Given the constraints of the program as follow:

- The input are of the form of arrays of objects (as seen in info.js).
- There are only 3 inputs given to the program: `Passing_score`, `Contact`, and `Test_info`.

Time complexity analysis:

```
16  //storing all test records, along with contact info of test takers into allTest_info
17  for(let testDay of Test_info){
18      testDay.scores.map(result =>{
19          result.day = counter
20          result.phone = Contact.find(contact => contact.name == result.name).phone
21      })
22      allTest_info.push(...testDay.scores)
23      counter++
24  }
```

This part uses `find()` method to get the contact of a particular student: O(N). As it is being run in the for-loop that goes through each day, the total time complexity is O(KN).

The following for-loop contains two major operations. First, the sorting of elements in the queue.

```
39  for(let i=0; i<Total_days; i++){
40      //provide short-circuiting if all students have been called
41      if(repeat_check.length === Total_students) break
42
43      let todayTest = allTest_info.filter(item => item.day === i+1)
44      let callLimit = Test_info[i].callLimit
45      queue.push(...todayTest)
46
47      //appending the failing performance to be called
48      //sorting mid-operation to follow the guideline of call sequence
49      queue.sort(function(a, b){
50          //First: sort by score (lowest first)
51          if(a.score > b.score) return 1
52          else if(a.score < b.score) return -1
53          else if(a.score == b.score){
54
55              //Second: sort by date (most recent first)
56              if(a.day > b.day) return -1
57              else if(a.day < b.day) return 1
58              else if(a.day == b.day){
59
60                  //Last: sort by name alphabetically (A-Z)
61                  if(a.name > b.name) return 1
62                  else return -1
63              }
64
65          }
66      })
```

The `sort()` function is needed in order to sort the performance recorded in the queue upon each day due to the call sequence guidelines in the question prompt. For example, after going through the first day of the record and starting to append the information of the second day, the program needs this sorting to help ensure the students with the same lowest score will be called by the order of test taken date as specified.

The maximum size of queue is KN, which is when every student takes an exam every day and fails all. As `sort()` is of O(nlogn), thus we have O(KNlogKN) for the sorting in our program. Hence, the overall time complexity of this sorting operation inside this for-loop is $O(K^2NlogKN)$.

```
74      while(callLimit > 0){
75          if(queue.length > 0 && !repeat_check.includes(queue[0].name)){
76              const callee = queue.shift()
77              callee.day = dateLookup[callee.day - 1]
78              temp.push(callee)
79              repeat_check.push(callee.name)
80          }
81          else{
82              break
83          }
84          callLimit--
85      }
86      final_ans.push({[dateLookup[i]]: temp})
87  }
```

This part is also under the aforementioned for-loop. Its time complexity due to `call limit` and being inside this for-loop is O(TK).

As O(KN) and O(TK) are asymptotically less than $O(K^2NlogKN)$, the overall time complexity of the program is $O(K^2NlogKN)$.

Footnote:

It is important to note that the time complexity above is purely an upper bound, for instance, where we upper-bounded the value of `queue.length` to be KN meaning that everyone takes a test every day and always fails, while the teacher also does not make any call. The actual program shall be of significantly better time complexity.

Suggestion for improvement:

If this implementation is not for web applications, C++ may be a better choice to program with. In C++, we may use `map` to maintain a structured set of data while being able to achieve better time

complexity. `Find` operation can be done in O(1), and the elements will always be ordered automatically following how we structure the 'key' and 'value,' hence we may not need to sort the queue upon all insertions. For example, we may use `map<string,<string, number>>` to store date, name, and score, ordered from left to right. However, it is important to note that it will be significantly more difficult in terms of working with the input and how to format it due to C++'s non-dynamic behavior.

Space complexity analysis:

```
var dateLookup = []
for(let i of Test_info){
    dateLookup.push(i.testDate)
}


var allTest_info = []  //store all test records
```

`dateLookup[]` → O(K). `allTest_info[]` → O(KN), as there could be at most KN test taken throughout the period that failed.

```
var queue = []  //queue will store the test performances that are to be called
var final_ans = [] //storing all calls to be made and their information
var repeat_check = [] //this array provides look-up table to avoid calling the same student more than once
```

Similarly, `queue[]` and `final_ans[]` will have O(KN), while `repeat_check[]` has O(N).

```
    var temp = []
    // make 'T' no. of calls in each day by dequeuing 'T' times
    while(callLimit > 0){
        if(queue.length > 0 && !repeat_check.includes(queue[0].name)){
            const callee = queue.shift()
            callee.day = dateLookup[callee.day - 1]
            temp.push(callee)
            repeat_check.push(callee.name)
        }
        else{
            break
        }
        callLimit--
    }
```

Lastly, `temp[]`, which stores all the contact information to be called in a particular day, will be of O(T).

As O(T) and O(N) are asymptotically smaller than O(KN), the oveall space complexity is O(KN).

Footnote:

I believe this is already the optimal space complexity. Take the following situation again as an example: the input only contains failing performance, while `call limit` of each day is 0. The queue or any other abstract data type that we can use to implement the program will reach O(KN). If we want to try to reduce such space complexity, we may have to sacrifice the time complexity, e.g., through only making iteration using `Test_info`, which has the unsorted records of everyone's exam in every day