CSE 421 Section 4

Divide and Conquer

Administrivia

Announcements & Reminders

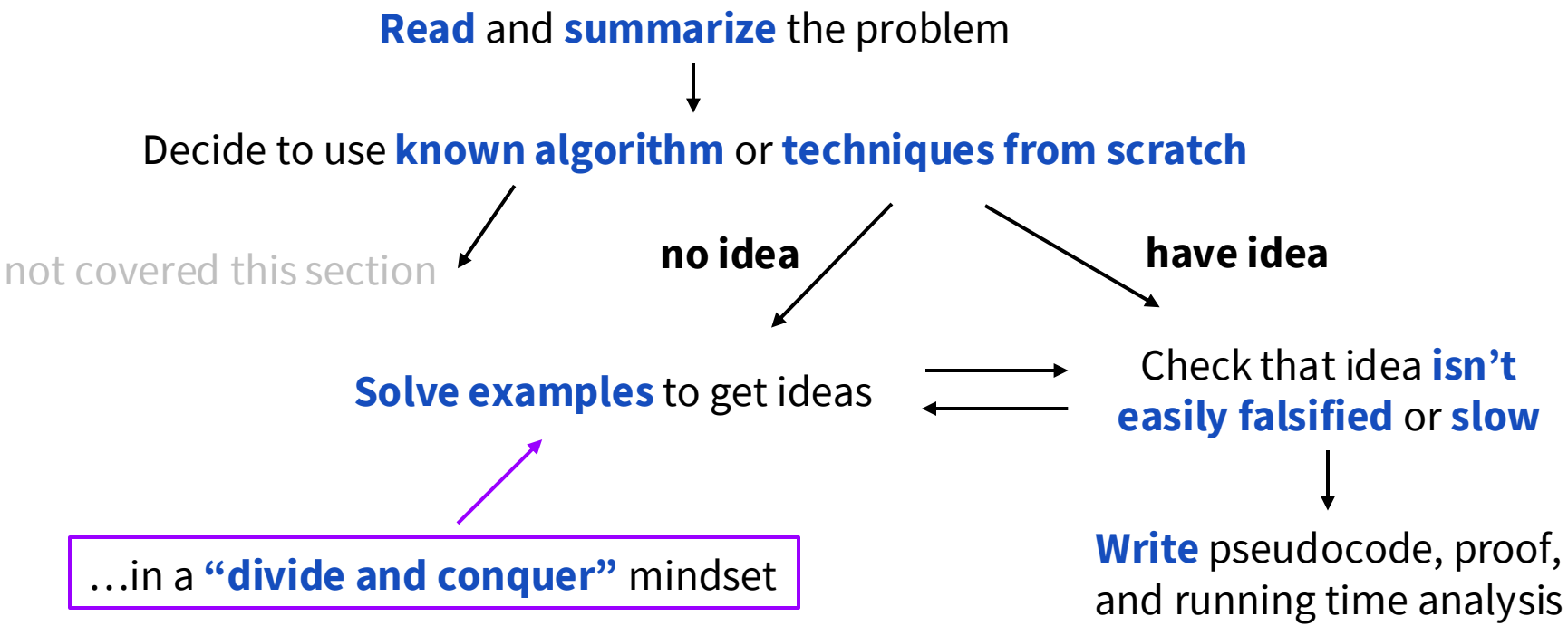
- **HW2**
  - Regrade requests are open
  - Answer keys available on Ed

- **HW3**
  - Was due yesterday, 10/16
  - Remember the **late problems** policy (NOT assignments)
    - Total of up to **10 late problem days**
    - At most **2 late days per problem**

- **HW4**
  - Due Wednesday 10/23 @ 11:59pm

Ideas for divide and conquer

Problem solving strategy overview



Problem 1 – Maximum subarray sum

Input: An array of integers A $= a_1, \ldots, a_n$ (possibly both positive and negative)
Expected output: The largest sum of any contiguous subarray A`[i..j]`

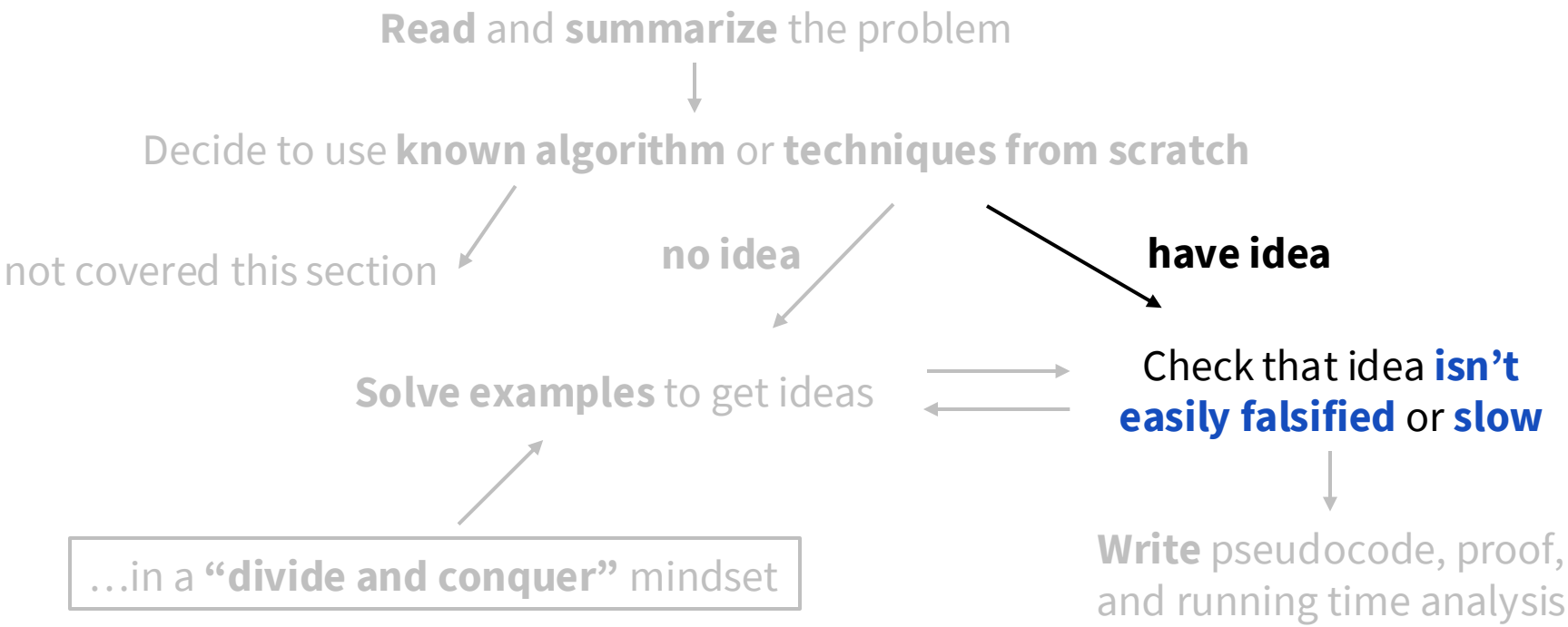
Notation: Denote A`[i..j]` the subarray $a_i, a_{i+1}, \ldots, a_j$.

Notes:

- The list of no elements is a valid subarray (the sum is 0).
- The expected output is the sum of the elements, not the actual subarray.

For divide and conquer word problems: Summary is extremely important, because recursion demands that you understand exactly what the input and output are.

Problem solving strategy overview



Problem 1 – Maximum subarray sum

For problems that can be solved with divide and conquer, there will almost always be **an easy but slow** baseline idea that you can try first.

Input: An array of integers $a_1, \ldots, a_n$ (possibly both positive and negative)
Expected output: The largest sum of any contiguous subarray `A[i..j]`

a) Let's come up with an easy baseline solution (no divide and conquer yet).
   i. What is the simplest idea that you can try? What is the running time?

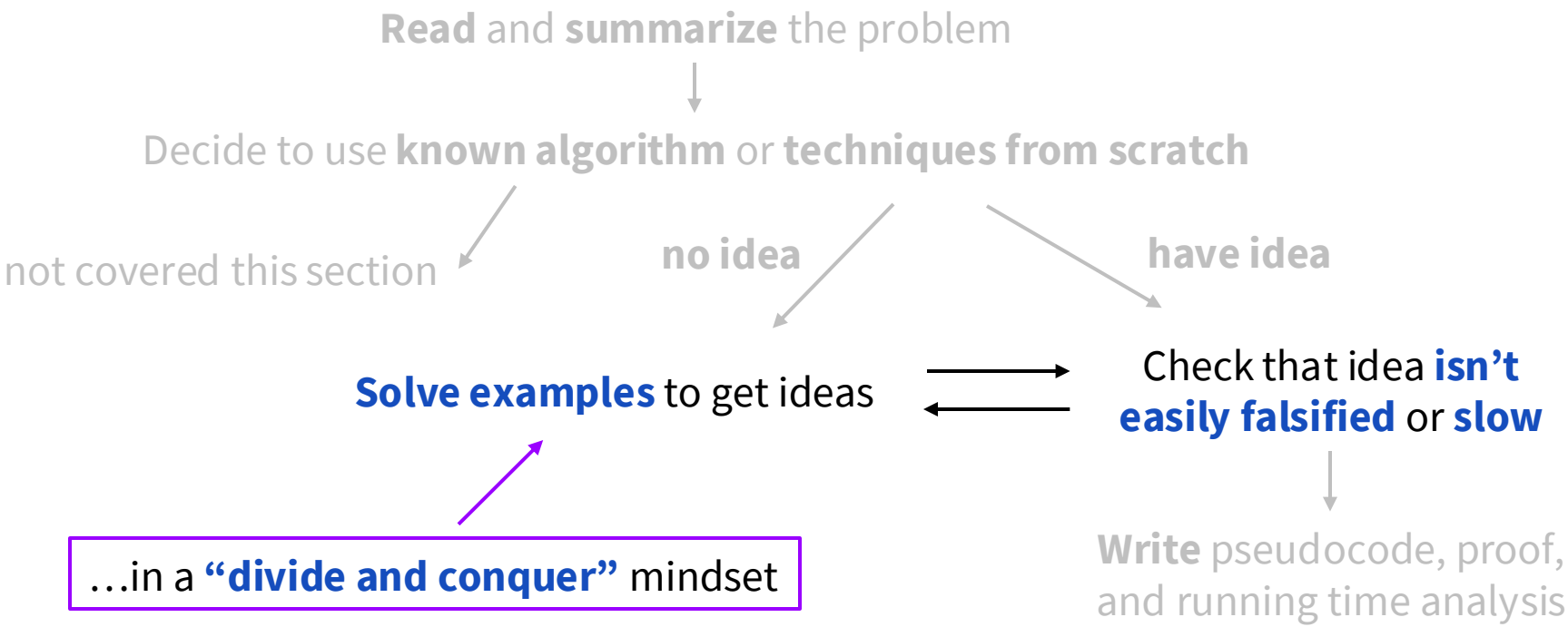
Feel free to work with the people around you!

Problem 1 – Maximum subarray sum

a) Let's come up with an easy baseline solution (no divide and conquer yet).
   i. What is the simplest idea that you can try? What is the running time?

   ii. Are there any inefficiencies with this idea that can be easily fixed (still no divide and conquer)? If so, what is the running time after fixing?

Problem solving strategy overview



Problem 1 – Maximum subarray sum

Now, we know that $O(n^2)$ **is easy**. Thus, we should **aim around** $O(n \log n)$.

b) Here are some basic questions to always ask yourself for divide and conquer:
   i. How do you want to split up the problem?

   ii. What is returned from the recursive calls?

   iii. How much work can you do in each call, in order to get $O(n \log n)$?

Feel free to work with the people around you!

Problem 1 – Maximum subarray sum

c) Solve these examples by hand, as well as the two recursive subproblems in each example (just one level of recursion). Then, think about the following to get ideas: **“How can I use the two answers to the subproblems to get the final answer?”** Remember how much work you are allowed to do.

i. 2, -10, -5, 8, -1, 7

ii. 6, -3, -4, 4, 2, 1, -7, 5

iii. -3, 2, 4, -1, 3, -10, 6, -4

Feel free to work with the people around you!

Continue trying more examples until you have an idea.

Writing about divide and conquer

Divide and conquer pseudocode

Reminders for divide and conquer pseudocode:

- Always **give your function a name**, since you will need to call it recursively.
- In pseudocode, our default will be that function parameters **pass by value**.
  - If you pass arrays by value, you automatically use $O(n)$ time.
  - To achieve sub-$O(n)$, you must use **references, pointers, global variables** (or generally variables scoped outside the function), or other equivalents.
    - These slides use global variables, but it's subjective.
  - Not relevant for this problem since we use $O(n)$ time anyways.

Problem 1 – Maximum subarray sum

d) Write the pseudocode for your solution.

Divide and conquer proofs

Reminders for divide and conquer proofs:

- Always use **strong induction**. Your IH should be:
  "My core function outputs its expected output for all inputs of size $\leq k$."
- The **structure can be inspired by your code**, which already has a "base case" and "recursive (inductive) step".
  - Also, if your code branches on anything (if, max, min, etc.), your proof should have **cases based on what kinds of inputs end up at each branch**.
- You should explain:
  - Why your output is the expected output, AND
  - If the input is "X such that Y holds", explain why Y holds for recursive calls.

Problem 1 – Maximum subarray sum

e) Write the proof that your pseudocode works.

Problem 1 – Maximum subarray sum

f) Analyze the running time of your code by solving a recurrence.

Final thoughts

- How to choose between divide and conquer vs. greedy?
  - Try easy algorithms first, like baselines or greedy.
  - If easy ones are slow and subproblems seem useful, try divide and conquer.
- Sometimes, it will be useful to **compute more than what's asked for**.
  - Examples:
    - Problem 2 in your section packet
    - Problem 2 on your homework: today's problem in $O(n)$! It will guide you.
  - In this case, your **IH should reflect what you actually compute**, not what you were asked to compute.
  - **Try the usual thing first**, only compute more if it doesn't work/is too slow.

Summary

- First, try an easy but slow **baseline** algorithm.
  - Use this to estimate how much time you can take per recursive call, in order to still get an improvement.
- Ask yourself: **How can I use answers to subproblems to find the full answer?**
- Keep in mind the cost of copying arrays, and avoid this with global variables.
- Prove using **strong induction**.

Thanks for coming to section this week!