# 621. Task Scheduler

You are given an array of CPU `tasks`, each represented by letters A to Z, and a cooling time, `n`. Each cycle or interval allows the completion of one task. Tasks can be completed in any order, but there's a constraint: **identical** tasks must be separated by at least `n` intervals due to cooling time.

Return the *minimum number of intervals* required to complete all tasks.

**Example 1:**

**Input:** tasks = ["A","A","A","B","B","B"], n = 2

**Output:** 8

**Explanation:** A possible sequence is: A -> B -> idle -> A -> B -> idle -> A -> B.

After completing task A, you must wait two cycles before doing A again. The same applies to task B. In the 3rd interval, neither A nor B can be done, so you idle. By the 4th cycle, you can do A again as 2 intervals have passed.

**Example 2:**

**Input:** tasks = ["A","C","A","B","D","B"], n = 1

**Output:** 6

**Explanation:** A possible sequence is: A -> B -> C -> D -> A -> B.

With a cooling interval of 1, you can repeat a task after just one other task.

**Example 3:**

**Input:** tasks = ["A","A","A", "B","B","B"], n = 3

**Output:** 10

**Explanation:** A possible sequence is: A -> B -> idle -> idle -> A -> B -> idle -> idle -> A -> B.

There are only two types of tasks, A and B, which need to be separated by 3 intervals. This leads to idling twice between repetitions of these tasks.

**Constraints:**

- `1 <= tasks.length <= 104`
- `tasks[i]` is an uppercase English letter.
- `0 <= n <= 100`

```java
class Solution {
    public int leastInterval(char[] tasks, int n) {

        int[] freq = new int[26];

        for(char task : tasks){
            freq[task - 'A']++;
        }

        Arrays.sort(freq);

        int chunk = freq[25] -1;
        int idle = chunk * n;


        for(int i = 24; i>=0; i--){
            idle -= Math.min(chunk, freq[i]);
        }

        return idle < 0 ? tasks.length : tasks.length +idle;
    }
}
```