LECTURE 4:

DIVIDE & CONQUER

GRADUATE ALGORITHMS

ANNOUNCEMENTS

- Homework 1
- Submit PDF version – LaTex or Markdown->PDF
- **Due:** two thursdays from now

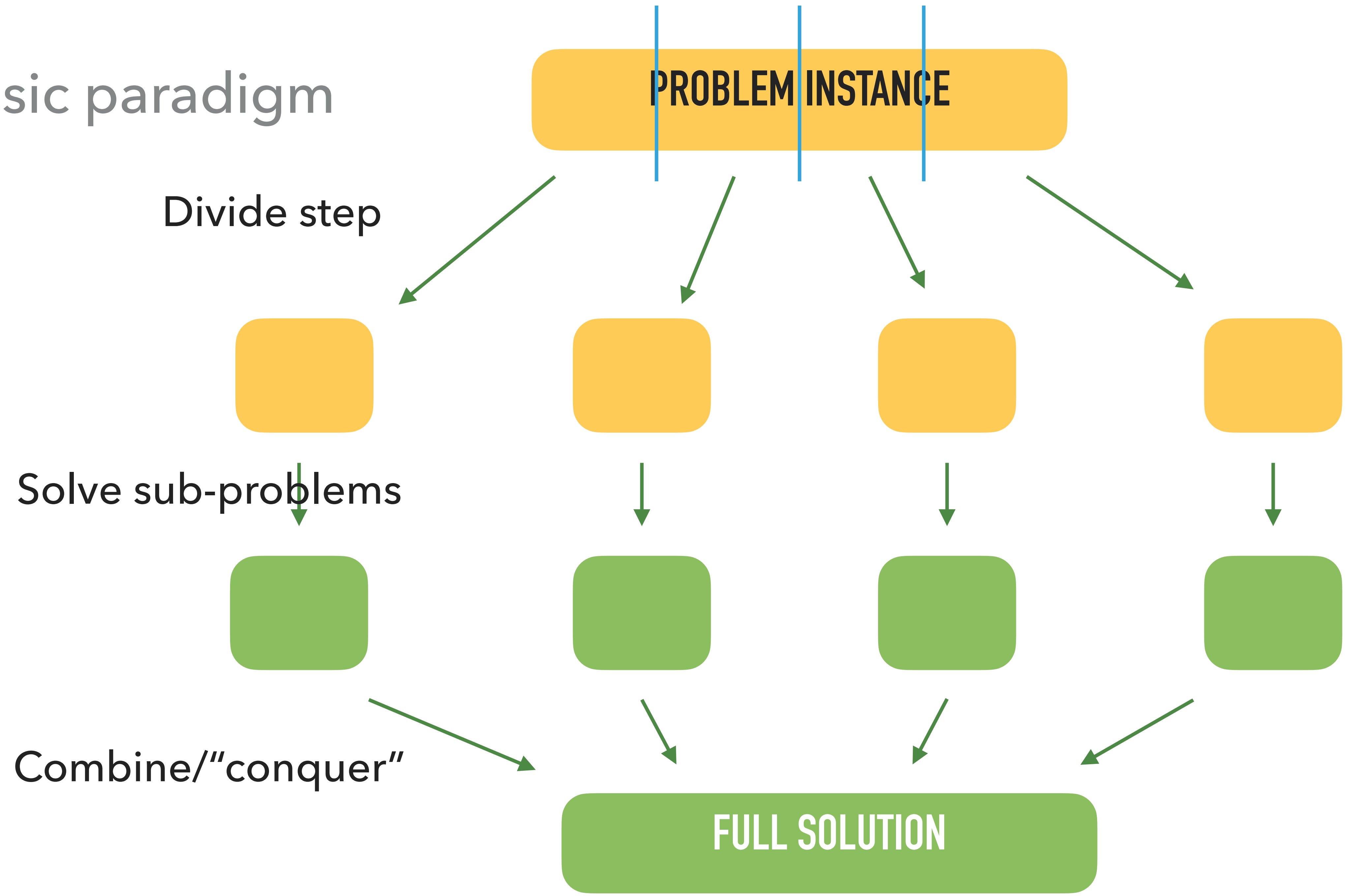
LAST CLASS: DATA STRUCTURES

- What is being stored?
- What operations need to be performed?
- Running time for each operation?
- How much memory used overall? (remember graphs)

Like a class' API; often we have trade-offs between these terms

LAST CLASS: DIVIDE AND CONQUER

- Classic paradigm



DIVIDE & CONQUER BASICS

- Useful when problem “cleanly” divides into sub-problems

- Analysis template: correctness by induction, complexity using “recurrences”

- Algorithmic analog of mathematical induction

EXAMPLE: MERGE SORT

Given an array A[0, 1, …, N-1] of distinct integers, place them in increasing order

- Partition into two sub-arrays B, C
- Sort recursively
- Merge arrays into A

Procedure MergeSort(A):

if length(A) ≤ 2, do brute force -- go over array and swap if necessary

partition A into B and C of size $(1/2)$ length(A)

recursively sort B and C

Merge B and C into A

Procedure Merge(B, C, A):

denote length(A) by n

maintain two indices $i_b = i_c = 0$

for $i = 0, \ldots, n-1$:

write the smaller of $B[i_b]$ and $C[i_c]$ to $A[i]$ and increment the corresponding index (if index goes out of bounds, treat value as ∞)

CORRECTNESS

- Induction: base-case, inductive step
  - **Standard math:** (a) prove statement for n=1, (b) assuming statement holds for integers r < n, show that it holds for n
  - **Divide & conquer:** (a) procedure behaves correctly in base case, (b) combination produces right answer for full problem, assuming right answer for sub-problems

CORRECTNESS — MERGE PROCESS WORKS CORRECTLY!

RUNNING TIME

- Partition into two sub-arrays B, C
- Sort recursively
- Merge arrays into A

"SOLVING" RECURRENCES

- Semi-general methods
  - Master theorem
  - Akra-Bazzi theorem
- Recursion "tree"
- Plug-n-chug
- Guess and prove

EXAMPLE: SEARCHING IN A SORTED ARRAY

Given an array A[0, 1, …, N-1], integers in increasing order, find if a query 'x' is present in A[]

CORRECTNESS

(Even if something seems obvious, formalize why)

RUNNING TIME

CAN ONE DO BETTER THAN LOG_2 N?

Given an array A[0, 1, …, N-1], integers in increasing order, find if a query ‘x’ is present in A[]

- Can partitioning into groups > 2 help?
- Query model
- More generally, computational model

EXAMPLE: LONG MULTIPLICATION

$A = a_1\ a_2 \dots a_n$

$B = b_1\ b_2 \dots b_n$

Find the product AB

- Isn't multiplication constant time?
- When would we multiply really long numbers?

STANDARD ALGORITHM

MULTIPLICATION

Each intermediate sum is shifted left!

```
        1932
     x  2142
     -------
        3864   (1932 x 2)
  +    7728    (1932 x 4)
      1932     (1932 x 1)
     3864      (1932 x 2)
     -------
     4138344
```

DIVIDE AND CONQUER?

RUNNING TIME

CAN WE DO BETTER?

- Reason for hope: we need to compute only three terms

  $$XZ, (XW + YZ), YW$$

- Can we do using three multiplications?

THREE MULTIPLICATIONS