

Data Structures and Algorithms

COSC 336 Assignment 2

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. The exercises will not be graded, but you still need to present your best attempt to solve them. If you do not know how to solve an exercise, say it. This will give me feedback about your understanding of the theoretical concepts.
4. Your programs must be written in Java.
5. 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.

6. You will submit on **Blackboard** 3 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 (see the remark above).
- (b) A short description of your algorithms for the Programming Tasks 1 and 2, where you explain the dynamic programming 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) A table with the results your program gives for the data sets indicated for the programming task.
- (d) The java code (so that the grader can make observations) of the 2 programs.

The **2-nd file** is the .java file containing the java source code for Programming Task 1.

The **3-rd file** is the .java file containing the java source code for Programming Task 2.

Exercise 1.

- a Find a Θ evaluation for the function $(4n + 1)4^{\log(n)}$. (Hint: $4^{\log(n)}$ can be written in a simpler way.)
- b Give an example of two functions $t_1(n)$ and $t_2(n)$ that satisfy the relations: $t_1(n) = \Theta(n^2)$, $t_2(n) = \Theta(n^2)$ and $t_1(n) - t_2(n) = o(n^2)$.
- c Give an example of a function $t_3(n)$ such that $t_3(n) = \Theta(t_3(2n))$.
- d Give an example of a function $t_4(n)$ such that $t_4(n) = o(t_4(2n))$.

(Note: For (b), (c), (d), the functions t_1, t_2, t_3, t_4 you pick must be selected from the common functions we have discussed, namely polynomials, logarithms, exponentials, factorial.)

Exercise 2. Fill the table from Exercise 3-2, page 61 (3-rd edition) in the textbook (also attached below), except row c, as asked in the exercise. For example the entry on the first cell in the top row is “yes” because $\log^k n = O(n^\epsilon)$. (Note: in row c all the entries are “no”, because $n^{\sin n}$ oscillates.)

3-2 Relative asymptotic growths

Indicate, for each pair of expressions (A, B) in the table below, whether A is O , o , Ω , ω , or Θ of B . Assume that $k \geq 1$, $\epsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with “yes” or “no” written in each box.

	A	B	O	o	Ω	ω	Θ
<i>a.</i>	$\lg^k n$	n^ϵ					
<i>b.</i>	n^k	c^n					
<i>c.</i>	\sqrt{n}	$n^{\sin n}$					
<i>d.</i>	2^n	$2^{n/2}$					
<i>e.</i>	$n^{\lg c}$	$c^{\lg n}$					
<i>f.</i>	$\lg(n!)$	$\lg(n^n)$					

Exercise 3. For each of the following program fragments give a $\Theta(\cdot)$ estimation of the running time as a function of n .

- (a)

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

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

for(int j = 0; j < n/2; j++){
    sum++;}
```
- (c)

```
sum = 0;
for (int i = 0; i < n * n; i++) {
    for(int j = 0; j < n * n; j++)
        sum++
}
```
- (d)

```
sum = 0;
for (int i = 1; i < n; i = 2*i)
    sum++
```
- (e)

```
sum = 0;
for (int i = 0; i < n; i++) {
    for(int j = 1; j < n * n; j = 2*j)
        sum++
}
```

Exercise 4. (a) Compute the sum $S_1 = 500 + 501 + 502 + 503 + \dots + 999$ (the sum of all integers from 500 to 999). Do not use a program.

(b) Compute the sum $S_2 = 1 + 3 + 5 + \dots + 999$ (the sum of all odd integers from 1 to 999). Do not use a program.

(c) A group of 30 persons need to form a committee of 4 persons. How many such committees are possible?

(d) Let C_n be the number of committees of 4 persons selected from a group of n persons. Is the estimation $C_n = o(n^3)$ correct? Justify your answer. (Hint: using the formula $\binom{n}{k}$, you can express the number of committees as a function of n .)

Exercise 5. Find a $\Theta(\cdot)$ evaluation for the sum

$$S = 1^2\sqrt{1} + 2^2\sqrt{2} + 3^2\sqrt{3} + \dots + n^2\sqrt{n}.$$

In other words, find a function f such that $S = \Theta(f(n))$.

Show the work for both the upper bound and the lower bound. You can use the technique with integrals, or the method with bounding the terms of the sum.

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, \dots, a_n of length k is given by k indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$ such that $a_{i_1} < a_{i_2} < \dots < a_{i_k}$. 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 contiguous 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, \dots, a_{n-1} . Then, you can calculate in order, one by one, the elements of an array $d[0], \dots, d[n-1]$, 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 $d[1], \dots, d[i-1]$ 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, \dots, a_n of length k is given by k indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$ such that $a_{i_1} < a_{i_2} < \dots < a_{i_k}$. 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