حلقه بیرونی (LI1) :

حقیقت ناوردایی : در مرحله i-ام، i عنصر ابتدایی A یعنی $A[1..i]$ مرتب می‌شود و لازم به ذکر است که این زیرآرایه، دقیقاً همانی است که در خروجی به همین ترتیب ظاهر می‌شود.

این جمله نادقیق است و ذکر آن ممکن است سبب ابهام شود ← در گام آغاز

بررسی گام‌های ناوردایی حلقه ←
- گام آغاز،
- گام نگهداری،
- گام پایان.

* تحلیل کارایی الگوریتم مرتب‌سازی حبابی :

در جلسات قبل دیدیم که هزینه هر خط از الگوریتم که یک دور ثابت است، تأثیری در تحلیل کارایی در حالت کلی ندارد. ← بنابراین هزینه هر خط از الگوریتم را $O(1)$ در نظر می‌گیریم.

$i=1 \longrightarrow 3(n-1)$ // $j = A..2 \rightarrow n-2+1 = n-1$

$i=2 \longrightarrow 3(n-2)$ // $j = n..3 \rightarrow n-3+1 = n-2$

$\vdots$

$i=n-1 \longrightarrow 3(1)$ // $j = n..n-1 \rightarrow n-n+1$

$$T(n) = \sum_{i=1}^{n-1} 3(n-i) = \theta(n^2)$$

سوال: پیچیدگی الگوریتم مرتب سازی حبابی در حالت متوسط، بهترین حالت، بدترین حالت چیست؟

همانطور که از الگوریتم مرتب سازی حبابی قابل مشاهده است، این الگوریتم بدون توجه به محتوای آرایه عمل می‌کند ← بنابراین هر سه حالت $\theta(n^2)$ می‌باشد.

یادآوری: الگوریتم مرتب سازی درجی دارای پیچیدگی:
- حالت متوسط $\theta(n^2)$
- بدترین حالت $\theta(n^2)$
- بهترین حالت $\theta(n)$

مثلا برای تفاوت زمان اجرا در $\theta(n^2)$ و $\theta(n\log n)$: ← بهترین الگوریتم مرتب ساز

فرض کنید $n=10^6$ و هر عمل 10^{-6} ثانیه نیاز دارد.

$$n^2 \rightarrow (10^6)^2 \times 10^{-6} = 10^6\ s \approx 12 \text{ روز}$$

$$n\log n \rightarrow 10^6 \log 10^6 \times 10^{-6} = 6\log 10 = 6 \text{ ثانیه}$$

مثلا رابطه بازگشتی در الگوریتم مرتب سازی حبابی:

$$T(n) = T(n-1) + 3(n-1)$$
$$\hookrightarrow T(n-2) + 3(n-2)$$
$$\vdots$$

روش تقسیم و حل (divide and conquer):

ساختار بسیاری از الگوریتم‌ها بازگشتی است ← یعنی در درون مسأله، خود را فراخوانی می‌کند. (روال حل)

Recursive

این الگوریتم‌ها در واقع از روش تقسیم و حل پیروی می‌کنند که شامل سه مرحله زیر است:

- تقسیم (divide): مسأله به تعدادی زیرمسأله تقسیم می‌شود.
- حل (conquer): زیرمسأله‌ها به صورت بازگشتی حل می‌شوند.
- ترکیب (combine): جواب زیرمسأله‌ها با هم ترکیب و مسأله‌ی اولیه حل می‌شود.

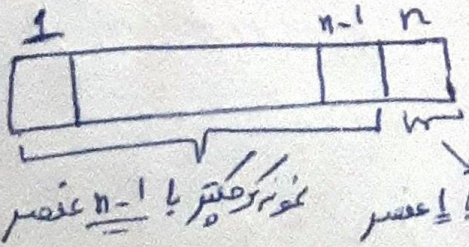
نکاتی در رابطه با تقسیم (گام تقسیم) (divide):

- تقسیم را تا کجا ادامه دهیم ← نمونه‌های کوچک: مثلاً $n \leq 1$ عنصر یا $n \leq 10$ عنصر و...
  نمونه‌های بزرگ: تقسیم به نمونه‌های کوچکتر k تایی $k = 2, 3, 4, \ldots$ مثلاً مساوی

- الگوریتم‌های ارائه شده با روش تقسیم و غلبه معمولاً زمانی بهترین پیچیدگی زمانی را دارند که نمونه‌های بزرگ به نمونه‌های کوچکتر با اندازه تقریباً یکسان تقسیم شوند.

مثال: مرتب سازی درجی (Insertion-Sort):

آرایه n عنصری A را در نظر بگیرید:



بنابراین در هر مرحله 2 تقسیم انجام می‌دهیم ← یک تقسیم با n-1 عنصر، یک تقسیم با 1 عنصر.

الگوریتم مرتب سازی درجی ← تکراری (Iterative) ← همانی که در جلسات قبلی تشریح شده.

بازگشتی (Recursive) ← با الگو گرفتن از ایده صفحه قبل به صورت زیر است.

حالت تکراری

↓

for m=2 to n do

```
R-Insertion-Sort( A[1..n] , m)
1.    if (m<=1) then
2.        return
3.    R-Insertion-Sort(A[1..n], m-1)
4.    key ← A[m]
5.    i ← m
6.    while i>1 and A[i-1] >key do
7.        A[i] ← A[i-1]
8.        i ← i-1
9.    A[i] ← key
```

سوال: اثبات درستی الگوریتم فوق به چه نحو قابل انجام است؟

R-Insertion-Sort(A[1..n], n) ← فراخوانی اولیه الگوریتم

سوال: تحلیل پیچیدگی الگوریتم فوق به چه نحو قابل محاسبه است؟

* اثبات درستی الگوریتم مرتب سازی درجی - نسخه بازگشتی :

* خط ۶ تا ۸ الگوریتم شبه قبل یا استفاده از مفهوم Loop Invariant قابل اثبات است.

* برای کل الگوریتم هم از اثبات استقرایی استفاده می‌کنیم

پایه استقرا: آرایه ۱ عضوی یک آرایه مرتب است.

گام استقرا: $k-1$ عضو اول آرایه مرتب است + درستی خط ۶ تا ۸ الگوریتم ← k عضو اول آرایه مرتب است.

* فرم کلی برای محاسبه پیچیدگی زمانی الگوریتم های بازگشتی :

$$T(n) = a\,T(n/b) + D(n) + C(n)$$

$D(n)$: Divide, $C(n)$: Combination

$$T(1) = c$$ ثابت

جایگیه :

$D(n)$: زمان اجرای مرحله تقسیم برای n عضر است،

$C(n)$: زمان اجرای مرحله ترکیب برای n عضر است،

ضریب a : بیانگر تعداد زیرمساله هاست که باید حل شود،

ضریب $1/b$: بیانگر آن است که اندازه زیرمساله، $1/b$ اندازه مساله اصلی است.

* تحلیل پیچیدگی الگوریتم مرتب سازی درجی - نسخه بازگشتی :

$$T(n) = a\,T(n/b) + \underbrace{D(n)}_{O(1)} + \underbrace{C(n)}_{O(n)}$$

$a = 1$

$b = \frac{n}{n-1}$

$$\left.\begin{array}{l} T(n) = T(n-1) + O(n) \\ \Rightarrow T(n) = n * O(n) = O(n^2) \end{array}\right\}$$