<https://leetcode.ca/all/1167.html>

You have some sticks with positive integer lengths.

You can connect any two sticks of lengths X and Y into one stick by paying a cost of X + Y.  You perform this action until there is one stick remaining.

Return the minimum cost of connecting all the given sticks into one stick in this way.

Example 1:

Input: sticks = [2,4,3]

Output: 14

Example 2:

Input: sticks = [1,8,3,5]

Output: 30

Constraints:

1 <= sticks.length <= 10^4

1 <= sticks[i] <= 10^4

**Attempt 1: 2023-11-10**

**Solution 1: Priority Queue (10 min)**

class Solution {

public int connectSticks(int[] sticks) {

PriorityQueue<Integer> minPQ = new PriorityQueue<>();

for(int stick : sticks){

minPQ.add(stick);

}

int cost = 0;

// Pull out two shortest sticks from the minPQ, connect them into one stick

// and put it back in the minPQ, repeat the process till there is only one

// stick left in the minPQ

while(minPQ.size() > 1){

int a = minPQ.poll();

int b = minPQ.poll();

int c = a + b;

cost = cost + c;

minPQ.offer(c);

}

return cost;

}

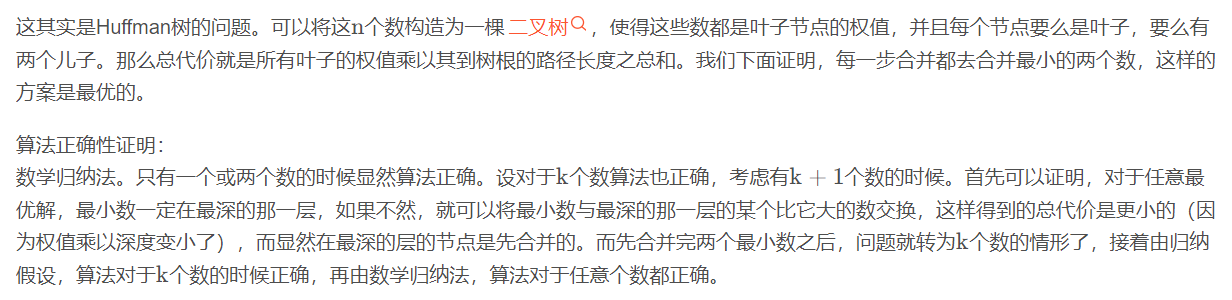
}

Time Complexity: O(N\*logN)

Space Complexity: O(N)

**Refer to**

<https://blog.csdn.net/qq_46105170/article/details/113750503>



class Solution {

public int connectSticks(int[] sticks) {

//Priority Queue: sort sticks based on their length with smallest stick on the top

PriorityQueue<Integer> pq = new PriorityQueue<>(new Comparator<Integer>(){

public int compare(Integer a, Integer b){

return Integer.compare(a, b);

}

});

//add all the sticks

for(int stick: sticks){

pq.add(stick);

}

int cost = 0;

//pull out two sticks from the priority Queue

//connect them into stick and put it back in the priority queue

// repeat the process till there is only one stick left in the priority queue

while(pq.size()> 1){

int a = pq.poll();

int b = pq.poll();

int c = a+b;

cost = cost + c;

pq.offer(c);

}

return cost;

}

}