CS340 - Analysis of Algorithms

Divide and Conquer II

Announcements:

Hw5 released

Due November 3rd

Divide and Conquer Quiz upcoming.

November 6th in Lab?

Project feedback sent

Warmup

Consider the following pseudo code, which searches for a number x in an array $A[1,...,m]$ by calling `BSearch(A, x, 1, m)`

```
Function BSearch(array A, integer x, integer left, integer right)
  if left > right
   then
   | return −1
  end
  m = (left+right)/2
  if A[m] = x
   then
   | return m
  end
  else
  | if A[m] > x
  |  then
  |  | return BSearch(A, x, left, m-1)
  |  end
  end
  else
  | return BSearch(A, x, m+1, right)
  end
```

Note that the function operates on an input size of `right - left + 1`. Write the recurrence $T(n)$ that denotes the maximum number of steps `BSeach` makes on an input of size n. In particular,

1. In the pseudocode above, identify the non-recursive parts and estimate their running time in big-Oh notation.
2. In the pseudocode above, identify the recursive parts and estimate their running time using $T(n)$.
3. Given the recurrence for $T(n)$. Don't worry about rounding. Do not forget the base case.

Master Theorem

If $T(n) = aT(n/b) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

How to apply it:

1. Identify a, b, and d from the recurrence relation
2. Calculate $\log_b(a)$
3. Compare d to logb(a) to determine which of the three cases applies

Merge Sort Psuedo Code

```
mergeSort(A, p, r){
  if (p<r) {
    m = (p+r)/2
    mergeSort(A, p, m)
    mergeSort(A, m+1, r)
    merge(A, p, m, r)
  }
}
```

```
merge(A, p, m, r){
  new B[0, r-p]
  i=p; j=m+1; k=0
  while(i<=m and j<=r){
    if (A[i]<=A[j])
      B[k++] = A[i++]
    else
      B[k++] = A[j++]
  }
  while(i<=m) B[k++]=A[i++]
  while(j<=r) B[k++]=A[j++]
  copy B back to A
}
```

Termination of Merge Sort

```
mergeSort(A, p, r){
  if (p<r) {
    m = (p+r)/2
    mergeSort(A, p, m)
    mergeSort(A, m+1, r)
    merge(A, p, m, r)
  }
}
```

Sizes of subproblems decreases by at least 1 in each recursive call, so merge sort will terminate in finite time

Correctness of Merge Sort

- Structural induction
  - base: $|A| = 1$
  - IH: assume mergeSort works $\forall i, 1 \leq i \leq n-1$
  - n:
    - divide works
    - $m-p+1 < n, r-m < n \rightarrow$ apply IH on the two recursive calls
    - needs a lemma that proves merge works on two sorted lists
- Lemma needs another induction

Correctness of Merge

```
merge(A, p, m, r){
  new B[0, r-p]
  i=p; j=m+1; k=0
  while(i<=m and j<=r){
    if (A[i]<=A[j])
      B[k++] = A[i++]
    else
      B[k++] = A[j++]
  }
  while(i<=m) B[k++]=A[i++]
  while(j<=r) B[k++]=A[j++]
  copy B back to A
}
```

Subarrays are sorted.

n_l = # of elems consumed from left subarray (i - p)
n_r = # of elems consumed from right subarray (j - m+1)

<u>Invariant P(n_l, n_r):</u> after consuming n_l elements from the left and n_r elems from the right, B contains the smallest elements from A[p..m] and A[m+1...r] in sorted order

Base case: P(0,0)

IH: P(k_l, k_r) holds for 0<= k_l , k_r < n_l , n_r

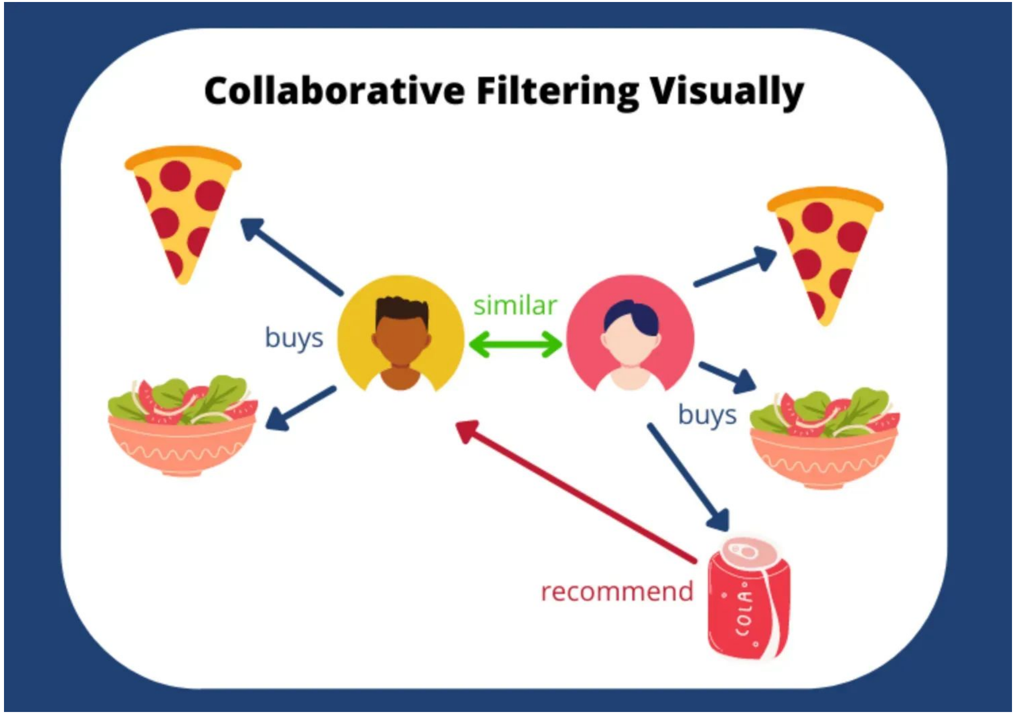
Case 1: A[p+**n□-1**] <= A[m+1+n_r]

Case 2: A[p+n□] > A[m+1+**n_r-1**]

Merge Sort Applications

Collaborative Filtering

- Match your preferences with those of other people online
- Identify people with similar tastes
- Recommend things that these other people have liked



Problem Formulation

Suppose you are given rank-ordered lists of movies

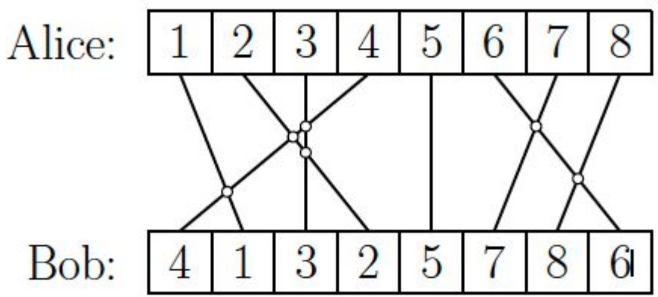
- Determine similarity between lists
- Find people with similar tastes

Movie Title	Alice	Bob	Carol
Gone with the Wind	1	4	6
Citizen Kane	2	1	8
The Seven Samurai	3	3	4
The Godfather	4	2	1
Titanic	5	5	7
My Cousin Vinny	6	7	2
Star Wars	7	8	5
Plan 9 from Outer Space	8	6	3

Collaborative Filtering

- Label movies from 1 to n according to Alice's ranking
- Order the labels according to Bob's ranking
- See how many are "out of order"

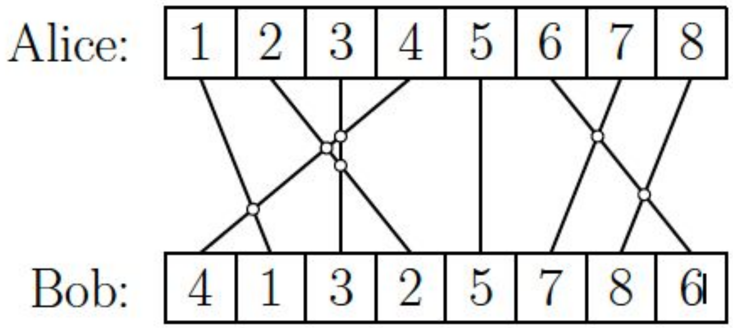
Given a sequence of n numbers a_1, ..., a_n (Bob's rankings), how far is this list from being sorted in ascending order?

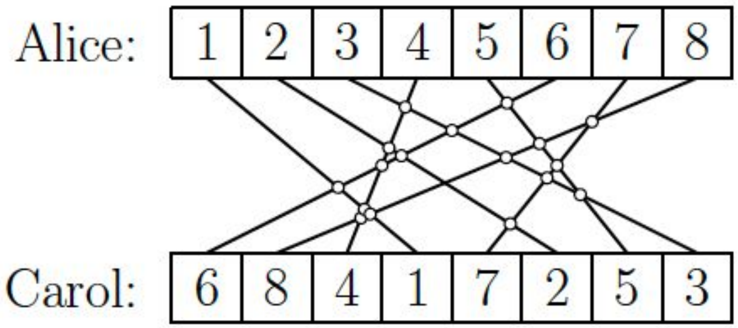


Inversions

Given two lists of preferences L_1 and L_2 define an inversion to be a pair of movie preferences x and y, such that L_1 has x before y and L_2 has y before x

We say two indices $i < j$ form an inversion if $a_i > a_j$





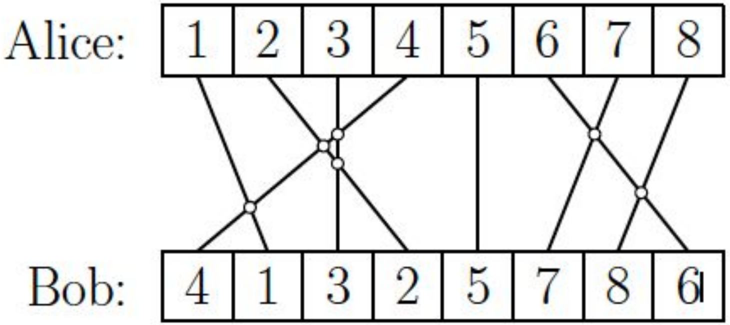
Inversions

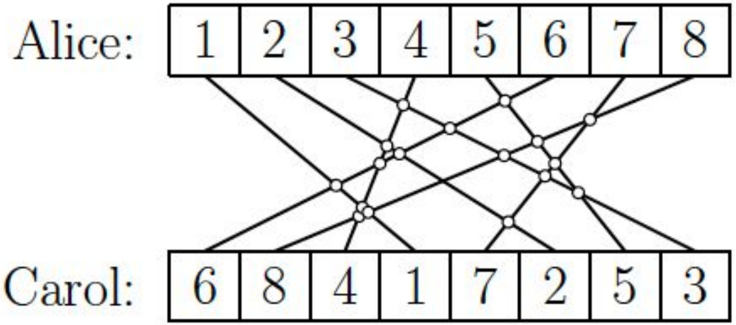
Inversions between Alice and Bob?

(4, 1), (4, 3), (4, 2), (3, 2), (7, 6), (8, 6)

How many max inversions can there be in a list of n numbers?

n^2





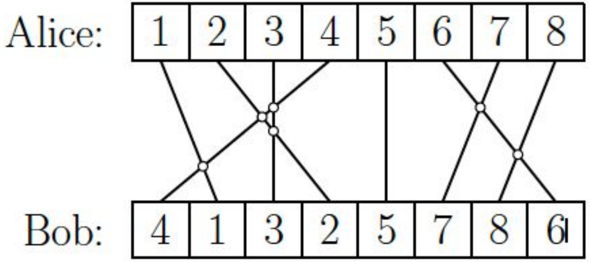
Brute Force Inversion Counting

Look at every pair of numbers (a_i , a_j) and determine whether they constitute an inversion

Runtime?

$O(n^2)$

Divide and Conquer Inversion Counting



- Base: $n = 1$, no inversions
- General
  - divide: split list into two halves of size $n/2$
  - conquer: compute the number of inversions in each half
  - combine: count the number of inversions between the two halves
- Use merge sort

Divide and Conquer Inversion Counting

```
COUNT(A)
    n = |A|
    if n ≤ 1 then
            return 0

   mid = n / 2
   l ← COUNT(A[0:mid])
   r ← COUNT(A[mid+1:n])
   s ← MERGE(A[0:mid], A[mid+1:n])
   return l + r + s
```

Inversion Counting

```
MERGE(L, R)
    i, j, c = 0, 0, 0

    while i ≤ |L| and j ≤ |R| do
        if L[i] ≤ R[j] then
            i++
        else
            j++
            // All remaining elements in L form inversions with R[j]
            c ← c + (|L| - i)

    return c
```

Inversion Counting

```
COUNT(A)
    n = |A|
    if n ≤ 1 then
            return 0

   mid = n / 2
   l ← COUNT(A[0:mid])
   r ← COUNT(A[mid+1:n])
   s ← MERGE(A[0:mid], A[mid+1:n])
   return l + r + s
```

```
MERGE(L, R)
    i, j, c = 0, 0, 0

    while i ≤ |L| and j ≤ |R| do
        if L[i] ≤ R[j] then
            i++
        else
            j++
            // All remaining elements in L form inversions with R[j]
            c ← c + (|L| - i)

    return c
```

Project Feedback

Checkpoint 1 Feedback

Greedy criteria: popularity-based (Per class preference count)

Adjust popularity by per class-pair conflict count?

- No: sort by popularity and schedule by room capacity
  - USE ARRAYS - students and classes are all small numbers that can be used as indices
- Yes:
  - Might want to use a more efficient data structure for dynamic recomputation

Checkpoint 1 Feedback

- Classes should be objects with member variables:

  id, prof, capacity (room count), etc.

- Schedule should be a 2D array of classes indexed by rooms/timeshots:

  `schedule[`r_i`, `t_j`] = 340`

- Register students after you have generated the schedule not during

- If a student doesn't get into a class they want, leave them unregistered

Checkpoint 2

- Your program should be executable
- Time analysis verification:
  a. Measure actual runtime and verify your time analysis as variables grow
  b. Generate instances of increasing sizes in your dominant variable (use the scripts given)
     - If you don’t know the dominant variable, simplify your Big-O expression
  c. Graph the run time vs size
     - Depending on your big-O,
     - Shape of the curve might be obvious
     - If not, you run regression to obtain a fit curve

Checkpoint 2

- Also test solution quality!
  - Generate different combinations of s,c,r,t
  - What are the preference scores as a % of max?

Summary

- Start your HW5

- Quiz on divide and conquer upcoming

- Continue working on project