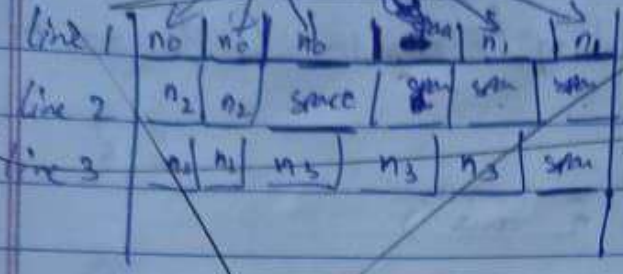
Word ~~Break~~ Wrap Problem

Date.
Page No.

nums = {3, 2, 2, 5} K = 6 output = 10

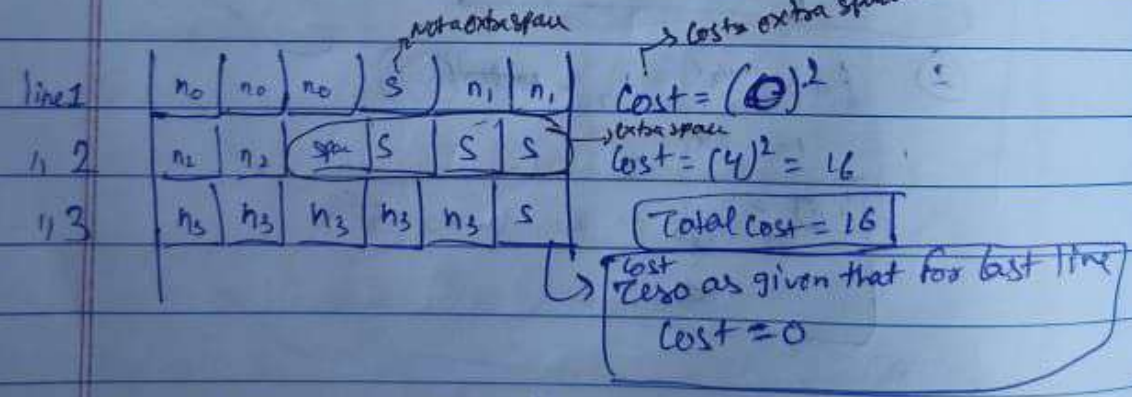
(3 → no. of char in one word; 6 → max limit of chars in line)



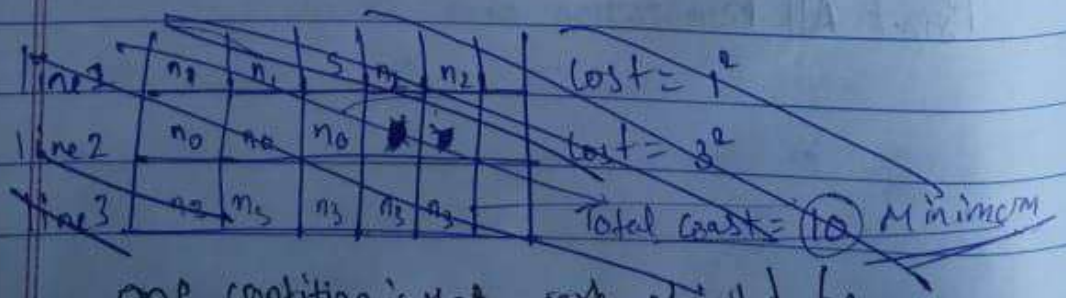
nums = {3, 2, 2} K = 4

nums = {3, 2, 2, 5} K = 6

(3 → no. of chars in one word; 6 → max no. of char in line)



But this is NOT the minimum cost



One condition is that word should be in the same order as in the array
↳ sequence

Date.
Page No.

line 1	no	no	no			cost = 3^2
line 2	n_1	n_1		n_2	n_2	cost = 1^2
line 3	n_3	n_3	n_3	n_3	n_3	Total = 10 ↓ minimum

So it is of choice type of ~~Prob~~ problem

Look Here

'car' 'in' 'by' 'night'

we can fill 'car' in the 1st line

line 1	c	a	r			
line 2						
line 3						

for 'i' 'n' I have 2 choice either fill it in line 1 or line 2

But we do not know which will give us the minimum Hence

We need to take the Both case in ~~parallel~~ finish one then 2 then find min in cost and store ~~it in a DS~~ the info of "extra space" in the Matrix (or any other DS) Hence D.P.

I Don't Know But How to code the DP problem

How to store cost

so look we need a Matrix ⇒ of size

dp[nums.size()][k]

constraint given

$1 \le$ nums.size ≤ 500 Hence dp[501][2001]

$\max(\text{nums}[i]) \le k \le 2000$

to Handle all Test Cases

Date.
Page No.

Q2 How to write code

Base condition

+

some edge condition (optional)

{

logic function call

↳ on the basis of pick or not pick

}

Q what to return

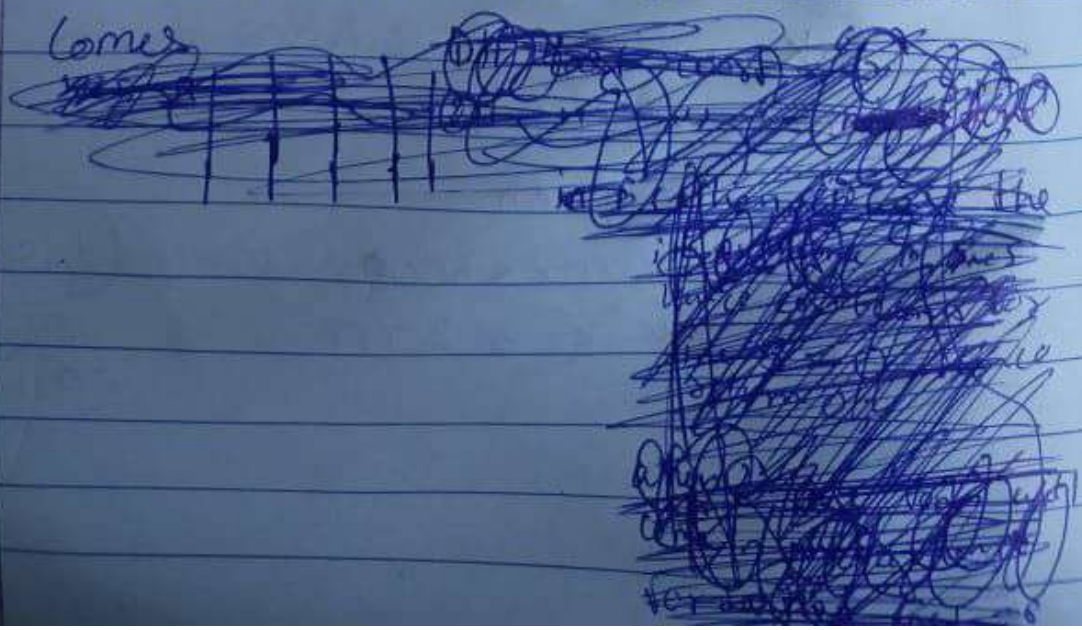
Here we will return the cost after making a particular choice for the given word

Q What will be the Base condition or edge condition

Base condition comes when we are at the last word of nums Hence .

(i == nums.size)

and we know for last word where ever it comes



Base case if(i == nums.size)
return 0;

and to use the dP we will Have

if (dp[i][rem] != -1) //means for this we Have cal. earlier cost
return dp[i][rem];

So. these are 2 possibility

Pick ① When the next word will be in same line

NotPick ② When the next word will be taken in next line

~~But in the ① case(Pick) there is also a thing if remaining space is < word length then we will do only ① function call which is of NotPick (word is in next line)~~

if (nums[i] > rem)
{ only one possibility Hence
ans = (rem+1)*(rem+1) + rec(i+1, K-arr[i]-1, _)
(rem+1)*(rem+1) → cal. wasting cost
K-arr[i]-1 → means

example
a| | | | |

nums[i] = 5 word = "hight"
rem is 4
Satisfy the above if condition
Hence before making call for Another word(Next Word) (i+1)
we will cal. cost for this line

we have reserved the (arr[i]+1) space in the next line Hence
arr[i] → word char of "hight"
we are passing K-arr[i]-1 as rem space for the [i+1]th word.

if the above if condt. unsatify means we have both possibility for that word(right)

① pick ② not pick

Hence

```
else
{
    int c1 = (rem+1)*(rem+1) + sol(i+1,
             // ^ cost
    int c2 = sol(i+1, rem-arr[i]-1, arr, k);
             // rem -> Because same line
```

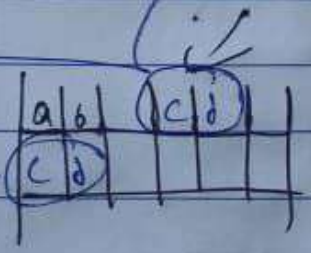
then we will choose the minimum cost

```
    ans = min(c1, c2);
}

dp[i][rem] = ans; // so that we reuse it if it comes again for evaluation

return ans;
}
```

because next line

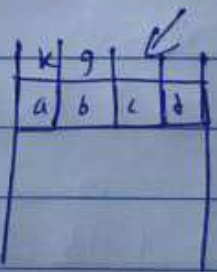
rem-arr[i]-1, ...); // condition of NOT pick that word on that particular line

// condition of pick that word on that particular line Hence there is NO cost as



num[i] = 2 word = "cd"
rem = 3

One more thing here
rem can become (-1) and we have use dp[i][rem] in our function but in some compiler it can give segmentation fault Hence you can use if(rem > -1)



line on

nums[0] = 4 word = "abcd"
K = 4

int c1 = 1 + rec(i+1, rem - arr[i] - 1 ...)

-1

int c2 = rec(i+1, rem - arr[i] - 1 ...)

-1 Hence for next word rem will be -1

nums = [3, 2, 2, 5]   k = 6

① rec(0, 6)

c1 = 3*3 + ② 10
c2 = ⑩ 10
ans = (19, 10)
ans = 10
~~return 10~~

② ~~rec(1, 2)~~

c1 = 3*3 + ③ ~~1~~
c2 = ⑧ 16
~~ans = min(18, 10)~~
~~ans = 10~~
dp[1][2] = 10
return 10;

③ ~~rec(2, 3)~~

c1 = 4*4 + ④ 16
c2 = ⑥ 1
ans = min(1)
dp[2][3] = 1
~~return 1 to fun②~~

④ ~~rec(3, 3)~~

Date: / /

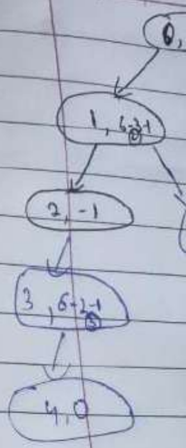
a[3] > 3
ans = 4*4 + ⑤ (i=4) 0
dp[3][3] = 16
return 16 to ④ call

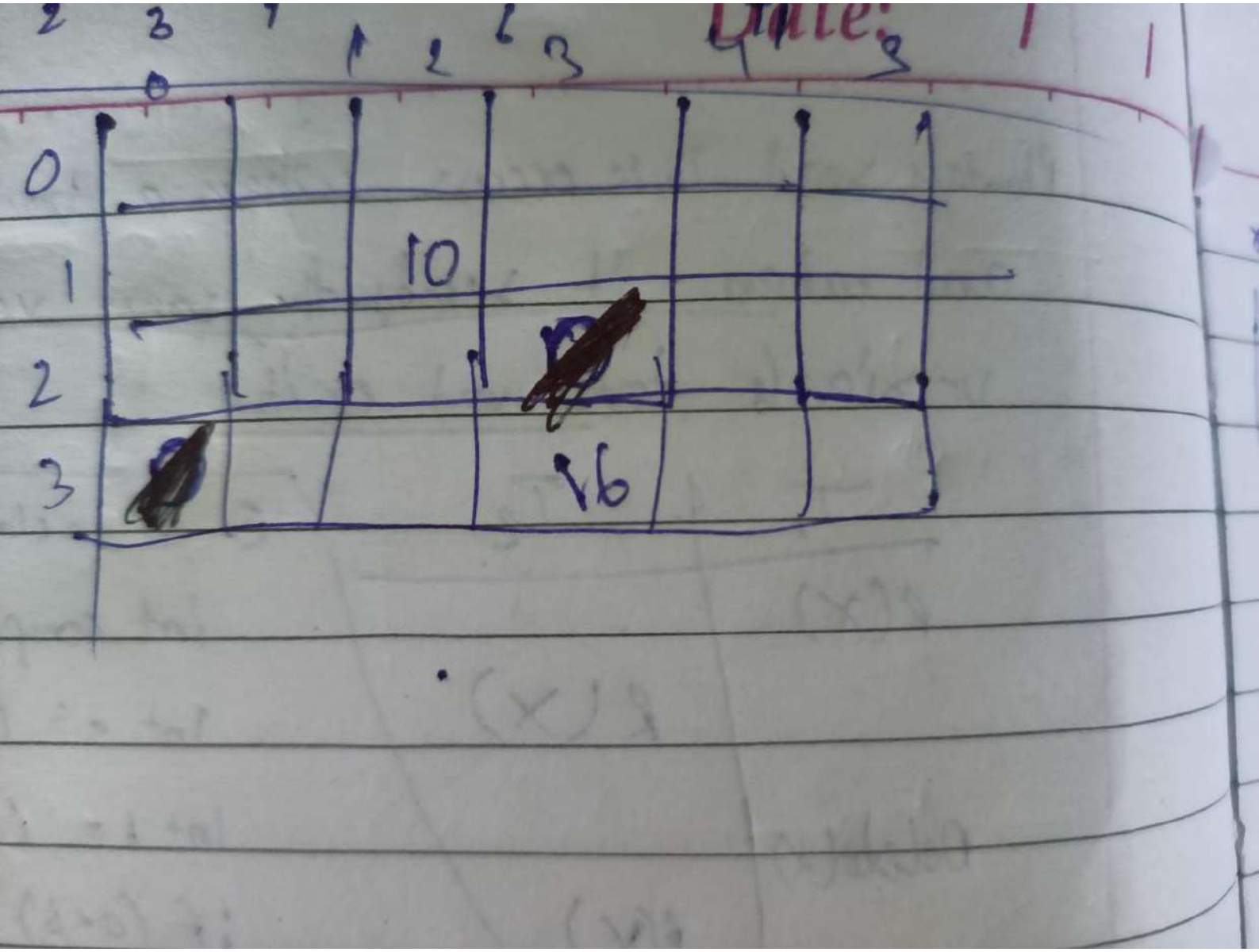
⑥ ~~rec(3, 3-2-1)~~
a[3] > 0
ans = 1 + ⑦ i=4 → 0
dp[3][0] = 1
return 1 to fun③

⑧ ~~rec(2, 2-2-1)~~
(arr[2] > -1)
ans = 0 + ⑨ 16
dp[2][-1] = 16
return 16 to fun②

⑨ ~~rec(3, 6-2-1)~~ (3)
dp[3][3] != -1 ✓
return 16

⑩ rec(1, 2)
dp[1][2] != -1 ✓
return 10




0
1
2
3
10
16