**Big O Calculations**

Note that, in the following explanations, the part of the code that reads the input files and store them in variables “calls” and “texts”, is disregarded.

**Task 0**

The time-complexity for the code is constant, or almost constant, regardless of the size of the “calls” and “texts” records, since the code only prints out the records corresponding to the given indices without any iteration over the records in “calls’ and “texts”.

Hence, the Big O notation for this problem is: O(1) or O(On + 1)

**Task 1**

The time-complexity for this code is a combination of linear order and constants.   
We define an empty list “phone\_list” only once and we also store the resulting output into a variable called “unique\_phone\_nos” only once. Note that, I disregarded the runtime contributed by the set() function here since it is being handled by python behind the scene.

Then we iterate over each call record in calls, a number of times equivalent to the size n1 of “calls” and run only one line that inserts new values to the “phone\_list”. Again, I disregarded the runtime contributed by the extend() method since it is being handled by python behind the scene. This process will be repeated for the “texts” record over n2 size of the “texts”.

We can write two different linear order notations for the two for loops and the constants as O(n1+n2+3). Instead, I combined them as one as: O(n + 3) where n = n1 +n2. This would not be the case if the coefficients of n1 and n2 were different.

In the example dataset provided, n1 = 5213 calls and n2 = 9072 texts. Inputting these values in the above notation yields 14,288.

Investigating one step further, we can see that adding 3 does not really impact the runtime for large dataset and can be dropped.

Hence the Big O notation for this problem is O(n).

**Task 2**

Investigating the constants in the code, we see that an empty diary “call\_diary” is only defined once, phone\_no\_list, call\_time\_list, max\_time, max\_index, and phone\_no all stores the values of the corresponding line operations once each (Note that just like before, the built-in functions are disregarded in my analysis since they are done behind the scene). At the points, we have 6 constants. Adding the print statement too, the constants becomes 7.   
  
Investigating the for loop, we can see that the loop iterates over the number of calls, n times, and performs a 3 line operations All these operations can be written as O(3n + 7) or O(3n) ignoring constant.

Hence the Big O notation for this problem is O(n).  
  
**Task 3**

**For Part A**We only define empty lists “fxd\_codes”, “mob\_codes”, and “tel\_codes” once. These amounts to 3 constants. We then iterate over “calls” a number n1 times, get the caller, get the receiver and perform two logical (if) operations to store values calling from Bangalore to either of the three predefined lists. The loop can be represented as: O(4n1). I assume that only one of either “if, elif, elif” would be satisfied for each iteration. Adding the above constant yields O(4n1 + 3).

We only print the given output statement “The numbers called by ….” just before the output codes once. This amount to 1 constant. Next is the line that determines the unique codes, sorts them and stores them in “output\_list”. Note that the set() function that is enclosed in the print function is disregarded in my analysis since it is done behind the scene and has O(1) notation. However, sorted() has a notation of O(nlogn). Adding this to the above notation yields O(4n1 + n2 logn2 + 4).

Then we print the number of codes in “output\_list” by iterating over the list a number, n3 times. (n3 varies depending on the number of unique codes stored in “output\_list”. This is represented as O(n3). Adding this to the above notation yields O(4n1 + n2 logn2 + 4).

Hence the Big O notation for this problem is O(n logn).

**Continuing to Part B**

The variable “percent” only stores the values of the corresponding line operations once and the print statement is done once. Both yields 2 constants. Then, the “count” variable stores the sum of the list that was first generated to filter “fxd\_codes” using a loop with iterations over “fxd\_codes”, say n times and a logical condition. This can be represented as O(2n + 1).

Hence the Big O notation for this problem is O(n).

**Task 4**

Empty lists receivers, texters, and tel\_nos are created once each. output\_list accepts the list of unique tele\_nos once but has a notation of (nlogn). The following print statement is also done once. All these amount to 5 constants. The following loop over output\_list that prints the numbers is done n1 times, where n1 varies depending on the size of the output\_list. We can write this as O(n1 + nlogn + 4).

The first for loop iterates over calls record. We represent this as O(n).

The second for loop iterates over texts. We represent this as O(n).

The third for loop iterates over tele\_nos. Within the loop is a line operation (an if condition) that also iterates over lists of callers and texters We also call that O(n). These can be treated as a nested loop with a quadratic notation O(n \* n) i.e. O(n^2).

Combining all these, we have: O(n1 + n2 + n3 + nlogn + n^2 + 4). The linear terms, n, and the constant can be omitted. Likewise, nlogn is dropped since O(n^2) > O(nlogn).

Hence the Big O notation for this problem is O(n^2).