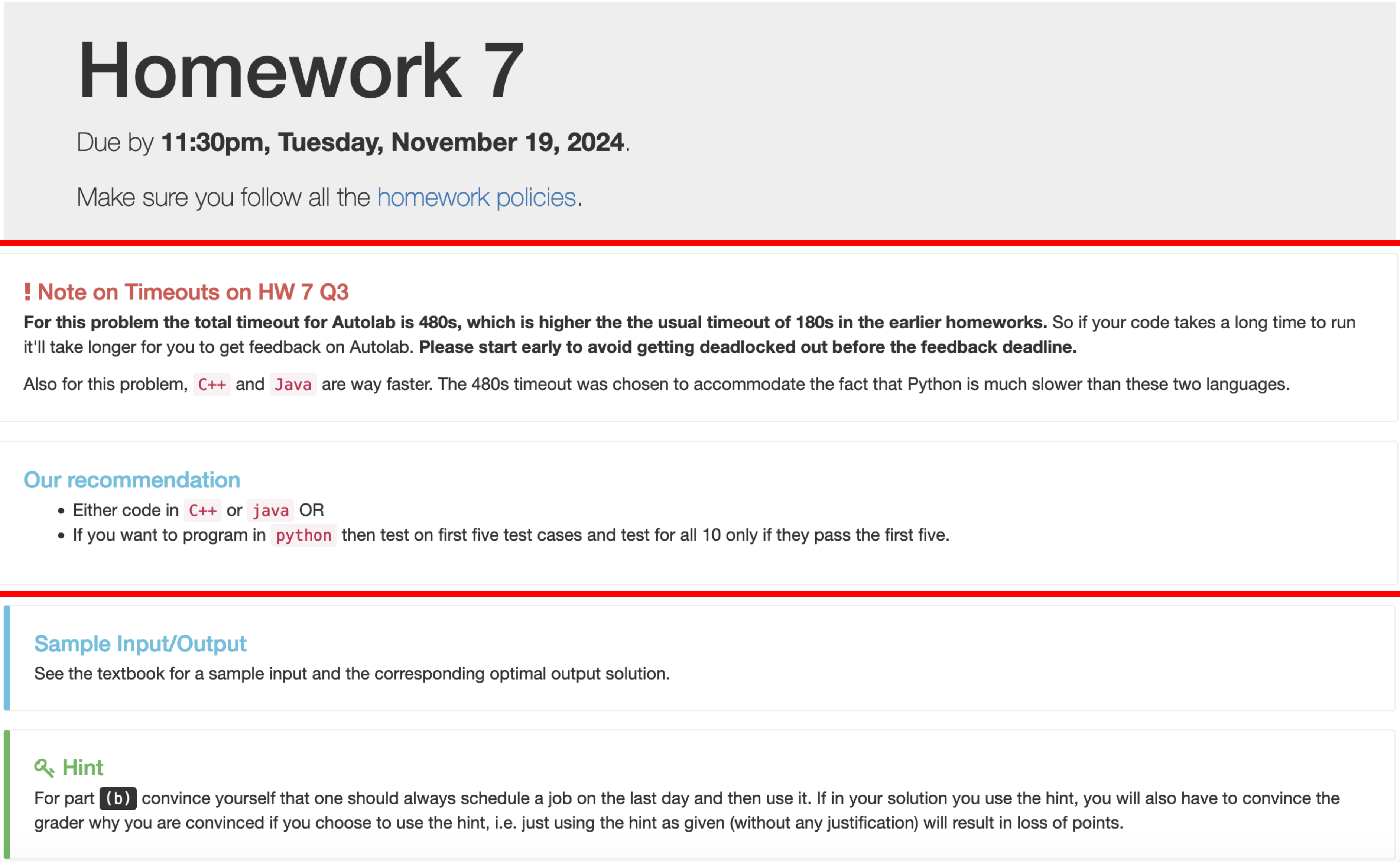
Lecture 31

CSE 331

Nov 13, 2024

Homework 7 out



Homework 7

Due by **11:30pm, Tuesday, November 19, 2024**.

Make sure you follow all the homework policies.

! Note on Timeouts on HW 7 Q3

For this problem the total timeout for Autolab is 480s, which is higher the the usual timeout of 180s in the earlier homeworks. So if your code takes a long time to run it'll take longer for you to get feedback on Autolab. **Please start early to avoid getting deadlocked out before the feedback deadline.**

Also for this problem, `C++` and `Java` are way faster. The 480s timeout was chosen to accommodate the fact that Python is much slower than these two languages.

Our recommendation

- Either code in `C++` or `java` OR
- If you want to program in `python` then test on first five test cases and test for all 10 only if they pass the first five.

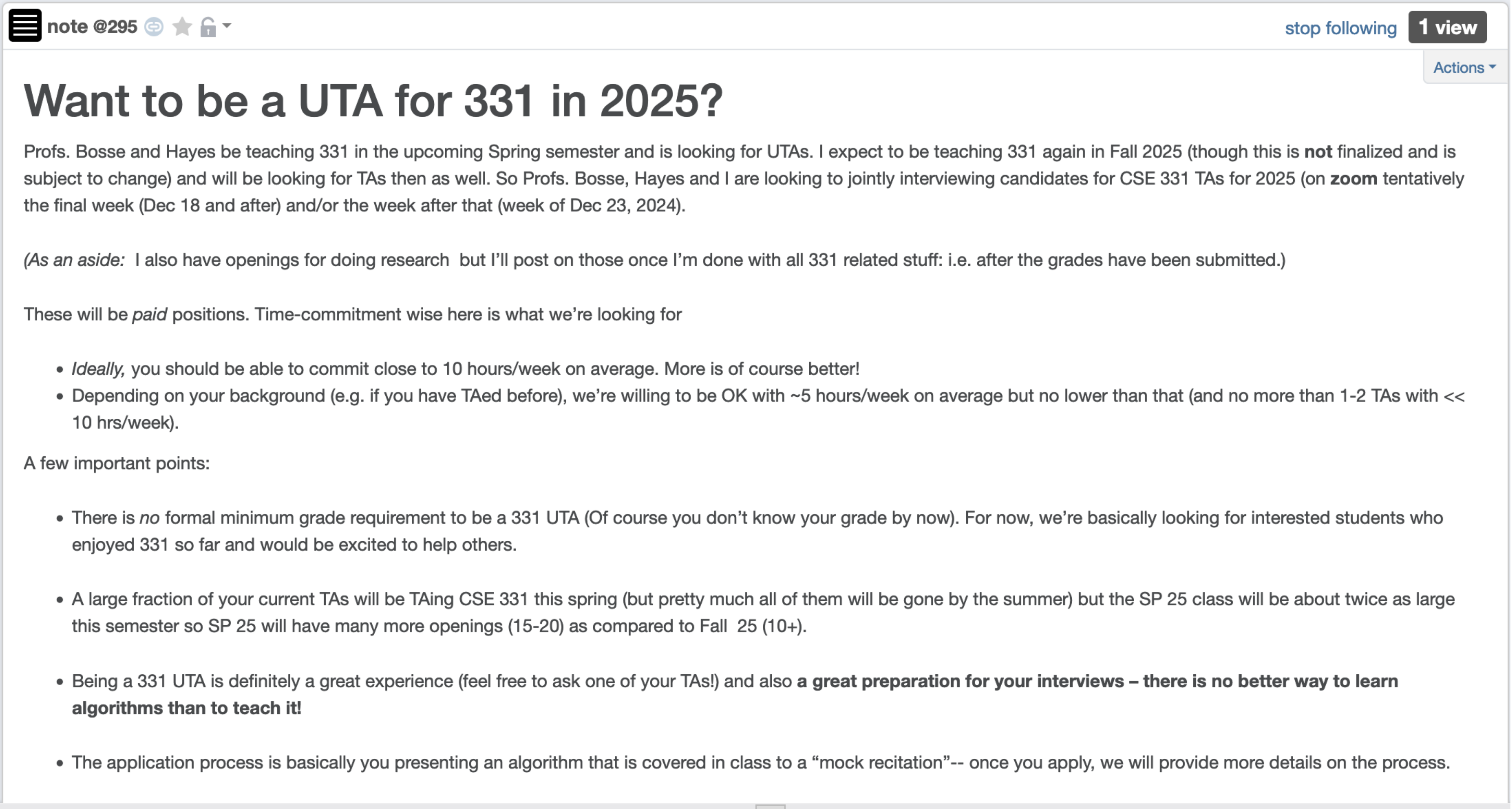
Sample Input/Output

See the textbook for a sample input and the corresponding optimal output solution.

Hint

For part `(b)` convince yourself that one should always schedule a job on the last day and then use it. If in your solution you use the hint, you will also have to convince the grader why you are convinced if you choose to use the hint, i.e. just using the hint as given (without any justification) will result in loss of points.

Apply to be a CSE 331 TA in 2025!



Want to be a UTA for 331 in 2025?

Profs. Bosse and Hayes be teaching 331 in the upcoming Spring semester and is looking for UTAs. I expect to be teaching 331 again in Fall 2025 (though this is **not** finalized and is subject to change) and will be looking for TAs then as well. So Profs. Bosse, Hayes and I are looking to jointly interviewing candidates for CSE 331 TAs for 2025 (on **zoom** tentatively the final week (Dec 18 and after) and/or the week after that (week of Dec 23, 2024).

(As an aside: I also have openings for doing research but I'll post on those once I'm done with all 331 related stuff: i.e. after the grades have been submitted.)

These will be *paid* positions. Time-commitment wise here is what we're looking for

- *Ideally,* you should be able to commit close to 10 hours/week on average. More is of course better!
- Depending on your background (e.g. if you have TAed before), we're willing to be OK with ~5 hours/week on average but no lower than that (and no more than 1-2 TAs with << 10 hrs/week).

A few important points:

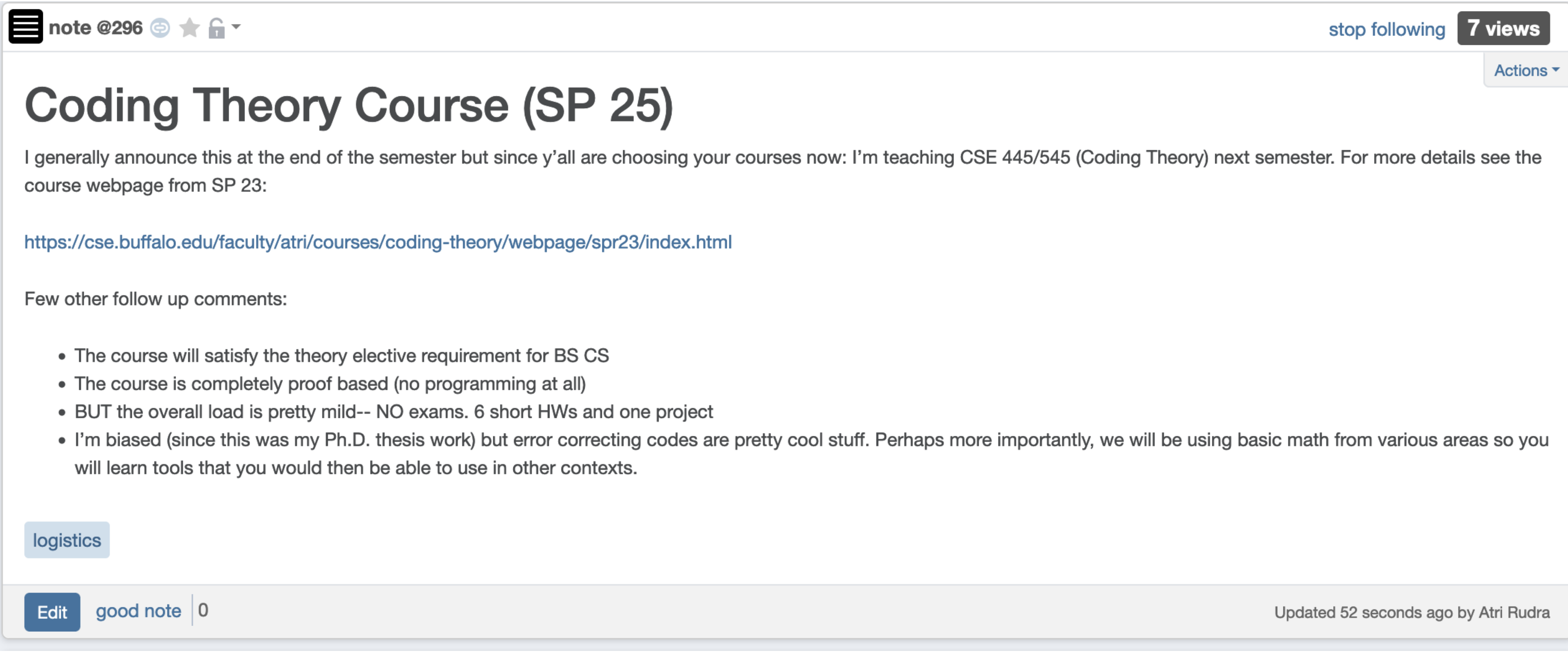
- There is *no* formal minimum grade requirement to be a 331 UTA (Of course you don't know your grade by now). For now, we're basically looking for interested students who enjoyed 331 so far and would be excited to help others.

- A large fraction of your current TAs will be TAing CSE 331 this spring (but pretty much all of them will be gone by the summer) but the SP 25 class will be about twice as large this semester so SP 25 will have many more openings (15-20) as compared to Fall 25 (10+).

- Being a 331 UTA is definitely a great experience (feel free to ask one of your TAs!) and also **a great preparation for your interviews – there is no better way to learn algorithms than to teach it!**

- The application process is basically you presenting an algorithm that is covered in class to a "mock recitation"-- once you apply, we will provide more details on the process.

Coding Theory (Spring 25)



Coding Theory Course (SP 25)

I generally announce this at the end of the semester but since y'all are choosing your courses now: I'm teaching CSE 445/545 (Coding Theory) next semester. For more details see the course webpage from SP 23:

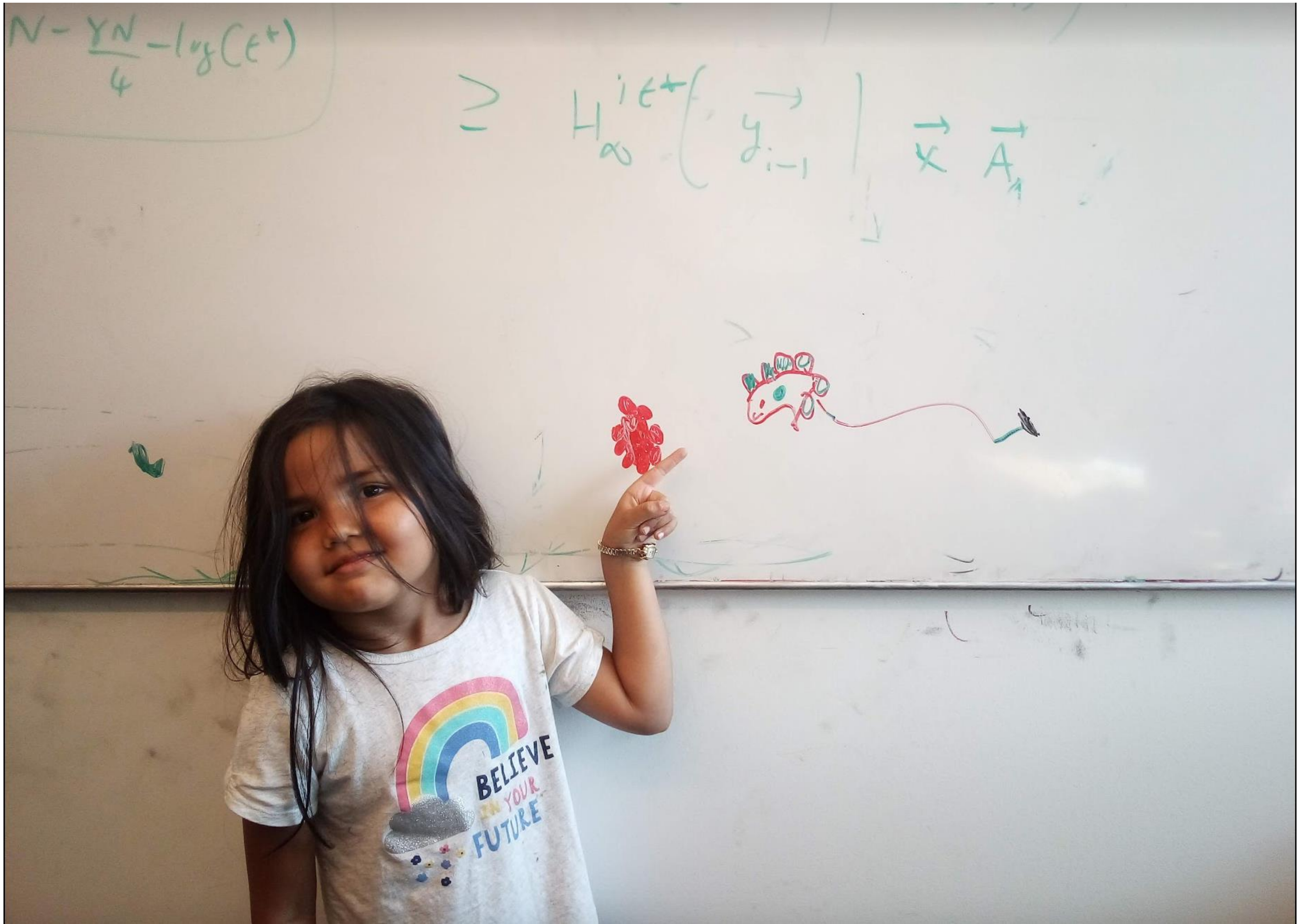
https://cse.buffalo.edu/faculty/atri/courses/coding-theory/webpage/spr23/index.html

Few other follow up comments:

- The course will satisfy the theory elective requirement for BS CS
- The course is completely proof based (no programming at all)
- BUT the overall load is pretty mild-- NO exams. 6 short HWs and one project
- I'm biased (since this was my Ph.D. thesis work) but error correcting codes are pretty cool stuff. Perhaps more importantly, we will be using basic math from various areas so you will learn tools that you would then be able to use in other contexts.

Updated 52 seconds ago by Atri Rudra

Questions/Comments?



$N - \frac{\gamma N}{4} - \log(\epsilon^+)$
$\geq H_\infty^{i\epsilon^+}(\vec{y}_{i-1} \mid \vec{x}\ \vec{A}_1$
BELIEVE IN YOUR FUTURE

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

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)

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

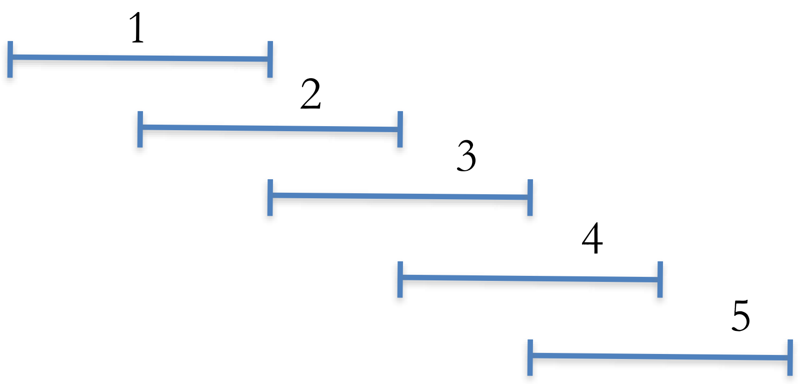
Given OPT(1),…., OPT(j-1), how can one figure out if j in optimal solution or not?

A recursive algorithm

```
Compute-Opt($j$)

If $j = 0$ then return $0$

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

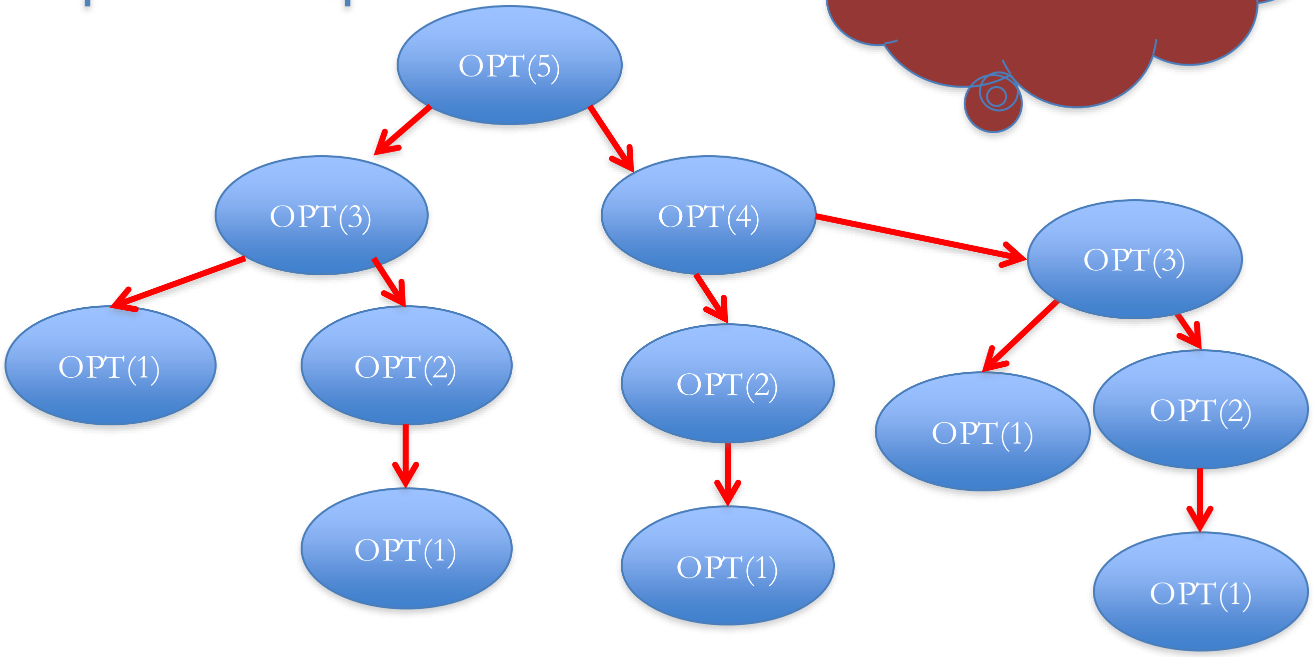
$$\mathrm{OPT}(j) = \max \{ v_j + \mathrm{OPT}(\, p(j)\,), \mathrm{OPT}(j-1) \}$$

Exponential Running Time



p(j) = j-2

Only 5 OPT values!



Formal proof: Ex.

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

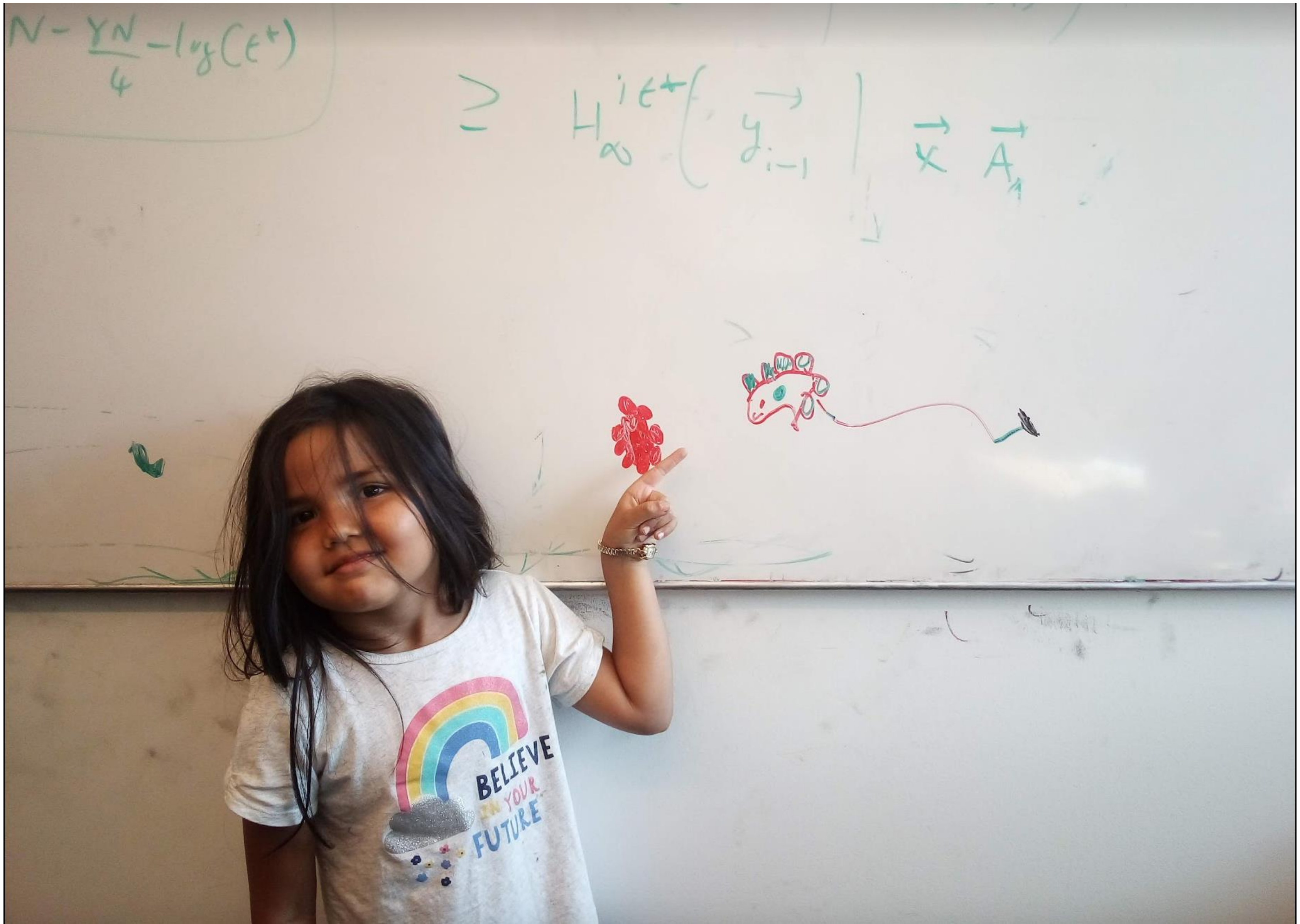
```
M-Compute-Opt(j)

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

Questions/Comments?


$N-\frac{\gamma N}{4}-\log(\epsilon^{+})$
$\geq H_\infty^{i\epsilon^+}(\vec{y}_{i-1} \mid \vec{x}\ \vec{A}_1$
BELIEVE IN YOUR FUTURE

Property of OPT

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

Given $OPT(1), \ldots, OPT(j-1)$, 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)

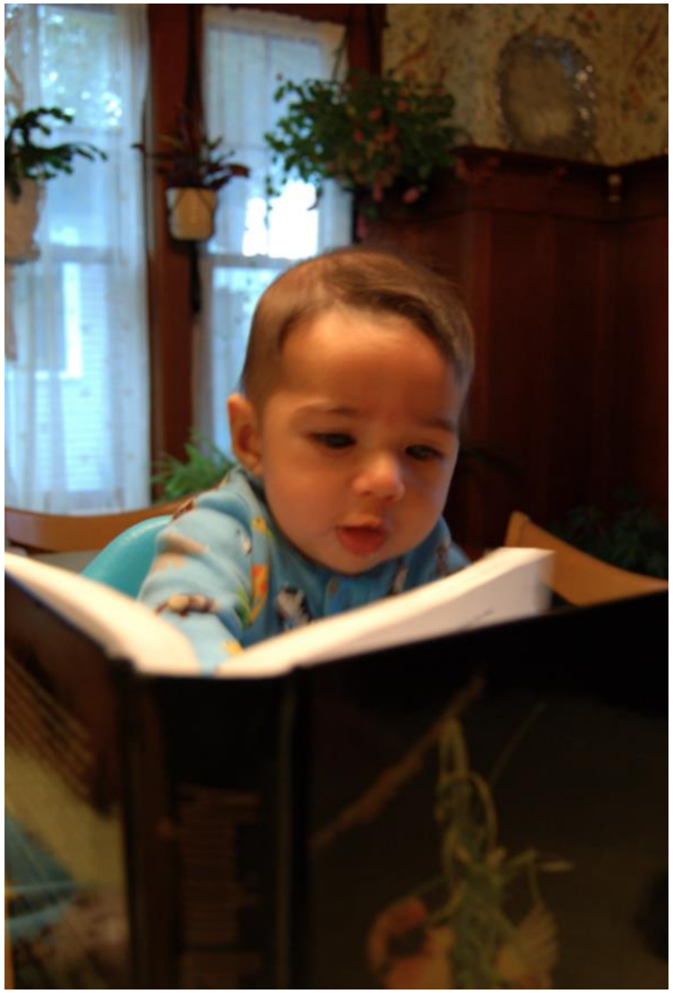
O(n) run time

?
ICANHASCHEEZBURGER.COM

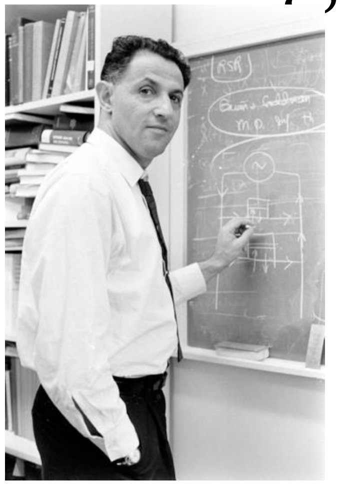
Algo run on the board…

Reading Assignment

Sec 6.1, 6.2 of [KT]



When to use Dynamic Programming


Richard Bellman

There are polynomially many sub-problems

OPT(1), …, OPT(n)

Optimal solution can be computed from solutions to sub-problems

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

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

OPT (j) only depends on OPT(j-1), …, OPT(1)