EECE 230X – Introduction to Computation and Programming
Programming Assignment 3

- This programming assignment consists of 3 problems.
- Prerequisites: Topic 4
- Related material: Lists, tuples, and strings

Problem 1. Check if a sequence is sorted: lists and boolean variables

a) **Using a list.** Write a program which asks the user to enter a sequence of integers and checks whether or not the input sequence is sorted in nondecreasing order. Thus your program is supposed to give a YES/NO answer. See the below examples.

It is easier to solve this problem by storing all the input sequence in memory. You are asked to solve this part by first storing the sequence in a list and then processing the list. Assume in this part that the integers in the input sequence are separated by spaces. First read the input sequence into a string `st` and turn it into a list of integers L using the `st.split` method. Use a boolean variable `isSorted`.

Sample Input/Output:

```
------------------------------------------
Enter integers spearated by spaces: 5 12 14 20 25
YES: input is sorted
------------------------------------------
Enter integers spearated by spaces: 5 12 12 20 25
YES: input is sorted
------------------------------------------
Enter integers spearated by spaces: 5
YES: input is sorted
------------------------------------------
Enter integers spearated by spaces: 5 20 12 25
NO: input is not sorted
------------------------------------------
Enter integers spearated by spaces: 10 2 4
NO: input is not sorted
```

b) **Without lists.** Repeat Part (a) without storing the input sequence in memory. In this part, ask the user to first enter the number of integers and then the integers each on a new line as shown in the below examples.

Don't try to read the following hint unless you are stuck (to read it you need to magnify the PDF file and properly read it in reverse order); try first to solve it on your own without help.

(*Hint:* .elbairav naeloob a osla esU .`verp` etadpu neht ,`verp` htiw `x` erapmoc ,`x` elbairav eht otni tupni resu eht daer ,0 > i fI .tupni resu eht ot `verp` tes ,0 == i fI .1 − n, . . . , 0 = i revo pooL .`verp` tupni suoiverp eht fo kcart peek ,`x` tupni resu tnerruc eht ot noitidda nI .sregetni fo n rebmun eht daer tsriF)

Sample Input/Output:

```
Enter the number of integers:5
Enter an integer:5

Enter an integer:12

Enter an integer:14

Enter an integer:20

Enter an integer:25
YES: input is sorted

------------------------------------------
Enter the number of integers:2

Enter an integer:10

Enter an integer:2
NO: input is not sorted
```

Problem 2. Compare lists

a) **Check if identical using a loop.** Write a program which asks the user to enter a two sequences of integers and checks whether or not the two sequences are identical.

First, read the first input sequence into a string and then turn it into a integer sequence $L1$. Similarly, store the second input sequence in a list $L2$. Thus you need to check if the $L1$ and $L2$ are identical.

In this part, you are not allowed to use the equality check operator `==` on lists (see Part (b) below).

Sample Input/Output:

```
Enter the  integers in the first sequence spearated by spaces: 1 2 10 3

Enter the  integers in the second sequence spearated by spaces: 1 2 10 3
Sequences are equal

-------------
Enter the  integers in the first sequence spearated by spaces: 1 2 10 3

Enter the  integers in the second sequence spearated by spaces: 2 1 10 3
Sequences are not equal
-------------
Enter the  integers in the first sequence spearated by spaces: 1 2 10 3

Enter the  integers in the second sequence spearated by spaces: 2 20
Sequences are not equal
```

b) **Check if identical using the equality check operator ==.** Now that you know how to do using a loop, repeat Part (a) using the equality check operator ==. Note that this part is straight forward.

c) **Check if permuted.** Write a program which asks the user to enter a two sequences of integers and checks whether or not the second sequence is a permutation of the first.

As in Part (a), read the two sequences into tow lists L_1 and L_2, Thus we would like to check if $L2$ is a reordering of $L1$.

In this part, you are not allowed to use methods associated with the list type that we didn't cover. Work with the list indexing operator.

Don't try to read the following hint unless you are stuck (to read it you need to magnify the PDF file and properly read it in reverse order); try first to solve it on your own without help.

(*Hint:* .lauqe era $2L$ dna $1L$ fo shtgnel eht fi kcehc tsrif. n ot lauqe era yeht taht emussA .ON si rewsna eht neht ,lauqe ton fI. $'X' =]i[2L$ tes ,.g.e ,)ecno ylno(ti ssorc ,dnuof fI .$2L$ ni $]i[L$ rof kool ,$n, \ldots, 1 = i$ roF)

Sample Input/Output:

```
Enter the  integers in the first sequence spearated by spaces:  1 2 10 3 10 10 10

Enter the  integers in the second sequence spearated by spaces: 10 10 2 10 1 3 10
YES: second sequence is a permutation of the first
-------------
Enter the  integers in the first sequence spearated by spaces: 1 2

Enter the  integers in the second sequence spearated by spaces: 1 1
NO: second sequence is not a permutation of the first
-------------
Enter the  integers in the first sequence spearated by spaces: 1 2 3

Enter the  integers in the second sequence spearated by spaces: 2 3
NO: Second sequence is not a permutation of the first
```

Problem 3. Max-sum

In **the max-sum problem**, given a sequence of integers, we are interested in finding a subsequence of contiguous elements of the input sequence whose sum is maximum.

For example, consider the sequence

-1 2 -1 3 -5

The following table shows all its contiguous subsequences underlined and the corresponding sums.

subsequence	sum	subsequence	sum
$\underline{-1}$ 2 -1 3 -5	-1	-1 $\underline{2\ -1\ 3\ -5}$	-1
$\underline{-1\ 2}$ -1 3 -5	1	-1 2 $\underline{-1}$ 3 -5	-1
$\underline{-1\ 2\ -1}$ 3 -5	0	-1 2 $\underline{-1\ 3}$ -5	2
$\underline{-1\ 2\ -1\ 3}$ -5	3	-1 2 $\underline{-1\ 3\ -5}$	-3
$\underline{-1\ 2\ -1\ 3\ -5}$	-2	-1 2 -1 $\underline{3}$ -5	3
-1 $\underline{2}$ -1 3 -5	2	-1 2 -1 $\underline{3\ -5}$	-2
-1 $\underline{2\ -1}$ 3 -5	1	-1 2 -1 3 $\underline{-5}$	-5
-1 $\underline{\mathbf{2\ -1\ 3}}$ -5	**4**		

Thus the maximum sum (of a subsequence) is 4 (second column, last row) and the corresponding subsequence is 2 -1 3

Note that what makes this problem nontrivial is that the integers in the sequence are allowed to be negative (if all the integers are nonnegative, then the maximum sum subsequences is the the whole sequence).

Below are other examples of integer sequences and their corresponding maximum sum subsequences (underlined).

maximum sum subsequence	maximum sum
-1 $\underline{\mathbf{2\ -1\ 3}}$ -5	**4**
-10 -1 $\underline{\mathbf{3\ 4\ -3\ 5\ -2\ 7}}$ -11 2 4	**14**
$\underline{\mathbf{1\ 4\ 0\ 2}}$	**7**
-10 $\underline{\mathbf{2\ 2}}$ -10 1 3	**4**
$\underline{\mathbf{1\ 2\ -3\ 4\ 5}}$ -9 2 6 -10 5	**9**
-1 $\underline{\mathbf{0}}$ -5 -2	**0**
-1 -2 -5 -2	**0**

Remarks:

* There might be more than one contiguous subsequence of the input sequence whose sum is maximum (e.g., in the fourth sequence in the above table, 1 3 is also a max sum sequence). You don't have to find all of them. We are satisfied with any one of them as an answer.

* If all the integers in the sequence are negative, then the maximum sum subsequence is the empty subsequence and the max sum is by convention zero (e.g., the last sequence in the above table).

a) **Three nested loops.** Write a program to solve this problem. First read the input sequence into a string and turn it into a list of integers L.

In this part we don't care about efficiency; do it in the most direct way. Let n be the length of L. Use a variable `maxSum` intialized to zero. Loop on $i = 0, \dots, n-1$ and $j = i, \dots, n-1$. For each (i, j), use a third loop to find the sum of the subsequence $L[i], \dots, L[j]$ and update `maxSum` if needed. You also need two variables a and b to keep track of the start and end of the max-sum subsequence found so far.

Sample Input/Output:

```
Enter integers spearated by spaces: -10 -1 3 4 -3 5 -2 7 -11 2 4
Max-sum =  14
A max-sum subsequence:
3 4 -3 5 -2 7
--------------------------------------------------------------
Enter integers spearated by spaces: -1 -2 -5 -2
Max-sum =  0
```

b) **Using slicing.** *Slicing operator* **(:)**: If `L` is a list and `a` and `b` are integers such that $0 < a < b \leq$`len(L)`, `L[a:b]` returns the sliced sub-list $[L[a], L[a+1], \dots, L[b-1]]$. The slicing operator works also with tuples and strings. In addition to a and b, the slicing operator takes a third parameter `step` which is by default set to 1. If `step`≥ 1, `L[a:b:step]` gives $[L[a], L[a+2], \dots, L[c]]$, where c is the largest integer less than b such that $c = a + i\times$`step`, for some integer i. We can also skip `a` and `b`: the default value of `a` is 0 and the default value of `b` is `len(L)`-1. Moreover, `step` can take negative values, e.g., $L[:: -1]$ returns the reverse list of L.

Repeat Part (a) while substituting the third loop with the slicing operator and the built-in `sum` function (if A is list or tuple, then `sum`(A) returns the summation of the elements of A).

Note that while algorithmically this solution is equivalent to Part (a), using the slicing operator can be faster due to Python's interpreter overhead.

Note: For a list L, `L[:]` is equivalent to `L.copy()`, i.e., they both return a clone of L.

c) **Two nested loops.** Do it more efficiently, in particular, using two nested loops and without slicing (in this part, you are only allowed to use the list indexing operator).

Don't try to read the following hint unless you are stuck (to read it you need to magnify the PDF file and properly read it in reverse order); try first to solve it on your own without help.

(*Hint:* .(j, i) hcae rof hctarcs morf $]j[L, \dots,]i[L$ fo mus eht etupmoc ot deen on si erehT. $1 - n$ ot i morf setareti j sa noitammus eht etalumucca ,i hcae roF)

d) ($\star$) **Without nested loop.** Now, do it without using nested loops and without slicing. This part is difficult. Emulating two nested loops using a single loop doesn't count as a valid solution since it is as slow as two nested loops. The aim is to do it **efficiently**: the number of steps should scale linearly NOT quadratically with the list length.