



# 1043. Partition Array for Maximum Sum

■ DSA 🧙	
■	55
■ topic 🐜	

## 📌 Problem Summary: Max Sum After Partitioning

Given an array `arr` and an integer `k`, split the array into contiguous subarrays of size  $\leq k$ . Replace each subarray with its maximum value repeated `len(subarray)` times. Return the maximum total sum after this transformation.

## ✨ Intuition

- At every index `i`, we can take a **partition of size 1 to k**, calculate:
  - Max element in the partition
  - Partition contribution = `max * length`
  - Then recursively calculate for the remaining array from `j+1`
- Use **recursion** to try all partitions, and return the **maximum possible sum**.

## 🧠 Recursive Logic Breakdown

For each `i` from `0` to `n-1`:

- Try all possible partition sizes `1 to k`
    - `j` goes from `i` to `min(i+k, n)`
    - Track the maximum element in `arr[i..j]`
    - Compute:
      - `cur_sum = max_element * (j - i + 1)`
      - `total = cur_sum + recursion(j+1)`
  - Take the **maximum** of all these totals.
- 

### Dry-Run with Example:

Let `arr = [1,15,7,9,2,5,10]` , `k = 3`

At `i = 0` :

- Try `[1]` → `max=1` → `1×1 + solve([ 15,7,9,2,5,10 ])`
- Try `[1,15]` → `max=15` → `15×2 + solve([ 7,9,2,5,10 ])`
- Try `[1,15,7]` → `max=15` → `15×3 + solve([ 9,2,5,10 ])`

We choose the one with max sum.

---

### Final Thoughts:

- Think of this as "**cutting a rope**" into segments of size  $\leq k$ .
  - At each cut, **maximize** the value you get from the segment.
  - This problem is a classic **DP with partitioning** pattern.
- 

### Trick to Remember:

- Use the variable `max_ele` to grow dynamically.
- Multiply by the number of elements.
- Recurse from the next index after the partition.
- Store or memoize to avoid recomputation (in top-down DP version).