Administration

- there is a mid-term next week (oct 7)
- one double-sided cheat-sheet
- topics: up to yesterday’s lecture (greedy)
DIVIDE-AND-CONQUER ALGORITHMS
**Mergesort**

**PROBLEM.** sort a list of $n$ numbers $a_1, \ldots, a_n$.

**ALGORITHM.** divide-and-conquer

1. **divide.** divide $a$ into two pieces: $a[:\text{mid}]$ and $a[\text{mid}:]$
2. **conquer.** recursively sort each half
3. **combine.** merge two sorted halves into a single sorted list

```python
def mergesort(a):
    if len(a) <= 1:
        return a
    else:
        mid = len(a)/2
        return merge(mergesort(a[:mid]), mergesort(a[mid:]))
```

**RUNNING TIME.** $T(n) = 2T(n/2) + O(n)$
Our first recurrence

PROBLEM. solve for $T(n)$, given that for some constant $c$

$$T(2) \leq c$$

$$\forall n > 2, \quad T(n) \leq 2T(n/2) + cn$$

1ST TECHNIQUE. “unroll” the recurrence

- see Textbook, chapter 5.1

2ND TECHNIQUE. guess and check

- guess $T(n) \leq cn \log_2 n$ for all $n \geq 2$

- prove by induction
PROBLEM 2. solve for $T(n)$, given that for some constant $c$

\[ T(2) \leq c \]

\[ \forall n > 2, \quad T(n) \leq T(n/2) + cn \]

1ST TECHNIQUE. “unroll” the recurrence

\[ T(n) \leq T(n/2) + cn \]

\[ \leq (T(n/4) + cn/2) + cn \]

\[ \leq \cdots + cn/8 + cn/4 + cn/2 + cn \]

\[ \leq 2cn \]
More recurrences

1. \( T(n) = qT(n/2) + O(n) \)
   - \( q = 1 \): \( T(n) = O(n) \)
   - \( q = 2 \): \( T(n) = O(n \log n) \)
   - \( q > 2 \): \( T(n) = O(n^{\log_2 q}) \)

2. \( T(n) = 2T(n/2) + O(n^2) \)
   - \( T(n) = O(n^2) \)
Recommendation systems

**GOAL.** match your preferences with those of others on the Internet.

- c.f. amazon (books), netflix (movies)
- preferences are given as rankings
- want to identify (cluster) people with similar “tastes”

**QUESTION.** how to compare rankings?
**PROBLEM.** count the number of inversions in a list of \( n \) numbers \( a_1, \ldots, a_n \).

- two indices \( i < j \) form an inversion if \( a_j > a_i \) (i.e., \( a_i, a_j \) are “out of order”)
- e.g. \([2, 1, 4, 3, 6, 5]\) has 3 inversions, \([4, 5, 6, 1, 2, 3]\) has 9 inversions
- between 0 (complete agreement) and \( n(n-1)/2 \) (complete disagreement)

**RANKINGS.**

- label movies from 1 to \( n \) according to your ranking
- order these labels according to stranger’s ranking
- count the number of inversions

**NAIVE ALGORITHM.** check all pairs, \( O(n^2) \) time

**DIVIDE-AND-CONQUER.** aiming for \( O(n \log n) \) time

1. divide. divide a into two pieces: \( a[:\text{mid}] \) and \( a[\text{mid}:] \)
2. conquer. recursively compute inversions in each half
3. combine. count inversions where \( a_i, a_j \) are in different halves, and return sum of 3 quantities

**EXAMPLE:** \([2, 1, 4, 3, 6, 5]\) : \( 1 + 1 + 1 \), \([4, 5, 6, 1, 2, 3]\) : \( 0 + 0 + 9 \)
Counting inversions

**PROBLEM.** count the number of *inversions* in a list of \( n \) numbers \( a_1, \ldots, a_n \).

**DIVIDE-AND-CONQUER ALGORITHM.** (first attempt)

```python
def invcount(a):
    if len(a) <= 1: return 0
    else:
        mid = len(a)/2
        return invcount(a[:mid]) ## left, left
        + invcount(a[mid:]) ## right, right
        + crosscount(a[:mid], a[mid:]) ## left, right
```

**REMARKS.**

- want `crosscount` to run in \( O(n) \) time like merge
- easier if we also sort the two halves \( a[:\text{mid}], a[\text{mid}:] \)
- next: `sortcount` returns number of inversions and sorted list
Counting inversions

**Problem.** Count the number of inversions in a list of $n$ numbers $a_1, \ldots, a_n$.

**Divide-and-Conquer Algorithm.**

```python
def sortcount(a):
    if len(a) <= 1: return (0, a)
    else:
        mid = len(a)/2
        (left, ans1) = sortcount(a[:mid])
        (right, ans2) = sortcount(a[mid:])
        (ans3, sorted) = mergecount(left, right)
        return (ans1 + ans2 + ans3, sorted)
```

**New Problem.** Implementing `mergecount(A, B)` in $O(n)$ time

- Given two sorted lists $A$, $B$
- Produce a single sorted list
- Count the number of “inverted pairs” $(a, b)$ where $a \in A$, $b \in B$, $a > b$
NEW PROBLEM. implementing \texttt{mergecount}(A, B) in $O(n)$ time

- given two sorted lists $A$, $B$
- count the number of “inverted pairs” $(a, b)$ where $a \in A$, $b \in B$, $a > b$

WARM-UP. similar to \texttt{merge}

- if $A[0] < B[0]$: peel off $A[0]$, contributes 0 inverted pairs
- if $A[0] > B[0]$: peel off $B[0]$, contributes $\text{len}(a)$ inverted pairs
THE END