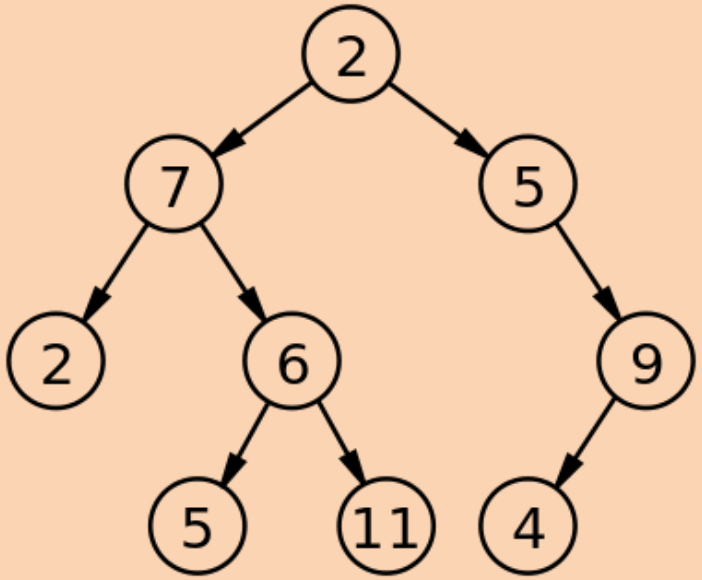
ساختمان‌های داده

مرتب‌سازی حبابی، درجی و انتخابی

Bubble Sort, Insertion Sort, and Selection Sort



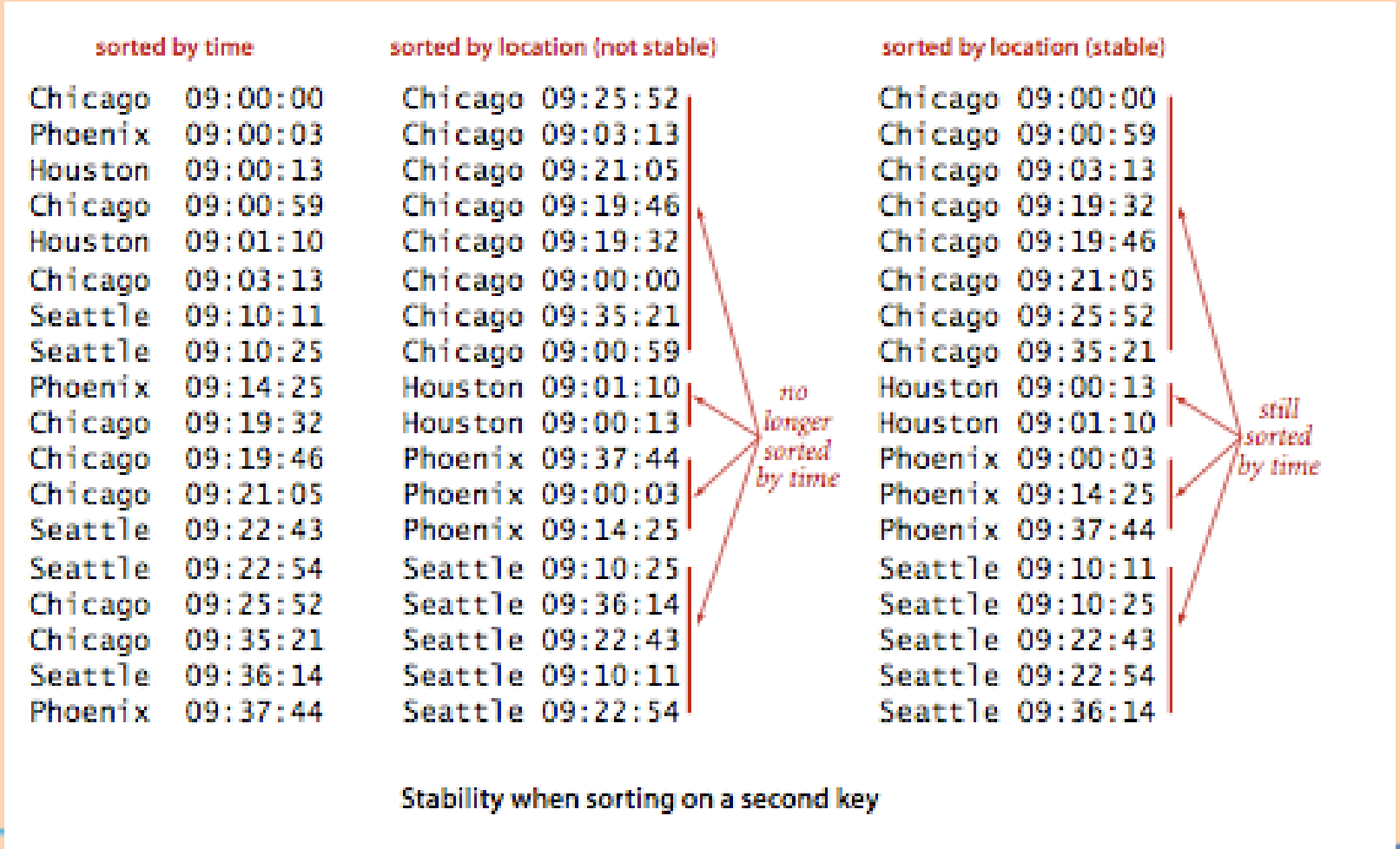
مدرس:

سید کمال الدین غیاثی شیرازی

مساله مرتب سازی

پایداری الگوریتم‌های مرتب سازی

- یک الگوریتم مرتب سازی پایدار (stable) است هرگاه ترتیب عناصر مساوی را به هم نزند.



Stability when sorting on a second key

مرتب‌سازی حبابی (Bubble Sort)

- الگوریتم مرتب‌سازی حبابی دا ئما دو عنصر مجاور را با یکدیگر مقایسه می‌کند و اگر ترتیب آنها صحیح نیست، جای آنها را عوض می‌کند.

مرتب‌سازی حبابی (Bubble Sort)

```
n = length(A)
for i = n-1 down to 1 do
 for j = 1 to i do
   if A[j-1] < A[j] then
     swap(A[j-1], A[j])
```

نسخه‌ی بهینه‌شده مرتب‌سازی حبابی

```
n = length(A)
repeat
 newn = 0
 for i = 1 to n-1 do
   if A[i-1] < A[i] then
     swap(A[i-1], A[i])
       newn = i
n = newn
until n = 0
```

تحلیل الگوریتم Bubble sort

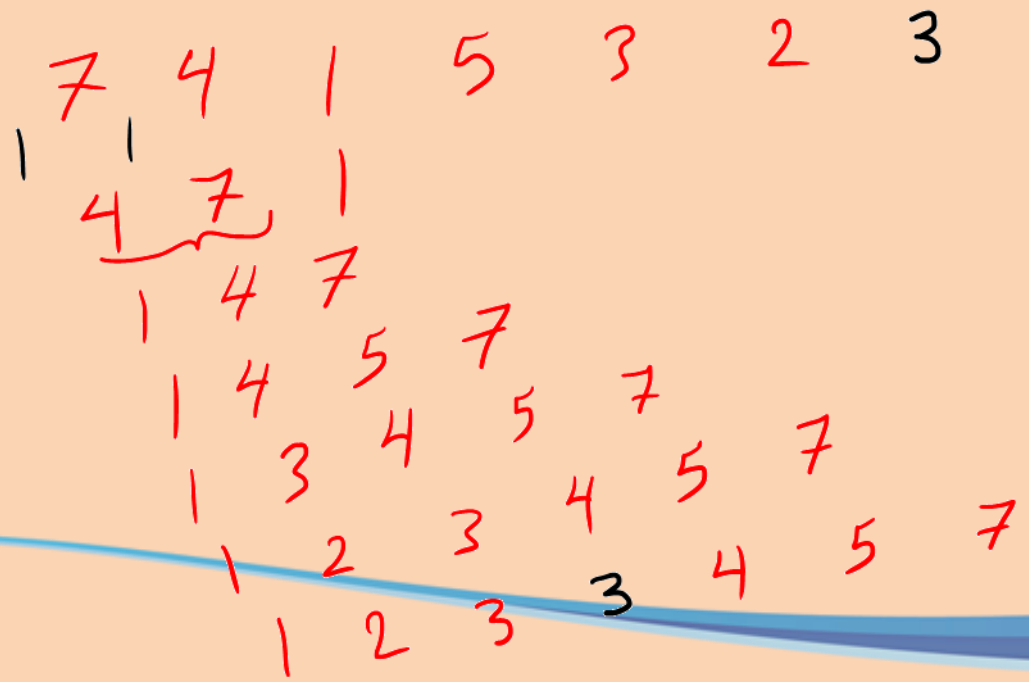
- زمان اجرا در بدترین حالت از مرتبه $\Theta(?)$
- پایدار؟
- تعداد دفعات جابجایی عناصر از مرتبه $\Theta(?)$ است.

تحلیل الگوریتم Bubble sort

- زمان اجرا در بدترین حالت از مرتبه $\Theta(n^2)$
- پایدار است.
- تعداد دفعات جابجایی عناصر از مرتبه $\Theta(n^2)$ است.
  - بنابراین اگر اندازه عناصر لیست بر حسب بایت بالا باشد، بسیار کند عمل خواهد کرد.

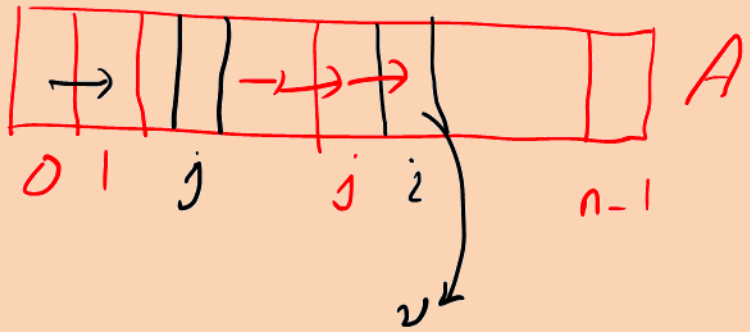
مرتب‌سازی درجی (Insertion Sort)

- الگوریتم Insertion sort همانند اثبات استقرایی عمل می‌کند.
- آرایه ای به طول یک مرتب است.
- فرض کنیم آرایه به طول $j-1$ مرتب باشد، در این صورت عنصر j ام را در محل صحیح در آرایه $j-1$ تایی درج می‌کنیم تا یک آرایه j تایی مرتب بوجود آید.



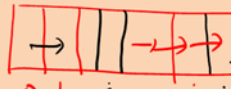
الگوریتم مرتب‌سازی درجی

```
n = length(A)
for i=1 to n-1 do
    v = A[i]
    j = i-1
    while j >= 0 and v < A[j] do
        A[j+1] = A[j]
        j = j - 1
    A[j+1] = v
```



$$\sum_{i=1}^{n-1} i = 1 + 2 + \cdots + n-1 = \frac{(n-1)n}{2} \in \theta(n^2)$$

```
n = length(A)
for i=1 to n-1 do
    v = A[i]
    j = i-1
    while j >= 0 and v < A[j] do
        A[j+1] = A[j]
        j = j - 1
    A[j+1] = v
```



n = len(A)

for i=1 to n-1 do

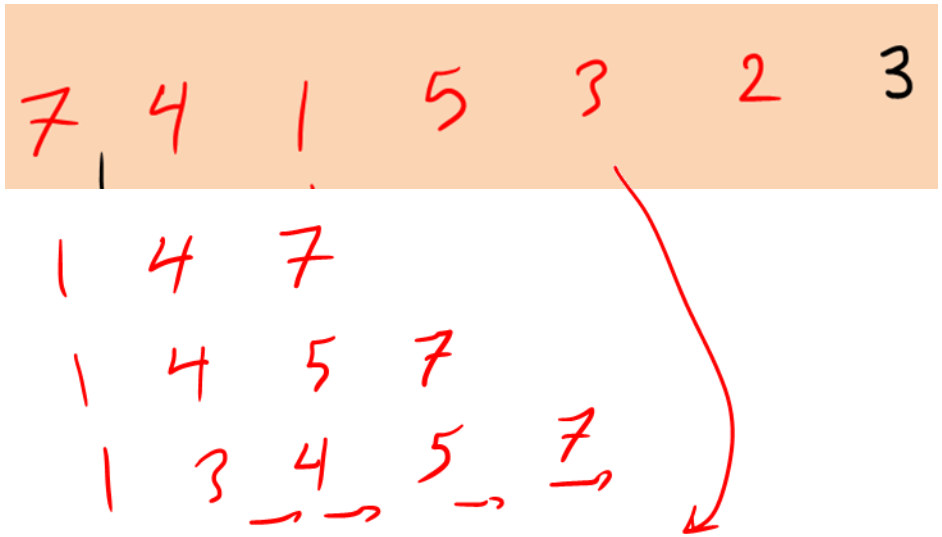
j = i-1

while j ≥ 0 and A[j+1] < A[j] do

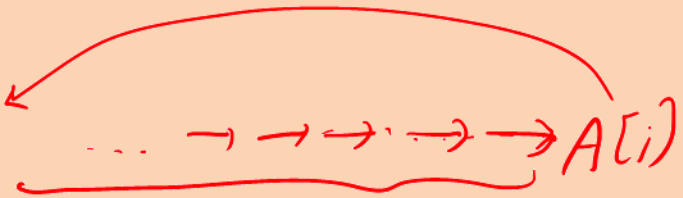
A[j+1] ⟷ A[j]

j = j-1

A[j+1] = v



تحلیل الگوریتم Insertion sort



- زمان اجرا در بدترین حالت از مرتبه $\Theta(?)$ n^2
- پایدار؟
- تعداد دفعات جابجایی عناصر از مرتبه $\Theta(?)$ n^2 است.

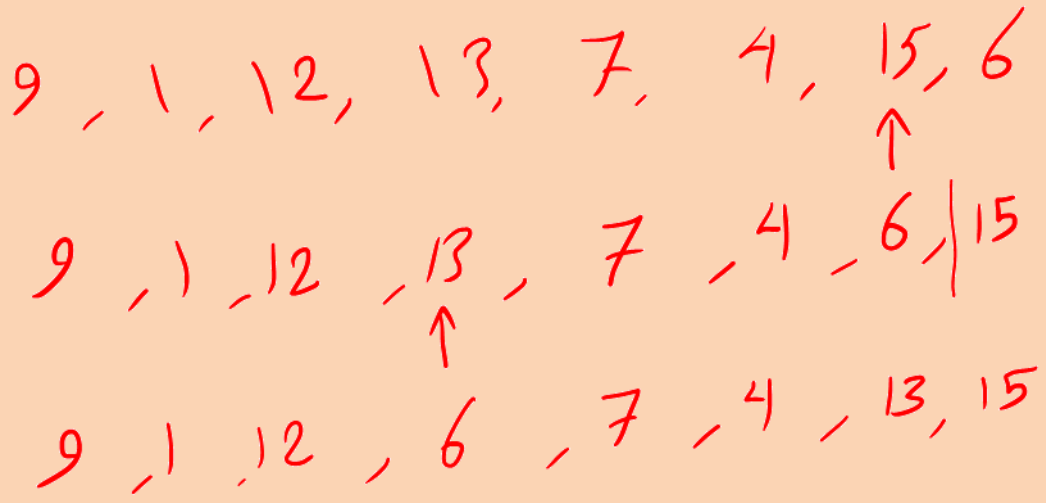
$n, n-1, n-2, \ldots, 1$

تحلیل الگوریتم Insertion sort

- زمان اجرا در بدترین حالت از مرتبه $\Theta(n^2)$
- پایدار است.
- تعداد دفعات جابجایی عناصر از مرتبه $\Theta(n^2)$ است.
  - بنابراین اگر اندازه عناصر لیست بر حسب بایت بالا باشد، بسیار کند عمل خواهد کرد.

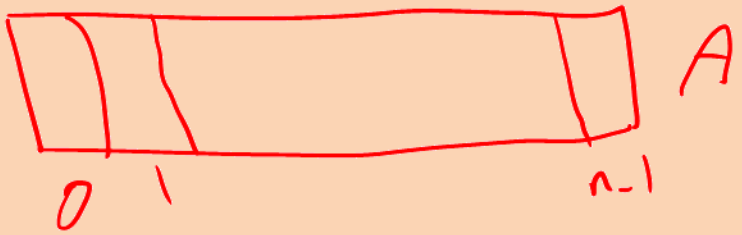
مرتب‌سازی انتخابی (Selection Sort)

- در Selection Sort محل بزرگ ترین عنصر در لیست عناصر مشخص می شود و جای آن با عنصر آخر لیست عوض می شود.
- با تکرار همین کار، لیست مرتب می شود.



الگوریتم مرتب‌سازی انتخابی

```
n = length(A)
for i=1 to n-1 do
    maxVal = A[0]
    maxIdx = 0
    for j=1 to n-i do
        if maxVal < A[j]
            maxVal = A[j]
            maxIdx = j
    swap (A[n-i], A[maxIdx])
```



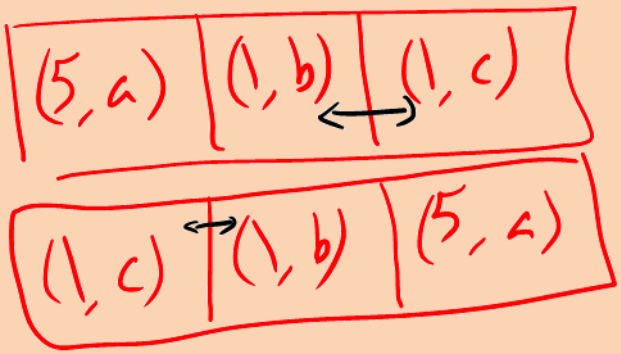
$$\sum_{i=1}^{n-1}(n-i) = n-1+\cdots+1 = \frac{(n-1)n}{2} \in \Theta(n^2)$$

مستقل از ترتیب داده‌ها

جابجایی‌ها = $n-1$

تحلیل الگوریتم مرتب‌سازی انتخابی

- زمان اجرا در بدترین حالت از مرتبه $\Theta(n^2)$
- پایدار؟ ✗
- تعداد دفعات جابجایی عناصر از مرتبه $\Theta(n)$ است.



تحلیل الگوریتم مرتب‌سازی انتخابی

- زمان اجرا در بدترین حالت از مرتبه $\Theta(n^2)$
- پایدار نیست. مثال زیر ناپایداری را نشان می‌دهد.

$$(5, a), (1, b), (1, c)$$

- تعداد دفعات جابجایی عناصر از مرتبه $\Theta(n)$ است. بنابراین اگر اندازه عناصر لیست بر حسب بایت بالا باشد، این الگوریتم نسبت به دو الگوریتم قبل رجحان دارد.