**[Partition Array for Maximum Sum](https://leetcode.com/problems/partition-array-for-maximum-sum/)**[**Partition Array for Maximum Sum**](https://leetcode.com/problems/partition-array-for-maximum-sum/)

Given an integer array arr, partition the array into (contiguous) subarrays of length **at most** k. After partitioning, each subarray has their values changed to become the maximum value of that subarray.

Return *the largest sum of the given array after partitioning. Test cases are generated so that the answer fits in a****32-bit****integer.*

**Example 1:**

**Input:** arr = [1,15,7,9,2,5,10], k = 3

**Output:** 84

**Explanation:** arr becomes [15,15,15,9,10,10,10]

**Example 2:**

**Input:** arr = [1,4,1,5,7,3,6,1,9,9,3], k = 4

**Output:** 83

**Example 3:**

**Input:** arr = [1], k = 1

**Output:** 1

**Constraints:**

* 1 <= arr.length <= 500
* 0 <= arr[i] <= 109
* 1 <= k <= arr.length

OUTPUT : -

class Solution {

public:

    int maxSumAfterPartitioning(vector<int>& arr, int k) {

        int N = arr.size();

        int K = k + 1;

        int dp[k + 1];

        memset(dp, 0, sizeof(dp));

        for (int start = N - 1; start >= 0; start--) {

            int currMax = 0;

            int end = min(N, start + k);

            for (int i = start; i < end; i++) {

                currMax = max(currMax, arr[i]);

                dp[start % K] = max(dp[start % K], dp[(i + 1) % K] + currMax \* (i - start + 1));

            }

        }

        return dp[0];

    }

};

LINK : - <https://leetcode.com/problems/partition-array-for-maximum-sum/description/?envType=daily-question&envId=2024-02-03>