Lecture 26

CSE 331

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)

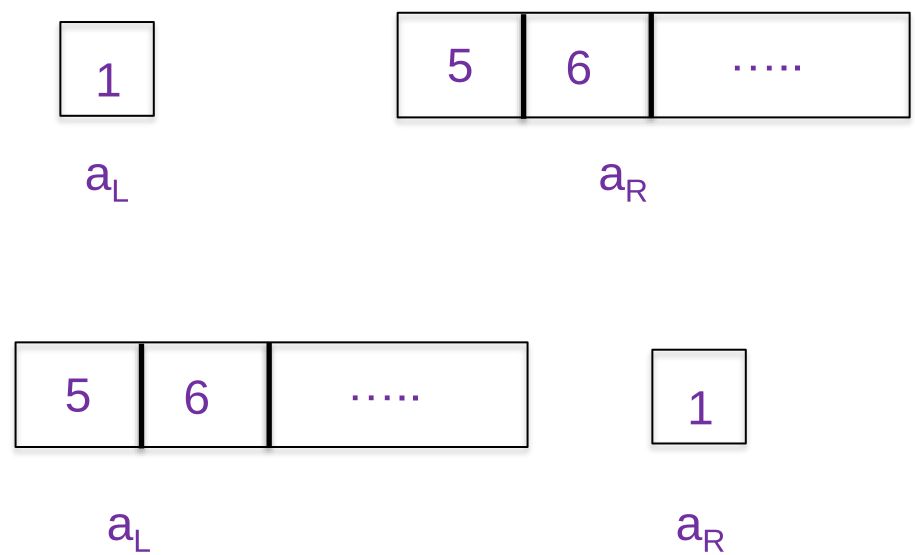
return $(c+c_L+c_R, a)$

Counts #crossing-inversions+ MERGE

MERGE-COUNT(a_L,a_R)

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

```
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
```



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** ($a1 > a2$, $\min(a_1,a_2)$; $\max(a_1,a_2)$)

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

(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)

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

Counts #crossing-inversions+ MERGE

$T(2) = c$

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

O(n log n) time

O(n)

Improvements on a smaller scale

Greedy algorithms: exponential → poly time

(Typical) Divide and Conquer: $O(n^2)$ → asymptotically smaller running time

Multiplying two numbers

Given two numbers a and b in binary

$a=(a_{n-1},..,a_0)$ and $b = (b_{n-1},...,b_0)$

Compute $c = a \times b$

Elementary school algorithm is $O(n^2)$

Problem definition on the board…