T.C. o(nlog(sum))
S.C. o(1)

Leetcode
Daily Challenge

31/03/2022

problem
Split Array Largest Sum

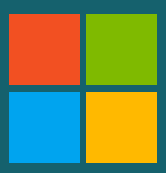
problem liked by
4606 people

difficulty
Hard

est. time
15-20 min

Let's build **Intuition**

can be asked in...


43%
Accuracy

Statement

Description

- Given an array nums which consists of non-negative integers and an integer m, you can split the array into m non-empty continuous subarrays.
- Write an algorithm to minimize the largest sum among these m subarrays.

i/p	o/p
nums = [7,2,5,10,8], m = 2	18

explanation

- The best way is to split it into [7,2,5] and [10,8],
- where the largest sum among the two subarrays is only 18.

Statement

- What are we asked ?
  - you are given nums[] & an integer m, split the nums[] in m parts.
  - There would be many ways with which you can split the nums[] in m parts.
  - Out of all the ways consider that way in which the max. sum of any split is smallest amongst other ways of split.

i/p	o/p
nums = [7,2,5,10,8], m = 2	18

- Above nums[] can be split in 2 parts in many ways

split 1	split 2	max sum
[7]	[2,5,10,8]	25 (split 2)
[7,2]	[5,10,8]	23 (split 2)
[7,2,5]	[10,8]	18 (split 2)
[7,2,5,8]	[8]	22 (split 1)

- out of all combinations 3rd combination has smallest max sum (18) & that's our answer

Approaches

- Approach #1
  - As we are considering all possible combinations & out of that we are concerned with one optimal one.
  - So what all ideas you got ...

All possible combinations Recursion

Optimal combination D.P. (dynamic programming)

- Approach #2
- Recursive solution was pretty intuitive but let's see other one
- Idea is, we would start with some 'low' limit & check if it is possible to split nums[] in 'm' split such that maxSum of any split in not more than 'low'
- Don't be confused, follow along & you will get the idea...

Intuition

i/p	o/p
nums = [7,2,5,10,8], m = 2	18

- We need to divide nums[] in 2 splits
- 10 is largest element so our smallest max. element can't be less than 10
- so let's say `low = 10`
- Now the largest possible sum for a split can be sum of nums
- so let's keep `high = sum(nums) = 92`

- Our answer will lie b/w low & high limits only

But what do these low to high limits tell ?

- Remember, we wanted to minimize the sum of largest split
- Now we start iterating from i = low to i <= high
- for every i we check if we `canSplit()` our nums[] in m parts such that no part has sum > i
- The first i that gives '`true`' for canSplit(), that's our answer (as we want the smallest sum with which we were able to split nums in m part)

Intuition

i/p

nums = [7,2,5,10,8],
m = 2

o/p

18

low = 10
high = 92

- let's start with i = 10 to i <= 92

i = 10

can we split nums[] in m parts such that no part has sum > 10 ?

No

[7,2] , [5,10,8]

we kept 1st split sum(9) <= 10 but 2nd split sum(23) > 10
so if you pass nums[], i, m in canSplit() i.e.

canSplit(nums, i, m)

it returns false

if we keep traversing linearly from i = 10,11,12,13,14 but we won't be able to split nums, as our canSplit() returns 'false'

i = 15

can we split nums[] in m parts such that no part has sum > 15 ?

[7,2,5] , [10,8]

we kept 1st part sum(14) <= 15 but 2nd split sum(18) > 15

so canSplit() returns false again

Intuition

i/p

nums = [7,2,5,10,8],
m = 2

o/p

18

you can iterate till 18 & for every i canSplit() returns false

i = 18

can we split nums[] in m parts such that no part has sum > 18 ?

[7,2,5] , [10,8]

yepp, we are able to split in 2 parts & sum of both parts sum(7+2+5)=14 or sum(10,8)=18 is <= 18

so canSplit() return 'true' this time

so 18 is our ans (remember we wanted to minimize the largest sum a split can have)

with i = 18 we are able to split nums such that the max sum is <= 18, so obviously for i > 18 (19,20,100...) also we would be able to split the nums[] such that the sum of any split is not > i

so from 18 onwards our canSplit() will return true

Intuition

i/p

nums = [7,2,5,10,8],
m = 2

low = 10
high = 92

o/p

18

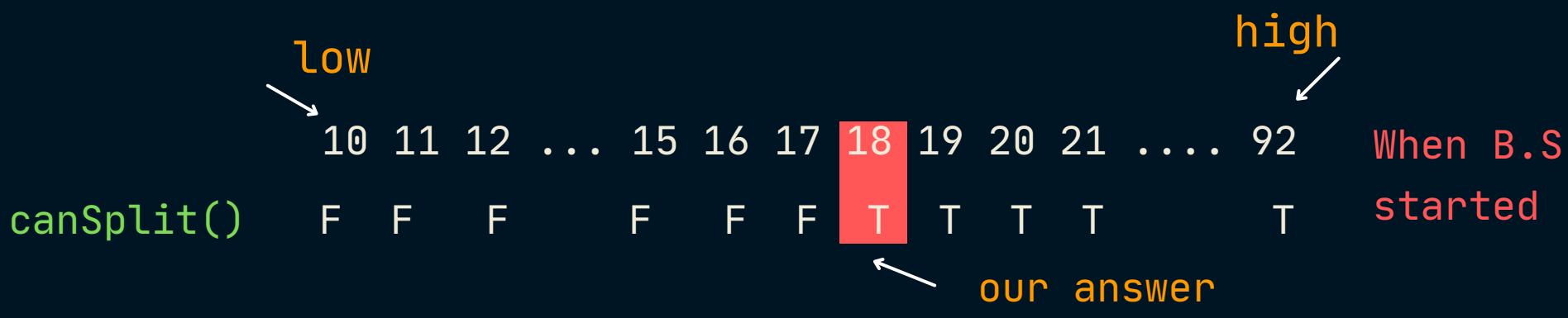
can you observe what we are doing, linear search on a sorted space so what's better than

Binary Search

Solving a Binary Search problem requires 2 steps(as I told in my CrashCamp SlideDeck, link in comment)

1) Divide the search space in 2 parts
2) After 'Binary Search' ends check whether low or high , who gives your answer.

Intuition

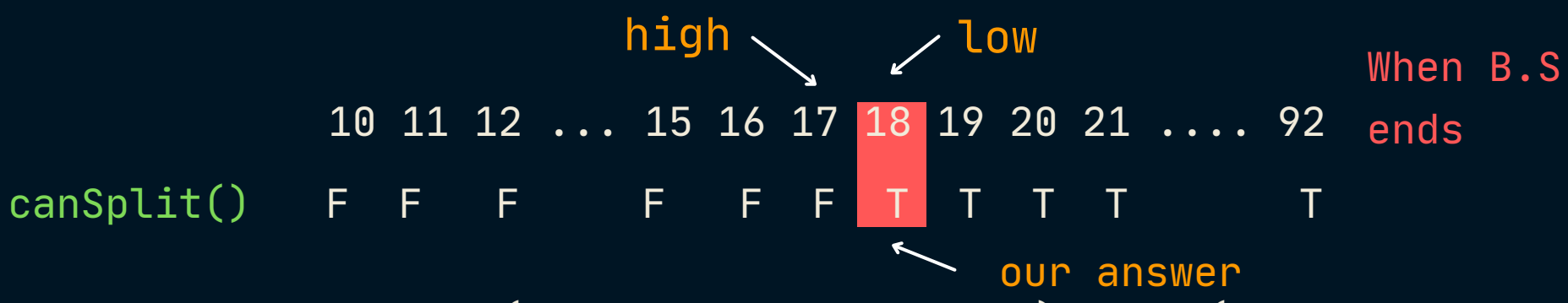


1) Divide the search space in 2 parts

1 We already divided the space in 2 parts, part 1 where canSplit() return 'false' & part 2, where canSplit() returns 'true'

2) After 'Binary Search' ends check whether low or high , who gives your answer.

- See in above no. line our answer is 1st element of 2nd space(true or favorable space)
- When B.S. started low point to 1st ele. of 1st space & high points to last ele of 2nd space

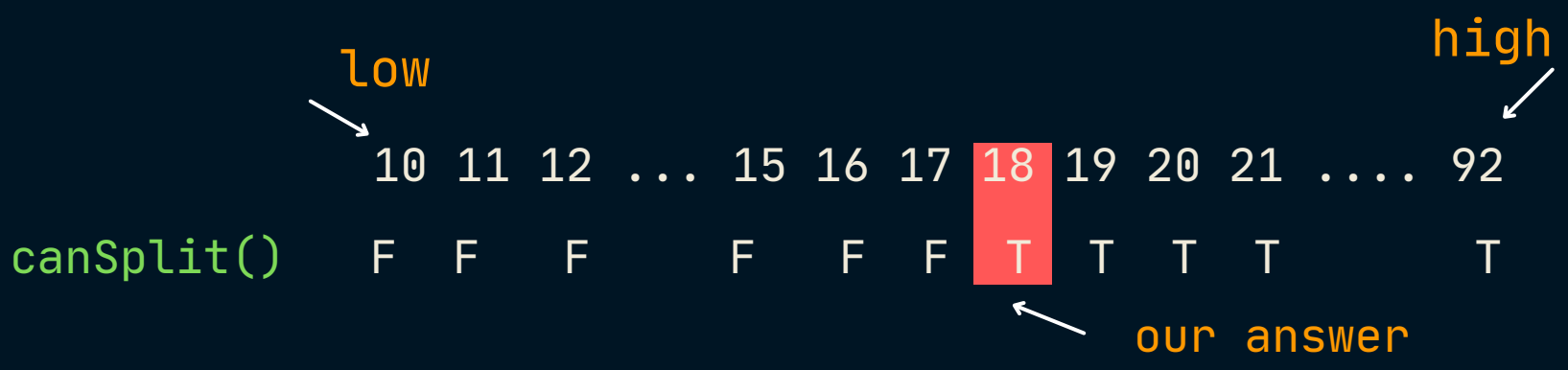


2
- When B.S ends (while loop terminates) i.e. (low becomes > high), who points to 1st ele. of 2nd space...

low

- So finally low gives your answer

Algorithm



3

when your 'mid' is at 'F' as you want to 1st 'T' move low to right of mid i.e `low = mid + 1`

when your 'mid' is at 'T' as you want to 1st 'T' move high to left of mid i.e `high = mid - 1`

And who tells whether you are at 'F' or 'T' ? `canSplit()`

let's conclude points 1,2,3 from prev. & current slides...

```
low = maxEle(nums), high = sum(nums)
while(low <= high)
  mid = (low+high)/2
  if(canSplit(nums, mid, m))
    high = mid - 1
  else
    low = mid + 1
return low
```

```cpp
class Solution {
public:

    bool canSplit(vector<int>& nums, int maxSum, int m) {

        int totalPart = 0;
        int currSum = 0;

        for(int i = 0;i < nums.size();i++) {
            if(currSum + nums[i] <= maxSum) {
                currSum += nums[i];
            } else{
                currSum = nums[i];
                totalPart++;
            }
        }

        return (totalPart + 1) <= m;
    }


    int splitArray(vector<int>& nums, int m) {

        int low = 0,high = 0;
        int sum = 0;
        for(int i = 0;i < nums.size();i++) {
            high += nums[i];
            low = max(low, nums[i]);
        }

        while(low <= high) {

            int mid = low + (high - low) / 2;

            if(canSplit(nums, mid, m)) {
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }

        return low;
```

My thoughts

- I started making these Decks when people comlained about being consistent
- When WFH ended & most of guys returned to office or college they complained about being consistent, so on March 1 I made a post about being consistent & making posts for entire month
- I think the purpose is fulfilled now, let me know in comments if you guys enjoyed this journey & new way of learning DSA
- Will be taking few days break from these Daily Leetcode posts

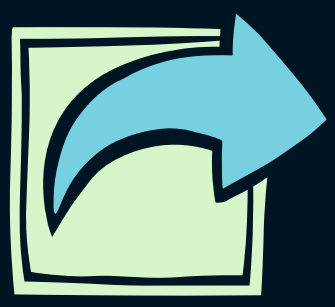
Yours trully

StoryTeller

BTW you can call me now, 'Deck Wale Bhaiyaa'

Leave a Like

Comment if you love posts like this, will motivate me to make posts like these



Share, with friends