

تحليل الكوريتةها

سید ناصر رضوی n.razavi@tabrizu.ac.ir

۱۳۹۵

دلایل تحلیل الگوریتم‌ها

۲

□ دلایل تحلیل الگوریتم‌ها.

□ پیش‌بینی کارایی

□ مقایسه‌ی الگوریتم‌ها

□ تضمین کارایی

□ دلیل عملی.

□ اجتناب از خطاهای کارایی

□ آیا برنامه‌ی من می‌تواند یک مسئله با اندازه‌ی ورودی **بزرگ** را حل کند؟

چرا برنامه‌ی من این قدر کند است؟

چرا برنامه‌ی من این قدر حافظه مصرف می‌کند؟



□ **نکته‌ی اصلی.** [کنوت دهه‌ی ۷۰] برای درک کارایی از **روش علمی** استفاده کنید.

استفاده از روش علمی برای تحلیل الگوریتم‌ها

۴

- یک چارچوب به منظور پیش‌بینی کارایی و مقایسه‌ی الگوریتم‌ها.
- روش علمی (مشاهده-فرضیه-آزمون).
- مشاهده‌ی یک ویژگی از دنیای طبیعی
- فرضیه‌پردازی بر اساس مشاهدات انجام شده
- پیش‌بینی رویدادهای آتی با استفاده از فرضیه
- تصدیق فرضیه با انجام مشاهدات بیشتر
- اعتبارسنجی با تکرار مراحل فوق تا زمانی که فرضیه و مشاهدات با یکدیگر همخوانی پیدا کنند.
- اصول.
- آزمایش‌ها باید تکرارپذیر باشند.
- فرضیه‌ها باید انکارپذیر باشند.



مشاهدات

مثال: مسئله‌ی ۳-مجموع

۶

□ ۳-مجموع. با داشتن N عدد صحیح متمایز، چند سه‌تایی وجود دارد که مجموع‌شان دقیقاً برابر با صفر است؟

```
% more 8ints.txt
8
30 -40 -20 -10 40 0 10 5

% java ThreeSum 8ints.txt
4
```

	a[i]	a[j]	a[k]	sum
1	30	-40	10	0
2	30	-20	-10	0
3	-40	40	0	0
4	-10	0	10	0

۳- مجموعه: الگوریتم کورکورانه

۷

```
public class ThreeSum
{
    public static int count(int[] a)
    {
        int N = a.length;
        int count = 0;

        for (int i = 0; i < N; i++)
            for (int j = i + 1; j < N; j++)
                for (int k = j + 1; k < N; k++)
                    if (a[i] + a[j] + a[k] == 0)
                        count++;

        return count;
    }

    public static void main(String[] args)
    {
        int[] a = In.readInts(args[0]);
        StdOut.println(count(a));
    }
}
```

بررسی تمام سه تایی‌ها

اندازه‌گیری زمان اجرا

۹

- س. چگونه زمان اجرای یک برنامه را اندازه بگیریم؟
- ج. به صورت خودکار

```
public class Stopwatch
```

```
    Stopwatch()
```

ایجاد یک شیء زمان‌سنج

```
    double elapsedTime()
```

زمان سپری شده از لحظه‌ی ایجاد (بر حسب ثانیه)

```
public static void main(String[] args)
```

```
{
```

```
    int[] a = In.readInts(args[0]);
```

```
    Stopwatch timer = new Stopwatch();
```

```
    StdOut.println(ThreeSum.count(a));
```

```
    double time = timer.elapsedTime();
```

```
}
```

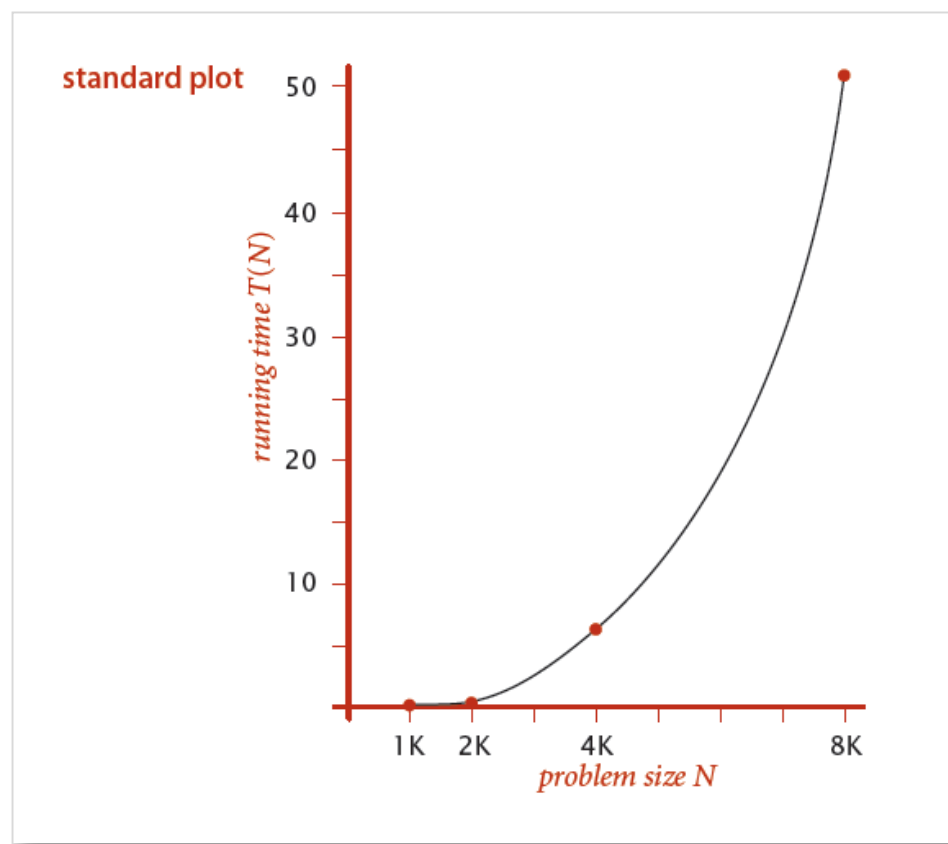
تحلیل تجربی

۱۰

□ مشاهده. برنامه را به ازای اندازه‌های مختلف ورودی اجرا کنید و زمان اجرا را اندازه بگیرید.

N	time (seconds)
250	0.0
500	0.0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?

□ نمودار استاندارد. نمودار زمان اجرای $T(N)$ بر حسب اندازه‌ی ورودی N .



□ نمودار **log-log**. نمودار زمان اجرا برحسب اندازه‌ی ورودی در مقیاس **log-log**.

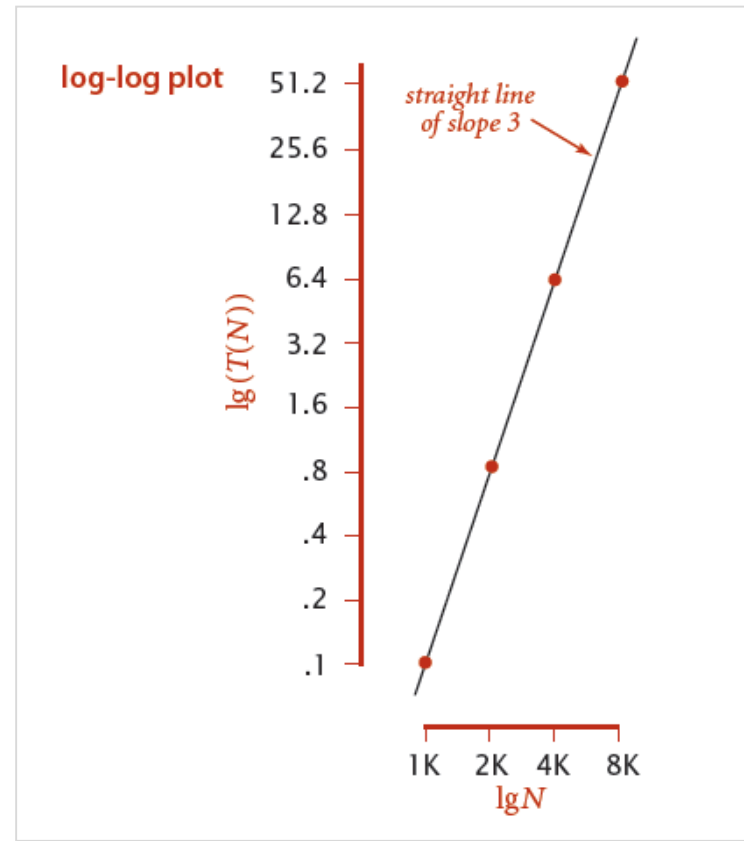
$$\lg(T(N)) = b \lg N + c$$

$$b = 2.999$$

$$c = -33.2103$$

$$T(N) = aN^b, \quad a = 2^c$$

$$T(N) = 1.006 \times 10^{-10} N^{2.999}$$



پیش‌بینی و اعتبارسنجی

۱۳

□ فرضیه. زمان اجرا تقریباً برابر است با $N^{2.999} \times 10^{-10} \times 1/0.06$ ثانیه.

□ پیش‌بینی.

□ $N = 8000$ ۵۱/۰ ثانیه برای

□ $N = 16000$ ۴۰۸/۱ ثانیه برای

□ مشاهدات.

□ اجرای الگوریتم

N	time (seconds)
8,000	51.1
16,000	410.8

مشاهدات جدید با پیش‌بینی‌های انجام شده سازگار است!

آزمون دو برابر کردن

۱۴

- آزمون دو برابر کردن. یک روش سریع به منظور تخمین **b**.
- برنامه را به دفعات اجرا و هر بار اندازه‌ی ورودی را **دو برابر** کن.

N	time (seconds)	ratio	lg ratio
250	0.0	-	-
500	0.0	4.8	2.3
1,000	0.1	6.9	2.8
2,000	0.8	7.7	2.9
4,000	6.4	8.0	3.0
8,000	?	8.0	3.0

همگرا شدن به مقدار ثابت **b** ←

آزمون دو برابر کردن

۱۵

□ آزمون دو برابر کردن. یک روش سریع به منظور تخمین **b**.

□ **س.** مقدار **a** را چگونه تخمین بزنیم؟

□ **ج.** برنامه را برای یک ورودی بزرگ اجرا کن و معادله‌ی حاصل را برای به دست آوردن **a** حل کن.

N	time (seconds)
8,000	51.1
16,000	410.8

$$51.1 = a \times 8000^3$$

$$\Rightarrow a = 0.998 \times 10^{-10}$$

$$T(N) = 0.998 \times 10^{-10} N^3$$

تحلیل الگوریتم‌ها به وسیله‌ی آزمایش

۱۶

□ عوامل مستقل از سیستم.

□ الگوریتم

□ داده‌های ورودی

تعیین کننده‌ی توان **b**
در قاعده‌ی توانی

تعیین کننده‌ی ثابت **a**
در قاعده‌ی توانی

□ عوامل وابسته به سیستم.

□ سخت‌افزار: پردازنده، حافظه، کش

□ نرم‌افزار: کامپایلر، مفسر، زباله جمع‌کن

□ سیستم: سیستم عامل، شبکه، برنامه‌های دیگر

□ **اخبار بد.** عوامل وابسته به سیستم اندازه‌گیری دقیق کارایی را مشکل می‌سازند.

□ **اخبار خوب.** استفاده از روش علمی در تحلیل الگوریتم‌ها در مقایسه با علوم دیگر بسیار ساده‌تر و ارزان‌تر است.

مثلاً توانایی انجام تعداد بسیار زیادی از آزمایش‌ها

مدل‌های ریاضی

مدل‌های ریاضی برای زمان اجرا

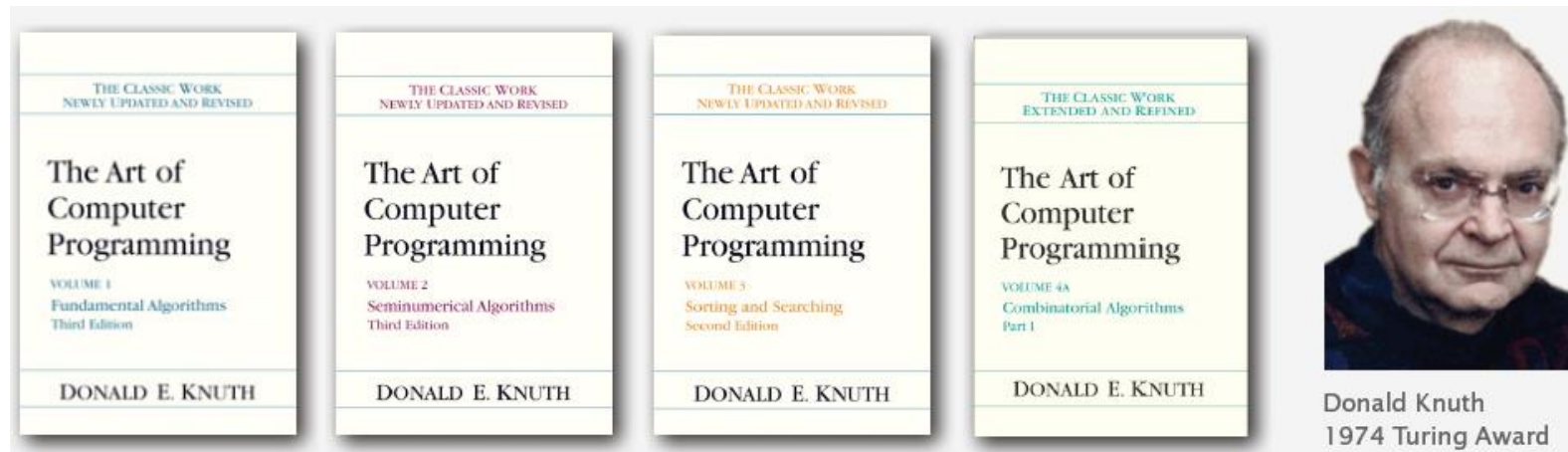
۱۸

□ زمان اجرای کل: مجموع هزینه $\times$ فراوانی به ازای همه‌ی عملیات.

□ نیاز به تحلیل برنامه به منظور تعیین مجموعه عملیات

□ هزینه به ماشین و کامپایلر بستگی دارد.

□ فراوانی به الگوریتم و داده‌های ورودی بستگی دارد.



□ به لحاظ اصولی، مدل‌های دقیق ریاضی قابل دسترسی هستند.

هزینه‌ی عملیات ابتدایی

۱۹

عمل	مثال	زمان (نانو ثانیه)
جمع اعداد صحیح	$a + b$	۲/۱
ضرب اعداد صحیح	$a * b$	۲/۴
تقسیم اعداد صحیح	a / b	۵/۴
جمع اعداد اعشاری	$a + b$	۴/۶
ضرب اعداد اعشاری	$a * b$	۴/۲
تقسیم اعداد اعشاری	a / b	۱۳/۵
محاسبه‌ی سینوس	<code>Math.sin(theta)</code>	۹۱/۳
محاسبه‌ی آرک تانژانت	<code>Math.atan2(y, x)</code>	۱۲۹/۰
...	...	...

یک کامپیوتر شفقی با سیستم عامل ویندوز ۷ و یک پردازنده‌ی ۲/۲ گیگا هرترزی و ۲ گیگا بایت حافظه

هزینه‌ی عملیات ابتدایی

۲۰

زمان (نانو ثانیه)	مثال	عمل
c_1	<code>int a</code>	اعلان متغیر
c_2	<code>a = b</code>	انتساب
c_3	<code>a < b</code>	مقایسه‌ی اعداد صحیح
c_4	<code>a[i]</code>	دستیابی به یک عنصر آرایه
c_5	<code>a.length</code>	طول آرایه
$c_6 \ N$	<code>new int[N]</code>	تخصیص آرایه‌ی یک بعدی
$c_7 \ N^2$	<code>new int[N][N]</code>	تخصیص آرایه‌ی دو بعدی
c_8	<code>s.length()</code>	طول رشته
c_9	<code>s.substring(N/2, N)</code>	استخراج زیررشته
$c_{10} \ N$	<code>s + t</code>	اتصال دو رشته

مثال: ۱-مجموع

۲۱

□ س. تعداد دستورالعمل‌ها بر حسب اندازه‌ی ورودی N چیست؟

```
int count = 0;
for (int i = 0; i < N; i++)
    if (a[i] == 0)
        count++;
```

عمل	فراوانی
اعلان متغیر	2
انتساب	2
مقایسه (کوچک‌تر)	$N + 1$
مقایسه (مساوی)	N
دستیابی به عنصر آرایه	N
افزایش	N to $2N$

مثال: ۲-مجموع

۲۲

□ س. تعداد دستورالعمل‌ها بر حسب اندازه‌ی ورودی N چیست؟

```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = i + 1; j < N; j++)
        if (a[i] + a[j] == 0)
            count++;
```

$$0 + 1 + 2 + \dots + (N - 1) = \frac{1}{2} N(N - 1)$$

$$= \binom{N}{2}$$

عمل	فراوانی
اعلان متغیر	$N + 2$
انتساب	$N + 2$
مقایسه (کوچک‌تر)	$\frac{1}{2} (N + 1) (N + 2)$
جمع صحیح	$\frac{1}{2} N (N - 1)$
مقایسه (مساوی)	$\frac{1}{2} N (N - 1)$
دستیابی به یک عنصر آرایه	$N (N - 1)$
افزایش	$\frac{1}{2} N (N - 1) \text{ to } N (N - 1)$

ساده‌سازی ۱: مدل هزینه

۲۳

□ مدل هزینه. استفاده از یک عمل ابتدایی به عنوان **عمل اصلی** به منظور تقریب زمان اجرا.

```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = i + 1; j < N; j++)
        if (a[i] + a[j] == 0)
            count++;
```

عمل اصلی: →
دستیابی به عناصر آرایه

عمل	فراوانی
اعلان متغیر	$N + 2$
انتساب	$N + 2$
مقایسه (کوچک‌تر)	$\frac{1}{2}(N + 1)(N + 2)$
جمع صحیح	$\frac{1}{2} N(N - 1)$
مقایسه (مساوی)	$\frac{1}{2} N(N - 1)$
دستیابی به یک عنصر آرایه	$N(N - 1)$
افزایش	$\frac{1}{2} N(N - 1) \text{ to } N(N - 1)$

ساده‌سازی ۲: نماد تیلدا

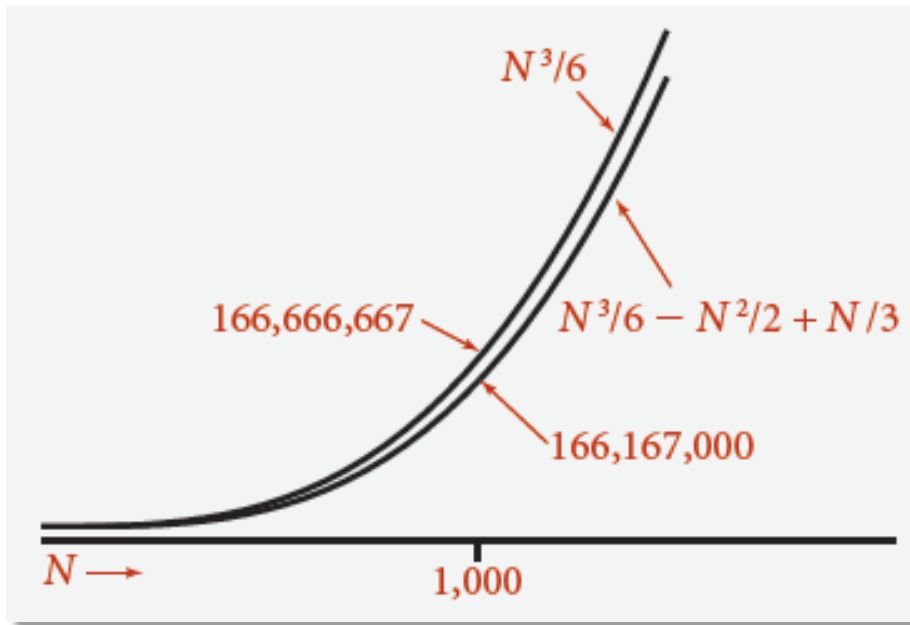
۲۴

□ تخمین زمان اجرا (یا حافظه) به عنوان تابعی از اندازه‌ی ورودی.

□ نادیده گرفتن جملات مرتبه پایین‌تر.

□ برای N های بزرگ، قابل چشم‌پوشی است.

□ برای N های کوچک، اهمیت نمی‌دهیم.



$$1/6 N^3 + 20N + 16 \sim 1/6 N^3$$

$$1/6 N^3 + 100N^{4/3} + 56 \sim 1/6 N^3$$

$$1/6 N^3 - 1/2 N^2 + 1/3 N \sim 1/6 N^3$$

$$\lim_{n \rightarrow \infty} \frac{f(N)}{g(N)} = 1 \quad \text{تعریف رسمی.} \quad f(N) \sim g(N) \quad \text{اگر و فقط اگر}$$

ساده سازی ۲: نماد تیلدا

۲۵

□ تخمین زمان اجرا (یا حافظه) به عنوان تابعی از اندازه‌ی ورودی.

□ نادیده گرفتن جملات مرتبه پایین‌تر.

□ برای N های بزرگ، قابل چشم‌پوشی است.

□ برای N های کوچک، اهمیت نمی‌دهیم.

نماد تیلدا	فراوانی	عمل
$\sim N$	$N + 2$	اعلان متغیر
$\sim N$	$N + 2$	انتساب
$\sim \frac{1}{2} N^2$	$\frac{1}{2} (N + 1) (N + 2)$	مقایسه (کوچک‌تر)
$\sim \frac{1}{2} N^2$	$\frac{1}{2} N (N - 1)$	جمع صحیح
$\sim \frac{1}{2} N^2$	$\frac{1}{2} N (N - 1)$	مقایسه (مساوی)
$\sim N^2$	$N (N - 1)$	دستیابی به یک عنصر آرایه
$\sim \frac{1}{2} N^2$ to $\sim N^2$	$\frac{1}{2} N (N - 1)$ to $N (N - 1)$	افزایش

مثال: ۲-مجموع

۲۶

□ س. تعداد تقریبی دستیابی‌ها به عناصر آرایه به صورت تابعی از اندازه‌ی ورودی N ؟

```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = i + 1; j < N; j++)
        if (a[i] + a[j] == 0)
            count++;
```

حلقه‌ی درونی ←

$$\binom{N}{2} = \frac{1}{2} N(N-1) \sim \frac{1}{2} N^2$$

□ ج. N^2 دسترسی به آرایه

□ خلاصه. از مدل هزینه و نماد تیلدا برای ساده‌سازی شمارش فراوانی استفاده کنید.

مثال: ۳-مجموع

۲۷

□ س. تعداد تقریبی دستیابی‌ها به عناصر آرایه به صورت تابعی از اندازه‌ی ورودی N ؟

```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = i + 1; j < N; j++)
        for (int k = j + 1; k < N; k++)
            if (a[i] + a[j] + a[k] == 0)
                count++;
```

حلقه‌ی درونی ←

□ ج. $N^3 \sim 1/2$ دسترسی به آرایه

$$\binom{N}{3} = \frac{N(N-1)(N-2)}{3!} \sim \frac{1}{6} N^3$$

□ خلاصه. از مدل هزینه و نماد تیلدا برای ساده‌سازی شمارش فراوانی استفاده کنید.

تخمین یک مجموع گسسته

۲۸

- س. چگونه می‌توان یک مجموع گسسته را تخمین زد؟
□ ج. مجموع را با یک **انتگرال** جایگزین کنید.

$$1 + 2 + \cdots + N.$$

$$\sum_{i=1}^N i \cong \int_{x=1}^N x dx \sim \frac{1}{2} N^2$$

$$1 + 1/2 + 1/3 + \cdots + 1/N.$$

$$\sum_{i=1}^N \frac{1}{i} \cong \int_{x=1}^N \frac{1}{x} dx = \ln N$$

$$3 - \text{sum.}$$

$$\sum_{i=1}^N \sum_{j=i}^N \sum_{k=j}^N 1 \cong \int_{x=1}^N \int_{y=x}^N \int_{z=y}^N dz dy dx \sim \frac{1}{6} N^3$$

نرخ رشد



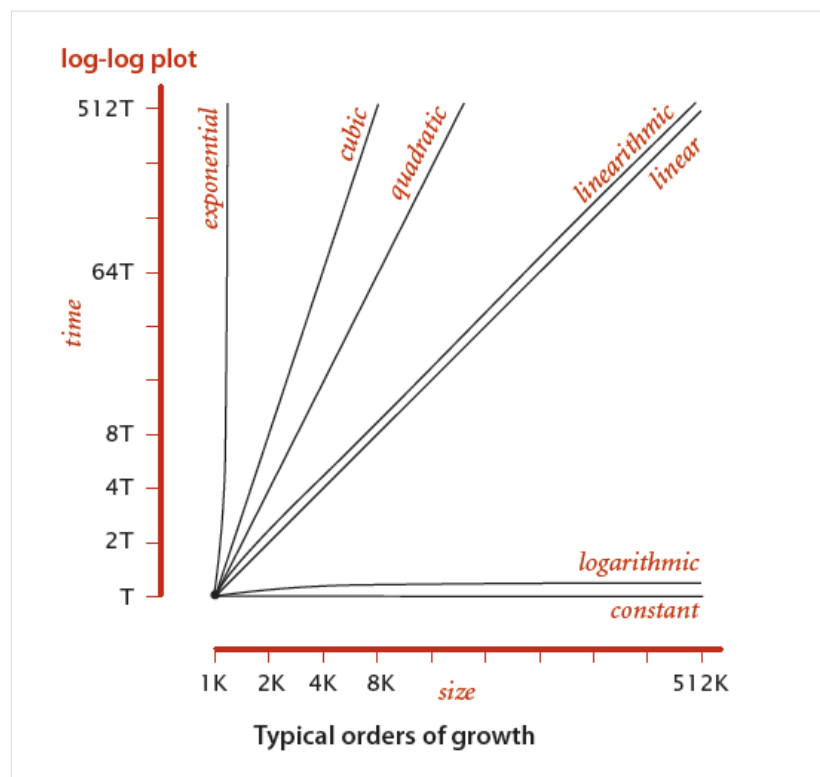
دسته‌بندی نرخ رشد الگوریتم‌ها

۳۰

□ اخبار خوب. یک مجموعه‌ی کوچک از توابع مانند زیر

$1, \log N, N, N \log N, N^2, N^3, 2^N$

برای توصیف نرخ رشد اغلب الگوریتم‌ها کفایت می‌کند.



دسته‌بندی نرخ رشد الگوریتم‌ها

۳۱

نرخ رشد	نام	یک کد مثالی	مثال	$T(2N) / T(N)$
1	ثابت	<code>a = b + c</code>	جمع دو عدد	1
$\log N$	لگاریتمی	<code>while (N > 1) { N = N / 2; ... }</code>	جستجوی دودویی	~ 1
N	خطی	<code>for (int i = 0; i < N; i++) { ... }</code>	یافتن بیشینه	2
$N \log N$	خطی لگاریتمی	مرتب‌سازی ادغامی [با ما باشید]	مرتب‌سازی ادغامی	~ 2
N^2	درجه دو	<code>for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }</code>	دو حلقه‌ی تو در تو	4
N^3	درجه سه	<code>for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }</code>	سه حلقه‌ی تو در تو	8
2^N	نمایی	برج‌های هانوی	جستجوی کامل	$T(N)$

مقایسه‌ی نرخ رشد توابع مختلف

۳۲

بزرگ‌ترین مسئله‌ی قابل حل در چند دقیقه				نرخ رشد
۲۰۰۰ به بعد	دهه ۹۰	دهه ۸۰	دهه ۷۰	
هر	هر	هر	هر	1
هر	هر	هر	هر	$\log N$
چند میلیارد	چند صد میلیون	چند ده میلیون	چند میلیون	N
چند صد میلیون	چند ده میلیون	چند میلیون	چند صد هزار	$N \log N$
چند ده هزار	چند هزار	هزار	چند صد	N^2
چند هزار	هزار	چند صد	صد	N^3
سی	بیست و چند	بیست و چند	بیست	2^N

نتیجه. برای عقب نماندن از قاعده‌ی مور نیاز به الگوریتم‌های خطی یا خطی-لگاریتمی داریم.

جستجوی دودویی

۳۳

□ هدف. با داشتن یک **آرایه‌ی مرتب** و یک کلید، اندیس کلید را در آرایه پیدا کنید.

□ جستجوی دودویی. کلید مورد نظر را با کلید وسط مقایسه کن:

□ اگر کوچک‌تر است، نیمه‌ی چپ را جستجو کن.

□ اگر بزرگ‌تر است، نیمه‌ی راست را جستجو کن.

□ اگر مساوی است، کلید پیدا شده است.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑							↑							↑
lo							mid							hi

جستجوی دودویی

۳۴

□ هدف. با داشتن یک آرایه‌ی مرتب و یک کلید، اندیس کلید را در آرایه پیدا کنید.

□ جستجوی دودویی. کلید مورد نظر را با کلید وسط مقایسه کن:

□ اگر کوچک‌تر است، نیمه‌ی چپ را جستجو کن.

□ اگر بزرگ‌تر است، نیمه‌ی راست را جستجو کن.

□ اگر مساوی است، کلید پیدا شده است.

جستجوی موفق ۳۳

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑							↑							↑
lo							mid							hi

جستجوی دودویی

۳۵

□ هدف. با داشتن یک آرایه‌ی مرتب و یک کلید، اندیس کلید را در آرایه پیدا کنید.

□ جستجوی دودویی. کلید مورد نظر را با کلید وسط مقایسه کن:

□ اگر کوچک‌تر است، نیمه‌ی چپ را جستجو کن.

□ اگر بزرگ‌تر است، نیمه‌ی راست را جستجو کن.

□ اگر مساوی است، کلید پیدا شده است.

جستجوی موفق ۳۳

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑			↑			↑								
lo			mid			hi								

جستجوی دودویی

۳۶

□ هدف. با داشتن یک آرایه‌ی مرتب و یک کلید، اندیس کلید را در آرایه پیدا کنید.

□ جستجوی دودویی. کلید مورد نظر را با کلید وسط مقایسه کن:

□ اگر کوچک‌تر است، نیمه‌ی چپ را جستجو کن.

□ اگر بزرگ‌تر است، نیمه‌ی راست را جستجو کن.

□ اگر مساوی است، کلید پیدا شده است.

جستجوی موفق ۳۳

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
				↑	↑	↑								
				lo	mid	hi								

جستجوی دودویی

۳۷

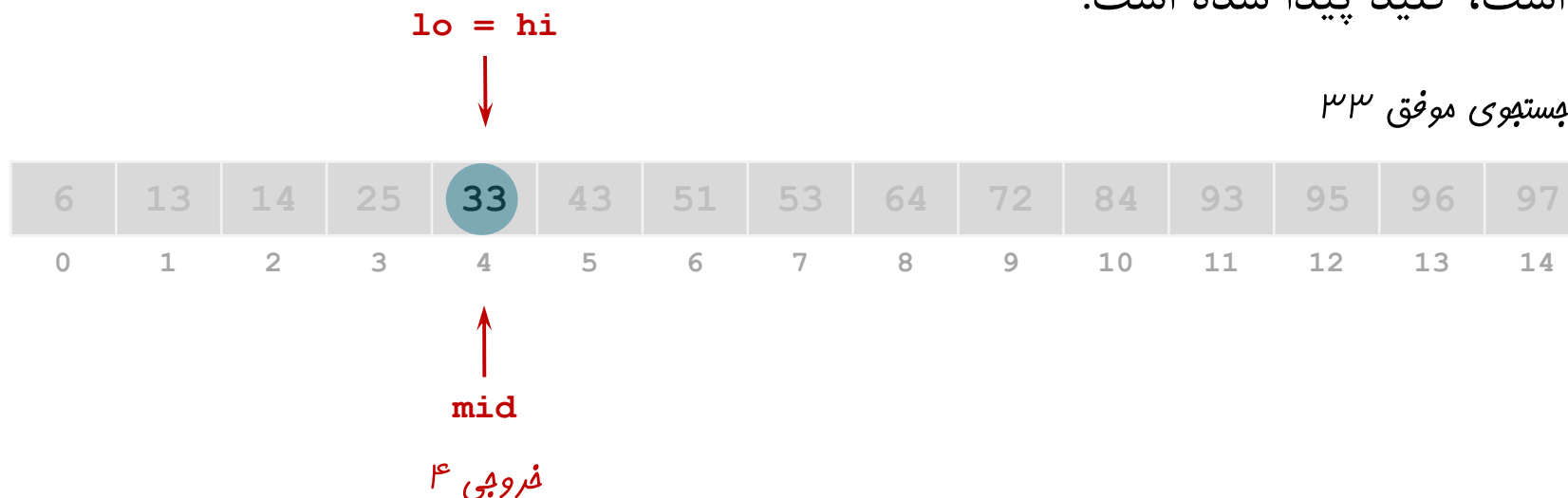
□ هدف. با داشتن یک آرایه‌ی مرتب و یک کلید، اندیس کلید را در آرایه پیدا کنید.

□ جستجوی دودویی. کلید مورد نظر را با کلید وسط مقایسه کن:

□ اگر کوچک‌تر است، نیمه‌ی چپ را جستجو کن.

□ اگر بزرگ‌تر است، نیمه‌ی راست را جستجو کن.

□ اگر مساوی است، کلید پیدا شده است.



جستجوی دودویی: پیاده‌سازی در جاوا

۳۸

□ پیاده‌سازی جستجوی دودویی چقدر آسان است؟

□ اولین پیاده‌سازی جستجوی دودویی سال ۱۹۴۶؛ اولین پیاده‌سازی بدون خطا سال ۱۹۶۲.

□ خطای جاوا در پیاده‌سازی `Arrays.binarySearch()` در سال ۲۰۰۶ کشف شد.

```
public static int binarySearch(int[] a, int key)
{
    int lo = 0, hi = a.length - 1;
    while (lo <= hi)
    {
        int mid = lo + (hi - lo) / 2;
        if (key < a[mid]) hi = mid - 1;
        else if (key > a[mid]) lo = mid + 1;
        else return mid;
    }
    return -1;
}
```

ثابت مستقل از حلقه. اگر کلید مورد نظر در آرایه باشد، آنگاه همواره $a[lo] \leq key \leq a[hi]$

جستجوی دودویی: تحلیل ریاضی

۳۹

□ گزاره. جستجوی دودویی برای جستجو در یک آرایه‌ی مرتب با اندازه‌ی N ، حداکثر $\lg N + 1$ مقایسه انجام می‌دهد.

□ تعریف.

□ $T(N)$ = حداکثر تعداد مقایسه‌های جستجوی دودویی در یک زیرآرایه‌ی مرتب با اندازه‌ی N .

□ رابطه‌ی بازگشتی. $T(N) \leq T(N/2) + 1, \quad T(1) = 1$

$$\begin{aligned} T(N) &\leq T(N/2) + 1 \\ &\leq T(N/4) + 1 + 1 \\ &\leq T(N/8) + 1 + 1 + 1 \\ &\dots \\ &\leq T(N/N) + 1 + 1 + \dots + 1 \\ &= 1 + \lg N \end{aligned}$$

یک الگوریتم $N^2 \lg N$ برای مسئله‌ی ۳-مجموع

۴۰

input

30 -40 -20 -10 40 0 10 5

sort

-40 -20 -10 0 5 10 30 40

binary search

(-40, -20) 60

(-40, -10) 50

(-40, 0) 40

(-40, 5) 35

(-40, 10) 30

...

(-40, 40) 0

...

(-10, 0) 10

...

(10, 30) -40

(10, 40) -50

(30, 40) -70

تکراری

□ الگوریتم.

□ N عدد (متمایز) را مرتب کن.

□ به ازای هر جفت از اعداد مانند $a[i]$ و $a[j]$ یک جستجوی دودویی برای یافتن $(a[i] + a[j])$ -انجام بده.

□ تحلیل. نرخ رشد برابر با $N^2 \lg N$ است.

□ مرحله ۱: N^2 با استفاده از مرتب سازی درجی

□ مرحله ۲: $N^2 \lg N$

مقایسه الگوریتم‌های ۳-مجموع

۴۱

□ فرضیه. الگوریتم $N \lg N$ در عمل بسیار سریع‌تر از الگوریتم کورکورانه‌ی N^3 است.

N	time (seconds)
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1

ThreeSum.java

N	time (seconds)
1,000	0.14
2,000	0.18
4,000	0.34
8,000	0.96
16,000	3.67
32,000	14.88
64,000	59.16

ThreeSumFast.java

وابستگی زمان اجرا به داده‌های ورودی

انواع تحلیل

۴۳

□ بهترین حالت. حد پایین روی هزینه.

□ با استفاده از «ساده‌ترین» ورودی

□ فراهم کننده‌ی یک هدف برای تمام ورودی‌ها

□ بدترین حالت. حد بالا روی هزینه.

□ با استفاده از «سخت‌ترین» ورودی

□ فراهم کننده‌ی یک تضمین برای تمام ورودی‌ها

□ حالت متوسط. هزینه‌ی مورد انتظار برای یک ورودی تصادفی.

□ نیاز به یک مدل برای ورودی «تصادفی»

□ فراهم کننده‌ی روشی برای تخمین کارایی

مثال ۱. تعداد دستیابی‌ها در الگوریتم کوکوراونه^۳ -مجموع

بهترین حالت: $\sim 1/2 N^3$

حالت متوسط: $\sim 1/2 N^3$

بدترین حالت: $\sim 1/2 N^3$

مثال ۲. تعداد مقایسه‌ها در جستجوی دودویی

بهترین حالت: ~ 1

حالت متوسط: $\sim \lg N$

بدترین حالت: $\sim \lg N$

نمادهای معمول در تحلیل الگوریتم‌ها

۴۴

نماد	مفهوم	مثال	شکل خلاصه برای	مورد استفاده
تیلدا	جمله‌ی بزرگ‌تر	$\sim 10N^2$	$10N^2$ $10N^2 + 22N \lg N$ $10N^2 + 2N + 37$	فراهم کننده مدل تقریبی
تتای بزرگ	نرخ رشد مجانبی	$\Theta(N^2)$	$\frac{1}{2} N^2$ $10N^2$ $5N^2 + 22N \lg N + 3N$	دسته‌بندی الگوریتم‌ها
ای بزرگ	$\Theta(N^2)$ و کوچک‌تر	$O(N^2)$	$10N^2$ $100N$ $22N \lg N + 3N$	فراهم کننده حد بالا
امگای بزرگ	$\Theta(N^2)$ و بزرگ‌تر	$\Omega(N^2)$	$\frac{1}{2} N^2$ N^5 $N^3 + 22N \lg N + 3N$	فراهم کننده حد پایین

یک اشتباه متداول. تفسیر ای بزرگ به عنوان یک مدل تقریبی!