**Technical Annexe PWD CW2:**

**Phase 1 Function output:**

A screenshot of a social media post

Description automatically generated

**A screenshot of a cell phone

Description automatically generated2a Function output:**

**2c)** ‘abnormalSignAnalytics’ algorithm utilises a random function to generate random numbers within range of the number of samples generated in phase 1. This chooses n number of time stamps to use as records. This is a non-deterministic approach which creates an assumption-free collection.

Also, a linear search is used which has a time complexity worst case scenario of O(n) which is fairly efficient. There are a few 'for loops' used but nothing more complicated (e.g. nested for loops) that would increase the time complexity. This also applies to ‘frequencyAnalytics’ and ‘frequencyAnalytics2’. The results are stored in a dictionary format within ‘mydict2’. More details of my code can be seen in the comments within the ‘.ipynb’ file submitted.

Space complexity is very minimal as most of the data printed is stored in a dictionary. Many small lists are also used within the functions.

Although binary search would have produced a lower time complexity of O (log n), Logarithmic time, it would require a sort algorithm to achieve this as the keys in the dictionary from phase 1 are not ordered. If I were to implement the lowest time complexity sort algorithm explored in lectures (Merge sort or Timsort) they, at best, would provide a time complexity of O (n log n) and O(n) respectively. However, they would also produce a worst-case scenario of O (n log n) which is worse than my current time complexity. This also applies to ‘frequencyAnalytics’ and ‘frequencyAnalytics2’.

From the scatter plot generated below of function (2a), it is evident that there is no correlation between time stamps and pulse rate chosen randomly. Very few values (9 red values) are abnormal out of the 50 selected. Again, emphasizing a non-deterministic approach for selection of records.

A screenshot of a cell phone

Description automatically generated



As seen in the histogram for function ‘frequencyAnalytics’ and frequencyAnalytics2’ (2b), there appears to be a random but slight normal distribution where majority of the pulse rates is between 60 and 99. The most common pulse rates in that sample seem to be between 75 and 80. There was only 1 record with pulse rate of 100. A total of 9 abnormal records and 41 normal records.

A random function is used to generate random numbers, within range of the number of samples generated in phase 1, to choose n number of time stamps to use as records, much like the previous function in 2a.

Space complexity is minimal as most of the data printed is stored in a dictionary. Many small lists are also used within the functions but there is the plotting and preparation (‘for’ loop) of the data for the histogram plot.

**A screenshot of a cell phone

Description automatically generated2b function output:**

A close up of a device

Description automatically generated

However, when using ‘frequencyAnalytics2’, where the code to plot the histogram was taken out and inputted into a new function ‘frequencyAnalyticsHisto’, the time to run the algorithm greatly improves from 0.321 seconds to 0.00087 seconds. I believe this is due to the time saved not printing and not having to use the additional ‘for loop’ to separate pulse rates into normal and abnormal lists before plotting.

A screenshot of a cell phone

Description automatically generated

**A screenshot of a cell phone

Description automatically generated3a function output:**

**3b)** The ‘sorted()’ function used in python, I believe is defaulted to the Timsort function so time complexity is worst-case O (n log n) and best-case is O (n), space complexity is O (n).

I also use a linear search algorithm which has a worst-case O (n) time complexity.

Overall, worst-case scenario is O (n log n) due to the Timsort function used.

However, if the outputted multidimensional list is not required to be in a sorted order then ‘healthAnalyser’ would only uses a linear search would have a lower complexity of O(n).

Below is the plotted heart rate values for records having pulse rate 56 which shows little correlation with timestamps.

A screenshot of a cell phone

Description automatically generated

Another solution, ‘healthAnalyzer2’, utilises a binary search in addition to the Timsort used in ‘healthAnalyzer1’. This has a worst-case scenario of O (log n) which is better than Timsort. It also has a better O (1) space complexity. However, this algorithm’s complexity is decided on the sorting of the list which utilises a ‘for loop’ so the complexity is O(n) overall.

**A screenshot of a social media post

Description automatically generated4a function output:**

4b) Above are the benchmarks taken for ‘myHealthcare’ and the times are shown in seconds. There shows a positive correlation between time and ‘n’ number of operations. This function has a O (n) time complexity as there are only ‘for loops’ used to generate the samples. Data generated is stored in a dictionary.

Below, is the plot of the above results. It generally shows a steep linear increase as operations increase. There is however a small decline between 5000 and 7500 operations.

A screenshot of a map

Description automatically generated