CompSci 161
Spring 2021 Lecture 6:
Divide and Conquer I:
Inversion Counting

2

Counting Inversions

- i, j are an *inverted pair* if $i < j$ and $A[i] > A[j]$.
  (the larger element appears earlier in the array)

The following is an $\Theta(n^2)$ time way to count the inversions in an array:

count $= 0$
for $i = 1 \ldots n$ **do**
  for $j = i + 1 \ldots n$ **do**
    if $A[i] > A[j]$ **then**
      count++
return count

- brute force
- polynomial time

3 Counting Inversions Faster: a subproblem

- want to count number of inverted pairs in A,
- we know $A[\overset{i}{1} \ldots \frac{n}{2}]$ is sorted, as is $A[\frac{n}{2}+1 \ldots n]$.
- Can we do better than $\Theta(n^2)$?

```
i=1, j= n/2+1, count=0        temp[1...n], k=1
while  i ≤ n/2  and  j ≤ n
    if  A[i] ≤ A[j]
          temp[k]= A[i];
          i++;  k++;
    else
          count += #elements A[i...n/2]
          temp[k]= A[j];
          j++
              k++
```

4 # Finishing the Merge Portion

- We want sorted list when done
- Let's keep the rest of the array

```
while  i ≤ n/2
       temp[k]= A[i]
       i++;    k++;
while  j ≤ n
       temp [k]= A[j]
       j++;    k++;
Copy temp to A
return count
```

5

Counting Inversions Faster

- Use the algorithm from the previous question
- count number of inversions in *unsorted* array
- How fast is your algorithm?

CountInv(A):
  if A is small
    brute force count and sort
    return count
  else // $L = A[1 \ldots n/2]$, $R = A[n/2+1, n]$
    $\left.\begin{array}{l} C_L = \text{CountInv}(L) \\ C_R = \text{CountInv}(R) \end{array}\right\}$ 2 recursive calls size $n/2$
    C_M = merge-and-count (A) } linear
    return $C_L + C_R + C_M$

6

Running Time for Counting Inversions

if list has one or zero elements **then** } or small : brute force
  return no inversions
Divide into $L = A[1 \ldots \frac{n}{2}]$ and $R = A[\frac{n}{2} + 1 \ldots n]$
Recursively solve on L; count is c_L
Recursively solve on R; count is c_R
Run earlier subproblem on L, R; count is c_M
return $c_L + c_R + c_M$

$T(n)$: Time needed/used to solve instance of size n

How long does this take? $T(n) = \underbrace{2}_{\text{two recursive calls}} T(\underbrace{n/2}_{\text{each size } n/2}) + \underbrace{n}_{\text{linear work not recursive}}$

Running Time for Counting Inversions

$T(n) = 2T(n/2) + n$ $\quad T(\frac{n}{4}) = 2T(\frac{n/4}{2}) + n/4$

$T(n/2) = 2T(n/4) + n/2$ $\quad 2T(n/8) + n/4$

- Two recursive of size $n/2$, plus local linear work
- $T(n) = 2\boxed{T(n/2)} + n$ $\quad T(n/2)$

$$T(n) = 2\left[\overline{2T(\tfrac{n}{4}) + \tfrac{n}{2}}\right] + n$$

$$= 4T(n/4) + n + n = 4\underbrace{T(\tfrac{n}{4})} + 2n$$

$$= 4\left[2T(n/8) + n/4\right] + 2n$$

$$= 8T(n/8) + n + 2n = 8T(n/8) + 3n$$

$$= 16T(n/16) + 4n \quad \ldots \quad \underbrace{2^i T(n/2^i)} + in$$

when $i = \log_2 n$ $\quad \underbrace{n \cdot c} + (\log_2 n) \cdot n$