i].productID AND LEFT[i].issue.code == RIGHT[i].issue.code AND LEFT[i].batchDateTime.month < RIGHT[j].batchDateTime.month OR

LEFT[i].productID == RIGHT[i].productID AND LEFT[i].issue.code == RIGHT[i].issue.code AND LEFT[i].batchDateTime.month == RIGHT[i].batchDateTime.month AND LEFT[i].batchDateTime.day < RIGHT[i].batchDateTime.day OR

LEFT[i].productID == RIGHT[i].productID AND LEFT[i].issue.code == RIGHT[i].issue.code AND LEFT[i].batchDateTime.month == RIGHT[i].batchDateTime.month AND LEFT[i].batchDateTime.day == RIGHT[i].batchDateTime.day AND LEFT[i].batchDateTime.hourandMin < RIGHT[i].batchDateTime.hourandMin:

emp[k] = LEFT[i]

k = k + 1

i = i + 1

ELSE:

emp[k] = RIGHT[j]

k = k + 1

j = j + 1

END IF

END WHILE

WHILE i < ptrL:

emp[k] = LEFT[i]

k = k + 1

i = i + 1

END WHILE

WHILE j < ptrR:

emp[k] = RIGHT[j]

k = k + 1

j = j + 1

END WHILE

END merge ALGORITHM

ALGORITHM displaySortedDepartment(emp):

FOR I = 0 to SIZE-1:

PRINT “Department”

PRINT “Product ID:”, emp[i].productID

PRINT “Issue Code:”, emp[i].issue.code

PRINT “Date:”, emp.batchDateTime.date, “/”, emp[i].batchDateTime.month, “/2024, Time:”, emp[i].batchDsteTime.hourandMin

END FOR

END displaySortedDepartment ALGORITHM

ALGORITHM issueSort(emp, low, high):

IF low < high:

mid = (low + high) / 2

issueSort(emp, low, mid)

issueSort(emp, mid+1, high)

IssueMerge(emp, low, mid, high)

END IF

END issueSort ALGORITHM

ALGORITHM issueMerge(emp, low, mid, high):

temp[high – low + 1]

int I, j

int k = low

int ptrL = mid – low + 1

int ptrR = high – mid

STRUCT department LEFT[LENGTH], RIGHT[LENGTH]

FOR I = 0 to ptrL:

LEFT[i] = emp[low + i]

END FOR

FOR j = 0 to ptrR:

RIGHT[j] = emp[mid + 1 + j]

END FOR

i = 0, j = 0

WHILE i < ptrL AND j < ptrR:

IF LEFT[i].productID < RIGHT[j].productID OR

LEFT[i].productID == RIGHT[i].productID AND LEFT[i].lineCode < RIGHT[i].lineCode:

emp[k] = LEFT[i]

k = k + 1

i = i + 1

ELSE:

emp[k] = RIGHT[j]

k = k + 1

j = j + 1

END IF

END WHILE

WHILE i < ptrL:

emp[k] = LEFT[i]

k = k + 1

i = i + 1

END WHILE

WHILE j < ptrR:

emp[k] = RIGHT[j]

k = k + 1

j = j + 1

END WHILE

END issueMerge ALGORITHM

ALGORITHM displayIssueReport(issueReport):

Int i;

FOR I = 0 to SIZE-1:

PRINT “Product ID:”, issueReport[i]->productID

PRINT “Issue Code:”, issueReport[i]->issue.code

PRINT “Line Code:”, issueReport[i]->lineCode

END FOR

END displayIssueReport ALGORITHM

ALGORITHM searching(emp):

Int search, i

Int key = 0

PRINT “please enter the issue code:”

READ search

FOR i = 0 to SIZE-1:

IF emp[i].issue.code == search:

key = 1

BREAK OUT OF LOOP

END IF

END FOR

IF key == 1:

PRINT “Initial Occurrence 🡪 Product ID:”, emp[i].productID, “Line Code:”, emp[i].lineCode, “Issue Code:”, emp[i].issue.code

ELSE:

PRINT “Issue Code Invalid”

END IF

END searching ALGORITHM

ALGORITHM noOFIssues(emp):

Int i, j

Int count[SIZE] = {0}

FOR i = 0 to SIZE-1:

FOR j = 0 to SIZE-1:

IF emp[i].productID == emp[j].productID:

count[i] = count[i] + 1

END IF

END INNER FOR

END OUTER FOR

FOR i = 0 to SIZE-1:

IF emp[i – 1].productID == emp[i].productID:

i = i + 1

END IF

PRINT “Product ID:”, emp[i].productID, “Number of Issues”, count[i]

END FOR

END noOFIssues ALGORITHM

END PROGRAM

**Code:**

*/\**

*Programs Description: This program demonstrates*

*(1) sorting line logs by product ID, issue code, and date/time*

*(2) sorting the files according to product id, then line code*

*(3) searches for the first occurence of an issue code*

*(4) finds the number of issues per product id*

*Author: Blessing Ugochukwu*

*Date: 29/03/24*

*\*/*

*#include* <stdio.h>

*#include* <stdlib.h>

*#define* SIZE 10

*#define* LENGTH 20

typedef char STRING;

*// structure templates*

struct dateTime {

    int day; *// DD*

    int month; *//MM*

    float hourandMin; *// HR.MIN*

};

struct desc {

    int code;

    STRING description[LENGTH];

};

struct department {

    int lineCode;

    int batchCode;

    struct dateTime batchDateTime;

    int productID;

    struct desc issue;

    struct desc resolution;

    int employeeID;

};

*// function assignments*

void displayDepartment(struct department \*);

void displayIssueReport(struct department \*\*);

void displaySortedDepartment(struct department \*);

void mergeSort(struct department \*, int, int);

void merge(struct department \*, int, int, int);

void issueSort(struct department \*\*, int, int);

void issueMerge(struct department \*\*, int, int, int);

void searching(struct department \*);

void noOFIssues(struct department \*);

int main() {

    int i, size;

    struct department \*issueReport[SIZE];

    struct department employees[SIZE] = {

        {2, 796, {4, 12, 15.05}, 23, {202, "Issue Desc..."}, {23, "Resolution Desc..."}, 101},

        {4, 648, {6, 2, 20.55}, 84, {365, "Issue Desc..."}, {69, "Resolution Desc..."}, 102},

        {6, 408, {28, 4, 16.45}, 39, {724, "Issue Desc..."}, {23, "Resolution Desc..."}, 103},

        {8, 384, {23, 11, 9.38}, 27, {592, "Issue Desc..."}, {18, "Resolution Desc..."}, 104},

        {3, 497, {13, 9, 18.27}, 51, {365, "Issue Desc..."}, {38, "Resolution Desc..."}, 105},

        {5, 297, {2, 3, 11.15}, 18, {104, "Issue Desc..."}, {49, "Resolution Desc..."}, 106},

        {7, 743, {16, 3, 12.45}, 23, {145, "Issue Desc..."}, {96, "Resolution Desc..."}, 109},

        {9, 392, {5, 9, 21.25}, 39, {202, "Issue Desc..."}, {18, "Resolution Desc..."}, 108},

        {1, 487, {18, 12, 8.15}, 21, {165, "Issue Desc..."}, {29, "Resolution Desc..."}, 107},

        {10, 276, {6, 4, 4.55}, 51, {120, "Issue Desc..."}, {18, "Resolution Desc..."}, 110}

    };

*// displays the unsorted department elements*

    printf("--------Unsorted Data--------\n");

    displayDepartment(employees);

*//TASK 1*

*// uses mergeSort to sort the elements in the employees array*

    mergeSort(employees, 0, SIZE - 1);

*// displays the sorted department elements*

    printf("--------Sorted Data--------\n");

    displaySortedDepartment(employees);

*// TASK 2*

*// loop to runs through each element in the array*

*for* (i = 0; i < SIZE; i++) {

*// allocates memory for the issueReport department struct*

        issueReport[i] = (struct department \*)malloc(sizeof(struct department));

*// checks if the allocation was successful*

*if* (issueReport[i] != NULL) {

*// assigns values to the issueReport array using values from the employees array*

            issueReport[i]->productID = employees[i].productID;

            issueReport[i]->issue.code = employees[i].issue.code;

            issueReport[i]->lineCode = employees[i].lineCode;

        } *// end if*

    } *// end for*

*// uses issueSort to sort the elements in the employees array*

    issueSort(issueReport, 0, SIZE - 1);

*// displays the issue report*

    printf("-----Issue Code Report-----\n");

    displayIssueReport(issueReport);

*// Remember to use and eventually free the allocated memory for issueReport*

*for* (i = 0; i < SIZE; i++) {

*if* (issueReport[i] != NULL) {

            free(issueReport[i]);

        } *// end if*

    } *// end for*

*// TASK 3*

*// preforms a search on the employee data*

    printf("\n\n---------Search---------\n");

    searching(employees);

*// TASK 4*

*// generates a summary report of issues*

    printf("\n\n------Summary Report of Issues------\n");

    noOFIssues(employees);

*return* 0;

} *// end main*

*// displays the unsorted data*

void displayDepartment(struct department *emp*[]) {

    int i;

*for* (i = 0; i < SIZE; i++) {

        printf("-----------------------\n");

        printf("Line Code: %d\n", *emp*[i].lineCode);

        printf("Batch Code: %d\n", *emp*[i].batchCode);

        printf("Date: %d/%d/2024, Time: %.2f\n", *emp*[i].batchDateTime.day, *emp*[i].batchDateTime.month, *emp*[i].batchDateTime.hourandMin);

        printf("Product ID: %d\n", *emp*[i].productID);

        printf("Issue Code: %d\n", *emp*[i].issue.code);

        printf("Issue Description: %s\n", *emp*[i].issue.description);

        printf("Resolution Code: %d\n", *emp*[i].resolution.code);

        printf("Resolution Description: %s\n", *emp*[i].resolution.description);

        printf("Employee ID: %d\n\n", *emp*[i].employeeID);

    } *// end for*

} *// end displayDepartment*

*// TASK 1*

*// uses the merge sort algorithm to sort each variable O(NLog(N))*

void mergeSort(struct department *emp*[], int *low*, int *high*) {

    int mid;

*// checks if the low value is lower than the high value*

*if* (*low* < *high*) {

        mid = (*low* + *high*) / 2;

        mergeSort(*emp*, *low*, mid);

        mergeSort(*emp*, mid + 1, *high*);

        merge(*emp*, *low*, mid, *high*);

    } *// end if*

} *// end mergeSort*

*// the second part of mergeSort (arranging the values in order)*

void merge(struct department *emp*[], int *low*, int *mid*, int *high*) {

    int temp[*high* - *low* + 1];

    int i, j, k = *low*;

    int ptrL = *mid* - *low* + 1, ptrR = *high* - *mid*;

*// creates temporary arrays*

    struct department LEFT[LENGTH], RIGHT[LENGTH];

*// copying data into the temp arrays*

*for* (i = 0; i < ptrL; i++) {

        LEFT[i] = *emp*[*low* + i];

    } *// end for*

*for* (j = 0; j < ptrR; j++) {

        RIGHT[j] = *emp*[*mid* + 1 + j];

    } *// end for*

    i = 0, j = 0;

*// merges the temp arrays back to emp*

*while* (i < ptrL && j < ptrR) {

*// compares the values by date and time*

*if* (LEFT[i].productID < RIGHT[j].productID ||

        LEFT[i].productID == RIGHT[i].productID && LEFT[i].issue.code < RIGHT[i].issue.code ||

        LEFT[i].productID == RIGHT[i].productID && LEFT[i].issue.code == RIGHT[i].issue.code && LEFT[i].batchDateTime.month < RIGHT[j].batchDateTime.month ||

        LEFT[i].productID == RIGHT[i].productID && LEFT[i].issue.code == RIGHT[i].issue.code && LEFT[i].batchDateTime.month == RIGHT[i].batchDateTime.month &&

        LEFT[i].batchDateTime.day < RIGHT[i].batchDateTime.day ||

        LEFT[i].productID == RIGHT[i].productID && LEFT[i].issue.code == RIGHT[i].issue.code && LEFT[i].batchDateTime.month == RIGHT[i].batchDateTime.month &&

        LEFT[i].batchDateTime.day == RIGHT[i].batchDateTime.day &&

        LEFT[i].batchDateTime.hourandMin < RIGHT[i].batchDateTime.hourandMin) {

*emp*[k++] = LEFT[i++];

        } *else* {

*emp*[k++] = RIGHT[j++];

        } *// end if*

    } *// end while*

*// copies the rest of the values into emp if there is any*

*while* (i < ptrL) {

*emp*[k++] = LEFT[i++];

    } *// end while*

*while* (j < ptrR) {

*emp*[k++] = RIGHT[j++];

    } *// end while*

} *// end merge*

*// displays the sorted departments in order of date and time*

void displaySortedDepartment(struct department *emp*[]) {

    int i;

*// reports each line log in productID, issueCode, an dateandTime*

*for* (i = 0; i < SIZE; i++) {

        printf("---Department---\n");

        printf("Product ID: %d\n", *emp*[i].productID);

        printf("Issue Code: %d\n", *emp*[i].issue.code);

        printf("Date: %d/%d/2024, Time: %.2f\n\n", *emp*[i].batchDateTime.day, *emp*[i].batchDateTime.month, *emp*[i].batchDateTime.hourandMin);

    } *// end for*

} *// end displaySortedDepartment*

*// TASK 2*

*// uses the merge sort algorithm to sort each variable O(NLog(N))*

void issueSort(struct department \*\**emp*, int *low*, int *high*) {

    int mid;

*// checks if the low value is lower than the high value*

*if* (*low* < *high*) {

        mid = (*low* + *high*) / 2;

        issueSort(*emp*, *low*, mid);

        issueSort(*emp*, mid + 1, *high*);

        issueMerge(*emp*, *low*, mid, *high*);

    } *// end if*

} *// end mergeSort*

*// the second part of mergeSort (arranging the values in order)*

void issueMerge(struct department \*\**emp*, int *low*, int *mid*, int *high*) {

    int temp[*high* - *low* + 1];

    int i, j, k = *low*;

    int ptrL = *mid* - *low* + 1, ptrR = *high* - *mid*;

*// creates temporary arrays*

    struct department LEFT[LENGTH], RIGHT[LENGTH];

*// copying data into the temp arrays*

*for* (i = 0; i < ptrL; i++) {

        LEFT[i] = \**emp*[*low* + i];

    } *// end for*

*for* (j = 0; j < ptrR; j++) {

        RIGHT[j] = \**emp*[*mid* + 1 + j];

    } *// end for*

    i = 0, j = 0;

*// merges the temp arrays back to emp*

*while* (i < ptrL && j < ptrR) {

*// compares the values by date and time*

*if* (LEFT[i].productID < RIGHT[j].productID ||

        LEFT[i].productID == RIGHT[i].productID && LEFT[i].lineCode < RIGHT[i].lineCode) {

            \**emp*[k++] = LEFT[i++];

        } *else* {

            \**emp*[k++] = RIGHT[j++];

        } *// end if*

    } *// end while*

*// copies the rest of the values into emp if there is any*

*while* (i < ptrL) {

        \**emp*[k++] = LEFT[i++];

    } *// end while*

*while* (j < ptrR) {

        \**emp*[k++] = RIGHT[j++];

    } *// end while*

} *// end issueMerge*

*// displays the departments one by one*

void displayIssueReport(struct department \*\**issueReport*) {

    int i;

*// prints the product id, issue code, and line code*

*for* (i = 0; i < SIZE; i++) {

        printf("Product ID: %d, ", *issueReport*[i]->productID);

        printf("Issue Code: %d, ", *issueReport*[i]->issue.code);

        printf("Line code: %d\n", *issueReport*[i]->lineCode);

    } *// end for*

} *// end displayIssueReport*

*// TASK 3*

*// searches for the initial occurence of an issue code*

void searching(struct department *emp*[]) {

    int search, i, key = 0;

*// asks the user to enter an issue code*

    printf("Please enter the issue code: ");

    scanf("%d", &search);

*// compares the given issue code with the issue codes in the structure*

*for* (i = 0; i < SIZE; i++) {

*if* (*emp*[i].issue.code == search) {

            key = 1;

*break*; *// breaks out of the loop if the issue code is found*

        } *// end if*

    } *// end for*

*// checks if the issue code given is in the structure and prints the corresponding product id and line code*

*if* (key == 1) {

        printf("Initial Occurrence --> Product ID: %d, Line Code: %d, Issue Code: %d \n", *emp*[i].productID, *emp*[i].lineCode, *emp*[i].issue.code);

    } *else* {

        printf("Issue Code Invalid\n");

    } *// end if*

} *// end searching*

*// TASK 4*

*// makes a report summary of the number of issues per product*

void noOFIssues(struct department *emp*[]) {

    int i, j, count[SIZE] = {0};

*// loop to find the amount of occurences of an issue*

*for* (i = 0; i < SIZE; i++) {

*for* (j = 0; j < SIZE; j++) {

*if* (*emp*[i].productID == *emp*[j].productID) {

                count[i]++;

            } *// end if*

        } *// end for*

    } *// end for*

*// checks if the current product id has been printed already*

*for* (i = 0; i < SIZE; i++) {

*if* (*emp*[i - 1].productID == *emp*[i].productID) {

            i++;

        } *// end if*

*// prints the product ID as well as the number of issues*

        printf("Product ID: %d ---- Number of Issues: %d\n", *emp*[i].productID, count[i]);

    } *// end for*

} *// end noOFIssues*

# **Flowchart**

Main:

A diagram of a function

Description automatically generated with medium confidence A diagram of a flowchart

Description automatically generated

issueSort:

A screenshot of a computer

Description automatically generated

issueMerge:

A screenshot of a computer flowchart

Description automatically generatedA diagram of a flowchart

Description automatically generated

A diagram of a flowchart

Description automatically generatedA diagram of a flowchart

Description automatically generated

k