# CS 330 – Weighted Interval Scheduling

deadline: Monday 11/18/19 at 11:59 pm

In this exercise you will implement a dynamic programming algorithm for solving the weighted interval scheduling problem. This algorithm is described in *Algorithm Design*, Section 6.1, with a memoized version on page 256 and a bottom-up version on 259. The pseudocode for calculating the set of intervals is on page 258. Your code should run in $\mathcal{O}(n \log n)$ time, where $n$ is the number of intervals provided. Note that the textbook states the algorithm runs in linear time, because it assumes the $p(j)$ values are already computed. You will have to implement this step yourself. The lecture slides contain some additional information about this.

You may use built-in sorting functions.

## Tips

- You'll need to compute the $p(j)$ array. One way you could do this is to create a copy of the intervals array and sort it by increasing start time, then walk over the two arrays backwards, advancing indices as needed.
- The book uses one-indexing for the intervals with zero standing in for the empty set. Our intervals are zero-indexed, so be aware of the discrepancy.

## Submission Format

On Gradescope submit `weighted_interval_scheduling.py` or `weighted_interval_scheduling.java` (names must match exactly) reading a text file `input` and creating a text file `output` containing your solution. You may submit other files containing additional functions and classes. The autograder will put your code in the same directory as `input`, call it from the command line, and look for the file `output` in the same directory.

## Input

The input file `input` will specify a list of intervals along with their associated weights. The first line will contain $n$, the number of intervals. Each line after that will contain the following values, separated by commas: the index of the interval, the start time, the finish time, and the weight. All four of these will be integers. Assume the intervals are half-open, so two intervals can share an end and start time while being compatible. You can assume the finish times will be distinct, but the start times may be repeated. The intervals will be ordered by increasing finish time.

We have uploaded files corresponding to the visible test cases on Gradescope. They may help you test and debug locally. Note: passing these test cases does not guarantee your code is correct!

## Output

Your algorithm must create a text file `output` listing the indices on individual lines. You can specify the indices in any order. Do not write any time or weight information, just the indices!

## Example

The following example is from the lecture slides on dynamic programming. We've changed the letter names to integer indices.

```
input:

8
0,1,4,4
1,3,5,3
2,0,6,10
3,4,7,3
4,3,8,12
5,5,9,15
6,6,10,9
7,8,11,8
```

The output is just the two indices of the optimal schedule. Interval 4 ends at time 8 and interval 7 begins at time 8, so they do not overlap.

```
output:

4
7
```