PROGRAMMING WITH DATA – COURSEWORK II

TECHNICAL ANNEX (REPORT)

AUTHOR: CARLO BELLATI

ID NUMBER: 13152563

PHASE 1

MyHealthcare

MyHealthcare is a wearable device that generates n vital sign records of a person.

In the implementation I provided, MyHealthcare generates a n number of random values for each variable (vital sign) and stores them in a dictionary. This way we are able to query our dataset by selecting a variable and an interval within it.

Eventually we can transform the dictionary into a Pandas data frame (each variable becomes a series), which would make manipulation and retrieval a lot easier.

PHASE 2

abnSignAnalytics

abnSignAnalytics is a function that returns the total number of abnormal values of a chosen variable between pulse and blood pressure and a list all abnormal values (and their relative timestamp).

I have developed the function by using a dictionary structure (which marks the limits of the abnormal values). The function searches linearly within the given array one element by one, append and count the abnormal values.

As we can see the computational complexity of the function is linear: the running time increases linearly when increasing the size of the input (the size of the dataset). The running time may marginally depend also on the particular scenario, namely on the number of append and count operations to take place (depending on how many elements satisfy the condition). The space complexity of the function is also linear.

frequencyAnalytics

frequencyAnalytics is a function that returns a list of the frequencies for pulse rate values in a given dataset.

I have provided 2 different implementations.

The first implementations is as follows:

1) The function imports all pulse values from the given dataset (other values are not relevant here); 2) pulse values (not the number of their occurrences) are saved in another list; 3) the latter list is sorted by using a merge sort algorithm (this allows us to visualize our data in order);

3) the number of occurrences of each value in the list is counted by linearly going through all data; 4) a list of all values with their relative number of occurrences is returned.

In terms of time complexity, the function is of the order of Θ (n log(n)). In fact the two main components are the merge sort algorithm, which has a complexity of n*log(n) in worst scenario, and the counting/appending loop, which is a nested loop whose complexity depends also on the number of elements in the list 'alist' (n * alist). In the worst scenario each element has only one occurrence and therefore the complexity of this component is equal to n*n. In this scenario the total complexity would be O(n^2).

frequencyAnalytics2

The second implementation takes advantage of some Pandas functionalities. The dataset is converted into a Pandas data frame, each 'pulse' value is counted and sorted and a final list of all results is returned at the end (for detailed explanation see the code comments). As the operations performed are hidden by Pandas implementation, it is difficult to make some inference.

<u>PHASE 3</u>

HealthAnalyzer is a function that provides a query mechanism to search for a
particular pulse rate value within a given dataset. The function returns all records containing the given pulse rate value.

I have provided 3 different implementations.

HealthAnalyzer

The first implementation adopts a simple model of linear search. The function search linearly across all the dataset and every time it finds an occurrence it appends the whole record into a list, which is returned at the end. Therefore, the time complexity is equal to Θ(n).

HealthAnalyzer2

The second implementation is much more complex. Instead of using a linear search, I decided to develop a binary search algorithm, customized in such a way that it does not stop when it finds an occurrence but rather keeps searching until all occurrences are found.

In order to use binary search, before searching, the function must firstly sort all values, and I have adopted one of the fastest sorting algorithms, which is quick sort. I have implemented a version of quicksort that takes two lists rather than one as arguments: the list of 'pulse' values is sorted in ascending order while the list of timestamps is sorted accordingly. The timestamps are used later as indexes to retrieve all the associated values for that record, which are appended to the list of results.

As we know the quick search algorithm has an average of Θ(n log(n)) time complexity, with Θ (n^2) in worst case. On the other hand, binary search has an average time complexity of O(log(n)). Therefore we may conclude that HealthAnalyzer2 has on average a complexity equal to (n log(n)) + (log(n)) = Θ(n log(n)).

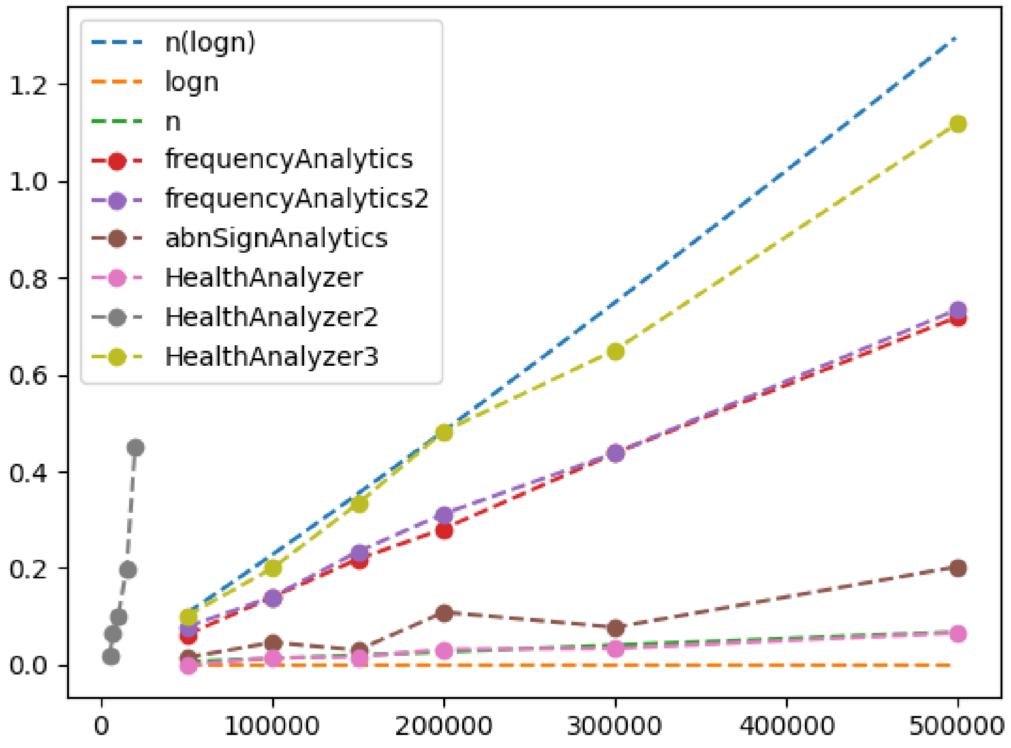
However, the benchmarking analysis reveals that this is not the case for this implementation. In fact, a complexity of nearly O(n^2) is what emerges when running the algorithm, which turns out to be quite slow and inefficient: the computer struggles to process an array of more than 20,000 records.

HealthAnalyzer3

The third implementation uses Pandas functionalities. It uses the .loc method to locate all values matching the given key (pulse value) and then it proceeds to append all found records through the .iterrows method.

PHASE 4

In the graph below we can observe the computational cost of all developed functions over an array of 6 different input size (except for HealthAnalyzer2). The graph displays 3 lines useful for comparison, n, n(logn) and logn.



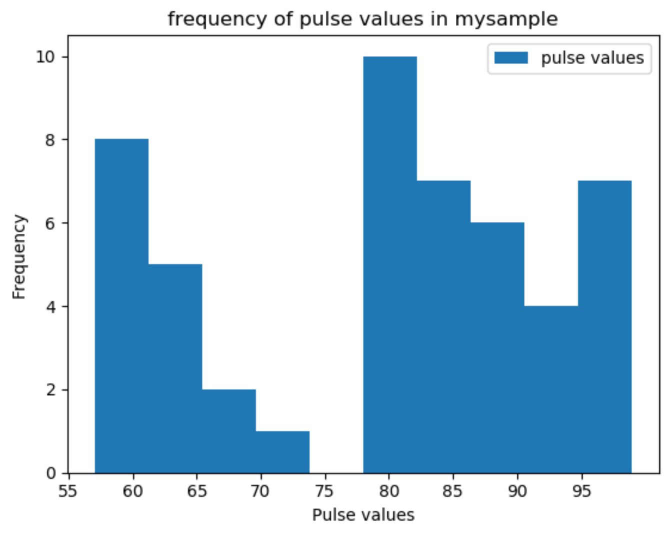
Most of the functions stands in the area between the logn and nlogn lines; HealthAnalyzer and abnSignAnalytics are the fastest (the former overlapping the n line), while frequencyAnalytics and frequencyAnalytics turn out to be very similar, almost overlapping. HealthAnalyzer3 is close to n(logn) line, as we may expect. HealthAnalyzer, despite the efforts, stands out as not suitable for processing a wide array of data.

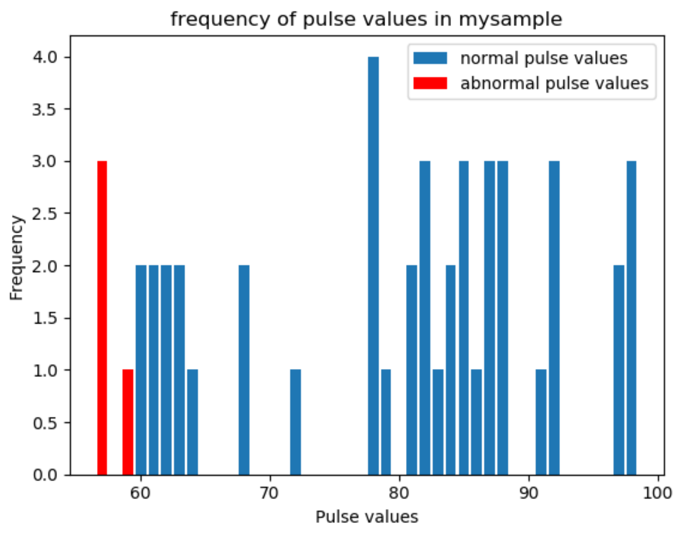
CHART AND GRAPHS

PHASE 2

I obtained a sample of 50 records through MyHealthcare (mysample) and counted the instances of pulse rate values.

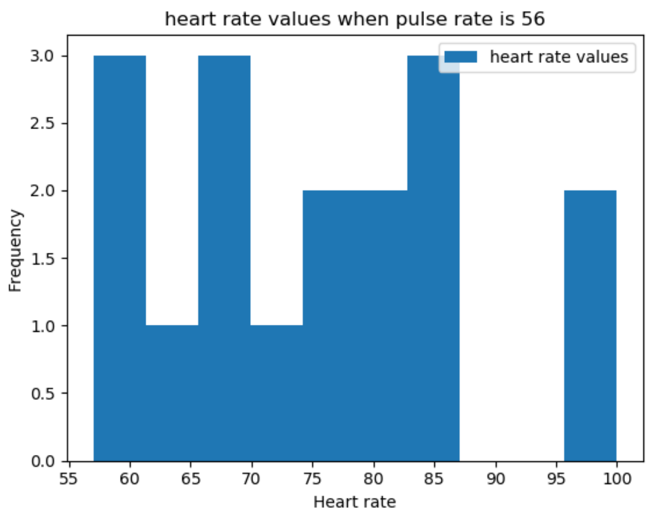
The following are a histogram and a bar chart. The bar chart considers each single different pulse rate value rather than an interval and distinguishes between normal and abnormal values. In the sample obtained values equal to 100 (the only high abnormal value) were not found.





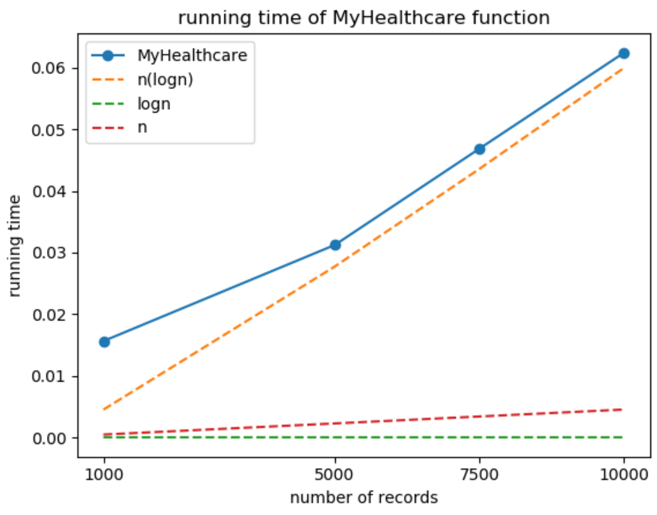
PHASE 3

I have plotted the heart rate values for records having a pulse rate value of 56.



PHASE 4

I have benchmarked the MyHealthcare application with 5 different size of input. The graph shows the results.



As we can see, the application's time complexity has values which are very close to the n(log(n)) line (slightly above it). We may conclude that it has a time complexity of Θ (n log(n)).