Fundamentals of Data Structures
Laboratory Projects Report

Project 3: Cars on Campus

Group 07
Programmer: Hao Xiangpeng
Tester: Yang Kefan
Report Writer: Li Haipeng

Date: 2016-12-24

Content

Chapter 1: Introduction

1.1 Background

Zhejiang University has 8 campuses and a lot of gates. From each gate we can collect the in/out times and the plate numbers of the cars crossing the gate.

1.2 Problem description

The information we have:

- The record of cars crossing the gate
  - **Plate number:** a string of 7 English capital letters or 1-digit numbers
  - **Time when it get in:** represents the time point in a day by “hour: minute: second”
  - **Time when it get out:** represents the time point in a day by “hour: minute: second”

The purpose we want to achieve:

- **Task 1:** Given a certain time, the number of cars parking on campus
- **Task 2:** At the end of the day, the cars that have parked for the longest time period.

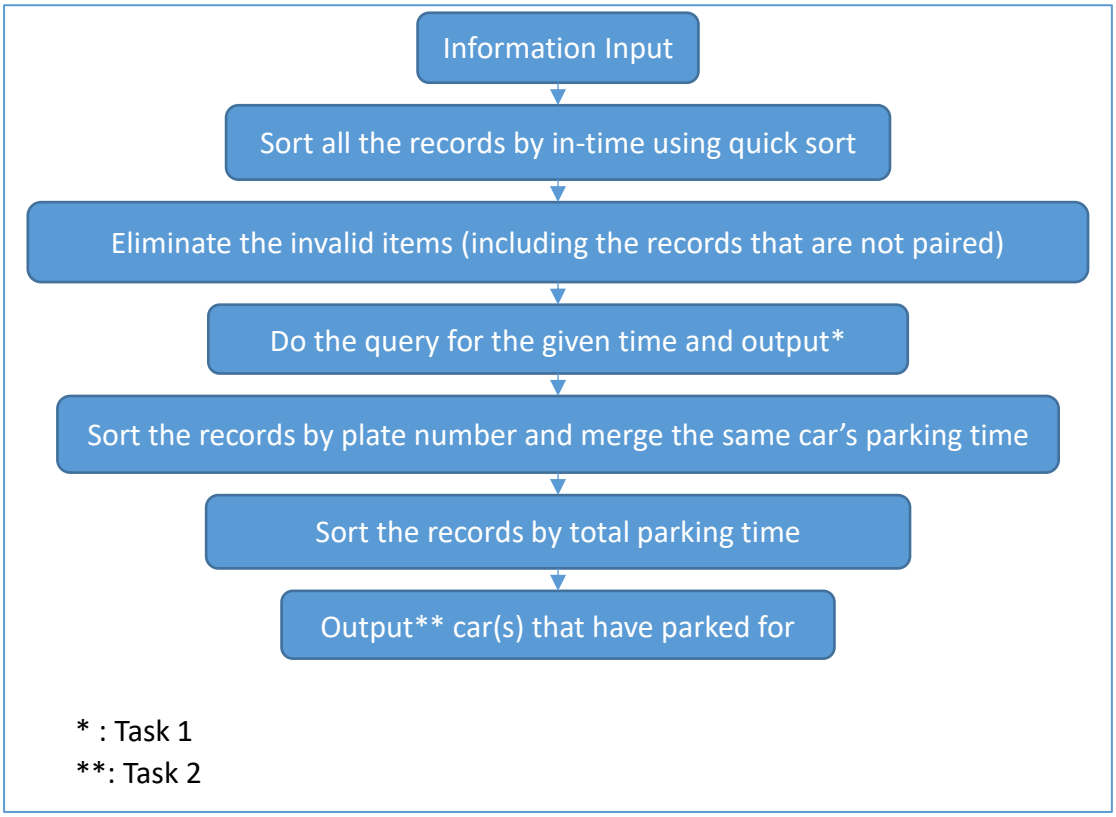
1.3 Sample problem&solution

Input:

16 7
JH007BD 18:00:01 in
ZD00001 11:30:08 out
DB8888A 13:00:00 out
ZA3Q625 23:59:50 out
ZA133CH 10:23:00 in
ZD00001 04:09:59 in
JH007BD 05:09:59 in
ZA3Q625 11:42:01 out
JH007BD 05:10:33 in
ZA3Q625 06:30:50 in
JH007BD 12:23:42 out
ZA3Q625 23:55:00 in
JH007BD 12:24:23 out
ZA133CH 17:11:22 out
JH007BD 18:07:01 out
DB8888A 06:30:50 in
05:10:00
06:30:50
11:00:00
12:23:42
14:00:00
18:00:00
23:59:00
Output:
1
4
5
2
1
0
1
JH007BD ZD00001 07:20:09

Chapter 2: Algorithm Specification

2.1 Flow Chart



2.2 Correctness of Algorithm

- We use function block "Sort all the records by in-time using quick sort" to make query easier.
- In function block "Eliminate the invalid items", all invalid records are deleted so that we get the right result.
- When we do the query, we pick the records of which the in-time is earlier than the given time and the out-time is later than the given time. So we can find all the parking cars
- In some cases, a car enters and leaves the campus more than one time, so we have to merge them to get the total time using "Sort the records by plate number and merge the same car's parking time" function block.
- At last, we Sort the records by total parking time and output the cars that parking for longest time. Mission completed.

2.3 Data Structure

- Records of cars

```
struct OneItem {
    long long plate_number;
    int time;
    int status;
    int valid;
    int stay_time;
};
```

  - plate_number： we convert the string to long long type so that we can easily sort the data
  - time: equal to “hour * 3600 + minute * 60 + second”
  - status : 1 represents “in” while -1 represents “out”
  - valid: represent whether a record is valid or not
  - stay_time: the total time of parking

2.4 Key Algorithm

- Quick sort (We call the qsort() function in <stdlib.h> and write functions that compares two elements.)
  - Compare by in-time

```
int comp_by_in_time(const void *a, const void *b) {
    const struct OneItem *aa = (struct OneItem *) a;
    const struct OneItem *bb = (struct OneItem *) b;//cast the void* to target type
    return aa->time - bb->time;// compare by their in time
}
```

  - Compare by stay-time

```
int comp_by_stay_time(const void *a, const void *b) {
    const struct OneItem *aa = (struct OneItem *) a;
    const struct OneItem *bb = (struct OneItem *) b;
    if (bb->stay_time != aa->stay_time)//cast the void* to target type
        return bb->stay_time - aa->stay_time;//compare by their in time
    else// if they have the same in time
        return bb->plate_number < aa->plate_number;//sort by their plate_number
}
```

  - Compare by plate number

```
int comp_by_stay_plate(const void *a, const void *b) {
    const struct OneItem *aa = (struct OneItem *) a;
    const struct OneItem *bb = (struct OneItem *) b;//cast the void* to target
type
```

```
        return bb->plate_number > aa->plate_number;// sort by their plate_number
    }
```

- The way to call them:
  - qsort (void* base, size_t num, size_t size, int (*compar)(const void*,const void*));
  - Parameters
    - base
      - Pointer to the first object of the array to be sorted, converted to a void*.
    - num
      - Number of elements in the array pointed to by base.
      - size_t is an unsigned integral type.
    - size
      - Size in bytes of each element in the array.
      - size_t is an unsigned integral type.
    - compar
      - Pointer to a function that compares two elements.

- Eliminating the invalid items

```
struct OneItem result_items[10000];//since we do not now how many valid item
int result_number = 0;//mark the valid number
for (int i = 0; i < RECORD; i++) {
    if (all_items[i].valid == 0) {
        continue;//already marked
    } else if (all_items[i].valid == 1) {
        result_items[result_number].stay_time = all_items[i].stay_time;
        result_items[result_number].plate_number = all_items[i].plate_number;
        result_items[result_number].status = all_items[i].status;
        result_items[result_number].time = all_items[i].time;
        result_number += 1;//copy the value to the target
        continue;
    }
    if (all_items[i].status == -1) {
        all_items[i].valid = 0;//if out first invalid then
        continue;
    }        //valid=-1 and status = 1
    int valid_flag = 0;
    for (int j = i + 1; j < RECORD; j++) {
        if (all_items[i].valid != -1)
            continue;
        if (all_items[i].plate_number == all_items[j].plate_number &&
            all_items[i].status == 1 && all_items[j].status == -1) {
            all_items[i].valid = 1;// if they have the same plate number
            all_items[j].valid = 1;// and they have different status, they got
paired
```

```
                all_items[i].stay_time = all_items[j].time - all_items[i].time;
                all_items[j].stay_time = 0;// set the stay time, in to the correct out
to zero

                result_items[result_number].stay_time = all_items[i].stay_time;
                result_items[result_number].plate_number = all_items[i].plate_number;
                result_items[result_number].status = all_items[i].status;//copy the
source information
                result_items[result_number].time = all_items[i].time;//to the target
one
                result_number += 1;//increment the counter
                valid_flag = 1;
                break;
            } else if (all_items[i].plate_number == all_items[j].plate_number &&
                    all_items[i].status == 1 && all_items[j].status == 1) {
                all_items[i].valid = 0;//if we encountered a same status
                valid_flag = 1;// it should be marked as illegal
                break;
            }
        }
        if (valid_flag == 0) {
            all_items[i].valid = 0;//not found and mark illegal
        }
    }
```

- Merge the stay-time

```
    for (int i = 0; i < result_number - 1; i++) {//merge the same car's plate time
        if (result_items[i].plate_number == result_items[i + 1].plate_number) {
            result_items[i + 1].stay_time += result_items[i].stay_time;
            result_items[i].stay_time = 0;//set one of the source to zero
        }
    }
```

  - Before merging we use quick sort to rearrange them by plate number
  - For the records with the same plate number, add them together

Chapter 3: Testing Results

In the following test cases, a script is written in order to generate random input.

```
//test.js
"use strict";
const fs = require("fs");
let shuffle = (a)=>{
```

```
    for (let i = a.length; i; i--) {
        let j = Math.floor(Math.random() * i);
        [a[i - 1], a[j]] = [a[j], a[i - 1]];
    }
}
let randomString = (length)=>{
    let text = "";
    let possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    for( let i=0; i < length; i++ )
        text += possible.charAt(Math.floor(Math.random() * possible.length));
    return text;
}
let pad = (num, n)=> {
    var len = num.toString().length;
    while(len < n) {
        num = "0" + num;
        len++;
    }
    return num;
}
const recordCount = 16,
    queryCount = 5;
let record= [];
for(let i=0;i<recordCount/2;i++){
    let car_plate=randomString(7);
    let time = [];
    for(let j=0;j<2;j++) {
        time.push(`${pad(Math.floor(Math.random() *
24),2)}:${pad(Math.floor(Math.random() * 60),2)}:${pad(Math.floor(Math.random() *
60),2)}`)
    }
    for(let j=0;j<2;j++) {
        record.push({
            state:time[j]<time[1^j]?"in":"out",
            car_plate: car_plate,
            time: time[j]
        });
    }
}
let query=[];
for(let i=0;i<queryCount;i++){
```

```
    query.push(`${pad(Math.floor(Math.random() * 24),2)}:${pad(Math.floor(Math.random()
* 60),2)}:${pad(Math.floor(Math.random() * 60),2)}`)
}
shuffle(record);
query.sort();
let fileData = `${recordCount} ${queryCount}\n`;
for(let i=0;i<recordCount;i++){
    fileData +=`${record[i].car_plate} ${record[i].time} ${record[i].state}\n`;
}
for(let i=0;i<queryCount;i++){
    fileData += `${query[i]}\n`;
}
fs.writeFile('test',fileData,(err)=>{
    if(err) console.log(err);
})
```

The generated test cases and the program results are listed below:

Test Case 1:

```
16 5
ZXNHC69 23:52:03 out
0HX98HK 03:22:57 out
7MDH45V 07:40:23 in
QWZRLAT 00:52:01 in
MJ0FRFG 01:30:33 in
ZV7EVKL 17:25:12 out
MJ0FRFG 14:10:37 out
7MDH45V 11:03:46 out
ZV7EVKL 13:56:24 in
7BC54ZL 04:05:54 out
8U2SR4G 13:55:31 out
8U2SR4G 11:32:02 in
0HX98HK 00:58:16 in
7BC54ZL 03:00:40 in
QWZRLAT 23:11:32 out
ZXNHC69 07:28:15 in
03:30:08
05:39:20
15:04:29
18:23:30
18:38:21
```

```
3
2
3
2
2
QWZRLAT 22:19:31
```

Test Case 2:

```
16 5
UTKS6HS 14:18:43 in
3D8M34V 05:08:35 in
GYDEBUK 01:21:38 in
3D8M34V 06:05:51 out
QNUNKTB 08:23:39 out
Z1AU6Z7 03:23:32 in
1UUZQGH 04:04:03 out
EH0CWEO 06:49:59 out
EH0CWEO 06:09:06 in
Z1AU6Z7 20:41:38 out
D9BVJ3N 22:46:19 out
UTKS6HS 17:48:46 out
1UUZQGH 03:57:25 in
D9BVJ3N 16:57:09 in
QNUNKTB 06:12:25 in
GYDEBUK 03:41:04 out
00:05:33
01:05:01
08:28:23
11:18:48
12:19:02
```

```
0
0
1
1
1
Z1AU6Z7 17:18:06
```

Test Case 3:

```
16 5
2TGL4OA 06:12:04 in
50I8RM8 12:40:46 out
LTBB80C 15:06:10 out
J8S41QM 12:28:38 out
LTBB80C 09:03:57 in
2TGL4OA 08:36:05 out
1KH9NOS 05:17:39 in
5BOE7D7 16:47:02 out
50I8RM8 08:09:43 in
4NLPE7G 21:23:31 out
10AIKQF 16:37:19 out
J8S41QM 04:33:09 in
1KH9NOS 18:41:50 out
5BOE7D7 09:31:39 in
10AIKQF 10:39:21 in
4NLPE7G 19:15:44 in
03:13:38
06:21:03
08:28:29
15:43:40
22:57:58
```

```
0
3
4
3
0
1KH9NOS 13:24:11
```

Test Case 4:

```
16 5
FYXHX8C 05:01:31 in
7JX8BJ5 14:18:54 in
JHUF2QW 02:02:58 in
FUKF90L 08:06:49 in
OOTIWWQ 23:42:33 out
JHUF2QW 17:11:24 out
GNNB8X8 07:47:29 out
T0N53PB 23:40:39 out
PMRSZPM 07:33:36 in
FYXHX8C 23:02:56 out
GNNB8X8 05:40:57 in
OOTIWWQ 00:25:57 in
T0N53PB 20:58:22 in
PMRSZPM 10:43:43 out
FUKF90L 14:45:34 out
7JX8BJ5 19:47:40 out
04:02:25
06:07:22
06:18:21
10:03:44
17:40:33
```

Test Case 5

```
16 5
GPCMU2B 13:49:17 out
GPCMU2B 04:40:51 in
68IEF4G 14:01:02 out
2T4EVOB 08:34:50 in
HKT2WZ1 19:48:55 out
580PCP5 05:02:29 in
2V25ZH6 00:02:24 in
ZOT6ZKM 03:18:28 in
V88ICRG 05:46:20 in
580PCP5 13:27:12 out
2T4EVOB 21:28:52 out
ZOT6ZKM 07:09:58 out
2V25ZH6 00:28:28 out
68IEF4G 00:16:37 in
V88ICRG 13:39:35 out
HKT2WZ1 15:07:10 in
05:40:09
10:51:00
16:50:18
17:26:14
20:44:31
```

The five test cases listed above are all generated randomly. Passing these test cases can convincingly prove the validity of this program in most common cases. However, hand-made test cases are needed to test the program's behavior in edge situations. The next two test cases are to test program's behavior when multiple solutions exist.

Test Case 6

```
8 4
KW0SRAH 00:09:00 in
WDPKQEO 22:30:03 out
WDPKQEO 06:21:32 in
K8ZY1B7 04:17:34 in
KW0SRAH 18:09:00 out
RDGYCE2 16:15:12 in
RDGYCE2 22:29:43 out
K8ZY1B7 22:17:34 out
18:14:22
20:58:46
21:37:36
23:24:06
```

```
3
3
3
0
K8ZY1B7 KW0SRAH 18:00:00
```

:

Test Case 7:

```
8 4
TOXOVHO 08:06:46 in
4UO55XZ 18:56:42 in
GZOSMO3 21:56:23 out
LACMB2H 02:10:52 in
TOXOVHO 12:06:46 out
LACMB2H 06:10:52 out
GZOSMO3 17:56:23 in
4UO55XZ 22:56:42 out
09:41:20
12:10:04
15:27:57
18:54:07
```

```
1
0
0
1
4UO55XZ GZOSMO3 LACMB2H TOXOVHO 04:00:00
```

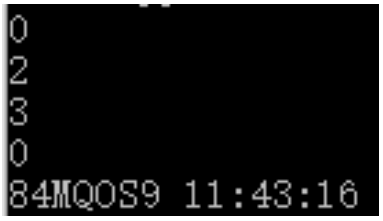
The next test case is to test program's behavior when only one car is paired in the input.

Test Case 8:

```
8 4
7I2WW8L 20:39:19 in
P68TJBK 16:50:16 out
P68TJBK 16:08:09 in
HF5BMTC 20:09:19 in
7I2WW8L 09:34:30 in
TZRBOHC 21:56:37 in
HF5BMTC 04:56:11 in
TZRBOHC 22:52:12 in
11:10:00
14:23:41
16:46:34
18:46:41
```

The last test case is to test program's behavior when the query time is exactly at the point when a car is moving in/out. The result under such situations are not mentioned in the problem specification. But we assume both moving in and out happen before the query time.
Test Case 9:

```
8 4
AS7DYO7 05:50:08 in
3YZBF4D 06:36:08 out
84MQOS9 05:39:46 in
84MQOS9 17:23:02 out
YDJALJ5 07:30:23 in
YDJALJ5 18:08:46 out
3YZBF4D 02:15:56 in
AS7DYO7 10:34:22 out
00:16:46
05:39:46
07:30:23
18:08:46
```



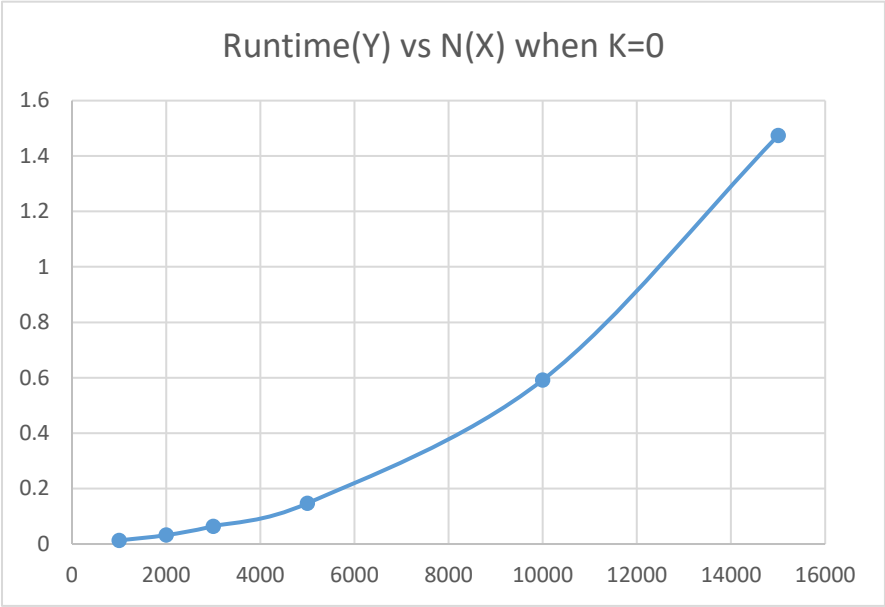
Chapter 4: Analysis and Comments

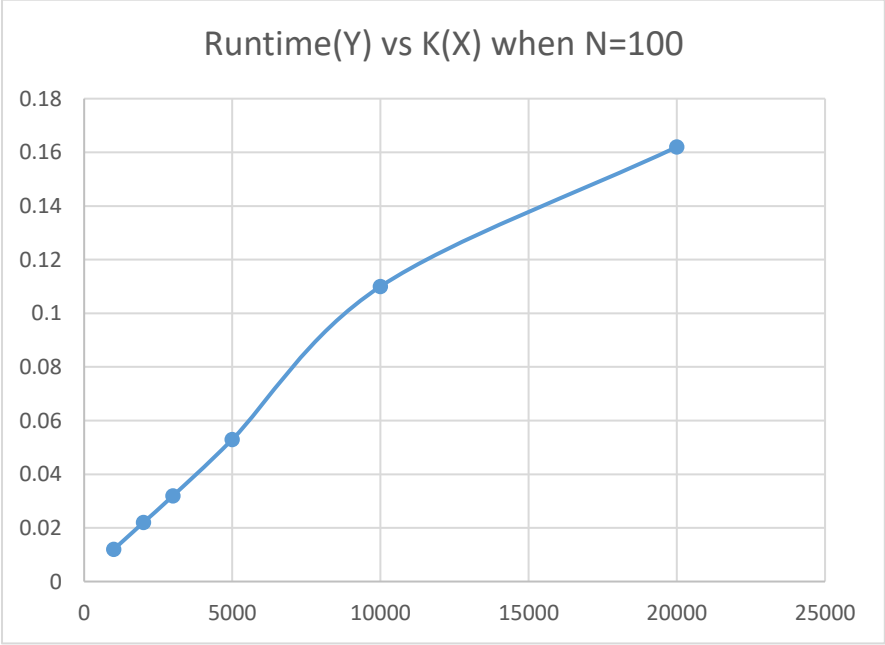
This program has successfully passed all the test cases above, therefore its validity has been proved both in common cases and in edge situations.

We denote **N** as the number of records and **K** as the number of queries. In order to analyze the complexity of this program, we can divide it into several parts and analyze the complexity of each part respectively.

Input	Quick sort(average complexity)	Pair in&out	Find plate number	Output	Total
O(N+K)	O(NlogN)	O(N^2)	O(N)	O(K)	O(N^2 + K)

As the form illustrates, we expect the time complexity of the whole program to be **O(N^2 + K)**. In order to prove this conclusion, we do mass data test for **N** and **K** respectively(so we can observe more cleverly).





The graphs above are drawn using1000,2000,3000,5000,10000 and 15000 lines data. The result is very consistent with our expectation. So we can conclude the average time complexity is **O(N^2 + K)**.

Square time complexity is acceptable in this problem, but there is still room for improvement. When finding the longest stay time, we can simply find out the maximal element(**O(N)**) instead of using a quick sort(**O(NlogN)**). And also, in order to reduce the time complexity of pairing the in and out record, we can store the records in an array and use a hash function to locate the element. Then only constant time is needed to find out an element with certain plate number. The ideal time complexity of the whole program would be **O(NlogN + K)** in this way.

Appendix: Source Code

```
1  //
2  // Created by Haoxiangpeng on 12/20/2016.
3  // Created and tested as per C99 standard
4  //
5  //Copyright (c) <year> <copyright holders>
6
7  //Permission is hereby granted, free of charge,
8  // to any person obtaining a copy of this software and associated documentation files
   (the "Software"),
9  // to deal in the Software without restriction, including without limitation the
   rights to
10 // use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
   the Software,
11 // and to permit persons to whom the Software is furnished to do so, subject to the
   following conditions:
12 //The above copyright notice and this permission notice shall be included in all
   copies or substantial portions of the Software.
13 //THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
14 // INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
15 // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
16 // IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
   DAMAGES OR OTHER LIABILITY,
17 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
   CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
18
19
20 #include <stdio.h>
21 #include <stdlib.h>
22 #include <string.h>
23
24
25 struct OneItem {
26     long long plate_number;
```

```
    int time;
    int status;// in:1 out:0
    int valid;
    int stay_time;
};

long long str_to_long(char *str) {
    long long num = 1;// convert the string to long long type
    int i = 0;// so that we can easily sort the data
    while (str[i]) {
        num *= 100;
        num += str[i];
        i++;
    }// to avoid conflict
    return num;
}

char *long_to_string(long long num) {
    char *plate_number;// convert the long long type back to the string
    plate_number = malloc(sizeof(char) * 7);
    for (int i = 6; i >= 0; i--) {
        int temp = num % 100;
        plate_number[i] = (char) temp;
        num /= 100;// to avoid the conflict
    }
    plate_number[7] = '\0';// to mark the stop of the plate_number
    return plate_number;
}

int comp_by_in_time(const void *a, const void *b) {
    const struct OneItem *aa = (struct OneItem *) a;
    const struct OneItem *bb = (struct OneItem *) b;//cast the void* to target type
    return aa->time - bb->time;// compare by their in time
}

int comp_by_stay_time(const void *a, const void *b) {
    const struct OneItem *aa = (struct OneItem *) a;
    const struct OneItem *bb = (struct OneItem *) b;
    if (bb->stay_time != aa->stay_time)//cast the void* to target type
        return bb->stay_time - aa->stay_time;//compare by their in time
    else// if they have the same in time
        return bb->plate_number < aa->plate_number;//sort by their plate_number
}

```

```c
int comp_by_stay_plate(const void *a, const void *b) {
    const struct OneItem *aa = (struct OneItem *) a;
    const struct OneItem *bb = (struct OneItem *) b;//cast the void* to target type
    return bb->plate_number > aa->plate_number;// sort by their plate_number
}


int main() {
    unsigned int RECORD, QURERY;

    struct OneItem *all_items;
    int *all_queries;

    scanf("%d%d", &RECORD, &QURERY);
    all_items = (struct OneItem *) malloc(sizeof(struct OneItem) * RECORD);//all the records
    all_queries = (int *) malloc(sizeof(int) * QURERY);// all the queries
    for (int i = 0; i < RECORD; i++) {
        char plate_number[10], tem_status[5];
        int hour, minute, second;
        scanf("%s %d:%d:%d %s", plate_number, &hour, &minute, &second, tem_status);//read from stdin
        all_items[i].plate_number = str_to_long(plate_number);
        all_items[i].time = hour * 3600 + minute * 60 + second;
        all_items[i].status = strcmp(tem_status, "in") ? -1 : 1;//if in then 1 else -1
        all_items[i].valid = -1;//initialize the data
        all_items[i].stay_time = 0;//initialize the data
    }
    for (int i = 0; i < QURERY; i++) {
        int hour, minute, second;
        scanf("%d:%d:%d", &hour, &minute, &second);
        all_queries[i] = hour * 3600 + minute * 60 + second;// convert the time to int, so that we can sort it
    }
    qsort(all_items, RECORD, sizeof(struct OneItem), comp_by_in_time);// Sort all the records per in time

    // eliminate the invalid items
    struct OneItem result_items[10000];//since we do not now how many valid item
    int result_number = 0;//mark the valid number
    for (int i = 0; i < RECORD; i++) {
        if (all_items[i].valid == 0) {
            continue;//already marked
```

```
110         } else if (all_items[i].valid == 1) {
111             result_items[result_number].stay_time = all_items[i].stay_time;
112             result_items[result_number].plate_number = all_items[i].plate_number;
113             result_items[result_number].status = all_items[i].status;
114             result_items[result_number].time = all_items[i].time;
115             result_number += 1;//copy the value to the target
116             continue;
117         }
118         if (all_items[i].status == -1) {
119             all_items[i].valid = 0;//if out first invalid then
120             continue;
121         }        //valid=-1 and status = 1
122         int valid_flag = 0;
123         for (int j = i + 1; j < RECORD; j++) {
124             if (all_items[i].valid != -1)
125                 continue;
126             if (all_items[i].plate_number == all_items[j].plate_number &&
127                 all_items[i].status == 1 && all_items[j].status == -1) {
128                 all_items[i].valid = 1;// if they have the same plate number
129                 all_items[j].valid = 1;// and they have different status, they got
    paired
130                 all_items[i].stay_time = all_items[j].time - all_items[i].time;
131                 all_items[j].stay_time = 0;// set the stay time, in to the correct
    out to zero
132 
133                 result_items[result_number].stay_time = all_items[i].stay_time;
134                 result_items[result_number].plate_number = all_items[i].plate_number;
135                 result_items[result_number].status = all_items[i].status;//copy the
    source information
136                 result_items[result_number].time = all_items[i].time;//to the target
    one
137                 result_number += 1;//increment the counter
138                 valid_flag = 1;
139                 break;
140             } else if (all_items[i].plate_number == all_items[j].plate_number &&
141                     all_items[i].status == 1 && all_items[j].status == 1) {
142                 all_items[i].valid = 0;//if we encountered a same status
143                 valid_flag = 1;// it should be marked as illegal
144                 break;
145             }
146         }
147         if (valid_flag == 0) {
148             all_items[i].valid = 0;//not found and mark illegal
149         }
```

```
150     }
151
152     // Print the query result
153     int car_number = 0, query_index = 0, result_index = 0;
154     while (query_index < QURERY) {
155         if (result_items[result_index].time <= all_queries[query_index]) {//earlier
    than queried,status can only be 1 or -1
156             car_number += (result_index < result_number ?
    result_items[result_index].status : 0);//if reached the end, status shall not change
157             result_index++;// go to next car item
158         } else {
159             printf("%d\n", car_number);//out put the status number
160             query_index++;//next query
161         }
162     }
163
164     // Sort by plate number first, so that we can merge the time with O(N)
165     qsort(result_items, (size_t) result_number, sizeof(struct OneItem),
    comp_by_stay_plate);
166     for (int i = 0; i < result_number - 1; i++) {//merge the same car's plate time
167         if (result_items[i].plate_number == result_items[i + 1].plate_number) {
168             result_items[i + 1].stay_time += result_items[i].stay_time;
169             result_items[i].stay_time = 0;//set one of the source to zero
170         }
171     }
172
173     // Sort, the first key is stay time, second key is plate number
174     qsort(result_items, (size_t) result_number, sizeof(struct OneItem),
    comp_by_stay_time);
175
176     int index_number = 0;
177     while (1) {//output the plate number
178         printf("%s ",
    long_to_string(result_items[index_number].plate_number));//judge if we have two same
    entry
179         if (result_items[index_number].stay_time != result_items[index_number +
    1].stay_time) break;
180         index_number++;//go to next
181     }
182     printf("%02d:%02d:%02d", result_items[0].stay_time / 3600,
    (result_items[0].stay_time % 3600) / 60,
183             result_items[0].stay_time % 3600 % 60);//out put the top time
184 }
```

Declaration

We hereby declare that all the work done in this project titled " Cars on Campus " is of our independent effort as a group.

Duty Assignments:

Programmer: Hao Xiangpeng
Tester: Yang Kefan
Report Writer: Li Haipeng