2530. Maximal Score After Applying K Operations

You are given a **0-indexed** integer array `nums` and an integer `k`. You have a **starting score** of `0`.

In one **operation**:

1. choose an index `i` such that `0 <= i < nums.length`,
2. increase your **score** by `nums[i]`, and
3. replace `nums[i]` with `ceil(nums[i] / 3)`.

Return *the maximum possible* ***score*** *you can attain after applying* ***exactly*** `k` *operations*.

The ceiling function `ceil(val)` is the least integer greater than or equal to `val`.

Example 1:

```
Input: nums = [10,10,10,10,10], k = 5
Output: 50
Explanation: Apply the operation to each array element exactly once. The final
score is 10 + 10 + 10 + 10 + 10 = 50.
```

Example 2:

```
Input: nums = [1,10,3,3,3], k = 3
Output: 17
Explanation: You can do the following operations:
Operation 1: Select i = 1, so nums becomes [1,4,3,3,3]. Your score increases by 10.
Operation 2: Select i = 1, so nums becomes [1,2,3,3,3]. Your score increases by 4.
Operation 3: Select i = 2, so nums becomes [1,1,1,3,3]. Your score increases by 3.
The final score is 10 + 4 + 3 = 17.
```

Constraints:

- `1 <= nums.length, k <=` 10^5
- `1 <= nums[i] <=` 10^9

2530. Maximal Score After Applying K Operations

```
/*
    **max heap**
    Time complexity: O(klogn)
    space complexity: O(n)
*/
class Solution {
    public:
        long long maxKelements(vector<int>& nums,int k){
            std::priority_queue<int,std::vector<int>> max_heap;
            for(auto& e: nums) max_heap.push(e);
            long long score=0;
            while(k--){
                int mx=max_heap.top();
                max_heap.pop();
                score+=mx;
                mx=ceil((double)mx/3.0);
                max_heap.push(mx);
            }
            return score;
        }
};
```