1. LIS
2. Russian doll envelope
3. Palindromic substrings
4. Palindrome partitioning.

Variations of LIS on geeksforgeeks.

28: Given ar[N], find the length of longest strictly increasing subsequence. [LIS]

ex1: ar[5] :

0	1	2	3	4
9	2	4	3	10

[9] [9 10] [2] [2 4] [2 4 10] [2 3 10] [2 10]

[4 10] [3 10]

=> ans = 3

ex2: ar[6] :

0	1	2	3	4	5
2	1	6	3	7	9

=> ans = 4.

Idea1: Generate all subsequences, $O(2^N * N)$

Idea2: Pick or not pick.

0	1	2	3	4
9	2	4	3	10

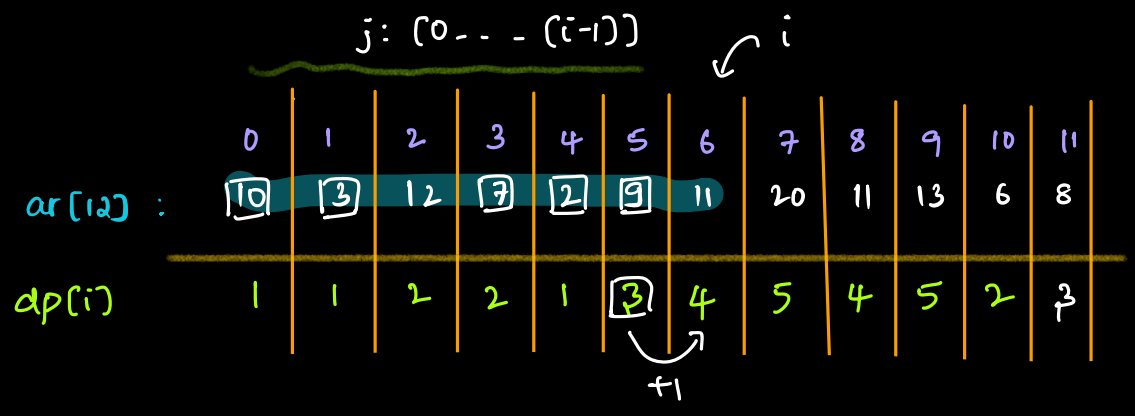
pick 10

not pick 10.

0	1	2	3
9	2	4	3

Idea3:

dp[i] : LIS ending at index 'i'.




```
int dp[N]

dp[0]=1, ans=1

for (i=1 ; i< N; i++)
    maxAns = 0
    for(j= 0; j < i ;j++)
        if (ar[j] < ar[i])
            maxAns = max(maxAns, dp[j])

    dp[i] = maxAns+1
    ans = max(ans, dp[i])

return ans
```

TC : $O(N^2)$

SC : $O(N)$

[Patience-sort algo. / Binary search] ⇒ TO-DO.

TC : $O(N \log N)$

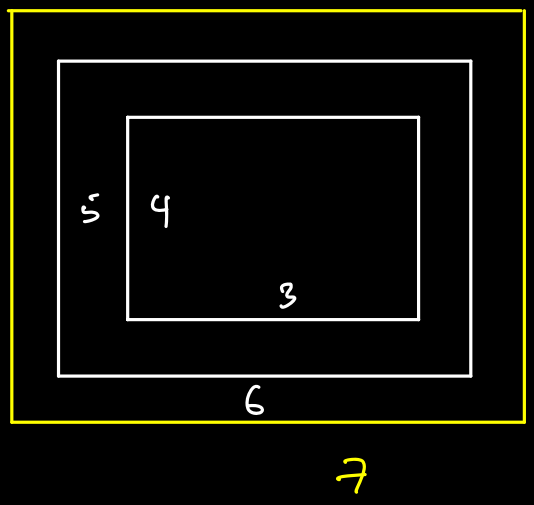
20. Russian doll envelope

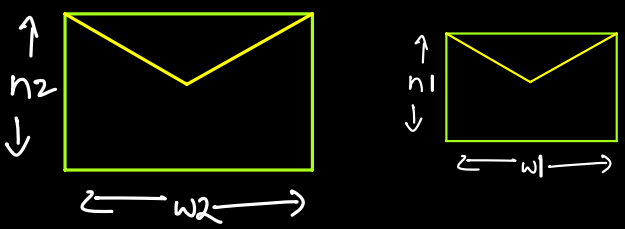
You've N-different envelops, find max count of envelops that can be put in a single envelope.

Note: Rotation of envelope is not allowed.

	h	w
A →	5	6
B →	6	4
C →	6	7
D →	4	3

ans: 3.





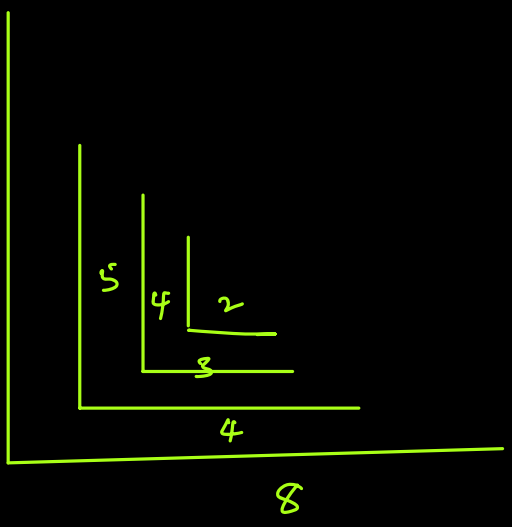
$h_1 < h_2$ && $w_1 < w_2$

---

h: [9 5 10 3 4 2]

w: [3 4 8 2 3 7]

ans = 4



Create a pair of envelops[]
↳ sort based on height
↳ read w[] from envelops
& Apply LIS on w[].

← SPOILER ALERT!

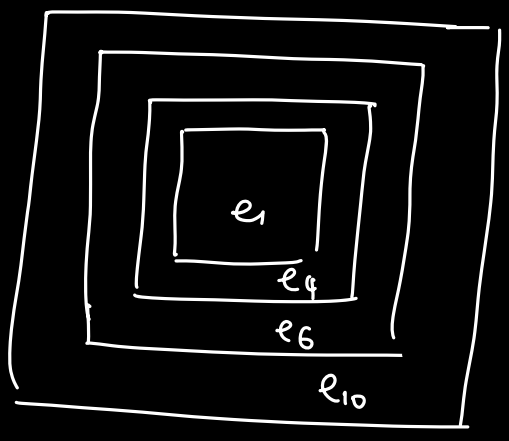
Let's say the envelopes can be represented as - .

$e_1 \quad e_2 \quad e_3 \quad e_4 \quad e_5 \quad --- \quad e_{10}$.

Let's find LIS.

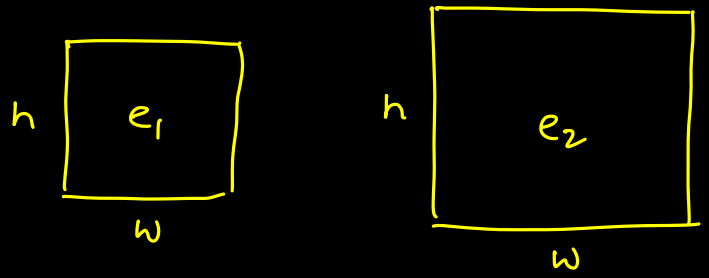
$\Rightarrow \quad e_1 \quad e_4 \quad e_6 \quad e_{10}$

$\Rightarrow \quad e_1 < e_4 < e_6 < e_{10}$



$\boxed{e_1 < e_2} \Rightarrow$ can e_1 fit inside e_2.

e_1: h, w e_2: h, w



$e_1.h < e_2.h$

&& $e_1.w < e_2.w$

h: [9 5 10 3 4 2] → Sort on height

w: [3 4 8 2 3 7]

h: [2 3 4 5 9 10]

w: [7 2 3 4 3 8] ⇒ LIS on w[].

Some variations

1. Minimum no. of ele's to be deleted from the array to make it sorted.

ex: [1 2 -5 6 -4 10 8 11]

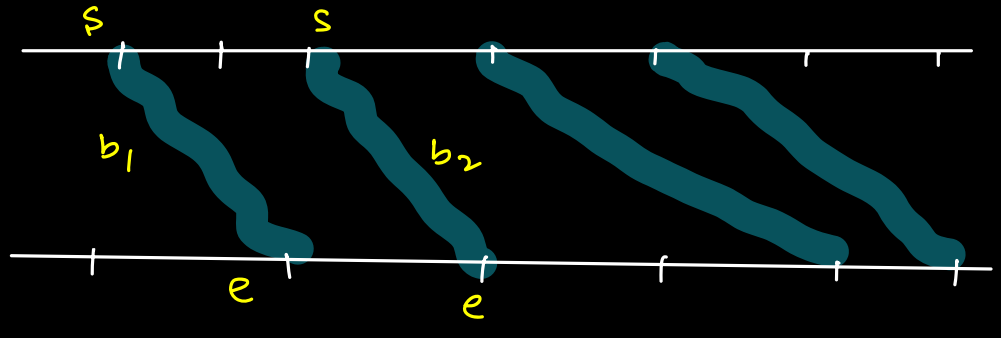
LIS: [1 2 6 10 11]

Ans: len(arr) - LIS.

2. Building the bridges.

s : [9 5 10 3 4 2]

e : [3 4 8 2 3 7]



[b_1 b_2 b_3 b_4 ___ b_{10}]

LIS ⇒ [b_1 b_3 b_6 b_{10}].

idea: Sort based on start/end & find the LIS of that array.

Q3: Check if every substring is a palindrome or not for a given string.

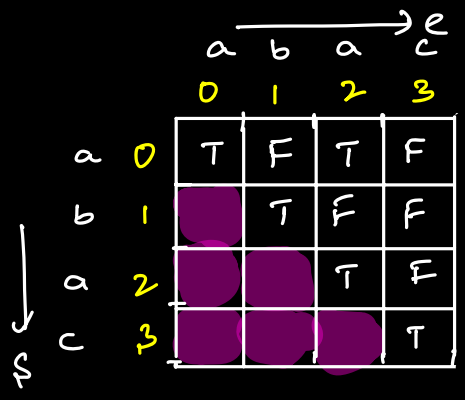
S = "a b a c"

a
a b
a b a
a b a c

b
b a
b a c

a
a c

c



Idea: For each substring check if its palindrome or not.

$O(N^3)$

Idea:

(i) s[i] != s[j] => dp[i][j] = false

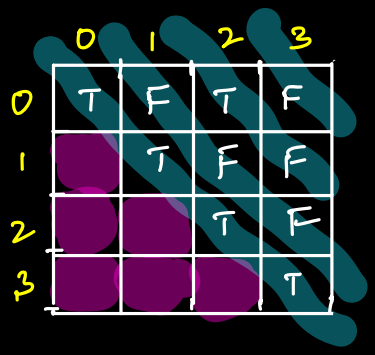
(ii) s[i] == s[j] => dp[i][j] = dp[i+1][j-1]

Start filling dp table from smallest subproblems.

Substring of length 1	Substring of length 2	Substring of length 3	Substring of length 4
(0,0)	(0,1)	(0,2) → dp[1][1]	(0,3)
(1,1)	(1,2)	(1,3) → dp[2][2]	(1,4) x
(2,2)	(2,3)	x (2,4) → dp[3][3]	(2,5) x
(3,3)	(3,4) x	x (3,5) x → dp[4][4]	(3,6) x
true	s[i]==s[j]		

Gap between i & j

=> 0, 1, 2 -- N-1




```
for( gap=0; gap<N; gap++)
```

For the diagonals.

i -> 0 to N-1

j -> gap to N-1

```
for (i=0, j=gap ; j<N; i++, j++)
```

gap = 0 => i=0, j=0, i=1, j=1, i=2, j=2, i=3, j=3.

gap=1 => i=0, j=1, i=1, j=2, i=2, j=3

gap=2 => i=0, j=2, i=1, j=3

gap=3 => i=0, j=3

code.

```
boolean dp[N][N]
for( gap=0; gap<N; gap++)
    for (i=0, j=gap ; j<N; i++, j++)
        // base cases.
        ① if (gap==0)
            dp[i][j]= true

        ② if (gap==1)
            if (s[i]==s[j])
                dp[i][j]=true
            else
                dp[i][j]=false
```

```
        else
            if(s[i] != s[j])
                dp[i][j] = false
            else
                dp[i][j] = dp[i+1][j-1]

return dp.
```

Variations:

1. count of all palindromic substrings : total 1's in dp[][]
2. length of longest palindromic substring :

   maxGap => max gap in dp[][] where dp[i][j] = true.

   ans: maxGap + 1

Q4. Find min no. of cuts required to partition the string into palindromes.

① x x | y

ans = 1

② x | a | b a a b | p

ans = 3

③ a b b a | c

ans = 1

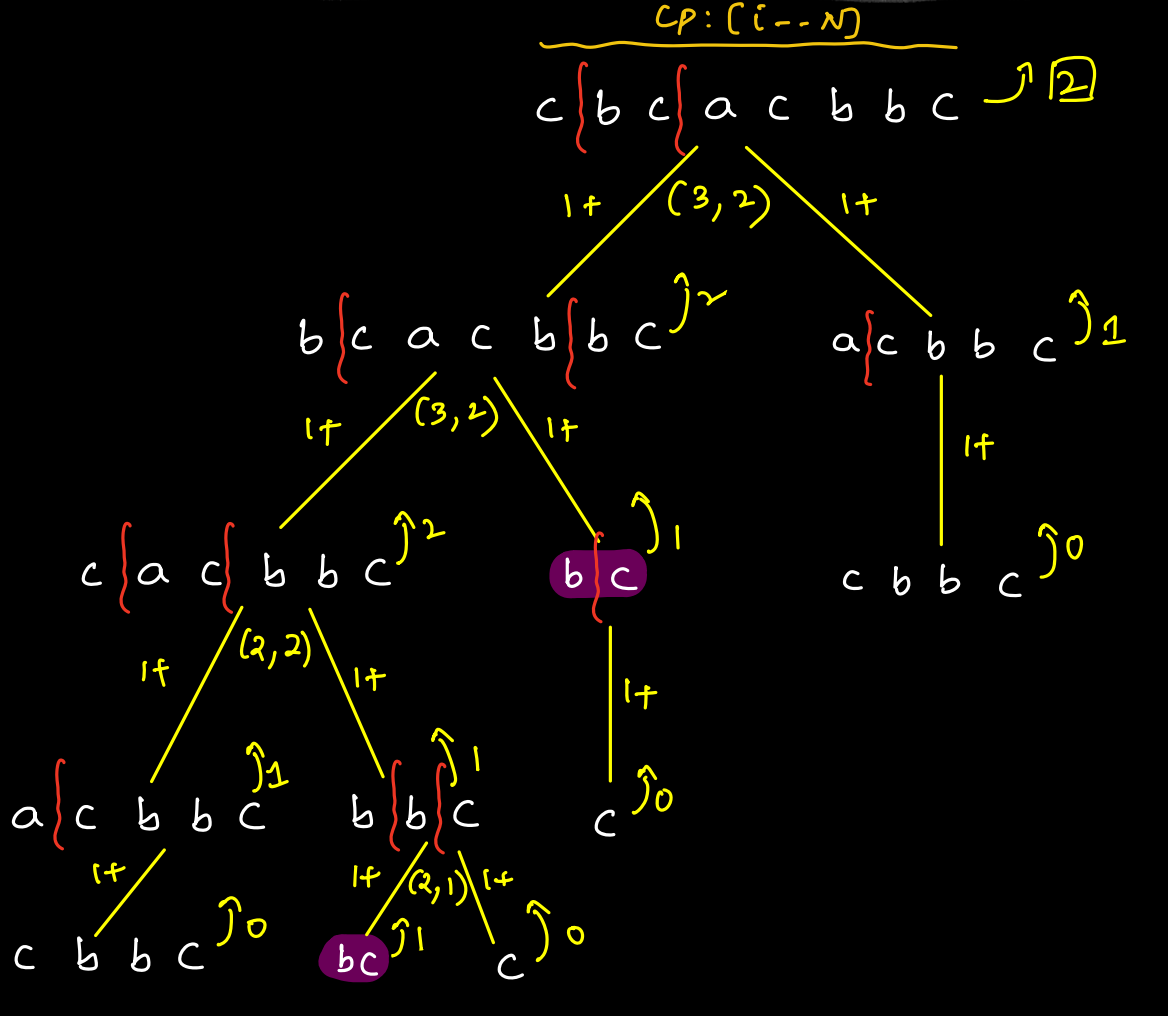
④ a | b b | c b c

ans = 2

Greedy: Select longest palindromic substring. ✗

c | b c a c b | b | c => ans = 3 [✗]

c b c | a | c b b c => ans = 2



```
int minCuts(string str, int i, int N)
    if(checkPalindrome(str, i, N))
        return 0
    cuts = ∞
    for(int cp = i ; cp < N; cp++)
        if(checkPalindrome(str, i, cp))
            cuts = min(minCuts(str, cp+1, N), cuts)
    return 1+cuts
```

DP: dp[i] : min no. of cuts req. to cut the substring (0-i) into palindromes.

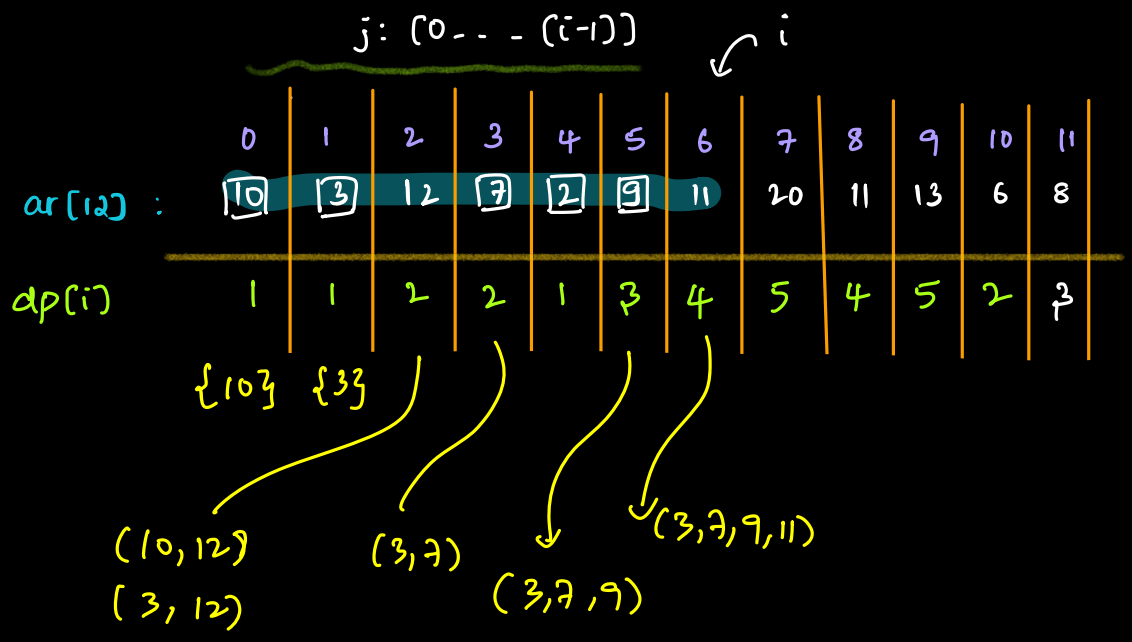
TC: (No. of states)(TC for each exp^n).

$O[(N) * (N)] \Rightarrow O(N^2)$.

↓ we can pre-populate isPalindrome ∀ (0--i's)

SC: No. of states

$O(N)$



multiple soln's. [comments]

Topic		hint 1	hint 2	soln
Arrays	Kadane's Algo			github link
	Trapping rain			