S.KIMMEL

Groups

Eliza	Zeke	Peter	Lillie
Elijah	Jackson	Alderik	Jacqueline
Nadani	Bruce	Henry	Ben

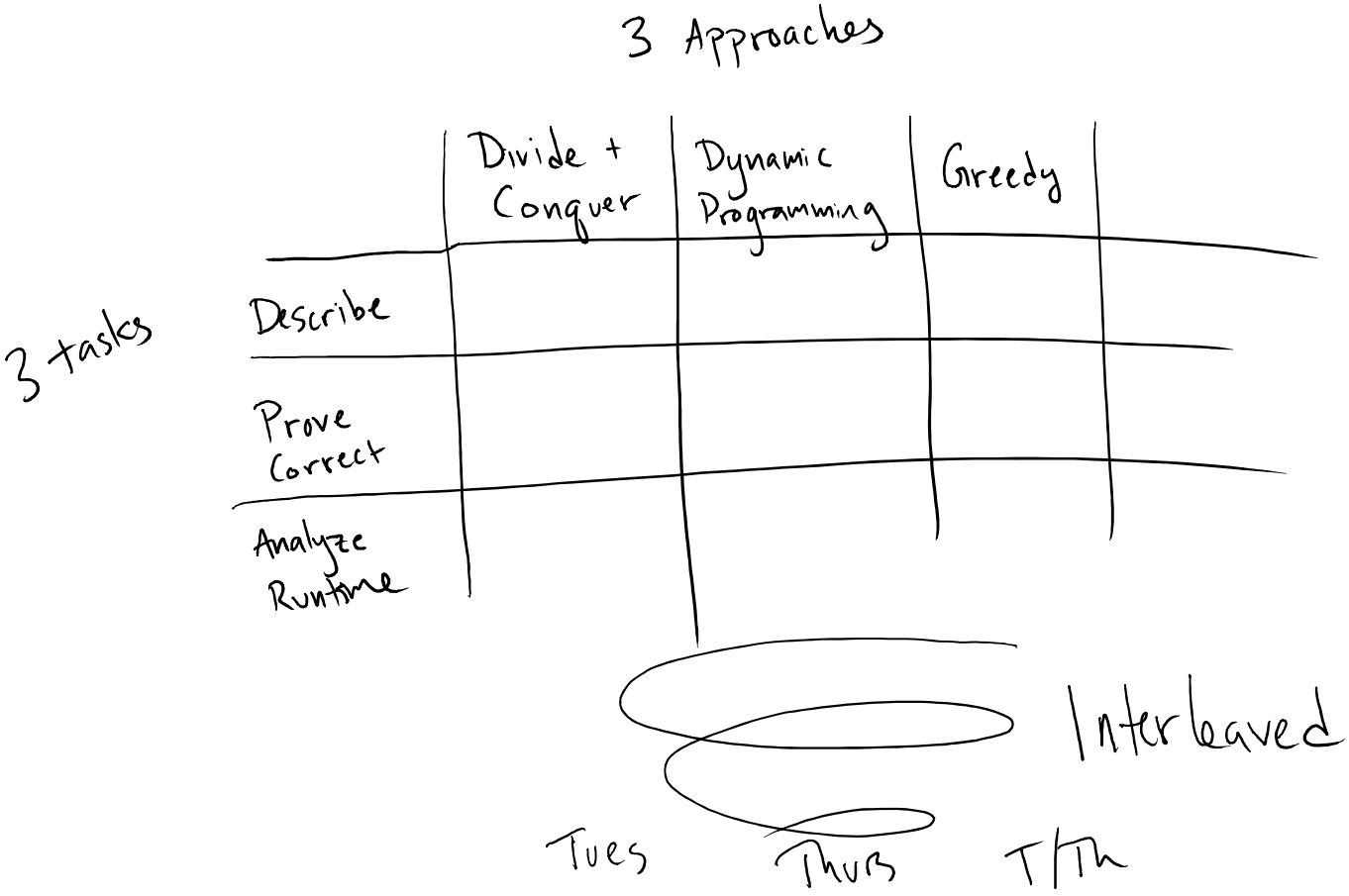
Michael	Graham	Christian
Tommaso	Alex	Pierce
Nick	Kai	Chris

Will	Farhan	Caroline
Zachary	Joonwoo	Angel
Dylan	Tenzin	Bryan

S. KIMMEL

Outline

Part 1: Problems Computers Can Solve



Part 2: Problems Computers Can't Solve (well) (as far as we know)

NP-completeness

Q: When I use a computer/phone in class, it contributes to my learning in that class

A. Most of the time

B. Some of the time

C. I really just use it to check Instagram + e-mail

D. I don't use a computer/phone in class.

S.KIMMEL

Divide & Conquer (& Combine)

- Split big problem into smaller versions of same problem.
- Solve smaller problems via recursion
- combine smaller solutions to get big solution

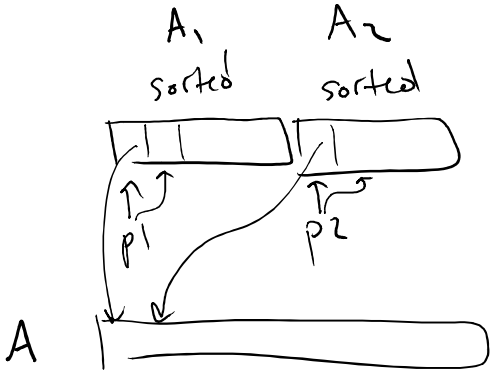
Already Seen!

Merge Sort

Input: Array A of integers of size n
Output: Sorted array

```
MergeSort(A)
    if length(A)==1 then return A          } Base Case
    A1 = MergeSort(A[1 : n/2])             } Divide +
    A2 = MergeSort(A[n/2+1 : n])             Conquer
    p1 = p2 = 1

    for i = 1 to n                         } Combine
        if A1[p1] < A2[p2]
            A[i] = A1[p1]
            p1++
        else
            A[i] = A2[p2]
            p2++
    return A
```



S. KIMMEL

Divide + Conquer

- Description:
  - Base Case
  - Recursion
  - Loops / Conditionals

  } Divide + Conquer

  } Combine

- Correctness:
  - (Strong) Inductive Proof
  - Proof by cases
  - Loop Invariants — New

- Time Complexity: Recurrence relation + master method — Might be new

For rest, we'll do small review, but I'm assuming familiarity

(See PS1 @ go/cs200 for standard inductive proof review.)

S.KIMMEL

Q: Which of the following is a correct recurrence relation for Merge Sort?

A) $T(n) = T\left(\frac{n}{2}\right) + O(n)$

B) $T(n) = T(n) + O\left(\frac{n}{2}\right)$

C) $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

D) None of the above

S.KIMMEL

Q: Which of the following is a correct recurrence relation for Merge Sort?

A) $T(n) = T\left(\frac{n}{2}\right) + O(n)$

B) $T(n) = T(n) + O\left(\frac{n}{2}\right)$

C) $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$ ⟸ Partly OK, need $T(1) = O(1)$

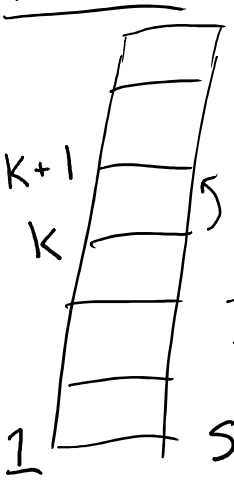
D) None of the above ⟸ Technically Correct

All are missing "base case"

S. KIMMEL

Review of Induction (+ application to Algorithm correctness)

Induction



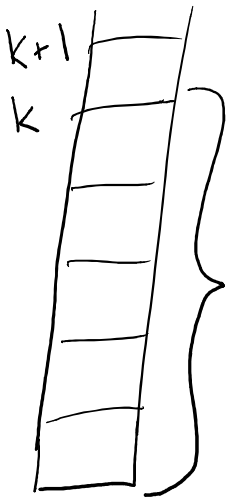
Step 2: Show how to get from rung K to K+1

Step 1: Show how to get on ladder

- Useful when problem size decreases by 1 in recursive call

Better for Divide & Conquer:

Strong Induction



Step 2: Assume all rungs from 1 to K are true, use to get to K+1

Step 1: Show how to get on ladder

- If problem size in recursive call is smaller than original input, can assume output is correct

by any amount, as long as greater than or equal to base case size

S.KIMMEL

Q: Start a Strong Inductive Proof of Correctness of Merge Sort

A: Let P(n) be the predicate that Merge Sort works correctly on arrays of size n. We will prove P(n) is true for all $n \geq 1$ using strong induction.

Base Case: P(1) is true because if the array is size 1, it is already sorted, so the algorithm returns it, which is correct.

Inductive Step: let $k \geq 1$. Assume for strong induction P(j) is true for all $j : 1 \leq j \leq k$. Now consider an input of size k+1. Since $k+1 > 1$, the algorithm goes to the recursive step. It applies MergeSort to the first + second half of the array. Since each half is smaller than k+1, but at least 1 in length, by inductive assumption, the output of these calls are sorted arrays.

... Need Loop Invariants for Rest