HW

- part of the engagement grade (30%)

> • Homeworks and programming assignments: Graded on a 10-point scale based on effort and achievement:
>
> ◦ 11-12 points — goes above and beyond (includes optional elements or extra credit)
> ◦ 10 points — solution is complete and correct, or largely so; program meets or largely meets the specifications
> ◦ 7 points — solution/program is generally on the right track but falls a bit short (incomplete and/or some flaws/bugs)
> ◦ 3 points — some effort but solution/program falls well short (very incomplete and/or major flaws/bugs)
> ◦ 0-1 point — generative AI and/or other resources used as a learning cheat
>
> Most assignments that earn 3 or more points on the initial handin can be revised and resubmitted once. The grade for the revised version will replace the original grade and, because understanding and correcting mistakes is a valuable part of learning, an additional point will be earned for a substantive revision effort.

- need at least 3 points on one problem to be eligible for revise-and-resubmit for the entire hw
  - if you scored less than 3 points total or did not hand in anything, see me to discuss revise-and-resubmit
- can revise and resubmit any or all of the problems
  - hand in your original submission along with the revision, especially if you only revise a portion of a problem

HW

- how do you know what to revise?
  - see the written individual and posted feedback
- why should you revise and resubmit?
  - additional practice and to increase your mastery of the material for exams
  - to improve the engagement score
  - a bonus point for substantive revisions
- should you revise and resubmit even if you got a 10?
  - if there's feedback and not just checkmarks, there's something to improve
  - a bonus point for substantive revisions

HW 1

- make sure you know how to use the sums and recurrence relations tables
  - in case you don't know the closed form or another trick for a sum or how to solve a particular recurrence relation
  - $\sum_{i=1}^{n} \log i$
- for #1a, using the definition of Θ is fine but you can also appeal to known principles as justification
  - drop constant factors
  - drop lower order terms
- for #1b, a useful strategy for comparing two functions is eliminating common divisors in order to reduce them to something more familiar
  - e.g. $n^{0.63}$ vs n log n → divide both sides by $n^{0.63}$ → 1 vs $n^{0.37}$ log n

HW1

- for #2, write the sums, then handle each one in turn (show your work!) from the inside out
- be careful to reflect the actual loop

```
for i = 1 to n do
  [loop body]
```

→ $\sum_{i=1}^{n} 1$

each term of the sum reflects the work done in one iteration of the loop

the sum variable *i* corresponds to the loop variable *i*

- handling tricky sums
  - be careful about dropped subtracted terms when it's the sum variable – see if you can do a change-of-variable and rewrite the sum in simpler terms

$\sum_{k=j}^{i+j}(i+j-k)$ → write out the terms to see the pattern – (i+j-j) + (i+j-(j+1)) + (i+j-(j+2)) + … + (i+j-(i+j)) = i + i-1 + i-2 + … + 0 → $\sum_{p=0}^{i} p$

HW1

- for #3, #4 the task is to give the running time for the code given in the problem, not to give a discussion of bubble sort or comb sort in general
  - address the code! “the best case is when the array is in sorted order…” vs “the best case is when swapped is false at the end of the inner loop, so the outer loop only repeats once…”
  - when there’s a different amount of work done in each iteration of a loop, write a sum