Christina Darstbanian ID : 40097340 and James Partsafas ID: 40170301
Programming Part :
A)
Observations on the timing measurements
All 4 algorithms were tested separately with different sample sizes. The sample sizes increased gradually and in this experiment 7 sample sizes were taken into account: N=1, N=10, N=100, N=1000, N=10 000 , N=100 000 , N= 1000 000 as required, where N represent the sample size (meaning the size of the list of participants including their name and their Date of birth).
Here is a table of required sample sizes of the experiment with their corresponding runtime time in milliseconds:

Sample size	Rearrange Participants method (Algorithm1) running time	Display Seniors Increasing Order method (Algorithm 2) running time	Display NonSeniors Increasing Order method (Algorithm 3) running time	Display Increasing Order method (Algorithm 4) running time	Tail recursive method part B running time
N= 1	1	0	0	1	0
N=10	1	3	3	6	1
N=100	7	20	24	24	19
N= 1000	26	161	167	164	328
N= 10 000	152	1150	1135	1159	Stack overflow
N= 100 000	953	10476	10979	10729	Stack overflow
N=1000000	10254	14654	12200	13228	Stack overflow

Based on obtained results one can clearly see that although most values of time are close to each other in between algorithms corresponding to the same sample size, but definitely there is an increase of time with the increase of sample size in all 4 algorithms. Therefore one can determine the fact of run time dependency on sample size.

Time complexity function f(n) and the time complexity of the function in terms of Big-O.

1. The f(n) obtained from algorithm 1 is f(n)= 1195nlogn+326logn, taking into account the order presented in the class notes the nlogn is bigger than logn therefore the resulting Big(O) time complexity becomes n logn.
2. The f(n) obtained from algorithm 2 is f(n)= 604nlogn+326logn, taking into account the order presented in the class notes the nlogn is bigger than logn therefore the resulting Big(O) time complexity becomes n logn.
3. The f(n) obtained from algorithm 3 is f(n)= 600nlogn+39logn, taking into account the order presented in the class notes the nlogn is bigger than logn therefore the resulting Big(O) time complexity becomes n logn.
4. The f(n) obtained from algorithm 4 is f(n)= 600nlogn+38logn, taking into account the order presented in the class notes the nlogn is bigger than logn therefore the resulting Big(O) time complexity becomes n logn.

B) the algorithm is not linear because the recursive method is called more than one time , Linear recursion calls are done at most one time when the method is invoked.
The original algorithm made in part A does not use tail recursion because algorithm with tail recursion in our case was less efficient, has less efficient space complexity and had much longer running time, the merge sort method by binary recursion that was made in order to have faster running time and more efficient implementation.In addition to the fact that in the original method merge sort was used and two halves of the problem was taken into consideration (right and left part of the array of list of participants) two recursive calls was supposed to be done and therefore binary recursion. The method original method implementation was also done when the Tail recursion method was not presented in class therefore that method was used.
Yes, the method can be written in tail recursive method by using insertion sort which is less efficient having a bigO(n^(2)) and can't run showing stack overflow when the sample size N reaches 10 000.

Note:Refer to Algorithms and pseudocodes that are all attached to this file with out.txt files.
There are also 4 pdf files that show the steps of calculating f(n) function highlighted in green for clarity reasons.