p7-ex1

We will use dynamic programming, exploiting the hint.

dp[i][j] = minimum value in the subsequence starting from index i of lenght 2^j

Obs: i lies in {0,..., n-1}, j lies in {0,..., floor(log(n))}.

How do we fill the table?
We compute minimum values starting from the shortest subsequences up to the largest ones.
Once the lenght is fixed, i.e. j is fixed, we compute values starting from earlier indeces (so top-down, left to right).
The idea is to compare the minimum of the first half with the minimum of the second half, simply choosing the smallest. (The two halfs have lenght 2^(j-1), so they have already been computed)

Initialize dp with inf

```
dp[i][0] = A[i]    //base case

for j = 0...floor(log(n))
   for i = 1...n-1

      if (i + 2^j) <= n    //avoiding impossible scenarios (for example, the only sequence starting from the last element has lenght 2^0 = 1)

         dp[i][j] = min(dp[i][j - 1], dp[i + 2^(j-1)][j - 1])
```

Note: this preprocessing takes O(nlog(n)).

How do we answer the query “Given l and r, what is min{ A[l] , A[l +1], . . . , A[r] }?”

Let’s compute the largest power of two smaller than (r - l + 1), that is: the largest k such that 2^k <= len(A[l]...A[r]).

The answer to the query will be:

min(dp[l][k], dp[r - 2^k + 1][k])

The idea here is to compare the minimums of the two overlapping intervals of lenght 2^k respectively starting from A[l] and ending to A[r].
Our choice of k ensures that the two intervals are actually overlapping: hence, we are considering all the integers from l to r.

Since l and r are given, computing k is O(1) and accessing dp table elements is O(1).