↑ چون ما به دنبال حل یک مجموعه از نمونه ها هستیم نه یک نمونه خاص

(۴) نکته: در تحلیل الگوریتم ها ← بهترین حالت فاقد اهمیت است و بدترین حالت دارای اهمیت است چرا که یک Bound بالا را نشان می دهد برای کلیه ورودی ها و ما مطمئن می شویم که بدتر از این زمان نیست.

(۵) نحوهٔ محاسبه کارایی الگوریتم مرتب سازی درجی در حالت متوسط،

در واقع در حالت متوسط زمان اجرا را تک تک ورودی ها (حالت حال) را سنجیده می گیریم و سپس میانگین آنها را بدست می آوریم

آرایه شامل n عنصر باشد ما $n!$ جایگشت از آنها خواهیم داشت ← نکته: کاری به محتوی عناصر نداریم، تمرکز روی ترتیب قرارگیری است چرا که در الگوریتم ما این مهم است.

پس داریم:

$$\delta_1 \rightarrow T(\delta_1)$$
$$\vdots$$
$$\delta_{n!} \rightarrow T(\delta_{n!})$$

$$\bar{T}(n) = \text{میانگین} \leftarrow \frac{\sum_{i=1}^{n!} T(\delta_i)}{n!}$$

برای راحتی کار با فرم کلی میانگین را محاسبه می کنیم، یعنی:

$$T(n) = An + B + C \sum_{k=2}^{n} t_k$$

$$\text{میانگین } \bar{T}(n) = \frac{\sum_{i=1}^{n!} T(\delta_i)}{n!}$$

$$T'(n) = \frac{\sum_{i=1}^{n!} T(\delta_i)}{n!} = \frac{\sum_{i=1}^{n!} \left(An + B + c\sum_{k=2}^{n} t_{k,i}\right)}{n!}$$

$$= An + B + \frac{C}{n!} \sum_{i=1}^{n!} \sum_{k=2}^{n} t_{k,i}$$

نکته:

$$\sum_{j=1}^{n} \sum_{i=1}^{m} a_{ij} = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}$$

← دو تا سیگمای تودرتو داشته باشیم می‌توانیم جابجا کنیم

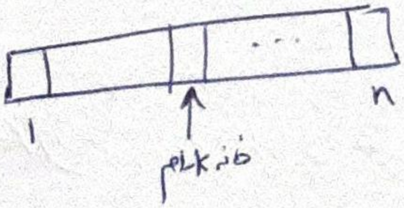
بنابراین داریم:

$$T'(n) = An + B + \frac{C}{n!} \sum_{k=2}^{n} \sum_{i=1}^{n!} t_{k,i}$$

$$= An + B + C \sum_{k=2}^{n} \left(\frac{\sum_{i=1}^{n!} t_{k,i}}{n!} \right)$$

متوسط هزینه‌ای که برای خانه kام پرداخت می‌کنیم

ادعا: این مقدار برابر $\frac{k+1}{2}$



$$\frac{1 + 2 + \cdots + k}{k} = \frac{k(k+1)}{2k} = \left(\frac{k+1}{2}\right)$$

بنابراین داریم:

$$T'(n) = An + B + C \sum_{k=2}^{n} \frac{k+1}{2}$$

$$= An + B + \frac{C}{2} \sum_{k=2}^{n} k+1 = An + B + \frac{C}{2} \left(\frac{n(n+1)}{2} - 1 + n - 2 + 1 \right)$$

$$= An + B + \frac{C}{2} \left(\frac{n(n+1)}{2} + n - 2 \right)$$

بنابراین میانگین هم از درجه ۲ است. ←

* آیا می‌شود الگوریتم Insertion-sort را سریعتر انجام داد؟

آرایه A: ۸, ۲, ۴, ۹, ۳, ۶, ۳

مرحله ۶: ۲, ۳, ۴, ۸, ۹, ۶, ۳

(۲, ۳, ۴, ۸, ۹ مرتب است)

به طور کلی در هر مرحله k ام، k-1 عنصر ابتدای آرایه مرتب است

→ استفاده از جستجوی دودویی یا Binary search

↓

اگر k-1 عنصر داشته باشیم، در زمان log (k-1) جای عنصری که باید درج شود را پیدا می‌کنیم و نیاز به زمان k (جستجوی خطی) نیست.

→ نکته مهم، در صورت استفاده از Binary search باز هم باید هزینه زمانی k را بپردازیم

→ به خاطر شیفت دادن عناصر برای درج عنصر جدید

یک راه حل استفاده از داده ساختار جدیدی است که شیفت دادن عناصر را به صورت مناسبی مدیریت کند که پیچیدگی زمانی افزایش نیابد ← استفاده از درخت متوازن

رشد توابع
Growth of Functions

یک مثال برای نمود در رابطه با

- ضرایب ثابت در پیچیدگی زمانی اهمیتی ندارند
- مهم خطی بودن، درجه دوم، نمایی بودن و ... است
- علت تعریف پیچیدگی مجانبی

* فرض کنید پردازنده با قدرت پردازش ۱۰۰ میلیون دستور در ثانیه زمان برای اجرای الگوریتم های مختلف در اختیار ما قرار دارد. همچنین فرض کنید هر واحد دستور در تابع مربوط به پیچیدگی زمانی 1s زمان برای اجرا نیاز دارد.

الگوریتم	T(n)	A	B	C	B/A	C/A
A_1	$100n$	10	100	1000	10	100
A_2	$5n^2$	15	45	142	3	9,4
A_3	$\frac{n^3}{2}$	13	28	59	2,15	4,5
A_4	2^n	10	14	17	1,4	1,7

۱۰ برابر سریعتر — ۱۰۰ برابر سریعتر

خطی هم است

A: حداکثر اندازه ورودی که پردازنده مفروض در زمان اجرای ۱۰۰۰ ثانیه

B: حداکثر اندازه ورودی که پردازنده ۱۰ برابر سریعتر در زمان اجرای ۱۰۰۰ ثانیه

C: حداکثر اندازه ورودی که پردازنده ۱۰۰ برابر سریعتر در زمان اجرای ۱۰۰۰ ثانیه

مثال عددی برای درک زمان اجرای یک برنامه:

الگوریتم A1 برای $n=100$ ← $T(n)=100n=10000S$

خطی

معادل 167 دقیقه

۲ ساعت و ۴۷ دقیقه

الگوریتم A4 برای $n=100$ ← $T(n)=2^n=2^{100}=$ (illegible)

نمایی

$(2^{10})^{10} \simeq (10^3)^{10} S = 10^{30} S = 10^{20}$ سال

یک معادل سازی با دنیای واقعی ← سن جهان $(13772)\times 10^9$ (حدود ۱۴ هزار میلیارد سال)

↳ یکسال ۳۱٫۵۳۶٫۰۰۰ ثانیه است در بالا فرض کردیم 10^{10} ثانیه یکسال است.

جمع بندی: مقایسه A1, A1 ← A4 ممکن است برای ورودی های کوچک خوب عمل کند ولی برای ورودی های بزرگ این گونه عمل نمی کند.

$(A_1, A_4) = \{ \underbrace{(500, 32)}_{n=5}, \underbrace{(1000, 1024)}_{n=10}, \underbrace{(1500, 32768)}_{T(n)=2^n \quad n=15}, \dots \}$