دانشکده علوم ریاضی

ساختمان داده‌ها

نیم‌سال دوم ۱۳۹۹-۱۴۰۰

مدرس: حسین بومری

جلسه‌ی چهارم

نگارنده: فاطمه عظیمی

در جلسه‌ی گذشته تعداد دستورهای الگوریتم زیر و ارتباط آن با زمان اجرا بررسی شد:

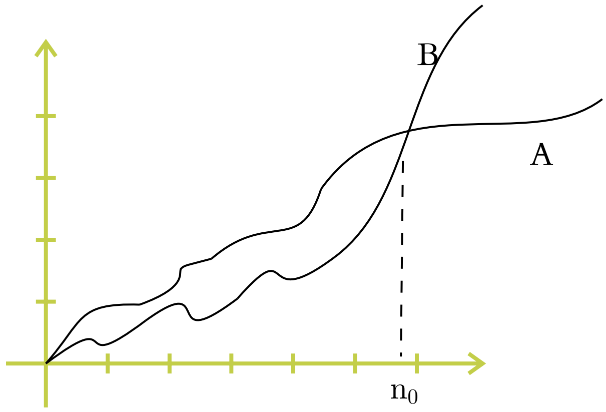
```
1. Input A[1000]
2. for i=1 to 100
3.     m=i
4.     for j=i+1 to 1000
5.         if A[m]<A[j]
6.             m=j
7.     swap(A[m], A[i])
8. print A[1:100]
```

به این صورت که در خط اول 1000 عملیات memory داریم.
خط دوم، 100 عملیات از جنس for دارد که شامل ۱۰۰ تساوی، ۱۰۰ جمع و ۱۰۰ مقایسه است.
خط سوم ۱۰۰ عملیات تساوی دارد.
در خط چهارم به صورت تقریبی (999+1000)50 تا عملیات از جنس for داریم که شامل همین تعداد تساوی، جمع و مقایسه است.
خط پنجم هم به صورت تقریبی (999+1000)50 عملیات مقایسه دارد.
در خط ششم با توجه به وضعیت اولیه ورودی‌ها حداکثر (999+1000)50 عملیات تساوی داریم و حداقل 0 بار اجرا می‌شود.
خط هفتم نیز ۱۰۰ بار اتفاق می‌افتد که می‌تواند با 3*100 عملیات تساوی و یک متغییر اضافی به دست آید.
خط هشتم نیز، ۱۰۰ عملیات print دارد.
البته تحلیل می‌تواند از این دقیق‌تر شود و دسترسی به حافظه‌ها را نیز در نظر بگیریم.

زمان اجرای الگوریتم:

برای اینکه بگوییم برنامه ما حدودا چقدر طول می‌کشد یک روش این است که عملیات‌ها را جداگانه در نظر بگیریم و با توجه به تکرار، زمان نهایی را محاسبه کنیم. اگر چه دقیق است اما لزوما خوب نیست؛ جراکه تحلیل پیچیده‌ای می‌شود و زمان هر عملیات ثابت نیست و ممکن است در سیستم‌ها یا زبان‌های برنامه‌نویسی مختلف تغییر کند. از طرف دیگر عملیات‌های ما در ضریب تفاوت دارند و همانطور که جلسه‌ی قبل هم گفته شد اگر تفاوت دو الگوریتم در ضریب باشد، با سیستم سریع‌تر می‌شود ضریب را جبران کرد؛ اما اگر اختلاف ضریب نباشد هرچقدر هم که سیستم سریع‌تری بیاوریم از جایی به بعد نمی‌توان جلوی اختلاف را گرفت. البته باید حواسمان باشد که نمی‌توان گفت این ضریب هرچقدر باشد اشکالی ندارد و اگر با توجه به محدوده‌ی ورودی‌هایمان مناسب نباشد، تحلیل ما را ضعیف می کند. اما وقتی اختلاف دو الگوریتم در یک ضریب است برایش راه‌حلی وجود دارد اما وقتی اختلاف در *"مرتبه‌ی زمانی"* باشد هیج ضریبی نمیتواند اختلاف دو تابع را جبران کند و اختلاف تابع‌های مقدار زمانی از هر ضریبی بیشتر است. که در ادامه به صورت دقیق‌تر به آن می‌پردازیم:

مرتبه‌ی زمانی:

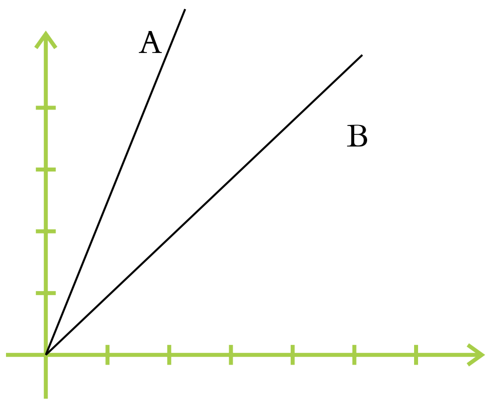


زمانی می‌گوییم $B \leq A$ که:

$$\exists n_0 \mid \forall\ x,\ x>n_0 :\ A(x) \leq B(x)$$

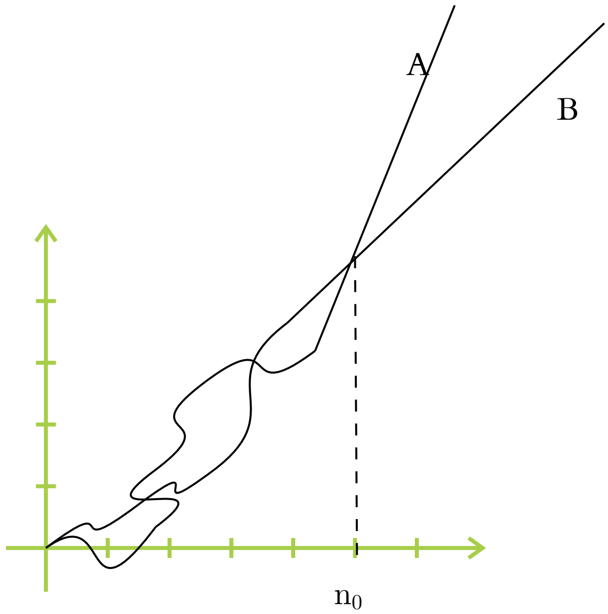
که یعنی از جایی به بعد هر چقدر آرگومان‌های مسئله بزرگتر شود مقادیر تابع A بیشتر از مقادیر تابع B نمی‌شود. برای مثال برای توابع n^2 و $100n$، چنین n_0 ای وجود دارد و مقدارش برابر ۱۰۰ خواهد بود.
دیدیم که اختلاف ضریب هم قابل جبران است پس تعریف بهتری برای $B \leq A$ وجود دارد...

نمودار زیر را در نظر بگیرید:



که $B=x$ و $A=2x$. در این جا هم می‌گوییم A بدتر از B نیست!
پس به سراغ تعریف بهتر می‌رویم!

نمودار زیر را در نظر بگیرید که از جایی به بعد توابع x و ۲x هستند:

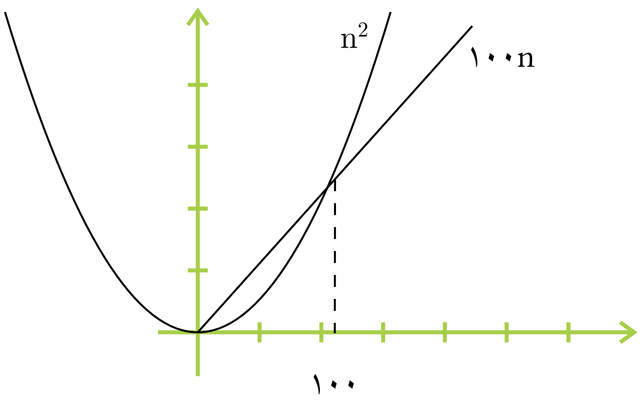


و $A \leq B$ را به صورت زیر تعریف می‌کنیم:

$\exists\ n_0, c \mid \forall x > n_0 :\ cA(x) \leq B(x)$

و می‌نویسیم: $A \in O(B)$

برای مثال $100n \in O(n^2)$:



در مرتبه‌ی زمانی می‌توانیم از n_0 به قبل را فراموش کنیم! و چیز دیگری که معمولا در اردر در نظر می‌گیریم، مقادیر ثابت است. یعنی می‌توانیم مرتبه‌ی زمانی را به صورت زیر نیز تعریف کنیم:

$\exists\ n_0, c_1, c_2 \mid \forall x > n_0 :\ c_1A(x) - c_2 \leq B(x)$

و این تعریف مثلا برای توابعی که از جایی به بعد 0 و 2 هستند به کارمان می‌آید! که از یک مرتبه‌ی زمانی هستند.

جال مرتبه‌ی زمانی الگوریتم زیر را حساب می کنیم:

```
1. Input A[n]
2. for i=1 to 100
3.      m=i
4.      for j=i+1 to n
5.           if A[m]<A[j]
6.                  m=j
7.      swap(A[m], A[i])
8. print A[1:100]
```

و با توجه به مطالب گفته شده درباره‌ی تعداد عملیات هر خط به ترتیب زیر طول می‌کشند:

1. n
2. 3*100
3. 100
4. 3*(50(n+n-100))
5. 50(n+n-100)
6. max: 50(n+n-100), min: 0
7. 3*100
8. 100

که با جمع مقادیر، متوجه می‌شویم، مرتبه‌ی زمانی آن از اردر n است.

حال مرتبه‌ی زمانی الگوریتمی که می‌خواهد همه‌ی اعداد را مرتب کند حساب می‌کنیم که الگوریتم به صورت زیر است:

```
1. Input A[n]
2. for i=1 to n
3.      m=i
4.      for j=i+1 to n
5.           if A[m]<A[j]
6.                  m=j
7.      swap(A[m], A[i])
8. print A[1:n]
```

خط اول n عملیات، خط دوم 3*n+1 عملیات، خط سوم n بار، خط چهارم 3n(n+1)/2 عملیات، خط پنجم n(n+1)/2 عملیات، خط ششم [max: n(n+1)/2, min: 0] عملیات، خط هفتم 3n عملیات و خط آخر نیز n عملیات خواهد داشت که در مجموع، $27/2n+9/2n^2+1$ عملیات می‌شود. که از اردر n نیست. اما برای توان‌های بزگتر مساوی 2 از اردر n^k هست.

در پرانتز راه‌هایی برای swap :):

روش اول(با متغییر کمکی):

$t = a$
$a = b$
$b = a$

روش دوم:

$a = a + b$
$b = a - b$
$a = a - b$

روش سوم:

$a = aXORb$
$b = aXORb$
$a = aXORb$

که روش سوم، سریع‌تر است؛ چون XOR عملیات پایه است! و carry نیز در عملیات ظاهر نمی‌شود که باعث می‌شود بتوان آن را به طور موازی اجرا کرد!

تعریف تابع O را دیدیم که در واقع برای حد بالای توابع است. حال به همین صورت تابعی برای حد پایین تعریف می‌کنیم به نام Ω.
زمانی می‌گوییم $\mathrm{B} \in \Omega(\mathrm{A})$ که:

$\exists\ \mathrm{n_0,c_1,c_2} \mid \forall \mathrm{x{>}n_0} :\ \mathrm{c_1B(x)+c_2} \geq \mathrm{A(x)}$

و توابع Ω و O منجر به تعریف تابع Θ می‌شوند...

زمانی می‌گوییم $\mathrm{B} \in \Theta(\mathrm{A})$ که:

$A \in \Omega(B) \wedge A \in O(B)$