Algorithm Design Project – Áine Carles, C24364956, TU856/1

Project Outline: A car parts firm takes orders from customers and ships them via a logistics company to its customers. During the day, customer orders are divided between 4 teams who work to pick the stock and package them. Each of the 4 teams stores its finished packages in an appropriate bin in the warehouse.

To Do: Process the delivery of car parts by completing the following tasks:

> **Task 1:** Pre-process the 4 files of production data and order each item by weight using an algorithm with running time O(NLog(N)) or better.
>
> **Task 2:** Produce a single dispatch list from the 4 files. Each file will need to be merged to create a single list using an algorithm with running time O(N) or better.
>
> **Task 3:** Provide a user interface to search for the earliest occurrence of a product with a particular weight using an algorithm with running time O(Log(N)) or better.
>
> **Task 4:** Provide a report which summarises the number of products included in the delivery for all vans using an algorithm with running time O(N) or better.

Task 1:

Pre-process the 4 files of production data and order each item by weight using an algorithm with running time O(NLog(N)) or better.

Design: To sort the data from the files, they must be stored in memory. To do this I stored all the information in an array of structures through a for loop . The files were written in CSV form so they could be parsed easily using scanf(). I used merge sort to sort each item by weight as it has a guaranteed time complexity of O(NLog(N)).

Test Plan:

Testing Errors:

- <u>Error – File cannot be found:</u>
  - Expected output - Error message is displayed, program ends.
  - Output matches expected output.

Testing Code:

- Initially test with one file containing a small amount of data.
- Expected output: All the data to be printed ordered by weight.
- The codes output matched my expected output
- Test the code with 4 files each containing information about 10 products.
- Expected output: All the data to be printed ordered by weight.
- The codes output matched my expected output

Pseudocode:

```
START PROGRAM1
        Struct Batch_Time
        {
           int day
           int hour
           int minute
        }

        struct Product
        {
           long line_code
           long batch_code
           Batch_Time BT
           long product_ID
           char name[SIZE]
           char targ_eng[SIZE]
           long bin
           double weight;
        }

        Function Signatures:
        int read_products(*file, Product products[], count)
        void print(Product p)
        void merge(Product products[], l, m, r)
        void merge_split(Product products[], l, r)

        START MAIN
                products[MAX]
```

```
    count = 0
    n = 0

    IF ((n = readP("Product1.txt", products, count) == -1)
        return 1;
    ENDIF

    count += n

    IF (n = readP("Product2.txt", products, count)
        return 1;
    ENDIF

    count += n

     IF ((n = readP("Product3.txt", products, count)) == -1)
        return 1
    ENDIF
    count += n;//20-29

     IF ((n = readP("Product4.txt", products, count)) == -1)
        return 1;
     ENDIF
     count += n

    merge_split(products, 0, count - 1)

    FOR int i = 0; i < count; i++
        print(products[i])
    ENDFOR
END MAIN
```

```
START READP
    open file with fopen
    array to store whats being read from files: store[]
    i = count

    IF  *file == NULL
        print Error
        return -1
    ENDIF

    WHILE store hasn’t reached EOF && i< MAX
        temp variable: p
        sscanf(store, delimiters, addresses of variables)
        products[i++] = p
    ENDWHILE
    close file with fclose
    return i – count
END READP

START PRINT
    print product info stored
END PRINT

START MERGE
    k = l
    n1 = (m-l) + 1
    n2 = r – m

    Product left[s2] right[s2]

    FOR i=0, i<s1, i++
        left[i] = products[l+i]
        ENDFOR

        FOR j=0, j<s1, j++
                right[j] = products[m+1+j]
        ENDFOR

        WHILE i<s1 and j<s2
                IF left[i].weight <= right[j].weight
                        products[k] = left[i]
                        i++
                ENDIF

                ELSE
                products[k] = right[j]
                j++
                ENDELSE
                k++
        ENDWHILE

        WHILE i<s1
                products[k] = left[i]
                i++
                k++
        ENDWHILE

        WHILE j<s2
                products[k] = right[j]
                j++
                k++
        ENDWHILE
END MERGE
```

```
    START MERGE_SPLIT
        IF l<r
            m = l + (r-1)/2
            merge_split(products, l, m)
            merge_split(products, m+1, r)
            merge(products, l, m, r)
        ENDIF
    END MERGE_SPLIT
END PROGRAM1
```

C Code:

```
/* Program to:
Task 1: Sort 4 files of production data based on weight with merge sort*/

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

#define SIZE 30
#define MAX_PRODUCTS 40
#define MAX_LINE 256

typedef struct Batch_Time
{
   int day;
   int hour;
   int minute;
}Batch_Time;

typedef struct Product
{
   long line_code;
   long batch_code;
   Batch_Time BT;
   long product_ID;
   char name[SIZE];
   char targ_eng[SIZE];
   long bin;
   double weight;
}Product;
```

```
// Function signatures
int readP(const char *file, Product products[], int count);
void print(Product p);
void merge(Product products[], int l, int m, int r);
void merge_split(Product products[], int l, int r);

int main()
{
    Product products[MAX_PRODUCTS];
    int count = 0;
    int n;

    if ((n = readP("Product1.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//index 0-9

    if ((n = readP("Product2.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//10-19

    if ((n = readP("Product3.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//20-29

    if ((n = readP("Product4.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//30-39

    merge_split(products, 0, count - 1);

    for (int i = 0; i < count; i++)
    {

        print(products[i]);
    }

    return 0;
}

int readP(const char *file, struct Product products[], int count)
{
    FILE *fp_in = fopen(file, "r");
    char store[MAX_LINE];
    int i = count;

    //Check file has opened correctly
    if (fp_in == NULL)
    {
        printf("Error opening file\n");
        return -1;
    }

    while ((fgets(store, sizeof(store), fp_in)!=NULL) && (i < MAX_PRODUCTS))
    {
        Product p;//temp variable
    //Parse strings in files and store them in array
        sscanf(store, "%ld,%ld,%d,%d,%d,%ld,%[^,],%[^,],%ld,%lf",
               &p.line_code, &p.batch_code,
               &p.BT.day, &p.BT.hour, &p.BT.minute,
               &p.product_ID,
               p.name, p.targ_eng,
               &p.bin, &p.weight);

        products[i++] = p;
    }

    fclose(fp_in);
    return i - count;
}

void print(Product p)
{
    printf("Line Code: %ld Batch Code: %ld Day: %d Hour: %d Minute: %d ", p.line_code,
p.batch_code, p.BT.day, p.BT.hour, p.BT.minute);
    printf("ID: %ld Product: %s Target Engine: %s Bin: %ld Weight: %.2lf\n", p.product_ID,
p.name, p.targ_eng, p.bin, p.weight);
}

void merge(Product products[], int l, int m, int r)
{
    int i, j;
    int k = l;
    int s1 = (m - l) + 1;//size of left array
    int s2 = r - m;//size of right array

    // Create temp arrays
    Product left[s1], right[s2];
```

```
// Copy data from products into temp arrays
for (i = 0; i < s1; i++)
{
    left[i] = products[l + i];
}

for (j = 0; j < s2; j++)
{
    right[j] = products[m + 1 + j];
}

i = 0;
j = 0;
//Compare and merge temp arrays, if one array finishes before the other.
while (i < s1 && j < s2)
{
    if (left[i].weight <= right[j].weight)
    {
        products[k] = left[i];
        i++;
    }

    else
    {
        products[k] = right[j];
        j++;
    }
    k++;
}

// Copy any remaining elements
```

```
    while (i < s1)
    {
        products[k] = left[i];
        i++;
        k++;
    }

    while (j < s2)
    {
        products[k] = right[j];
        j++;
        k++;
    }
}

void merge_split(Product products[], int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;

        merge_split(products, l, m);//Split left side
        merge_split(products, m + 1, r);//Split right side

        merge(products, l, m, r);
    }
}
```

Task 2

Produce a single dispatch list from the 4 files. Each file will need to be merged to create a single list using an algorithm with running time O(N) or better.

Design: Used a function (readP) and a for loop to read and store all the data into an array of structures (a single dispatch list). Files were written in CSV form so they could be parsed easily using sscanf(). Each product is read and added into the array exactly once and only one loop is used, resulting in a time complexity of O(N) for this algorithm.

> Note - While Task 2 may have been intended as a separate step, I combined it with Task 1 by loading all four files into a single array before sorting. This approach still achieves the goal of producing one sorted dispatch list, just in a slightly different way.

Test Plan:

Test Errors:

- Error – File cannot be found:
    - Expected output - Error message is displayed, program ends.
    - Tested by passing a file name that does not exist in my files.
    - Output matches expected output.

Testing Code:

- Initially test with one file containing a small amount of data.
- Expected output: All the data to be printed ordered by weight.
    - (Though ordering it was not part of task 2, the code was already there from task 1 so I still expected it to work)
- The codes output matched my expected output
- Test the code with 4 files each containing information about 10 products.
- Expected output: All the data to be printed ordered by weight.
- The codes output matched my expected output

Pseudocode:

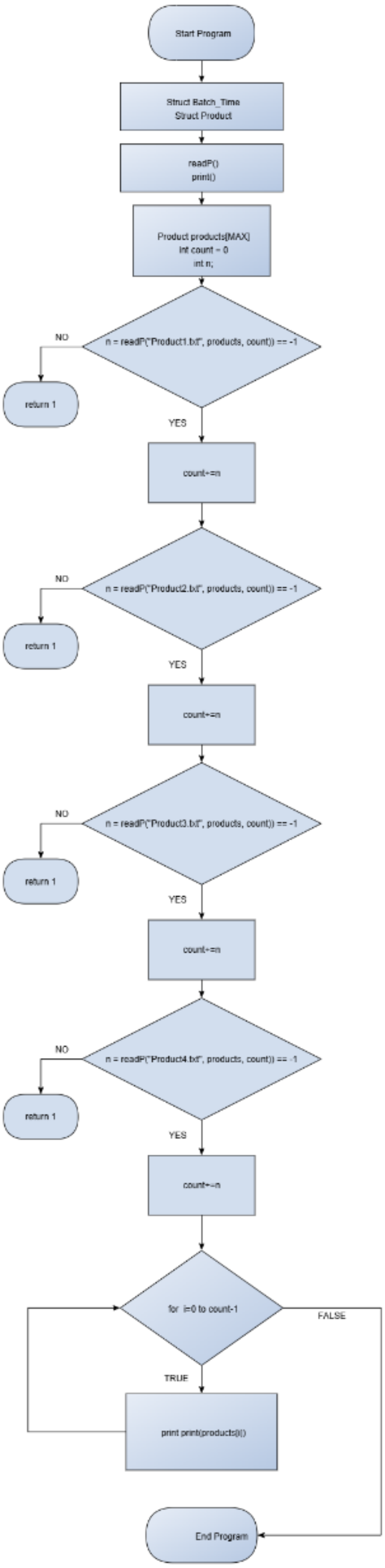
```
START PROGRAM2
        Struct Batch_Time
        {
           int day
           int hour
           int minute
        }

        struct Product
        {
           long line_code
           long batch_code
           Batch_Time BT
           long product_ID
           char name[SIZE]
           char targ_eng[SIZE]
           long bin
           double weight;
        }

        Function Signatures:
        int read_products(*file, Product products[], count)
        void print(Product p)
        void merge(Product products[], l, m, r)
        void merge_split(Product products[], l, r)

        START MAIN
                products[MAX]
                count = 0
                int n = 0
                IF ((n = readP("Product1.txt", products, count) == -1)
                    return 1;
                ENDIF

                count += n

                IF (n = readP("Product2.txt", products, count)
                    return 1;
                ENDIF

                count += n

                 IF ((n = readP("Product3.txt", products, count)) == -1)
                    return 1
                ENDIF
                count += n;//20-29

                 IF ((n = readP("Product4.txt", products, count)) == -1)
                    return 1;
                 ENDIF
                 count += n

                merge_split(products, 0, count - 1)

                FOR int i = 0; i < count; i++
                    print(products[i])
                ENDFOR
        END MAIN

        START READP
                open file with fopen
                array to store what’s being read from files: store[]
                i = count
```

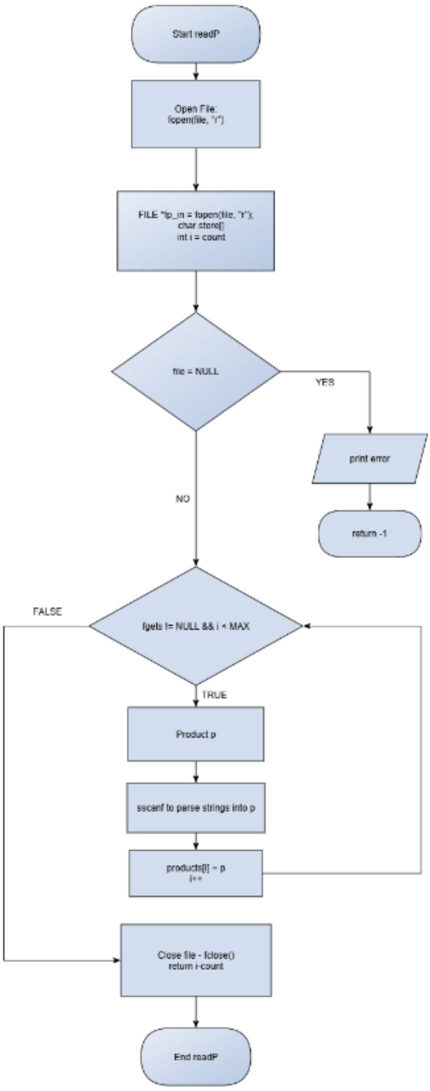
```
            IF  *file == NULL
                print Error
                return -1
            ENDIF

            WHILE store hasn’t reached EOF && i< MAX
                temp variable: p
                sscanf(store, delimiters, addresses of variables)
                products[i++] = p
            ENDWHILE
            close file with fclose
            return i – count
        END READP

        START PRINT
            print product info stored
        END PRINT
    END PROGRAM2
```

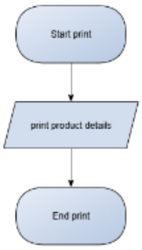
C code:

```
/* Program to:
Task 2: Produce a single dispatch list from the 4 files using an array of structures*/

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

#define SIZE 30
#define MAX_PRODUCTS 40
#define MAX_LINE 256

typedef struct Batch_Time
{
   int day;
   int hour;
   int minute;
}Batch_Time;

typedef struct Product
{
   long line_code;
   long batch_code;
   Batch_Time BT;
   long product_ID;
   char name[SIZE];
   char targ_eng[SIZE];
   long bin;
   double weight;
}Product;

// Function signatures
```

```
int readP(const char *file, Product products[], int count);
void print(Product p);
void merge(Product products[], int l, int m, int r);
void merge_split(Product products[], int l, int r);

int main()
{
    Product products[MAX_PRODUCTS];
    int count = 0;
    int n;

    if ((n = readP("Product1.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//index 0-9

    if ((n = readP("Product2.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//10-19

    if ((n = readP("Product3.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//20-29

    if ((n = readP("Product4.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//30-39

    for (int i = 0; i < count; i++)
    {
        print(products[i]);
    }

    return 0;
}

int readP(const char *file, struct Product products[], int count)
{
    FILE *fp_in = fopen(file, "r");
    char store[MAX_LINE];
    int i = count;

    //Check file has opened correctly
    if (fp_in == NULL)
    {
        printf("Error opening file\n");
        return -1;
    }

    while ((fgets(store, sizeof(store), fp_in)!=NULL) && (i < MAX_PRODUCTS))
    {
        Product p;//temp variable

    //Parse strings in files and store them in array
        sscanf(store, "%ld,%ld,%d,%d,%d,%ld,%[^,],%[^,],%ld,%lf",
                &p.line_code, &p.batch_code,
                &p.BT.day, &p.BT.hour, &p.BT.minute,
                &p.product_ID,
                p.name, p.targ_eng,
                &p.bin, &p.weight);

        products[i++] = p;
    }

    fclose(fp_in);
    return i - count;
}

void print(Product p)
{
    printf("Line Code: %ld Batch Code: %ld Day: %d Hour: %d Minute: %d ",
p.line_code, p.batch_code, p.BT.day, p.BT.hour, p.BT.minute);
    printf("ID: %ld Product: %s Target Engine: %s Bin: %ld Weight: %.2lf\n",
p.product_ID, p.name, p.targ_eng, p.bin, p.weight);
}
```


Start Program
Struct Batch_Time
Struct Product
readP()
print()
Product products[MAX]
int count = 0
int n;
n = readP("Product1.txt", products, count)) == -1
NO
return 1
YES
count+=n
n = readP("Product2.txt", products, count)) == -1
NO
return 1
YES
count+=n
n = readP("Product3.txt", products, count)) == -1
NO
return 1
YES
count+=n
n = readP("Product4.txt", products, count)) == -1
NO
return 1
YES
count+=n
for i=0 to count-1
FALSE
TRUE
print print(products[i])
End Program


Start readP
Open File:
fopen(file, "r")
FILE *fp_in = fopen(file, "r");
char store[]
int i = count
file = NULL
YES
print error
return -1
NO
fgets != NULL && i < MAX
FALSE
TRUE
Product p
sscanf to parse strings into p
products[i] = p
i++
Close file - fclose()
return i-count
End readP


Start print
print product details
End print

Task 3:

Provide a user interface to search for the earliest occurrence of a product with a particular weight using an algorithm with running time O(Log(N)) or better.

Design: Create a variable to store the user's input. Use scanf() to read in user's input. Create a function(binary_search_weight) to find the earliest occurrence of the inputted weight if it exists. I used the binary search algorithm for this as it works for sorted arrays and has a guaranteed time complexity of O(Log(N)).

Test Plan:

Test Errors:

- Error – Weight Doesn't Exist:
  - binary_search_weight returns -1 and error message is displayed

Test Code:

- Expected Output: The earliest index where the weight occurs is displayed.
- Issue – At first, if you entered the number of the very last weight, the error message was displayed.
- I realised this was because I was passing count-1 as the size of the array.
- Once fixed, the code's output matched my expected output.

Pseudocode:

```
START PROGRAM3
        Struct Batch_Time
        {
           int day
           int hour
           int minute
        }

        struct Product
        {
           long line_code
           long batch_code
           Batch_Time BT
           long product_ID
           char name[SIZE]
           char targ_eng[SIZE]
           long bin
           double weight;
        }

        Function Signatures:
        int read_products(*fp, Product products[], count)
```

```
void print_products(Product p)
void merge(Product products[], l, m, r)
void merge_split(Product products[], l, r)
int binary_search_weight(Product products[], int size, double input_weight)


START MAIN
    products[MAX]
    count = 0
    input_weight = 0
    n = 0
    IF ((n = readP("Product1.txt", products, count) == -1)
        return 1;
    ENDIF

    count += n

    IF (n = readP("Product2.txt", products, count)
        return 1;
    ENDIF

    count += n

     IF ((n = readP("Product3.txt", products, count)) == -1)
        return 1
    ENDIF
    count += n;//20-29

     IF ((n = readP("Product4.txt", products, count)) == -1)
        return 1;
     ENDIF
     count += n
```

```
        merge_split(products, 0, count - 1)

        FOR int i = 0; i < count; i++
            print(products[i])
        ENDFOR

        print “Enter weight”
        scanf input

        index = binary_search_weight(products, count, input_weight)

        IF index = -1
            print “Weight not found”
        ENDIF

        ELSE
            print index of earliest occurrence
        ENDELSE
ENDMAIN




START READP
        open file with fopen
        array to store whats being read from files: store[]
        i = count

        IF  *fp == NULL
            print Error
            return -1
        ENDIF
        WHILE store hasn't reached EOF && i< MAX
                temp variable: p
                sscanf(store, delimiters, addresses of variables)
                products[i++] = p
        ENDWHILE
        close file with fclose
        return i – count
END READ_PRODUCTS

START PRINT
        print product info stored
END PRINT_PRODUCTS

START MERGE
        k = l
        n1 = (m-l) + 1
        n2 = r – m

        Product left[s2] right[s2]

        FOR i=0, i<s1, i++
                left[i] = products[l+i]
        ENDFOR

        FOR j=0, j<s1, j++
                right[j] = products[m+1+j]
        ENDFOR

        WHILE i<s1 and j<s2
                IF left[i].weight <= right[j].weight
                        products[k] = left[i]
```

```
                    i++
                ENDIF

                ELSE
                products[k] = right[j]
                j++
                ENDELSE
                k++
        ENDWHILE

        WHILE i<s1
                products[k] = left[i]
                i++
                k++
        ENDWHILE

        WHILE j<s2
                products[k] = right[j]
                j++
                k++
        ENDWHILE
END MERGE

START MERGE_SPLIT
        IF l < r
                m = l + (r-1)/2
                merge_split(products, l, m)
                merge_split(products, m+1, r)
                merge(products, l, m, r)
        ENDIF
END MERGE_SPLIT
```

```
    START BINARY_SEARCH_WEIGHT
        left = 0
        right = size-1
        result = -1

        WHILE left <= right
            Mid =left + (right-left)/2
            IF products[mid].weight == input_weight
                result = mid
                right = mid-1
            ENDIF
            ELSEIF products[mid].weight < input_weight
                Left = mid + 1
        END ELSEIF

        ELSE
            right = mid – 1
        ENDELSE
    ENDWHILE
    return result
END PROGRAM3
```

C Code:

```
/* Program to:
Task 1: Sort 4 files of production data based on weight with merge sort
Task 2: Produce a single dispatch list from the 4 files using an array of structures
Task 3: Provide a user interface to search for the earliest occurrence of a product with a particular weight using binary search*/


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

```
#define SIZE 30
#define MAX_PRODUCTS 40
#define MAX_LINE 256

typedef struct Batch_Time
{
   int day;
   int hour;
   int minute;
}Batch_Time;

typedef struct Product
{
   long line_code;
   long batch_code;
   Batch_Time BT;
   long product_ID;
   char name[SIZE];
   char targ_eng[SIZE];
   long bin;
   double weight;
}Product;

// Function signatures
int readP(const char *file, Product products[], int count);
void print(Product p);
void merge(Product products[], int l, int m, int r);
void merge_split(Product products[], int l, int r);
int binary_search_weight(Product products[], int size, double input_weight);

int main()
```

```
{
    Product products[MAX_PRODUCTS];
    int count = 0;
    double input_weight;
    int n;

    if ((n = readP("Product1.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//index 0-9

    if ((n = readP("Product2.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//10-19

    if ((n = readP("Product3.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//20-29

    if ((n = readP("Product4.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//30-39

    merge_split(products, 0, count - 1);
    for (int i = 0; i < count; i++)
    {

        print(products[i]);
    }

    printf("\nEnter the weight you want to search:");
    scanf("%lf", &input_weight);

    int index = binary_search_weight(products, count, input_weight);

    if(index == -1)
    {
        printf("Weight not found\n");
    }

    else
    {
        printf("The earliest occurence of weight %.2lf is at index %d", input_weight, index);
    }

    return 0;
}

int readP(const char *file, struct Product products[], int count)
{
        FILE *fp_in = fopen(file, "r");
        char store[MAX_LINE];
        int i = count;

    //Check file has opened correctly
    if (fp_in == NULL)
    {
        printf("Error opening file\n");
        return -1;
    }

    //Parse strings in files and store them in array
    while ((fgets(store, sizeof(store), fp_in)!=NULL) && (i < MAX_PRODUCTS))
    {
        Product p;//temp variable

        sscanf(store, "%ld,%ld,%d,%d,%d,%ld,%[^,],%[^,],%ld,%lf",
               &p.line_code, &p.batch_code,
               &p.BT.day, &p.BT.hour, &p.BT.minute,
               &p.product_ID,
               p.name, p.targ_eng,
               &p.bin, &p.weight);

        products[i++] = p;
    }

    fclose(fp_in);
    return i - count;
}

void print(Product p)
{
    printf("Line Code: %ld Batch Code: %ld Day: %d Hour: %d Minute: %d ", p.line_code,
p.batch_code, p.BT.day, p.BT.hour, p.BT.minute);
    printf("ID: %ld Product: %s Target Engine: %s Bin: %ld Weight: %.2lf\n", p.product_ID,
p.name, p.targ_eng, p.bin, p.weight);
}

void merge(Product products[], int l, int m, int r)
```

```
{
    int i, j;
    int k = l;
    int s1 = (m - l) + 1;//size of left array
    int s2 = r - m;//size of right array

    // Create temp arrays
    Product left[s1], right[s2];

    // Copy data from products into temp arrays
    for (i = 0; i < s1; i++)
    {
        left[i] = products[l + i];
    }

    for (j = 0; j < s2; j++)
    {
        right[j] = products[m + 1 + j];
    }

    i = 0;
    j = 0;
    //Compare and merge temp arrays, if one array finishes before the other.
    while (i < s1 && j < s2)
    {
        if (left[i].weight <= right[j].weight)
        {
            products[k] = left[i];
            i++;
        }

        else
        {
            products[k] = right[j];
            j++;
        }
        k++;
    }

    // Copy any remaining elements
    while (i < s1)
    {
        products[k] = left[i];
        i++;
        k++;
    }

    while (j < s2)
    {
        products[k] = right[j];
        j++;
        k++;
    }
}

void merge_split(Product products[], int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;

        merge_split(products, l, m);//Split left side
        merge_split(products, m + 1, r);//Split right side
        merge(products, l, m, r);
    }
}

int binary_search_weight(Product products[], int size, double input_weight)
{
    int left = 0;
    int right = size - 1;
    int result = -1;

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

        if (products[mid].weight == input_weight) //starts search at middle index, does it equal inputted weight?
        {
            result = mid;
            right = mid - 1; // keep searching left to find the first occurrence
        }

        else if (products[mid].weight < input_weight)
        {
            left = mid + 1;//keep searching right
        }

        else
        {
            right = mid - 1;//keep searching left
        }
    }
    return result;
}
```

Task 4

Provide a report which summarises the number of products included in the delivery for all vans using an algorithm with running time O(N) or better.

Design: Created a function (summary_report) and passed the variable count as an argument. "count" contains the total amount of products. It has a time complexity of O(1) as it doesn't contain any loops or operations that scale with the number of products.

Test Plan:

- Expected output: For the total amount of products to be displayed.
- This is exactly what happened as count tracks the number of products throughout the whole program.

Pseuodocode

```
START PROGRAM4

        Struct Batch_Time
        {
           int day
           int hour
           int minute
        }

        struct Product
        {
           long line_code
           long batch_code
```

```
  Batch_Time BT
  long product_ID
  char name[SIZE]
  char targ_eng[SIZE]
  long bin
  double weight;
}
```

Function Signatures:

```
int read_products(*fp, Product products[], count)
void print_products(Product p)
void merge(Product products[], l, m, r)
void merge_split(Product products[], l, r)
int binary_search_weight(Product products[], int size, double input_weight)
void summary_report(count)
```

```
START MAIN
      products[MAX]
      count = 0
      input_weight = 0
      n = 0
      IF ((n = readP("Product1.txt", products, count) == -1)
          return 1;
      ENDIF

      count += n

      IF (n = readP("Product2.txt", products, count)
          return 1;
      ENDIF
```

```
    count += n

     IF ((n = readP("Product3.txt", products, count)) == -1)
        return 1
    ENDIF
    count += n;//20-29

     IF ((n = readP("Product4.txt", products, count)) == -1)
        return 1;
     ENDIF
     count += n

    merge_split(products, 0, count - 1)

    FOR int i = 0; i < count; i++
        print(products[i])
    ENDFOR

    print “Enter weight”
    scanf input

    index = binary_search_weight(products, count, input_weight)

    IF index = -1
          print “Weight not found”
    ENDIF

    ELSE
          print index of earliest occurrence
    ENDELSE
    summary_report(count)
ENDMAIN
```

```
START READP
        open file with fopen
        array to store whats being read from files: store[]
        i = count

        IF  *fp == NULL
                print Error
                return -1
        ENDIF

        WHILE store hasn’t reached EOF && i< MAX
                temp variable: p
                sscanf(store, delimiters, addresses of variables)
                products[i++] = p
        ENDWHILE
        close file with fclose
        return i – count
END READ_PRODUCTS

START PRINT
        print product info stored
END PRINT_PRODUCTS

START MERGE
        k = l
        n1 = (m-l) + 1
        n2 = r – m

        Product left[s2] right[s2]
```

```
FOR i=0, i<s1, i++
        left[i] = products[l+i]
ENDFOR

FOR j=0, j<s1, j++
        right[j] = products[m+1+j]
ENDFOR

WHILE i<s1 and j<s2
        IF left[i].weight <= right[j].weight
                products[k] = left[i]
                i++
        ENDIF


        ELSE
        products[k] = right[j]
        j++
        ENDELSE
        k++
ENDWHILE

WHILE i<s1
        products[k] = left[i]
        i++
        k++
ENDWHILE

WHILE j<s2
        products[k] = right[j]
        j++
        k++
    ENDWHILE
END MERGE

START MERGE_SPLIT
    IF l < r
        m = l + (r-1)/2
        merge_split(products, l, m)
        merge_split(products, m+1, r)
        merge(products, l, m, r)
    ENDIF
END MERGE_SPLIT

START BINARY_SEARCH_WEIGHT
    left = 0
    right = size-1
    result = -1

    WHILE left <= right
        Mid =left + (right-left)/2
        IF products[mid].weight == input_weight
            result = mid
            right = mid-1
        ENDIF
        ELSEIF products[mid].weight < input_weight
            Left = mid + 1
        END ELSEIF

        ELSE
            right = mid – 1
        ENDELSE
    ENDWHILE
    return result
    END BINARY_SEARCH_WEIGHT

    START SUMMARY_REPORT
        print total number of products included in delivery
    END SUMMARY_REPORT
END PROGRAM4
```

C Code:

```
/* Program to:
Task 1: Sort 4 files of production data based on weight with merge sort
Task 2: Produce a single dispatch list from the 4 files using an array of structures
Task 3: Provide a user interface to search for the earliest occurrence of a product with a particular weight using binary search
Task 4: Provide a report which summarises the number of products included in the delivery for all vans.  */

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

#define SIZE 30
#define MAX_PRODUCTS 40
#define MAX_LINE 256

typedef struct Batch_Time
{
  int day;
  int hour;
  int minute;
}Batch_Time;

typedef struct Product
{
    long line_code;
    long batch_code;
    Batch_Time BT;
    long product_ID;
    char name[SIZE];
    char targ_eng[SIZE];
    long bin;
    double weight;
}Product;

// Function signatures
int readP(const char *file, Product products[], int count);
void print(Product p);
void merge(Product products[], int l, int m, int r);
void merge_split(Product products[], int l, int r);
int binary_search_weight(Product products[], int size, double input_weight);
void summary_report(int count);

int main()
{
    Product products[MAX_PRODUCTS];
    int count = 0;
    double input_weight;
    int n;

    if ((n = readP("Product1.txt", products, count)) == -1)
    {
        return 1;
    }
    count += n;//index 0-9

    if ((n = readP("Product2.txt", products, count)) == -1)
```

```
{
   return 1;
}
count += n;//10-19

if ((n = readP("Product3.txt", products, count)) == -1)
{
   return 1;
}
count += n;//20-29

if ((n = readP("Product4.txt", products, count)) == -1)
{
   return 1;
}
count += n;//30-39

merge_split(products, 0, count - 1);

for (int i = 0; i < count; i++)
{

   print(products[i]);
}

printf("\nEnter the weight you want to search:");
scanf("%lf", &input_weight);

int index = binary_search_weight(products, count, input_weight);

if(index == -1)
{
        printf("Weight not found\n");
    }

    else
    {
        printf("The earliest occurence of weight %.2lf is at index %d", input_weight, index);
    }

    summary_report(count);

    return 0;
}

int readP(const char *file, struct Product products[], int count)
{
        FILE *fp_in = fopen(file, "r");
        char store[MAX_LINE];
        int i = count;

    //Check file has opened correctly
    if (fp_in == NULL)
    {
        printf("Error opening file\n");
        return -1;
    }

    //Parse strings in files and store them in array
    while ((fgets(store, sizeof(store), fp_in)!=NULL) && (i < MAX_PRODUCTS))
    {
        Product p;//temp variable

        sscanf(store, "%ld,%ld,%d,%d,%d,%ld,%[^,],%[^,],%ld,%lf",
                &p.line_code, &p.batch_code,
                &p.BT.day, &p.BT.hour, &p.BT.minute,
                &p.product_ID,
                p.name, p.targ_eng,
                &p.bin, &p.weight);

        products[i++] = p;
    }

    fclose(fp_in);
    return i - count;
}

void print(Product p)
{
    printf("Line Code: %ld Batch Code: %ld Day: %d Hour: %d Minute: %d ", p.line_code,
p.batch_code, p.BT.day, p.BT.hour, p.BT.minute);
    printf("ID: %ld Product: %s Target Engine: %s Bin: %ld Weight: %.2lf\n", p.product_ID,
p.name, p.targ_eng, p.bin, p.weight);
}

void merge(Product products[], int l, int m, int r)
{
    int i, j;
    int k = l;
    int s1 = (m - l) + 1;//size of left array
    int s2 = r - m;//size of right array

    // Create temp arrays
    Product left[s1], right[s2];

    // Copy data from products into temp arrays
    for (i = 0; i < s1; i++)
```

```
{
    left[i] = products[l + i];
}

for (j = 0; j < s2; j++)
{
    right[j] = products[m + 1 + j];
}

i = 0;
j = 0;
//Compare and merge temp arrays, if one array finishes before the other.
while (i < s1 && j < s2)
{
    if (left[i].weight <= right[j].weight)
    {
        products[k] = left[i];
        i++;
    }

    else
    {
        products[k] = right[j];
        j++;
    }
    k++;
}

// Copy any remaining elements
while (i < s1)
{
    products[k] = left[i];
```

```
        i++;
        k++;
    }

    while (j < s2)
    {
        products[k] = right[j];
        j++;
        k++;
    }
}

void merge_split(Product products[], int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;

        merge_split(products, l, m);//Split left side
        merge_split(products, m + 1, r);//Split right side

        merge(products, l, m, r);
    }
}

int binary_search_weight(Product products[], int size, double input_weight)
{
    int left = 0;
    int right = size - 1;
    int result = -1;

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


        if (products[mid].weight == input_weight) //starts search at middle index, does it equal inputted weight?
        {
            result = mid;
            right = mid - 1; // keep searching left to find the first occurrence
        }


        else if (products[mid].weight < input_weight)
        {
            left = mid + 1;//keep searching right
        }


        else
        {
            right = mid - 1;//keep searching left
        }
    }


    return result;
}


void summary_report(int count)
{
    printf("\nThe total number of products included in the delivery = %d", count);
}
```