

CS/ECE 374 A ✧ Spring 2018

🌀 Homework 4 🌀

Due Tuesday, February 27, 2018 at 8pm

1. At the end of the second act of the action blockbuster *Fast and Impossible XIII^{3/4}: Guardians of Expendable Justice Reloaded*, the villainous Dr. Metaphor hypnotizes the entire Hero League/Force/Squad, arranges them in a long line at the edge of a cliff, and instructs each hero to shoot the closest taller heroes to their left and right, at a prearranged signal.

Suppose we are given the heights of all n heroes, in clockwise order around the circle, in an array $Ht[1..n]$. (To avoid salary arguments, the producers insisted that no two heroes have the same height.) Then we can compute the Left and Right targets of each hero in $O(n^2)$ time using the following algorithm.

```
WHOTARGETSWHOM( $Ht[1..n]$ ):
  for  $j \leftarrow 1$  to  $n$ 
    «Find the left target  $L[j]$  for hero  $j$ »
     $L[j] \leftarrow \text{NONE}$ 
    for  $i \leftarrow 1$  to  $j - 1$ 
      if  $Ht[i] > Ht[j]$ 
         $L[j] \leftarrow i$ 
    «Find the right target  $R[j]$  for hero  $j$ »
     $R[j] \leftarrow \text{NONE}$ 
    for  $k \leftarrow n$  down to  $j + 1$ 
      if  $Ht[k] > Ht[j]$ 
         $R[j] \leftarrow k$ 
  return  $L[1..n], R[1..n]$ 
```

- (a) Describe a divide-and-conquer algorithm that computes the output of WHOTARGETSWHOM in $O(n \log n)$ time.
- (b) Prove that at least $\lfloor n/2 \rfloor$ of the n heroes are targets. That is, prove that the output arrays $R[0..n-1]$ and $L[0..n-1]$ contain at least $\lfloor n/2 \rfloor$ distinct values (other than NONE).
- (c) Alas, Dr. Metaphor's diabolical plan is successful. At the prearranged signal, all the heroes simultaneously shoot their targets, and all targets fall over the cliff, apparently dead. Metaphor repeats his dastardly experiment over and over; after each massacre, he forces the remaining heroes to choose new targets, following the same algorithm, and then shoot their targets at the next signal. Eventually, only the shortest member of the Hero Crew/Alliance/Posse is left alive.¹

Describe an algorithm that computes the number of rounds before Dr. Metaphor's deadly process finally ends. For full credit, your algorithm should run in $O(n)$ time.

¹In the thrilling final act, Retcon the Squirrel, the last surviving member of the Hero Team/Group/Society, saves everyone by traveling back in time and retroactively replacing the other $n - 1$ heroes with lifelike balloon sculptures.

2. Describe and analyze a recursive algorithm to reconstruct an arbitrary binary tree, given its preorder and inorder node sequences as input.

The input to your algorithm is a pair of arrays $Pre[1..n]$ and $In[1..n]$, each containing a permutation of the same set of n distinct symbols. Your algorithm should return an n -node binary tree whose nodes are labeled with those n symbols (or an error code if no binary tree is consistent with the input arrays). You solved an instance of this problem in Homework 0.

3. Suppose we are given a set S of n items, each with a *value* and a *weight*. For any element $x \in S$, we define two subsets:

- $S_{<x}$ is the set of all elements of S whose value is smaller than the value of x .
- $S_{>x}$ is the set of all elements of S whose value is larger than the value of x .

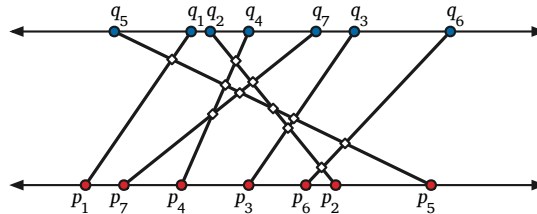
For any subset $R \subseteq S$, let $w(R)$ denote the sum of the weights of elements in R . The **weighted median** of R is any element x such that $w(S_{<x}) \leq w(S)/2$ and $w(S_{>x}) \leq w(S)/2$.

Describe and analyze an algorithm to compute the weighted median of a given weighted set in $O(n)$ time. Your input consists of two unsorted arrays $S[1..n]$ and $W[1..n]$, where for each index i , the i th element has value $S[i]$ and weight $W[i]$. You may assume that all values are distinct and all weights are positive.

[Hint: Use or modify the linear-time selection algorithm described in class on Thursday.]

Solved problem

4. Suppose we are given two sets of n points, one set $\{p_1, p_2, \dots, p_n\}$ on the line $y = 0$ and the other set $\{q_1, q_2, \dots, q_n\}$ on the line $y = 1$. Consider the n line segments connecting each point p_i to the corresponding point q_i . Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect, in $O(n \log n)$ time. See the example below.



Seven segments with endpoints on parallel lines, with 11 intersecting pairs.

Your input consists of two arrays $P[1..n]$ and $Q[1..n]$ of x -coordinates; you may assume that all $2n$ of these numbers are distinct. No proof of correctness is necessary, but you should justify the running time.

Solution: We begin by sorting the array $P[1..n]$ and permuting the array $Q[1..n]$ to maintain correspondence between endpoints, in $O(n \log n)$ time. Then for any indices $i < j$, segments i and j intersect if and only if $Q[i] > Q[j]$. Thus, our goal is to compute the number of pairs of indices $i < j$ such that $Q[i] > Q[j]$. Such a pair is called an *inversion*.

We count the number of inversions in Q using the following extension of mergesort; as a side effect, this algorithm also sorts Q . If $n < 100$, we use brute force in $O(1)$ time. Otherwise:

- Color the elements in the Left half $Q[1.. \lfloor n/2 \rfloor]$ **blue**.
- Color the elements in the Right half $Q[\lfloor n/2 \rfloor + 1.. n]$ **red**.
- Recursively count inversions in (and sort) the **blue** subarray $Q[1.. \lfloor n/2 \rfloor]$.
- Recursively count inversions in (and sort) the **red** subarray $Q[\lfloor n/2 \rfloor + 1.. n]$.
- Count **red/blue** inversions as follows:
 - MERGE the sorted subarrays $Q[1.. \lfloor n/2 \rfloor]$ and $Q[\lfloor n/2 \rfloor + 1.. n]$, maintaining the element colors.
 - For each **blue** element $Q[i]$ of the now-sorted array $Q[1.. n]$, count the number of smaller **red** elements $Q[j]$.

The last substep can be performed in $O(n)$ time using a simple for-loop:

```

COUNTREDBLUE(A[1..n]):
  count ← 0
  total ← 0
  for i ← 1 to n
    if A[i] is red
      count ← count + 1
    else
      total ← total + count
  return total

```

MERGE and COUNTREDBLUE each run in $O(n)$ time. Thus, the running time of our inversion-counting algorithm obeys the mergesort recurrence $T(n) = 2T(n/2) + O(n)$. (We can safely ignore the floors and ceilings in the recursive arguments.) We conclude that the overall running time of our algorithm is $O(n \log n)$, as required.

Rubric: This is enough for full credit.

In fact, we can execute the third merge-and-count step directly by modifying the MERGE algorithm, without any need for “colors”. Here changes to the standard MERGE algorithm are indicated in red.

```

MERGEANDCOUNT(A[1..n], m):
  i ← 1; j ← m + 1; count ← 0; total ← 0
  for k ← 1 to n
    if j > n
      B[k] ← A[i]; i ← i + 1; total ← total + count
    else if i > m
      B[k] ← A[j]; j ← j + 1; count ← count + 1
    else if A[i] < A[j]
      B[k] ← A[i]; i ← i + 1; total ← total + count
    else
      B[k] ← A[j]; j ← j + 1; count ← count + 1
  for k ← 1 to n
    A[k] ← B[k]
  return total

```

We can further optimize MERGEANDCOUNT by observing that *count* is always equal to $j - m - 1$, so we don't need an additional variable. (Proof: Initially, $j = m + 1$ and $count = 0$, and we always increment j and $count$ together.)

```

MERGEANDCOUNT2(A[1..n], m):
  i ← 1; j ← m + 1; total ← 0
  for k ← 1 to n
    if j > n
      B[k] ← A[i]; i ← i + 1; total ← total + j - m - 1
    else if i > m
      B[k] ← A[j]; j ← j + 1
    else if A[i] < A[j]
      B[k] ← A[i]; i ← i + 1; total ← total + j - m - 1
    else
      B[k] ← A[j]; j ← j + 1
  for k ← 1 to n
    A[k] ← B[k]
  return total

```

MERGEANDCOUNT2 still runs in $O(n)$ time, so the overall running time is still $O(n \log n)$, as required. ■

Rubric: 10 points = 2 for base case + 3 for divide (split and recurse) + 3 for conquer (merge and count) + 2 for time analysis. Max 3 points for a correct $O(n^2)$ -time algorithm. This is neither the only way to correctly describe this algorithm nor the only correct $O(n \log n)$ -time algorithm. No proof of correctness is required.

Notice that each boxed algorithm is preceded by an English description of the task that algorithm performs. **Omitting these descriptions is a Deadly Sin.**