

# 2636. Promise Pool

## Description

Given an array of asynchronous functions `functions` and a **pool limit** `n`, return an asynchronous function `promisePool`. It should return a promise that resolves when all the input functions resolve.

**Pool limit** is defined as the maximum number promises that can be pending at once. `promisePool` should begin execution of as many functions as possible and continue executing new functions when old promises resolve. `promisePool` should execute `functions[i]` then `functions[i + 1]` then `functions[i + 2]`, etc. When the last promise resolves, `promisePool` should also resolve.

For example, if `n = 1`, `promisePool` will execute one function at a time in series. However, if `n = 2`, it first executes two functions. When either of the two functions resolve, a 3rd function should be executed (if available), and so on until there are no functions left to execute.

You can assume all `functions` never reject. It is acceptable for `promisePool` to return a promise that resolves any value.

### Example 1:

**Input:**  
functions = [  
 () => new Promise(res => setTimeout(res, 300)),  
 () => new Promise(res => setTimeout(res, 400)),  
 () => new Promise(res => setTimeout(res, 200))  
]  
n = 2  
**Output:** [[300,400,500],500]  
**Explanation:**  
Three functions are passed in. They sleep for 300ms, 400ms, and 200ms respectively. They resolve at 300ms, 400ms, and 500ms respectively. The returned promise resolves at 500ms. At t=0, the first 2 functions are executed. The pool size limit of 2 is reached. At t=300, the 1st function resolves, and the 3rd function is executed. Pool size is 2. At t=400, the 2nd function resolves. There is nothing left to execute. Pool size is 1. At t=500, the 3rd function resolves. Pool size is zero so the returned promise also resolves.

### Example 2:

**Input:**  
functions = [  
 () => new Promise(res => setTimeout(res, 300)),  
 () => new Promise(res => setTimeout(res, 400)),  
 () => new Promise(res => setTimeout(res, 200))  
]  
n = 5  
**Output:** [[300,400,200],400]  
**Explanation:**  
The three input promises resolve at 300ms, 400ms, and 200ms respectively. The returned promise resolves at 400ms. At t=0, all 3 functions are executed. The pool limit of 5 is never met. At t=200, the 3rd function resolves. Pool size is 2. At t=300, the 1st function resolved. Pool size is 1. At t=400, the 2nd function resolves. Pool size is 0, so the returned promise also resolves.

### Example 3:

**Input:**  
functions = [  
 () => new Promise(res => setTimeout(res, 300)),  
 () => new Promise(res => setTimeout(res, 400)),  
 () => new Promise(res => setTimeout(res, 200))  
]  
n = 1  
**Output:** [[300,700,900],900]  
**Explanation:**  
The three input promises resolve at 300ms, 700ms, and 900ms respectively. The returned promise resolves at 900ms. At t=0, the 1st function is executed. Pool size is 1. At t=300, the 1st function resolves and the 2nd function is executed. Pool size is 1. At t=700, the 2nd function resolves and the 3rd function is executed. Pool size is 1. At t=900, the 3rd function resolves. Pool size is 0 so the returned promise resolves.

### Constraints:

- `0 <= functions.length <= 10`
- `1 <= n <= 10`

