Assignment 1

1. Given the following algorithms, answer the questions.

   - Linear Search
   - Bubble Sort
   - Selection Sort

   1.1. (T) Analyze the time complexity of above algorithms using the RAM model (Include the handwritten analysis of these algorithms as an image in the text or latex file. Make sure the images/contents are readable.).

   1.2. (L) Implement the above algorithms using the programming language of your choice.

   1.3. (L) Provide the details of Hardware/Software you used to implement algorithms and to measure the time.

   1.4. (L) Submit the code (complete programs).

   1.5. (L) Measure the best-case time and worst-case time of linear search for all ten files. Plot a graph.

   1.6. (L) Measure the average-case time (considering current data of ten files) of bubble sort, and selection sort for all ten files. Plot a graph.

   1.7. (L) Measure the best-case time of bubble sort, and selection sort for all ten files. Plot a graph.

   1.8. (L) Measure the worst-case time of bubble sort, and selection sort for all ten files. Plot a graph.

   1.9. (T) Assume that you don't know the time complexity of above algorithms.

      1.9.1. Can you predict the same based on your implementation of above algorithms?

      1.9.2. Do they match with theoretical time complexity? Yes/No.

      1.9.3. If yes, then write the time complexity of each algorithm. If no, then write the difference.

Note:

- You can use any programming language such as C, C++, Java, or Python. However, in any chosen programming language, you are not allowed to use any library functions or any language specific features that hide the logic of the algorithm. Even if you use any language, the pseudocode and the program code should look nearly the same, irrespective of the chosen language. If the specific language does not support it, then that language is not allowed.
- Best case analysis for sorting - You have to first sort the data (files), and then supply the sorted values as an input to the algorithm for which you are taking the results. For worst case analysis, you have to reverse sort the data (files), and then supply the reverse sorted values as an input to the algorithm for which you are taking the results. For the best case input to the linear search algorithm, search the first value of a specific file. For the worst case input of the linear search algorithm, search the value that is not in the file (may be negative numbers are not in the file).
- For the above assignment, due to limitations of hardware, if you do not get the answer within an hour (for any question), you can stop the algorithm. However, you have to predict the time algorithm requires to complete the sorting/searching (based on your analysis and previous results). In the plot, you have to isolate those points (that you have predicted from those that you have obtained as a result.)
- The size of different files are usually 2^x numbers where x may be 10, 12, etc.
- Instead of running the algorithm only once and recording the running time, it is better to run the algorithm k times (e.g., 10 or 1000) using the loop and at the end divide it by k to get the actual running time. The same will be more accurate.
- While measuring results, if the average-case running time of the bubble sort is more than the worst-case running time of the bubble-sort, then it is because of the in-build optimization performed by JVM. In order to correctly measure the readings, you can use the `-Xint` command-line switch (as mentioned below) and then measure the results. Those interested in the same can also read the following two blogs. Those who are using other programming languages (e.g., C, Python, etc.) should also go through similar language specific optimization and disable it while measuring the results, to ensure the correctness of the result (to ensure that the results will match with theory).

  ```
  java -Xint bubble_sort
  ```

  Link 1 Link 2
- If you submit a text file or zip file containing other files, keep your student ID (roll no) as a file name (for all files, e.g., U19CS001 or U19CS001-1 and U19CS001-2, etc.).