Week 6 - Friday

COMP 4500

Last time

- What did we talk about last time?
- Recurrence relations

Questions?

Assignment 3

Logical warmup

- A man offers you a bet
- He shows you three cards
  - One is red on both sides
  - One is green on both sides
  - One is red on one side and green on the other

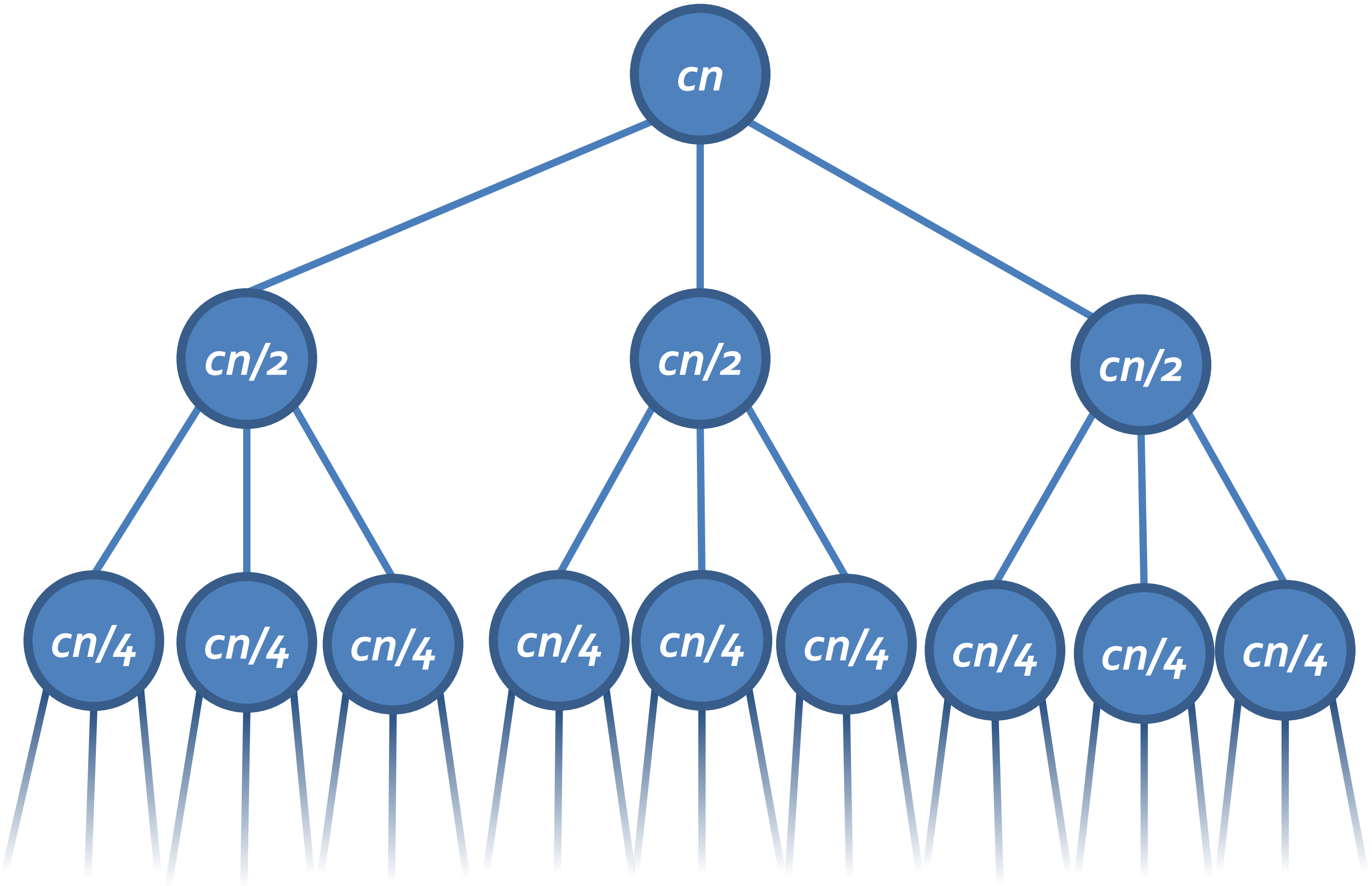
- He will put one of the cards, at random, on the table, with a random side up
- If you can guess the color on the other side, you win
- If you bet $100
  - You gain $60 on a win
  - You lose your $100 on a loss
- Should you take the bet? Why or why not?

Further Recurrence Relations

Further recurrence relations

- We have seen that recurrence relations of the form $T(n) \leq 2T\left(\frac{n}{2}\right) + cn$ are bounded by O(***n*** log ***n***)
- What about $T(n) \leq qT\left(\frac{n}{2}\right) + cn$ where ***q*** is bigger than 2 (more than two sub-problems)?
- There will still be $\log_2$***n*** levels of recursion
- However, there will not be a consistent ***cn*** amount of work at each level

Consider $q = 3$



cn

$(3/2)cn$

$(9/4)cn$

Converting to summation

- For $\boldsymbol{q}$ = 3, it's $T(n) \leq \sum_{j=0}^{\log_2 n-1} \left(\frac{3}{2}\right)^j cn$
- In general, it's

$$T(n) \leq \sum_{j=0}^{\log_2 n-1} \left(\frac{q}{2}\right)^j cn = cn \sum_{j=0}^{\log_2 n-1} \left(\frac{q}{2}\right)^j$$

- This is a geometric series, where $r = \frac{q}{2}$

$$T(n) \leq cn \left(\frac{r^{\log_2 n} - 1}{r - 1}\right) \leq cn \left(\frac{r^{\log_2 n}}{r - 1}\right)$$

Final bound

$$T(n) \leq cn\left(\frac{r^{\log_2 n} - 1}{r - 1}\right) \leq cn\left(\frac{r^{\log_2 n}}{r - 1}\right)$$

- Since $r - 1$ is a constant, we can pull it out
- $T(n) \leq \left(\frac{c}{r-1}\right) nr^{\log_2 n}$
- For $a > 1$ and $b > 1$, $a^{\log b} = b^{\log a}$, thus $r^{\log_2 n} = n^{\log_2 r} = n^{\log_2(q/2)} = n^{(\log_2 q)-1}$
- $T(n) \leq \left(\frac{c}{r-1}\right) n \cdot n^{(\log_2 q)-1} \leq \left(\frac{c}{r-1}\right) n^{\log_2 q}$ which is $O\left(n^{\log_2 q}\right)$

What about a single sub-problem?

- We will still have $\log_2 n - 1$ levels
- However, we'll cut our work in half each time

$$T(n) \leq T\left(\frac{n}{2}\right) + cn \leq \sum_{j=0}^{\log_2 n - 1} \left(\frac{1}{2}\right)^j cn = cn \sum_{j=0}^{\log_2 n - 1} \frac{1}{2^j}$$

- Summing all the way to infinity, $1 + \frac{1}{2} + \frac{1}{4} + \cdots = 2$
- Thus, $T(n) \leq 2cn$ which is $O(n)$

What might that look like in code?

- Here's a non-recursive version in Java

```
int counter = 0;
for( int i = 1; i <= n; i *= 2 )
    for( int j = 1; j <= i; j++ )
        counter++;
```

- We've just shown that this is O(***n***), in spite of the two `for` loops

Three-Sentence Summary of Counting Inversions

Counting Inversions

Rankings

- Let's say that you like the following 2024 Oscar nominees in this order:

1. *American Fiction*
2. *Barbie*
3. *Oppenheimer*
4. *Poor Things*

- The correct ordering is:

1. *Barbie*
2. *Poor Things*
3. *Oppenheimer*
4. *American Fiction*

Ranking similarity

- What if we wanted to measure the similarity of your ranking to the given ranking?
- **Inversions** are pairs of elements that are out of order in one ranking with respect to the other
- Formally, for indices $i < j$, there's an inversion if ranking $r_i > r_j$

Minimum and maximum inversions

- If two rankings are the same, they would have no inversions
- If two rankings were sorted in opposite directions, they would have ***n*** – 1 inversions for the first element, ***n*** – 2 inversions for the second element, ***n*** – 3 inversions for the third …

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

Visualization of inversions

- You can visualize inversions as the number of line segments crossings if you match up items in one list with the other

1. *American Fiction*	1. *Barbie*
2. *Barbie*	2. *Poor Things*
3. *Oppenheimer*	3. *Oppenheimer*
4. *Poor Things*	4. *American Fiction*

- A total of 4 inversions

Getting the counting right

- Since we're dealing with two different orders, it's sometimes hard to understand what we're supposed to be counting

1. *American Fiction*	Other rank: 4
2. *Barbie*	Other rank: 1
3. *Oppenheimer*	Other rank: 3
4. *Poor Things*	Other rank: 2

- *American Fiction* contributes 3 (because 4 is bigger than 1, 3, and 2)
- *Barbie* contributes 0
- *Oppenheimer* contributes 1 (because 3 is bigger than 2)
- The last one always contributes nothing

Example

- Consider the following items whose rankings from another list are given
  - 9
  - 4
  - 2
  - 6
  - 1
  - 8
  - 7
  - 10
  - 3
  - 5
- Count the inversions

Why do we care?

- The process of **collaborative filtering** tries to match preferences of different people on the Internet
- If your preferences are similar to someone else's, Netflix can recommend shows that they liked
- Counting inversions is just one way to measure similarity between preferences

Algorithm design

- Given a list of rankings, it's easy to count how many rankings are out of order with respect to the rankings that come after them

```
int inversions = 0;
for (int i = 0; i < n - 1; ++i)
    for (int j = i + 1; j < n; ++j)
        if( rankings[i] > rankings[j] )
            ++inversions;
```

- What's the problem with this algorithm?
- It's $O(n^2)$

Can we do better?

- Of course!
- We can borrow from the Mergesort algorithm
- Divide the problem in half
- Then, we will get the number of inversions in the first half and in the second half
- Are we done?
  - No, we also have to count the inversions between the first half and the second half
  - Those are exactly those elements in the first half that are bigger than elements from the second half
  - We can find those during the merge process

Merge-and-Count(*A*, *B*)

- Maintain a ***Current*** pointer into each list, initialized to point to the front elements
- Set ***Count*** = 0
- While both lists have elements
  - Let $\boldsymbol{a_i}$ and $\boldsymbol{b_j}$ be the elements pointed to by the ***Current*** pointer
  - Append the smaller one to the output list
  - If $\boldsymbol{b_j}$ is smaller then
    - Increment ***Count*** by the number of elements left in ***A***
  - Advance the ***Current*** pointer in the list that had the smaller element

Sort-and-Count(*L*)

- If the list has one element then
  - Return 0 inversions and the list ***L***
- Else
  - Divide the list into two halves:
    - ***A*** has the first $\left\lceil \frac{n}{2} \right\rceil$ elements
    - ***B*** has the remaining $\left\lfloor \frac{n}{2} \right\rfloor$ elements
  - ($\boldsymbol{inversions_A}$, ***A***) = Sort-and-Count(***A***)
  - ($\boldsymbol{inversions_B}$, ***B***) = Sort-and-Count(***B***)
  - (***inversions***, ***L***) = Merge-and-Count(***A***, ***B***)
  - Return ***inversions*** + $\boldsymbol{inversions_A}$ + $\boldsymbol{inversions_B}$ and sorted list ***L***

Running time

- Since Merge-and-Count is bounded by O(***n***), the running time for Sort-and-Count is clearly:
  - $T(1) \leq c$
  - $T(n) \leq 2T\left(\frac{n}{2}\right) + cn$, for $n \geq 2$
- By the same analysis as for Mergesort, ***T***(***n***) is O(***n*** log ***n***)

Upcoming

Next time…

- Closest pair of points

Reminders

- **Assignment 3 is due tonight by midnight**
- Read section 5.4
- Extra credit opportunities (0.5% each):
  - Hristov teaching demo: 2/19 11:30-12:25 a.m. in Point 113
  - Hristov research talk: 2/19 4:30-5:30 p.m. in Point 139