Name: B M RAUF
ID: 22201782

CSE221
LAB01

SEC20
Spring'24

Task 1

a

I run a loop for n times and read the input line by line. ~~Then~~, I take the input number and mod it by 2. We know, if we mod even number by 2, we will get 0. In my code, I do the same thing. If ~~it~~ num%2 ==0, I identify the number as even number. and if the num%2 !=0, then I identify the number as odd number. Finally, I compiled the output in the output 1a. txt file.

b

Firstly, I split the string based on space and then I choose the last 3 elements because these 3 elements are the 2 operands and 1 operators. Then, I add/subtract/multiply/divide the 2 operands based on the operator. Finally, I compiled the output in the output 1b. txt file.

Task02

Firstly, I run a loop for len(arr)-1 times and inside the loop I take a flag and run another loop for len(arr)-i-1 times. Inside the 2nd loop I compare numbers and swap if needed. If the elements are not in sorted order, elements will swap their position and the flag will become true. Then again when the first loop runs flag will become False again. If no swaps are made in any pass it means the array is sorted and as the flag is False it will break the loop! As, the loops will not run anymore, it will achieve the $\theta(n)$ for the best-case scenario.

Task3

I use selection sort technique for this code. because it minimizes the number of swaps by only performing swaps when indentify the maximum value in each iteration. It divides the array into a sorted portion and an unsorted portion, consistently positioning the maximum element in its rightful place with a solitary swap. If my arr length is N, it sorts the array by performing a maximum of n-1 swaps.

Task04

Firstly, I split the array and store train name, location & departure time in a list and pass it as arguments. Then, I use bubble sort to compare the names of the trains and sorts the trains in the lexicographical order based on the name of the trains. If the name of two or more trains are same then I compare its departure time. If departure hours are same then comparing minutes are enough for sorting. If departure hours are not same then comparing departure hours are enough for sorting. But, if the departure times are also same then automatically the train with the latest departure time will get prioritized. Finally, I compiled the output in the output 4.txt file.