Lecture 34

CSE 331

Nov 26, 2018

Mini project video has been graded

note ☆ 15 views

Mini project video grades and stats

I apologize for the delay on getting this but the mini project video has now been graded and scores released on Autolab.

Some important points before I go ahead:

- The peer evaluation part **HAS NOT BEEN GRADED YET**. That'll get done by the end of the week: please wait for a post when that is done (see next point).
- The video points will be used for the peer eval score so this one will have a short re-grade request: **all regrade requests have to be sent to me by 5pm on WED NOV 28**
- As promised in @749, I paused the video at the 3min mark if it was longer. **Your submission was graded only on the first 3 mins**

Here are the **stats:**

Problem	Mean	Median	StdDev	Max	Min
Video Quality	12.6	12.0	2.0	15.0	7.5

Incentive to fill in course evals

note ☆ **0 views**

Incentive for filling in course evals

As I have done in the past few years, depending on the level of response on the official course evals, I will release come questions on the final exam. (See @975 to see what Q I mean below)

- If **>=85%** students submit the course evals, I will release **Q1(a)**
- If **>=90%** students submit the course evals, I will release **Q1(a) AND Q2(a)**

Some other relevant comments:

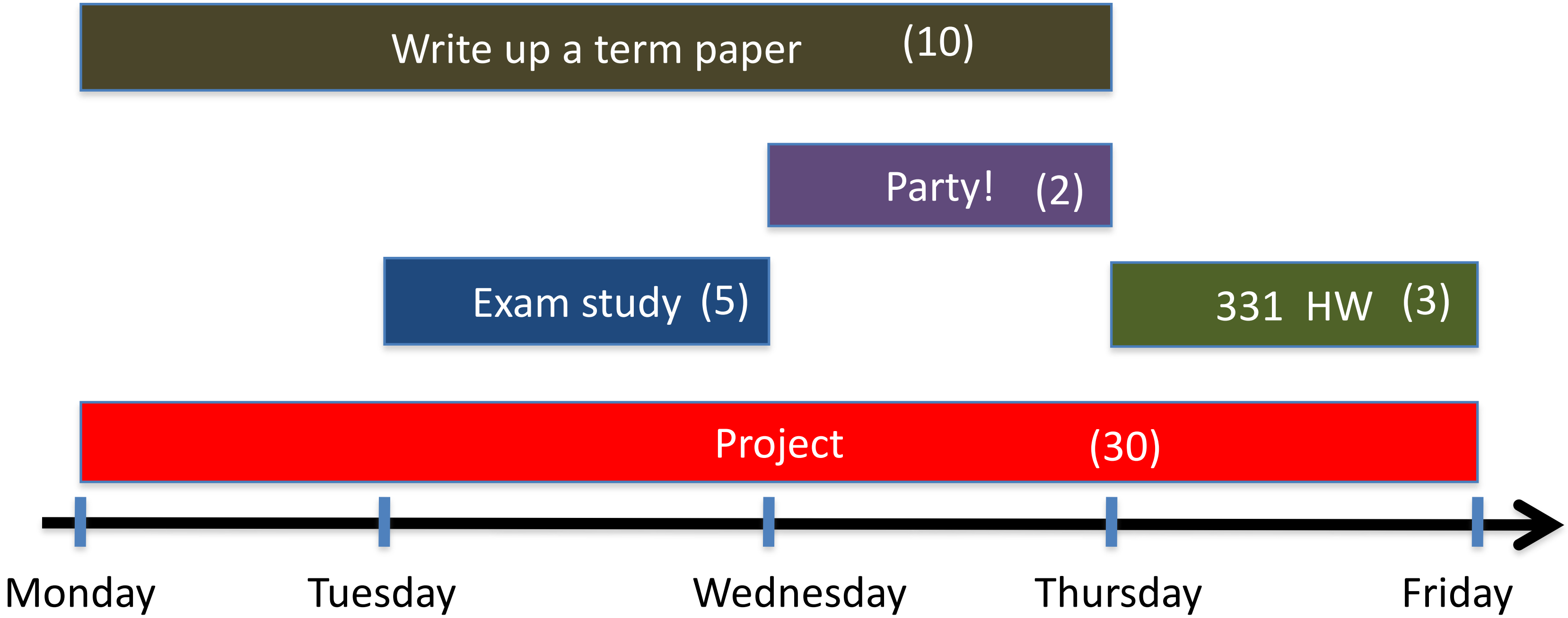
- I will post the current response rate in the comments section below every 3 days till the deadline
- The % is based on current student registered (236): i.e. it does not include students who have resigned
- I believe this is the link to the course evals: https://sunyub.smartevals.com/
  - But double check the email you might have received on this.

feedback

edit · good note | 0 Updated Just now by Atri Rudra

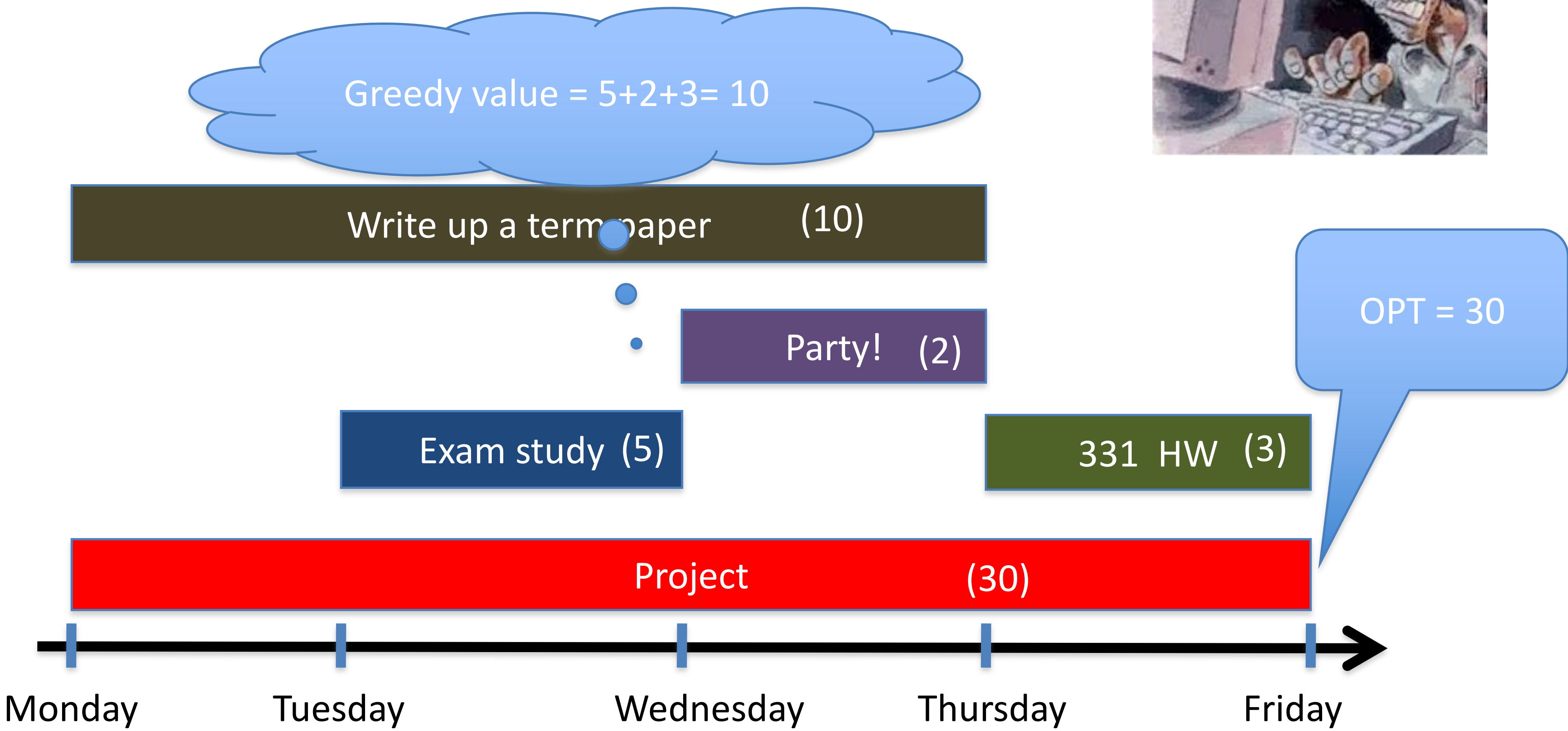
End of Semester blues

Can only do one thing at any day: what is the optimal schedule to obtain maximum value?



Previous Greedy algorithm

Order by end time and pick jobs greedily



Weighted Interval Scheduling

Input: n jobs (s_i, f_i, v_i)

Output: A schedule S s.t. no two jobs in S have a conflict

Goal: max $\Sigma_{i \text{ in } S} v_j$

Assume: jobs are sorted by their finish time

Today’s agenda

Finish designing a recursive algorithm for the problem

Couple more definitions

$p(j)$ = largest $i < j$ s.t. i does not conflict with j

= 0 if no such i exists

$p(j) < j$

$OPT(j)$ = optimal value on instance $1,..,j$

Property of OPT

j in OPT(j)

j not in OPT(j)

$$OPT(j) = \max \{ v_j + OPT(p(j)), OPT(j-1) \}$$

Given OPT(1),…., OPT(j-1), how can one figure out if j in optimal solution or not?


ICANHASCHEEZBURGER.COM

A recursive algorithm

Proof of correctness by induction on j

Compute-Opt(j)

Correct for $j=0$

If $j = 0$ then return 0

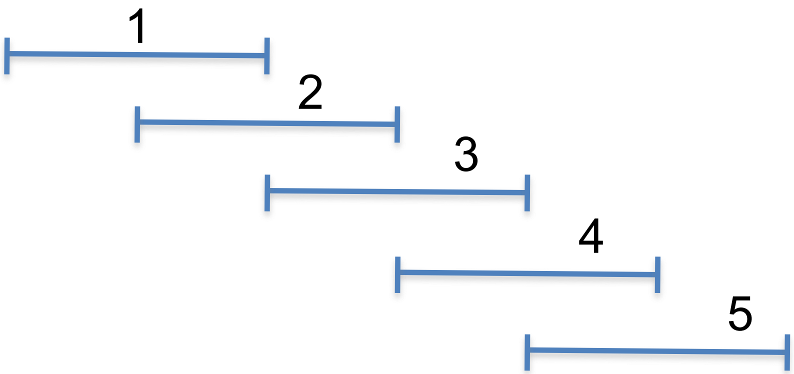
return max { v_j + Compute-Opt($p(j)$), Compute-Opt($j-1$) }

= OPT($p(j)$)

= OPT($j-1$)

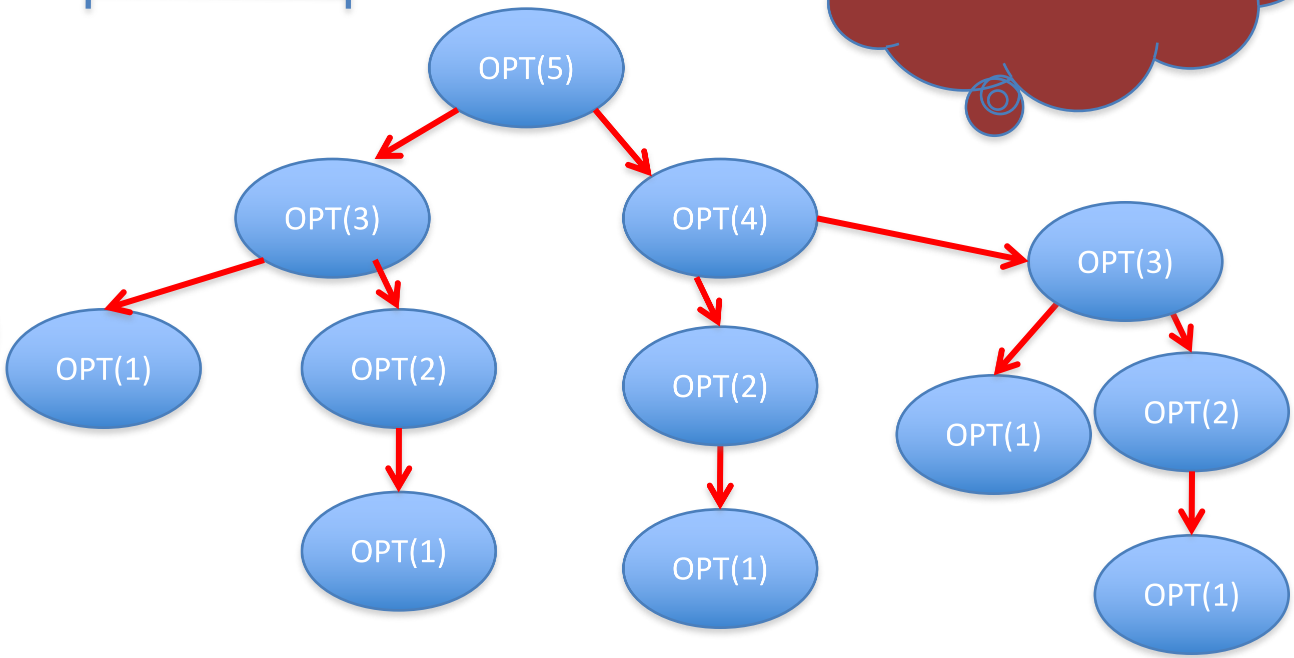
$$OPT(j) = \max \{ v_j + OPT(p(j)), OPT(j-1) \}$$

Exponential Running Time



$p(j) = j-2$

Only 5 OPT values!



Formal proof: Ex.


?
ICANHASCHEEZBURGER.COM

Using Memory to be smarter

```
Pow (a,n)
      .
      .
      .
      .
   // n is even and ≥ 2
   return Pow(a,n/2) * Pow(a, n/2)
      .
      .
      .
      .
```

O(n) as we recompute!

```
Pow (a,n)
      .
      .
      .
      .
   // n is even and ≥ 2
   t= Pow(a,n/2)
   return t * t
      .
      .
      .
      .
```

O(log n) as we compute only once

How many distinct OPT values?

A recursive algorithm

M-Compute-Opt(j)

If j = 0 then return 0

If M[j] is not null then return M[j]

M[j] = max { v_j + M-Compute-Opt(p(j)), M-Compute-Opt(j-1) }

return M[j]

M-Compute-Opt(j) = OPT(j)

Run time = O(# recursive calls)

Bounding # recursions

M-Compute-Opt(j)

If j = 0 then return 0

If M[j] is not null then return M[j]

M[j] = max { v_j + M-Compute-Opt(p(j)), M-Compute-Opt(j-1) }

return M[j]

O(n) overall

Whenever a recursive call is made an M value is assigned

At most n values of M can be assigned


ICANHASCHEEZBURGER.COM

Property of OPT

$$OPT(j) = \max \{ v_j + OPT(\ p(j)\ ),\ OPT(j-1) \}$$

Given OPT(1), …, OPT(j-1), one can compute OPT(j)

Recursion+ memory = Iteration

Iteratively compute the OPT(j) values

Iterative-Compute-Opt

M[0] = 0

For j=1,…,n

M[j] = max { v_j + M[p(j)], M[j-1] }

M[j] = OPT(j)

O(n) run time


ICANHASCHEEZBURGER.COM

Reading Assignment

Sec 6.1, 6.2 of [KT]

When to use Dynamic Programming

Richard Bellman

There are polynomially many sub-problems

Optimal solution can be computed from solutions to sub-problems

There is an ordering among sub-problem that allows for iterative solution