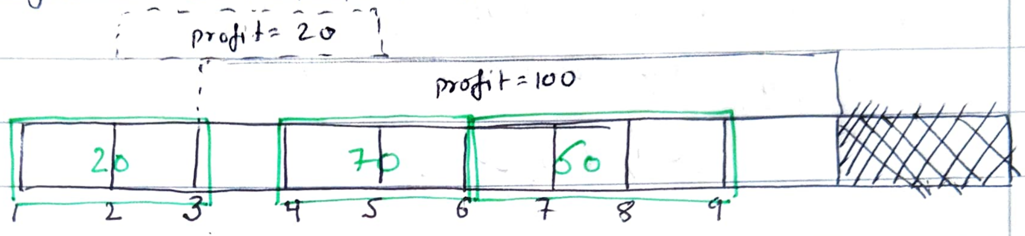
1235: Maximum Profit in Job Scheduling

starttime = [1, 2, 3, 4, 6]
endtime = [3, 5, 10, 6, 9]
profit = [20, 20, 100, 70, 60]



max_profit = 20 + 70 + 60 = 150

- Schedule jobs in such a way that you get maximum profit.

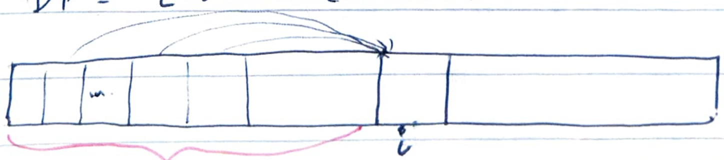
① Approach but high memory consumption (memory limit exceeded)

→ DP problem: As we have to calculate maximum and also plus ~~~~ to calculate profit we have to add the profit from previous schedules.

- → Here I first tried with the tabulation approach ~~~~, although I got the logic right but it backfired.
  → First start with Brute-force, try to create a tree, if that doesn't work, then see if you can use tabulation directly.

Approach

DP = [0] * max(endtime)



assume you know the answer of previous path.

- $dp[i] = \cancel{profit[i]} \max_{c \in hash[i]} (dp[c] + profit(c-i))$

There would be multiple ways to reach i (i means end time) here. We have to iterate those ways and get the maximum profit at time i.

We can have a hash-map, which indicates in what ways particular time can be reached.

9 : [[6, 60]] → profit from 6 – 9
6 : [[4, 70]]
10 : [[3, 100]]
5 : [[2, 20]]
3 : [[1, 20]]

hash

Theoretically:

i : $[[x_1, \text{profit}\_x_1 i], [x_2, \text{profit}\_x_2 i] \dots [x_n, \text{profit}\_x_n i]]$

$$\boxed{dp[i] = \max_{x \in hash[i]} \left(dp[x] + \text{profit}\_xi \right)}$$

dp[max(end time)] ⇒ will return the answer.

Memory consumption and time consumption depends on the maximum time present in end time list.

For example:- [1, 2, 3, 4] = start Time

[2, 4, 7, 10] → endTime

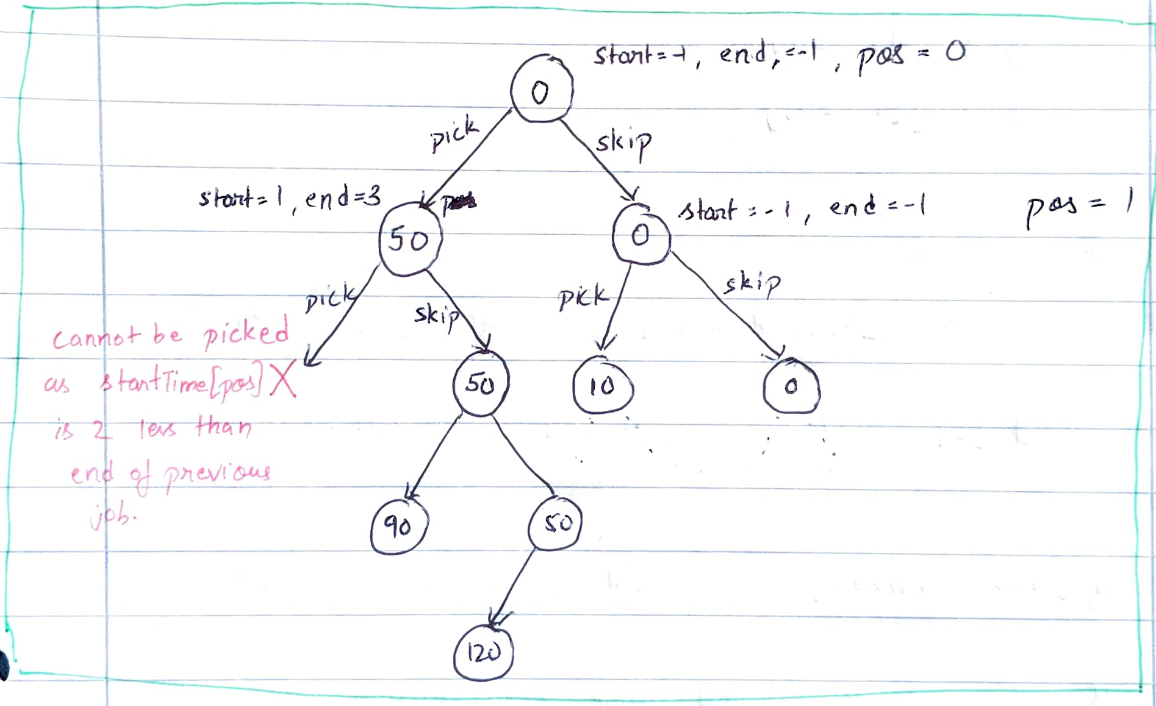
In this case although above list size is quite small. The size of the dp array will be $O(\max(\text{endTime}))$

Brute force approach with memorization

At any position we can skip a job or take the job into account, the job at current position can only be taken into the account if endTime of previous job is less than starttime of current job.

→ All three list have to be sorted in ascending order of startTime.

start time = [1, 2, 3, 3]
end time = [3, 4, 5, 6]
profit = [50, 10, 40, 70]



This is a simple approach and the recursive function is also simple to write for this. But the recursive calls are two much This also exceeds the memory limit

Binary search approach

The arrays are sorted, in the previous approach we were incrementing (pos) everytime and checking if current starttime is more than previous endTime.

(This can lead to more recursive calls.

Instead of incrementing by 1, we can directly find the index ~~~~ to which we should increment by using binary search.

Use binary search (bisect_left) to find the next startTime that is greater than current end Time, this removes a lot of unnecessary recursive calls.