**ALGORITHM DESIGN & PROBLEM-SOLVING CA PROJECT REPORT**

STUDENT NUMBER: C23310756

COURSE: TU856/1

LAB GROUP: A

**DATA STRUCTURE**

For my data structure, I created a structure for the batch date & time, Issue code & description, and resolution code & description. I then nested these 3 structures inside my Production Line structure.

//header files

#include <stdio.h>

#include <stdlib.h>

//symbolic names

#define SIZE 100//for the issue and resolution descriptions

#define PRODUCTS 15//for the size of the structure array

#define PRODLINES 60 //for the size of the combined array

//structure for the  batch date and time

struct dateTime

{

    int batchday;

    int batchhour;

    int batchminute;

};

//structure for the issue code and description

struct Issue

{

    int issueCode;

    char issuedescription[SIZE];

};

//structure for the resolution code and description

struct resolution

{

    int resolutionCode;

    char resolution\_description[SIZE];

};

//structure for the production lines

struct productionLines

{

    int lineCode;

    int batchCode;

    struct dateTime DT;//nested structure

    int productID;

    struct Issue issue;//nested structure

    struct resolution res;//nested strucutre

    int employeeID;

};

For my data structures, I named them record1, record2, etc, and had 10 products in each. Some had multiple issues and some products share the same issues. Overall, in my structure, I had an array with size 15 in my main:

//structure array for the first production line

    struct productionLines record1[PRODUCTS] =

    {

        {1, 123461, {23, 15, 40}, 567895, {106,"Battery failure detected in emergency backup system"}, {206, "Replaced the faulty battery with a new one"}, 1000006},

        {1, 103464, {25, 18, 55}, 567898, {109,"Power supply failure reported in critical systems"}, {209, "Replaced power supply unit to ensure uninterrupted operation"}, 1000009},

        {1, 123458, {03, 12, 25}, 567892, {137, "Power supply failure detected in primary power distribution unit"}, {237, "Replaced power supply unit to ensure continuous operation"}, 101000},

        {1, 153463, {20, 17, 50}, 567897, {108,"Sensor malfunction detected in environmental monitoring system"}, {208, "Calibrated sensor for accurate readings"}, 1000008},

        {1, 123457, {13, 11, 20}, 567891, {102,"Assembly error found during final inspection"}, {202, "Reassembled the unit correctly according to specifications"}, 1000002},

        {1, 173465, {05, 19, 00}, 567899, {110,"Display screen defect reported in user interface module"}, {210, "Replaced defective display unit with a new one"}, 1000010},

        {1, 113461, {31, 16, 45}, 567896, {107,"Memory corruption issue observed in system logs"}, {207, "Updated firmware to address memory management issues"}, 1000007},

        {1, 193460, {07, 14, 35}, 567891, {134, "Hardware malfunction detected in critical subsystem"}, {234, "Repaired hardware components to restore functionality"}, 100700},

        {1, 123456, {26, 10, 15}, 567890, {125, "Power surge detected in primary power supply unit"}, {226, "Installed surge protection to mitigate power surges"}, 101101},

        {1, 163459, {18, 13, 30}, 567893, {104,"Critical software bug identified in the operating system"}, {204, "Fixed the software bug with a patch update"}, 1000004},

        {1, 123457, {15, 11, 20}, 567898, {125, "Power surge detected in primary power supply unit"}, {226, "Installed surge protection to mitigate power surges"}, 101101},

        {1, 143462, {12, 16, 45}, 567896, {117, "Input/output error observed in data transfer process"}, {217, "Resolved I/O issues to ensure data integrity"}, 100800},

        {1, 123460, {07, 14, 35}, 567894, {105,"Network connectivity failure reported by end-users"}, {205, "Reconfigured network settings to resolve connectivity issues"}, 1000005},

        {1, 113470, {14, 13, 30}, 567893, {123, "Hardware failure observed in I/O subsystem"}, {224, "Replaced faulty hardware components in I/O subsystem"}, 100801},

        {1, 113461, {29, 15, 40}, 567890, {101,"Faulty component detected in the main control unit"}, {201, "Replaced faulty component with a new and tested one"}, 1000001}

    };

//structure array for the second production line

    struct productionLines record2[PRODUCTS] =

    {

        {2, 128709, {01, 12, 30}, 589000, {111, "Faulty connection detected in the communication module"}, {211, "Repaired connection by replacing damaged components"}, 100201},

        {2, 108002, {15, 14, 40}, 589200, {113, "Temperature sensor error reported in thermal monitoring system"}, {213, "Replaced temperature sensor with a calibrated unit"}, 100402},

        {2, 168709, {25, 21, 15}, 589900, {120, "Power failure detected in primary power supply unit"}, {220, "Restored power supply to prevent system downtime"}, 101101},

        {2, 158002, {23, 15, 45}, 589300, {104,"Critical software bug identified in the operating system"}, {204, "Fixed the software bug with a patch update"}, 1000004},

        {2, 138204, {02, 16, 50}, 589400, {109,"Power supply failure reported in critical systems"}, {209, "Replaced power supply unit to ensure uninterrupted operation"}, 1000009},

        {2, 167800, {06, 12, 30}, 589000, {121, "Communication error detected between subsystems"}, {222, "Debugged communication protocols to resolve errors"}, 100201},

        {2, 178406, {07, 18, 00},  589600,{117, "Input/output error observed in data transfer process"}, {217, "Resolved I/O issues to ensure data integrity"}, 100800},

        {2, 197901, {19, 13, 35}, 589100, {112, "Component misalignment detected in mechanical assembly"}, {212, "Realigned components to ensure proper functionality"}, 100300},

        {2, 168608, {16, 20, 10}, 589800, {119, "Firmware corruption detected in control firmware"}, {219, "Restored firmware from backup to resolve corruption"}, 101000},

        {2, 128709, {11, 21, 15}, 589900, {125, "Power surge detected in primary power supply unit"}, {226, "Installed surge protection to mitigate power surges"}, 101101},

        {2, 118507, {24, 19, 05},  589700,{118, "Hardware malfunction detected in critical subsystem"}, {218, "Repaired hardware components to restore functionality"}, 100900},

        {2, 188103, {12, 15, 45}, 589300, {122, "Critical security vulnerability detected in control software"}, {223, "Patched software to fix security vulnerability"}, 100501},

        {2, 148406, {13, 18, 00},  589600,{123, "Hardware failure observed in I/O subsystem"}, {224, "Replaced faulty hardware components in I/O subsystem"}, 100801},

        {2, 158002, {29, 14, 40}, 589200, {137, "Power supply failure detected in primary power distribution unit"}, {237, "Replaced power supply unit to ensure continuous operation"}, 101000},

        {2, 167800, {30, 12, 30}, 589000, {135, "Software bug detected in control software"}, {235, "Fixed software bug to prevent system errors"}, 100800}

    };

//structure array for the third production line

    struct productionLines record3[PRODUCTS] =

    {

        {3, 138408, {29, 18, 30}, 546509, {134, "Hardware malfunction detected in critical subsystem"}, {234, "Repaired hardware components to restore functionality"}, 100700},

        {3, 187802, {04, 11, 55}, 545802, {102,"Assembly error found during final inspection"}, {202, "Reassembled the unit correctly according to specifications"}, 1000002},

        {3, 128307, {18, 16, 20}, 546307, {125, "Power surge detected in primary power supply unit"}, {226, "Installed surge protection to mitigate power surges"}, 101101},

        {3, 178105, {22, 14, 10}, 546105, {126, "Fan failure detected in cooling system"}, {226, "Replaced fan to prevent overheating"}, 100800},

        {3, 158004, {16, 13, 05}, 546004,{137, "Power supply failure detected in primary power distribution unit"}, {237, "Replaced power supply unit to ensure continuous operation"}, 101000},

        {3, 138206, {19, 15, 15}, 546206, {127, "Voltage surge detected in power supply unit"}, {227, "Installed surge protector to prevent damage from surges"}, 100900},

        {3, 187600, {23, 9, 45},  545903, {124, "System overload detected in processing unit"}, {224, "Optimized system resources to prevent overload"}, 100600},

        {3, 138408, {27, 17, 25}, 546408, {107,"Memory corruption issue observed in system logs"}, {207, "Updated firmware to address memory management issues"}, 1000007},

        {3, 188307, {05, 16, 20}, 546307, {129, "Hardware failure detected in communication channel"}, {229, "Replaced faulty hardware components to restore functionality"}, 101005},

        {3, 127903, {10, 12, 00}, 545903,{124, "System overload detected in processing unit"}, {224, "Optimized system resources to prevent overload"}, 100600},

        {3, 168105, {24, 14, 10}, 546105, {122, "Leakage detected in cooling system"}, {222, "Fixed leakage and installed additional seals"}, 100805},

        {3, 187600, {30, 9, 45},  545600, {137, "Power supply failure detected in primary power distribution unit"}, {237, "Replaced power supply unit to ensure continuous operation"}, 101000},

        {3, 197802, {21, 11, 55}, 545802, {123, "Hardware failure observed in I/O subsystem"}, {224, "Replaced faulty hardware components in I/O subsystem"}, 100801},

        {3, 158004, {15, 10, 50}, 545701, {122, "Leakage detected in hydraulic system"}, {222, "Fixed leakage to prevent system damage"}, 100400},

        {3, 168406, {01, 18, 00}, 589600,{117, "Input/output error observed in data transfer process"}, {217, "Resolved I/O issues to ensure data integrity"}, 100800}

    };

//structure array for the fourth production line

    struct productionLines record4[PRODUCTS] =

    {

        {4, 123806, {19, 20, 50}, 579500, {137, "Power supply failure detected in primary power distribution unit"}, {237, "Replaced power supply unit to ensure continuous operation"}, 101000},

        {4, 143604, {05, 18, 40}, 579300, {125, "Power surge detected in primary power supply unit"}, {226, "Installed surge protection to mitigate power surges"}, 101101},

        {4, 153503, {12, 17, 35}, 579200, {134, "Hardware malfunction detected in critical subsystem"}, {234, "Repaired hardware components to restore functionality"}, 100700},

        {4, 143402, {23, 16, 30}, 579100, {127, "Voltage surge detected in power supply unit"}, {227, "Installed surge protector to prevent damage from surges"}, 100900},

        {4, 173907, {07, 21, 55}, 579600, {103,"Calibration drift detected in sensor readings"}, {203, "Recalibrated the equipment to ensure accurate measurements"}, 1000003},

        {4, 143705, {06, 19, 45}, 579400, {122, "Leakage detected in hydraulic system"}, {222, "Fixed leakage to prevent system damage"}, 100400},

        {4, 143200, {11, 14, 20}, 578900, {131, "Faulty battery detected in backup power system"}, {231, "Replaced battery with a new one to ensure uninterrupted power supply"}, 100400},

        {4, 194008, {24, 22, 00}, 579700, {139, "Memory allocation error reported during system startup"}, {239, "Optimized memory usage to prevent resource conflicts"}, 101200},

        {4, 143301, {17, 15, 25}, 579000, {132, "Sensor calibration error detected in environmental monitoring system"}, {232, "Recalibrated sensor to ensure accurate measurements"}, 100500},

        {4, 103109, {31, 23, 05}, 579800, {120, "Power failure detected in primary power supply unit"}, {220, "Restored power supply to prevent system downtime"}, 101101},

        {4, 143402, {27, 17, 35}, 579000, {135, "Software bug detected in control software"}, {235, "Fixed software bug to prevent system errors"}, 100800},

        {4, 143503, {15, 17, 35}, 579600, {134, "Hardware malfunction detected in critical subsystem"}, {234, "Repaired hardware components to restore functionality"}, 100700},

        {4, 113806, {16, 17, 35}, 579100, {124, "System overload detected in processing unit"}, {224, "Optimized system resources to prevent overload"}, 100600},

        {4, 173907, {02, 17, 35}, 579200, {137, "Power supply failure detected in primary power distribution unit"}, {237, "Replaced power supply unit to ensure continuous operation"}, 101000},

        {4, 103109, {18, 17, 35}, 579700, {101,"Faulty component detected in the main control unit"}, {201, "Replaced faulty component with a new and tested one"}, 1000001}

    };

I then printed them all using this layout for all 4:

//for loop to print off all the details for the first production line

    printf("\n\nReport 1:\n");

    for(i=0; i < PRODUCTS; i++)

    {//beginning of for loop

        printf("\nRecord %d", i + 1);

        printf("\nLine Code: %d", record1[i].lineCode);

        printf("\nBatch Code: %d", record1[i].batchCode);

        printf("\nBatch Date & Time : day %d, hour %d, minute %d", record1[i].DT.batchday, record1[i].DT.batchhour, record1[i].DT.batchminute);

        printf("\nProduct ID: %d", record1[i].productID);

        printf("\nIssue Code: %d", record1[i].issue.issueCode);

        printf("\nIssue Description: %s", record1[i].issue.issuedescription);

        printf("\nResolution Code: %d", record1[i].res.resolutionCode);

        printf("\nResolution Description: %s", record1[i].res.resolution\_description);

        printf("\nReporting Employee ID: %d", record1[i].employeeID);

        printf("\n");

    }//end of for loop

To combine my 4 arrays so I can use it in the merge and binary search (Task 1 & Task 3) I declared a new structure array at the top and added all my arrays in it using a for loop:

 struct productionLines Combinedrecords[PRODLINES];

for ( i = 0; i < PRODUCTS; i++)

    {//beginning of for loop

        Combinedrecords[i] = record1[i];

    }//end of for loop

    for ( i = 0; i < PRODUCTS; i++)

    {//beginning of for loop

        Combinedrecords[i + PRODUCTS] = record2[i];

    }//end of for loop

    for ( i = 0; i < PRODUCTS; i++)

    {//beginning of for loop

        Combinedrecords[i + 2 \* PRODUCTS ] = record3[i];

    }//end of for loop

    for ( i = 0; i < PRODUCTS; i++)

    {//beginning of for loop

        Combinedrecords[i +  3 \* PRODUCTS ] = record4[i];

    }//end of for loop

Lastly these are all the variables I declared in my main at the top:

//declaring variables

    int i;

    //for binary search

    int productID = 0;

    int issueCode = 0;

    //for linear search

    int no\_ofProducts = 0;

    struct productionLines Combinedrecords[PRODLINES];

This is what the outcome looks like when I print out the products and the records  
A screenshot of a computer

Description automatically generated

**TASK 1**

**WHAT I DID FOR THIS TASK:**

The algorithm that I chose for Task 1 was a Merge Sort

**WHY DID I CHOOSE THIS ALGORITHM:**

I chose this algorithm because its running time meets the criteria for the task and offers a stable O (N log(N)) time complexity which helps when storing large amounts of data. I used the merge sort for my sorting algorithm as the running time for this algorithm is O(N Log(N)) in its worst-case scenario. Its stability also ensures that the data is stored correctly.

**HOW DOES THE ALGORITHM MEET THE TIME COMPLEXITY:**

1. The running time of the dividing step is O (1) when splitting the array into 2 halves.
2. Sorting each half of the array has a running time of T(N/2) for each half, where N is the size of the array.
3. Merging the 2 halves has a running time of O (N) where N is the size of the array
4. The recursion part in the merge Sort function has a running time of log base 2 (N) since the algorithm keeps using recursion to split the arrays till there is only one element left in the array
5. When combining all these running times, you are left with a running time of O( N log(N)) which meets the requirement of the task.

**WHAT I DID TO MEET THE REQUIREMENTS FOR THIS TASK:**

I started off by creating my functions and calling them at the top:

//Task 1 Function calls

void merge(struct productionLines [], int low , int  mid , int high);//function to merge the halves together

void mergeSort( struct productionLines [], int , int );//function to sort the numbers in the mergesort algorithm

void printedSortedAlgorithm( struct productionLines [], int n );//function to print out the mergesort algorithm

I then called the mergeSort function and printedSortedAlgorithm function in my main:

// Task1 :Calling mergeSort for each record array

    mergeSort(Combinedrecords, 0, PRODLINES - 1);

    //Task 1: Printing sorted data from merge sort

    printedSortedAlgorithm(Combinedrecords, PRODLINES);

Here is my merge Function:

//TASK 1

void merge(struct productionLines records[], int low , int  mid , int high)//function to merge the halves together

{//beginning of merge function

    //declaring and initialising variables

    int n1 = mid - low + 1;//calculating the 2 sizes of the subarrays to be merged together

    int n2 = high - mid;//calculating the 2 sizes of the subarrays to be merged together

    int i = 0;

    int j = 0;

    int k = low;

    //creating temporary arrays to store the subarrays

    struct productionLines Left[n1], Right[n2];

    for (int i = 0; i < n1; i++)//for loop to copy data to the temporary left array

    {//beginning of for loop

        Left[i] = records[low + i];

    }//end of for loop

    for (int j = 0; j < n2; j++)//for loop to copy data to the temporary  right array

    {//beginning of inner for loop

        Right[j] = records[mid + 1 + j];

    }//end of inner for loop

    //merging the temporary arrays back into record array

    while (i < n1 && j < n2)

    {

        //comparing product ID, issue code, and batch date/time to determine the order of elements

        if  (Left[i].productID < Right[j].productID ||

            (Left[i].productID == Right[j].productID && Left[i].issue.issueCode < Right[j].issue.issueCode) ||

            (Left[i].productID == Right[j].productID && Left[i].issue.issueCode == Right[j].issue.issueCode &&

            (Left[i].DT.batchday < Right[j].DT.batchday ||(Left[i].DT.batchday == Right[j].DT.batchday &&

            (Left[i].DT.batchhour < Right[j].DT.batchhour ||(Left[i].DT.batchhour == Right[j].DT.batchhour &&Left[i].DT.batchminute < Right[j].DT.batchminute))))))

        {

            records[k++] = Left[i++];

        }

        else

        {

            records[k++] = Right[j++];

        }

    }

    while (i < n1)//copying the remaining elements of Left array

    {//beginning of while loop

        records[k++] = Left[i++];

    }//end of while loop

    while (j < n2)//copying the remaining elements of Right array

    {//beginning of while loop

        records[k++] = Right[j++];

    }//end of while loop

}//end of merge function

Then my mergeSort function:

void mergeSort( struct productionLines records[], int low, int high)//function to sort the numbers in the mergesort algorithm

{//beginning of mergeSort function

    if (low < high)//if statement to check if the array has more than 1 element

    {//beginning of if statement

        //calculating the middle index of the array

        int mid = low + (high - low) / 2;

        //calling the mergesort on the left half of the array using recursion

        mergeSort(records, low, mid);

        //calling the mergesort on the left half of the array using recursion

        mergeSort(records, mid + 1, high);

        //merging the sorted left and right halves together

        merge(records, low, mid , high);

    }//end of if statement

}//end of mergeSort function

Then lastly my printedSortedAlgorithm function to print the merge sort:

void printedSortedAlgorithm( struct productionLines records[], int n  )//function to print out the mergesort algorithm

{//beginning of printedSortedAlgorithm function

    printf("\n\n\n");

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

    printf("| %-10s | %-10s | %-7s | %7s    |\n", "Product ID", "Issue Code", "Date", "Time");

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

    for (int i = 0; i < n; i++)

    {

        printf("| %-10d | %-10d | %-7d | %02d:%02d      |\n",records[i].productID, records[i].issue.issueCode, records[i].DT.batchday, records[i].DT.batchday);

    }

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

}//end of printedSortedAlgorithm function

This part of the merge sort is the main part where it sorts the production line first by productID then in those duplicate productIDs, it sorts it by issue code and lastly, it sorts the duplicate issue code by date and time.

//merging the temporary arrays back into record array

    while (i < n1 && j < n2)

    {

        //comparing product ID, issue code, and batch date/time to determine the order of elements

        if  (Left[i].productID < Right[j].productID ||

            (Left[i].productID == Right[j].productID && Left[i].issue.issueCode < Right[j].issue.issueCode) ||

            (Left[i].productID == Right[j].productID && Left[i].issue.issueCode == Right[j].issue.issueCode &&

            (Left[i].DT.batchday < Right[j].DT.batchday ||(Left[i].DT.batchday == Right[j].DT.batchday &&

            (Left[i].DT.batchhour < Right[j].DT.batchhour ||(Left[i].DT.batchhour == Right[j].DT.batchhour &&Left[i].DT.batchminute < Right[j].DT.batchminute))))))

        {

            records[k++] = Left[i++];

        }

        else

        {

            records[k++] = Right[j++];

        }

    }

This is what my Task 1 looks like when its printed:

A screenshot of a computer

Description automatically generated

***PSEUDOCODE TASK1***

function merge( records[], low, mid, high)

n1 = mid - low + 1

n2 = high - mid

i = 0

j = 0

k = low

declare Left[n1], Right[n2]

for i = 0 to n1 -1 do

Left[i] = records[low + i]

for j = 0 to n2 - 1 do

Right[j] = records[mid + 1 + i]

Endfor

Endfor

while i < n1 AND j < n2

if Left[i].productID < Right[j].productID OR

(Left[i].productID == Right[j].productID AND Left[i].issue.issueCode < Right[j].issue.issueCode ) OR

(Left[i].productID == Right[j].productID && Left[i].issue.issueCode == Right[j].issue.issueCode AND

(Left[i].DT.batchday < Right[j].DT.batchday OR (Left[i].DT.batchday == Right[j].DT.batchday AND

(Left[i].DT.batchhour < Right[j].DT.batchhour OR ( Left[i].DT.batchhour == Right[j].DT.batchhour

AND Left[i].DT.batchminute < Right[j].DT.batchminute ))))) then

records[k++] = Left[i++]

Endif

Else

records[k++] = Right[j++]

Endelse

Endwhile

While i < n1

records[k++] = Left[i++]

EndWhile

While j < n2

records[k++] = Right[j++]

EndWhile

EndFunction merge

function mergeSort ( records[], low, high )

if low < high then

delare mid = low + (high - low) / 2

mergeSort(records , low, mid)

mergeSort(records , mid + 1, high)

merge(records, low, mid, high)

Endif

EndFunction merge

function printedSortedAlgorithm( records, n , lineCode)

print("\n\n\n")

print("+------------------------------------------------+\n")

print("| %-10s | %-10s | %-7s | %7s |\n", "Product ID", "Issue Code", "Date", "Time")

print("|------------------------------------------------|\n")

for i = 0 to n -1 do

printf("| %-10d | %-10d | %-7d | %02d:%02d |\n",records[i].productID, records[i].issue.issueCode, records[i].DT.batchday, records[i].DT.batchday)

EndFor

print("+------------------------------------------------+\n")

EndFunction printedSortedAlgorithm

**TASK 2**

**WHAT I DID FOR THIS TASK:**

The algorithm that I chose for Task 2 was a single Link List

**WHY DID I CHOOSE THIS ALGORITHM:**

I chose the algorithm because of its running time (O (N)) and how generating a report using the linked list can be done in a straightforward manner, making it ideal for the task requirement. It’s a nice and efficient way to store and perform operations required (like this task).

**HOW DOES THE ALGORITHM MEET THE RUNNING TIME:**

1. Creating a new node for each production line record involves time and allocating memory which makes the running time for this part 1.
2. Adding a new node to the end of the linked list means you have to find the last node in the list resulting in a running time of O(N) for this step.
3. Both these 2 steps combined result in a running time of O(N) for this algorithm which meets the criteria for this task. N is the number of production line records.

**WHAT I DID TO MEET THE REQUIREMENTS FOR THIS TASK:**

For task 2, I first started off by creating 3 functions and placing their function calls at the top. I also created a node structure for my linked list:

//Task 2 Function calls

int Node(struct node \*\*, struct productionLines data);//function to create a node

void createList(struct productionLines []);//function to link the nodes together in a linked list

void Printlinkedlist(struct node \*);//function to print out the linked list for each production line

//structure for a node

struct node

{

    struct productionLines values;//nested structure

    struct node \*link;//pointer for a linked list

};

I then called these createList function in my main:

//calling functions to create a linked list for each production line

    createList(record1);

    createList(record2);

    createList(record3);

    createList(record4);

Next I created the Node function to create my nodes:

//TASK 2

int Node(struct node \*\*head, struct productionLines values)// Function to create a new node and add it to the end of the linked list

{//beginning of Node function

    //here we are allocating memory for the new node

    struct node \*nextNode = malloc(sizeof(struct node));

    if (nextNode == NULL)

    {//beginning of if statement

        //if statement if there is not enough memory to allocate for the node

        printf("Failed to allocate memory\n");

        return -1;

    }//end of if statement

    //assigning data to the new node

    nextNode->values = values;

    //setting the link of the new node to NULL because it will be the last node in the list

    nextNode->link = NULL;

    if (\*head == NULL)//if statment for if the list is empty

    {//beginning of if statement

        //here we set the head to point to a new node

        \*head = nextNode;

    }//end of if statement

    else //if the list list isn't empty

    {//beginning of else statement

        //getting the list to go over and find the last node

        struct node \*temp = \*head;

        while (temp->link != NULL)

        {

            temp = temp->link;

        }

        //linking the last node to the new node

        temp->link = nextNode;

    }//end of else statement

    //here we return 0 to indicate the success of adding a new node to the linked list

    return 0;

}//end of Node function

My createList function to join my nodes together and making a linked list:

//function to create a linked list using the nodes that we created from the 4 production lines

void createList(struct productionLines records[])

{//beginning of createList function

    //initailising a pointer to the head of the linked list

    struct node \*head = NULL;

    //going through each product in the production line

    for (int i = 0; i < PRODUCTS; i++)

    {//beginning of for loop

        //calling Node function to create a new node from the data of the current production line

        Node(&head, records[i]);//passing the address of the head pointer so that we can modify it if needed

    }//end of for loop

    //printing the content of the linked list

    Printlinkedlist(head);

    // Free the memory allocated for the linked list

    struct node \*temp = head;//starting from the head node

    while (temp != NULL) //while loop to free the memory for each node

    {//beginning of while loop

        //store the link of the current node

        struct node \*next = temp->link;

        //free the memory of the current node

        free(temp);

        //move onto the next node

        temp = next;

    }//end of while loop

}//end of createList function

My printlinkedList function to print off my linked list:

void Printlinkedlist(struct node \*head)//function to print out the content of linked list

{//beginning of Printlinkedlist

    //initialising a temporary pointer to the head of the linked list

    struct node \*temp = head;

    printf("\n\n|Product ID\t|Line ID\t|Issue Codes\n");

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

    while (temp != NULL)//while loop to display the content for the linked list for each production line

    {//beginning of while loop

        printf("| %d\t| %d\t\t| %d\n", temp->values.productID, temp->values.lineCode, temp->values.issue.issueCode);

        //here we are moving temp to the next node in the linked list

        temp = temp->link;

    }//end of while loop

}//end of Printlinkedlist

***PSEUDOCODE TASK2***

function Node(head, values)

nextNode = allocate memory for a new node

if nextNode is NULL then

print("Failed to allocate memory")

return -1

Endif

nextNode->values = values

nextNode->link = NULL

if head is NULL then

head = nextNode

EndIf

else

temp = head

while temp->link is not NULL

temp = temp->link

EndWhile

temp->link = nextNode

Endelse

return 0

EndFunction Node

Function createList(records[])

Declare head = NULL

i = 0

for i = 0 to PRODUCTS - 1 do

Node(head, records[])

Endfor

Printlinkedlist(head)

temp = head

while temp is not NULL

next = temp->link

free(temp)

temp = next

Endwhile

EndFunction createList

Function Printlinkedlist(head)

temp = head

print("Product ID\tLine ID\tIssue Codes")

while temp is not NULL

print("temp->values.productID,\t\t temp->values.lineCode,\t\t temp->values.issue.issueCode")

temp = temp->link

Endwhile

EndFunction Printlinkedlist

***FLOWCHART FOR EACH FUNCTION IN TASK 2***

A diagram of a flowchart

Description automatically generated

A diagram of a flowchart

Description automatically generated

A diagram of a flowchart

Description automatically generated

This is what my task 2 looks like printed or displayed:

A screenshot of a computer screen

Description automatically generated

**TASK 3**

**WHAT I DID FOR THIS TASK:**

The algorithm that I chose for Task 3 was a Binary Search. When using the binary search, you must have a sorted array, and rather than using the structure arrays like normally for the other tasks, I used the sorted array from my merge sort in task 1 to complete this task

**WHY DID I CHOOSE THIS ALGORITHM:**

I chose this algorithm because its running time meets the criteria. It was also very easy to implement into my code, understand, and work with it. It also aids you in finding something quickly and since we want to find the first occurrence of a product with a specific issue code, it’s suitable.

**HOW DOES THE ALGORITHM MEET THE RUNNING TIME:**

1. The running time to initialise the left and right pointers is 1.
2. The binary search halves the array and compares the target ProductID and issue code with the middle index of the array. This comparison has a running time of 1.
3. The number of times the array is halved till the target is reached has a running time of O( Log N) where N is the size of the array.
4. The running times combined for this altogether is O 9Log N) which meets the criteria of this task.

**WHAT I DID TO MEET THE REQUIREMENTS FOR THIS TASK:**

First, I created 2 functions, one for the binary search itself and one for printing out the results.

//Task 3 Function calls

int binarySearch(struct productionLines [], int , int, int );//function to find the issue code and productID using binary search

void printSearchResult(struct productionLines [], int , int, int );//function to find the index

I then declared 2 variables in my main , one for productID and issueCode

//declaring variables

    int i;

    //for binary search

    int productID = 0;

    int issueCode = 0;

    //for linear search

    int no\_ofProducts = 0;

    struct productionLines Combinedrecords[PRODLINES];

Next I prompted the user to enter in the productID and the issue code they would like to search for in the production lines and called the printSearchResult function:

 //calling function for binary search to find the index of the issue code and productID

    printf("\nPlease enter in a product you would like to search for:\n");

    scanf("%d", &productID);

    printf("\nPlease enter in the issue code you would like to search for:\n");

    scanf("%d", &issueCode);

    printSearchResult(Combinedrecords, PRODLINES, productID, issueCode);

Then I created both of my functions:

//TASK 3

int binarySearch(struct productionLines record[], int size, int productID, int issueCode)//function to find the issue code and productID using binary search

{//beginning of binarySearch function

    //initialising and declaring variables for left and right pointers for the binary search

    int left = 0;

    int right = size - 1;

    while (left <= right)

    {//beginning of while loop

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

        if (record[middle].productID == productID &&  record[middle].issue.issueCode == issueCode)//checking if the product ID at the middle index matches the target product ID

        {//beginning of if statment

            //if found, return the index of the product ID

            return middle;

        }//end of if statment

        else if (record[middle].productID < productID || record[middle].productID == productID && record[middle].issue.issueCode < issueCode )

        {//beginning of else if statment

            //if the product ID at middle index is less than the target product ID, update the left pointer to search in the right half of the array

            left = middle + 1;

        }//end of  else if statment

        else

        {//beginning of else statment

            //if the product ID at middle index is greater than the target product ID, update the right pointer to search in the left half of the array

            right = middle - 1;

        }//end of else statment

    }//end of while loop

    //return -1 indicating failure if the productID isn't found in the array

    return -1;

}//end of binarySearch function

void printSearchResult(struct productionLines record[], int size, int productID, int issueCode)//function to find the index

{//beginning of printSearchResult function

    //calling the functionto find the index of the productID in the record array

    int index = binarySearch(record, size, productID, issueCode);

    if (index != -1)//check if the productID is found

    {//beginning of if statement

        //finding the issueCode for the productID if it's found

        issueCode = record[index].issue.issueCode;

        printf("\nEarliest occurrence of issue code %d for product ID %d found at index %d \n", issueCode, productID, index);

    }//end of if statement

    else //if productID isn't found

    {//beginning of else statement

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

    }//end of else statement

}//end of printSearchResult function

***PSEUDOCODE TASK3***

Function binarySearch(record, size, productID, issueCode)

left = 0

right = size - 1

while left <= right

delare middle = left + (right - left) / 2

if Record[middle].productID is productID AND record[middle].issue.issueCode is issueCode then

return middle

EndIf

else if Record[middle].productID < productID OR record[middle].productID == productID AND record[middle].issue.issueCode < issueCode then

left = middle + 1

EndElseif

else

right = middle - 1

EndElse

EndWhile

return -1

EndFunction binarySearch

Function printSearchResult( record[], size, productID, issueCode)

index = binarySearch(record, size, productID, issueCode)

if index is not -1 then

issueCode = record[index].issue.issueCode

print("\nEarliest occurrence of issue code %d for product ID %d found at index %d \n", issueCode, productID, index)

EndIf

else

print("Product ID %d not found ", productID)

EndElse

EndFunction printsearchResult

This is what my task 3 displays:

A screenshot of a computer screen

Description automatically generated

**TASK 4**

**WHAT I DID FOR THIS TASK:**

The algorithm that I chose for Task 4 was Linear Search.

**WHY DID I CHOOSE THIS ALGORITHM:**

I chose this data because its running time meets the criteria for this task and the linear search allows us to go through each piece of data one by one. This is good since we are looking to count the number of issues for each product, it helps us check each record exactly once while also making sure all the data is covered without missing one.

**HOW DOES THE ALGORITHM MEET THE RUNNING TIME:**

1. Declaring and initialising variables have a running time of 1.
2. The algorithm goes through each record and each product has a unique ID which is the worst-case scenario that leads to the running time of O(n) for this step where N is the comparisons
3. Combining the running times above leaves you with O(N) which meets the criteria for this task.

**WHAT I DID TO MEET THE REQUIREMENTS FOR THIS TASK:**

First, I declared the function at the top and declared a variable for the number of products in my main:

//Task 4 Function calls

void IssueCount(struct productionLines [], int );//Linear search to count the number of issues for all the productIDs

//for linear search

    int no\_ofProducts = 0;

Next I called the function 4 times for each production line in my main:

//calling funtions to count the number of issues for all the productIDs

    printf("\nIssue count for each Product ID in Production Line 1:\n");

    IssueCount(record1,PRODUCTS);

    printf("\n\nIssue count for each Product ID in Production Line 2:\n");

    IssueCount(record2,PRODUCTS);

    printf("\n\nIssue count for each Product ID in Production Line 3:\n");

    IssueCount(record3,PRODUCTS);

    printf("\n\nIssue count for each Product ID in Production Line 4:\n");

    IssueCount(record4,PRODUCTS);

Lastly, I created my IssueCount function:

//TASK 4

void IssueCount(struct productionLines records[], int no\_ofProducts)//Linear search to count the number of issues for all the productIDs

{//beginning of IssueCount function

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

    for(int i = 0; i < no\_ofProducts; i ++)//for loop to go through each record in the array

    {//beginning of for loop

        //get the productID of the current record

        int productID = records[i].productID;

        //initialise a counter for the total number of issues for the current productID

        int totalissues = 1;

        for(int j = i + 1; j < no\_ofProducts; j++)//for loop to find the occurences of the current productID in the remaining records

        {//beginning of inner for loop

            if(productID == records[j].productID)//if a duplicate productID is found

            {//beginning of if statement

                //increment the total issues counter

                totalissues = totalissues + 1;

            }//end of if statement

        }//end of inner for loop

        printf("Product ID: %d | Total Issues: %d\n", productID, totalissues);

        //skiping to the next unique productID by incrementing the outer loop variable. This avoids redundant counting for consecutive occurrences of the same productID

        while( i + 1 < no\_ofProducts && productID == records[i + 1].productID)

        {//beginning of while loop

            i = i + 1;

        }//end of while loop

    }//end of for loop

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

}//end of IssueCount function

***PSEUDOCODE TASK4***

Function IssueCount( records[], no\_ofProducts)

i = 0

j = 0

for i = 0 to no\_ofProducts -1 do

delare productID = records[i].productID

declaretotalissues = 1

for j = i + 1 to no\_ofProducts - 1 do

if ProductID is records[j].productID)

Increment totalissues

EndIf

Endfor

printf("Product ID:", productID, "Total Issues:", totalissues)

while i + 1 < no\_ofProducts AND productID is records[i + 1].productID

Increment i

Endwhile

Endfor

EndFunction IssueCount

This is what my task 4 displays:

A screenshot of a computer

Description automatically generated