Analyzing Running Time (Chapter 2)

- What is efficiency?
- Tools: asymptotic growth of functions
- Practice finding asymptotic running time of algorithms

First: a bit of sorting

- Which of these grows faster?
  $n^{4/3}$
  $n(\log n)^3$
- Example(s) on board

Bean Counting 101: Analyze Gale-Shapley

```
Initialize each college and student to be free.
while (some college is free and hasn't made
offers to every student) {
   Choose such a college c
   s = 1st student on c's list to whom c has not
     made offer
   if (s is free)
      assign c and s to be engaged
   else if (s prefers c to current college c')
      assign c and s to be engaged, and c' to be
      free
   else
      s rejects c
}
```

Bean Counting

- Count how many lines of code execute
- Be wary of pseudocode: make sure each line is really $O(1)$
  - May need to think carefully about data structures to determine if this is the case

Four Patterns

- For Loops
- Accounting
- Enumeration
- Divide-and-(maybe)-conquer

Pattern 1: For Loops

Compute the maximum

```
max = a_1
for i = 2 to n  {
    if (a_i > max)
        max = a_i
}
```

$O(n)$

```
for i = 2 to n {
    for j = 2 to n {
        // constant time
        // operations
}
```

$O(n^2)$

Pattern 2: Accounting

- For while() loops or more complex constructions, may not be obvious how many times a line of code executes
- Use accounting scheme to count executions

Accounting: Gale-Shapley

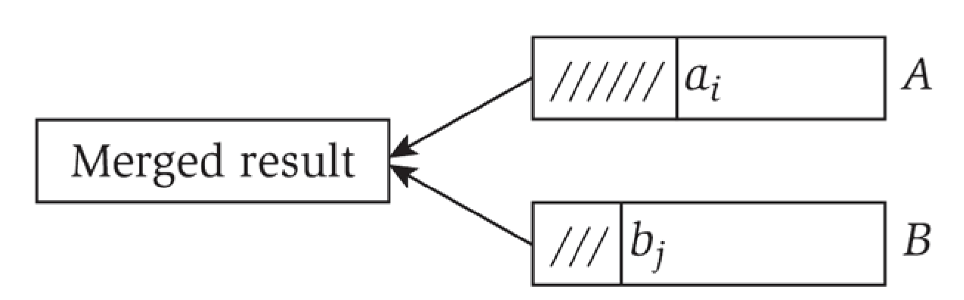
```
Initialize each college and student to be free
while (some college is free and ...) {
    //
    // etc., etc.
    //
}
```

Each loop execution makes a new offer. Charge each loop to an offer!

--> at most n^2 times through loop

Accounting: Merge Sorted Lists

- Input: sorted lists $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_n$
- Output: combined sorted list



Accounting: Merge Two Sorted Lists

```
i = 1, j = 1
while (both lists are nonempty) {
    if (ai ≤ bj) {
      append ai to output list
      increment i
    }
    else {
      append bj to output list
      increment j
    }
}
append remainder of nonempty list
to output list
```

Accounting scheme?

Pattern 3: Enumeration

- Brute force solution: examine all possibilities
- Running time will depend on the structure of the problem. How many possible answers are there?
- (Seems ugly, but sometimes the best we can do!)

Closest Points

- Closest pair of points in a plane
  Given a list of n points in the plane (x_1, y_1), ..., (x_n, y_n), find the pair that is closest.

```
min = infinity
for i = 1 to n {
    for j = i+1 to n {
        d = (x_i - x_j)^2 + (y_i - y_j)^2
        if (d < min)
            min = d
    }
}
```

$O(n^2)$

More Enumeration

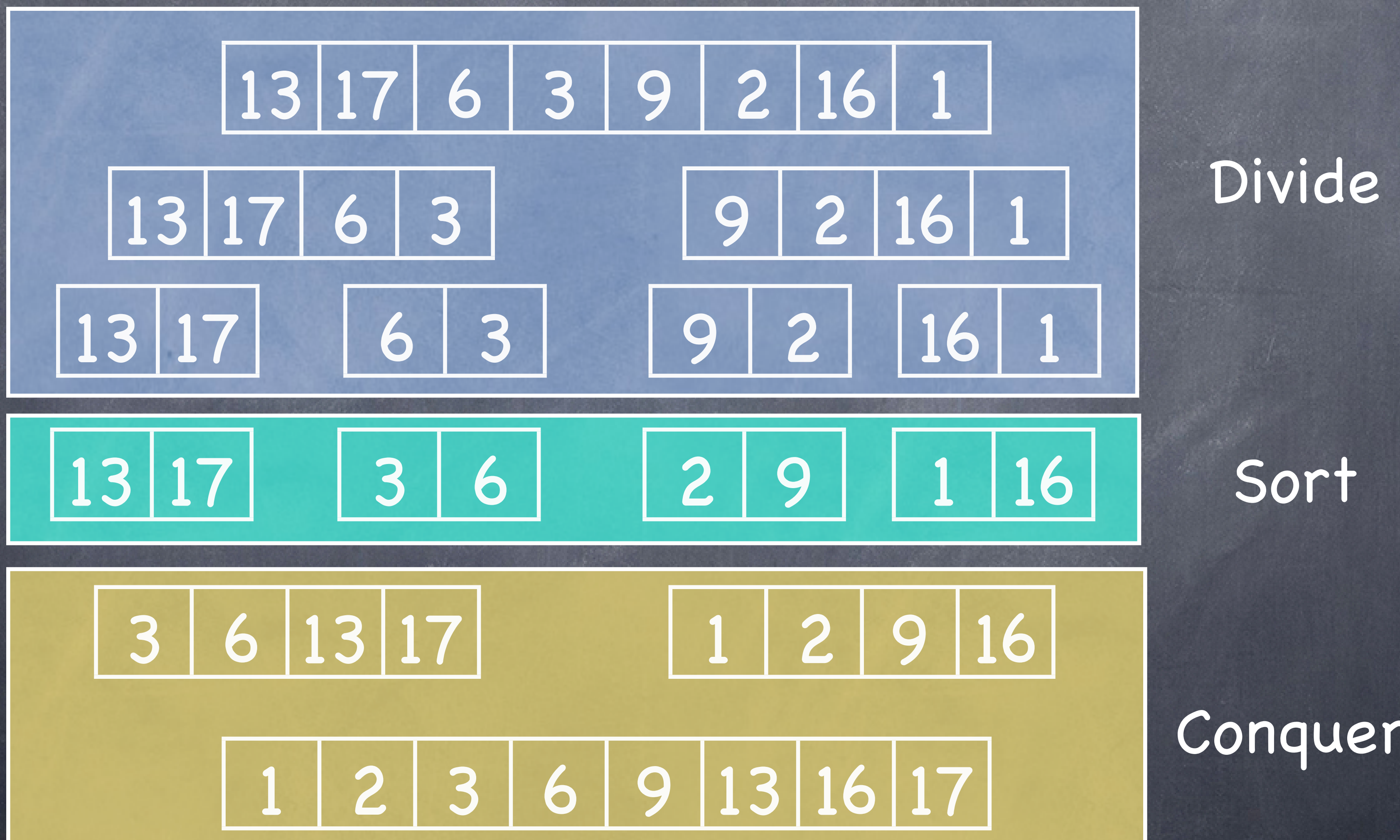
Examine all _____ of n items

- Pairs - $O(n^2)$
- Triples - $O(n^3)$
- Subsets of size k - $O(n^k)$
- Subsets of any size - $O(2^n)$
- Permutations - $O(n!)$

Pattern 4: Divide-and-(maybe)-conquer

- O(log n) "logarithmic time". Do a constant amount of work to discard a constant fraction of the input (often 1/2)
  - Binary search (illustrate on board)
- O(n log n). Divide-and-conquer (much more later in course)
  - Mergesort

Mergesort



13 17 6 3 9 2 16 1
13 17 6 3
9 2 16 1
13 17
6 3
9 2
16 1
Divide
13 17
3 6
2 9
1 16
Sort
3 6 13 17
1 2 9 16
1 2 3 6 9 13 16 17
Conquer

Summary

- Bean counting: count each line, be careful of pseudocode
- Patterns: for loops, accounting, enumeration, divide-and-maybe-conquer
- Common running times
  - $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(2^n)$, $O(n!)$