دانشکده علوم ریاضی و آمار

**مدرس: دکتر مجتبی رفیعی** نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

# جلسه ۹: تحلیل کارایی الگوریتم‌ها

نگارنده: زهرا سادات میرطالبی

۲۲ مهر ۱۴۰۰

## فهرست مطالب



## ۱ تحلیل کارایی الگوریتم‌ها- حالت متوسط

**یادآوری:** در تحلیل الگوریتم‌ها بهترین حالت فاقد اهمیت است و در مقابل بدترین حالت دارای اهمیت است چرا که یک کران (Bound) بالا به ما معرفی می‌کند و مطمئن هستیم برای کلیه ورودی‌ها بدتر از این زمان رخ نمی‌دهد.
**نحوه‌ی محاسبه کارایی الگوریتم مرتب‌سازی درجی در حالت متوسط:** در واقع در حالت متوسط زمان اجرای تک تک نمونه‌های ورودی‌ها را محاسبه می‌کنیم و سپس میانگین آن‌ها را بدست می‌آوریم. به عبارت دیگر،

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

$$\text{میانگین} = T^{'}(n) = \frac{\sum_{i=1}^{n!} T(\delta_i)}{n!}$$

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

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

$$\text{میانگین} = T^{'}(n) = \frac{\sum_{i=1}^{n!} T(\delta_i)}{n!}$$

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

**نکته**

**اگر دو سیگمای تودرتو داشته باشیم می‌توانیم آن‌ها را با هم جابه‌جا کنیم.**

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

بنابراین داریم:

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

که در آن $\frac{\sum_{i=1}^{n!} t_{k,i}}{n!}$ برابر با متوسط هزینه‌ای است که برای خانه $k$-ام پرداخت می‌کنیم.

**ادعا:** مقدار $\frac{\sum_{i=1}^{n!} t_{k,i}}{n!}$ برابر با $\frac{k+1}{2}$ است.

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

بنابراین داریم:

$$\begin{aligned} T^{'}(n) &= An + B + C\sum_{k=2}^{n}\frac{k+1}{2} \\ &= An + B + \frac{C}{2}\sum_{k=2}^{n} k + 1 = An + B + \frac{C}{2}\left(\frac{n(n+1)}{2} - 1 + n - 2 + 1\right) \\ &= An + B + \frac{C}{2}\left(\frac{n(n+1)}{2} + n - 2\right) \end{aligned}$$

سوال

**آیا می توان الگوریتم Insertion-Sort را سریعتر انجام داد؟**

$$A: 8,2,4,9,3,6,3$$ آرایه

$\downarrow$

مرحله ۶ $\underbrace{2,3,4,8,9}_{\text{مرتب است}},6,3$

به‌طور کلی در مرحله $k$-ام، $k-1$ عنصر ابتدای آرایه مرتب است.

$\Downarrow$

استفاده از جستجوی دودویی یا $(Binary\ Search)$

$\downarrow$

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 (به‌خاطر 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 که این راه حل استفاده از درخت توازن است که در جلسات آتی به آن پرداخته خواهد شد .

در این بخش با یک مثال نکات زیر را در رابطه با رشد توابع بررسی خواهیم کرد.

- ضرایب ثابت در پیچیدگی زمانی اهمیتی ندارند زیرا می‌توان با افزایش تعداد پردازنده آن‌ها را بی‌اثر کرد.
- خطی بودن، درجه دوم، نمایی بودن و... علت تعریف پیچیدگی زمانی مجانبی هستند .

## ۲ رشد توابع (Growth of Functions)

**مثال:** فرض کنید یک پردازنده با قدرت پردازشی مشخص و 1000ثانیه زمان برای اجرای الگوریتم های مختلف در اختیار داریم . همچنین فرض کنید که تابع T(n) بیانگر تعداد واحدهای عملیاتی موررد نیاز الگوریتم است، و هر واحد عملیاتی نیز فرض شده است که در 1 ثانیه قابل انجام است.

جدول ۱: رابطه پیچیدگی زمانی، اندازه ورودی و قدرت پردازشی

| الگوریتم | T(n) | A | B | C | B/A (10 برابر سریعتر) | C/A (100 برابر سریعتر) |
|---|---|---|---|---|---|---|
| $A1$ | $100n$ | 10 | 100 | 1000 | 10 | 100 |
| $A2$ | $5n^2$ | 15 | 45 | 142 | 3 | 9.4 |
| $A3$ | $\frac{n^3}{2}$ | 13 | 28 | 59 | 2.15 | 4.5 |
| $A4$ | $2^n$ | 10 | 14 | 17 | 1.4 | 1.7 |

A: حداکثر اندازه ورودی روی پردازنده مفروض و زمان اجرای 1000 ثانیه.
B: حداکثر اندازه ورودی روی پردازنده 10 برابر سریعتر و زمان اجرای 1000 ثانیه.
C: حداکثر اندازه ورودی روی پردازنده 100 برابر سریعتر و زمان اجرای 1000 ثانیه.

**مثال برای درک زمان اجرای یک برنامه:**

الگوریتم $A1$ برای $n = 100$

$$T(n) = 100n = 10000S$$

که 10000 ثانیه معادل است با 167دقیقه، یعنی 2 ساعت و 47 دقیقه.

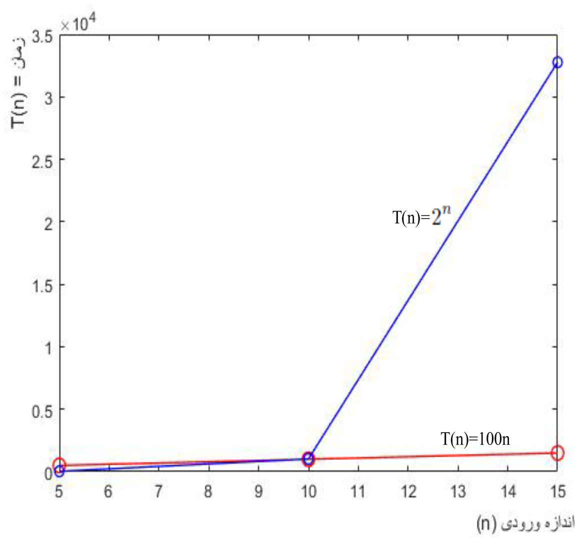
الگوریتم $A4$ برای $n = 100$

$$T(n) = 2^n = 2^{100} = (2^{10})^{10} \approx (10^3)^{10}S = 10^{30}S = 10^{20} \text{ سال}$$

یک سال $31,536,000$ ثانیه است که بالا فرض کردیم $10^{10}$ ثانیه یک سال است.
یک معادل‌سازی واقعی سن جهان $10^9 \times (13772)$ حدود 14 هزار میلیارد سال.

**جمع‌بندی**: در مقایسه $A1$ و $A4$، اگر چه $A4$ ممکن است برای نمونه های کوچک خوب عمل کند ولی برای هایn بزرگ این گونه عمل نمی‌کند برای همین است که گاهی الگوریتم ها را برای نمونه های مختلف ورودی آزمایش می‌کنند. شکل 1 مقایسه الگوریتم های $A1$ و $A4$ را برای چند نمونه کاندید به تصویر کشیده است.

$$(A1, A4) = \{\underbrace{(500, 32)}_{n=5}, \underbrace{(1000, 1024)}_{n=10}, \underbrace{(1500, 32768)}_{n=15}, \cdots\}$$




شکل ۱: رابطه پیچیدگی زمانی و اندازه ورودی برای دو الگوریتم کاندید