Problem 1 For each of the following situations, give the time-complexity of the described algorithm in big-O notation.

a) An algorithm takes a list of length n as input and needs around 5n + 8 operations to return.

Sol: O(n)

b) An algorithm takes an integer k as input and then proceeds to do at most k^2 + k operations.

Sol: O(k^2) [where K^2 means k square]

c) A program takes an array-list of length n as argument and prints it in reverse.

Sol : O(n)

d) An application has a list of all its n registered user in alphabetical order. It uses Binary Search to check if a username already exists.

Sol: O(log n)

e)To generate a three dimensional grid, a program takes a list of length n as input and returns a new list of all combinations of three elements from the list.

Sol : O(n^3) [where n^3=cube of n]

f) To return the smallest element of a sequence, a program uses Insertion Sort and then outputs the first element of the sorted sequence.

Sol: O(n^2) [where n^2 means n square]

Problem 2 The following function takes as input an array-list of numbers. Read the pseudo-code and answer the questions below.

function secret_function(numbers):

a) Write the output after calling secret_function([-5, 10, 9, 11, 3, 5, 0, 21])

Sol: [0, 3, 5, 9, 10]

b) In one or two sentences, describe what the above function does.

Sol: it return the list having numbers in increasing order between zero and ten including the numbers equal to zero and ten from the input array-list of numbers.

c) What is the time-complexity of secret_function ?

Sol: O(n^2) [where n^2 means n square]

d) Suggest a way to improve the time-complexity of the function and give the new time-complexity.

Sol: Instead of sorting the array list with InsertionSort, we can sort with MergeSort which will change the time complexity to O(nlog(n)).

Problem 3 In the previous assignment, you were asked to implement parentheses matching for an IDE. This algorithm allowed to detect a parentheses mismatch, but only for the characters ( ) . You are now asked to extend that algorithm to implement bracket matching.

Bracket matching is similar to parentheses matching, but it allows more than one type of parentheses. You are asked to implement bracket matching for the characters ( ) , [ ] and { } .

Using pseudo-code , implement a function called match which takes a string of brackets as argument and returns True if the brackets match and False if the brackets mismatch.

Sol :

Function match takes string of brackets as argument and if all the open parentheses are closed then it returns True else it returns False. Information : " (" opposite ") " , " { " opposite " } " , " [" opposite "] "

```
function match(list):
    finalList=emptyList
    open=array-list having values[ " ( " , " { " , " [ " ]
    close=array-list having values[ " ) " , " } " , " ] " ]
    for each element in a list:
        If element in open:
            finalList.append(element)
        else if element in close:
            If length of finalList == 0 :
                return False
            else:
                If element matched to the opposite:
                    finalList.pop()
                else:
                    return False
If length of finalList == 0:
        return True
else:
        return False
```