Task 1: Revive the project :
I imported the project and converted it to a maven project. When I imported the project, there were problems in the RandomPyramidGenerator class that were caused by the Java version, so I upgraded it to Java 17 and the problem was cured. To resolve the error in YourSolverTest, I must also include a junit4 dependency in pom.xml. Now I can run the project, and when I run the OurProgram class, I get the following result in the console.

```
[00066] [00095] [00012] [00029] [00048]
   [00075] [00002] [00026] [00037]
      [00093] [00041] [00074]
         [00062] [00073]
            [00025]
```

This result is wrong, do you know why ?
255

It is now time to consider what is wrong with the code.

Task 2: NaivePyramidSolver :
As a result, what I'm getting currently is incorrect. So I examined the NaivePyramidSolver implementation. At this point, I can tell the issue is with getTotalAbove(int row, int column, Pyramid pyramid) and the problem is with if condition i.e. if (row == 0) return 0; we are returning 0 when row index value is 0 which is incorrect because array is 0 index based. As a result, I altered it to if (row == -1) return 0; to consider 0th index row too.

```
  private long getTotalAbove(int row, int column, Pyramid pyramid) {
      //if (row == 0) return 0;
       if (row == -1) return 0;

      int myValue = pyramid.get(row, column);
      long left  = myValue + getTotalAbove(row - 1, column, pyramid);
      long right = myValue + getTotalAbove(row - 1, column + 1, pyramid);
      return Math.max(left, right);
  }
```

I've added test cases to demonstrate that NaivePyramidSolver finds the max sum path in a pyramid.

Task 3: YourSolver
The NaivePyramidSolver solution is a recursive strategy that takes time to process the enormous data set. I employed the DP in a bottom-up approach to reduce time complexity. With my implementation, the solver's time

complexity is lowered, and all YourSolverTest test cases pass.