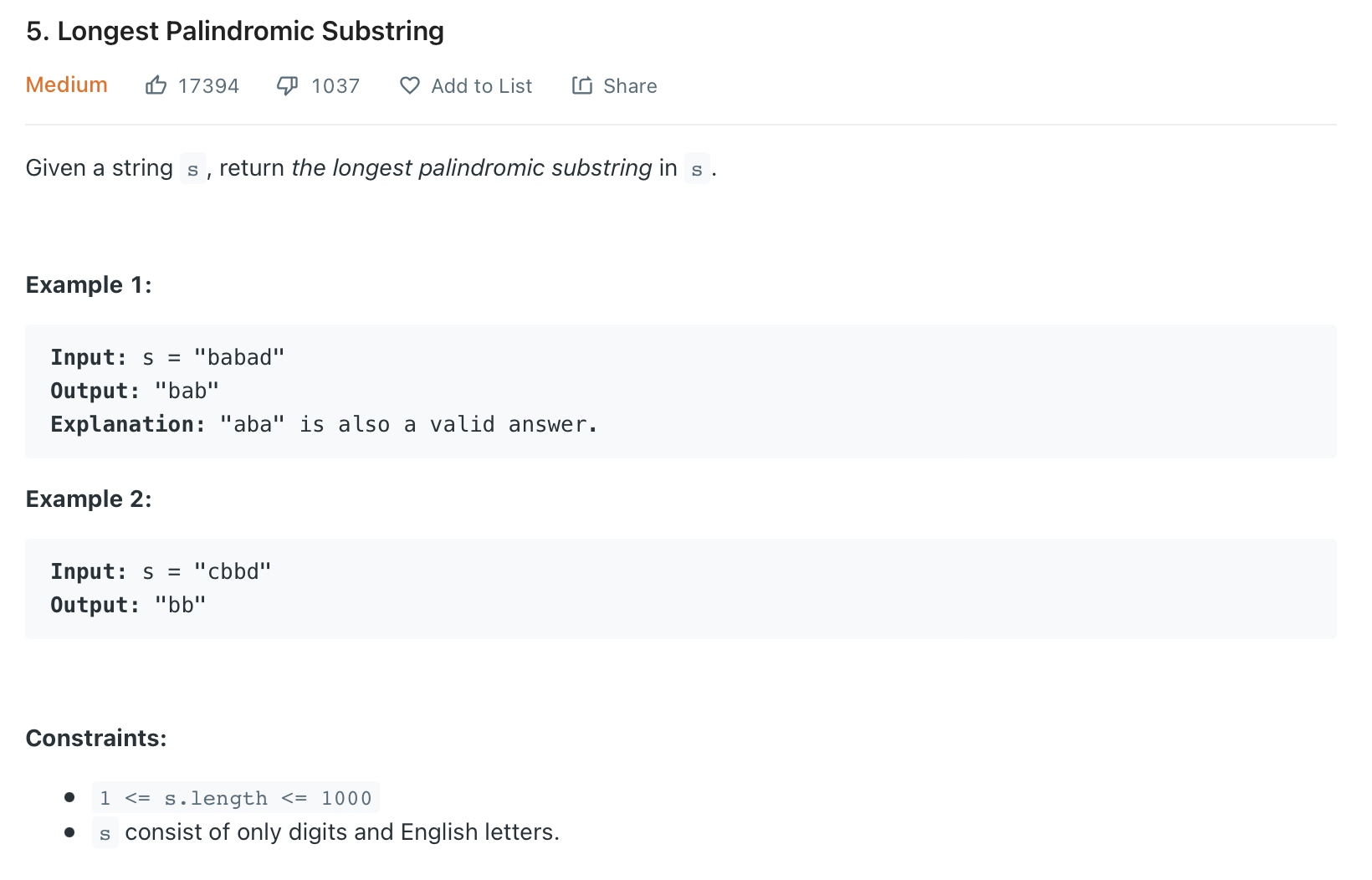
**LeetCode - Algorithm - Medium - Longest Palindromic Substring**



**Idea :**

(1)Failed : Time limited Exceeded  
Input : String : S

If (S == S[::-1]) :

Return S

Elif (len(Algo(S[:-1]) > len(Algo(S[1:])) :

Return Algo(S[:-1])

Else  
 Return Algo(S[1:])

(2) Failed : Time limited Exceeded  
Input : String : S

If (len(S) == 1)

Return S

Elif (len(S) == 2 and S[0] == S[1] )

Return S

If (len(Algo(S[1:-1]) == (len(S)-2)) and S[0] == S[-1])

Return (S[0] + Algo(S[1:-1]) + S[0])

Return ( max(Algo(S[1:-1] , Algo(S[:-1]) , Algo(S[1:]) ) )

(3)DP : TLE

s : target string  
k : length of substring

DP = [] (including head and tail of any Palindromic Substring)

For k in range(len(s))

Set = [i in range(len(s)) if (i+k) < len(s)]

if k == 0 :

for j in set :

DP.append([j,j])

elif k == 1 :

for j in set :

DP.append([j,j+1])

Else

for j in set :

if ([j+1 , j+k-1] in DP and s[j] == s[j+k])

DP.append([j,j+1])  
DP.remove([j+1 , j+k-1])

(4): Expanding middle

res = ""

resLen = 0

for i in range(len(s)):

L = i

R = i

while (L >= 0 and R < len(s)):

if (s[L] == s[R]):

if ((R-L+1) > resLen):

res = s[L : R+1]

resLen = R-L+1

L = L - 1

R = R + 1

else :

break

for i in range(len(s)):

L = i

R = i + 1

while (L >= 0 and R < len(s)):

if (s[L] == s[R]):

if ((R-L+1) > resLen) :

res = s[L : R+1]

resLen = R-L+1

L = L - 1

R = R + 1

else :

break

return (res)