<https://leetcode.com/problems/minimum-cost-tree-from-leaf-values/description/>

Given an array arr of positive integers, consider all binary trees such that:

Each node has either 0 or 2 children;

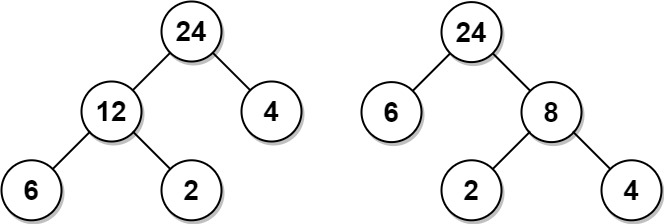
The values of arr correspond to the values of each leaf in an in-order traversal of the tree.

The value of each non-leaf node is equal to the product of the largest leaf value in its left and right subtree, respectively.

Among all possible binary trees considered, return the smallest possible sum of the values of each non-leaf node. It is guaranteed this sum fits into a 32-bit integer.

A node is a leaf if and only if it has zero children.

**Example 1:**



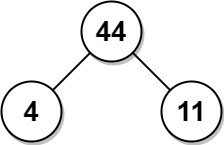
**Input:** arr = [6,2,4]

**Output:** 32

**Explanation:** There are two possible trees shown.

The first has a non-leaf node sum 36, and the second has non-leaf node sum 32.

**Example 2:**



**Input:** arr = [4,11]

**Output:** 44

**Constraints:**

2 <= arr.length <= 40

1 <= arr[i] <= 15

It is guaranteed that the answer fits into a **32-bit** signed integer (i.e., it is less than 2^31).

**Attempt 1: 2024-11-06**

**Solution 1: Monotonic Decreasing Stack (180min)**

class Solution {

    public int mctFromLeafValues(int[] arr) {

        int result = 0;

        Stack<Integer> stack = new Stack<>();

        stack.push(Integer.MAX\_VALUE);

        for(int a : arr) {

            while(stack.peek() <= a) {

                int cur\_min = stack.pop();

                result += cur\_min \* Math.min(stack.peek(), a);

            }

            stack.push(a);

        }

        while(stack.size() > 2) {

            result += stack.pop() \* stack.peek();

        }

        return result;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/minimum-cost-tree-from-leaf-values/solutions/339959/one-pass-o-n-time-and-space/>

**DP Solution**

Find the cost for the interval [i,j].

To build up the interval [i,j], we need to split it into left subtree and sub tree,

dp[i, j] = dp[i, k] + dp[k + 1, j] + max(A[i, k]) \* max(A[k + 1, j])

If you don't understand dp solution, I won't explain it more and you won't find the answer here.

Take your time, read any other solutions, and come back at your own will.

If you got it, continue to read.

**DP Complexity**

Second question after this dp solution,

**what's the complexity?**

N^2 states and O(N) to find each.

So this solution is O(N^3) time and O(N^2) space.

You thought it's fine.

After several nested for loop, you got a happy green accepted.

You smiled and released a sigh as a winner.

What a great practice for DP skill!

Then you noticed it's medium.

That's it, just a standard medium problem of dp.

Nothing can stop you. Even dp problem.

**True story**

So you didn't Read and Upvote this post.

One day, you meet exactly the same solution during an interview.

Your heart welled over with joy, and you bring up your solution with confidence.

One week later, you receive an email.

The second paragraph starts with a key word "Unfortunately".

**What the heck!?**

You solved the interview problem perfectly, but the company didn't appreciate your talent.

What's more on earth did they want?

**WHY?**

Here is the reason.

This is not a dp problem at all.

Because dp solution test all ways to build up the tree, including many unnecessay tries.

Honestly speaking, it's kinda of brute force.

Yes, brute force testing, with memorization.

**Intuition**

Let's review the problem again.

When we build a node in the tree, we compared the two numbers a and b.

In this process, the smaller one is removed and we won't use it anymore, and the bigger one actually stays.

**The problem can translated as following:**

**Given an array A, choose two neighbors in the array a and b, we can remove the smaller one min(a,b) and the cost is a \* b.**

**What is the minimum cost to remove the whole array until only one left?**

**To remove a number a, it needs a cost a \* b, where b >= a.**

**So a has to be removed by a bigger number.**

**We want minimize this cost, so we need to minimize b, b has two candidates, the first bigger number on the left, the first bigger number on the right.**

**The cost to remove a is a \* min(left, right).**

**Solution 1**

With the intuition above in mind, the explanation is short to go.

We remove the element form the smallest to bigger.

We check the min(left, right),

For each element a, cost = min(left, right) \* a

Time O(N^2)

Space O(N)

**Python**

def mctFromLeafValues(self, A):

res = 0

while len(A) > 1:

i = A.index(min(A))

res += min(A[i - 1:i] + A[i + 1:i + 2]) \* A.pop(i)

return res

**Solution 2: Stack Soluton**

we decompose a hard problem into reasonable easy one:

Just find the next greater element in the array, on the left and one right.

Refer to the problem 503. Next Greater Element II

Time O(N) for one pass

Space O(N) for stack in the worst cases

**Java**

public int mctFromLeafValues(int[] A) {

int res = 0;

Stack<Integer> stack = new Stack<>();

stack.push(Integer.MAX\_VALUE);

for (int a : A) {

while (stack.peek() <= a) {

int mid = stack.pop();

res += mid \* Math.min(stack.peek(), a);

}

stack.push(a);

}

while (stack.size() > 2) {

res += stack.pop() \* stack.peek();

}

return res;

}

**Refer to**

**1.Take array [6,2,4,5] as an example.**

30

/ \

6 20

/ \

8 5

/ \

2 4

Actually what we do is:

For[6,2,4,5], 2 is the smallest, 2 \* Math.min(6, 4) = 2 \* 4 = 8;

For[6, 4, 5], 4 is the samllest, 4 \* Math.min(6, 5) = 4 \* 5 = 20;

For[6,5], 6\*5 = 30.

So result = 8 + 20 + 30 = 58.

The key is that if one element e is smaller than its two neighbors, result adds e \* Math.min(two neighbors).

**2.Take array [15,13,5,3,15] as example**

idea: make sure always use smallest integers to get product

e.g for [15,13,5,3,15], the initial smallest integers are 5 and 3, we can use Monotone Stack to get the smallest integers in one pass

use Stack: if currentVal >= stack.peek(), we should make sure stack.peek() multiply the smallest value

Final tree build to guarantee always smallset integers get product first

N1

/ \

N2 15

/ \

15 N3

/ \

13 N4

/ \

5 3

So the actual procedure for above stack is:

3 \* 5 ------> N4

+ 5 \* 13 ------> N3

\* 13 \* 15 ------> N2

+ 15 \* 15 ------> N1

**Refer to**

(单调栈) O(n)

题解2:单调栈。根据题意我们知道，每次两个相邻元素组成一棵子树后，会将较小的元素删去，留下较大的元素。所以我们的目标就是每次删除局部最小的那个元素。

比如[6,2,4]中2就是局部最小因为他小于左右两边的元素。我们将局部最小的元素和两边较小的元素相乘加入答案，同时将这个局部最小的元素抹去。如：

[6,2,4,8] res = 0

[6,4,8] res = 8

[6,8] res = 8 + 24

[8] res = 8 + 24 + 48

我们考虑使用一个单调递减栈来存储数组元素，如果当前元素小于等于栈顶元素直接入栈；否则，说明当前栈顶元素是一个局部最小值，我们用这个局部最小值的

两端较小与栈顶元素相乘，同时将栈顶元素出栈，重复上述操作，直至栈顶元素不是局部最小值，再将当前元素入栈。处理完整个数组后，如果栈中还有多的元素，

我们从栈顶依次往下乘加入答案即可。

这里为了处理边界情况，先在栈中插入了一个INT\_MAX。

时间复杂度：每个元素入栈一次，出栈一次。所以总的时间复杂度是O(n)。

**Refer to**

[L739.Daily Temperatures](note://6FB3A11E2B7746DCA411301F941DD948)

[L503.Next Greater Element II (Refer L496.Next Greater Element I)](note://87BFD7FE36A8434A8DB6259C196945A9)