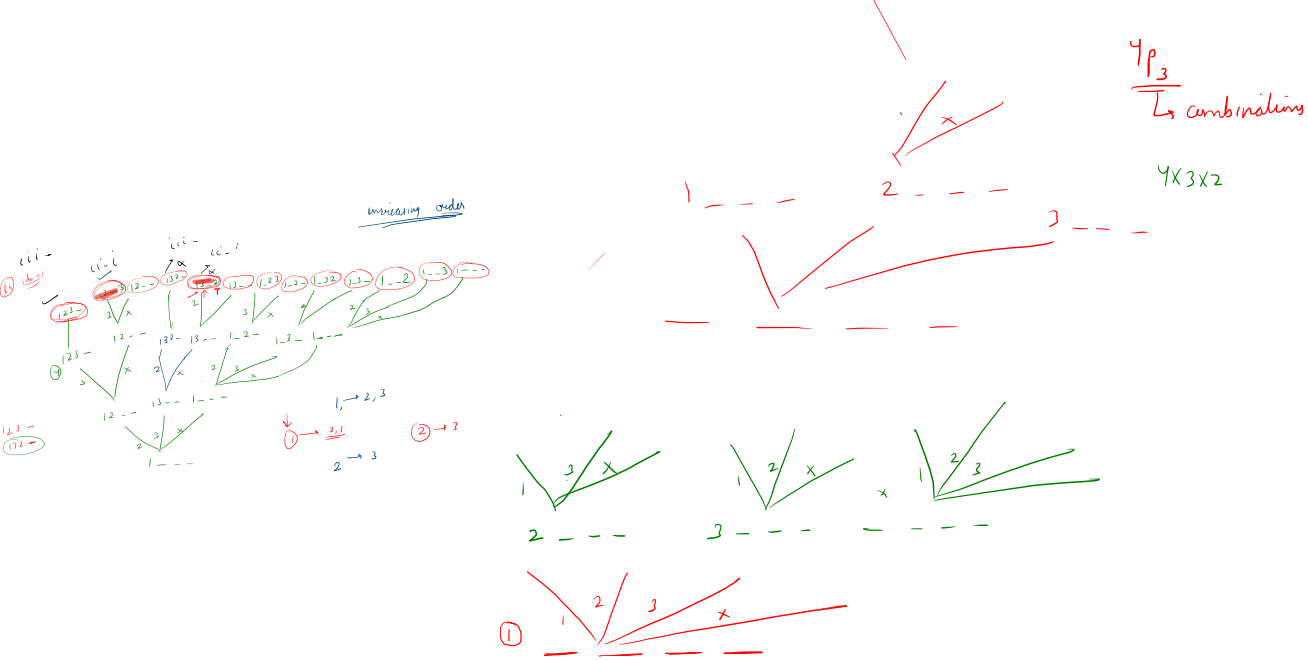


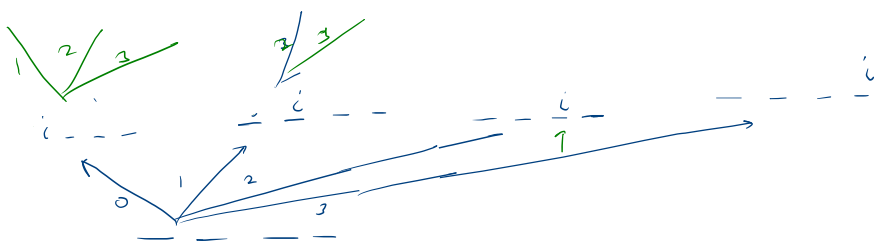

any doubts?

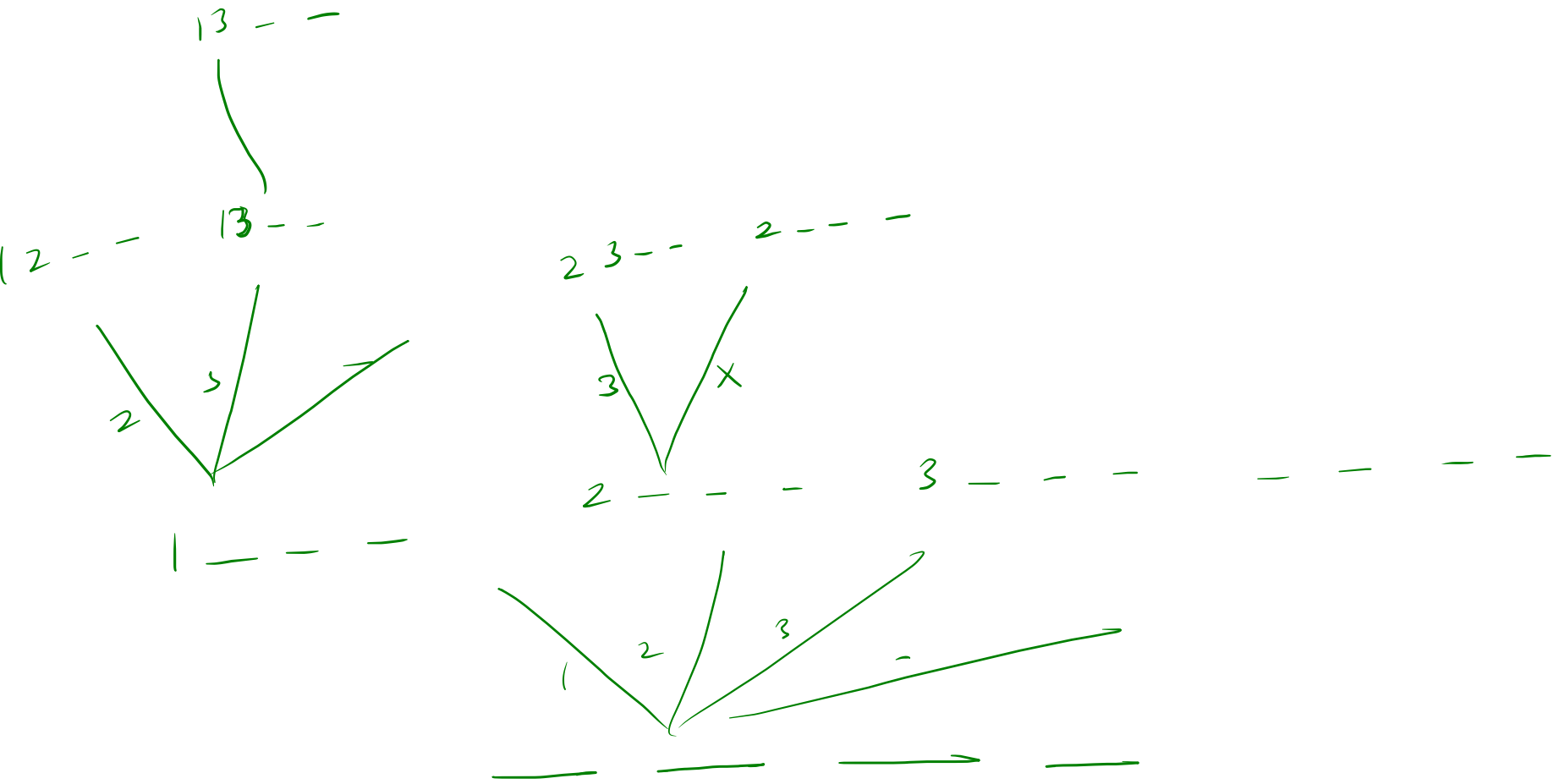
$P_{NC}$

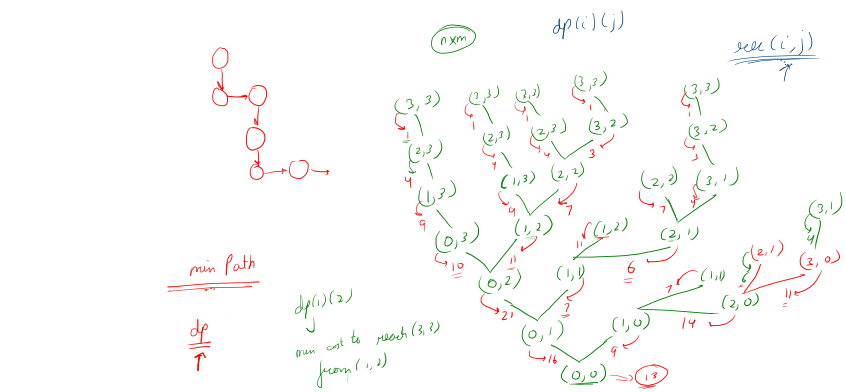
what
how
why

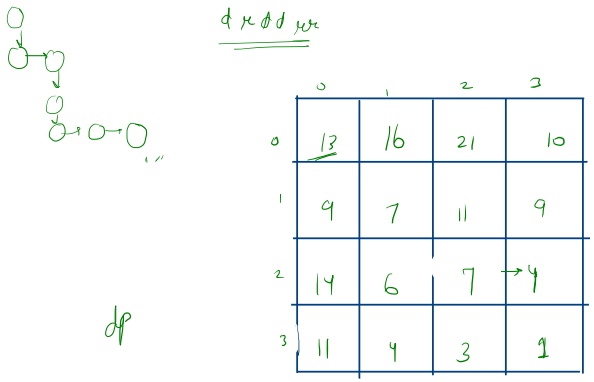
dry run
watch the
dry run

```java
public static void combinations(boolean[] boxes, int ci, int ti, int lb){
  if(ci > ti){
      for(int i=0; i<boxes.length; i++){
          if(boxes[i]==true){
              System.out.print(s: "i");
          } else {
              System.out.print(s: "-");
          }
      }
      System.out.println();
      return;
  }

  for(int i = lb +1; i<boxes.length; i++){
      if(boxes[i]==false){
          boxes[i]=true;
          combinations(boxes,ci+1,ti,i);
          boxes[i]=false;
      }
  }
}
```



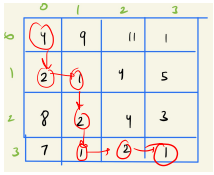

13 - -
12 - -
13 - -
2
3
1 - - -
23 - -
2 - - -
3
X
2 - - -
3 - - -
1
2
3
-

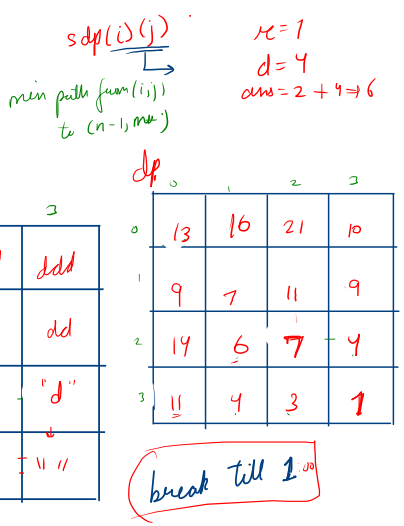


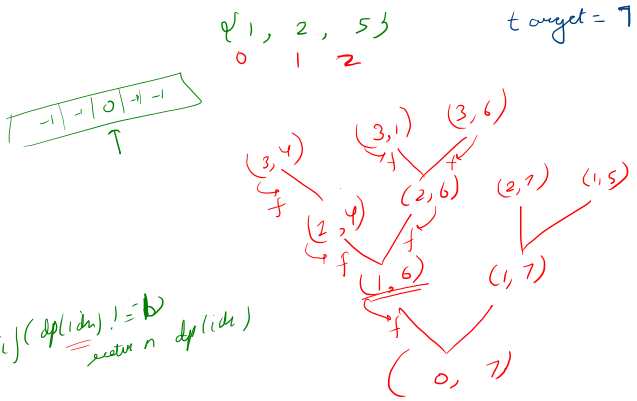




```java
public int minPath_tab(int[][] grid, int i, int j, int[][] dp){
    for(i=grid.length-1; i>=0; i--){
        for(j=grid[0].length-1; j>=0; j--){
            if(i==grid.length-1 && j==grid[0].length-1){
                dp[i][j] = grid[i][j];
                continue;
            }

            int right = Integer.MAX_VALUE;
            int down  = Integer.MAX_VALUE;

            if(j+1 < grid[0].length){
                right = dp[i][j+1];//minPath_memo(grid, i, j+1, dp);
            }

            if(i+1 < grid.length){
                down = dp[i+1][j];  //minPath_memo(grid, i+1, j, dp);
            }

            int ans = grid[i][j] + Math.min(right,down);

            dp[i][j] = ans;
        }
    }

    return dp[0][0];
}
```

```java
public int minPath_tabString(int[][] grid, int i, int j, int[][] dp){
    int n = grid.length;
    int m = grid[0].length;

    String[][] sdp = new String[n][m];

    for(i=grid.length-1; i>=0; i--){
        for(j=grid[0].length-1; j>=0; j--){
            if(i==grid.length-1 && j==grid[0].length-1){
                dp[i][j] = grid[i][j];
                sdp[i][j] = "";
                continue;
            }

            int right = Integer.MAX_VALUE;
            int down  = Integer.MAX_VALUE;

            if(j+1 < grid[0].length){
                right = dp[i][j+1];//minPath_memo(grid, i, j+1, dp);
            }

            if(i+1 < grid.length){
                down = dp[i+1][j];  //minPath_memo(grid, i+1, j, dp);
            }

            int ans = grid[i][j];

            if(right < down){
                ans += right;
                sdp[i][j] = "r" + sdp[i][j+1];
            } else {
                ans += down;
                sdp[i][j] = "d" + sdp[i+1][j];
            }

            dp[i][j] = ans;
        }
    }

    System.out.println(sdp[0][0]);
    return dp[0][0];
}
```







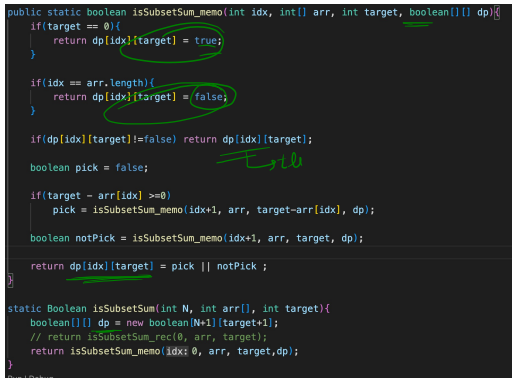



true, false

idx = {0, n}

target = {0, target}

memo



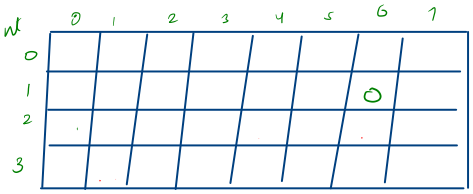
```java
public static boolean isSubsetSum_memo(int idx, int[] arr, int target, boolean[][] dp){
    if(target == 0){
        return dp[idx][target] = true;
    }

    if(idx == arr.length){
        return dp[idx][target] = false;
    }

    if(dp[idx][target]!=false) return dp[idx][target];

    boolean pick = false;

    if(target - arr[idx] >=0)
        pick = isSubsetSum_memo(idx+1, arr, target-arr[idx], dp);

    boolean notPick = isSubsetSum_memo(idx+1, arr, target, dp);

    return dp[idx][target] = pick || notPick ;
}

static Boolean isSubsetSum(int N, int arr[], int target){
    boolean[][] dp = new boolean[N+1][target+1];
    // return isSubsetSum_rec(0, arr, target);
    return isSubsetSum_memo(idx: 0, arr, target,dp);
}
```







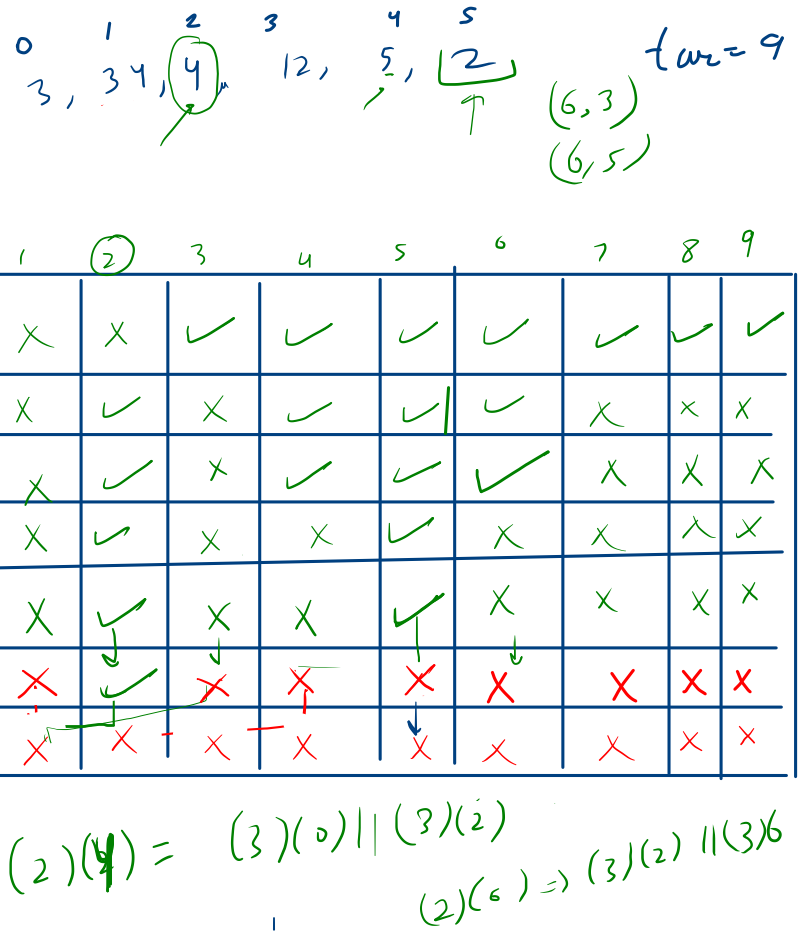

b, int



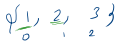

false = 0

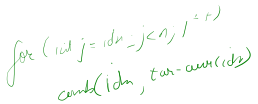
true = 1

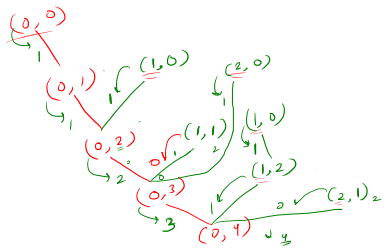
```java
public static int isSubsetSum_memo(int idx, int[] arr, int target, int[][] dp){
    if(target == 0){
        return dp[idx][target] = 1;
    }

    if(idx == arr.length){
        return dp[idx][target] = 0;
    }

    if(dp[idx][target]!=-1) return dp[idx][target];

    int pick = 0;

    if(target - arr[idx] >=0)
        pick = isSubsetSum_memo(idx+1, arr, target-arr[idx], dp);

    int notPick = isSubsetSum_memo(idx+1, arr, target, dp);

    return dp[idx][target] = pick | notPick ;
}

static Boolean isSubsetSum(int N, int arr[], int target){
    int[][] dp = new int[N+1][target+1];
    for(int[] d:dp){
        Arrays.fill(d,-1);
    }
    // return isSubsetSum_rec(0, arr, target);
    int ans =  isSubsetSum_memo(idx: 0, arr, target,dp);
    return ans == 1  ? true : false;
}
```



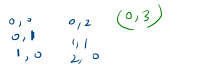

```java
public static boolean isSubsetSum_tab(int idx, int[] arr, int Target, boolean[][] dp){
    for(idx = arr.length; idx>=0; idx--){
        for(int target = 0; target<=Target; target++){
            if(target == 0){
                dp[idx][target] = true;
                continue;
            }

            if(idx == arr.length){
                dp[idx][target] = false;
                continue;
            }

            boolean pick = false;

            if(target - arr[idx] >=0)
                pick = dp[idx+1][target-arr[idx]]; //isSubsetSum_memo(idx+1, arr, target-arr[idx], dp);

            boolean notPick = dp[idx+1][target];  //isSubsetSum_memo(idx+1, arr, target, dp);

            dp[idx][target] = pick | notPick ;
        }
    }

    return dp[0][Target];
}
```

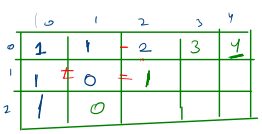


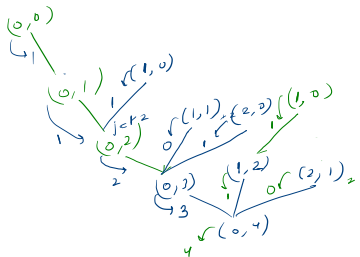




```java
public long combination_rec(int[] coins, int idx, int tar){
    if(tar==0){
        return 1;
    }

    long ans = 0;
    for(int j=idx; j<coins.length; j++){
        if(tar - coins[j]>=0){
            ans += combination_rec(coins, j, tar - coins[j]);
        }
    }

    return ans;
}
```







```java
public long combination_memo(int[] coins, int idx, int tar, long[][] dp){
    if(tar==0){
        return dp[idx][tar]=1;
    }

    if(dp[idx][tar]!=-1) return dp[idx][tar];

    long ans = 0;
    for(int j=idx; j<coins.length; j++){
        if(tar - coins[j]>=0){
            ans += combination_memo(coins, j, tar - coins[j],dp);
        }
    }

    return dp[idx][tar]=ans;
}
```

no two consecutive houses have same color

k colors

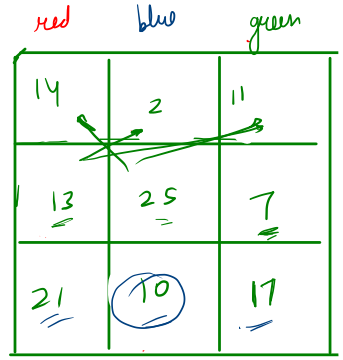
| | r | b | g |
|---|---|---|---|
| | 3 | | |
| 0th | 14 | 2 | 11 |
| 1st | 11 | 14 | 5 |
| 2nd | 14 | 3 | 10 |

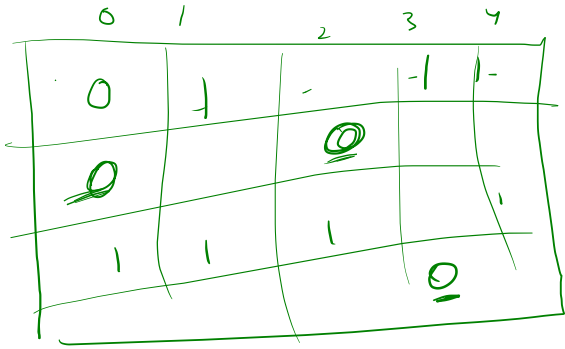
```
public int solve(int cost[][], int n) {
    int[][] dp = new int[n][3];

    for(int house=0; house<n; house++){
        if(house==0){
            dp[house][0]=cost[house][0];
            dp[house][1]=cost[house][1];
            dp[house][2]=cost[house][2];
        } else {
            dp[house][0] = cost[house][0] + Math.min(dp[house-1][1],dp[house-1][2]);
            dp[house][1] = cost[house][1] + Math.min(dp[house-1][0],dp[house-1][2]);
            dp[house][2] = cost[house][2] + Math.min(dp[house-1][0],dp[house-1][1]);
        }
    }

    int ans = Integer.MAX_VALUE;
    for(int color=0; color<3; color++){
        ans = Math.min(ans,dp[n-1][color]);
    }

    return ans;
}
```

color 1st house with red

11 + min (b, g) -

11 + 11(2, 11) → 13

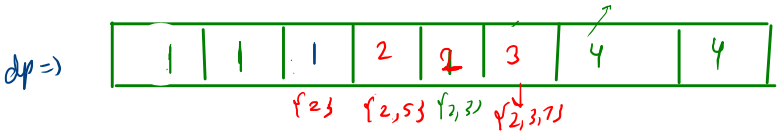


# LIS

↳ longest increasing subsequence.

0 1 2 3 4 5 6 7




dp(j) => length of increasing subs ending with arr(j)

dp(4) => 2

dp(5) = max(dp(5), dp(4)+1)

dp(5) = 3

max(dp(5), dp(2)+1)

↓ 3 ↓ 1+1

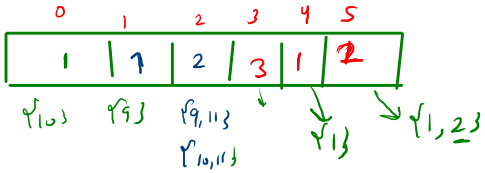
3 2

i = 3 4 5

j = 2 3

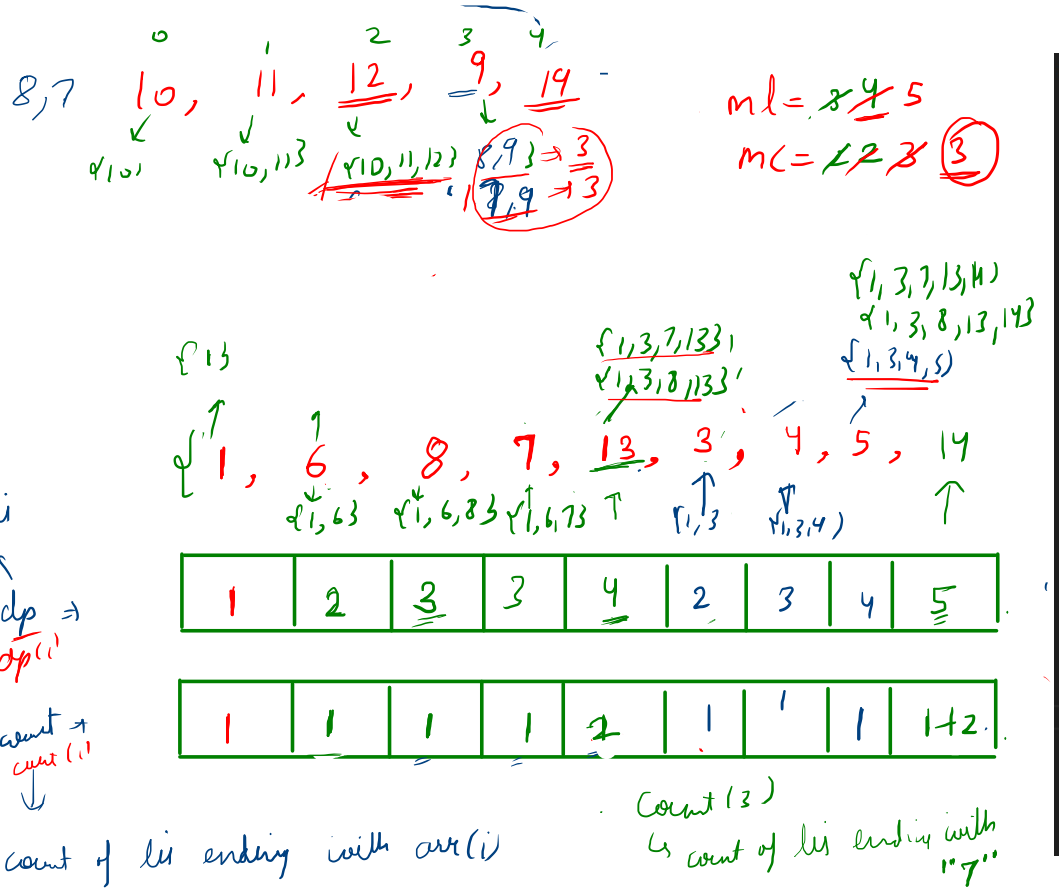
after jth point, (i) can be added

{10, 9, 11, 12, 1, 2}




dp(i) ↳ max sum of increasing sub ending with nums(i)

```java
public int lengthOfLIS(int[] nums) {
    int n = nums.length;
    int[] dp = new int[n];
    int ans = 0;
    for(int i=0; i<n; i++){
        dp[i]=1;

        for(int j=i-1; j>=0; j--){
            if(nums[i] > nums[j]){
                dp[i] = Math.max(dp[i],dp[j]+1);
            }
        }

        ans = Math.max(ans,dp[i]);
    }

    return ans;
}
```




```java
public int findNumberOfLIS(int[] nums) {
    int n= nums.length;

    int[] dp = new int[n];
    int[] count = new int[n];
    int maxLength = 0;
    int maxCount = 0;

    for(int i=0; i<n; i++){
        dp[i] = 1;
        count[i] = 1;

        for(int j=i-1; j>=0; j--){
            if(nums[i] > nums[j]){
                if(dp[i] < dp[j]+1){
                    dp[i] = dp[j]+1;
                    count[i] = count[j];
                } else if(dp[i] == dp[j]+1){
                    count[i] += count[j];
                }
            }
        }

        if(dp[i] > maxLength){
            maxLength = dp[i];
            maxCount = count[i];
        } else if(dp[i] == maxLength){
            maxCount += count[i];
        }
    }

    return maxCount;
}
```

max length

breach
tabl 10:20

new

max length

b=+6

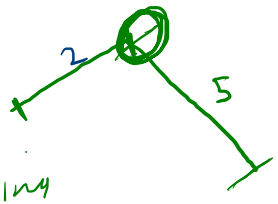
increasing, decreasing

{1,11} {11,10,4,2,1}

{1,2,4} {1,2,4,1} {1,2} {1}

[1,11,2,10,4,5,2,1]

{1,11} {1,2} {1,2,10}




lis ending with arr(i) ← lis =>

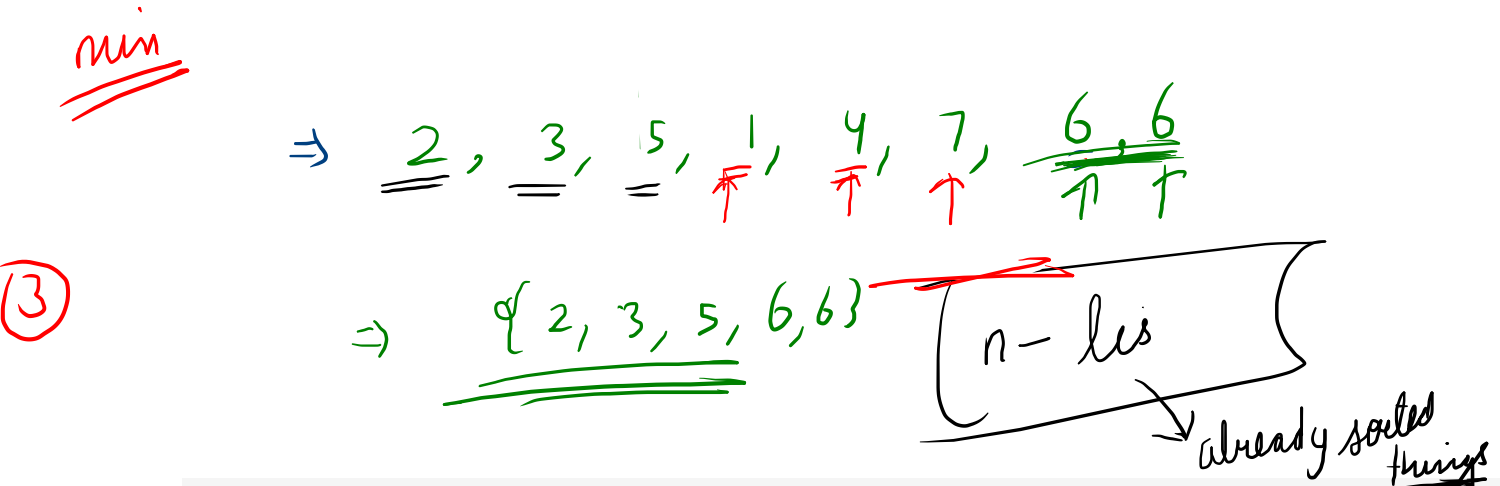
| 1 | 2 | 2 | 3 | 3 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|

lds starting with arr(i) ← lds

| 1 | 5 | 2 | 4 | 3 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|

{1} {11,10,4,2,1} (2,1) {10,4,2,1} {4,2,1} (5,2,1) {2,1} {1}

bitonic => lis ending with arr(i) + lds starting with arr(i) - 1




We can sort above array in 3 insertion steps as shown below,
1 before array value 2
4 before array value 5
6 before array value 7

[10,9,2,5,3,7,101,18]

$n^2$

nlogn

dp => {2,3,7,18} → sorted & increasing

lis

Binary Search

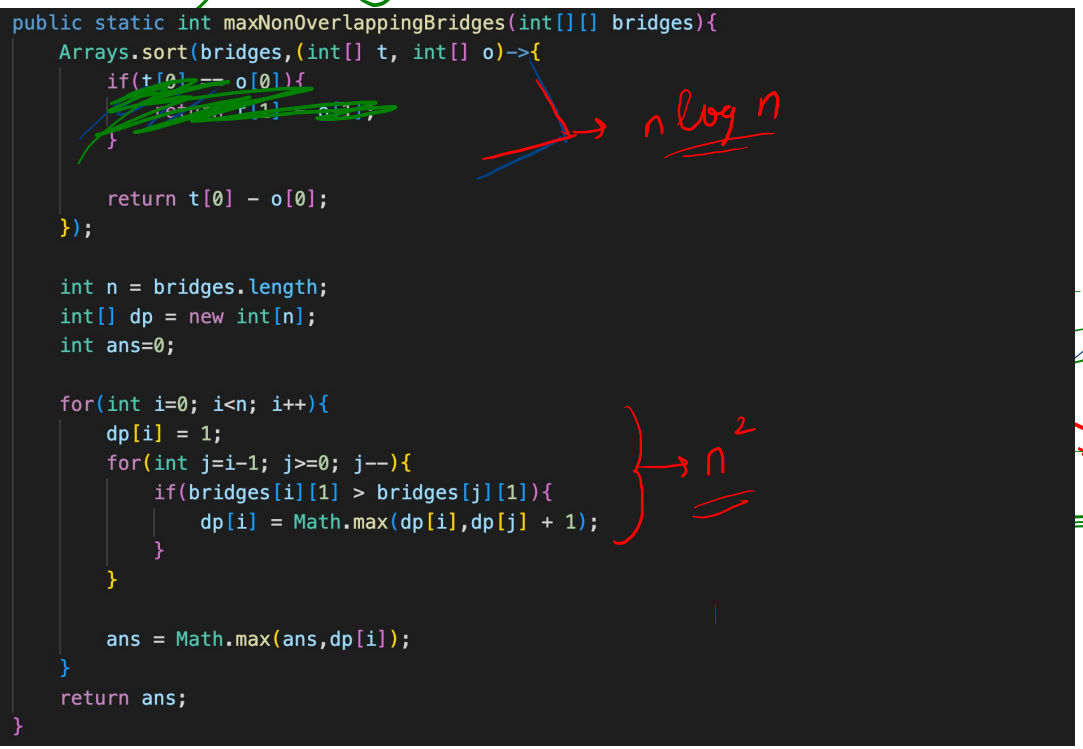
Count lis

```java
// LIS Better ==========
public int findPos(ArrayList<Integer> dp, int ele){
    int left = 0;
    int right = dp.size() - 1;

    while(left <= right){
        int mid = (left+right)/2;

        if(dp.get(mid) < ele){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return left;
}

public int lengthOfLIS(int[] nums) {
    int n = nums.length;

    ArrayList<Integer> dp = new ArrayList<>();

    for(int i=0; i<n; i++){
        int pos = findPos(dp,nums[i]);

        if(pos==dp.size()){
            dp.add(nums[i]);
        } else {
            dp.set(pos,nums[i]);
        }
    }

    return dp.size();
}
```

BS

logn

Run | Debug


public static int maxNonOverlappingBridges(int[][] bridges){
Arrays.sort(bridges,(int[] t, int[] o)->{
if(t[0] == o[0]){
}
return t[0] - o[0];
});
int n = bridges.length;
int[] dp = new int[n];
int ans=0;
for(int i=0; i<n; i++){
dp[i] = 1;
for(int j=i-1; j>=0; j--){
if(bridges[i][1] > bridges[j][1]){
dp[i] = Math.max(dp[i],dp[j] + 1);
}
}
ans = Math.max(ans,dp[i]);
}
return ans;
}
nlogn
n²


6 4 2 1
2 3 6 5
{{6,2},{4,3},{2,6},{1,5}}
{6,2}
{4,3}
{2,6}
{1,5}
{1,5}
{2,6}
{4,3}
{6,2}
0 1 2 3
1 2 1 1
6