Lab: Dynamic Programming

his document defines the lab for the "Algorithms – Fundamentals (Java)" course @ Software University.

Please submit your solutions (source code) to all below-described problems in Judge.

1. Fibonacci

Write a **dynamic programming** solution for finding n^{th} Fibonacci members.

- $F_0 = 0$
- $F_1 = 1$
- $F_2 = 2$

Examples

Input	Output
2	1
5	5
20	6765

2. Longest Increasing Subsequence

Let's have a sequence of numbers **S = {a_1, a_2, … a_n}**. An **increasing** subsequence is a sequence of numbers within S where each number is **larger** than the previous. We **do not change the relative positions** of the numbers, e.g. we do not move smaller elements to the left to obtain longer sequences. If several sequences with equal length exist, find the left-most of them.

Examples

Input	Output
1 2 5 3 4	1 2 3 4
4 3 2 1	4
4 2 -1 3 5 5	2 3 5

Solution

Dynamic Programming Approach

The LIS problem can be solved by dividing it into sub-problems – for each element at **index `i`** of the **sequence S**, find the LIS in the range [S_0 … S_i].

Example for **S** = { 3, 14, 5, 12, 15, 7, 8, 9, 11, 10, 1 }:

- LIS { 3 } => { 3 }
- LIS { 3, 14 } => { 3, 14 }

- LIS { 3, 14, 5 } => { 3, 5 }
- LIS { 3, 14, 5, 12 } => { 3, 5, 12 }
- etc.

For each index, we'll **keep track of the length of the LIS up to that index** and the **previous index** of the LIS. E.g., the length of the LIS at index **5** is **3**, the longest sequence ending in **`seq[5]`** is {3, 5, 7} and the index of the previous element of the subsequence (the number 5) is 2. The table below illustrates these computations:

index	0	1	2	3	4	5	6	7	8	9	10
S[]	**3**	**14**	**5**	**12**	**15**	**7**	**8**	**9**	**11**	**10**	**1**
len[]	1	2	2	3	4	3	4	5	6	6	1
prev[]	-1	0	0	2	3	2	5	6	7	7	-1
LIS	{3}	{3,14}	{3,5}	{3,5,12}	{3,5,12,15}	{3,5,7}	{3,5,7,8}	{3,5,7,8,9}	{3,5,7,8,9,11}	{3,5,7,8,9,10}	{1}

We need to calculate the info in the above table for every element of the original sequence **S**, so we'll need two additional arrays with lengths equal to the length of **S**. Translating this into code within our method, we get:

```
int[] len = new int[sequence.length];
int[] prev = new int[sequence.length];
```

We also need to keep track of the **length of the longest subsequence** found so far and the index at which it ends (we'll use **-1** to mark that there is no such index found currently):

```
int maxLength = 0;
int bestIndex = -1;
```

Calculate LIS at Each Index

To obtain the longest increasing sequence up to a given index, we just have to find the LIS up to that point to which the current element can be appended as the largest. That is why, when considering the sequence {3, 14, 5} we obtained {3, 5}; we want to know the longest sequence in which the current number (5) participates.

We'll do the following:

- loop through each number in the sequence;
- find the longest sequence up to that point which ends with a number that is smaller than the current.

Remember that we keep track of the length of each LIS in the **`len[]`** array.

```
for (int i = 0; i < sequence.length; i++) {
    int bestLength = 1;
    int index = -1;
    for (int j = i - 1; j >= 0; j--) {
        if (bestLength <= lengths[j] + 1 && sequence[j] < sequence[i]) {
            bestLength = lengths[j] + 1;
            index = j;
        }
    }
    lengths[i] = bestLength;
    if (bestLength > maxLength) {
        maxLength = bestLength;
        bestIndex = i;
    }

    indeces[i] = index;
}
```

Don't forget to keep track of the length of the longest increasing subsequence and the index of its last element.

Recover the LIS by Walking through prev[]

Knowing the index of the last element of the LIS, we can get the whole sequence by continuously taking each previous element using the info we keep in the **`prev[]`** array. Store the elements in a list, reverse it and return it:

3. Move Down/Right

Given a **matrix of N by M** cells filled with positive integers, find the path from **top left** to **bottom right** with the **highest sum** of cells by moving only down or right.

Examples

Input	Output
4 4 1 3 2 1 5 3 2 1 1 7 3 1 1 3 1 1	[0, 0] [1, 0] [1, 1] [2, 1] [3, 1] [3, 2] [3, 3]
3 3 1 1 1 1 1 1 1 1 1	[0, 0] [1, 0] [2, 0] [2, 1] [2, 2]

3 3 1 0 6 8 3 7 2 4 9	[0, 0] [1, 0] [1, 1] [1, 2] [2, 2]

4. Rod Cutting

Find the best way to cut up a rod with a specified length. You are also given prices of all possible lengths starting from 0.

Examples

Input	Output
0 1 5 8 9 10 17 17 20 24 30 4	10 2 2
0 1 5 8 9 10 17 17 20 24 30 8	22 2 6
0 1 5 8 9 10 17 17 20 24 30 10	30 10