

# 2558. Take Gifts From the Richest Pile

Easy   Array   Simulation   Heap (Priority Queue)

[Leetcode Link](#)

## Problem Description

In this problem, you are given an array called `gifts`, which contains integers representing the number of gifts in different piles. You have a task to perform every second for `k` seconds, and the task goes as follows:

- Identify the pile with the most gifts. If there are multiple piles with the same maximum number of gifts, you may choose any one of them.
- Leave behind the largest integer less than or equal to the square root of the number of gifts in that pile, also known as the floor of the square root. Take the rest of the gifts from the pile.

After performing this task every second for `k` seconds, you need to determine the total number of gifts left across all piles.

## Intuition

To efficiently manage the process of finding and updating the largest pile every second, a max heap data structure is employed. A max heap is a binary tree where the parent node is always greater than its children nodes, making it easy to retrieve and remove the maximum element in the heap.

Here is the intuition behind the solution approach:

- Convert the list of gifts into a max heap. In Python, a max heap is not provided by default, but a min heap is provided by the `heapq` library. Therefore, we can simulate a max heap by negating all values in the `gifts` array when adding them to the heap. This way, the smallest number (after negation, which was actually the largest) comes at the top of the heap.
- Iterate `k` times to simulate the seconds passing. During each iteration, pop the maximum element from the heap (which will be returned as a negative number due to our earlier negation), calculate the floor of the square root of that number (negate it again to maintain the heap property), and put the result back into the heap.
- After doing this for `k` seconds, the heap contains negative numbers representing the gifts left in each pile. Sum up the negative numbers and negate the result to get the total number of gifts remaining.

This approach lets us update the piles and find out the total number of remaining gifts efficiently.

## Solution Approach

The key data structure used in the solution approach is a heap, which is a specialized tree-based data structure that satisfies the heap property. In a max heap, for any given node `i`, the value of `i` is greater than or equal to the values of its children, and the maximum element is always at the root node. The algorithm utilizes a min heap with negated values to mimic the behavior of a max heap, as Python's `heapq` module only provides a min heap implementation.

The process of the algorithm is as follows:

- We first negate all the values in the `gifts` array and convert this negated list into a heap using `heapify`. The `heapify` function transforms the list into a heap in  $O(n)$  time.

```
1 h = [-v for v in gifts]
2 heapify(h)
```

- We then repeatedly perform `k` iterations to simulate each second. In each iteration, we use the `heapreplace` function, which first pops the root element of the heap (the smallest element, or, in our negated heap, the pile with the maximum gifts), then calculates the floor of the square root of the negated number (effectively the square root of the original maximum), negates it (to keep it consistent with our negated heap), and finally places this value back into the heap.

```
1 for _ in range(k):
2     heapreplace(h, -int(sqrt(-h[0])))
```