Array-Based Sequences III

Sareh Taebi
COP3410 – Florida Atlantic University

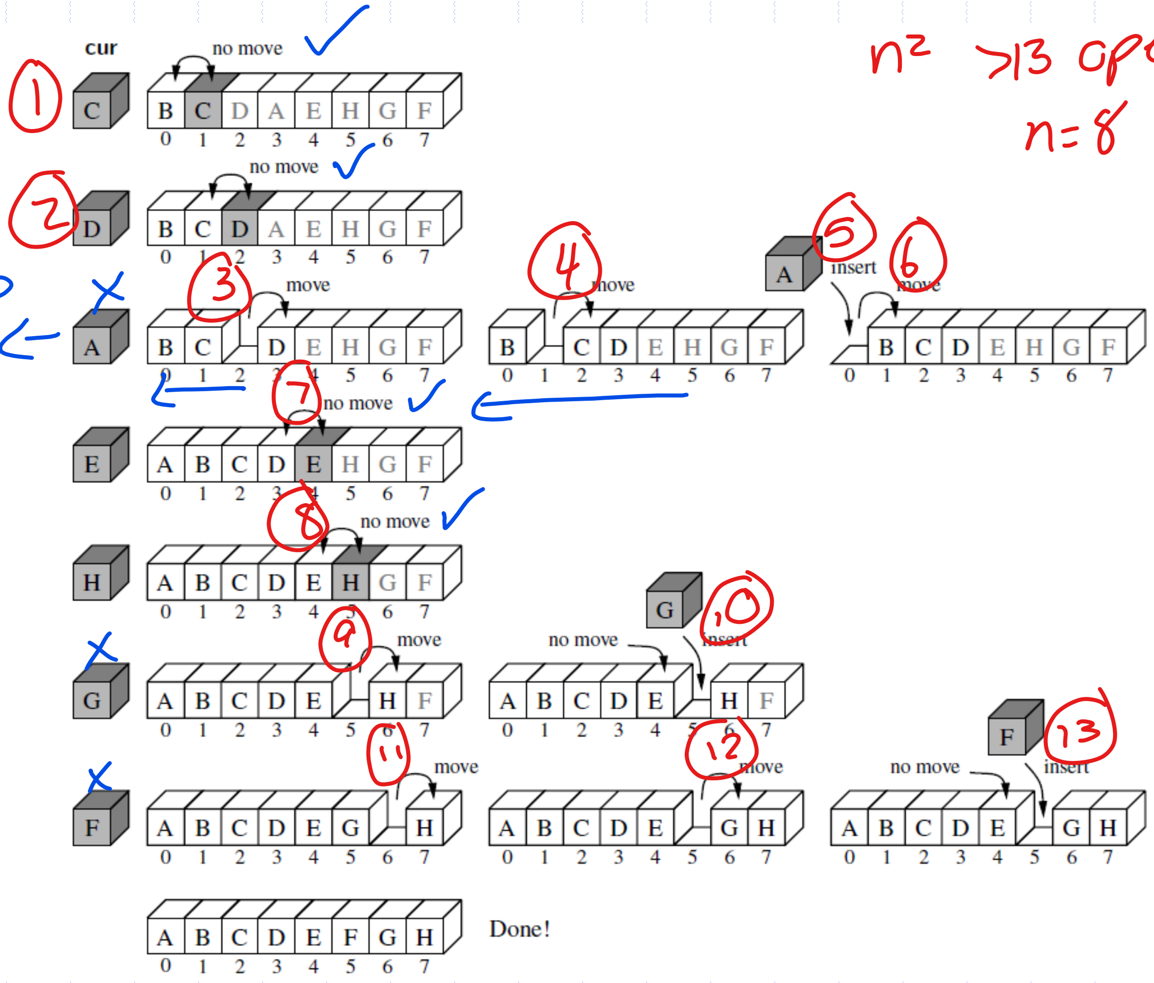
Sorting a Sequence

- **Insertion Sort:** Simple sorting algorithm.

Algorithm InsertionSort(A):

- *Input:* An array A of *n* elements
- *Output:* The array A with elements rearranged in increasing order
- **for** k from 1 to n – 1 **do**
  - Insert A[k] at its proper location within A[0], A[1], . . ., A[k].
  - swap elements if necessary

Execution of Insertion Sort



n^2 >13 operations
$n=8$

write loop moving left
↳ if stat.

Python Code

need 2 loops -> O(n²)
↳ can write other ways

```
def insertion_sort(A):
    """
sort by ascending order
    """"""
    for i in range(1,len(A)):     #outer loop goes through the list
        for j in range(i,0,-1):   #inner loop moves to the left of the current value
            if A[j] < A[j-1]:     #Swap two values if out of order
                temp = A[j]
                A[j] = A[j-1]
                A[j-1] = temp
    return A;
```

$\rightarrow$ $O(n^2)$

backwards

→ compare j to left

index 0

index 1

swap

#return sorted array

How good is insertion sort?

- The nested loops of insertion-sort lead to an $O(n^2)$ running time in the worst case.
  - The most work is done if the array is initially in reverse order.
- If the array is nearly sorted or perfectly sorted, insertion-sort runs in $O(n)$ time.
- What's the space complexity? -> do we need extra array for this?

↳ because we can move around inside array
  ↳ $O(n)$

heapsort? -> need more

list of lists

Multidimensional Data

- A two-dimensional array is also called a ***matrix.***
  - Matrices have important applications

$$\begin{matrix} 22 & 18 & 709 & 5 & 33 \\ 45 & 32 & 830 & 120 & 750 \\ 4 & 880 & 45 & 66 & 61 \end{matrix}$$

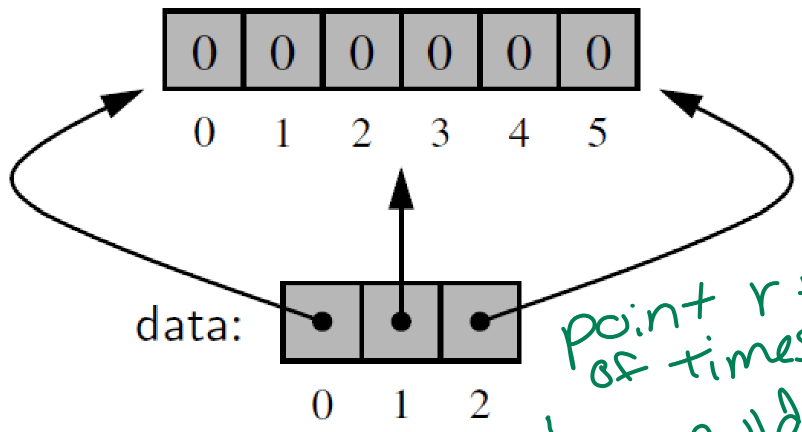
$$\text{data} = [\ \underbrace{[22, 18, 709, 5, 33]}_{\text{Row 1}},\ \underbrace{[45, 32, 830, 120, 750]}_{\text{Row 2}},\ \underbrace{[4, 880, 45, 66, 61]}_{\text{Row 3}}\ ]$$

A list of lists

Rows • Columns

Building an r * c matrix

- data = [[0] * c] * r   #Wrong



point r # of times
↳ would change every row

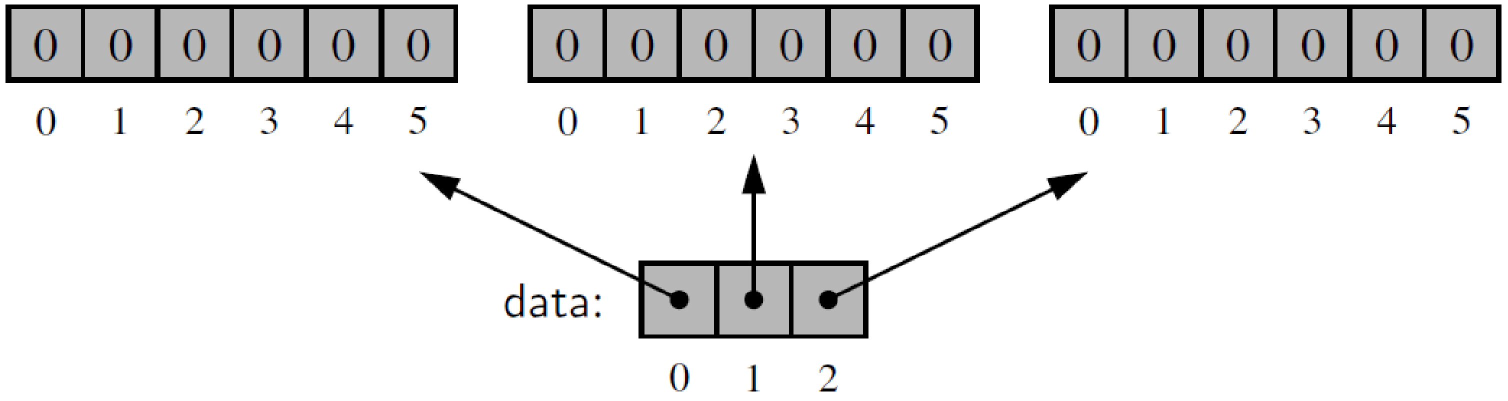
all *r* entries of the list known as data are references to the same instance of a list of *c* zeros

Valid Matrix

- data = [[0] * c for j in range(r)]

$\hookrightarrow$ Rows

multiple independent lists



each cell of the primary list refers to an *independent* instance of a secondary list