Data Structures and Algorithms

COSC 336 Assignment 1

Instructions.

1. Due date and time: As indicated on Blackboard.

2. This is a team assignment. Work in teams of 3-4 students. Submit on Blackboard one assignment per team, with the names of all students making the team.

3. Your programs must be written in Java.

4. Write your programs neatly - imagine yourself grading your program and see if it is easy to read and understand.

   Comment your programs reasonably: there is no need to comment lines like "i++" but do include brief comments describing the main purpose of a specific block of lines.

5. You will submit on **Blackboard** three files.

   The **1-st file** is a pdf file (produced ideally with latex and Overleaf) and it will contain the following:

   (a) The solution to the Exercises.

   (b) A short description of your algorithms for the Programming Tasks 1 and 2, where you explain the dynamic programing approach (see the sketch of the **Algorithm** below). More precisely, you need to indicate how you compute $d[0]$ (this is the initialization step), and how you compute for every $i \geq 1$, the value of $d[i]$ using the values of some of the previous $d[j]$'s, for $j < i$).

   (c) For each of the tasks, a table with the results your program gives for the three data sets given below for each task.

   (d) The java code (so that the grader can make observations).

   The **2-nd file** is the .java file containing the java source code for Programming Task 1 and the **3-rd file** is the .java file containing the java source code for Programming Task 2, so that the grader can run your programs.

Exercise 1

Consider the following three program fragments (a), (b), and (c).

(a)
```
sum = 0;
for (int i = 0; i<100 * n ; i++) {
        sum++;
}
```

(b)
```
sum = 0;
for (int i = 0; i< 2*n ; i++) {
        sum++;
}
```

(c)
```
sum = 0;  i=n*n;
while (i > 1) {
        sum++;
        i= i/2;
}
```

We denote by $T_a(n), T_b(n), T_c(n)$ the running time of the three fragments.

1. Give Θ evaluations for $T_a(n), T_b(n), T_c(n)$.
2. Is $T_b(n) = O(T_a(n))$? Answer YES or NO and justify your answer.
3. Is $T_c(n) = \Theta(T_a(n))$? Answer YES or NO and justify your answer.

Exercise 2.

Give an example of a function $f(n)$ with the property that $f(n)$ is $\omega(n^2)$ and also $f(n)$ is $o(n^3)$.

Exercise 3. Indicate the running time of the following program fragment in the $\Theta(\cdot)$ notation.

```
x = 0;

for {i=1 ; i <= 2n+3; i++)

    for (j= 1; j < = 3n+7; j++)

        x = x+1
```

Programming Task 1.

You will write a program that computes the length of a longest **increasing** subsequence of a sequence of integers.

Formally, an increasing subsequence of the sequence $a_1, a_2, \ldots, a_n$ of length k is given by k indices $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ such that $a_{i_1} < a_{i_2} < \ldots < a_{i_k}$. So the goal is to find the largest k for which there exists an increasing subsequence of the input sequence of length k. Note: There is one major difference from the problem with *max contiguos subsequence sum* which we discussed in class, namely in this problem the subsequence is **not contiguous**, meaning that the numbers in the subsequence do not have to be in consecutive positions.

For example, if the input sequence is $10, 9, 2, 5, 3, 101, 7, 18$ then a longest increasing subsequence is $2, 5, 7, 18$, which has length 4 (there is another increasing subsequence, namely $2, 3, 7, 18$, also of length 4). Therefore your program should return 4 because there is no increasing subsequence of length 5 or larger.

Your program will read the initial sequence which is entered by the user, and will print the length of a longest subsequence. As a bonus, you may want your program to also print one longest increasing subsequence.

Algorithm You will implement an algorithm using the dynamic programming paradigm, which is similar to Algorithm 3 for *max contiguous subsequence sum* that we discussed in our meeting (see Notes1-Intro on Blackboard).

Suppose the initial sequence is $a_0, a_1, \ldots, a_{n-1}$. Then, you can calculate in order, one by one, the elements of an array $d[0], \ldots, d[n-1]$, in which $d[i]$ is the length of the longest increasing subsequence whose last term is a_i. Think how to calculate $d[0]$ and then how to calculate $d[i]$ as a function of the previous entries $d[1], \ldots, d[i-1]$ and the sequence $a[]$.

Example:

Input: $10, 9, 2, 5, 3, 101, 7, 18$. Output: 4, or for the bonus solution $4, (2, 5, 7, 18)$.

Test your program on the following sequences and insert in the first file (the pdf file) that you submit tables with the results for each sequence:

- $10, 9, 2, 5, 3, 101, 7, 18$
- 186, 359, 274, 927, 890, 520, 571, 310, 916, 798, 732, 23, 196, 579,
  426,188, 524, 991, 91, 150, 117, 565, 993, 615, 48, 811, 594, 303, 191,
  505, 724, 818, 536, 416, 179, 485 , 334 , 74, 998, 100, 197, 768, 421,
  114, 739, 636, 356, 908 , 477, 656
- 318 , 536 , 390 , 598 , 602 , 408 , 254 , 868 , 379 , 565 , 206 , 619 , 936 , 195 ,
  123 , 314 , 729 , 608 , 148 , 540, 256 , 768 , 404 , 190 , 559 , 1000 , 482 , 141 , 26,
  230 , 550 , 881 , 759 , 122 , 878, 350, 756, 82, 562, 897, 508, 853, 317 ,
  380 , 807 , 23 , 506 , 98 , 757 , 247

Programming Task 2.

You will write a program that computes the length of a longest **decreasing** subsequence of a sequence of integers.

Formally, an increasing subsequence of the sequence $a_1, a_2, \ldots, a_n$ of length k is given by k indices $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ such that $a_{i_1} > a_{i_2} > \ldots > a_{i_k}$. So the goal is to find the largest k for which there exists an decreasing subsequence of the input sequence of length k.

For example, if the input sequence is $4, 9, 2, 5, 3, 101, 7, 18, 2, 1$ then a longest decreasing subsequence is $9, 5, 3, 2, 1$, which has length 5. Therefore your program should return 5 because there is no decreasing subsequence of length 6 or larger.

Your program will read the initial sequence which is entered by the user, and will print the length of a longest decreasing subsequence. As a bonus, you may want your program to also print one longest decreasing subsequence.

Algorithm The algorithm is very similar to the for Programming Task 1. You need to change only one thing in the way $d[i]$ is calculated from the previous values of the $d[]$ array.

Example:

Input: $4, 9, 2, 5, 3, 101, 7, 18, 2, 1$. Output: 5, and for the bonus solution $5, (9, 5, 3, 2, 1)$.

Test your program on the following sequences and insert in the first file (the pdf file) that you submit tables with the results for each sequence:

- $4, 9, 2, 5, 3, 101, 7, 18, 2, 1$
- 186, 359, 274, 927, 890, 520, 571, 310, 916, 798, 732, 23, 196, 579,
  426,188, 524, 991, 91, 150, 117, 565, 993, 615, 48, 811, 594, 303, 191,
  505, 724, 818, 536, 416, 179, 485 , 334 , 74, 998, 100, 197, 768, 421,
  114, 739, 636, 356, 908 , 477, 656
- 318 , 536 , 390 , 598 , 602 , 408 , 254 , 868 , 379 , 565 , 206 , 619 , 936 , 195 ,
  123 , 314 , 729 , 608 , 148 , 540, 256 , 768 , 404 , 190 , 559 , 1000 , 482 , 141 , 26,
  230 , 550 , 881 , 759 , 122 , 878, 350, 756, 82, 562, 897, 508, 853, 317 ,
  380 , 807 , 23 , 506 , 98 , 757 , 247