You are given a **0-indexed** 2D integer array `questions` where `questions[i] = [points_i, brainpower_i]`.

The array describes the questions of an exam, where you have to process the questions **in order** (i.e., starting from question `0`) and make a decision whether to **solve** or **skip** each question. Solving question `i` will **earn** you `points_i` points but you will be **unable** to solve each of the next `brainpower_i` questions. If you skip question `i`, you get to make the decision on the next question.

- For example, given `questions = [[3, 2], [4, 3], [4, 4], [2, 5]]`:
  - If question `0` is solved, you will earn `3` points but you will be unable to solve questions `1` and `2`.
  - If instead, question `0` is skipped and question `1` is solved, you will earn `4` points but you will be unable to solve questions `2` and `3`.

Return *the* ***maximum*** *points you can earn for the exam.*

skip skip

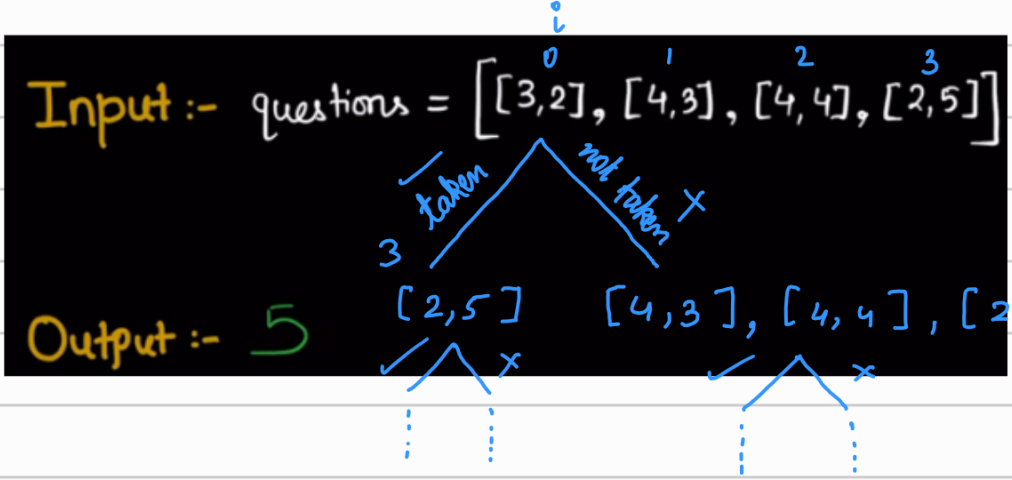
Input :- questions = [[3,2], [4,3], [4,4], [2,5]]

meaning of above is that let's say we solved first ques so we will get 3 points but we won't be able to solve since brainpower is 2 of the first question.

[3, 2] next 2 questions

Output :- 5

dp question → taken / not taken.



if taken grab that el's pt & go to i + (2) + 1 element
Brainpower of the element
[Since we have to skip brainpower no of next elements].

if not taken, go to next element.

Recursion + Memoization

```
7   //Approach-1 (Using Simple Recursion + Memo) - Knapsack variant
8   class Solution {
9   public:
10
11      int n;
12
13      long long solve(int i, vector<vector<int>>& questions, vector<long long> &t) {
14
15          if(i >= n)
16              return 0;
17
18          if(t[i] != -1)
19              return t[i];
20
21          long long taken     = questions[i][0] + solve(i+questions[i][1]+1, questions, t);
22
23          long long not_taken = solve(i+1, questions, t);
24
25          return t[i] = max(taken, not_taken);
26
27      }
28
29      long long mostPoints(vector<vector<int>>& questions) {
30          n = questions.size();
31          vector<long long> t(n+1, -1);
32          return solve(0, questions, t);
33      }
34  };
```

Ex: [3, 2] → q[i][0] → points.
→ q[i][1] → brain power

taken → pts
go to i + brain power + 1
not taken → go to next el.

BOTTOM UP APPROACH

during recursion + memo only index was changing solve(i, qus)
(i)

∴ we can solve it using 1D array.

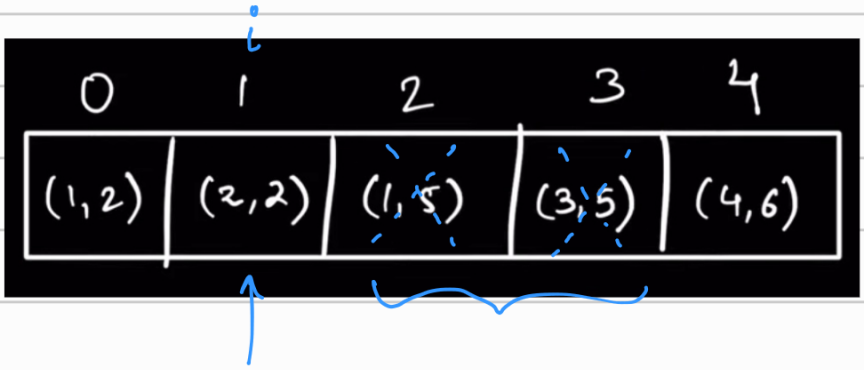
vector<int> t

// t[i]

State defination :

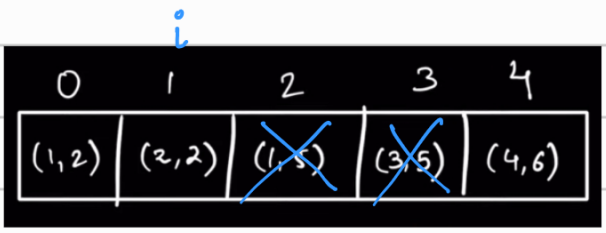
In bottom up what we do is to find answer of t[i]
we try to find answer by relating from the past Ex: t[i-1]
t[i-2], t[i-3] ..... .
t[i]

But the problem here in this question is ↓

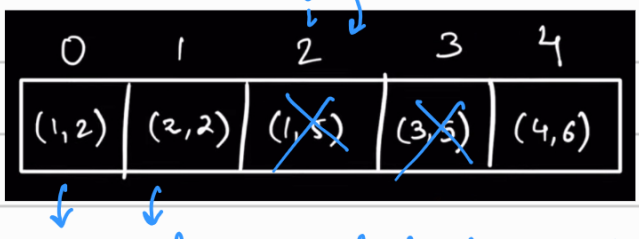


Let's say i is here its impacting future 2 tasks. [Brainpower]
so how to solve using bottom up [where it dipendent on the past]
but here its diff. ↓ (any task is impacting future task)

Let's say i was here, so index 2 & 3 will be skipped
↓ (discarded)



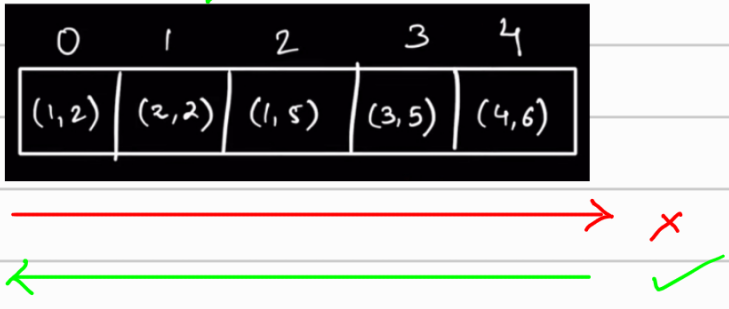
now i comes here



now how would he know if its been discarded or not. where in the past it has been discarded from (either index 0 or 1) we do not know.

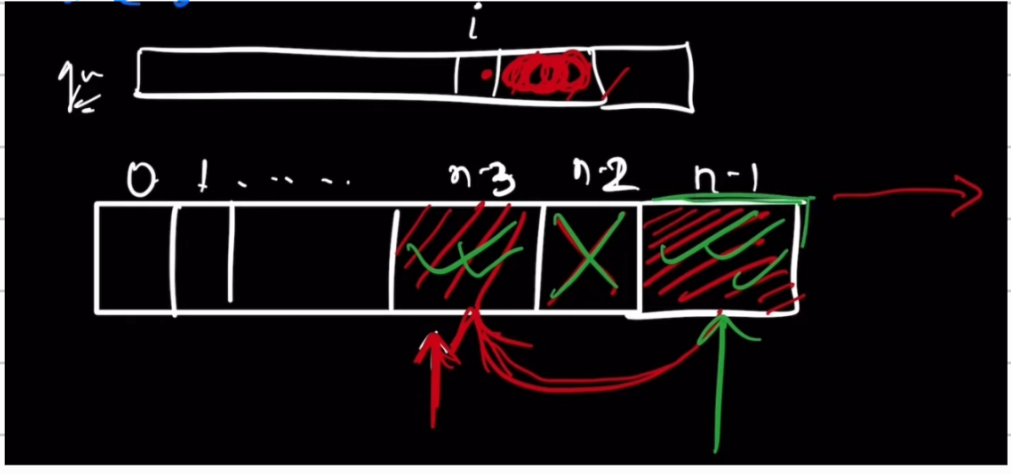
∴ we can not go from left to right (as any task is impacting future tasks).

∴ we will move from right to left.



Right to left :-

t[i] =



Let 's say we are at i & brainpower is 1 ∴ we will skip one element

now in bottom up array we are solving from right to left ∴ to solve n-3 (i 's equivalent) we can take help of n-1 & after (future elements) but here its past since we are solving from R to L & skip n-2 (due to brainpower)

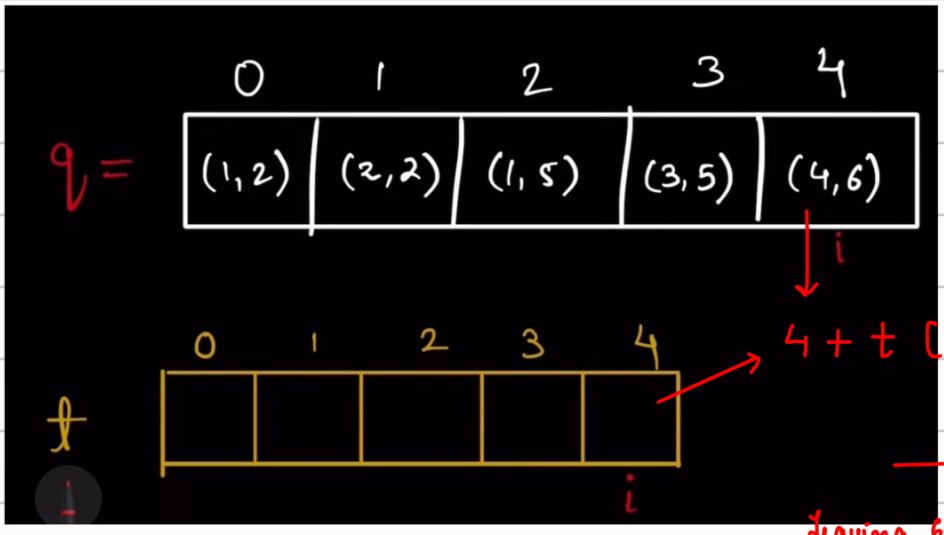
$t[i]$ = max points gained by Qns from $[i \rightarrow n-1]$

given in question 0 to n-1
& $t[i]$ = i to n-1

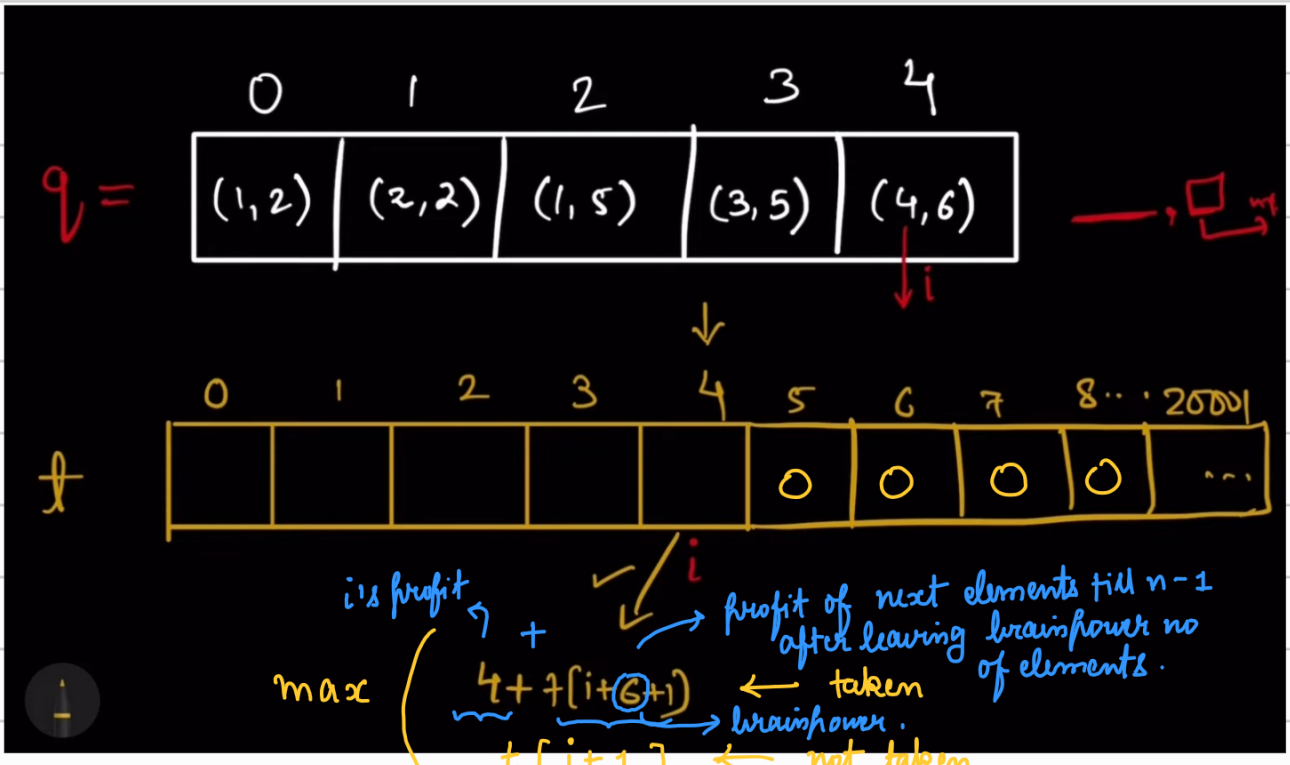
∴ result = $t[0]$ → max pts gained by Qns from $(0 \rightarrow n-1)$

Hence we need to return $t[0]$.

DRY RUN...

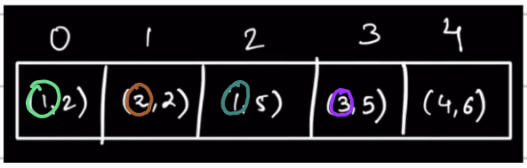


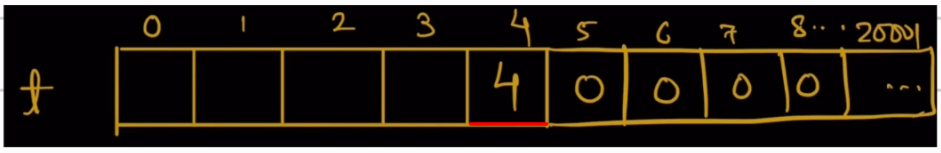
But it will get out of bounds.
∴ take a big array & put 0.
[so out of bound issue is resolved].

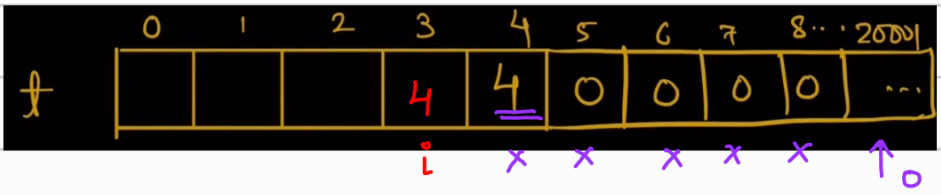


↑
in not taken case since we are not taking i ∴ we won't get any profit ∴ whatever profit we got from right side we will put the same profit here only (since not taken).

$$t[i] = \max\big(\underbrace{q[i][0] + t[i+q[i][1]+1]}_{\text{taken}},\ \underbrace{t[i+1]}_{\text{not taken}}\big);$$





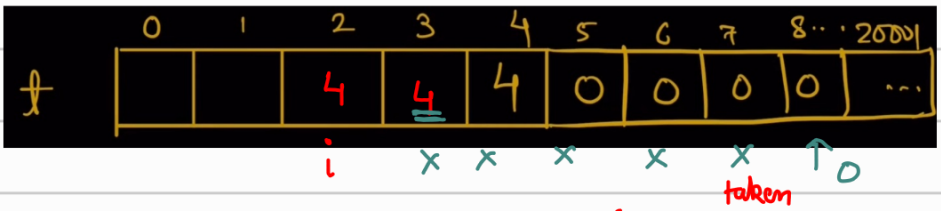


$\max\left(\underline{3} + t[i+5+1],\ \underline{4}\right)$

taken — not taken.

(∴ take previous point)

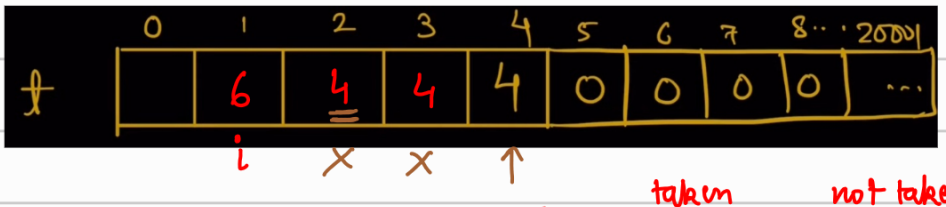
(discussed earlier)



taken

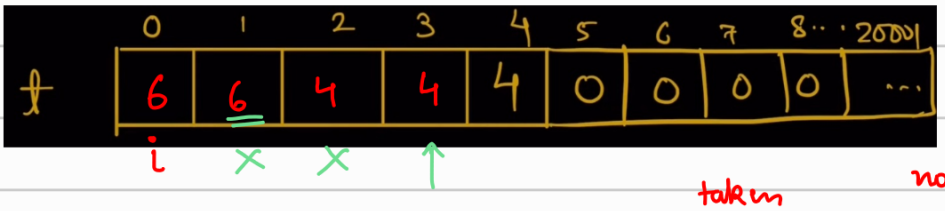
$\max\left(\underline{1+0=1},\ \underline{4}\right)$

not taken



taken — not taken.

$\left(\underline{2+4=6},\ \underline{4}\right)$



taken — not taken.

$\left(\underline{1+4=5},\ \underline{6}\right)$

∴ answer → $\underline{t[0] = 6}$

```cpp
class Solution {
public:
    long long mostPoints(vector<vector<int>>& q) {
        int n = q.size();

        if (n == 1)
            return q[0][0];

        vector<long long> t(200001, 0);
        //t[i] = max points gained by Qns from i to n-1
        //return t[0] -> 0 to n-1

        for(int i = n-1; i>=0; i--) {
            t[i] = max(q[i][0] + t[i + q[i][1] + 1], t[i+1]);
        }

        return t[0];
    }
};
```

taken

not taken