Counting Inversions

CSC 225 - LAB 5

In a sequence $S = [s_1, s_2, \ldots, s_n]$ of n integers, an *inversion* is a pair of elements s_i and s_j where $i < j$ (that is, s_i appears before s_j in the sequence) and $s_i > s_j$. For example, in the sequence

$$S = 2, 1, 5, 3, 4$$

the pairs (2,1), (5,3) and (5,4) are inversions.

An array with n elements may have as many as

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

inversions. When the number of inversions, k, may be any value between 0 and $\frac{n(n-1)}{2}$, the best algorithm for counting inversions has running time $O(n \log n)$. There also exists a $O(n + k)$ algorithm for counting inversions, which is $O(n^2)$ when $k \in O(n^2)$.

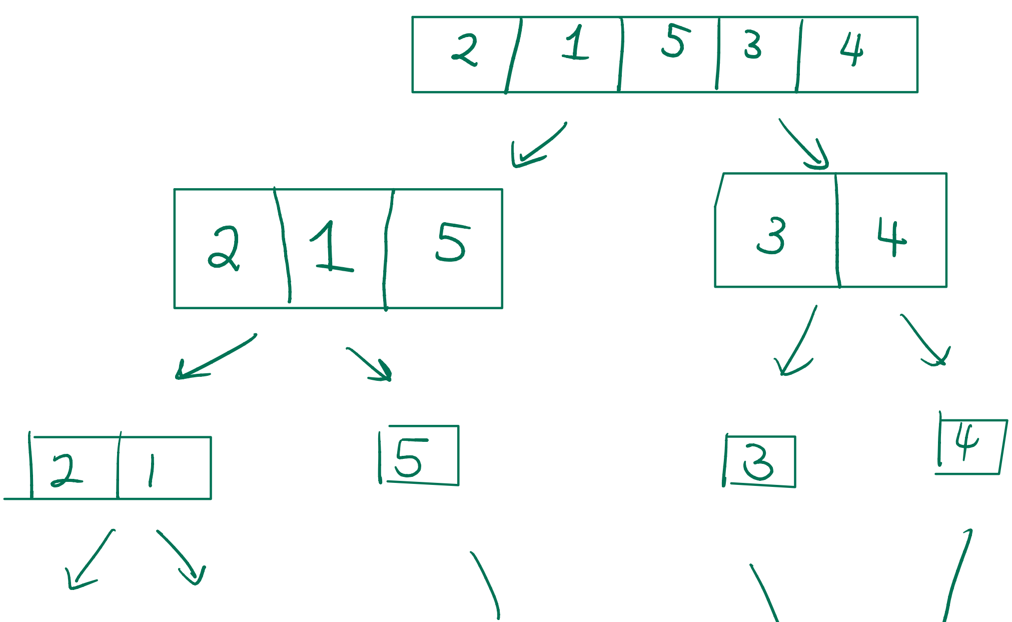
Your goal in this lab is to create two algorithms which count the number of inversions in an input sequence:

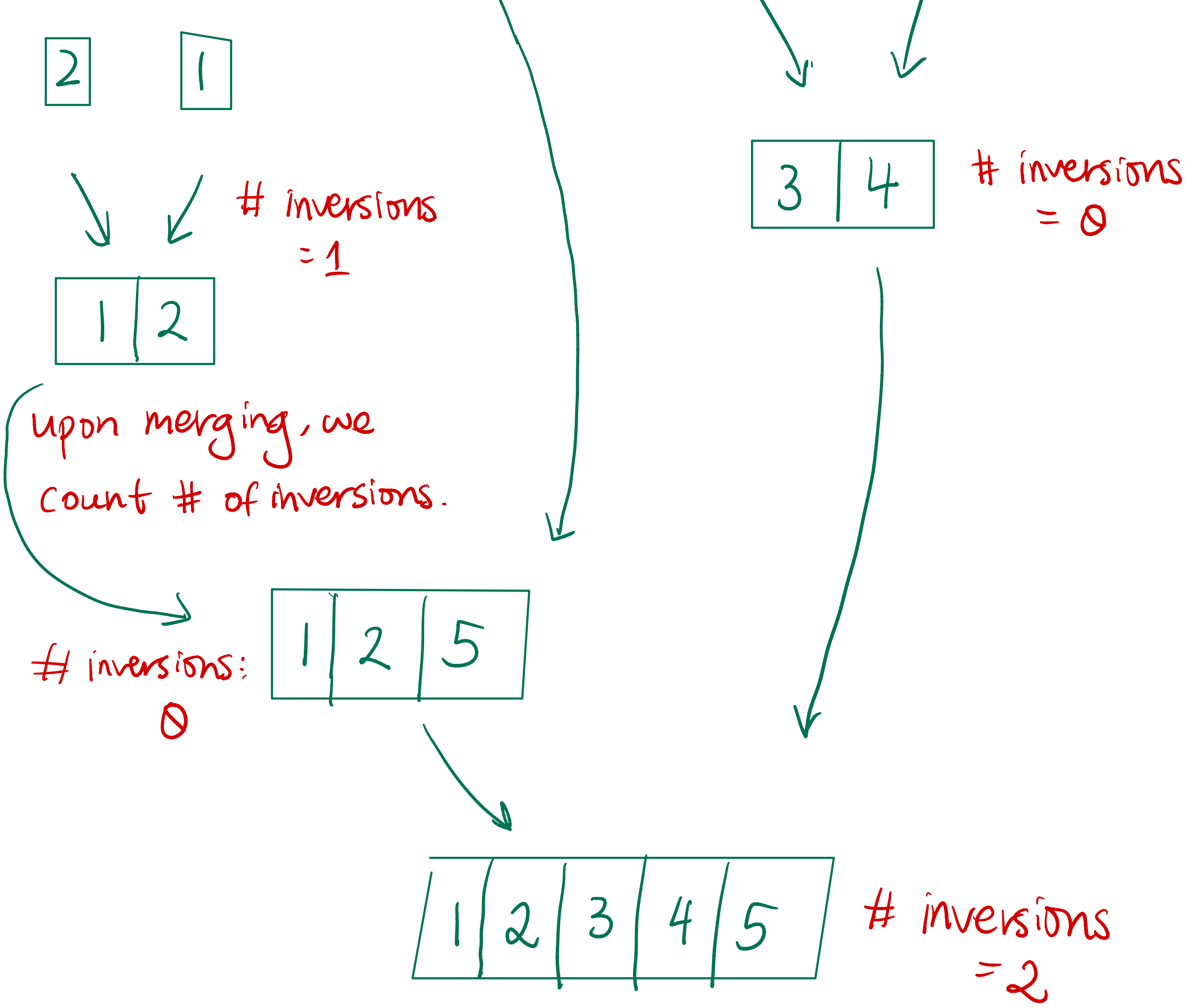
Input: An array A of n integers in the range 1 to n.
Output: An integer, corresponding to the number of inversions in A.

The first algorithm will have a $O(n \log n)$ runtime and will be better for counting inversions when $k > n$. The second algorithm will have runtime $O(n + k)$ and will be better when $k \leq n$ (i.e. $O(n + n) = O(n)$).

Bonus:
Time permitting, you should try to implement them.

① Merge sort





Total # of inversions: $1+2=3$

Every time we merge something on the right list, it must be an inversion with everything remaining in the left list

Merge(S1, S2, S)

★ assumes indexing from 0!

$n_1 \leftarrow |S1|$

$n_2 \leftarrow |S2|$

$i \leftarrow 0$

$j \leftarrow 0$

count $\leftarrow 0$

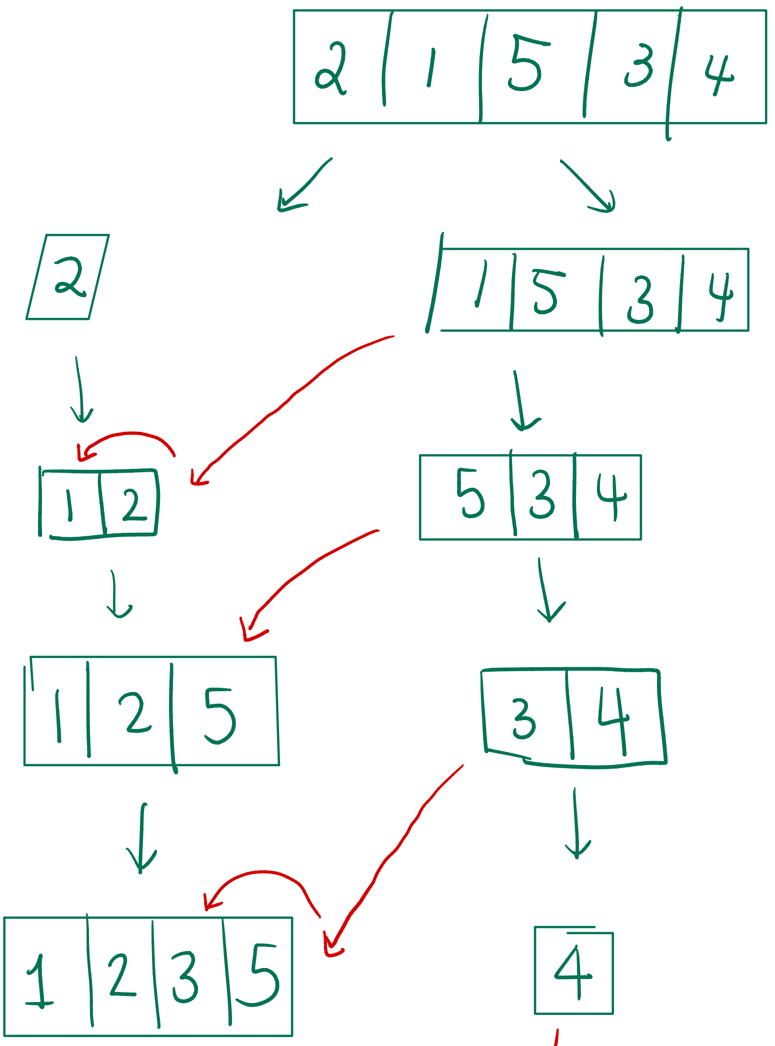
while ($i < n_1$ and $j < n_2$):

  if $S_1[i] \leq S_2[j]$:

    $S[i+j] \leftarrow S_1[i]$

```
            i++
        else:
            S[i+j] ← S2[j]
            count ← count + (n1 - i)
            j++
    while (i < n1):
        S[i+j] = S1[i]
        i++
    while (j < n2):
        S[i+j] = S2[j]
        j++
```

Insertion Sort



\# inversions = 0

\# inversions = 1

\# inversions = 0

\# inversions = 1

1	2	3	4	5

\# inversions $= 1$

Total # of inversions: $1 + 1 + 1 = 3$

```
InsertionSortWithCounting(A, n):
    count ← 0
    for i = 1 to i = n-1 do:
        val ← A[i]
        j ← i-1
        while (j ≥ 0 and A[j] > val):
            A[j+1] = A[j]
            count ← count + 1
            j ← j - 1
        A[j+1] = val
```

Normally this is $O(n^2)$ time. But we have that the number of inversions $k \leq n$.
So the while loop must execute $k \leq n$ times.
Thus, this takes $O(n+n) = O(n)$ time