

## 553. Optimal Division

April 15, 2017 | 19.1K views

PreviousNext

★★★★★

Average Rating: 4.67 (21 votes)

Given a list of **positive integers**, the adjacent integers will perform the float division. For example,  $[2,3,4] \rightarrow 2 / 3 / 4$ .

However, you can add any number of parenthesis at any position to change the priority of operations. You should find out how to add parenthesis to get the **maximum** result, and return the corresponding expression in string format. **Your expression should NOT contain redundant parenthesis.**

**Example:**

```
Input: [1000,100,10,2]
Output: "1000/(100/10/2)"
Explanation:
1000/(100/10/2) = 1000/((100/10)/2) = 200
However, the bold parenthesis in "1000/((100/10)/2)" are redundant,
since they don't influence the operation priority. So you should return "1000/(100/10/2)"

Other cases:
1000/(100/10)/2 = 50
1000/(100/(10/2)) = 50
1000/100/10/2 = 0.5
1000/100/(10/2) = 2
```

**Note:**

- The length of the input array is  $[1, 10]$ .
- Elements in the given array will be in range  $[2, 1000]$ .
- There is only one optimal division for each test case.

## Solution

### Approach #1 Brute Force [Accepted]

#### Algorithm

Brute force of this problem is to divide the list into two parts *left* and *right* and call function for these two parts. We will iterate *i* from *start* to *end* so that *left* = (*start*, *i*) and *right* = (*i* + 1, *end*).

*left* and *right* parts return their maximum and minimum value and corresponding strings.

Minimum value can be found by dividing minimum of left by maximum of right i.e. *minVal* = *left.min*/*right.max*.

Similarly, Maximum value can be found by dividing maximum of left value by minimum of right value. i.e. *maxVal* = *left.max*/*right.min*.

Now, how to add parenthesis? As associativity of division operator is from left to right i.e. by default left most divide should be done first, we need not have to add parenthesis to the left part, but we must add parenthesis to the right part.

eg- "2/(3/4)" will be formed as leftPart+"/"+"(+rightPart+)", assuming leftPart is "2" and rightPart is "3/4".

One more point, we also don't require parenthesis to right part when it contains single digit.

eg- "2/3", here left part is "2" and right part is "3" (contains single digit) . 2/(3) is not valid.

```
Java
1 public class Solution {
2     public String optimalDivision(int[] nums) {
3         T t = optimal(nums, 0, nums.length - 1, "");
4         return t.max_str;
5     }
6     class T {
7         float max_val, min_val;
8         String min_str, max_str;
9     }
10    public T optimal(int[] nums, int start, int end, String res) {
11        T t = new T();
12        if (start == end) {
13            t.max_val = nums[start];
14            t.min_val = nums[start];
15            t.min_str = "" + nums[start];
16            t.max_str = "" + nums[start];
17            return t;
18        }
19        t.min_val = Float.MAX_VALUE;
20        t.max_val = Float.MIN_VALUE;
21        t.min_str = t.max_str = "";
22        for (int i = start; i < end; i++) {
23            T left = optimal(nums, start, i, "");
24            T right = optimal(nums, i + 1, end, "");
25            if (t.min_val > left.min_val / right.max_val) {
26                t.min_val = left.min_val / right.max_val;
27                t.min_str = left.min_str + "/" + (i + 1 != end ? "(" : "") + right.max_str + (i + 1 !=
28                end ? ")" : "");
29            }
30        }
31    }
32 }
```

#### Complexity Analysis

- Time complexity :  $O(n!)$ . Number of permutations of expression after applying brackets will be in  $O(n!)$  where *n* is the number of items in the list.
- Space complexity:  $O(n^2)$ . Depth of recursion tree will be  $O(n)$  and each node contains string of maximum length  $O(n)$ .

### Approach #2 Using Memorization [Accepted]

#### Algorithm

In the above approach we called optimal function recursively for ever *start* and *end*. We can notice that there are many redundant calls in the above approach, we can reduce these calls by using memorization to store the result of different function calls. Here, *memo* array is used for this purpose.

```
Java
16 t.max_val = nums[start];
17 t.min_val = nums[start];
18 t.min_str = "" + nums[start];
19 t.max_str = "" + nums[start];
20 memo[start][end] = t;
21 return t;
22 }
23 t.min_val = Float.MAX_VALUE;
24 t.max_val = Float.MIN_VALUE;
25 t.min_str = t.max_str = "";
26 for (int i = start; i < end; i++) {
27     T left = optimal(nums, start, i, "", memo);
28     T right = optimal(nums, i + 1, end, "", memo);
29     if (t.min_val > left.min_val / right.max_val) {
30         t.min_val = left.min_val / right.max_val;
31         t.min_str = left.min_str + "/" + (i + 1 != end ? "(" : "") + right.max_str + (i + 1 !=
32         end ? ")" : "");
33     }
34     if (t.max_val < left.max_val / right.min_val) {
35         t.max_val = left.max_val / right.min_val;
36         t.max_str = left.max_str + "/" + (i + 1 != end ? "(" : "") + right.min_str + (i + 1 !=
37         end ? ")" : "");
38     }
39     memo[start][end] = t;
40     return t;
41 }
```

#### Complexity Analysis

- Time complexity :  $O(n^3)$ . *memo* array of size  $n^2$  is filled and filling of each cell of the *memo* array takes  $O(n)$  time.
- Space complexity :  $O(n^3)$ . *memo* array of size  $n^2$  where each cell of array contains string of length  $O(n)$ .

### Approach #3 Using some Math [Accepted]

#### Algorithm

Using some simple math we can find the easy solution of this problem. Consider the input in the form of [a,b,c,d], now we have to set priority of operations to maximize a/b/c/d. We know that to maximize fraction *p/q*, *q* (denominator) should be minimized. So, to maximize a/b/c/d we have to first minimize b/c/d. Now our objective turns to minimize the expression b/c/d.

There are two possible combinations of this expression, b/(c/d) and (b/c)/d.

b/(c/d)	(b/c)/d = b/c/d
(b*d)/c	b/(d*c)
d/c	1/(d*c)

Obviously,  $d/c > 1/(d * c)$  for  $d > 1$ .

You can see that second combination will always be less than first one for numbers greater than 1. So, the answer will be a/(b/c/d). Similarly for expression like a/b/c/d/e/..., answer will be a/(b/c/d/e/f...).

```
Java
1 public class Solution {
2     public String optimalDivision(int[] nums) {
3         if (nums.length == 1)
4             return nums[0] + "";
5         if (nums.length == 2)
6             return nums[0] + "/" + nums[1];
7         StringBuilder res = new StringBuilder(nums[0] + "/" + nums[1]);
8         for (int i = 2; i < nums.length; i++) {
9             res.append("/" + nums[i]);
10        }
11        res.append("(");
12        return res.toString();
13    }
14 }
15 }
```

#### Complexity Analysis


- Time complexity :  $O(n)$ . Single loop to traverse *nums* array.
- Space complexity :  $O(n)$ . *res* variable is used to store the result.

Rate this article: ★★★★★

PreviousNext


Comments: 19

Sort By

- 

Type comment here... (Markdown is supported)

Preview

Post
- 

zhoubowei

★ 871


April 14, 2018 9:44 PM

such a boring problem ("v")

25

Share

Reply

SHOW 4 REPLIES
- 

Kaa1el

★ 43

May 20, 2017 4:58 AM


your math proof is far from perfect... If a0/.../an = p/q, how do you know p has to be a0 and q has to be (a1/.../an)?

as a learner of algebraic number theory, I will give the following proper proof:

12

Share

Reply

Read More
- 

leetcode\_deleted\_user

★ 275

December 27, 2017 12:39 PM

@vinod23, for the approach #1 & #2, there are two places can be optimized, see details below:


1. Remove "String res" from signature of method optimal();

2. Remove repeated statement of `(i + 1 != end ? "(" : "")`, here is my sample:

1

Share

Reply

Read More
- 

Simsso

★ 0

July 27, 2019 11:06 PM


Short Python solution:

```
def opt_div_2(nums: list) -> str:
    if len(nums) == 1: return str(nums[0])
    if len(nums) == 2: return f'{nums[0]}/{nums[1]}
```

0

Share

Reply

Read More
- 

hankzhangcb


★ 0

July 23, 2019 12:22 PM

Your math is blowing my mind.

0

Share

Reply
- 

Jasim9


★ 2

July 8, 2019 7:39 AM

Approach 2: It is Memorization not Memorization.

0

Share

Reply
- 

domfarolino


★ 3

October 8, 2018 11:08 AM

Wouldn't the time complexity of the first approach be bounded by the nth Catalan number? For example, see the solution to 1, and 2 for reference. It seems like a very similar approach.

0

Share

Reply
- 

usc\_trojans

★ 4

July 5, 2018 12:55 AM

Have a simpler proof for math. We know that in the problem, minimizing the denominator will give the solution. (As mentioned in @Kaa1el's comment b has to be part of denominator)

Consider,

Denominator = b/c/d/e... = b / ( cde... )


This arrangement gives the minimum value of Denominator, given that all numbers are > 1.

0

Share

Reply

Read More

SHOW 1 REPLY
- 

sean46


★ 106

April 25, 2017 4:22 AM

@vinod23 Looks great!

0

Share

Reply
- 

vinod23

★ 461

April 25, 2017 3:40 AM

@sean46 I have added the explanation. Is this enough? Please let me know. Thanks

0

Share

Reply