Good Evening Everyone!!

- L.I.S
- Russian Doll Envelopes
- Count of palindromic substrings.
- Palindromic partition

Longest Increasing Subsequence (L.I.S)

arr[] → [6, 9, 10, 13, 20] ans = 5

arr[] → [13, 6, 2, 1] ans = 1

[0, 2, 6, 9, 11, 15]

arr[] → [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]

ans = 6.

B.f idea. → Consider all the subsequences. T.C → $O(2^N * N)$

2nd Approach →

arr[] → [10, 3, 12, 7, 9, 11, 20, 11, 13, 6, 8] ans = 5

0 1 2 3 4 5 6 7 8 9 10

dp-

1	1	2	2	3	4	5	4	5	2	3
0	1	2	3	4	5	6	7	8	9	10

[10] [3] [10,12] [3,7] [3,7,9]
[3,12]
[3,7,9,11] [3,7,9,11,20] [3,7,9,11] [3,6] [3,6,8]
[3,7,9,11,13]

dp[i] - Longest increasing subsequence ending at index i.

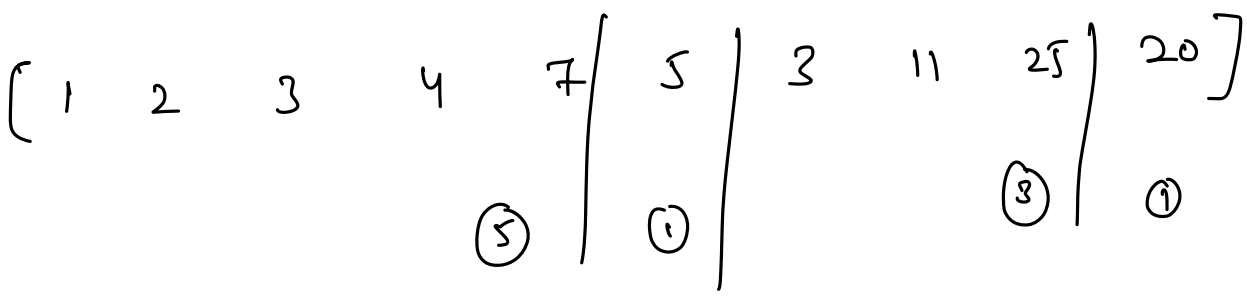
code →

```
int dp[N];
dp[0] = 1;    ans = 1
for( i = 1; i < N; i++){
        max = 0;
        for( j = 0; j < i; j++){
            if( arr[j] < arr[i]){
                max = Max( max, dp[j]);
            }
        }
        dp[i] = max + 1;
        ans = Max( ans, dp[i]);
}
return ans;
```

T.C → $O(N^2)$
S.C → $O(N)$

Longest Increasing Subarray

ans = 5.

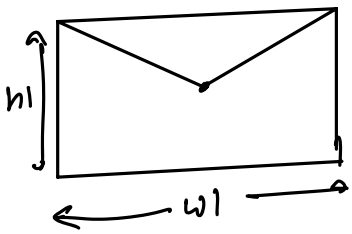


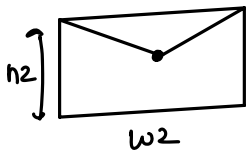
⑤ Russian Doll Envelopes.

N- different envelopes.

find max count of envelopes that can be put in a single envelope.

* Rotation of envelope is not allowed.

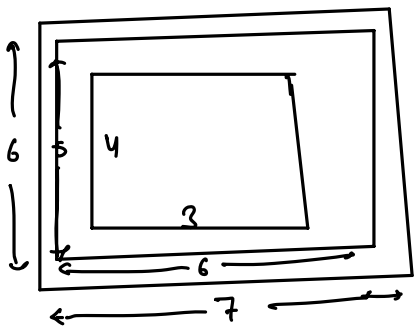




$[h2 < h1$
$w2 < w1]$

	h	w
A →	5	6
B →	6	4
C →	6	7
D →	4	3

ans = 3.



h → [9 ✓ 5 10 ✓ 3 ✓ 4 2]

w → [3 4 8 2 3 7]

Area → 27 20 80 6 12 14

ans = 4

Sorting on the basis of

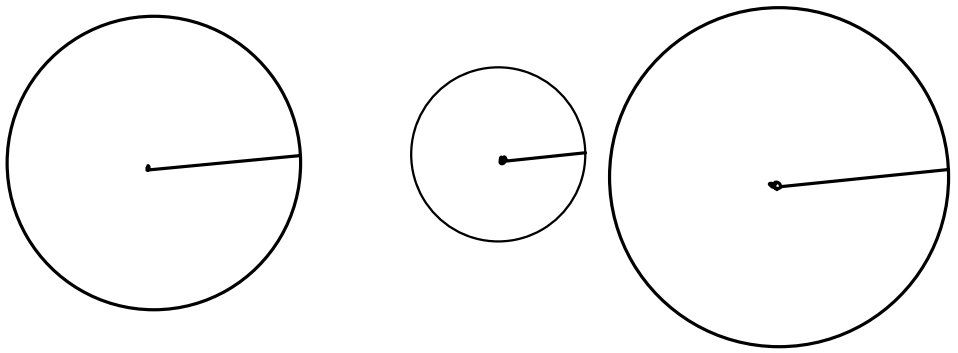
↙ ↓ ↘ ✗

Area Ht. Width

r → 5, 6, 7, 3, 9, 4, 9

⇓

3, 4, 5, 6, 7, 9, 9



ht[] → [1 | 2 | 3 | 4 | 4 | 5 | 7 | 10 | 10 | 12 | 15]

w[] → [10 | 3 | 7 | 9 | 11 | 20 | 11 | 6 | 13 | 8 | 2]

Longest increasing subsequence on width.

① Sort on the basis of ht.

Pair {
    ht;
    wd;
};

② int dp[N];

dp[0] = 1; ans = 1

```
for( i = 1; i < N; i++){
    max = 0;
    for( j = 0; j < i; j++){
        if( arr[j].width < arr[i].width && arr[j].ht < arr[i].ht){
            max = Max(max, dp[j]);
        }
    }
    dp[i] = max + 1;
    ans = Max( ans, dp[i]);
}
return ans;
```

T.C → O(N²)

S.C → O(N)

③ Given a string. For every substring, check if that is a palindrome or not.

$1 \le N \le 10^3$

S = "a b a c"
0 1 2 3

a ab aba abac

b ba bac

a ac

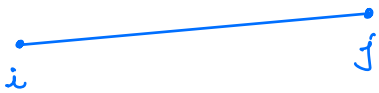
c

Expected O/P

si \ ei	0	1	2	3
0	t	f	t	f
1	x	t	f	f
2	x	x	t	f
3	x	x	x	t

Brute force Consider all the substrings & check for every substring if it is a palindrome or not. T.C → $O(N^3)$

observation



{ str[i] == str[j], copy the ans. present at dp[i+1][j-1]
str[i] != str[j], dp[i][j] = false }

gap=0	gap=1	gap=2	gap=3
0,0	0,1	0,2	0,3
1,1	1,2	1,3	
2,2	2,3		
3,3			

#code→

```
boolean dp[N][N];
for(gap = 0; gap < N; gap++){
    for(i=0, j=gap; j<N; i++,j++){
        if(gap==0){ dp[i][j] = true }
        else if(gap==1){ dp[i][j] = (str[i] == str[j]) }
        else{
            if(str[i] == str[j]){ dp[i][j] = dp[i+1][j-1] }
            else { dp[i][j] = false }
        }
    }
}
return dp[N][N];
```

[T.C → $O(N^2)$
S.C → $O(1)$]

→ Count all palindromic substrings ⇒ No. of true's in the dp[][]

→ Longest palindromic substring ⇒ max gap where true is present.

ans → gap+1.

④ Find min no. of cuts to partition the string such that all the partitions are palindrome.

Eg → x x | y ans = 1.

x | a | b a a b | p ans = 3

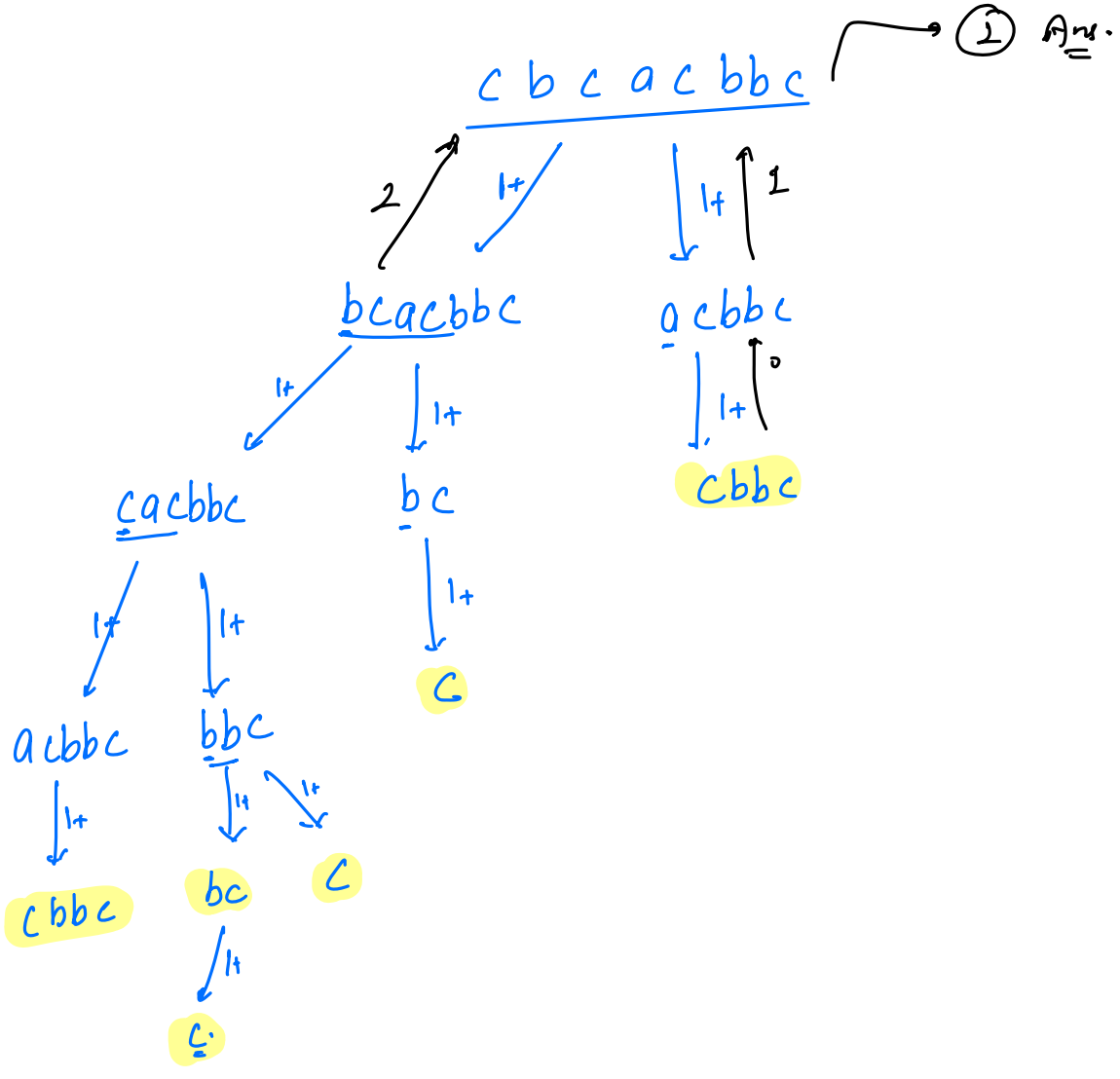
a b b a | c ans = 1

a | b b | c b c ans = 2

Greedy → select the longest palindromic substring first. ✗

c | b c a c b | b | c

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



#code:-

0 N-1

```
int minCuts (str, i, j, int dp[N]){
    if( checkPalindrome(str, i, j) == true){
        return 0;
    }
    if( dp[i] != -1) { return dp[i]; }
    min → ∞
    for( cp = i; cp < j ; cp++) {
        if( checkPalindrome(str, i, cp) == true){
            min = Min( min , minCuts(str, cp+1, j, dp) );
        }
    }
    dp[i] = min+1;
    return dp[i];
}
```

T.C → $O(N^2 + N^2)$

S.C → $O(N^2 + N)$

⇒ [#bottom-up]

x x