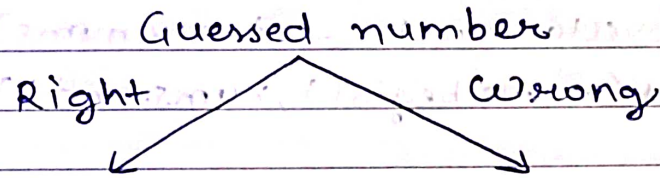
17/01/2024
Wednesday

* **Guess Number Higher or lower-II (Leetcode-375)**

I pick a no. between 1 and n.

You guess a number.



win

whether guessed no. is higher or lower and continue guessing. For every wrong guess 'x', you will pay x $.

If run out of money, you will lose game.
for a particular n, return minimum amount of money you need to guarantee win regardless of what no. i pick.

I/P - n = 5

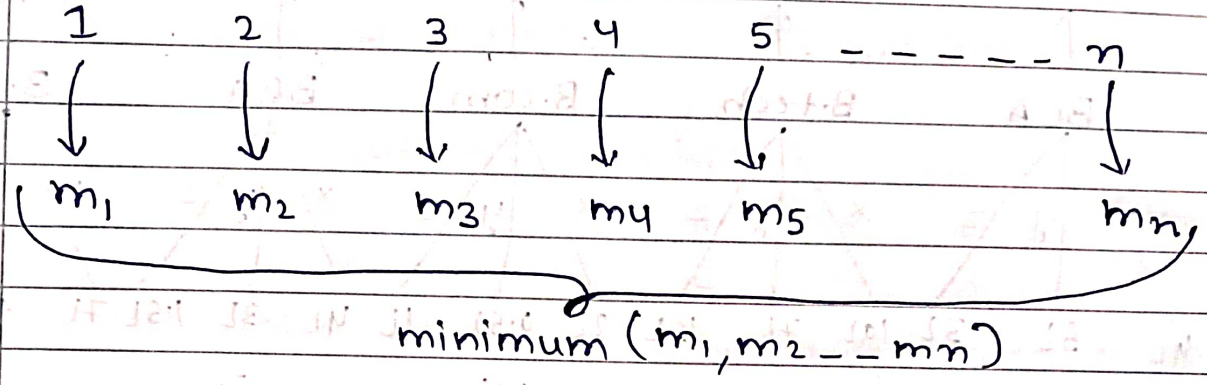
O/P - 6

Merge Interval Pattern -

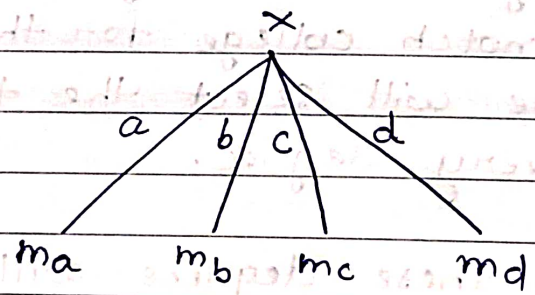
for range [i, J] ⇒

ans ⟶ [i, mid] and [mid+1, J]

Approach– For every number in range 1 to n, we will find the amount of money required corresponding to that number.

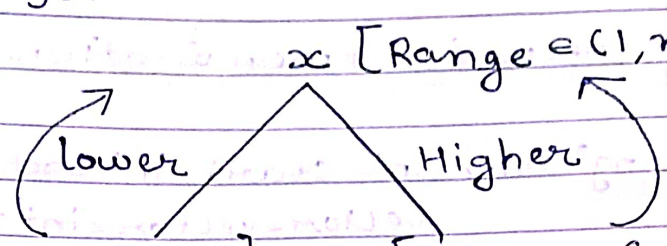


Let for number x in range 1 to n, we will have different paths that will lead to the coin:



m_a, m_b, m_c, m_d are the different money which i have paid for wrong guesses and let m_b be the largest amount of all. So, if i have m_b money in-hand, whatever path i will choose, i will not run out of money i.e. i will not lose the game.

Note - As hume worst-case scenerio leke chlna he ki humare pas kitna amount ho ki hum game lose na kre. So, hum only wrong guesses consider krenge.



For x, ans = penalty of x $ + max (lower, higher)
Choose the maximum between lower and higher.
For every number in range, we have to do the same and finally, we will choose the minimum ans among them.

- **Code using recursion -**

```
int solveUsingRec (int start, int end) {
        //base case
        if (start >= end) {
         //if out of range ho ya single element
            return 0;
        }

        int ans = INT_MAX;
        for (int i= start; i<=end; i++) {
         int lower= solveUsingRec (start, i-1);
         int higher= solveUsingRec (i+1, end);
         ans = min (ans, i+ max (lower, higher));
        }

        return ans;
}
```

- Code using recursion + memoization

```
int solveUsingMemo(int start, int end,
                   vector<vector<int>>& dp){

    //base case
    if(start >= end) return 0;

    //if ans 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, max(solveUsingMemo(start, i-1, dp),
                  solveUsingMemo(i+1, end, dp)));
    }

    return dp[start][end] = ans;
}

int getMoneyAmount(int n){

        int start = 1;
        int end = n;

    vector<vector<int>> dp(n+1, vector<int>(n+1, -1));
    return solveUsingMemo(start, end, dp);
}
```

- **Code using Tabulation –**

```
int solveUsingTabu (int n){

vector<vector<int>> dp (n+2, vector<int>(n+1, 0));

for (int s_index = n; s_index >= 1; s_index --) {
 for (int e_index = s_index + 1; e_index <= n; e_index++){

    int ans = INT_MAX;
    for (int i= s_index; i<= e_index; i++){
      ans = min (ans, i + max (dp[s_index][i-1],
                               dp[i+1][e_index]));
    }
    dp[s_index][e_index] = ans;
  }
}

return dp[1][n];
}
```

* **Minimum cost tree from leaf values**

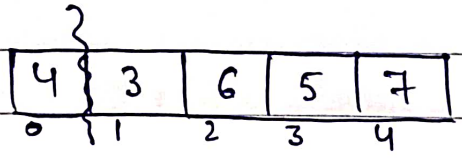
Given array arr, consider all binary trees such that -

- Each node has either 0 or 2 children.
- The value of arr correspond to the values of each leaf in an inorder traversal of tree.
- The value of each non-leaf node is equal to product of largest leaf values in its left subtree & right subtree

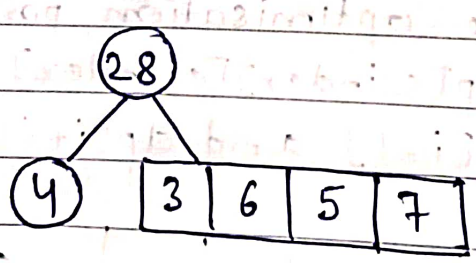
Among all possible binary trees, return smallest possible sum of values of each non-leaf node.

I/P - arr = [4, 3, 6, 5, 7]

O/P - 108

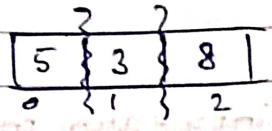


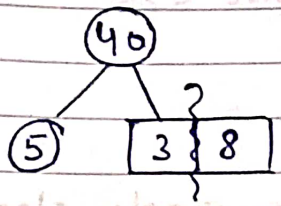
Range for left subtree = [0,0] → max = 4
Range for right subtree = [1,4] → max = 7
} x = 28

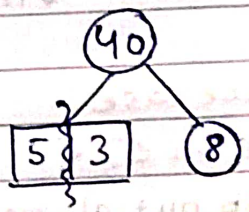


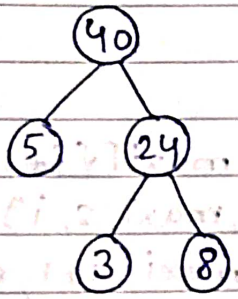
Same with the rest of array.

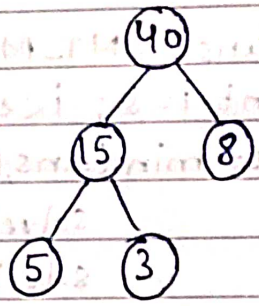
Let the array be [5, 3, 8]











Sum of non-leaf node = 40+24 = 64

Sum of non-leaf node = 40+15 = 55

Minimum of 64, 55 is 55.

Note - for a particular range in array, we will pre-compute the maximum for that range using map.
The first in map represents a pair where pair stores start & end of the range.
The second in map represents the maximum element corresponding to that range.

- Code using recursion -

```
class Solution {
public:

int solveUsingRec (vector<int> &arr, map<pair<int,int>,int>
                   &maxi, int s, int e){

    //base case
    if (s>=e) {
      //Jb out of range ho ya fir single element ho
      // to us case me non-leaf node hogi nhi, so return 0.
       return 0;
    }

    int ans=INT_MAX;
   for(int i=s; i<e ; i++) {
      ans= min (ans,(maxi[{s,i}]*maxi[{i+1,e}]) +
                 solveUsingRec (arr, maxi, s, i) +
                 solveUsingRec (arr, maxi, i+1, e));
    }
    return ans;
}
int mctFromLeafValues (vector<int> &arr) {
    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}]);
      }
    }

    return solveUsingRec (arr, maxi, 0, arr.size()-1);
  }
};
```

- Code using memoization -

```
class Solution{
public:

int solveUsingMemo( vector<int>& arr,
                    map<pair<int, int>, int>& maxi,
                    int s, int e, vector<vector<int>>& dp){

    if (s>=e) return 0;
    if (dp[s][e] != -1) return dp[s][e];

    int ans = INT_MAX;
    for(int i=s; i<e; i++){
        ans= min(ans, (maxi[{s,i}] * maxi[{i+1, e}]) +
            solveUsingMemo(arr, maxi, s, i, dp)+
            solveUsingMemo(arr, maxi, i+1, e, dp));
        };

    return dp[s][e]= ans;
}
int mctfromleafValues (vector<int>& arr){

    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}]);
        }
    }
    int n= arr.size();
    vector<vector<int>> dp(n+1, vector<int>(n+1, -1));
    return solveUsingMemo(arr, maxi, 0, n-1, dp);
}
};
```

- Code Using Tabulation-

```
class Solution{
public:

int solveUsingTabulation(vector<int>&arr,
                         map<pair<int,int>,int>& maxi){

    int n= arr.size();
    vector<vector<int>>dp(n+1,vector<int>(n+1,0));

    for(int s_index=n-2; s_index>=0; s_index--)
      for(int e_index=s_index+1; e_index<n; e_index++){

          int ans=INT_MAX;
          for(int i=s_index; i<e_index; i++){
              ans=min(ans,(maxi[{s_index,i}]*maxi[{i+1,e_index}]
                    +dp[s_index][i]+dp[i+1][e_index]);
          }
          dp[s_index][e_index]=ans;
      }
    }

    return dp[0][n-1];
}
int mctfromleafValues(vector<int>& arr){
    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}]);
      }
    }

    return solveUsingTabulation(arr,maxi);
}
```