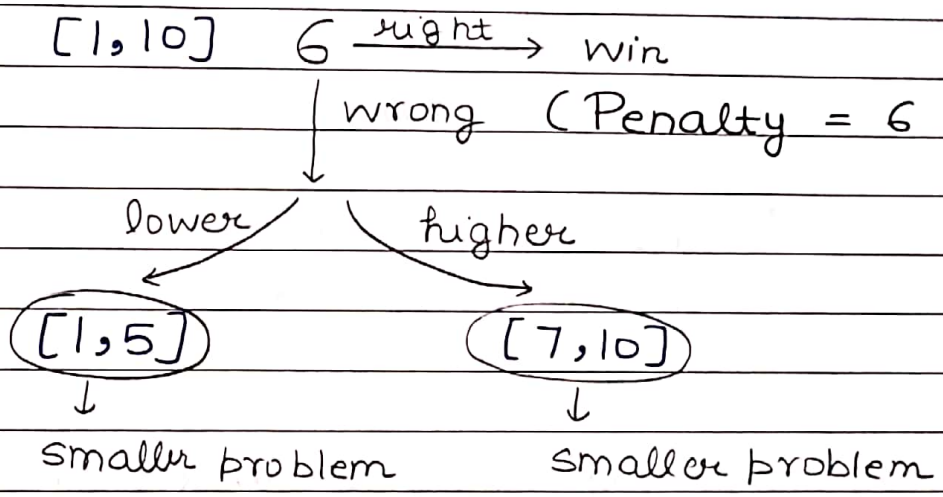
11/06/2023

Q1 Guess number higher or lower II (Leetcode 375)

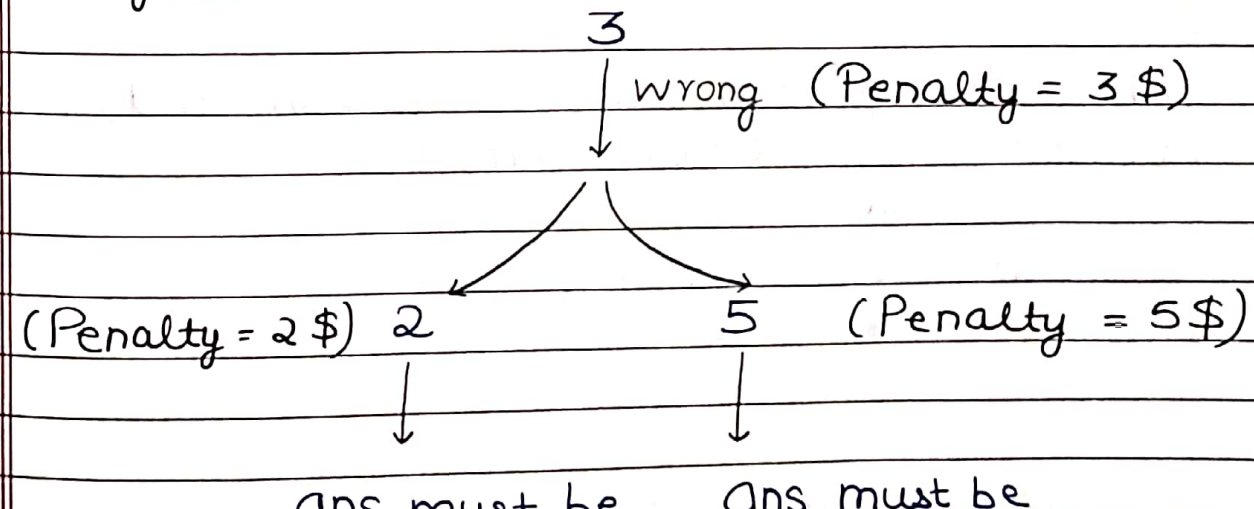
i/p → [1, 10]

Suppose that we guess 6.




Dry run

Range → [1, 5]




3 + 2 = 5 $

3 + 5 = 8 $ (Selected as safe in both sides)

1, 2, 3, 4, 5
↓ ↓ ↓ ↓ ↓
a1 a2 a3 a4 a5 } We have to return minimum of all these answers.

Code

```
//Recursive code
int solveRec(int start, int end){
    // Base case → Invalid range
    if(start >= end)
        return 0;
    int ans = INT_MAX;
    for(int i = start; i <= end; i++){
    // Considering the current penalty & then
    trying to fetch answer from left & right
    ans = min(ans, i + max(solveRec(start, i-1),
    solveRec(i+1, end)));
    }
    return ans;
}
// Top down approach
int solveTopDown(int start, int end,
vector<vector<int>> &dp){
    // Base case
    if(start >= end){
        return 0;
    }
    //Step 3: Check if answer already exists
    if(dp[start][end] != -1)
        return dp[start][end];
    int ans = INT_MAX;
    for(int i = start; i <= end; i++){
```

```
    ans = min (ans, i + max (solveTopDown (start,
      i-1, dp), solveTopDown (i+1, end, dp)));
}

//Step 2 : Store answer in dp array.
dp [start][end] = ans;
return dp[start][end];
}
```

Note → dp array has been created in main () as vector <vector <int>> dp (n+1, vector<int>(n+1), -1));

```
//Bottom up approach
int solveTab (int n) {
    // Step 1 : Create dp array
    vector <vector <int>> dp (n+2, vector <int> (n+2, 0));
    // Step 2 : Observe base case of top-down
    // Already done in initialization
    // Step 3 : Reverse flow of top-down
    for (int start = n ; start >= 1 ; start --) {
        for (int end = 1 ; end <= n ; end++) {
            // Base case handled
            if (start >= end)
                continue;
            else {
                int ans = INT_MAX;
                for (int i=start ; i<=end ; i++){
                    ans = min (ans, i + max (dp [start]
                    [i-1] , dp[i+1][end]));
                }
                dp [start][end] = ans;
            }
        }
    }
```

//As we have made call from 1 to n in top-do
return dp[1][n];
}

* Why space optimization is not possible?
dp[start][end] depends on previous column and next row and also row can be from start to end & hence not possible.

Q2 Minimum cost tree from leaf values.

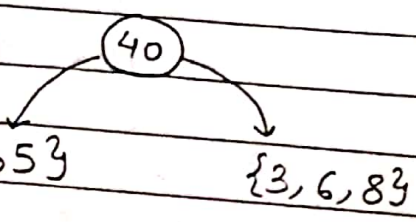
i/p → [6,2,4] {inorder traversal}
leaf node values

Binary tree conditions
→ 0 or 2 children
→ Non-leaf node is made via product of max leaf node in left subtree and max leaf node in right subtree.
→ We have to make / return the minimum sum of non-leaf nodes

Dry run

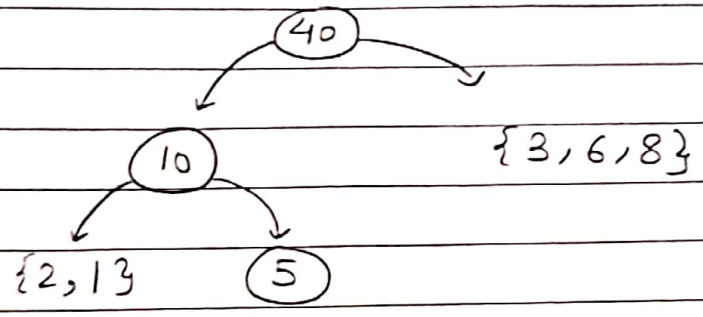
i/p → {2,1,5,3,6,8}
max from left part = 5
max from right part = 8

5X8 = 40



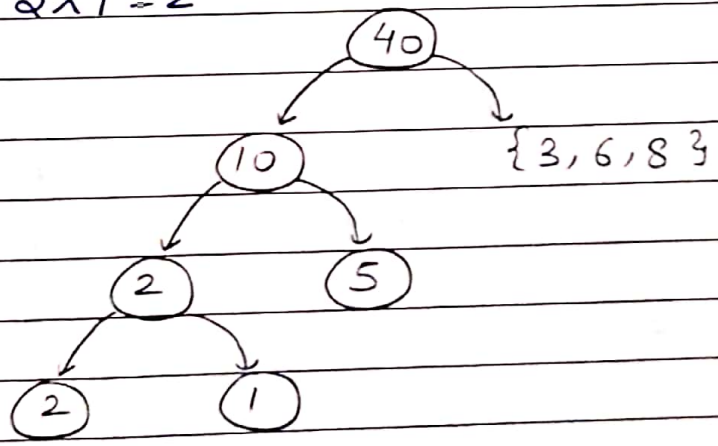

{2,1,5}

max from left part = 2
max from right part = 5
2 × 5 = 10

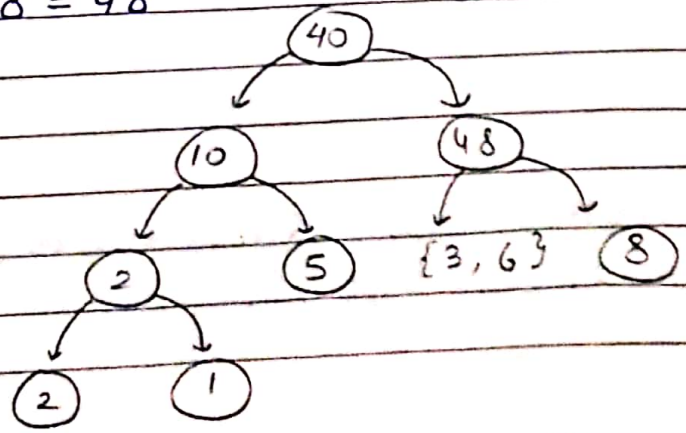



{2,1}

max from left part = 2
max from right part = 1
2 × 1 = 2

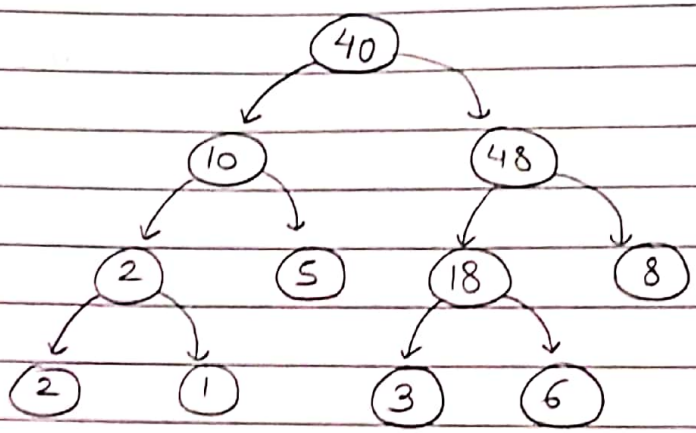



{3,6,8}
max from left part = 6
max from right part = 8
6 × 8 = 48



{3, 6}

3 X 6 = 18




The above tree is formed by partitioning from middle.

{2, 1, 5, 3, 6, 8} ⇒ T1

{2, 1, 5, 3, 6, 8} ⇒ T2

{2, 1, 5, 3, 6, 8} ⇒ T3

{2, 1, 5, 3, 6, 8} ⇒ T4

{2, 1, 5, 3, 6, 8} ⇒ T5

} Return minimum sum

Note → For finding maximum value, we need to create a map having range as key and max as value.

Code

```
int solveRec (vector <int> & arr, map <pair <int, int>, int> & maxi, int left, int right) {
```

```
// Base case
if (left == right)
        return 0; // leaf node & we don't have to
                  include in sum
int ans = INT_MAX;
for (int i = left; i < right; i++){
   ans = min (ans, maxi [{left, i}] *
   maxi [{i+1, right}] + solveRec (arr, maxi,
   left, i) + solveRec (arr, maxi, i+1, right)); }
}
return ans;
}

// Top-down approach
int solveTopDown (vector <int> &arr, map <pair
<int, int>, int> &maxi, int left, int right,
vector <vector <int>> &dp) {
   // Base case
   if (left == right)
        return 0;
   // Step 3: Check if answer already exists
   if (dp [left][right] != -1)
        return dp[left][right];
   int ans = INT_MAX;
   for (int i = left; i < right; i++){
      ans = min (ans, maxi [{left, i}] * maxi [{
      i+1, right}] + solveTopDown (arr, maxi, left,
      i, dp) + solveTopDown (arr, maxi, i+1, right, dp));
   }
   // Step 2: Store answer in dp array
   dp [right][left] = ans;
   return dp [right][left];
}
```

Note → vector<vector<int>> dp (n+1, vector<int>(n+1,-1
has been created in main().

```
// Bottom up approach
int solveTab(vector<int>& arr, map<pair<int,in
>,int>& maxi){
        int n = arr.size();
        //Step 1: Create dp array
        vector<vector<int>>dp(n+1, vector<int>
        (n+1,0));
        //Step 2: Observe base case in top down
        //Already initialized to 0
        //Step 3: Reverse flow of top-down
      for(int left=n-1; left>=0; left--){
        for(int right=0; right<=n-1; right++){
            //Handled base case
            if(left>=right)
                continue;
            else{
              int ans= INT_MAX;
              for(int i=left; i<right; i++){
               ans=min(ans, maxi[{left,i}]*
               maxi[{i+1,right}] + dp[left][i]+
               dp[i+1][right]);
            }
            dp[left][right]=ans;
        }
    }
}
// As call in top-down is from 0 to n-1.
return dp[0][n-1];
}
```

* How map was created ?
Range → max value in range
↑ key ↑ value

```
map <pair <int, int>, int > maxi;
for (int i=0; i<arr.size(); i++){
    maxi [{i, i}] = arr [i];
    for (int j = i+1; j<arr.size(); j++){
      maxi [{i, j}] =max (arr [j], maxi[{i, j-1}]
    }
}
```

Note: unordered-map was not used as pair can't be made as a key due to unavailability of hash function for pair.

* Space optimization is possible or not ?
Not possible as dp[left][right] depends on the same column and next row but row can be anything from 0 to n-1 & same for column.