Find solution by Brue force:

N jobs:

There are two cases that the maximum pay off time can be recalculated.
The set containing the optimal solution at most has N jobs.

The problem can be divided into sub problems, so it's ideal to use recurrence relations to solve the problem.

Max payoff job contains the cost of Job (i=N):

- A part of optimal solution will contain the cost of Job(i=N). Since all the indices from [largest non-overlapping job(N), N] will be overlapping with N, we will look for optimal solution in [1, largest non-overlapping job(N)]. Hence the overall maximum payoff job is cost_job_job(N) + max_payoff(largest non-overlapping job(N))

Max payoff job does not contain the job(i=N):

- We must look for the optimal solution in [1, N-1]. Hence the overall maximum payoff job of is in the remaining jobs(N-1)

Therefore, to maximize the value of intervals, we pick the larger case as our optimal solution.

Recurrence relation:

$$\mathrm{Optimal} - \mathrm{solution}(\mathrm{N}) = \begin{cases} 0 & \mathrm{N} = 1 \\ \mathrm{Max}\Big(w_{\mathrm{N}} + \mathrm{OptimalSolution}\big(\mathrm{largestNonoverlappingIndexJob}(\mathrm{N})\big), \mathrm{OptimalSolution}(\mathrm{N} - 1)\Big) & \mathrm{Otherwise} \end{cases}$$

Optimizing the brute force algorithm

Rather than naively doing the recursive calls, we began calculating Optimal-Solution for values 1 to N-1. This allows us to look up a value for a recursive call if it has already been created, and it's guaranteed that the any given index value is calculated only once because the optimal solution for every index is fetched from a hashmap if it's been previously calculated.

```
for i in range(1, n):
    non_overlapping_index = find_largest_index_for_none_overlapping_job(i)
OPTIMAL[i] = max(jobs_sorted_by_finish_time[i].cost
         + OPTIMAL[non_overlapping_index], OPTIMAL[i - 1])
```

Largest Non-overlapping job's index

To find the largest index of a non-overlapping job given a job, we follow the same logic the binary search. $Complexity\ O(\log N)$

1) Sort the jobs based on their finish time
2) Find the mid job
3)
4) If the mid job's finish time is larger than the jobs started, then we have found a candidate. However, this may not be the largest index. Therefore, we repeat the process for the jobs from index Mid +1 to the end.

Find the optimal path given the optimal solution. $Complexity\ O(NlogN)$

1) we write down the recurrence relation for index N-1

$$\mathrm{OptimalSolution}(\mathrm{N}-1) = \mathrm{Max}\Big(\mathrm{w}_{\mathrm{N}-1} + \mathrm{OptimalSolution}\big(\mathrm{largestNonoverlappingIndexJob}(\mathrm{N}-2)\big), \mathrm{OptimalSolution}(\mathrm{N}-2)\Big)$$

2) program terminates when the recurrence relation is laid out for index 1
3) if $\mathrm{w}_{\mathrm{N}-1} + \mathrm{OptimalSolution}\big(\mathrm{largestNonoverlappingIndexJob}(\mathrm{N}-1)\big)$ is larger than $\mathrm{OptimalSolution}(\mathrm{N}-1)$
    a. Job N-1 is part of the optimal path. Record Job N-1
    b. Repeat step 1 for index $\mathrm{largestNonoverlappingIndexJob}(\mathrm{N}-1)$
4) if Step 2's condition is false,
    a. Repeat step 1 for index $\mathrm{N}-2$