

# Neil Jiang - C24510496 – CMPU 1001 Project Report

## Data Structure

```
1 struct date
2 {
3     int day;
4     int hour;
5     int minute;
6 };
7 struct product
8 {
9     int lineCode;
10    int batchCode;
11    struct date batchDate;
12    int productId;
13    char productName[SIZE];
14    char targetEngineCode[SIZE];
15    int binNumber;
16    int weight;
17    float price;
18 };
```

## Test Data

A CSV file was used to store the test data. Each line is then read using C code, and stored in its respective structure arrays.

Line 1:

	A	B	C	D	E	F	G	H	I	J	K
1	lineCode	batchCode	Day	Hour	minute	productId	productName	targetEngineCode	binNumber	weight	price
2	1	1001	15	9	30	5001	Spark Plug	V6-2.0L	12	92	8.99
3	1	1002	15	10	15	5002	Oil Filter	I4-1.5L	5	93	12.52
4	1	1003	15	10	30	5003	Air Filter	V8-5.0L	8	17	9.99
5	1	1004	15	11	0	5004	Brake Pad Set	R6-3.0L	10	77	43.29
6	1	1005	15	11	45	5005	Windshield Wiper	L4-2.4L	3	86	6.52
7	1	1006	15	12	0	5006	Fuel Pump	D4-2.0L	7	76	32.12
8	1	1007	15	12	30	5007	Ignition Coil	V6-3.6L	9	34	22.36
9	1	1008	15	13	15	5008	Timing Belt	H4-1.8L	4	97	18.23
10	1	1009	15	14	0	5009	Alternator	V8-6.2L	11	79	112.36
11	1	1010	15	14	45	5010	Radiator	F4-1.6L	6	52	86.12

Line 2:

	A	B	C	D	E	F	G	H	I	J	K
1	lineCode	batchCode	Day	Hour	minute	productId	productName	targetEngineCode	binNumber	weight	price
2	2	2001	15	8	45	6001	Battery	T5-2.5L	2	11	150.02
3	2	2002	15	9	20	6002	Clutch Kit	V6-3.0L	5	48	95.32
4	2	2003	15	9	55	6003	Exhaust Manifold	I4-1.6L	8	83	65.25
5	2	2004	15	10	30	6004	Steering Rack	L4-2.0L	4	13	200.01
6	2	2005	15	11	5	6005	Cabin Filter	H4-1.4L	7	66	10.67
7	2	2006	15	11	40	6006	Glow Plug	D4-2.2L	9	47	14.92
8	2	2007	15	12	15	6007	Throttle Body	V8-5.7L	11	96	75.24
9	2	2008	15	12	50	6008	Wheel Bearing	R6-3.6L	6	35	40.81
10	2	2009	15	13	25	6009	Oxygen Sensor	I4-2.0L	3	98	25.22
11	2	2010	15	14	0	6010	Shock Absorber	L4-1.8L	10	69	90.03

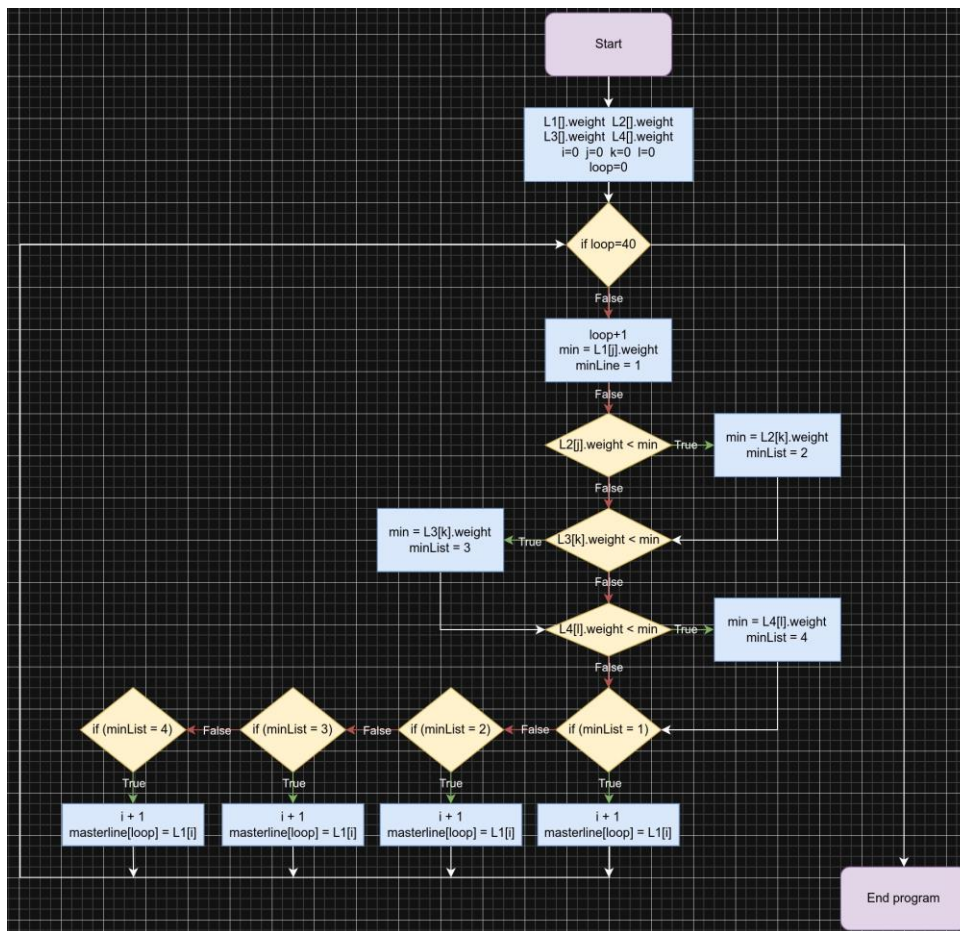
Line 3:

	A	B	C	D	E	F	G	H	I	J	K
1	lineCode	batchCode	Day	Hour	minute	productId	productName	targetEngineCode	binNumber	weight	price
2	3	3001	15	9	0	7001	Headlight Bulb	H4-1.6L	1	88	5.52
3	3	3002	15	9	35	7002	EGR Valve	V6-2.7L	5	37	35.46
4	3	3003	15	10	10	7003	Transmission Fluid	F4-1.5L	9	27	12.02
5	3	3004	15	10	45	7004	Catalytic Converter	V8-6.0L	12	38	300.14
6	3	3005	15	11	20	7005	Power Steering Pump	R6-3.3L	6	72	80.95
7	3	3006	15	11	55	7006	Turbocharger	I4-2.0L	7	60	450.72
8	3	3007	15	12	30	7007	ABS Sensor	L4-2.2L	4	68	18.51
9	3	3008	15	13	5	7008	Drive Belt	V6-3.5L	8	80	15.23
10	3	3009	15	13	40	7009	Water Pump	H4-1.8L	10	62	55.62
11	3	3010	15	14	15	7010	Engine Mount	D4-2.5L	3	75	30.48

Line 4:

	A	B	C	D	E	F	G	H	I	J	K
1	lineCode	batchCode	Day	Hour	minute	productId	productName	targetEngineCode	binNumber	weight	price
2	4	4001	15	8	30	8001	Spark Plug Wire Set	V6-3.2L	2	30	20.12
3	4	4002	15	9	5	8002	PCV Valve	I4-1.8L	5	57	8.23
4	4	4003	15	9	40	8003	Camshaft Position Sensor	V8-5.4L	9	73	25.53
5	4	4004	15	10	15	8004	Idler Pulley	L4-2.5L	7	24	12.81
6	4	4005	15	10	50	8005	Fuel Injector	R6-3.8L	6	81	50.66
7	4	4006	15	11	25	8006	MAF Sensor	H4-1.6L	4	56	35.33
8	4	4007	15	12	0	8007	Starter Motor	V6-2.8L	10	82	130.12
9	4	4008	15	12	35	8008	Valve Cover Gasket	I4-2.4L	8	85	10.54
10	4	4009	15	13	10	8009	Differential Fluid	F4-1.8L	3	23	15.91
11	4	4010	15	13	45	8010	Control Arm	L4-2.0L	11	70	75.43

## Flow Chart for Task 2



## Task 1

For this task, I was asked to use an appropriate sorting algorithm to sort 4 structure arrays by their weight. The sorting algorithm must run with a time complexity of  $O(n \log(n))$  or better, so I decided to use merge sort. This is because merge sort has a worst case time complexity of  $O(n \log(n))$ , which was perfect.

```
mergesort(line[], left, right)
// Find the middle
IF (left < right)
    middle = left + (right - left) / 2

    // Sort first and second half
    mergesort(line, left, middle)
    mergesort(line, middle + 1, right)

    // Merge the two halves
    merge(line, left, middle, right)
END if
END function
```

Here is a picture of the pseudocode I used to design the algorithm:

```
merge(line[], left, middle, right):
    n1 = middle - left + 1
    n2 = right - middle

    // Create temporary structures for storing values
    struct product tempLeft[LINESIZE], tempRight[LINESIZE]

    // Copy data from lines to temp structs
    FOR (i=0; i<n1; i++)
        tempLeft[i] = line[left + i]
    END for

    FOR (i=0; i<n2; i++)
        tempRight[i] = line[middle + i + 1]
    END for

    // Merge temp structs
    i = 0
    j = 0
    k = left

    WHILE (i < n1 AND j < n2)
        IF (tempLeft[i].weight <= tempRight[j].weight)
            line[k] = tempLeft[i]
            i++
        ELSE
            line[k] = tempRight[j]
            j++
        END if
        k++
    END while

    WHILE (i < n1)
        line[k] = tempLeft[i]
        i++
        k++
    END while

    WHILE (j < n2)
        line[k] = tempRight[j]
        j++
        k++
    END while
END FUNCTION
```

Here is the algorithm implemented in C:

```
1 // Sorts the 4 lines
2 void mergesort(struct product line[LINESIZE], int left, int right) {
3     int middle;
4
5     // Find the middle
6     if (left < right) {
7         middle = left + (right - left) / 2;
8
9         // Sort first and second half
10        mergesort(line, left, middle);
11        mergesort(line, middle+1, right);
12
13        // Merge the two halves
14        merge(line, left, middle, right);
15    } // END if
16 } // END function
17
18 void merge(struct product line[LINESIZE], int left, int middle, int right) {
19     int i;
20     int j;
21     int k;
22
23     int n1 = middle - left + 1;
24     int n2 = right - middle;
25
26     // Create temporary structures for storing values
27     struct product tempLeft[LINESIZE];
28     struct product tempRight[LINESIZE];
29
30     // Copy data from lines to temp structs
31     for (i=0; i<n1; i++) {
32         tempLeft[i] = line[left + i];
33     }
34     for (i=0; i<n2; i++) {
35         tempRight[i] = line[middle + i + 1];
36     }
37
38     // Merge temp structs
39     i = 0;
40     j = 0;
41     k = left;
42
43
44     while (i<n1 && j<n2) {
45         if (tempLeft[i].weight <= tempRight[j].weight) {
46             line[k] = tempLeft[i];
47             i++;
48         }
49         else {
50             line[k] = tempRight[j];
51             j++;
52         }
53         k++;
54     } // END while
55
56     while (i<n1) {
57         line[k] = tempLeft[i];
58         i++;
59         k++;
60     } // END while
61
62     while (j<n2) {
63         line[k] = tempRight[j];
64         j++;
65         k++;
66     } // END while
67 } // END function
```

## Task 2

For this task, I had to design an algorithm to merge the 4 sorted structure arrays. The running speed of the algorithm must have a time complexity of  $O(n)$  or better. When I initially started designing this algorithm, I realised it was very similar to the “merge” part from merge sort. However, instead of checking and merging two arrays at a time, I expanded upon it to merge 4 arrays.

To explain how this algorithm works, it first goes through the first element of each of the lines, finding the smallest between them. Once that minimum value is found, it is appended to the “master line” (The master line is essentially a structure array that holds all the merged lines). Then, it finds the next minimum value between the lines, and appends that to the master line. This process is repeated until each product in the 4 lines has been appended to the master line. This algorithm has a worse case time complexity of  $O(n)$ , as it only uses one loop to go through each element.

Here is the pseudocode:

```
mergeLines(L1, L2, L3, L4, masterLine)
  n1 = 0
  n2 = 0
  n3 = 0
  n4 = 0

  // Loop through each the nth element of each line, and find the min
  FOR (i = 0: i<MASTERSIZE; i++)
    min = (biggest possible number, so that the next min can be found)

    // Compare current elements of all 4 lines
    IF (L1[n1].weight < min AND n1 < LINESIZE)
      min = L1[n1].weight
      minList = 1
    END if

    IF (L2[n2].weight < min AND n2 < LINESIZE)
      min = L2[n2].weight
      minList = 2
    END if

    IF (L3[n3].weight < min AND n3 < LINESIZE)
      min = L3[n3].weight
      minList = 3
    END if

    IF (L4[n4].weight < min AND n4 < LINESIZE)
      min = L4[n4].weight
      minList = 4
    END if

    // Append the min value to master line, then move that line up by 1
    IF (minList == 1)
      masterLine[i] = L1[n1]
      n1++
    ELSE IF (minList == 2)
      masterLine[i] = L2[n2]
      n2++
    ELSE IF (minList == 3)
      masterLine[i] = L3[n3]
      n3++
    ELSE IF (minList == 4)
      masterLine[i] = L4[n4]
      n4++
    END IF
  END FOR
END FUNCTION
```

Here is the algorithm implemented in C:

```
1 // Merges all 4 lines
2 void mergeLines(struct product L1[LINESIZE], struct product L2[LINESIZE], struct product L3[LINESIZE], struct product L4[LINESIZE], struct product masterLine[MASTERSIZE]) {
```

```
1     int n1 = 0;
2     int n2 = 0;
3     int n3 = 0;
4     int n4 = 0;
5     int min;
6     int minList;
7
8     // Loop through each the nth element of each line, and find the min
9     for (int i=0; i<MASTERSIZE; i++) {
10         min = INT_MAX;
11
12         // Compare current elements of all 4 lines
13         if (L1[n1].weight < min && n1 < LINESIZE) {
14             min = L1[n1].weight;
15             minList = 1;
16         }
17         if (L2[n2].weight < min && n2 < LINESIZE) {
18             min = L2[n2].weight;
19             minList = 2;
20         }
21         if (L3[n3].weight < min && n3 < LINESIZE) {
22             min = L3[n3].weight;
23             minList = 3;
24         }
25         if (L4[n4].weight < min && n4 < LINESIZE) {
26             min = L4[n4].weight;
27             minList = 4;
28         }
29
30         // Append that min value to the master line, then move that line up by 1
31         if (minList == 1) {
32             masterLine[i] = L1[n1];
33             n1++;
34         }
35         else if (minList == 2) {
36             masterLine[i] = L2[n2];
37             n2++;
38         }
39         else if (minList == 3) {
40             masterLine[i] = L3[n3];
41             n3++;
42         }
43         else if (minList == 4) {
44             masterLine[i] = L4[n4];
45             n4++;
46         }
47     } // END for
48 } // END function
```

\*image split for readability



### Task 3

For this task, I had to implement a searching algorithm to find a specific car part by weight. The running time of the searching algorithm needed to have a time complexity of  $O(\log(N))$  or better. Due to the line of car parts being already sorted by tasks 1-2, I decided to use binary search, which has a worst case time complexity of  $O(\log(n))$ . Binary search uses “divide and conquer” to repeatedly divide the array in half, until the target is found.

Here is the pseudocode:

```
binarySearch(line[], size):
    // Ask user to enter a weight
    PRINT "Enter weight to find: "
    READ target

    // Set boundaries for binary search
    left = 0
    right = size - 1
    targetIndex = -1

    WHILE (left <= right)
        mid = left + (right-left)/2

        // Regular binary search
        IF (line[mid].weight == target)
            targetIndex = mid
            break
        ELSE IF (line[mid].weight < target)
            left = mid + 1 // Search the right half
        ELSE
            right = mid - 1 // Search the left half
        END if
    END while

    // Tell user if weight has been found or not
    IF (targetIndex == -1)
        PRINT "Target weight {target} not found"
    ELSE
        PRINT "Target weight {target} FOUND"
    END if
END function
```

Here is binary search implemented in C, to work with the car parts.

```
1 // Search for car part by weight
2 void binarySearch(struct product line[], int size) {
3     int left;
4     int right;
5     int mid;
6     int target;
7     int targetIndex;
8
9     // Ask user to enter a weight
10    printf("\nEnter weight to find\n");
11    printf("-> ");
12    scanf("%d", &target);
13    while(getchar() != '\n'); //clear input buffer
14
15    // Set boundaries for binary search
16    left = 0;
17    right = size-1;
18    targetIndex = -1;
19
20    while (left <= right) {
21        mid = left + (right-left)/2;
22
23        // Regular binary search
24        if (line[mid].weight == target) {
25            targetIndex = mid;
26            break;
27        }
28        else if (line[mid].weight < target) {
29            left = mid+1; // Search the right half
30        }
31        else {
32            right = mid-1; // Search the left half
33        }
34    } // END while
35
36    // Tell user if weight has been found or not
37    if (targetIndex == -1) {
38        printf("\nTarget weight %d not found\n", target);
39    }
40    else {
41        printf("\nTarget weight %d FOUND\n",target);
42        showStructVar(line, targetIndex, MASTERSIZE, 2);
43    }
44 } // END function
```

## Task 4

For this task, I had to design an algorithm that can create a report/delivery docket on the car parts included in the delivery for all vans. Each product must be stored in the van in weight order, however this is not a problem, as the master line is already sorted. The speed of this algorithm needed to be  $O(n)$  or better. To do this, I first created variables/symbolic names, containing the number of vans present, and the total weight limit for each van. Then, from the master line, the first product is stored in the first van, the second product is stored in the second van, and so on and so forth. If and when one of the vans hits the weight limit, the product will instead be stored inside the van after it. This repeats until either all the products are stored in the vans, or until the weight limit for each van is exceeded, in which the user is then informed. The worst case time complexity of this algorithm is  $O(n)$ , as it only contains one loop to go through each element in the master line.

Here is the pseudocode:

```
vanReport(line[], van1[], van2[], van3[], van4[], van5[], vanCount[], totalWeight[])
    vanIndex[NUMVAN] = Set to All Zeros
    PRINT "Generating van report..."

    // Goes through each product in the masterline
    FOR (int i=0; i<MASTERSIZE;) // DO NOT INCREMENT!!
        // Adds current product to van 1 if weight limit not exceeded
        IF (totalWeight[0]+line[i].weight <= WEIGHTLIMIT)
            van1[vanIndex[0]] = line[i]
            totalWeight[0] += line[i].weight

            vanIndex[0]++
            vanCount[0]++
            i++
        // END if

        // Adds current product to van 2 if weight limit not exceeded
        ELSE IF (totalWeight[1] + line[i].weight <= WEIGHTLIMIT) THEN
            van1[vanIndex[1]] = line[i]
            totalWeight[1] += line[i].weight

            vanIndex[1]++
            vanCount[1]++
            i++
        // END else if

        // Adds current product to van 3 if weight limit not exceeded
        ELSE IF (totalWeight[2] + line[i].weight <= WEIGHTLIMIT) THEN
            van1[vanIndex[2]] = line[i]
            totalWeight[2] += line[i].weight

            vanIndex[2]++
            vanCount[2]++
            i++
        // END else if

        // Adds current product to van 4 if weight limit not exceeded
        ELSE IF (totalWeight[3] + line[i].weight <= WEIGHTLIMIT) THEN
            van1[vanIndex[3]] = line[i]
            totalWeight[3] += line[i].weight

            vanIndex[3]++
            vanCount[3]++
            i++
        // END else if

        // Adds current product to van 5 if weight limit not exceeded
        ELSE IF (totalWeight[4] + line[i].weight <= WEIGHTLIMIT) THEN
            van1[vanIndex[4]] = line[i]
            totalWeight[4] += line[i].weight

            vanIndex[4]++
            vanCount[4]++
            i++
        // END else if

        // Alternative case for when no van can store product without exceeding weight limit
        ELSE
            PRINT "Weight limit [WEIGHTLIMIT] is too small. Increase limit or add more vans."
            RETURN // Terminate function early
        END if
    END for
    PRINT "Van report generated!"
END FUNCTION
```

Here is the C code created from the pseudocode:

```
1 // Create report for storing products inside the vans
2 void vanReport(struct product line[], struct product van1[], struct product van2[], struct product van3[], struct product van4[], struct product van5[], int vanCount[], int totalWeight[]) {
```

```
1   int vanIndex[NUMVAN] = {0};
2
3   printf("\nGenerating van report...\n");
4
5   // Goes through each product in the masterline
6   for (int i=0; i<MASTERSIZE;) {
7       // Adds the current product to van 1 if weight limit has not been exceeded
8       if (totalWeight[0]+line[i].weight <= WEIGHTLIMIT) {
9           van1[vanIndex[0]] = line[i];
10          totalWeight[0] += line[i].weight;
11
12          vanIndex[0]++;
13          vanCount[0]++;
14          i++;
15      }
16      // Adds the current product to van 2 if weight limit has not been exceeded
17      else if (totalWeight[1]+line[i].weight < WEIGHTLIMIT) {
18          van2[vanIndex[1]] = line[i];
19          totalWeight[1] += line[i].weight;
20
21          vanIndex[1]++;
22          vanCount[1]++;
23          i++;
24      }
25      // Adds the current product to van 3 if weight limit has not been exceeded
26      else if (totalWeight[2]+line[i].weight < WEIGHTLIMIT) {
27          van3[vanIndex[2]] = line[i];
28          totalWeight[2] += line[i].weight;
29
30          vanIndex[2]++;
31          vanCount[2]++;
32          i++;
33      }
34      // Adds the current product to van 4 if weight limit has not been exceeded
35      else if (totalWeight[3]+line[i].weight < WEIGHTLIMIT) {
36          van4[vanIndex[3]] = line[i];
37          totalWeight[3] += line[i].weight;
38
39          vanIndex[3]++;
40          vanCount[3]++;
41          i++;
42      }
43      // Adds the current product to van 5 if weight limit has not been exceeded
44      else if (totalWeight[4]+line[i].weight < WEIGHTLIMIT) {
45          van5[vanIndex[4]] = line[i];
46          totalWeight[4] += line[i].weight;
47
48          vanIndex[4]++;
49          vanCount[4]++;
50          i++;
51      }
52      // Alternative case for when no van can store product without exceeding weight limit
53      else {
54          printf("\nWeight limit %d is too small, no possible van report possible");
55          printf("Please increase weight limit or number of vans\n", WEIGHTLIMIT);
56          return;
57      }
58  } // END for
59
60  printf("Van report generated!\n");
61 } // END function
```

\*Image was split up for readability

## Images showing the entire code for the program

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  #include <limits.h>
5
6  // Symbolic Names
7  #define SIZE      50 // Maximum string length
8
9  #define LINESIZE   10 // Sets size of each line
10 #define MASTERSIZE 40 // Sets size for masterline, containing all lines
11 #define BUFFERSIZE 200 // Maximum size of a single line in a file
12
13 #define NUMVAN      5 // How many vans are available
14 #define WEIGHTLIMIT 400 // Maximum weight for one van
15
16 // Structure Tags
17 struct date
18 {
19     int day;
20     int hour;
21     int minute;
22 };
23 struct product
24 {
25     int lineCode;
26     int batchCode;
27     struct date batchDate;
28     int productId;
29     char productName[SIZE];
30     char targetEngineCode[SIZE];
31     int binNumber;
32     int weight;
33     float price;
34 };
35
```

```

36 // Function Signatures
37 void readCSV(FILE *, struct product[], int);
38 void mergesort(struct product[], int, int);
39 void merge(struct product[], int, int, int);
40 void mergelines(struct product[], struct product[], struct product[], struct product[], struct product[]);
41 void binarySearch(struct product[], int);
42 void showStructVar(struct product[], int, int, int);
43 void vanReport(struct product[], struct product[], struct product[], struct product[], struct product[], struct product[], int[], int[]);
44
45 // Main function
46 ~ int main(void) {
47     struct product L1[LINESIZE], L2[LINESIZE], L3[LINESIZE], L4[LINESIZE], masterLine[LINESIZE * 4];
48     struct product van1[LINESIZE], van2[LINESIZE], van3[LINESIZE], van4[LINESIZE], van5[LINESIZE];
49     struct product temp;
50
51     // Create file pointer, and open files for reading
52     FILE *fpLine1 = fopen("line1.csv", "r");
53     FILE *fpLine2 = fopen("line2.csv", "r");
54     FILE *fpLine3 = fopen("line3.csv", "r");
55     FILE *fpLine4 = fopen("line4.csv", "r");
56
57     // Check if all files has been opened successfully
58     if (fpLine1 == NULL || fpLine2 == NULL || fpLine3 == NULL || fpLine4 == NULL) {
59         printf("\nError opening files");
60         printf("\nExiting program...");
61         return 0;
62     }
63     else {
64         printf("Successfully opened files for reading\n");
65     }
66

```

```

67     // Skip past header, for the CSV files
68     fseek(fpLine1, 97, SEEK_SET);
69     fseek(fpLine2, 97, SEEK_SET);
70     fseek(fpLine3, 97, SEEK_SET);
71     fseek(fpLine4, 97, SEEK_SET);
72
73     // Read and store data into structures
74     readCSV(fpLine1, L1, 1);
75     readCSV(fpLine2, L2, 2);
76     readCSV(fpLine3, L3, 3);
77     readCSV(fpLine4, L4, 4);
78
79     // Show contents of CSV
80     showStructVar(L1, 1, LINESIZE, 0);
81     showStructVar(L2, 2, LINESIZE, 0);
82     showStructVar(L3, 3, LINESIZE, 0);
83     showStructVar(L4, 4, LINESIZE, 0);
84
85     printf("\nPress (enter) to continue: "); while(getchar() != '\n');
86
87     // Sort each line using mergesort O(log(n))
88     printf("\nSorting each line...");
89     mergesort(L1, 0, LINESIZE-1);
90     mergesort(L2, 0, LINESIZE-1);
91     mergesort(L3, 0, LINESIZE-1);
92     mergesort(L4, 0, LINESIZE-1);
93     printf("\nFinished sorting\n");
94
95     //showStructVar(struct product line[], int index, int size, int showAll)
96     printf("\nPress (enter) to show weights:\n"); while(getchar() != '\n');
97     showStructVar(L1, 1, LINESIZE, 1);
98     showStructVar(L2, 2, LINESIZE, 1);
99     showStructVar(L3, 3, LINESIZE, 1);
100    showStructVar(L4, 4, LINESIZE, 1);
101

```

```

102 // Merge 4 sorted lines
103 printf("\nMerging lines...");
104 mergelines(L1,L2,L3,L4,masterLine);
105 printf("\nFinished merging lines\n");
106
107 printf("\nPress (enter) to show merged weights"); while(getchar() != '\n');
108 showStructVar(masterLine, 0, MASTERSIZE, 1);
109
110 // Ask user to search for product by weight
111 while (1) {
112     char find;
113
114     printf("\nDo you want to find a product? (Y/N)\n");
115     printf("-> ");
116     scanf("%c", &find);
117
118     if (find == 'y' || find == 'Y') {
119         binarySearch(masterLine, MASTERSIZE);
120     }
121     else if (find == 'n' || find == 'N') {
122         while(getchar() != '\n');
123         break;
124     }
125     else {
126         printf("Input not recognised. Please try again.\n\n");
127     }
128 }
129
130 // Generate van report
131 int vanCount[NUMVAN];
132 int totalWeight[NUMVAN];
133 vanReport(masterLine, van1, van2, van3, van4, van5, vanCount, totalWeight);
134

```



```
135 // Prints report if weight limit hasn't been hit (when all products have been assigned)
136 if ((vanCount[0]+vanCount[1]+vanCount[2]+vanCount[3]+vanCount[4])==MASTERSIZE) {
137     printf("\nPress (enter) to view van report (van 1): "); while(getchar() != '\n');
138     showStructVar(van1, 0, vanCount[0], 0);
139     printf("Number of Items: %d", vanCount[0]);
140     printf("\nTotal Weight:    %d", totalWeight[0]);
141
142     printf("\nPress (enter) to view van report (van 2): "); while(getchar() != '\n');
143     showStructVar(van2, 0, vanCount[1], 0);
144     printf("Number of Items: %d", vanCount[1]);
145     printf("\nTotal Weight:    %d", totalWeight[1]);
146
147     printf("\nPress (enter) to view van report (van 3): "); while(getchar() != '\n');
148     showStructVar(van3, 0, vanCount[2], 0);
149     printf("Number of Items: %d", vanCount[2]);
150     printf("\nTotal Weight:    %d", totalWeight[2]);
151
152     printf("\nPress (enter) to view van report (van 4): "); while(getchar() != '\n');
153     showStructVar(van4, 0, vanCount[3], 0);
154     printf("Number of Items: %d", vanCount[3]);
155     printf("\nTotal Weight:    %d", totalWeight[3]);
156
157     printf("\nPress (enter) to view van report (van54): "); while(getchar() != '\n');
158     showStructVar(van5, 0, vanCount[4], 0);
159     printf("Number of Items: %d", vanCount[4]);
160     printf("\nTotal Weight:    %d", totalWeight[4]);
161 }
162
163
```

```

164 // Closes files
165 fclose(fpLine1);
166 fclose(fpLine2);
167 fclose(fpLine3);
168 fclose(fpLine4);
169
170 printf("\nExiting program...\n");
171 return 0;
172 } // END main()
173
174 // Stores data from csv to structure
175 void readCSV(FILE *fpLine, struct product line[], int x)
176 {
177     char buffer[BUFFERSIZE];
178     char *elementPtr;
179     int loop = 0;
180
181     // Read each line in file
182     while (fgets(buffer, BUFFERSIZE, fpLine) != NULL) {
183         buffer[strlen(buffer) - 1] = '\0'; // Remove '\n' added by fgets
184
185         // Find first character up to the comma ','
186         elementPtr = strtok(buffer, ",");
187         line[loop].lineCode = atoi(elementPtr); // lineCode
188
189         elementPtr = strtok(NULL, ",");
190         line[loop].batchCode = atoi(elementPtr); // batchCode
191
192         elementPtr = strtok(NULL, ",");
193         line[loop].batchDate.day = atoi(elementPtr); // day
194
195         elementPtr = strtok(NULL, ",");
196         line[loop].batchDate.hour = atoi(elementPtr); // hour
197
198         elementPtr = strtok(NULL, ",");
199         line[loop].batchDate.minute = atoi(elementPtr); // minute

```

```

200
201     elementPtr = strtok(NULL, ",");
202     line[loop].productId = atoi(elementPtr); // productId
203
204     elementPtr = strtok(NULL, ",");
205     strcpy(line[loop].productName, elementPtr); // productName
206
207     elementPtr = strtok(NULL, ",");
208     strcpy(line[loop].targetEngineCode, elementPtr); // targetEngineCode
209
210     elementPtr = strtok(NULL, ","); // binNumber
211     line[loop].binNumber = atoi(elementPtr);
212
213     elementPtr = strtok(NULL, ","); // weight
214     line[loop].weight = atoi(elementPtr);
215
216     elementPtr = strtok(NULL, ","); // price
217     line[loop].price = atof(elementPtr);
218
219     loop++;
220 } // END while
221 }
222
223 // Sorts the 4 lines
224 void mergesort(struct product line[LINESIZE], int left, int right) {
225     int middle;
226
227     // Find the middle
228     if (left < right) {
229         middle = left + (right - left) / 2;
230
231         // Sort first and second half
232         mergesort(line, left, middle);
233         mergesort(line, middle+1, right);
234

```

```

235         // Merge the two halves
236         merge(line, left, middle, right);
237     } // END if
238 } // END function
239
240 void merge(struct product line[LINESIZE], int left, int middle, int right) {
241     int i;
242     int j;
243     int k;
244
245     int n1 = middle - left + 1;
246     int n2 = right - middle;
247
248     // Create temporary structures for storing values
249     struct product tempLeft[LINESIZE];
250     struct product tempRight[LINESIZE];
251
252     // Copy data from lines to temp structs
253     for (i=0; i<n1; i++) {
254         tempLeft[i] = line[left + i];
255     }
256     for (i=0; i<n2; i++) {
257         tempRight[i] = line[middle + i + 1];
258     }
259
260     // Merge temp structs
261     i = 0;
262     j = 0;
263     k = left;
264
265
266     while (i<n1 && j<n2) {
267         if (tempLeft[i].weight <= tempRight[j].weight) {
268             line[k] = tempLeft[i];
269             i++;
270         }
271         else {
272             line[k] = tempRight[j];
273             j++;
274         }
275         k++;
276     } // END while
277
278     while (i<n1) {
279         line[k] = tempLeft[i];
280         i++;
281         k++;
282     } // END while
283
284     while (j<n2) {
285         line[k] = tempRight[j];
286         j++;
287         k++;
288     } // END while
289 } // END function
290
291 // Merges all 4 lines
292 void mergeLines(struct product L1[LINESIZE], struct product L2[LINESIZE], struct product L3[LINESIZE], struct product L4[LINESIZE], struct product masterLine[MASTERSIZE]) {
293     int n1 = 0;
294     int n2 = 0;
295     int n3 = 0;
296     int n4 = 0;
297     int min;
298     int minList;
299
300     // Loop through each the nth element of each line, and find the min
301     for (int i=0; i<MASTERSIZE; i++) {
302         min = INT_MAX;
303     }

```

```

304      // Compare current elements of all 4 lines
305      if (L1[n1].weight < min && n1 < LINESIZE) {
306          min = L1[n1].weight;
307          minList = 1;
308      }
309      if (L2[n2].weight < min && n2 < LINESIZE) {
310          min = L2[n2].weight;
311          minList = 2;
312      }
313      if (L3[n3].weight < min && n3 < LINESIZE) {
314          min = L3[n3].weight;
315          minList = 3;
316      }
317      if (L4[n4].weight < min && n4 < LINESIZE) {
318          min = L4[n4].weight;
319          minList = 4;
320      }
321
322      // Append that min value to the master line, then move that line up by 1
323      if (minList == 1) {
324          masterLine[i] = L1[n1];
325          n1++;
326      }
327      else if (minList == 2) {
328          masterLine[i] = L2[n2];
329          n2++;
330      }
331      else if (minList == 3) {
332          masterLine[i] = L3[n3];
333          n3++;
334      }
335      else if (minList == 4) {
336          masterLine[i] = L4[n4];
337          n4++;
338      }
339      } // END for
340  } // END function
341

```

```

342 // Search for car part by weight
343 void binarySearch(struct product line[], int size) {
344     int left;
345     int right;
346     int mid;
347     int target;
348     int targetIndex;
349
350     // Ask user to enter a weight
351     printf("\nEnter weight to find\n");
352     printf("-> ");
353     scanf("%d", &target);
354     while(getchar() != '\n'); //clear input buffer
355
356     // Set boundaries for binary search
357     left = 0;
358     right = size-1;
359     targetIndex = -1;
360
361     while (left <= right) {
362         mid = left + (right-left)/2;
363
364         // Regular binary search
365         if (line[mid].weight == target) {
366             targetIndex = mid;
367             break;
368         }
369         else if (line[mid].weight < target) {
370             left = mid+1; // Search the right half
371         }
372         else {
373             right = mid-1; // Search the left half
374         }
375     } // END while
376
377     // Tell user if weight has been found or not
378     if (targetIndex == -1) {
379         printf("\nTarget weight %d not found\n", target);
380     }
381     else {
382         printf("\nTarget weight %d FOUND\n",target);
383         showStructVar(line, targetIndex, MASTERSIZE, 2);
384     }
385 } // END function

```

```

386
387 ~ void showStructVar(struct product line[], int index, int size, int type) {
388     // This shows structures in CSV format
389 ~     if (type == 0) {
390         printf("\nlineCode,batchCode,Day,Hour,minute,productId,productName,targetEngineCode,binNumber,weight,price\n");
391         for (int i=0; i<size; i++) {
392             printf("%d,", line[i].lineCode);
393             printf("%d,", line[i].batchCode);
394             printf("%d,", line[i].batchDate.day);
395             printf("%d,", line[i].batchDate.hour);
396             printf("%d,", line[i].batchDate.minute);
397             printf("%d,", line[i].productId);
398             printf("%s,", line[i].productName);
399             printf("%s,", line[i].targetEngineCode);
400             printf("%d,", line[i].binNumber);
401             printf("%d,", line[i].weight);
402             printf("%.2f\n", line[i].price);
403         }
404     }
405     // This shows the weights for a line
406 ~     else if (type == 1) {
407         if (index==0) {
408             printf("Weights for masterLine (all 4 lines merged):\n");
409         }
410         else {
411             printf("Weights for line %d: ", index);
412         }
413         for (int i=0; i<size; i++) {
414             printf("%d ", line[i].weight);
415         }
416         printf("\n");
417     }
418     // Show info for specific index in structure array
419 ~     else if (type == 2) {
420         printf("Line code:          %d\n", line[index].lineCode);
421         printf("Batch code:          %d\n", line[index].batchCode);
422         printf("Day:                  %d\n", line[index].batchDate.day);
423         printf("Hour:                 %d\n", line[index].batchDate.hour);
424         printf("Minute:               %d\n", line[index].batchDate.minute);
425         printf("Product ID:           %d\n", line[index].productId);
426         printf("Product name:         %s\n", line[index].productName);
427         printf("Target engine code: %s\n", line[index].targetEngineCode);
428         printf("Bin number:           %d\n", line[index].binNumber);
429         printf("Weight:                %d\n", line[index].weight);
430         printf("price:                %.2f\n", line[index].price);

```

```

386
387 void showStructVar(struct product line[], int index, int size, int type) {
388     // This shows structures in CSV format
389     if (type == 0) {
390         printf("\nlineCode,batchCode,Day,Hour,minute,productId,productName,targetEngineCode,binNumber,weight,price\n");
391         for (int i=0; i<size; i++) {
392             printf("%d,", line[i].lineCode);
393             printf("%d,", line[i].batchCode);
394             printf("%d,", line[i].batchDate.day);
395             printf("%d,", line[i].batchDate.hour);
396             printf("%d,", line[i].batchDate.minute);
397             printf("%d,", line[i].productId);
398             printf("%s,", line[i].productName);
399             printf("%s,", line[i].targetEngineCode);
400             printf("%d,", line[i].binNumber);
401             printf("%d,", line[i].weight);
402             printf("%.2f\n", line[i].price);
403         }
404     }
405     // This shows the weights for a line
406     else if (type == 1) {
407         if (index==0) {
408             printf("Weights for masterLine (all 4 lines merged):\n");
409         }
410         else {
411             printf("Weights for line %d: ", index);
412         }
413         for (int i=0; i<size; i++) {
414             printf("%d ", line[i].weight);
415         }
416         printf("\n");
417     }

```

```

418 // Show info for specific index in structure array
419 else if (type == 2) {
420     printf("Line code:      %d\n", line[index].lineCode);
421     printf("Batch code:     %d\n", line[index].batchCode);
422     printf("Day:           %d\n", line[index].batchDate.day);
423     printf("Hour:          %d\n", line[index].batchDate.hour);
424     printf("Minute:         %d\n", line[index].batchDate.minute);
425     printf("Product ID:      %d\n", line[index].productId);
426     printf("Product name:     %s\n", line[index].productName);
427     printf("Target engine code: %s\n", line[index].targetEngineCode);
428     printf("Bin number:       %d\n", line[index].binNumber);
429     printf("Weight:          %d\n", line[index].weight);
430     printf("Price:           %.2f\n", line[index].price);
431 }
432 } // END function
433
434 // Create report for storing products inside the vans
435 void vanReport(struct product line[], struct product van1[], struct product van2[], struct product van3[], struct product van4[], struct product van5[], int vanCount[], int totalWeight[]) {
436     int vanIndex[NUMVAN] = {0};
437
438     printf("\nGenerating van report...\n");
439 }

```



```

440 // Goes through each product in the masterline
441 ~ for (int i=0; i<MASTERSIZE;) {
442     // Adds the current product to van 1 if weight limit has not been exceeded
443     ~ if (totalWeight[0]+line[i].weight <= WEIGHTLIMIT) {
444         van1[vanIndex[0]] = line[i];
445         totalWeight[0] += line[i].weight;
446
447         vanIndex[0]++;
448         vanCount[0]++;
449         i++;
450     }
451     // Adds the current product to van 2 if weight limit has not been exceeded
452     ~ else if (totalWeight[1]+line[i].weight < WEIGHTLIMIT) {
453         van2[vanIndex[1]] = line[i];
454         totalWeight[1] += line[i].weight;
455
456         vanIndex[1]++;
457         vanCount[1]++;
458         i++;
459     }
460     // Adds the current product to van 3 if weight limit has not been exceeded
461     ~ else if (totalWeight[2]+line[i].weight < WEIGHTLIMIT) {
462         van3[vanIndex[2]] = line[i];
463         totalWeight[2] += line[i].weight;
464
465         vanIndex[2]++;
466         vanCount[2]++;
467         i++;
468     }
469     // Adds the current product to van 4 if weight limit has not been exceeded
470     ~ else if (totalWeight[3]+line[i].weight < WEIGHTLIMIT) {
471         van4[vanIndex[3]] = line[i];
472         totalWeight[3] += line[i].weight;
473
474         vanIndex[3]++;
475         vanCount[3]++;
476         i++;
477     }
478     // Adds the current product to van 5 if weight limit has not been exceeded
479     ~ else if (totalWeight[4]+line[i].weight < WEIGHTLIMIT) {
480         van5[vanIndex[4]] = line[i];
481         totalWeight[4] += line[i].weight;
482
483         vanIndex[4]++;
484         vanCount[4]++;
485         i++;
486     }
487     // Alternative case for when no van can store product without exceeding weight limit
488     else {
489         printf("\nWeight limit %d is too small, no possible van report possible");
490         printf("Please increase weight limit or number of vans\n", WEIGHTLIMIT);
491         return;
492     }
493 } // END for
494
495 printf("Van report generated!\n");
496 } // END function

```