Lecture 24

CSE 331

Oct 29, 2021

Please have a face mask on

Masking requirement

UR requires all students, employees and visitors – regardless of their vaccination status – to wear face coverings while inside campus buildings.

https://www.buffalo.edu/coronavirus/health-and-safety/health-safety-guidelines.html

Coding P1 due TODAY!

Fri, Oct 29	Counting Inversions F19 F18 F17 x^2	[KT, Sec 5.3] (**Project (Problem 1 Coding) in**)
Mon, Nov 1	Multiplying large integers F19 F18 F17 x^2	[KT, Sec 5.5] (**Project (Problem 1 Reflection) in**) *Reading Assignment:* Unraveling the mystery behind the identity
Wed, Nov 3	Closest Pair of Points F19 F18 F17 x^2	[KT, Sec 5.4]
Fri, Nov 5	Kickass Property Lemma F19 F18 F17 x^2	[KT, Sec 5.4] (**Project (Problem 2 Coding) in**)
Mon, Nov 8	Weighted Interval Scheduling F19 F17 x^2	[KT, Sec 6.1] (**Project (Problem 2 Reflection) in**)

Group formation instructions

Autolab group submission for CSE 331 Project

The lowdown on submitting your project (especially the coding and reflection) problems as a group on Autolab.

Follow instructions **EXACTLY** as they are stated

The instruction below are for Coding Problem 1

You will have to repeat the instructions below for EACH coding AND reflection problem on project on Autolab (with the appropriate changes to the actual problem).

Form your group on Autolab

Groups on Autolab will NOT be automatically created

You will have to form a group on Autolab by yourself (as a group). Read on for instructions on how to go about this.

Click to add notes

Preliminary grading rubric

note @401

stop following 5 views

Preliminary rubrics for reflections problems up

We have added **preliminary** grading rubrics for each reflection question:

http://www-student.cse.buffalo.edu/~atri/cse331/fall21/project/reflection.html

As noted in the page above, please keep in mind that in **actual grading, we will use a grading rubric that expands on the preliminary grading rubric,** i.e. you are NOT seeing the final rubric that will be used to grade your submissions.

We hope this preliminary grading rubric helps as y'all start working on the reflection questions.

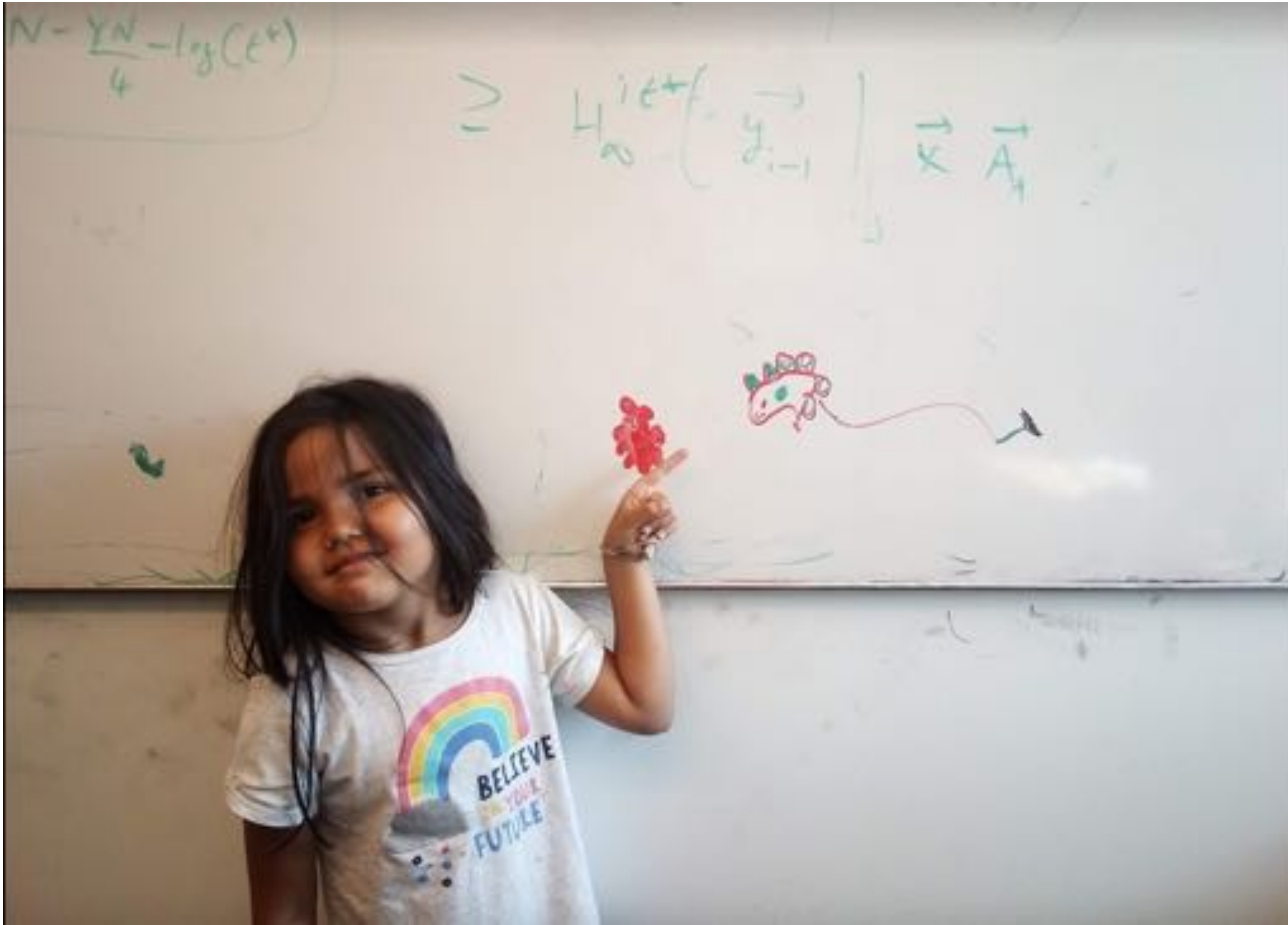
project

Preliminary Grading Guidelines

Below is a *preliminary* instantiation of the generic grading rubric above for (all ten parts of) Problem 1. In **actual grading, we will use a grading rubric that expands on the preliminary grading rubric below.**

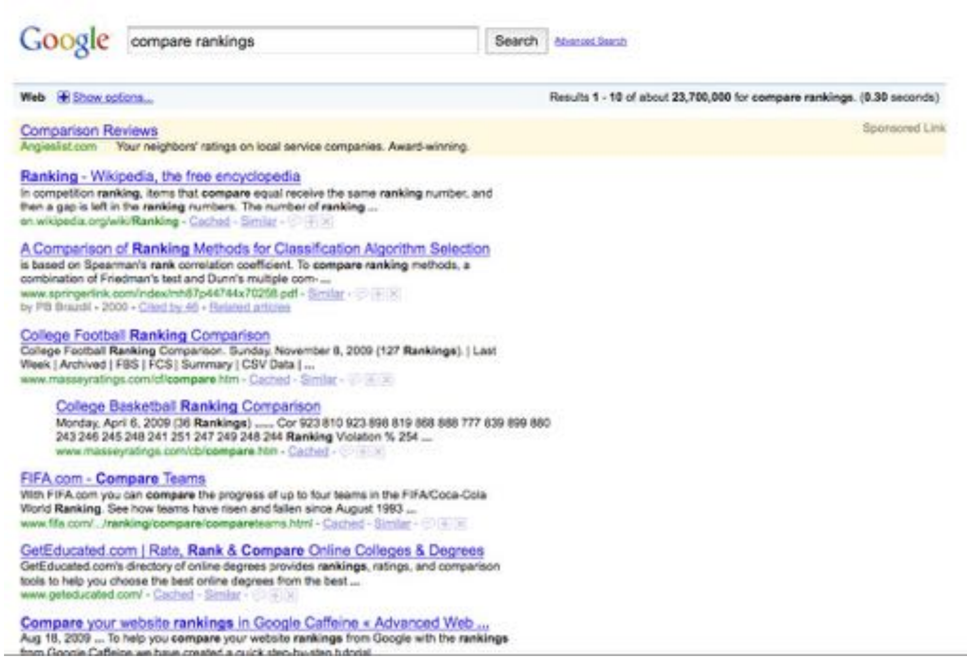
- `Level 0`
  1. The authors did not respond with all 10 stakes; OR
  2. Answers may not be entirely relevant to the assignment.
- `Level 1`
  1. The authors did respond with all 10 stakes. Although, the responses may be underdeveloped; AND
  2. The authors clearly understand the questions, but have not demonstrated much effort in thinking through the different interests each stakeholder would have. Answers may seem perfunctory.
- `Level 2`
  1. The authors respond with all 10 stakes thoroughly and thoughtfully; AND
  2. The authors clearly demonstrate their grasp of the questions and the various perspectives each stakeholder might have on the same design; AND
  3. They demonstrate that what stakeholders' value differs depending on their own context.

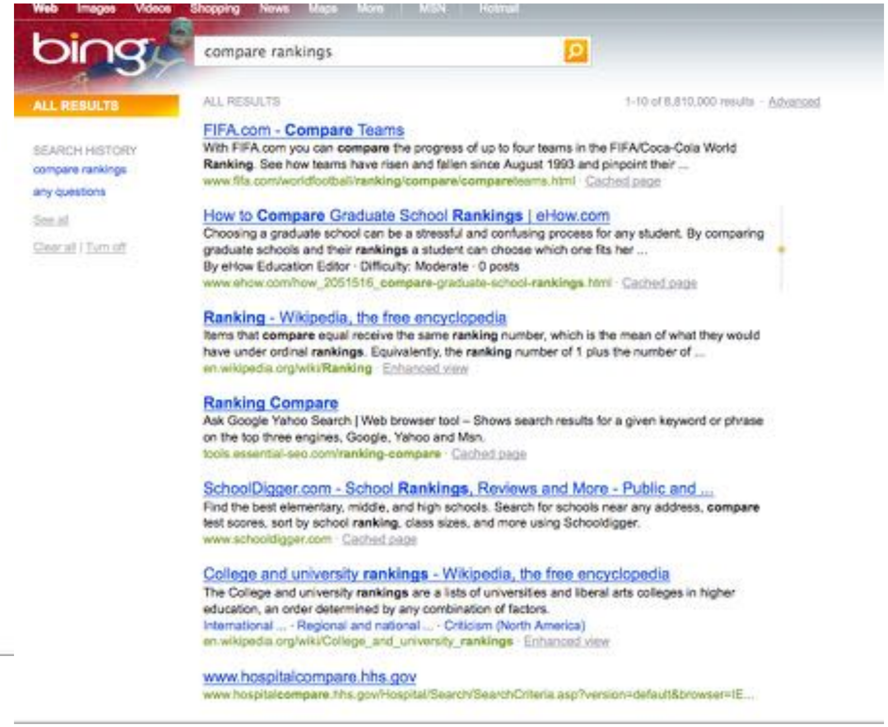
Questions/Comments?



Rankings

How close are two rankings?





Rest of today’s agenda

Formal problem: Counting inversions

Divide and Conquer algorithm

Problem definition on the board…

Solve a harder problem

Input: $a_1, .., a_n$

Output: LIST of all inversions

```
L = ϕ
for i in 1 to n-1
    for j in i+1 to n
        If a_i > a_j
            add (i,j) to L

return  L
```

Optimal for the listing problem

Example 1: All inversions-- (2i-1,2i)

2	1	3	4	6	5	7	8

Only check (i,i+1) pairs

Q1: Solve listing problem in O(n) time?

Q2: Recursive divide and conquer algorithm to count the number of inversions?

CountInv (a,n)

if $n = 1$ return 0

if $n = 2$ return $a_1 > a_2$

$a_L = a_1 , .., a_{[n/2]}$

$a_R = a_{[n/2]+1} , .., a_n$

return CountInv(a_L, [n/2]) + CountInv(a_R, n- [n/2])

Can be horribly wrong in general

CountInv (a,n)

if $n = 1$ return 0

if $n = 2$ return $a_1 > a_2$

$a_L = a_1 , .., a_{[n/2]}$

$a_R = a_{[n/2]+1} , .., a_n$

return CountInv(a_L, $[n/2]$) + CountInv(a_R, $n- [n/2]$)

Example where instance has non-zero (can be $\Omega(n^2)$) inversions and algo returns 0?

5	6	1	2

All 4 “crossing” pairs are inversions

Bad case: “crossing inversions”

CountInv (a,n)

  if $n = 1$ return 0

  if $n = 2$ return $a_1 > a_2$

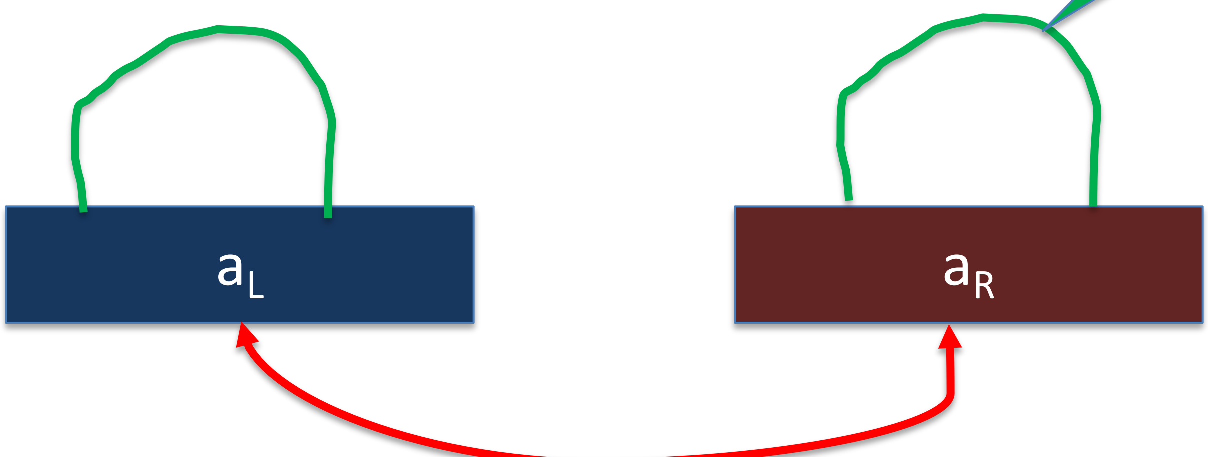
  $a_L = a_1, .., a_{[n/2]}$

  $a_R = a_{[n/2]+1}, .., a_n$

  return CountInv(a_L, [n/2]) + CountInv(a_R, n- [n/2])

Yes!

Are a_L and a_R sorted?



Example 2: Solving the bad case

a_L is sorted

First element is a_L is larger than first/only element in a_R

O(1) algorithm to count number of inversions?

return size of a_L

Example 3: Solving the bad case

1

a_L

5	6	…..

a_R

a_R is sorted

First/only element is a_L is smaller than first element in a_R

O(1) algorithm to count number of inversions?

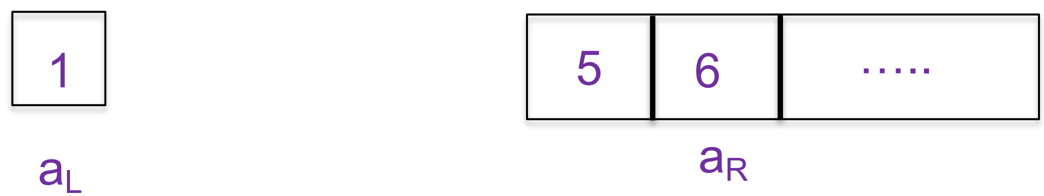
return 0

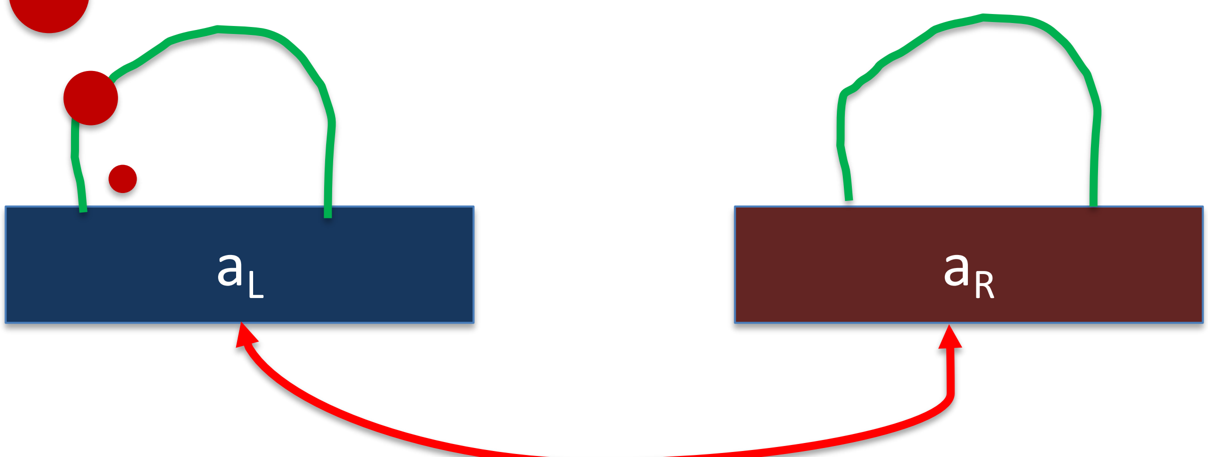
Solving the bad case

Try to modify the MERGE algorithm

First element of a_L is larger than first element of a_R

First element of a_L is smaller than first element of a_R





Divide and Conquer

Divide up the problem into at least two sub-problems

Solve all sub-problems: Mergesort

Recursively solve the sub-problems

Solve stronger sub-problems: Inversions

"Patch up" the solutions to the sub-problems for the final solution

MergeSortCount algorithm

Input: $a_1, a_2, \ldots, a_n$

Output: Numbers in sorted order+ #inversion

MergeSortCount(a, n)

If $n = 1$ **return** $(0 , a_1)$

If $n = 2$ **return** $(a1 > a2, \min(a_1,a_2); \max(a_1,a_2))$

$a_L = a_1,\ldots, a_{n/2}$ $a_R = a_{n/2+1},\ldots, a_n$

(c_L, a_L) = MergeSortCount(a_L, $n/2$)

(c_R, a_R) = MergeSortCount(a_R, $n/2$)

(c, a) = MERGE-COUNT(a_L,a_R)

Counts #crossing-inversions+ MERGE

return $(c+c_L+c_R,a)$

$T(2) = c$

$T(n) = 2T(n/2) + cn$

O(n log n) time

O(n)

MERGE-COUNT(a_L,a_R)

$a_L = l_1,\ldots, l_{n'}$ $\qquad$ $a_R = r_1,\ldots, r_m$

```
c = 0
i,j = 1
while i ≤ n' and j ≤ m
        if l_i ≤ r_j
            i ++
            add l_i to output
        else
            add r_j to output
            j ++
            c += n'- i +1
Output any remaining items
return c
```