# **TU856/8 Assignment 2023/24**

# Name: Beatrice Maria Giurgica

**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 (group) 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

**DESIGN REQUIREMENTS**

**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.

**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 (order) 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.

**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.

**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.

Your documentation should show how each algorithm meets the running time requirement.

**DELIVERABLES**

1. Design a data structure for the project.
2. Create test data for each line. e.g. >10 issues/products per line.
3. Test your project – outline how you will ensure that your implementation meets the design requirement.
4. Produce a flowchart for Task 2.
5. Produce pseudocode for Tasks 1-4.
6. Produce working C code for tasks 1-4.
7. Create a project report outlining the brief, your design for each task, your test plan for each task and the pseudocode + code.
   1. The report should be submitted in MS Word .docx format OR Adobe PDF format .pdf. Reports not meeting these requirements will not be marked.
   2. Code should be readable.
   3. Submit by 5pm 7th April 2024. Late submission -10% per day.

Projects will be demoed in the lab. The sequence of the demo should be:

1. Display data from each production line
2. Display data from each production line meeting requirements of Task1
3. Display data from all production lines meeting requirements of Task2
4. Prompt for search and display result meeting requirements of Task3
5. Display summary data meeting requirements of Task4

Unanswered or incorrect responses to questions on the design or implementation will result in an NG. Marks are allocated equally to all sections, including the demo.

# **DATA STRUCTURE**

To tackle the following tasks effectively, a structured approach has been adopted to enclose the essential attributes for each log, which are Line Code, Batch Code, Batch Date & Time, Product Id, Issue Code & Description, Resolution Code & Description, and Reporting Employee Id. In addition, the “DateTime” structure has been created specifically to articulate the Batch Date & Time attributes within the “ProductionLine\_Log” structure. The “DateTime” structure consists of three integer fields: “dayofmonth”, “hourofday”, and “minuteofhour”.

#include <stdio.h>

#include <stdlib.h>

// Define DateTime structure

struct DateTime

{

    int dayofmonth;

    int hourofday;

    int minuteofhour;

};

// Define ProductionLine\_Log structure

struct ProductionLine\_Log

{

    int LineCode;

    int BatchCode;

    struct DateTime BatchDateTime;

    int ProductId;

    int IssueCode;

    char IssueDescription[100];

    int ResolutionCode;

    char ResolutionDescription[100];

    int ReportingEmployeeId;

};

# **test DATA**

In the main function, a set of test data representative of ProductionLine\_Logs has been generated to illustrate the information in each of these logs.

// Example ProductionLine\_Log array

    struct ProductionLine\_Log logs\_data[] =

    {

        {1, 101, {1, 20, 45}, 1001, 10, "Defect in wing", 10, "Resolved by replacing

faulty component", 100},

        {2, 102, {1, 5, 45}, 1003, 12, "Fire detection system malfunction", 12,

"Resolved by conducting system testing", 105},

        {5, 105, {1, 11, 45}, 1002, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {1, 107, {1, 7, 45}, 1001, 1, "Product damage", 1, "Resolved by adding final

quality checks", 100},

        {9, 102, {1, 16, 45}, 1003, 3, "Shipping delay", 3, "Resolved by increasing

number of couriers", 105},

        {6, 103, {1, 23, 45}, 1005, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {1, 111, {1, 10, 45}, 1001, 17, "Transportation issue", 17, "Resolved by

improving transportation methods", 100},

        {3, 116, {1, 6, 45}, 1006, 20, "Customer complaint", 20, "Resolved with

personalised solutions", 105},

        {1, 109, {1, 22, 45}, 1002, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {8, 107, {1, 21, 45}, 1005, 6, "Labelling issue", 6, "Resolved by

implementing revised labelling procedures", 100},

        {7, 112, {1, 18, 45}, 1001, 12, "Fire detection system malfunction", 12,

"Resolved by conducting system testing", 105},

        {1, 108, {1, 4, 45}, 1007, 30, "Staff shortage", 30, "Resolved by hiring

temporary staff", 102},

    };

# **Task 1**

The primary objective of this task revolves around the creation of a report for each production line, sorted based on Product Id, Issue Code, and Date & Time attributes. To effectively tackle this task, the Merge Sort algorithm emerges as the most suitable choice due to its efficiency in managing large datasets. In fact, Merge Sort demonstrates admirable performance with a worst-case time complexity of O(NLog(N)).

To address the task, a total of 3 functions have been implemented.

The initial function, named "merge", plays an important role in merging two subarrays within the logs\_data array. Using temporary arrays, this function copies data from the original array into these temporary arrays, and then merges them back into the original array based on specific criteria such as Product ID, Issue Code, and Batch Date & Time. During this process, the merge function operates with a time complexity of O(N), moving across each element precisely once.

Subsequently, the "MergeSort" function is employed to divide the array into smaller subarrays before merging them in a sorted order using the merge function. This divide-and-conquer strategy contributes to MergeSort's time complexity of O(NLog(N)), ensuring efficient sorting even with substantial amount of data.

In conclusion, the "PrintReport" function is created to meet the specific requirements of printing the sorted Production Line Report.

**Pseudocode**

#Merge two subarrays of logs\_data[]

#First subarray goes from left to mid

#Second subarray goes from mid+1 to right

Function merge(logs\_data[], left, mid, right)

left\_size := mid - left + 1

right\_size := right - mid

#Create temporary arrays

temp\_left := left\_size

temp\_right := right\_size

#Copy data to temporary arrays temp\_left[] and temp\_right[]

for i := 0 to left\_size-1 do

temp\_left[i] := logs\_data[left + i]

end for

for j := 0 to right\_size-1 do

temp\_right[j] := logs\_data[mid + 1 + j]

end for

#Merge the temporary arrays back into logs\_data[]

i := 0 #initial index of first subarray

j := 0 #initial index of second subarray

k := left #initial index of merged subarray

while i < left\_size and j < right\_size do

#Compare based on Product ID, Issue Code and Batch Date & Time

if temp\_left[i].ProductId < temp\_right[j].ProductId or

(temp\_left[i].ProductId = temp\_right[j].ProductId and

temp\_left[i].IssueCode < temp\_right[j].IssueCode) or

(temp\_left[i].ProductId = temp\_right[j].ProductId and

temp\_left[i].IssueCode = temp\_right[j].IssueCode and

(temp\_left[i].BatchDateTime.dayofmonth <

temp\_right[j].BatchDateTime.dayofmonth or

(temp\_left[i].BatchDateTime.dayofmonth =

temp\_right[j].BatchDateTime.dayofmonth and

(temp\_left[i].BatchDateTime.hourofday <

temp\_right[j].BatchDateTime.hourofday or

(temp\_left[i].BatchDateTime.hourofday =

temp\_right[j].BatchDateTime.hourofday and

temp\_left[i].BatchDateTime.minuteofhour <

temp\_right[j].BatchDateTime.minuteofhour))))) then

logs\_data[k] := temp\_left[i]

i := i + 1

else

logs\_data[k] := temp\_right[j]

j := j + 1

end if

k := k + 1

end while

#Copy the remaining elements of temp\_left[], if there are any

while i < left\_size do

logs\_data[k] := temp\_left[i]

i := i + 1

k := k + 1

end while

#Copy the remaining elements of temp\_right[], if there are any

while j < right\_size do

logs\_data[k] := temp\_right[j]

j := j + 1

k := k + 1

end while

end function

#Merge Sort function to sort logs\_data[] based on Product ID, Issue Code and Batch Date & Time

Function mergeSort(logs\_data[], left, right)

if left < right then

mid := left + (right - left) / 2

#Sort left and right halves

mergeSort(logs\_data, left, mid)

mergeSort(logs\_data, mid + 1, right)

#Merge sorted halves

merge(logs\_data, left, mid, right)

end if

end function

Function printReport(logs\_data[], size)

print("Sorted Production Line Report:")

for i := 0 to size-1 do

print("Product ID: ", logs\_data[i].ProductId)

print("Issue Code: ", logs\_data[i].IssueCode)

print("Date & Time: ", logs\_data[i].BatchDateTime.dayofmonth, "

(day of the month) ", logs\_data[i].BatchDateTime.hourofday, ":",

logs\_data[i].BatchDateTime.minuteofhour, " (time)")

print()

end for

end function

**C code**

#include <stdio.h>

#include <stdlib.h>

// Define DateTime structure

struct DateTime

{

    int dayofmonth;

    int hourofday;

    int minuteofhour;

};

// Define ProductionLine\_Log structure

struct ProductionLine\_Log

{

    int LineCode;

    int BatchCode;

    struct DateTime BatchDateTime;

    int ProductId;

    int IssueCode;

    char IssueDescription[100];

    int ResolutionCode;

    char ResolutionDescription[100];

    int ReportingEmployeeId;

};

// Merge two subarrays of logs\_data[]

// First subarray goes from left to mid

// Second subarray goes from mid+1 to right

void merge(struct ProductionLine\_Log logs\_data[], int left, int mid, int right)

{

    int i, j, k;

    int left\_size = mid - left + 1;

    int right\_size = right - mid;

    // Create temporary arrays

    struct ProductionLine\_Log temp\_left[left\_size], temp\_right[right\_size];

    // Copy data to temporary arrays temp\_left[] and temp\_right[]

    for (i = 0; i < left\_size; i++)

    {

        temp\_left[i] = logs\_data[left + i];

    }

    for (j = 0; j < right\_size; j++)

    {

        temp\_right[j] = logs\_data[mid + 1 + j];

    }

    // Merge the temporary arrays back into logs\_data[]

    i = 0; //initial index of first subarray

    j = 0; //initial index of second subarray

    k = left; //initial index of merged subarray

    while (i < left\_size && j < right\_size)

    {

        // Compare based on Product ID, Issue Code and Batch Date & Time

        if (temp\_left[i].ProductId < temp\_right[j].ProductId ||

            (temp\_left[i].ProductId == temp\_right[j].ProductId && temp\_left[i].IssueCode

< temp\_right[j].IssueCode) ||(temp\_left[i].ProductId ==

temp\_right[j].ProductId && temp\_left[i].IssueCode == temp\_right[j].IssueCode

&& (temp\_left[i].BatchDateTime.dayofmonth <

temp\_right[j].BatchDateTime.dayofmonth ||

(temp\_left[i].BatchDateTime.dayofmonth ==

temp\_right[j].BatchDateTime.dayofmonth &&

           (temp\_left[i].BatchDateTime.hourofday < temp\_right[j].BatchDateTime.hourofday

||(temp\_left[i].BatchDateTime.hourofday ==

temp\_right[j].BatchDateTime.hourofday &&

            temp\_left[i].BatchDateTime.minuteofhour <

temp\_right[j].BatchDateTime.minuteofhour))))))

        {

            logs\_data[k] = temp\_left[i];

            i++;

        }

        else

        {

            logs\_data[k] = temp\_right[j];

            j++;

        }

        k++;

    }

    // Copy the remaining elements of temp\_left[], if there are any

    while (i < left\_size)

    {

        logs\_data[k] = temp\_left[i];

        i++;

        k++;

    }

    // Copy the remaining elements of temp\_right[], if there are any

    while (j < right\_size)

    {

        logs\_data[k] = temp\_right[j];

        j++;

        k++;

    }

}

/\* Merge Sort function to sort logs\_data[] based on Product ID, Issue Code and Batch Date & Time \*/

void mergeSort(struct ProductionLine\_Log logs\_data[], int left, int right)

{

    if (left < right)

    {

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

        // Sort left and right halves

        mergeSort(logs\_data, left, mid);

        mergeSort(logs\_data, mid + 1, right);

        // Merge sorted halves

        merge(logs\_data, left, mid, right);

    }

}

// Function to print the sorted report

void printReport(struct ProductionLine\_Log logs\_data[], int size)

{

    printf("Sorted Production Line Report:\n");

    for (int i = 0; i < size; i++)

    {

        printf("Product ID: %d\n", logs\_data[i].ProductId);

        printf("Issue Code: %d\n", logs\_data[i].IssueCode);

        printf("Date & Time: %d (day of the month) %d:%d (time)\n",

logs\_data[i].BatchDateTime.dayofmonth, logs\_data[i].BatchDateTime.hourofday,

logs\_data[i].BatchDateTime.minuteofhour);

        printf("\n");

    }

}

int main()

{

    // Example ProductionLine\_Log array

    struct ProductionLine\_Log logs\_data[] =

    {

        {1, 101, {1, 20, 45}, 1001, 10, "Defect in wing", 10, "Resolved by replacing

faulty component", 100},

        {2, 102, {1, 5, 45}, 1003, 12, "Fire detection system malfunction", 12,

"Resolved by conducting system testing", 105},

        {5, 105, {1, 11, 45}, 1002, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {1, 107, {1, 7, 45}, 1001, 1, "Product damage", 1, "Resolved by adding final

quality checks", 100},

        {9, 102, {1, 16, 45}, 1003, 3, "Shipping delay", 3, "Resolved by increasing

number of couriers", 105},

        {6, 103, {1, 23, 45}, 1005, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {1, 111, {1, 10, 45}, 1001, 17, "Transportation issue", 17, "Resolved by

improving transportation methods", 100},

        {3, 116, {1, 6, 45}, 1006, 20, "Customer complaint", 20, "Resolved with

personalised solutions", 105},

        {1, 109, {1, 22, 45}, 1002, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {8, 107, {1, 21, 45}, 1005, 6, "Labelling issue", 6, "Resolved by

implementing revised labelling procedures", 100},

        {7, 112, {1, 18, 45}, 1001, 12, "Fire detection system malfunction", 12,

"Resolved by conducting system testing", 105},

        {1, 108, {1, 4, 45}, 1007, 30, "Staff shortage", 30, "Resolved by hiring

temporary staff", 102},

    };

    /\* Determine the number of logs in the array to ensure that the logs\_number

variable holds the correct number of logs, even if the size of the array

changes in the future. \*/

    int logs\_number = sizeof(logs\_data) / sizeof(logs\_data[0]);

    // Display logs\_data

    printf("Unsorted Production Line Report:\n");

    for (int i = 0; i < logs\_number; i++)

    {

        printf("Production Line: %d\n", logs\_data[i].LineCode);

        printf("Batch Code: %d\n", logs\_data[i].BatchCode);

        printf("Batch Date & Time: %d (day of the month) %d:%d (time)\n",

logs\_data[i].BatchDateTime.dayofmonth, logs\_data[i].BatchDateTime.hourofday,

logs\_data[i].BatchDateTime.minuteofhour);

        printf("Product ID: %d\n", logs\_data[i].ProductId);

        printf("Issue Code: %d\n", logs\_data[i].IssueCode);

        printf("Issue Description: %s\n", logs\_data[i].IssueDescription);

        printf("Resolution Code: %d\n", logs\_data[i].ResolutionCode);

        printf("Resolution Description: %s\n", logs\_data[i].ResolutionDescription);

        printf("Reporting Employee ID: %d\n", logs\_data[i].ReportingEmployeeId);

        printf("\n");

    }

    // Perform merge sort on logs\_data based on Product ID

    mergeSort(logs\_data, 0, logs\_number - 1);

    // Print the sorted report

    printReport(logs\_data, logs\_number);

    return 0;

}

# **Task 2**

The primary aim of this task is to create a report that groups together issues related to the same product and production line. In order to meet the requirements of this task, a linked list data structure is used to organize and manage the logs from the production lines. This choice of design is driven by the necessity to preserve the order of logs based on their Product ID and Line Code attributes, which in turn facilitates the efficient sorting of data.

The key functions that have been implemented include the “insertLog” function and the “generateReport” function.

The “insertLog” function is responsible for inserting logs into the linked list while ensuring that they remain in sorted order. This is achieved by allocating memory for a new node, navigating the linked list to find the correct position for insertion based on Product ID and Line Code, and then inserting the new node at the appropriate position in the linked list.

The “generateReport” function generates and prints the sorted production line report by going through the linked list.

**Flowchart**

**A diagram of a company

Description automatically generated**

**A diagram of a software flowchart

Description automatically generated**

**A diagram of a computer program

Description automatically generated**

**Pseudocode**

#Function to insert a log into the linked list while maintaining order based on Product ID and Line Code

Function insertLog(head: pointer to struct Node, struct ProductionLine\_Log log)

#Allocate memory for the new node

newNode := allocate memory for Node

if newNode = NULL then

print "Memory allocation failed."

return

end if

#Assign log data to the new node

newNode.data := log

newNode.next := NULL

current := head

prev := NULL

#Navigate through the list to find the correct position based on

Product ID and Line Code

while current != NULL and (current.data.ProductId <

newNode.data.ProductId or (current.data.ProductId =

newNode.data.ProductId and current.data.LineCode <

newNode.data.LineCode)) do

prev := current

current := current.next

end while

#Insert newNode at the correct position

if prev = NULL then

newNode.next := head

head := newNode

else

prev.next := newNode

newNode.next := current

end if

end function

Function generateReport(head: pointer to Node)

current := head

print "Production Line Report:"

#Print log details

while current != NULL do

print "Product ID: " + current.data.ProductId + " Line Code: " +

current.data.LineCode + " Issue Code: " + current.data.IssueCode

current := current.next

end while

end function

**C code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Define DateTime structure

struct DateTime

{

    int dayofmonth;

    int hourofday;

    int minuteofhour;

};

// Define ProductionLine\_Log structure

struct ProductionLine\_Log

{

    int LineCode;

    int BatchCode;

    struct DateTime BatchDateTime;

    int ProductId;

    int IssueCode;

    char IssueDescription[100];

    int ResolutionCode;

    char ResolutionDescription[100];

    int ReportingEmployeeId;

};

// Define a linked list node for production line logs

struct Node

{

    struct ProductionLine\_Log data;

    struct Node \*next;

};

/\* Function to insert a log into the linked list while maintaining order based on Product ID and Line Code \*/

void insertLog(struct Node \*\*head, struct ProductionLine\_Log log)

{

    // Allocate memory for the new node

    struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

    if (newNode == NULL)

    {

        printf("Memory allocation failed.\n");

        return;

    }

    // Assign log data to the new node

    newNode->data = log;

    newNode->next = NULL;

    struct Node \*current = \*head;

    struct Node \*prev = NULL;

    /\* Navigate through the list to find the correct position based on Product ID

and Line Code \*/

    while (current != NULL && (current->data.ProductId < newNode->data.ProductId ||

(current->data.ProductId == newNode->data.ProductId && current->

data.LineCode < newNode->data.LineCode)))

    {

        prev = current;

        current = current->next;

    }

    // Insert newNode at the correct position

    if (prev == NULL)

    {

        newNode->next = \*head;

        \*head = newNode;

    } else

    {

        prev->next = newNode;

        newNode->next = current;

    }

}

// Function to generate and print the report

void generateReport(struct Node \*head)

{

    struct Node \*current = head;

    printf("Production Line Report:\n");

    // Print log details

    while (current != NULL)

    {

        printf("Product ID: %d  Line Code: %d  Issue Code: %d\n", current->

data.ProductId, current->data.LineCode, current->data.IssueCode);

        current = current->next;

    }

}

int main() {

    // Example ProductionLine\_Log array

    struct ProductionLine\_Log logs\_data[] =

    {

        {1, 101, {1, 20, 45}, 1001, 10, "Defect in wing", 10, "Resolved by replacing

faulty component", 100},

        {2, 102, {1, 5, 45}, 1003, 12, "Fire detection system malfunction", 12,

"Resolved by conducting system testing", 105},

        {5, 105, {1, 11, 45}, 1002, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {1, 107, {1, 7, 45}, 1001, 1, "Product damage", 1, "Resolved by adding final

quality checks", 100},

        {9, 102, {1, 16, 45}, 1003, 3, "Shipping delay", 3, "Resolved by increasing

number of couriers", 105},

        {6, 103, {1, 23, 45}, 1005, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {1, 111, {1, 10, 45}, 1001, 17, "Transportation issue", 17, "Resolved by

improving transportation methods", 100},

        {3, 116, {1, 6, 45}, 1006, 20, "Customer complaint", 20, "Resolved with

personalised solutions", 105},

        {1, 109, {1, 22, 45}, 1002, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {8, 107, {1, 21, 45}, 1005, 6, "Labelling issue", 6, "Resolved by

implementing revised labelling procedures", 100},

        {7, 112, {1, 18, 45}, 1001, 12, "Fire detection system malfunction", 12,

"Resolved by conducting system testing", 105},

        {1, 108, {1, 4, 45}, 1007, 30, "Staff shortage", 30, "Resolved by hiring

temporary staff", 102},

    };

    int logs\_number = sizeof(logs\_data) / sizeof(logs\_data[0]);

    // Create an empty linked list

    struct Node \*head = NULL;

    // Insert logs into the linked list while maintaining order

    for (int i = 0; i < logs\_number; i++)

    {

        insertLog(&head, logs\_data[i]);

    }

    // Generate and print the report

    generateReport(head);

    return 0;

}

# **Task 3**

The primary aim of this task is to analyse data from different production lines to search for the earliest occurrence of an issue code for a given product ID.

To solve it efficiently, the binary search algorithm is chosen due to its logarithmic time complexity O(Log(N)), which ensures efficient searching even with large datasets.

The “searchEarliestOccurrence” function implements binary search, narrowing the search space through a divide-and-conquer approach. This process repeatedly divides the search interval in half and compares the target value to the middle element, efficiently locating the earliest occurrence while meeting time complexity requirements.

**Pseudocode**

#Binary search function to find the earliest occurrence of issue code for a product ID

Function searchEarliestOccurrence(struct ProductionLine\_Log logs\_data[], logs\_number, productID)

left := 0

right := logs\_number - 1

earliestIndex := -1

while left <= right do

#Calculate mid point

mid := left + (right - left) / 2

#Check if the current log matches the productID

if logs\_data[mid].ProductId = productID then

earliestIndex := mid

return earliestIndex

else if logs\_data[mid].ProductId > productID then

right := mid - 1 #Search in the left subarray

else

left := mid + 1 #Search in the right subarray

return earliestIndex

end function

**C code**

#include <stdio.h>

// Define DateTime structure

struct DateTime

{

    int dayofmonth;

    int hourofday;

    int minuteofhour;

};

// Define ProductionLine\_Log structure

struct ProductionLine\_Log

{

    int LineCode;

    int BatchCode;

    struct DateTime BatchDateTime;

    int ProductId;

    int IssueCode;

    char IssueDescription[100];

    int ResolutionCode;

    char ResolutionDescription[100];

    int ReportingEmployeeId;

};

/\* Binary search function to find the earliest occurrence of issue code for a product ID \*/

int searchEarliestOccurrence(struct ProductionLine\_Log logs\_data[], int logs\_number, int productID)

{

    int left = 0;

    int right = logs\_number - 1;

    int earliestIndex = -1;

    while (left <= right)

    {

        // Calculate mid point

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

        // Check if the current log matches the productID

        if (logs\_data[mid].ProductId == productID)

        {

            earliestIndex = mid;

            return earliestIndex;

        }

        else if (logs\_data[mid].ProductId > productID)

        {

            right = mid - 1;  // Search in the left subarray

        }

        else

        {

            left = mid + 1;  // Search in the right subarray

        }

    }

    return earliestIndex;

}

int main()

{

    // Example ProductionLine\_Log array

    struct ProductionLine\_Log logs\_data[] =

    {

        {1, 101, {1, 20, 45}, 1001, 10, "Defect in wing", 10, "Resolved by replacing

faulty component", 100},

        {2, 102, {1, 5, 45}, 1003, 12, "Fire detection system malfunction", 12,

"Resolved by conducting system testing", 105},

        {5, 105, {1, 11, 45}, 1002, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {1, 107, {1, 7, 45}, 1001, 1, "Product damage", 1, "Resolved by adding final

quality checks", 100},

        {9, 102, {1, 16, 45}, 1003, 3, "Shipping delay", 3, "Resolved by increasing

number of couriers", 105},

        {6, 103, {1, 23, 45}, 1005, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {1, 111, {1, 10, 45}, 1001, 17, "Transportation issue", 17, "Resolved by

improving transportation methods", 100},

        {3, 116, {1, 6, 45}, 1006, 20, "Customer complaint", 20, "Resolved with

personalised solutions", 105},

        {1, 109, {1, 22, 45}, 1002, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {8, 107, {1, 21, 45}, 1005, 6, "Labelling issue", 6, "Resolved by

implementing revised labelling procedures", 100},

        {7, 112, {1, 18, 45}, 1001, 12, "Fire detection system malfunction", 12,

"Resolved by conducting system testing", 105},

        {1, 108, {1, 4, 45}, 1007, 30, "Staff shortage", 30, "Resolved by hiring

temporary staff", 102},

    };

    int logs\_number = sizeof(logs\_data) / sizeof(logs\_data[0]);

    int productID;

    printf("Enter Product ID to search: ");

    scanf("%d", &productID);

    // Perform binary search for earliest occurrence

    int earliestIndex = searchEarliestOccurrence(logs\_data, logs\_number, productID);

    if (earliestIndex != -1)

    {

        printf("Earliest occurrence of Product ID %d found at index %d\n",

productID, earliestIndex);

        printf("Related Issue Code: %d\n", logs\_data[earliestIndex].IssueCode);

    }

    else

    {

        printf("Product ID %d not found in logs.\n", productID);

    }

    return 0;

}

# **Task 4**

The primary aim of this task is to efficiently summarise the number of issues reported for a specific product across all production lines. Given the substantial amount of data involved, it is crucial to employ an algorithm with a running time of O(N) or better to ensure optimal efficiency.

Considering these requirements, Linear Search emerges as a suitable choice due to its simplicity and effectiveness in counting occurrences within unsorted data. Although Linear Search typically operates with a time complexity of O(N), its structure enables efficient processing of large datasets, meeting this task's time complexity goals. The algorithm goes through each element in the dataset until it finds the target value, which in this case is the specified product ID. By comparing each element's product ID with the target ID and incrementing a counter upon a match, Linear Search ensures that all occurrences of the product ID are counted. The function responsible for implementing the linear search on the provided ProductionLine\_Log array is named "countIssues."

**Pseudocode**

#Function to perform linear search and count issues for a product ID

Function countIssues(struct ProductionLine\_Log logs\_data[], num\_logs, productID)

count = 0

for i = 0 to num\_logs

if logs\_data[i].ProductId = productID then

count = count + 1

end if

end for

return count

end function

**C code**

#include <stdio.h>

// Define DateTime structure

struct DateTime

{

    int dayofmonth;

    int hourofday;

    int minuteofhour;

};

// Define ProductionLine\_Log structure

struct ProductionLine\_Log

{

    int LineCode;

    int BatchCode;

    struct DateTime BatchDateTime;

    int ProductId;

    int IssueCode;

    char IssueDescription[100];

    int ResolutionCode;

    char ResolutionDescription[100];

    int ReportingEmployeeId;

};

// Function to perform linear search and count issues for a product ID

int countIssues(struct ProductionLine\_Log logs\_data[], int num\_logs, int productID)

{

    int count = 0;

    for (int i = 0; i < num\_logs; i++)

    {

        if (logs\_data[i].ProductId == productID)

        {

            count++;

        }

    }

    return count;

}

int main()

{

    // Example ProductionLine\_Log array

    struct ProductionLine\_Log logs\_data[] =

    {

        {1, 101, {1, 20, 45}, 1001, 10, "Defect in wing", 10, "Resolved by replacing

faulty component", 100},

        {2, 102, {1, 5, 45}, 1003, 12, "Fire detection system malfunction", 12,

"Resolved by conducting system testing", 105},

        {5, 105, {1, 11, 45}, 1002, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {1, 107, {1, 7, 45}, 1001, 1, "Product damage", 1, "Resolved by adding final

quality checks", 100},

        {9, 102, {1, 16, 45}, 1003, 3, "Shipping delay", 3, "Resolved by increasing

number of couriers", 105},

        {6, 103, {1, 23, 45}, 1005, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {1, 111, {1, 10, 45}, 1001, 17, "Transportation issue", 17, "Resolved by

improving transportation methods", 100},

        {3, 116, {1, 6, 45}, 1006, 20, "Customer complaint", 20, "Resolved with

personalised solutions", 105},

        {1, 109, {1, 22, 45}, 1002, 15, "Engine malfunction", 15, "Resolved by

replacing affected parts", 102},

        {8, 107, {1, 21, 45}, 1005, 6, "Labelling issue", 6, "Resolved by

implementing revised labelling procedures", 100},

        {7, 112, {1, 18, 45}, 1001, 12, "Fire detection system malfunction", 12,

"Resolved by conducting system testing", 105},

        {1, 108, {1, 4, 45}, 1007, 30, "Staff shortage", 30, "Resolved by hiring

temporary staff", 102},

    };

    int logs\_number = sizeof(logs\_data) / sizeof(logs\_data[0]);

    int productID;

    printf("Enter Product ID to count issues: ");

    scanf("%d", &productID);

    // Perform linear search and count issues for the given Product ID

    int issue\_count = countIssues(logs\_data, logs\_number, productID);

    if (issue\_count > 0)

    {

        printf("Number of issues for Product ID %d: %d\n", productID, issue\_count);

    }

    else

    {

        printf("No issues found for Product ID %d.\n", productID);

    }

    return 0;

}