Maximum Cost of Ropes

There are given **N** ropes of different lengths, we need to connect these ropes into one rope. The cost to connect two ropes is equal to sum of their lengths.
The task is to connect the ropes with maximum cost. Given **N** size array **arr[]** contains the lengths of the ropes.

Example: [4, 3, 2, 6] ⇒ Sum: (6 + 4 = 10) ⇒ [3, 2, 10] ⇒ Sum: (10 + (10 + 3) = 23) ⇒ [2, 13] ⇒ Sum: (23 + (13 + 2) = 38)

Output: 38

Approach 1: Function to find the maximum rope cost using a brute force approach

- **Function Purpose:** To find the maximum rope cost using a brute force approach.
- **Explanation:**
    - The **maxRopeCostBruteForce** function iteratively finds the two largest elements in the array.
    - It calculates the cost of merging these two largest elements and updates the array by replacing them with the merged element.
    - The process continues until only one element remains in the array, and the total rope cost is calculated.
- **Time Complexity: O(n^2), where n is the size of the array, because for each element, the function finds the two largest elements.**
- **Space Complexity: O(1) as no extra data structures are used.**

Approach 2: Function to find the maximum rope cost using an optimized approach with a max heap

- **Function Purpose:** To find the maximum rope cost using an optimized approach with a max heap.
- **Explanation:**
    - The **maxRopeCost** function uses a max heap (priority queue) to efficiently keep track of the largest elements.
    - It initializes a priority queue with the elements from the input array, ensuring the largest elements are always at the front.
    - The function repeatedly extracts the two largest elements, calculates the cost of merging them, and inserts the merged element back into the max heap.

    - The process continues until only one element remains in the heap, and the total rope cost is calculated.
- **Time Complexity: O(n * log(n)), as each insertion and extraction operation on the max heap takes logarithmic time.**
- **Space Complexity: O(n) for the max heap.**

Conclusion:

- **Approach 2, which uses an optimized approach with a max heap, is significantly more efficient with a time complexity of O(n * log(n)) compared to the O(n^2) time complexity of the brute force approach**.