Algorithm design Project Report

Delivery Docket for Car Parts Firm

By: Isobel Bloomer

C24303713

TU856

April 24, 2025

Problem Overview:

The company needs an automated system to generate a delivery docket for each van that sorts the products by weight and then merges each sorted list into one single sorted list. There also must be a user interface to search for the earliest occurrence of a product with a particular weight. Additionally, it then must provide a report which summarises the number of products included in the delivery for all vans.

Objective:

The goal of the system is to process data from the four production lines, sort them according to weight and then merge them all into a single sorted list to generate a final dispatch list.

Task 1: Pre-process and sort products by weight

This task involves reading in the data from 4 separate files, storing the data in a structure and then sorting each product by weight from lightest to heaviest.

For this task I decided to use the Merge Sort algorithm. I decided this algorithm would be most appropriate as the sorting is done efficiently with a time complexity of O(NLog(N)) which was one of the design requirements for this task.

Each production line is stored in a separate file which are opened using fopen() and the lines are read using fgets(). Each line is parsed into a structure using sscanf(). The data from each file is stored in separate arrays.

Each production array is sorted using Merge Sort which recursively splits the array, sorts each half and then merges them while comparing product weights. This makes sure the products from each line are sorted by weight in ascending order.

Each array Is then printed using a function. The output shows the product ID and the corresponding weight.

Test plan for task 1;

- I tested valid input files by reading well formatted files with 12 entries. 4 text files were inputted containing 12 valid product records each. the expected output is each production line array is filled and sorted in ascending order by weight
- I tested the sorting accuracy by printing out the sorted lists and checking whether they were correct.

Task 2: Merging Production Files into One Dispatch List

The goal of this task is to merge the four sorted production line arrays into one single dispatch list, ensuring the merged list is also sorted based on product weight.

To achieve this I implemented a circular queue for each production line. This way the merging process dynamically pulls the lightest available product across all lines.

This approach ensures that the merged dispatch list is still sorted according to weight while still meeting the time complexity requirement of (O(N)).

Firstly four circular queue structures are initialised one for each production line. Each queue holds up to 12 products and supports enqueue, dequeue and is_empty operations.

Then each sorted array from Task 1 is prepared for merging, by being enqueued into its respective queue.

In a loop the algorithm then repeatedly checks the front of each non empty queue to find the lightest product. This product is copied into the dispatch list array and dequeued. This process continues until all queues are empty meaning all of the products have been merged.

The merged dispatch list is printed displaying all the products in a single list sorted from lightest to heaviest.

Test plan for task 2:

- I tested what would happen if some queues are empty. If production lines have no items and one or more production arrays are empty the dispatch list includes only items from non empty lines. This ensures merging logic handles missing input lines correctly
- To test dispatch count accuracy and assure it reflects total items I input mixed length production lines. the purpose is to validate count tracking during the merging process.

Task 3: Search for the Earliest Occurrence of a Product by Weight

This task requires providing a user interface to allow the user to search for a product in the final dispatch list by entering a specific weight. This program uses a Binary Search algorithm

which is good for searching sorted arrays and it also has a low time complexity as a running time of O(Log(N)) is required.

The program searches through the dispatch list to find the first occurrence of a product which matches the weight once the user has entered it. If the product is found, the details are displayed. Otherwise a message is shows to tell the user the product doesn't exist.

The program starts by prompting the user to enter a weight value using scanf(). It then calls the binary search function which searches through the dispatch list array using the binary search algorithm. This algorithms checks the middle element and compares its weight to the users input. If it matches the index is returned, if not it recursively searches either the left or the right half. This function returns the first matching element it finds.

If a product is found its ID and weight are printed. If not a message is printed indicating no match was found.

Test plan for task 3:

- If the product is found the user has inputted a known weight from one of the products in the list. the index of the first occurrence and the correct product details are printed. This confirms binary search finds and returns the correct result.
- To test what happens if multiple items have the same weight I input a dispatch list with repeated weights. binary search returns the index of only the first occurrence and this shows that the search logic returns the earliest match.

Task 4: Summary Report of the Number of Products in the Delivery

This task provides a summary report of the final dispatch list after all merging and sorting operations have been completed. The goal is to inform the user of how many total products are in the final delivery.

This gives an overview of the delivery output and confirms the merging and processing of data has been handled correctly.

The function summarise_delivery() takes in the dispatch list array and the count of the number of items in that list.

A message is printed showing the total number of products in the dispatch list. This works because the count was already calculated when merging the production lines into one list.

Test plan for task 4:

- I tested the output with a full dispatch list and it output 44.
- I then tested the output with an empty dispatch list and it output 0

- I would then test large scale delivery summary to check the performance of the program.

Pseudocode for Task 1:

PROGRAM: Read File

```
OPEN file WITH filename
IF file NOT FOUND THEN
    PRINT “Error opening File”
    RETURN 0
END IF

DECLARE line[] AS STRING
count = 0
// Read each line and parse it into the item array
WHILE (fgets(line, sizeof(line), file) && count < MAX_ISSUES)
    DECLARE issue AS product
    //Parse line into issue properties
    READ line INTO issue (lineCode, batchCode, day, hour, minute, productID,
    productName, weight, price, targetEngineCode, binNumber)
    item[count] = issue
    count++
END WHILE
CLOSE file
RETURN count
```

END

PROGRAM: merge

```
leftLength = mid – left + 1
rightLength = right – mid


DECLARE leftArr[leftLength], rightArr[rightLength]
//Copy data into temp arrays
FOR i FROM 0 TO leftLength – 1
        leftArr[i] = arr[left + i]
END FOR
FOR j FROM 0 TO rightLength – 1
        rightArr[j] = arr[mid + 1 + j]
END FOR
i = 0
j = 0
k = left
//Merge arrays back into arr[]
WHILE I < leftLength AND j < rightLength
        IF leftArr[i].weight < rightArr[j].weight THEN
        arr[k] = leftArr[i]
        i++
        ELSE
        arr[k] = rightArr[j]
        j++
        END IF
        k++
END WHILE
//Copy remaining elements from leftArr into arr[]
WHILE I < leftLength
```

```
            arr[k] = leftArr[i]
            i++
            k++
        END WHILE
END
```

```
PROGRAM: FUNCTION merge_sort
        IF left < right THEN
            mid = left + (right – left) / 2
            CALL merge_sort(arr, left, mid)
            CALL merge_sort(arr, mid + 1, right)
            CALL merge(arr, left, mid, right)
        END IF
END
```

```
PROGRAM: FUNCTION print ptoduction line
        PRINT “Product id | Product Weight”
        FOR i FROM 0 TO 11
            PRINT arr[i].productionID, arr[i].weight
        END FOR
END
```

Pseudocode for Task 2:

PROGRAM: MergeQueuesToDispatchList

```
// initialise circular queues for each production line
circular_queue q1, q2, q3, q4
CALL init_queue(q1)
CALL init_queue(q2)
CALL init_queue(q3)
CALL init_queue(q4)

// load each production line into its corresponding queue
FOR I = 0 TO size1 – 1
        CALL enqueue(q1, line1[i]
END FOR
FOR I = 0 TO size2 – 1
        CALL enqueue(q2, line2[i]
END FOR
FOR I = 0 TO size3 – 1
        CALL enqueue(q3, line3[i]
END FOR
FOR I = 0 TO size4 – 1
        CALL enqueue(q4, line4[i]
END FOR
index = 0
# Process queues until all are empty
While (NotEmpty(q1) OR NotEmpty(q2) OR NotEmpty(q3) OR NotEmpty(q4)):
        //Find the product with the smallest weight
        minProduct = NULL
        IF NotEmpty(q1):
                minProduct = q1.head
```

```
END IF

IF NotEmpty(q2) AND (minProduct == NULL OR q2.head.weight < minProduct.weight):
        minProduct = q2.head
END IF

IF NotEmpty(q3) AND (minProduct == NULL OR q3.head.weight < minProduct.weight):
        minProduct = q3.head
END IF

IF NotEmpty(q4) AND (minProduct == NULL OR q4.head.weight < minProduct.weight):
        minProduct = q4.head
END IF

//Copy min product into the merged list
IF minProduct != NULL:
        merged[index] = minProduct
        index = index + 1

        //Remove the product from the corresponding queue
        IF minProduct is from q1:
                Dequeue(q1)
        ELSE IF minProduct is from q2:
                Dequeue(q2)
        ELSE IF minProduct is from q3:
                Dequeue(q3)
        ELSE IF minProduct is from q4:
                Dequeue(q4)
```

```
                    END IF
                END IF
            END WHILE
            # Set the merged size
            merged_size = index
END
```

PROGRAM: initqueue

```
    q.head = -1
    q.tail = -1
    q.queue_size = 0
END
```

PROGRAM: IsEmpty

```
    RETURN q.queue_size == 0
END
```

PROGRAM: Enqueue

```
    IF q.queue_size == MAXSIZE
        RETURN
    END IF
    IF isEmpty(q)
        q.head = 0
        q.tail = 0
    ELSE
        q.tail = (q.tail + 1) % MAXSIZE
    END ELSE
```

```
        q.elements[q.tail] = value
        q.queue_size = q.queue_size + 1
END
```

PROGRAM: Dequeue

```
        IF isEmpty(q)
                RETURN EmptyProduct()
        END IF
        value = q.elements[q.head]
        IF q.head == q.tail
                q.head = -1
                q.tail = -1
        ELSE
                q.head = (q.head + 1) % MAXSIZE
        END ELSE
        q.queue_size = q.queue_size – 1
        RETURN value
END
```

Pseudocode for task 3:

PROGRAM: Provide a user interface to search for the earliest occurrence of a product with a particular weight

MAIN:

```
        entered_weight
        PRINT “Please enter the weight of the product”
        READ entered_weight
        first_occurence
```

```
    first_occurence = binary_search(dispatch_list, 0, 47, entered_weight)
    IF first_occurence >= 0
        PRINT "Product ID, Product Name, Product Weight"
    ELSE
        PRINT "No product with that weight found"
    END ELSE
    RETURN 0
END
```

```
FUNCTION: BinarySearch
    mid
    IF right >= left
        mid = left + (right – left) / 2
        IF arr[mid].weight == weight
            RETURN mid
        ELSE IF arr[mid].weight > weight
            RETURN binary_search(arr, left, mid – 1, weight)
        ELSE
            RETURN binary_search(arr, mid + 1, right, weight)
        END ELSE
    END IF
    RETURN -1
END
```

Pseudocode for Task 4

PROGRAM: provide a report which summarises the number of products included in the delivery for all vans

MAIN:

CALL summarise_delivery(dispatch_list, dispatch_count)

RETURN 0

END MAIN

FUNCTION: SummariseDelivery

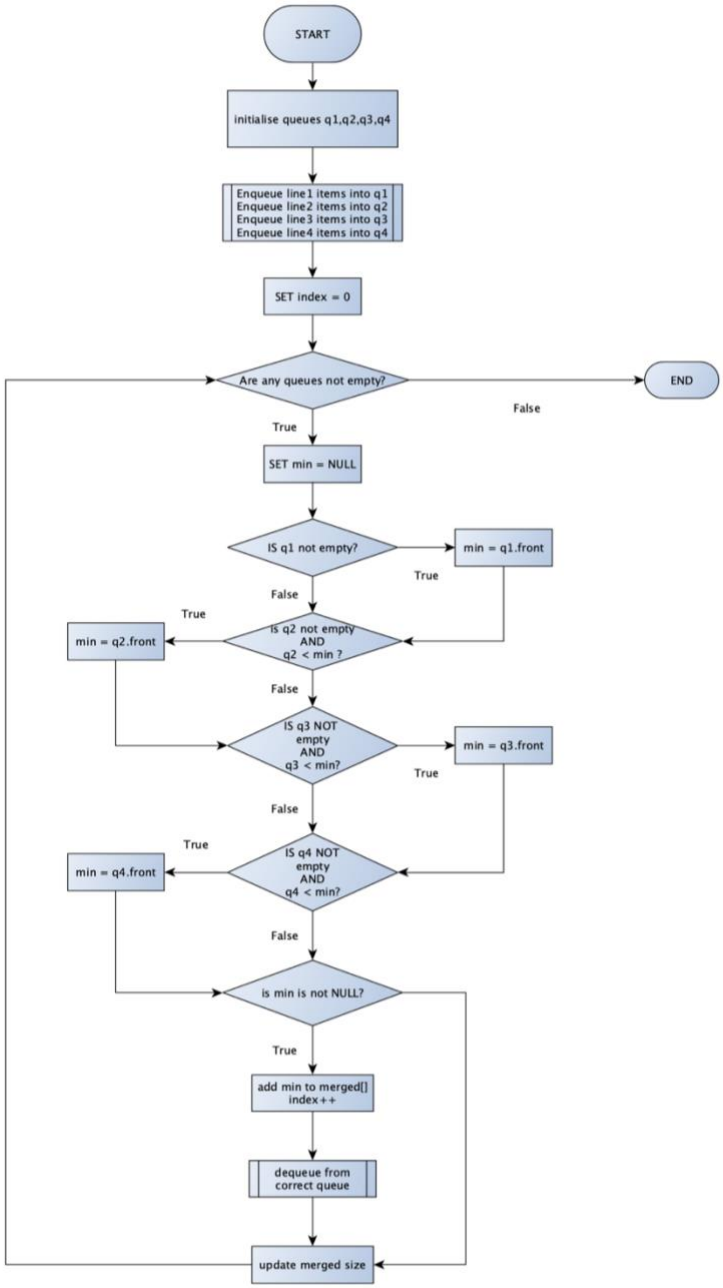
PRINT “Delivery summary report”

PRINT “Total number of products delivered: “ count

END FUNCTION

Flow Chart for task 2:

I made this in yed



C Code for each task

Within the code I commented which programs were from which task for readability

C Code for task 1:

```
/*
Program: Sorting each file of production data by product weight using merge sort
Author: Isobel Bloomer
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_LINE_LENGTH 225
#define MAX_ISSUES 12

//Structure templates
struct date
{
	int day;
	int hour;
	int minute;
};

typedef struct
{
	int lineCode;
```

```
        int batchCode;
        struct date BDT;
        int productID;
        char productName[50];
        float weight;
        float price;
        char targetEngineCode[20];
        int binNumber;
} product;


//Functions
int read_file(const char *filename, product productionLine[]) ;

void merge(product arr[], int left, int mid, int right) ;

void merge_sort(product arr[], int left, int right) ;

void print_production_line(product arr[]) ;

int main()
{
        // pointer array for each file
        const char *files[4] = {"line1.txt", "line2.txt", "line3.txt", "line4.txt"};

        //One array for each production line
        product production_line1[12], production_line2[12], production_line3[12],
        production_line4[12];
        int count1, count2, count3, count4;
```

```
    //Read data from each file into its array
    count1 = read_file(files[0], production_line1);
    count2 = read_file(files[1], production_line2);
    count3 = read_file(files[2], production_line3);
    count4 = read_file(files[3], production_line4);

    //Sort each array by weight
    merge_sort(production_line1, 0, count1 - 1);
    merge_sort(production_line2, 0, count2 - 1);
    merge_sort(production_line3, 0, count3 - 1);
    merge_sort(production_line4, 0, count4 - 1);

    // Print results for each production Line
    printf("\n--- Sorted Production Line 1 ---\n");
    print_production_line(production_line1);

    printf("\n--- Sorted Production Line 2 ---\n");
    print_production_line(production_line2);

    printf("\n--- Sorted Production Line 3 ---\n");
    print_production_line(production_line3);

    printf("\n--- Sorted Production Line 4 ---\n");
    print_production_line(production_line4);

    return 0;
}
```

```
//Function to read each file and store it in the arrays
int read_file(const char *filename, product item[])
{
	// file pointer
	FILE *file = fopen(filename, "r");

	If(!file)
	{
		perror("\nError opening file\n");
		return 0;
	}

	char line[MAX_LINE_LENGTH] ;
	int count = 0 ;

	//read each line into array
	while(fgets(line, sizeof(line), file) && count < MAX_ISSUES)
	{
		product issue;
		// parse line using scanf

		sscanf(line, "%d,%d,%d,%d,%d,%d,%49[^,],%f,%f,%19[^,],%d",
		&issue.lineCode, &issue.batchCode, &issue.BDT.day, &issue.BDT.hour,
		&issue.BDT.minute, &issue.productID, &issue.productName, &issue.weight,
		&issue.price, &issue.targetEngineCode, &issue.binNumber);

		item[count++] = issue ;
	}
```

```
        //close the file
        fclose(file);
        return count;



}

// function to sort arr of production items
void merge_sort(product arr[], int left, int right)
{
        if(left < right)
        {
                //initialising middle of array
                int mid = left + (right - left) / 2 ;
                merge_sort(arr, left, mid);
                merge_sort(arr, mid + 1, right);
                merge(arr, left, mid, right) ;
        } // end if
}

//Function to merge and sort the arrays
void merge(product arr[], int left, int mid, int right)
{
        int i, j, k;
        int leftLength = mid - left + 1; //Length of left side of the array
        int rightLength = right - mid ; //Length of right side of the array

        // temp arrays
```

```
product leftArr[leftLength], rightArr[rightLength] ;

//copying data to temp arrays
for(i = 0; i < leftLength; i++)
{
      leftArr[i] = arr[left + i] ;
}
for(j = 0; j < rightLength; j++)
{
      rightArr[j]  = arr[mid + 1 + j] ;
}

//Initialise indexes for arrays
i = 0;
j = 0;

//Initialise index for merged array
k = left;

// Merge the temp arrays back
while(i < leftLength && j < rightLength)
{
          // if left weught is then put it into arr
          if(leftArr[i].weight < rightArr[j].weight)
          {
          arr[k] = leftArr[i] ;
                    i++ ;
          }
```

```
            // Otherwise put the element from the right into arr
            else
            {
            arr[k] = rightArr[j] ;
            j++ ;
            }
            //move onto next part of array
            k++ ;
        }

        //Copy remaining elements from the right temp array into arr
        while(i < leftLength)
        {
            arr[k] = leftArr[i] ;
            i++ ;
            k++ ;
        }

        // Copy the remaining elements from the left temp array into arr
        while(j < rightLength)
        {
            arr[k] = rightArr[j] ;
            j++ ;
            k++ ;
        }
}

//Function to print production lines
```

```
void print_production_line(product arr[])
{
	printf("\n|Product id| Product weight\n");
	for(int i = 0; i < 12; i++)
	{
		printf("|%d	|%f \n", arr[i].productID, arr[i].weight);
	}

}
```

C Code for Task 2:

```
/*
Program: Merge the files into a single dispatch list
Author: Isobel Bloomer
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_LINE_LENGTH 225
#define MAX_ISSUES 12
#define MAXSIZE 12

//Structure templates

struct date
{
```

```
        int day;
        int hour;
        int minute;
};

typedef struct
{
        int lineCode;
        int batchCode;
        struct date BDT;
        int productID;
        char productName[50];
        float weight;
        float price;
        char targetEngineCode[20];
        int binNumber;
} product;

//circular queue implementation
typedef struct
{
        product elements[MAXSIZE];
        int head, tail, queue_size;
} circular_queue;

//Function to initialise empty queue
void init_queue(circular_queue *q) ;
```

```
// function to check if queue is empty
int is_empty(circular_queue *q) ;

//function to add a production item to the queue
void enqueue(circular_queue *q, product value);

//function to return a production item from the queue
product dequeue(circular_queue *q);

//merge 4 production lines into one dispatch list using circular queues
void merge_queues_to_dispatch_list(product line1[], int size1, product line2[], int size2, product line3[], int size3, product line4[], int size4, product merged[], int *mergedSize) ;

void print_production_line2(product arr[], int count);

int main()
{
    //TASK 1
    const char *files[4] = {"line1.txt", "line2.txt", "line3.txt", "line4.txt"};

    //One array for each production line
    product production_line1[12], production_line2[12], production_line3[12], production_line4[12];
    int count1, count2, count3, count4;

    //Read data from each file into its array
    count1 = read_file(files[0], production_line1);
    count2 = read_file(files[1], production_line2);
    count3 = read_file(files[2], production_line3);
```

```
count4 = read_file(files[3], production_line4);

//Sort each array by weight
merge_sort(production_line1, 0, count1 - 1);
merge_sort(production_line2, 0, count2 - 1);
merge_sort(production_line3, 0, count3 - 1);
merge_sort(production_line4, 0, count4 - 1);

// Print results for each production Line
printf("\n--- Sorted Production Line 1 ---\n");
        print_production_line(production_line1);

printf("\n--- Sorted Production Line 2 ---\n");
print_production_line(production_line2);

printf("\n--- Sorted Production Line 3 ---\n");
print_production_line(production_line3);

printf("\n--- Sorted Production Line 4 ---\n");
print_production_line(production_line4);

//TASK 2
// making an array to store all production items from all production lines
product dispatch_list[48] ;
int dispatch_count;

merge_queues_to_dispatch_list(production_line1, count1, production_line2, count2,
production_line3, count3, production_line4, count4, dispatch_list, &dispatch_count);
```

```
    printf("\n--- Final Dispatch List (Sorted by weight) ---\n");
    print_production_line2(dispatch_list, dispatch_count);

    return 0;
}

//TASK 1
int read_file(const char *filename, product item[])
{
    // file pointer
    FILE *file = fopen(filename, "r");

    if(!file)
    {
        perror("\nError opening file\n");
        return 0;
    }

    char line[MAX_LINE_LENGTH] ;
    int count = 0 ;

    //read each line into array
    while(fgets(line, sizeof(line), file) && count < MAX_ISSUES)
    {
        product issue;
        // parse line using scanf
```

```
            sscanf(line, "%d,%d,%d,%d,%d,%d,%49[^,],%f,%f,%19[^,],%d",
            &issue.lineCode, &issue.batchCode, &issue.BDT.day, &issue.BDT.hour,
            &issue.BDT.minute, &issue.productID, &issue.productName, &issue.weight,
            &issue.price, &issue.targetEngineCode, &issue.binNumber);

            item[count++] = issue ;

        }


        //Close file

        fclose(file);

        return count;


}


//TASK 1

// function to sort arr of production items

void merge_sort(product arr[], int left, int right)

{

        if(left < right)

        {

            //initialising middle of array

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

            merge_sort(arr, left, mid);

            merge_sort(arr, mid + 1, right);

            merge(arr, left, mid, right) ;

        } // end if

}


//TASK 1

void merge(product arr[], int left, int mid, int right)
```

```
{
        int i, j, k;
        int leftLength = mid - left + 1; //Length of left side of the array
        int rightLength = right - mid ; //Length of right side of the array

        // temp arrays
        product leftArr[leftLength], rightArr[rightLength] ;

         //copying data to temp arrays
        for(i = 0; i < leftLength; i++)
        {
                leftArr[i] = arr[left + i] ;
        }
        for(j = 0; j < rightLength; j++)
        {
                rightArr[j]  = arr[mid + 1 + j] ;
        }

        //Initialise indexes for arrays
        i = 0;
        j = 0;

        //Initialise index for merged array
        k = left;

        // Merge the temp arrays back
        while(i < leftLength && j < rightLength)
        {
```

```
            // if left weught is then put it into arr
            if(leftArr[i].weight < rightArr[j].weight)
            {
                arr[k] = leftArr[i] ;
                i++ ;
            }
            // Otherwise put the element from the right into arr
            else
            {
                arr[k] = rightArr[j] ;
                j++ ;
            }
            //move onto next part of array
            k++ ;
}

//Copy remaining elements from the right temp array into arr
while(i < leftLength)
{
    arr[k] = leftArr[i] ;
    i++ ;
    k++ ;
}

// Copy the remaining elements from the left temp array into arr
while(j < rightLength)
{
    arr[k] = rightArr[j] ;
```

```
        j++ ;
        k++ ;
    }
}

//TASK 1
//Function to print production lines
void print_production_line(product arr[])
{
        printf("\n|Product id| Product weight\n");
        for(int i = 0; i < 12; i++)
        {
                printf("|%d      |%f \n", arr[i].productID, arr[i].weight);
        }

}End

//TASK 2
void merge_queues_to_dispatch_list(product line1[], int size1, product line2[], int size2, product line3[], int size3, product line4[], int size4, product merged[], int *merged_size)
{
        //Initialise circular queues for each production line

        circular_queue q1, q2, q3, q4;

        init_queue(&q1);
        init_queue(&q2);
        init_queue(&q3);
```

```
init_queue(&q4);

// Load each production line into its queue
for(int i = 0; i < size1; i++)
{
	enqueue(&q1, line1[i]);
}
for(int i = 0; i < size2; i++)
{
	enqueue(&q2, line2[i]);
}
for(int i = 0; i < size3; i++)
{
	enqueue(&q3, line3[i]);
}
for(int i = 0; i < size4; i++)
{
	enqueue(&q4, line4[i]);
}

int index = 0;

while(!is_empty(&q1) || !is_empty(&q2) || !is_empty(&q3) || !is_empty(&q4))
{
	product *min = NULL;

	if(!is_empty(&q1))
	{
```

```
        min = &q1.elements[q1.head] ;
}
if(!is_empty(&q2) && (!min || q2.elements[q2.head].weight < min -> weight))
{
        min = &q2.elements[q2.head];
}
if(!is_empty(&q3) && (!min || q3.elements[q3.head].weight < min -> weight))
{
        min = &q3.elements[q3.head];
}
if(!is_empty(&q4) && (!min || q4.elements[q4.head].weight < min -> weight))
{
        min = &q4.elements[q4.head];
}

if(min)
{
        //Copy min item into merged list
        merged[index++] = *min;

        //Remove from the correct queue
        if(min == &q1.elements[q1.head])
        {
                dequeue(&q1);
        }
        else if(min == &q2.elements[q2.head])
        {
                dequeue(&q2);
```

```
                        }
                        else if(min == &q3.elements[q3.head])
                        {
                                dequeue(&q3);
                        }
                        else if(min == &q4.elements[q4.head])
                        {
                                dequeue(&q4);
                        }
                }

                *merged_size* = index;
        }

}

//TASK 2
//Function to initialise empty queue
void init_queue(circular_queue *q)
{
        q -> head = -1;
        q -> tail = -1;
        q -> queue_size = 0;
}

//TASK 2
//Function to check if queue is empty
int is_empty(circular_queue *q)
```

```
{
	return(q -> queue_size == 0);
}

//TASK 2
//Function to add a production item to the queue
void enqueue(circular_queue *q, product value)
{
	if(q -> queue_size == MAXSIZE)
	{
		return; // queue full
	}

	if(isEmpty(q))
	{
		q -> head = 0;
		q -> tail = 0;
	}
	else
	{
		q -> tail = (q -> tail + 1) % MAXSIZE;
	}

	q -> elements[q -> tail] = value;
	q -> queue_size++;
}
```

```
//TASK 2
//Function to return a production item from the queue
product dequeue(circular_queue *q)
{
	product empty = {0};

	if(is_empty(q))
	{
		return empty;
	}

	product value = q -> elements[q -> head];

	if(q -> head == q -> tail)
	{
		q -> head = -1;
		q -> tail = -1;
	}
	else
	{
		q -> head = (q -> head + 1) % MAXSIZE;
	}

	q -> queue_size-- ;
	return value;
}
```

```
//Task2
void print_production_line2(product arr[], int count)
{
	printf("\n|Product id| Product weight\n");
	for(int i = 0; i < count; i++)
	{
		printf("|%-10d	|%-13.2f \n", arr[i].productID, arr[i].weight);
	}
}
```

C Code for Task 3:

```
/*
Program: Provide a user interface to search for the earliest occurence of a product with a particular weight
Author: Isobel Bloomer
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_LINE_LENGTH 225
#define MAX_ISSUES 12
#define MAXSIZE 12

//Structure templates

struct date
```

```
{
	int day;
	int hour;
	int minute;
};

typedef struct
{
	int lineCode;
	int batchCode;
	struct date BDT;
	int productID;
	char productName[50];
	float weight;
	float price;
	char targetEngineCode[20];
	int binNumber;
} product;

//circular queue implementation
typedef struct
{
	product elements[MAXSIZE];
	int head, tail, queue_size;
} circular_queue;

int read_file(const char *filename, product productionLine[]) ;
```

```
void merge(product arr[], int left, int mid, int right) ;

void merge_sort(product arr[], int left, int right) ;

void print_production_line(product arr[]) ;

//Function to initialise empty queue
void init_queue(circular_queue *q) ;

// function to check if queue is empty
int is_empty(circular_queue *q) ;

//function to add a production item to the queue
void enqueue(circular_queue *q, product value);

//function to return a production item from the queue
product dequeue(circular_queue *q);

//merge 4 production lines into one dispatch list using circular queues
void merge_queues_to_dispatch_list(product line1[], int size1, product line2[], int size2,
product line3[], int size3, product line4[], int size4, product merged[], int *mergedSize) ;

void print_production_line2(product arr[], int count);

//Function to find earliest occurence of a product with a particular weight
int binary_search(product arr[], int left, int right, float weight);

int main()
```

```
{
    //TASK 1
    const char *files[4] = {"line1.txt", "line2.txt", "line3.txt", "line4.txt"};

    //One array for each production line
    product production_line1[12], production_line2[12], production_line3[12], production_line4[12];
    int count1, count2, count3, count4;

    //Read data from each file into its array
    count1 = read_file(files[0], production_line1);
    count2 = read_file(files[1], production_line2);
    count3 = read_file(files[2], production_line3);
    count4 = read_file(files[3], production_line4);

    //Sort each array by weight
    merge_sort(production_line1, 0, count1 - 1);
    merge_sort(production_line2, 0, count2 - 1);
    merge_sort(production_line3, 0, count3 - 1);
    merge_sort(production_line4, 0, count4 - 1);

     // Print results for each production Line
    printf("\n--- Sorted Production Line 1 ---\n");
    print_production_line(production_line1);

    printf("\n--- Sorted Production Line 2 ---\n");
    print_production_line(production_line2);
```

```
printf("\n--- Sorted Production Line 3 ---\n");
print_production_line(production_line3);

printf("\n--- Sorted Production Line 4 ---\n");
print_production_line(production_line4);

//TASK 2
// making an array to store all production items from all production lines
product dispatch_list[48] ;
int dispatch_count;

merge_queues_to_dispatch_list(production_line1, count1, production_line2, count2, production_line3, count3, production_line4, count4, dispatch_list, &dispatch_count);

printf("\n--- Final Dispatch List (Sorted by weight) ---\n");
print_production_line2(dispatch_list, dispatch_count);

//TASK 3
// ask user to enter the weight
float entered_weight;
printf("\nPlease enter the weight of the product:");
scanf("%f", &entered_weight);

// initialise an integer for the position in the array where the earliest occurence of the entered weight is
int first_occurence;

//binary search to find first occurence
first_occurence = binary_search(dispatch_list, 0, 47, entered_weight);
```

```
        // print details of first occurence
        if(first_occurence >= 0)
        {
                printf("\n\n|Product ID |Product Weight");
                printf("\n|%d          |%f          ", dispatch_list[first_occurence].productID,
                dispatch_list[first_occurence].weight);
        }
        else
        {
                printf("\n\nNo product with weight %f found.\n", entered_weight);
        }

        return 0;
}

//TASK 1
int read_file(const char *filename, product item[])
{
        // file pointer
        FILE *file = fopen(filename, "r");

        if(!file)
        {
                perror("\nError opening file\n");
                return 0;
        }
```

```
        char line[MAX_LINE_LENGTH] ;
        int count = 0 ;

        //read each line into array
        while(fgets(line, sizeof(line), file) && count < MAX_ISSUES)
        {
                product issue;
                // parse line using scanf

                sscanf(line, "%d,%d,%d,%d,%d,%d,%49[^,],%f,%f,%19[^,],%d",
                &issue.lineCode, &issue.batchCode, &issue.BDT.day, &issue.BDT.hour,
                &issue.BDT.minute, &issue.productID, &issue.productName[50],
                &issue.weight, &issue.price, &issue.targetEngineCode[50],
                &issue.binNumber);
                item[count++] = issue ;
        }

        //Close file
        fclose(file);
        return count;

}

//TASK 1
// function to sort arr of production items
void merge_sort(product arr[], int left, int right)
{
        if(left < right)
        {
```

```
        //initialising middle of array
        int mid = left + (right - left) / 2 ;
        merge_sort(arr, left, mid);
        merge_sort(arr, mid + 1, right);
        merge(arr, left, mid, right) ;
        } // end if
}

//TASK 1
void merge(product arr[], int left, int mid, int right)
{
        int i, j, k;
        int leftLength = mid - left + 1; //Length of left side of the array
        int rightLength = right - mid ; //Length of right side of the array

        // temp arrays
        product leftArr[leftLength], rightArr[rightLength] ;

        //copying data to temp arrays
        for(i = 0; i < leftLength; i++)
        {
                leftArr[i] = arr[left + i] ;
        }
        for(j = 0; j < rightLength; j++)
        {
                rightArr[j]  = arr[mid + 1 + j] ;
        }
```

```
//Initialise indexes for arrays
i = 0;
j = 0;

//Initialise index for merged array
k = left;

// Merge the temp arrays back
while(i < leftLength && j < rightLength)
{
        // if left weught is then put it into arr
        if(leftArr[i].weight < rightArr[j].weight)
        {
                arr[k] = leftArr[i] ;
                i++ ;
        }
        // Otherwise put the element from the right into arr
        else
        {
                arr[k] = rightArr[j] ;
                j++ ;
        }
        //move onto next part of array
        k++ ;
}

//Copy remaining elements from the right temp array into arr
while(i < leftLength)
```

```
        {
                arr[k] = leftArr[i] ;
                i++ ;
                k++ ;
        }

        // Copy the remaining elements from the left temp array into arr
        while(j < rightLength)
        {
                arr[k] = rightArr[j] ;
                j++ ;
                k++ ;
        }
}

//TASK 1
//Function to print production lines
void print_production_line(product arr[])
{
        printf("\n|Product id| Product weight\n");
        for(int i = 0; i < 12; i++)
        {
                printf("|%d      |%.2f \n", arr[i].productID, arr[i].weight);
        }

}

//TASK 2
```

```
void merge_queues_to_dispatch_list(product line1[], int size1, product line2[], int size2, product line3[], int size3, product line4[], int size4, product merged[], int *merged_size)
{
    //Initialise circular queues for each production line

    circular_queue q1, q2, q3, q4;

    init_queue(&q1);
    init_queue(&q2);
    init_queue(&q3);
    init_queue(&q4);

    // Load each production line into its queue
    for(int i = 0; i < size1; i++)
    {
        enqueue(&q1, line1[i]);
    }
    for(int i = 0; i < size2; i++)
    {
        enqueue(&q2, line2[i]);
    }
    for(int i = 0; i < size3; i++)
    {
        enqueue(&q3, line3[i]);
    }
    for(int i = 0; i < size4; i++)
    {
        enqueue(&q4, line4[i]);
```

```
}

int index = 0;

while(!is_empty(&q1) || !is_empty(&q2) || !is_empty(&q3) || !is_empty(&q4))
{
	product *min = NULL;

	if(!is_empty(&q1))
	{
		min = &q1.elements[q1.head] ;
	}
	if(!is_empty(&q2) && (!min || q2.elements[q2.head].weight < min -> weight))
	{
		min = &q2.elements[q2.head];
	}
	if(!is_empty(&q3) && (!min || q3.elements[q3.head].weight < min -> weight))
	{
		min = &q3.elements[q3.head];
	}
	if(!is_empty(&q4) && (!min || q4.elements[q4.head].weight < min -> weight))
	{
		min = &q4.elements[q4.head];
	}

	if(min)
	{
		//Copy min item into merged list
```

```
                    merged[index++] = *min;

                    //Remove from the correct queue
                    if(min == &q1.elements[q1.head])
                    {
                        dequeue(&q1);
                    }
                    else if(min == &q2.elements[q2.head])
                    {
                        dequeue(&q2);
                    }
                    else if(min == &q3.elements[q3.head])
                    {
                        dequeue(&q3);
                    }
                        else if(min == &q4.elements[q4.head])
                    {
                        dequeue(&q4);
                    }
                }

                *merged_size = index;
        }

}

//TASK 2
//Function to initialise empty queue
```

```
void init_queue(circular_queue *q)
{
	q -> head = -1;
	q -> tail = -1;
	q -> queue_size = 0;
}

//TASK 2
//Function to check if queue is empty
int is_empty(circular_queue *q)
{
	return(q -> queue_size == 0);
}

//TASK 2
//Function to add a production item to the queue
void enqueue(circular_queue *q, product value)
{
	if(q -> queue_size == MAXSIZE)
	{
		return; // queue full
	}

	if(is_empty(q))
	{
		q -> head = 0;
		q -> tail = 0;
	}
```

```
        else
        {
                q -> tail = (q -> tail + 1) % MAXSIZE;
        }

        q -> elements[q -> tail] = value;
        q -> queue_size++;
}

//TASK 2
//Function to return a production item from the queue
product dequeue(circular_queue *q)
{
        product empty = {0};

        if(is_empty(q))
        {
                return empty;
        }

        product value = q -> elements[q -> head];

        if(q -> head == q -> tail)
        {
                q -> head = -1;
                q -> tail = -1;
        }
        else
```

```
        {
                q -> head = (q -> head + 1) % MAXSIZE;
        }

        q -> queue_size-- ;
        return value;
}

//Task2
void print_production_line2(product arr[], int count)
{
        printf("\n|Product id| Product weight\n");
        for(int i = 0; i < count; i++)
        {
                printf("|%-10d      |%-13.2f \n", arr[i].productID, arr[i].weight);
        }
}

//TASK 3
//Function to find earliest occurence of a product with a particular weight
int binary_search(product arr[], int left, int right, float weight)
{
        int mid;

        if(right >= left)
        {
                mid = left + (right - left) / 2;
```

```
            //if value is found
            if(arr[mid].weight == weight)
            {
                return mid;
            }
            else if(arr[mid].weight > weight)
            {
                //search to left
                return binary_search(arr, left, mid - 1, weight);
            }
            else
            {
                //search to right
                return binary_search(arr, mid + 1, right, weight);
            }
        }

        return -1;
}
```

C Code for Task 4:

```
/*
Program: provide a report which summarises the number of products included in the delivery for all vans
Author: Isobel Bloomer
*/
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_LINE_LENGTH 225
#define MAX_ISSUES 12
#define MAXSIZE 12

//Structure templates

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

typedef struct
{
	int lineCode;
	int batchCode;
	struct date BDT;
	int productID;
	char productName[50];
	float weight;
	float price;
	char targetEngineCode[20];
```

```
        int binNumber;
} product;

//circular queue implementation
typedef struct
{
        product elements[MAXSIZE];
        int head, tail, queue_size;
} circular_queue;

int read_file(const char *filename, product productionLine[]) ;

void merge(product arr[], int left, int mid, int right) ;

void merge_sort(product arr[], int left, int right) ;

void print_production_line(product arr[]) ;

//Function to initialise empty queue
void init_queue(circular_queue *q) ;

// function to check if queue is empty
int is_empty(circular_queue *q) ;

//function to add a production item to the queue
void enqueue(circular_queue *q, product value);

//function to return a production item from the queue
```

```
product dequeue(circular_queue *q);

//merge 4 production lines into one dispatch list using circular queues
void merge_queues_to_dispatch_list(product line1[], int size1, product line2[], int size2, product line3[], int size3, product line4[], int size4, product merged[], int *mergedSize) ;

void print_production_line2(product arr[], int count);

//Function to find earliest occurence of a product with a particular weight
int binary_search(product arr[], int left, int right, float weight);

//Task 4
//Function to summarise the number of products in the delivery
void summarise_delivery(product dispatch_list[], int count);


int main()
{
    //TASK 1
    const char *files[4] = {"line1.txt", "line2.txt", "line3.txt", "line4.txt"};

    //One array for each production line
    product production_line1[12], production_line2[12], production_line3[12], production_line4[12];
    int count1, count2, count3, count4;

    //Read data from each file into its array
    count1 = read_file(files[0], production_line1);
    count2 = read_file(files[1], production_line2);
```

```
count3 = read_file(files[2], production_line3);
count4 = read_file(files[3], production_line4);

//Sort each array by weight
merge_sort(production_line1, 0, count1 - 1);
merge_sort(production_line2, 0, count2 - 1);
merge_sort(production_line3, 0, count3 - 1);
merge_sort(production_line4, 0, count4 - 1);

 // Print results for each production Line
printf("\n--- Sorted Production Line 1 ---\n");
print_production_line(production_line1);

printf("\n--- Sorted Production Line 2 ---\n");
print_production_line(production_line2);

printf("\n--- Sorted Production Line 3 ---\n");
print_production_line(production_line3);

printf("\n--- Sorted Production Line 4 ---\n");
print_production_line(production_line4);

//TASK 2
// making an array to store all production items from all production lines
product dispatch_list[48] ;
int dispatch_count;
```

```
merge_queues_to_dispatch_list(production_line1, count1, production_line2, count2,
production_line3, count3, production_line4, count4, dispatch_list, &dispatch_count);


printf("\n--- Final Dispatch List (Sorted by weight) ---\n");

print_production_line2(dispatch_list, dispatch_count);


//TASK 3

// ask user to enter the weight

float entered_weight;

printf("\nPlease enter the weight of the product:");

scanf("%f", &entered_weight);


// initialise an integer for the position in the array where the earliest occurence of the
entered weight is

int first_occurence;


//binary search to find first occurence

first_occurence = binary_search(dispatch_list, 0, 47, entered_weight);


 // print details of first occurence

if(first_occurence >= 0)

{

        printf("\n\n|Product ID |Product Weight");

        printf("\n|%d          |%f          ", dispatch_list[first_occurence].productID,
        dispatch_list[first_occurence].weight);

}

else

{

        printf("\n\nNo product with weight %f found.\n", entered_weight);
```

```
	}

	//TASK 4
	summarise_delivery(dispatch_list, dispatch_count);

	return 0;
}

//TASK 1
int read_file(const char *filename, product item[])
{
	// file pointer
	FILE *file = fopen(filename, "r");

	if(!file)
	{
		perror("\nError opening file\n");
		return 0;
	}

	char line[MAX_LINE_LENGTH] ;
	int count = 0 ;

	//read each line into array
	while(fgets(line, sizeof(line), file) && count < MAX_ISSUES)
	{
		product issue;
		// parse line using scanf
```

```
            sscanf(line, "%d,%d,%d,%d,%d,%d,%49[^,],%f,%f,%19[^,],%d",
            &issue.lineCode, &issue.batchCode, &issue.BDT.day, &issue.BDT.hour,
            &issue.BDT.minute, &issue.productID, &issue.productName[50],
            &issue.weight, &issue.price, &issue.targetEngineCode[50],
            &issue.binNumber);

            item[count++] = issue ;

        }


        //Close file

        fclose(file);

        return count;


}


//TASK 1

// function to sort arr of production items

void merge_sort(product arr[], int left, int right)

{

        if(left < right)

        {

        //initialising middle of array

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

        merge_sort(arr, left, mid);

        merge_sort(arr, mid + 1, right);

        merge(arr, left, mid, right) ;

        } // end if

}
```

```
//TASK 1
void merge(product arr[], int left, int mid, int right)
{
	int i, j, k;
	int leftLength = mid - left + 1; //Length of left side of the array
	int rightLength = right - mid ; //Length of right side of the array

	// temp arrays
	product leftArr[leftLength], rightArr[rightLength] ;

	//copying data to temp arrays
	for(i = 0; i < leftLength; i++)
	{
		leftArr[i] = arr[left + i] ;
	}
	for(j = 0; j < rightLength; j++)
	{
		rightArr[j]  = arr[mid + 1 + j] ;
	}

	//Initialise indexes for arrays
	i = 0;
	j = 0;

	//Initialise index for merged array
	k = left;

	// Merge the temp arrays back
```

```
while(i < leftLength && j < rightLength)
{
        // if left weught is then put it into arr
        if(leftArr[i].weight < rightArr[j].weight)
        {
                arr[k] = leftArr[i] ;
                i++ ;
        }
        // Otherwise put the element from the right into arr
        else
        {
                arr[k] = rightArr[j] ;
                j++ ;
        }
        //move onto next part of array
        k++ ;
}

//Copy remaining elements from the right temp array into arr
while(i < leftLength)
{
        arr[k] = leftArr[i] ;
        i++ ;
        k++ ;
}

// Copy the remaining elements from the left temp array into arr
while(j < rightLength)
```

```
        {
                arr[k] = rightArr[j] ;
                j++ ;
                k++ ;
        }
}

//TASK 1
//Function to print production lines
void print_production_line(product arr[])
{
        printf("\n|Product id| Product weight\n");
        for(int i = 0; i < 12; i++)
        {
                printf("|%d      |%.2f \n", arr[i].productID, arr[i].weight);
        }

}

//TASK 2
void merge_queues_to_dispatch_list(product line1[], int size1, product line2[], int size2, product line3[], int size3, product line4[], int size4, product merged[], int *merged_size)
{
        //Initialise circular queues for each production line

        circular_queue q1, q2, q3, q4;

        init_queue(&q1);
```

```
init_queue(&q2);
init_queue(&q3);
init_queue(&q4);

// Load each production line into its queue
for(int i = 0; i < size1; i++)
{
        enqueue(&q1, line1[i]);
}
for(int i = 0; i < size2; i++)
{
        enqueue(&q2, line2[i]);
}
for(int i = 0; i < size3; i++)
{
        enqueue(&q3, line3[i]);
}
for(int i = 0; i < size4; i++)
{
        enqueue(&q4, line4[i]);
}

int index = 0;

while(!is_empty(&q1) || !is_empty(&q2) || !is_empty(&q3) || !is_empty(&q4))
{
        product *min = NULL;
```

```
if(!is_empty(&q1))
{
        min = &q1.elements[q1.head] ;
}
if(!is_empty(&q2) && (!min || q2.elements[q2.head].weight < min -> weight))
{
        min = &q2.elements[q2.head];
}
if(!is_empty(&q3) && (!min || q3.elements[q3.head].weight < min -> weight))
{
        min = &q3.elements[q3.head];
}
if(!is_empty(&q4) && (!min || q4.elements[q4.head].weight < min -> weight))
{
        min = &q4.elements[q4.head];
}

if(min)
{
        //Copy min item into merged list
        merged[index++] = *min;

        //Remove from the correct queue
        if(min == &q1.elements[q1.head])
        {
                dequeue(&q1);
        }
        else if(min == &q2.elements[q2.head])
```

```
                {
                        dequeue(&q2);
                }
                else if(min == &q3.elements[q3.head])
                {
                        dequeue(&q3);
                }
                        else if(min == &q4.elements[q4.head])
                {
                        dequeue(&q4);
                }
        }

        *merged_size* = index;
    }

}

//TASK 2
//Function to initialise empty queue
void init_queue(circular_queue *q)
{
    q -> head = -1;
    q -> tail = -1;
    q -> queue_size = 0;
}

//TASK 2
```

```
//Function to check if queue is empty
int is_empty(circular_queue *q)
{
	return(q -> queue_size == 0);
}

//TASK 2
//Function to add a production item to the queue
void enqueue(circular_queue *q, product value)
{
	if(q -> queue_size == MAXSIZE)
	{
		return; // queue full
	}

	if(is_empty(q))
	{
		q -> head = 0;
		q -> tail = 0;
	}
	else
	{
		q -> tail = (q -> tail + 1) % MAXSIZE;
	}

	q -> elements[q -> tail] = value;
	q -> queue_size++;
}
```

```
//TASK 2
//Function to return a production item from the queue
product dequeue(circular_queue *q)
{
	product empty = {0};

	if(is_empty(q))
	{
		return empty;
	}

	product value = q -> elements[q -> head];

	if(q -> head == q -> tail)
	{
		q -> head = -1;
		q -> tail = -1;
	}
	else
	{
		q -> head = (q -> head + 1) % MAXSIZE;
	}

	q -> queue_size-- ;
	return value;
}
```

```
//Task2
void print_production_line2(product arr[], int count)
{
	printf("\n|Product id| Product weight\n");
	for(int i = 0; i < count; i++)
	{
		printf("|%-10d	|%-13.2f \n", arr[i].productID, arr[i].weight);
	}
}

//TASK 3
//Function to find earliest occurence of a product with a particular weight
int binary_search(product arr[], int left, int right, float weight)
{
	int mid;

	if(right >= left)
	{
		mid = left + (right - left) / 2;

		//if value is found
		if(arr[mid].weight == weight)
		{
			return mid;
		}
		else if(arr[mid].weight > weight)
		{
			//search to left
```

```
                return binary_search(arr, left, mid - 1, weight);
            }
            else
            {
                //search to right
                return binary_search(arr, mid + 1, right, weight);
            }
        }

        return -1;
}

//TASK 4
//Function to summarise the number of products in the delivery
void summarise_delivery(product dispatch_list[], int count)
{
        printf("\n--- Delivery Summary Report ---\n");
        printf("Total number of products delivered: %d\n", count);
}
```

Test data files:

line1.txt

1,101,12,8,30,1001,BrakePad,1.2,30.99,ECO123,5

1,101,12,8,35,1002,AirFilter,0.8,15.50,TDI200,2

1,101,12,8,40,1003,OilFilter,0.9,12.30,ECO123,3

1,101,12,8,45,1004,FuelPump,3.1,75.00,HYB567,1

1,101,12,8,50,1005,Alternator,4.5,120.00,GTI300,6

1,101,12,8,55,1006,SparkPlug,0.2,5.99,TDI200,3

1,101,12,9,00,1007,TimingBelt,1.8,45.00,ECO123,4

1,101,12,9,05,1008,WaterPump,2.5,65.00,TDI200,7

1,101,12,9,10,1009,Radiator,6.8,130.00,HYB567,2

1,101,12,9,15,1010,ExhaustPipe,5.5,95.00,GTI300,5

1,101,12,9,20,1011,Headlight,2.0,85.00,ECO123,8

line2.txt

2,102,12,10,00,2001,WheelBearing,1.3,34.99,ECO123,1

2,102,12,10,05,2002,ClutchKit,6.5,199.00,GTI300,4

2,102,12,10,10,2003,DriveShaft,7.1,210.00,HYB567,5

2,102,12,10,15,2004,GasketSet,0.5,9.99,ECO123,3

2,102,12,10,20,2005,BrakeDisc,3.3,70.00,TDI200,2

2,102,12,10,25,2006,Camshaft,4.7,125.00,ECO123,6

2,102,12,10,30,2007,StarterMotor,4.0,99.00,HYB567,8

2,102,12,10,35,2008,EGRValve,2.4,68.00,TDI200,7

2,102,12,10,40,2009,TurboCharger,8.5,300.00,GTI300,9

2,102,12,10,45,2010,ShockAbsorber,6.0,140.00,ECO123,2

2,102,12,10,50,2011,WindshieldWiper,0.4,12.00,HYB567,1

line3.txt

3,103,12,11,00,3001,FogLamp,1.0,28.99,TDI200,2

3,103,12,11,05,3002,ControlArm,4.3,88.00,GTI300,5

3,103,12,11,10,3003,Battery,9.0,250.00,ECO123,7

3,103,12,11,15,3004,ACCompressor,6.2,190.00,HYB567,6

3,103,12,11,20,3005,Thermostat,0.3,8.50,ECO123,3

3,103,12,11,25,3006,Grille,2.6,72.00,TDI200,4

3,103,12,11,30,3007,Muffler,5.2,102.00,GTI300,9

3,103,12,11,35,3008,HoodLatch,1.7,40.00,ECO123,1

3,103,12,11,40,3009,StrutMount,3.4,85.00,TDI200,2

3,103,12,11,45,3010,Axle,6.6,150.00,HYB567,8

3,103,12,11,50,3011,DoorHandle,0.6,14.00,GTI300,5

line4.txt

4,104,12,12,00,4001,MirrorGlass,0.7,22.00,ECO123,6

4,104,12,12,05,4002,WindowMotor,2.3,65.00,TDI200,4

4,104,12,12,10,4003,IgnitionCoil,3.8,95.00,GTI300,1

4,104,12,12,15,4004,DashCam,1.5,75.00,HYB567,9

4,104,12,12,20,4005,Crankshaft,7.8,210.00,ECO123,7

4,104,12,12,25,4006,O2Sensor,0.9,18.00,TDI200,3

4,104,12,12,30,4007,ValveCover,3.2,70.00,GTI300,5

4,104,12,12,35,4008,RockerArm,4.1,88.00,ECO123,2

4,104,12,12,40,4009,BlowerMotor,5.6,120.00,HYB567,8

4,104,12,12,45,4010,SideMirror,2.1,55.00,TDI200,4

4,104,12,12,50,4011,BumperCover,8.2,180.00,GTI300,6