از کتاب بخوانید

فصل‌های ۱ و ۳ کتاب را بخوانید.

روش‌های تحلیل الگوریتم‌ها

هدف از تحلیل الگوریتم‌ها:

- بررسی رفتار الگوریتم قبل از پیاده‌سازی، از نظر زمان اجرا و مقدار حافظه‌ی مصرفی
- مقایسه‌ی الگوریتم‌ها با هم از نظر کارایی

زمان اجرای الگوریتم‌ها

عوامل زیر در زمان اجرای یک برنامه موثرند:

۱) سرعت سخت افزار

۲) نوع کامپایلر

۳) اندازه‌ی داده‌ی ورودی مسئله

۴) ترکیب داده‌ی ورودی

۵) پیچیدگی الگوریتم

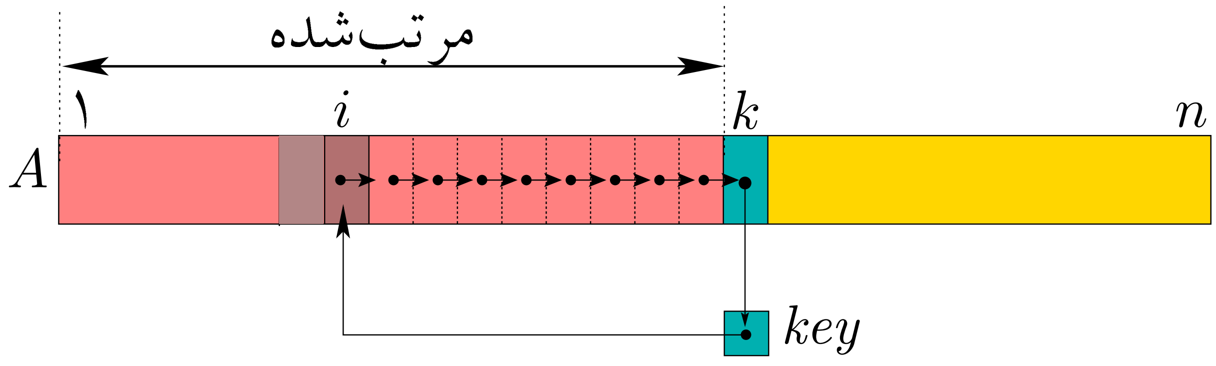
۶) پارامترهای دیگر که تاثیر ثابت در زمان اجرا دارند

زمان اجرای الگوریتم: $T(n)$، n **اندازه‌ی ورودی مسئله**

توجه:

- **ممکن است چند داده‌ی ورودی داشته باشیم: مثلاً**
- **یک گراف، تعداد راس‌ها برابر** n، **تعداد یال‌ها برابر** m، **زمان اجرا** $T(n,m)$
- **چند پارامتر، انتزاع** (abstraction)

مثال: مرتب‌ساز درجی (Insertion-Sort)



<u>Insertion-Sort</u> (A, n)

$\triangleright$ A: (the array to sort)
$\triangleright$ n: (size of array)
for $k \leftarrow 2$ **to** n
  do $key \leftarrow A[k]$
    $i \leftarrow k - 1$
    while $i > 0$ **and** $A[i] > key$
      do $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$

۸ ۲ ۴ ۹ ۳ ۶ ۳

۲ ۸ ۴ ۹ ۳ ۶ ۳

۲ ۴ ۸ ۹ ۳ ۶ ۳

۲ ۴ ۸ ۹ ۳ ۶ ۳

۲ ۳ ۴ ۸ ۹ ۶ ۳

۲ ۳ ۴ ۶ ۸ ۹ ۳

۲ ۳ ۳ ۴ ۶ ۸ ۹

سطر	هزینه	تعداد
۱	$c_{۱}$	n
۲	$c_{۲}$	$n-۱$
۳	$c_{۳}$	$n-۱$
۴	$c_{۴}$	$\sum_{k=۲}^{n} t_k$
۵	$c_{۵}$	$\sum_{k=۲}^{n} (t_k-۱)$
۶	$c_{۶}$	$\sum_{k=۲}^{n} (t_k-۱)$
۷	$c_{۷}$	$n-۱$

$$
\begin{aligned}
T(n) &= c_۱ n + (c_۲ + c_۳ + c_۷)(n - ۱) + c_۴ \sum_{k=۲}^{n} t_k + \\
&\quad (c_۵ + c_۶) \sum_{k=۲}^{n} (t_k - ۱) \\
&= An + B \sum_{k=۲}^{n} t_k + C
\end{aligned}
$$

بدترین حالت

در بدترین حالت (worst-case):

حداکثر مقدار t_k برابر k است و این زمانی است که آرایه برعکس مرتب شده باشد.

$$
\begin{aligned}
T(n) &= An + B\sum_{k=۲}^{n} t_k + C \\
&= An + B\frac{(n+۱)(n+۲)}{۲} + C \\
&= an^{۲} + bn + c
\end{aligned}
$$

به‌ترین حالت

به‌ترین حالت (best-case)**: آرایه از قبل مرتب باشد. در این حالت داریم** $t_k = ۱$ **و**

$$T(n) = An + B(n - ۱) + C$$

که برحسب n **خطی است و با بدترین حالت که درجه دو است تفاوت دارد.**

پس مسئله‌ی یافتن حالت میانگین (average-case) **مطرح می‌شود.**

حالت میانگین

داده‌ی ورودی به صورت تصادفی داده است.

فرض می‌شود که همه‌ی حالت‌های مختلف داده‌های ورودی، احتمال برابر دارند $(\frac{1}{n!})$

حالت میانگین در مورد مثال فوق برابر است با

$$\overline{T}(n) = An + C + \frac{1}{n!}\sum_{i=1}^{n!}\sum_{k=2}^{n} Bt_{k,i}$$

$$\frac{1}{n!}\sum_{i=1}^{n!} t_{k,i} = \bar{t}_k.$$

یعنی

$$\overline{T}(n) = An + C + B \sum_{k=۲}^{n} \bar{t}_k.$$

$$\overline{T}(n) = An + C + B \sum_{k=۲}^{n} \bar{t}_k$$

i: **مکانی را که عضو باید در آن جا بگیرد** $(۱ \leq i \leq k)$. $\Longleftarrow$ $t_k = k - i + ۱$

$$\begin{aligned} \bar{t}_k &= \sum_{i=۱}^{k} \frac{۱}{k}(k - i + ۱) \\ &= \frac{۱}{k} \times \frac{k(k+۱)}{۲} \\ &= \frac{k+۱}{۲} \end{aligned}$$

$$\overline{T}(n) = An + C + B\sum_{k=۲}^{n}\frac{k+۱}{۲}$$
$$= a'n^{۲} + b'n + c'$$

پس رفتار حالت میانگین و بدترین حالت یکسان است.

مرتب‌سازی درج دودویی

الگوریتم قبلی: مرتب‌سازی درج مستقیم. اما درج یک عنصر به‌صورت دودویی

```
Binary-Search(A, l, r, key)
      ▷ آرایه‌ی A مرتب است
      ▷ اندیس l ≤ i ≤ r را پیدا کن به‌طوری‌که key حتماً قبل از A[i] در آرایه درج شود
if l ≥ r
  then i = l
m ← ⌊(l+r)/2⌋
if key ≤ A[m]
  then i ← Binary-Search(A, l, m − 1)
  else i ← Binary-Search(A, m + 1, r)
return i
```

تحلیل مرتب‌سازی درج دودویی

با چه زمان اجرایی و چگونه می‌توان مرتب‌سازی درج دودویی را پیاده‌سازی کرد؟

تحلیل مرتب‌سازی درج دودویی

با چه زمان اجرایی و چگونه می‌توان مرتب‌سازی درج دودویی را پیاده‌سازی کرد؟

ظاهراً این کار با هزینه‌ای متناسب با $n \lg n$ امکان‌پذیر است.

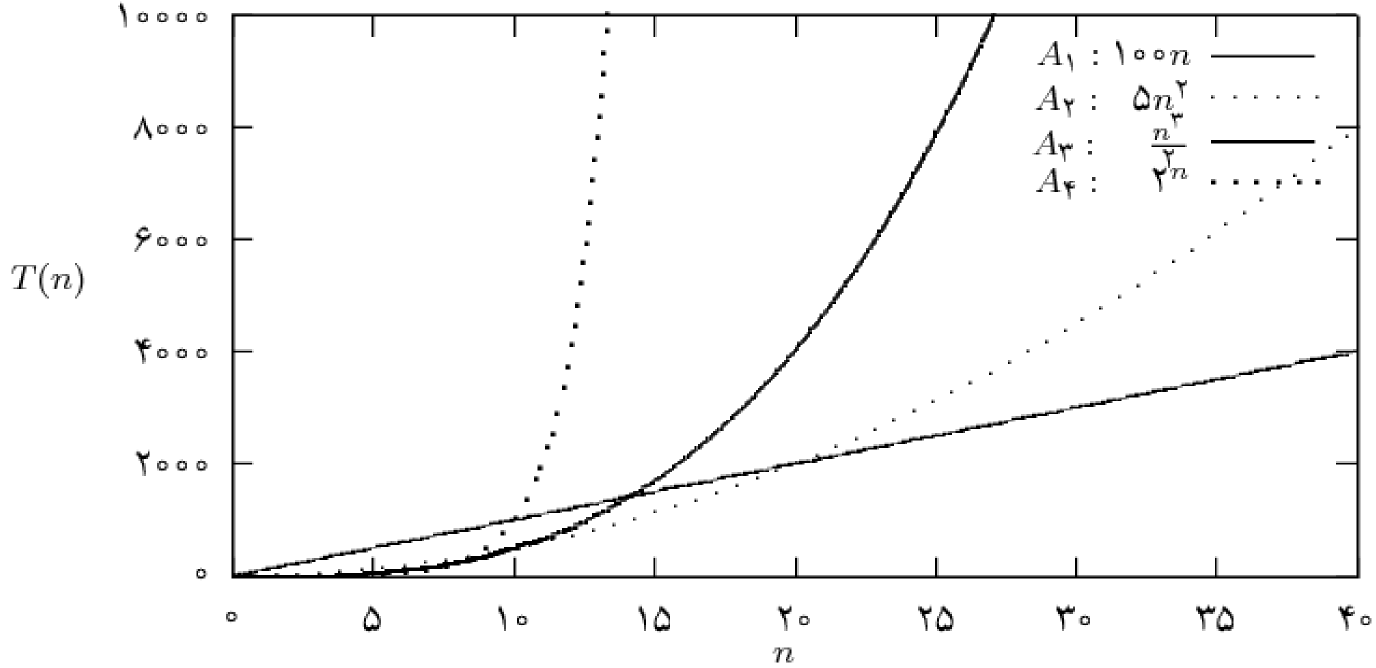
اما پیاده‌سازی با آرایه با این هزینه ممکن نیست.

برای کارایی بهینه باید بخش مرتب را با داده‌ساختار پیش‌رفته (مانند درخت قرمز–سیاه) پیاده‌سازی کرد.

مرتبه الگوریتم‌ها

پیچیدگی الگوریتم‌ها (complexity of algorithms)

الگوریتم	$T(n)$	$\mathcal{O}(.)$	حداکثر اندازه‌ی مسئله، قابل حل در ۱۰۰۰ ثانیه	حداکثراندازه برای ماشین ۱۰ برابر سریع‌تر	نسبت	نسبت برای ماشین ۱۰۰۰ برابر سریع‌تر
A_1	$100n$	$\mathcal{O}(n)$	۱۰	۱۰۰	۱۰	۱۰۰۰
A_2	$5n^2$	$\mathcal{O}(n^2)$	۱۴	۴۵	۳/۲	۱۳۱/۹۴
A_3	$n^3/2$	$\mathcal{O}(n^3)$	۱۲	۲۷	۲/۳	۱۲۵/۹۹
A_4	2^n	$\mathcal{O}(2^n)$	۱۰	۱۳	۱/۳	۲



زمان‌های اجرای چهار الگوریتم برای یک مسئله.

تابع‌های رشد (growth functions)

سه نماد $\mathcal{O}$، Θ و Ω

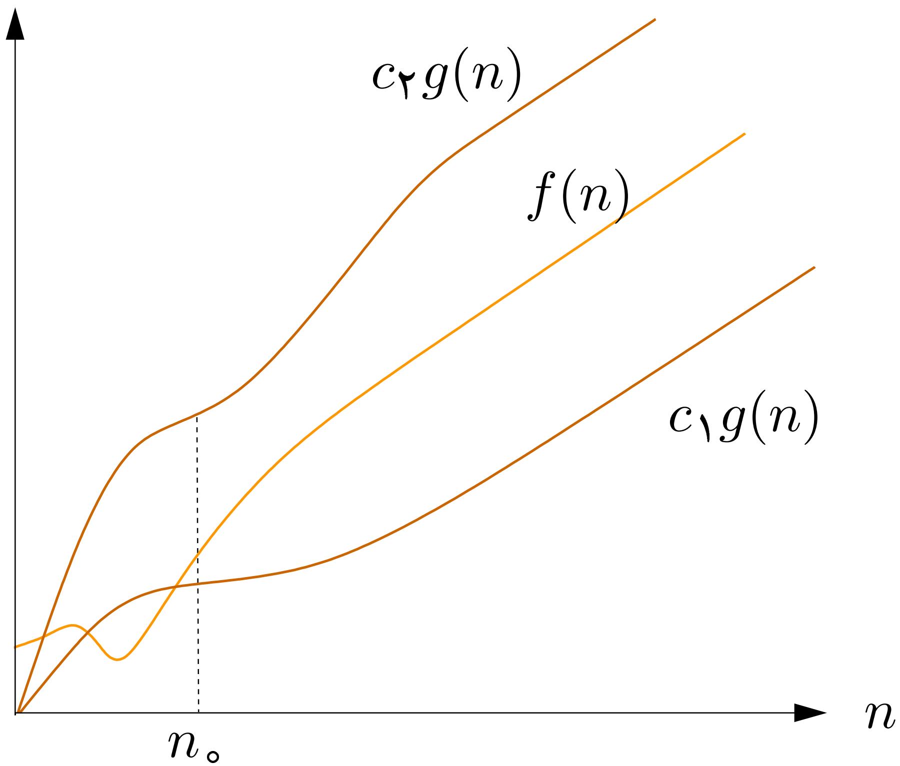
می‌گوییم

- بدترین حالت الگوریتم مرتب‌سازی درجی $T(n) = \Theta(n^۲)$ است.
  یعنی از مرتبه‌ی دقیق $n^۲$ است.
  در حالت میانگین نیز $\Theta(n^۲)$ است.
- الگوریتم مرتب‌سازی هرمی (heap sort) از $\mathcal{O}(n \lg n)$ یا از مرتبه‌ی $n \lg n$ است.
- کلیه‌ی الگوریتم‌های مرتب‌سازی مقایسه‌ای، از $\Omega(n \lg n)$ هستند.

تعریف

$$\Theta(g(n)) = \{f(n) : \exists c_۱, c_۲ > ۰ \text{ and } n_۰ > ۰ \text{ such that } \forall n \geq n_۰,\ ۰ \leq c_۱ g(n) \leq f(n) \leq c_۲ g(n)\}$$

عبارت $c_۱ g(n) \leq f(n) \leq c_۲ g(n)$ **به این معنی است که برای مقادیر بزرگ** n **درجه‌ی رشد** f **و** g **یکسان است.**



$$f(n) = \Theta(g(n))$$

$$f(n) \in \Theta(g(n))$$

می‌گوییم:

تابع $g(n)$ کران بالای بسته‌ی مجانبی (asymptotically tight bound) برای $f(n)$ است.

به‌شکل ساده‌تر

$$f(n) = \Theta(g(n))$$

در مورد الگوریتم مرتب‌ساز درجی، $T(n) = \Theta(n^2)$.

زیرا می‌توان مقادیر مثبتی برای c_1 **و** c_2 **را طوری یافت که**

$$c_1 n^2 \leq an^2 + bn + c \leq c_2 n^2$$

مسئله: ثابت کنید که $100n^2 + 5n - 4 = \Theta(n^2)$.

مسئله: ثابت کنید که $100n^2 + 5n - 4 = \Theta(n^2)$.

حل:

اگر $c_1 = 1$ **و** $c_2 = 200$، **برای همه‌ی مقادیر** $n > 1$ **داریم**

$$c_1 n^2 \leq 100n^2 + 5n - 4 \leq c_2 n^2$$

مسئله: نشان دهید که $100n^2 + 5n - 4 \neq \Theta(n^3)$.

حل:

باید نشان دهیم که هیچ مقادیر مثبت برای c_1 **و** c_2 **و یک مقدار برای** n_0 **پیدا نمی‌شود که رابطه‌ی**

$$c_1 n^3 \leq 100n^2 + 5n - 4 \leq c_2 n^3$$

برای همه‌ی مقادیر $n > n_0$ **برقرار باشد.**

به‌وضوح به‌ازای هر مقدار $c_1 > 0$ **و برای** n**های بزرگ داریم** $c_1 n^3 \not\leq 100n^2 + 5n - 4$.

می‌توان ثابت کرد که هر چندجمله‌ای از مرتبه‌ی دقیق جمله‌ی با بزرگ‌ترین توانش است، یعنی

$$a_k n^k + a_{k-۱} n^{k-۱} + \cdots = \Theta(n^k)$$

در مورد مرتب‌ساز درجی داریم:

$$\begin{aligned} T(n) &= \Theta(n^۲) \\ &= \Theta(۱۰۰n^۲) \\ &\neq \Theta(n^۳) \\ &\neq \Theta(n^۲ \lg n) \\ &\neq \Theta(n \lg n) \end{aligned}$$

برای اثبات $T(n) \neq \Theta(n \lg n)$ باید نشان داد که نمی‌توان ثابت‌های $c_۱$ و $c_۲$ را پیدا کرد که برای $n > n_۰$ داشته‌باشیم:

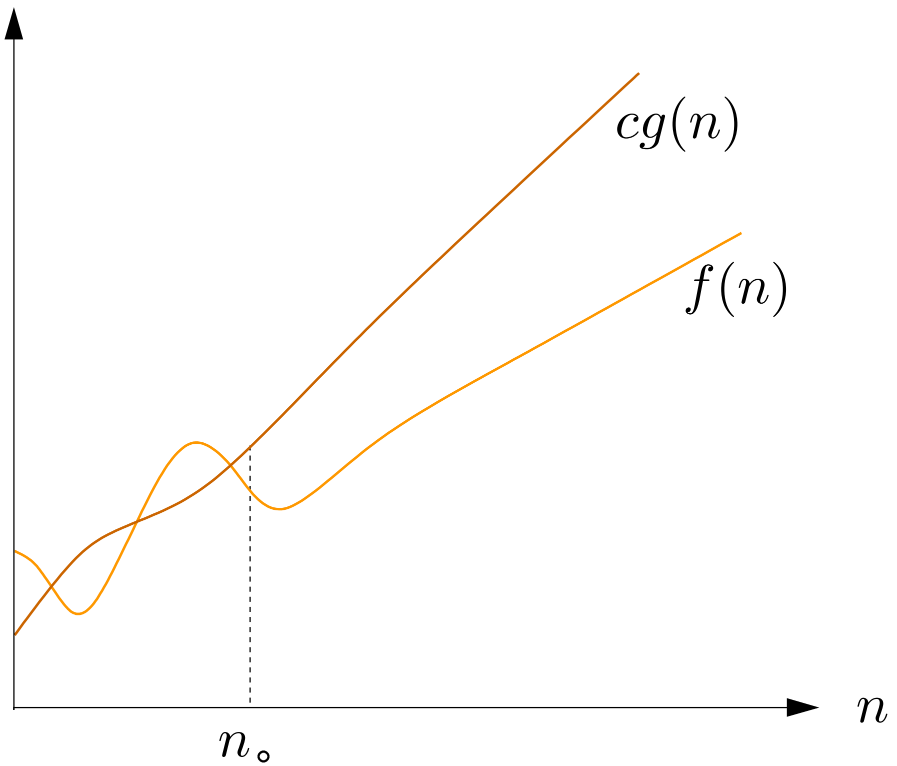
$$c_۱ n \lg n \leq n^۲ \leq c_۲ n \lg n$$

نماد $\mathcal{O}$

برای $g(n)$ داده‌شده، $O(g(n))$ را به شکل مجموعه‌ی توابع زیر تعریف می‌کنیم:

$$\mathcal{O}(g(n)) = \{f(n) : \exists c > 0 \text{ and } n_0 \quad \text{such that} \quad \forall n \geq n_0,\ 0 \leq f(n) \leq cg(n)\}$$

$g(n)$ کران بالای مجانبی (asymptotically upper bound) برای $f(n)$ است.



$$f(n) = O(g(n))$$

مثلاً در مورد مرتب‌ساز درجی، داریم

$$\begin{aligned} T(n) = \Theta(n^۲) &= \mathcal{O}(n^۲) \\ &= \mathcal{O}(۱۰۰n^۲) \\ &= \mathcal{O}(n^۳) \\ &= \mathcal{O}(n^۲ \lg n) \\ &\neq \mathcal{O}(n \lg n) \\ &\neq \mathcal{O}(n). \end{aligned}$$

می‌توان گفت مسئله از $\mathcal{O}(n^{۱۰۰})$ است، ولی این اطلاع چندان مفید نیست.

به‌همین جهت در حالت‌هایی که محاسبه‌ی Θ مشکل است، باید تابع داخل پرانتز $\mathcal{O}$ را تا حد امکان کوچک‌ترین باشد.

نکته: براساس تعریف روشن است که $\Theta(g(n)) \subseteq \mathcal{O}(g(n))$.

نماد Ω

برای $g(n)$ داده‌شده، $\Omega(g(n))$ را به‌شکل مجموعه‌ی توابع زیر تعریف می‌کنیم:

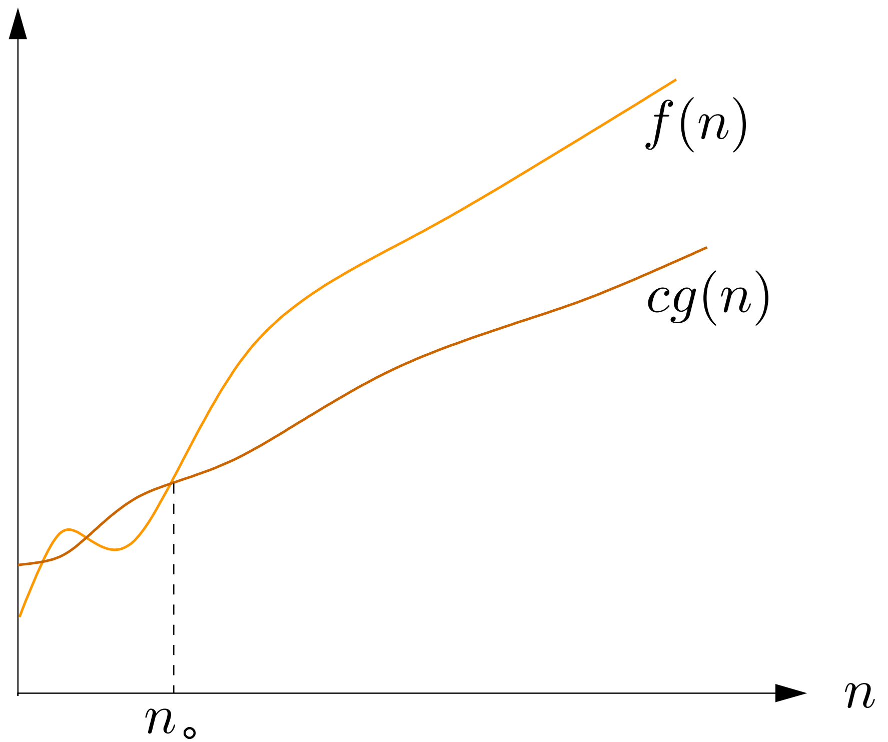
$$\Omega(g(n)) = \{f(n) : \exists c > 0 \text{ and } n_0 \text{ such that } \forall n \geq n_0 \; 0 \leq cg(n) \leq f(n)\}$$

$g(n)$ کران پایین مجانبی (asymptotically lower bound) برای $f(n)$ است.

مثلاً برای مرتب‌ساز درجی داریم

$$\begin{aligned} T(n) &= \Omega(n \lg n) \\ &= \Omega(n) \\ &= \Omega(n^{۲}) \\ &\neq \Omega(n^{۲} \lg n) \end{aligned}$$

در عمل نماد Ω برای نشان دادن کران پایین یک تابع دیگر به‌کار می‌رود.



$$f(n) = \Omega(g(n))$$

تابع Θ در واقع عطف $\mathcal{O}$ و Ω است یعنی

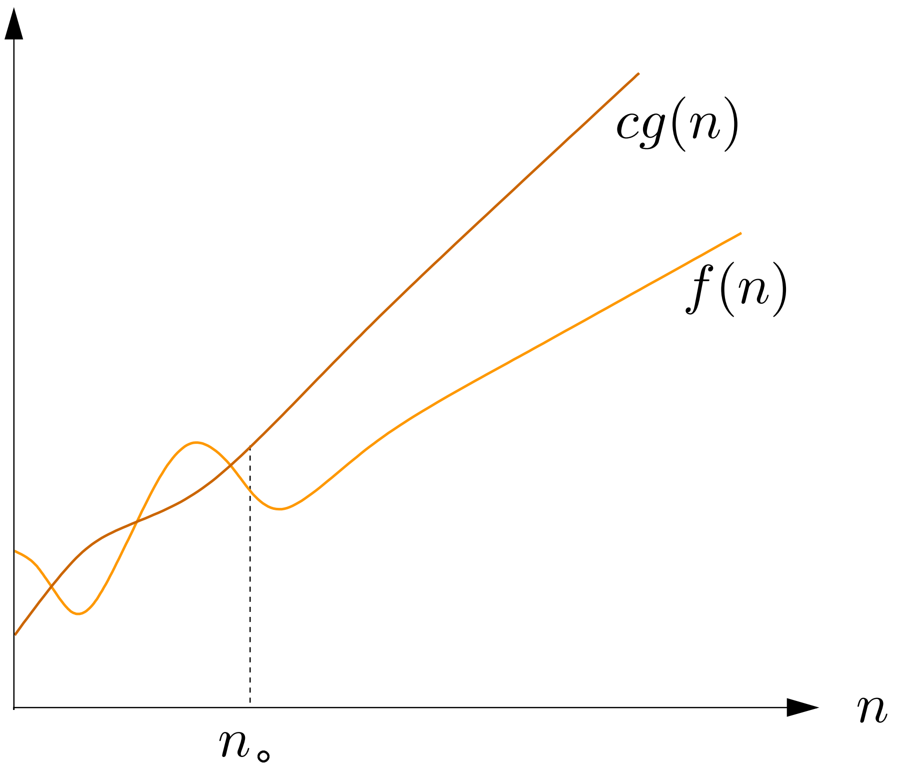
$$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = \mathcal{O}(g(n)) \wedge f(n) = \Omega(g(n))$$

یا

قضیه: شرط لازم و کافی برای $f(n) = \Theta(g(n))$ **آن است که** $f(n) = \mathcal{O}(g(n))$ **و** $f(n) = \Omega(g(n))$.

نماد o

$$o(g(n)) = \{f(n)| \quad \text{for any positive constant} \quad c > 0, \\ \exists n_0 > 0, s.t. \forall n > n_0 : 0 \leq f(n) < cg(n)\}$$



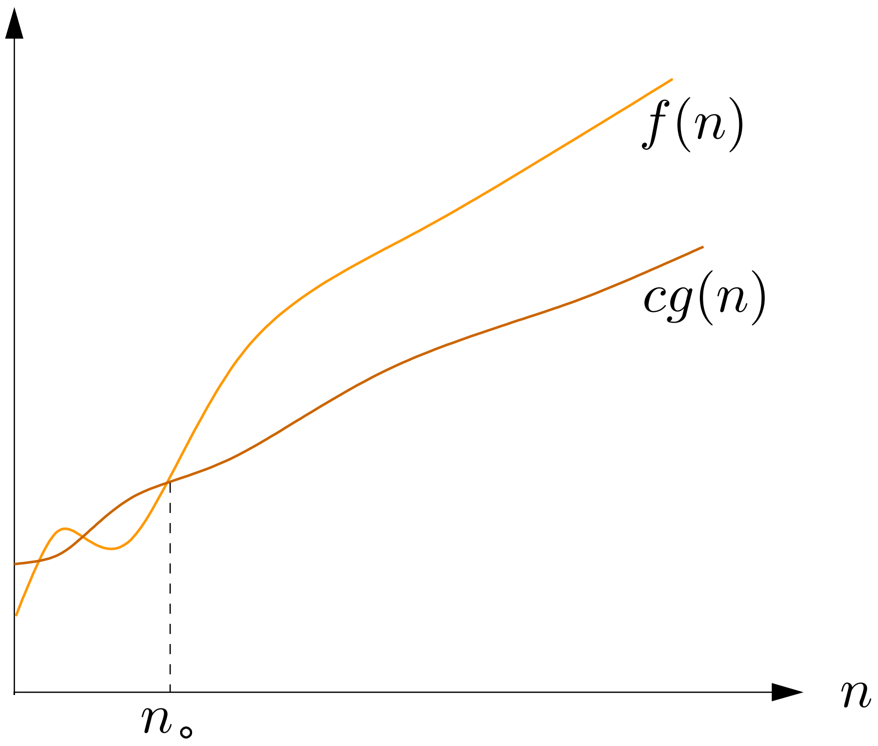
$$f(n) = o(g(n))$$

این نماد به $\mathcal{O}$ شبیه است، بااین تفاوت که درجه‌ی رشد $f(n)$ نمی‌تواند با $g(n)$ برابر باشد و باید الزاماً کوچک‌تر باشد.

نکته: اگر $f(n) = o(g(n))$**، داریم:** $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

نماد ω

$$\omega(f(n)) = \{g(n) | \quad \text{for any positive constant} \quad c > \circ, \exists n_\circ > \circ, \\ \text{such that} \quad \forall n > n_\circ, \circ \leq cg(n) < f(n)\}$$



$$f(n) = \omega(g(n))$$

رابطه‌ی ω با Ω مثل o با $\mathcal{O}$ است.

نکته: $f(n) = o(g(n))$ **اگر و فقط اگر** $g(n) = \omega(f(n))$.

نکته: اگر $f(n) = \omega(g(n))$، **داریم:** $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

مثلاً $n^۲/۲ = \omega(n)$ **ولی** $n^۲/۲ \neq \omega(n^۲)$.

نتیجه‌ای که از تعریف‌های فوق به‌دست می‌آید را در عبارت‌های زیر (هرچند نادقیق از نظر ریاضی) خلاصه می‌کنیم:

اگر F درجه‌ی رشد f و G درجه‌ی رشد g باشد،

$$f = \Theta(g) \iff F = G$$

$$f = \mathcal{O}(g) \iff F \leq G$$

$$f = o(g) \iff F < G$$

$$f = \Omega(g) \iff F \geq G$$

$$f = \omega(g) \iff F > G$$

خواص تابع‌های رشد

خاصیت تراگذری (transitive)

۱) از $f(n) = \Theta(g(n))$ **و** $g(n) = \Theta(h(n))$ **نتیجه می‌شود که** $f(n) = \Theta(h(n))$

۲) از $f(n) = \mathcal{O}(g(n))$ **و** $g(n) = \mathcal{O}(h(n))$ **نتیجه می‌شود که** $f(n) = \mathcal{O}(h(n))$

۳) از $f(n) = \Omega(g(n))$ **و** $g(n) = \Omega(h(n))$ **نتیجه می‌شود که** $f(n) = \Omega(h(n))$

۴) از $f(n) = o(g(n))$ **و** $g(n) = o(h(n))$ **نتیجه می‌شود که** $f(n) = o(h(n))$

۵) از $f(n) = \omega(g(n))$ **و** $g(n) = \omega(h(n))$ **نتیجه می‌شود که** $f(n) = \omega(h(n))$

خاصیت بازتابی (reflexive)

۱) $f(n) = \Theta(f(n))$

۲) $f(n) = \mathcal{O}(f(n))$

۳) $f(n) = \Omega(f(n))$

خاصیت تقارن (symmetry)

$f(n) = \Theta(g(n))$ **اگر و فقط اگر** $g(n) = \Theta(f(n))$

خاصیت تقارن ترانهاده (transpose symmetry)

۱) شرط لازم و کافی برای $f(n) = \mathcal{O}(g(n))$ آن است که $g(n) = \Omega(f(n))$

۲) شرط لازم و کافی برای $f(n) = o(g(n))$ آن است که $g(n) = \omega(f(n))$

تمرین: همین‌جا حل کنید!

۱) آیا $f(n) + g(n) = \Theta(\min(f(n), g(n))$ است؟

۲) آیا $f(n) = \Theta(f(n)/۲)$ است؟

۳) آیا $f(n) + o(f(n)) = \Theta(f(n))$؟

روش‌های تحلیل الگوریتم‌ها

الگوریتم‌های ترتیبی

- یک یا تعداد ثابتی عبارت ساده (Statement) $\Leftarrow \Theta(1)$

• $T(n)$ زمان اجرای یک تکه‌برنامه که به‌صورت زیر است:

◁ k تکه‌برنامه با زمان‌های اجرای زیر

$$\begin{aligned} T_۱(n) &= \mathcal{O}(f_۱(n)) \\ T_۲(n) &= \mathcal{O}(f_۲(n)) \\ &\cdots \\ T_k(n) &= \mathcal{O}(f_k(n)) \end{aligned}$$

در آن‌صورت،

$$T(n) = \sum_{i=۱}^{k} T_i(n) = \mathcal{O}(\max_{۱\leq i\leq k}(f_i(n))$$

◁ $g(n)$ **بار تکرار تکه‌برنامه‌ای با زمان اجرای** $S(n) = \mathcal{O}(f(n))$

$$T(n) = g(n)S(n) = \mathcal{O}(g(n)f(n))$$

◁ **ساختار if که قسمت‌های then و else آن به ترتیب زمان‌های** $T_1(n) = \mathcal{O}(f_1(n))$ **و** $T_2(n) = \mathcal{O}(f_2(n))$ **داشته باشند:**

$$T(n) = \max\{T_1(n), T_2(n)\} = \mathcal{O}(\max\{f_1(n), f_2(n)\})$$

مثال: مرتب‌ساز حبابی

```
Bubble-Sort (A)
  for i ← 1 to length[A] − 1
      do for j ← length[A] downto i + 1
             do if A[j] > A[j − 1]
                   then Swap (a[j], A[j − 1] )
```

مثال: مرتب‌ساز حبابی

```
BUBBLE-SORT (A)
for i ← 1 to length[A] − 1
    do for j ← length[A] downto i + 1
        do if A[j] > A[j − 1]
            then SWAP (a[j], A[j − 1] )
```

$$
\begin{aligned}
T(n) &= \sum_{i=۱}^{n-۱} \sum_{i+۱}^{n} \mathcal{O}(۱) \\
&= \sum_{i=۱}^{n-۱} c(n-i) \\
&= c[n(n-۱) - \frac{n(n-۱)}{۲}] = c\frac{n(n-۱)}{۲} = \mathcal{O}(n^{۲})
\end{aligned}
$$

تمرین: همین‌جا حل کنید!

زمان اجرای الگوریتم‌های زیر را محاسبه کنید:

```
MYSTRY (n)
for i ← 1 to n − 1
    do for j ← i + 1 to n
        do for k ← 1 to j
            do some O(1) statements
```

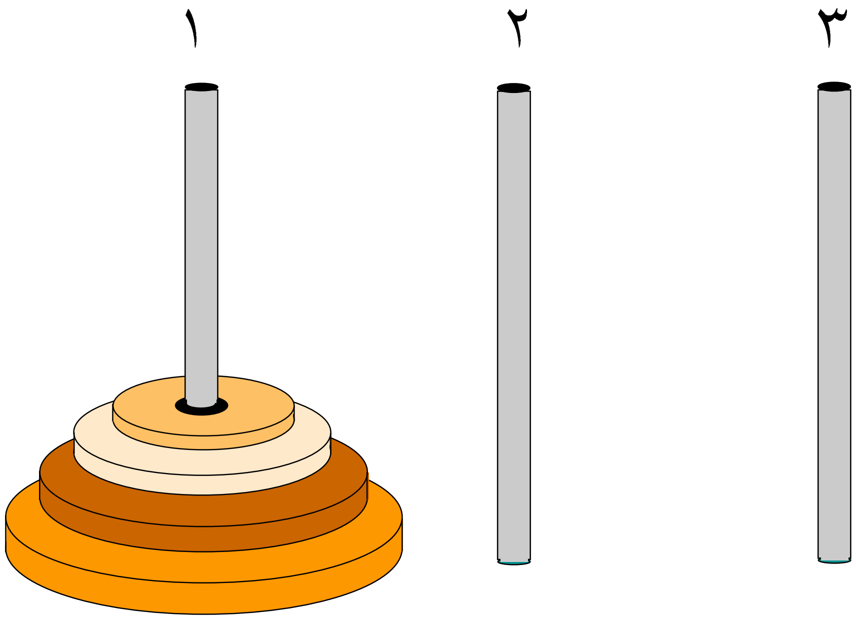
جواب: $\mathcal{O}(n^{۳})$

```
VeryOdd(n)
for i ← 1 to n
    do if odd(i)
          then for j ← 1 to n
                   do x ← x + 1
               for k ← 1 to i
                   do y ← y + 1
```

جواب: $\mathcal{O}(n^۲)$

الگوریتم‌های بازگشتی

مثال: برج‌های هانوی

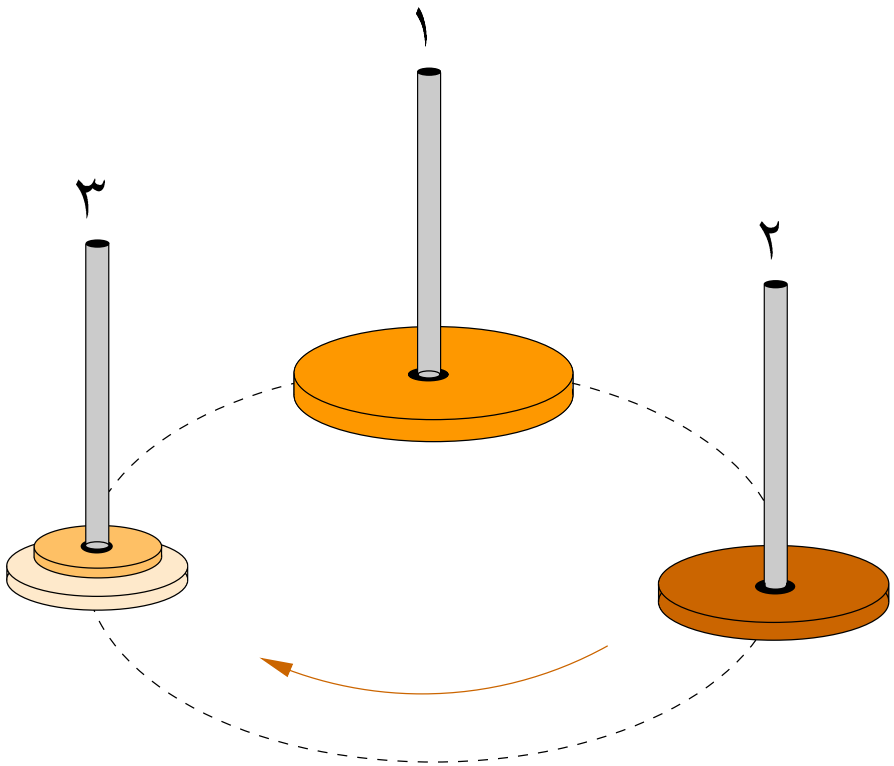


```
TOWER-OF-HONOI (n, f, t, h)
      ▷ moving n coins from leg f to leg t with the help of leg h
  if n = 1
     then Move the top coin from leg f to leg t
     else TOWER-OF-HONOI(n − 1, f, h, t)
          Move the top coin from leg f to leg t
          TOWER-OF-HONOI(n − 1, h, t, f)
```

$$T(n) = \begin{cases} ۱, & n = ۱ \\ T(n-۱) + ۱ + T(n-۱) & n > ۱ \end{cases}$$

یک رابطه‌ی بازگشتی (Recurrence Relation)

راه‌حل ترتیبی



راه‌حل ترتیبی

۱) اگر n زوج است «جهت حرکت» را ساعت‌گرد و گرنه پادساعت‌گرد فرض کن.

۲) تکرار کن تا همه‌ی سکه‌ها در میله‌ی مقصد قرار گیرند.

الف) کوچک‌ترین سکه را به‌میله‌ی بعدی در جهت حرکت ببر، و

ب) در صورت امکان تنها حرکتی را که شامل کوچک‌ترین سکه نباشد انجام بده.

چرا راه‌حل ترتیبی درست است؟

و تعداد حرکت‌ها چند تاست؟

تمرین

اگر در مسئله‌ی برج هانوی امکان انتقال سکه‌ها از میله‌ی ۱ به ۲ و یا از ۲ به ۱ نباشد، کم‌ترین تعداد انتقال n سکه از میله ۱ به ۲ چه‌قدر است؟

حل

$$T_{۱۲}(n) = \begin{cases} ۲, & n = ۱ \\ T_{۱۲}(n-۱) + ۱ + T_{۲۱}(n-۱) + ۱ + T_{۱۲}(n-۱) & n > ۱ \end{cases}$$

می‌دانیم $T_{۱۲} = T_{۲۱}$

اگر بخواهیم از ۱ به ۳ ببریم چه‌طور؟

حل

$$T_{۱۳}(n) = \begin{cases} ۱, & n = ۱ \\ T_{۱۲}(n-۱) + ۱ + T_{۲۳}(n-۱) & n > ۱ \end{cases}$$

می‌دانیم $T_{۲۳} = T_{۱۳}$

تمرین

- اگر در برج هانوی، سکه‌های با شماره‌ی فرد و زوج به‌ترتیب در میله‌های ۱ و ۲ باشند، با حداقل حرکت‌ها می‌خواهیم همه‌ی سکه‌ها را در میله‌ی ۳ قرار دهیم. چه‌گونه این کار را می‌شود انجام داد؟
- مسئله‌ی برج هانوی را اگر در ابتدا $n_۱$ سکه در میله‌ی ۱، $n_۲$ سکه در میله‌ی ۲ و بقیه‌ی n سکه در میله‌ی سوم باشد را حل کنید به‌طوری که در انتها همه‌ی سکه‌ها در میله‌ی سوم قرار گیرند. تعداد حرکت‌های بهینه را به‌دست آورید.

مثال: مرتب‌سازی ادغامی

<u>MERGE-SORT</u> (A, p, r)

$\triangleright$ A: آرایه‌ای که قرار است بین دو اندیس زیر مرتب شود

$\triangleright$ p: اندیس شروع

$\triangleright$ r: اندیس پایانی

if $p < r$

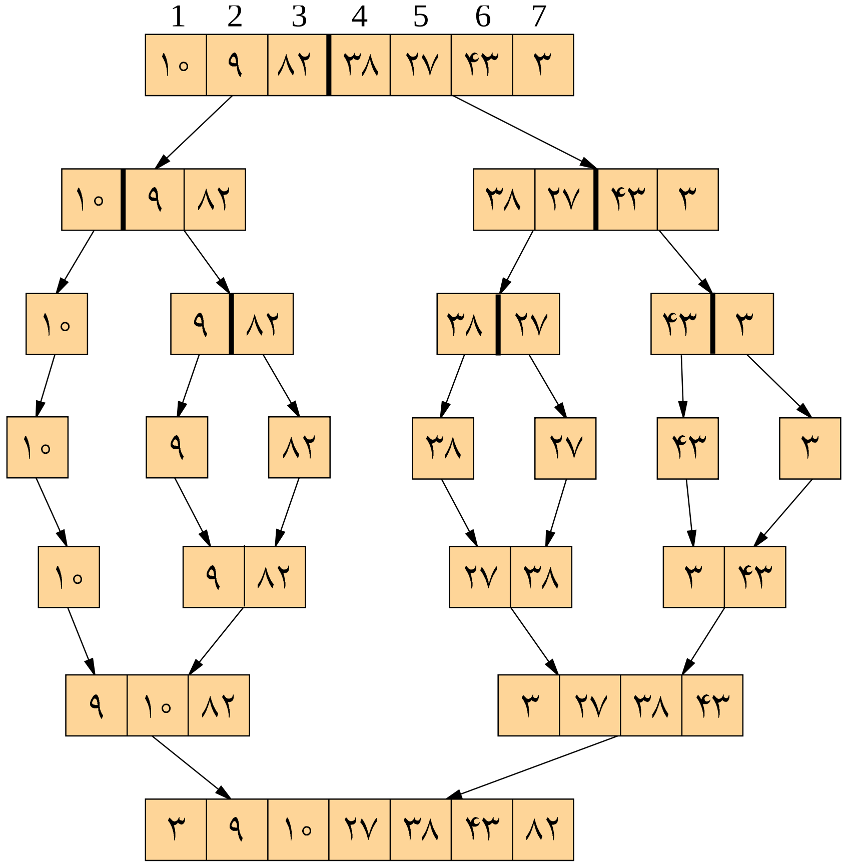
  then $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$

    MERGE-SORT(A, p, q)

    MERGE-SORT$(A, q+1, r)$

    MERGE (A, p, q, r)

ادغام دو آرایه‌ی مرتب: مثال


1 2 3 4 5 6 7
۱۰ ۹ ۸۲ ۳۸ ۲۷ ۴۳ ۳
۱۰ ۹ ۸۲
۳۸ ۲۷ ۴۳ ۳
۹ ۱۰ ۸۲
۳ ۲۷ ۳۸ ۴۳
۳ ۹ ۱۰ ۲۷ ۳۸ ۴۳ ۸۲

```
MERGE(A, B)
n ← length[A];   m ← length[B]
i ← 1;   j ← 1;   k ← 0
while i ≤ n or j ≤ m
      do k ← k + 1
         if j > m  or ((i ≤ n) and (A[i] ≤ B[j]))
           then C[k] ← A[i]
                i ← i + 1
           else C[k] ← B[j]
                j ← j + 1
length[C] ← n + m
return C
```

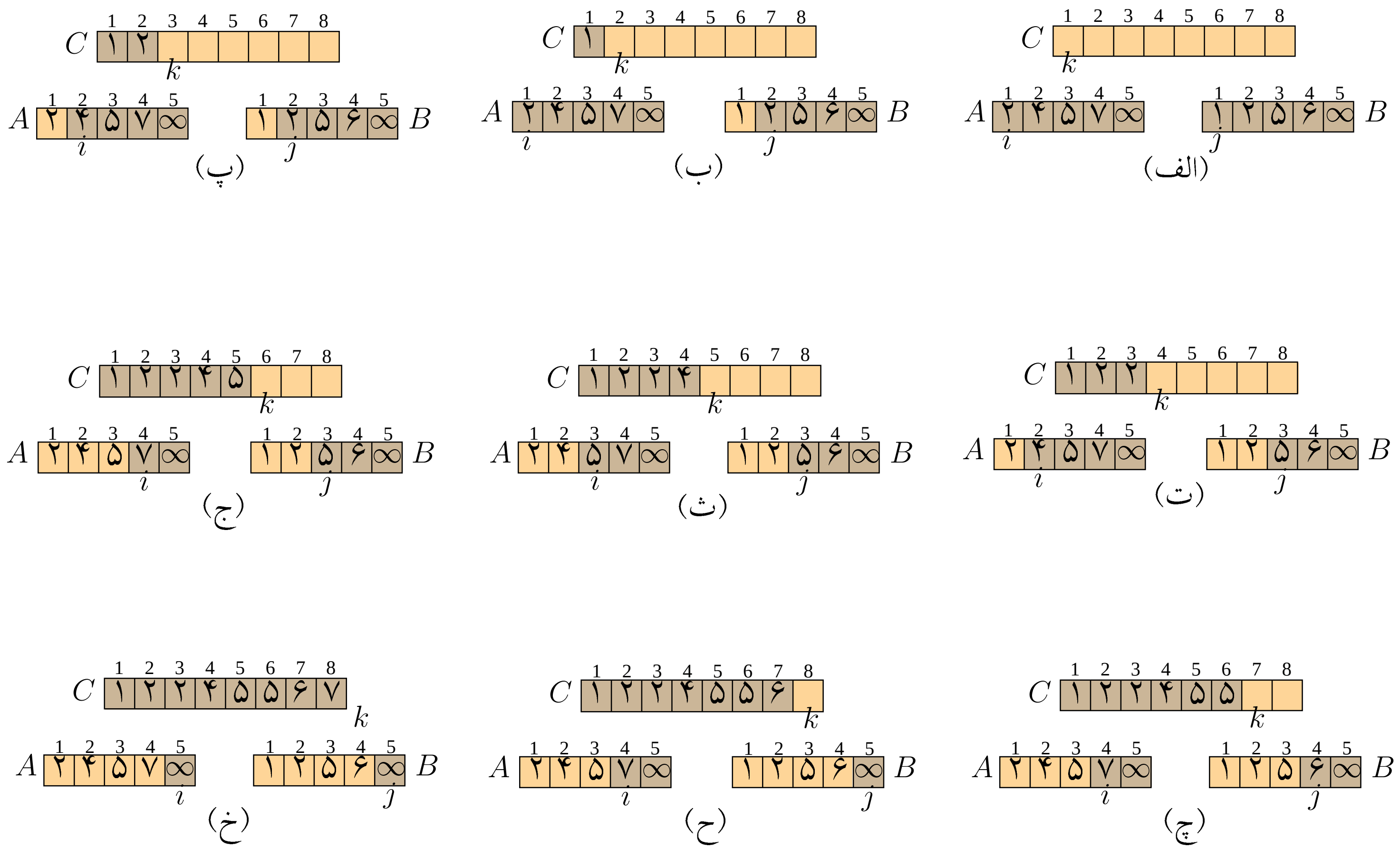
تعداد مقایسه‌های کلیدها:

در به‌ترین حالت: $\min\{n, m\}$ **(وقتی که همه‌ی عناصر یک آرایه از همه‌ی عناصر آرایه‌ی دیگر کم‌تر باشد)**

بدترین حالت: $n + m - ۱$ **(همه با هم مقایسه شوند؛ به‌ازای هر مقایسه یک عنصر در خروجی نوشته می‌شود، به‌جز عنصر آخر)**

ادغام: راه‌حل ساده‌تر

```
MERGE(A, B)
A[n+1] ← ∞;   B[m+1] ← ∞
i ← 1;   j ← 1
for k ← 1 to n+m
    do if A[i] < B[j]
          then C[k] ← A[i]
               i ← i+1
          else C[k] ← B[j]
               j ← j+1
length[C] ← n+m
return C
```


(الف)
(ب)
(پ)
(ت)
(ث)
(ج)
(چ)
(ح)
(خ)

ادغام دو لیست مرتب به طول‌های m و n در زمان $\mathcal{O}(m+n)$ و فضای $\mathcal{O}(m+n)$

ادغام درجا (in-place merging) **از فضای اضافی $\mathcal{O}(۱)$ استفاده می‌کند.**

آیا مرتب‌سازی ادغامی درجاست؟

مرتب‌سازی ادغامی

اثبات درستی الگوریتم بااستقرا

تحلیل:

$$T(n) = \begin{cases} ۱ & n = ۲ \\ T(\lfloor n/۲ \rfloor) + T(\lceil n/۲ \rceil) + cn & n > ۲ \end{cases}$$

نشان خواهیم داد که

$$T(n) = \Theta(n \lg n)$$

روش‌های حل رابطه‌های بازگشتی

۱) حدس و استقرا: حدس خوب و استقرا

۲) تکرار با جای‌گذاری: بازکردن فرمول و به‌دست آوردن جواب به‌طور صریح

۳) قضیه‌ی اصلی

۴) روش‌های حل رابطه‌های بازگشتی همگن و ناهمگن

۵) روش‌های دیگر مانند تابع مولد

حدس و استقرا: مثال

$$
\begin{aligned}
T(n) &= T(\lfloor \frac{n}{۲} \rfloor) + n \\
T(۲) &= T(۱) = \mathcal{O}(۱)
\end{aligned}
$$

حدس و استقرا:مثال

$$\begin{aligned} T(n) &= T(\lfloor \frac{n}{۲} \rfloor) + n \\ T(۲) &= T(۱) = \mathcal{O}(۱) \end{aligned}$$

حدس: $T(n) \leq Cn$ (چه‌گونه حدس می‌زنیم؟)

برای عدد مثبت C (برای آن‌که اثبات کنیم $T(n) = \mathcal{O}(n)$)

حدس و استقرا: مثال

$$
\begin{aligned}
T(n) &= T(\lfloor \frac{n}{۲} \rfloor) + n \\
T(۲) &= T(۱) = \mathcal{O}(۱)
\end{aligned}
$$

پایه‌ی استقرا:

$$n = ۲ \Rightarrow T(۲) \leq ۲C \Rightarrow ۲C \leq \mathcal{O}(۱) = r$$

فرض استقرا: برای $k < n$ فرض می‌کنیم $T(k) \leq Ck$

حکم استقرا: باید ثابت کنیم: $T(n) \leq Cn$.

$$\begin{aligned} T(n) &\leq C\lfloor \tfrac{n}{۲} \rfloor + n \\ &\leq Cn/۲ + n \end{aligned}$$

برای آن‌که $Cn/۲ + n \leq Cn$ **شود، باید داشته باشیم** $C \geq ۲$.

پس برای $۲ \leq C \leq r/۲$ **گام استقرا اثبات می‌شود. اگر** $C \geq ۱$ **حکم اثبات می‌شود.**

مثال: مرتب‌سازی ادغامی

$$
\begin{aligned}
T(۲) &= a \\
T(n) &= ۲T(\frac{n}{۲}) + bn
\end{aligned}
$$

مثال: مرتب‌سازی ادغامی

$$\begin{aligned} T(۱) = T(۲) &= a \\ T(n) &= ۲T(\frac{n}{۲}) + bn \end{aligned}$$

۱) **حدس:** $T(n) \leq cn\lg n \Rightarrow T(n) = \mathcal{O}(n\lg n)$

۲) **پایه:** $T(۲) = a \leq ۲c$

۳) فرض استقرا: برای $\frac{n}{۲}$ داریم، $T(n/۲) <= c\frac{n}{۲}\lg\frac{n}{۲}$

$$\begin{aligned} T(n) &\le ۲c\frac{n}{۲}\lg\frac{n}{۲} + bn \\ &= cn\lg n + (b-c)n \le cn\lg n \\ &\le cn\lg n,\ \ if\ \ b-c \le ۰\ \ and\ \ c \ge b, c \ge a/۲ \end{aligned}$$

برای این که اثبات کنیم $T(n) = \Theta(n\lg n)$ **باید اثبات کنیم که** $T(n) = \Omega(n\lg n)$ **هم هست.**

برای این کار باید اثبات کنیم که یک c **هست که برای** n **های بزرگ داریم،** $T(n) \ge cn\lg n$

پایه: $T(۲) = ۲ \geq ۲c$

فرض و حکم:

$$
\begin{aligned}
T(n) &\geq ۲c\frac{n}{۲}\lg\frac{n}{۲} + bn \\
&= cn\lg n + (b-c)n \leq cn\lg n \\
&\geq cn\lg n,\ \ if\ \ b-c \geq ۰\ \ and\ \ c \leq a/۲
\end{aligned}
$$

مثال

$$T(n) = T(\lfloor \frac{n}{۲} \rfloor) + T(\lceil \frac{n}{۲} \rceil) + ۱$$

مثال

$$T(n) = T(\lfloor \frac{n}{۲} \rfloor) + T(\lceil \frac{n}{۲} \rceil) + ۱$$

حدس: $T(n) = \mathcal{O}(n)$. **باید ثابت کنیم** $T(n) \leq Cn$.

مثال

$$T(n) = T(\lfloor \frac{n}{۲} \rfloor) + T(\lceil \frac{n}{۲} \rceil) + ۱$$

حدس: $T(n) = \mathcal{O}(n)$. **باید ثابت کنیم** $T(n) \leq Cn$.

$$T(n) \leq C\lfloor \frac{n}{۲} \rfloor + C\lceil \frac{n}{۲} \rceil + ۱ = Cn + ۱ \geq Cn$$

اشتباه!

حدس دیگر: $T(n) \leq Cn + B$.

$$T(n) \leq C\lfloor\frac{n}{۲}\rfloor + C\lceil\frac{n}{۲}\rceil + ۲B + ۱ = Cn + ۲B + ۱ \leq Cn + B \Rightarrow B < ۰$$

مثال دیگر از حدس اشتباه

$$T(n) = ۲T(\lfloor \frac{n}{۲} \rfloor) + n$$

مثال دیگر از حدس اشتباه

$$T(n) = ۲T(\lfloor \frac{n}{۲} \rfloor) + n$$

حدس: $T(n) \leq Cn$

مثال دیگر

$$T(n) = ۲T(\lfloor\frac{n}{۲}\rfloor) + n$$

حدس: $T(n) \leq Cn$

$$T(n) \leq ۲C\lfloor\frac{n}{۲}\rfloor + n \leq Cn + n = (C + ۱)n \geq Cn$$

اشتباه!

حدس جدید: $T(n) \leq Cn\lg n + B$

درست است؟

می‌توان نشان داد که اگر $n = ۲^k$ فرض کنیم، حل رابطه‌ی بازگشتی با این فرض از نظر O همان جواب است. چرا؟

برای هر $۲^k \leq n < ۲^{k+۱}$ می‌توان نشان داد که مرتبه‌ی $T(n)$ همان مرتبه‌ی $T(۲^k)$ است.

مثال

$$T(n) = ۲T(\sqrt{n}) + \lg n$$

مثال

$$T(n) = ۲T(\sqrt{n}) + \lg n$$

متغیر جدید: $m = \lg n \Leftarrow T(۲^m) = ۲T(۲^{\frac{m}{۲}}) + m$.

$$S(m) = ۲S(\frac{m}{۲}) + m$$

مثال

$$T(n) = ۲T(\sqrt{n}) + \lg n$$

$$S(m) = \mathcal{O}(m \lg m) \Rightarrow T(n) = \mathcal{O}(\lg n \lg \lg n)$$

مسئله: پیدا کردن کوچک‌ترین و بزرگ‌ترین عناصر

پیدا کردن کوچک‌ترین و بزرگ‌ترین عناصر آرایه‌ی n تای A با کم‌ترین تعداد مقایسه‌ها

پیدا کردن کوچک‌ترین و بزرگ‌ترین عناصر (ادامه)

این مقدار $\lceil \frac{3n}{2} \rceil - 2$ است. چرا؟

پیدا کردن کوچک‌ترین و بزرگ‌ترین عناصر: روش اول

- بزرگترینِ n عنصر با $n - ۱$ مقایسه
- کوچکترین عنصر با $n - ۲$ مقایسه
- در کل $۲n - ۳$ مقایسه

کوچک‌ترین و بزرگ‌ترین عناصر: روش تقسیم‌وحل

```
MaxMin(A, p, q)
      ▷ A: array, p, q: the first and last indices
      ▷ will return the indices of both max and min
if p=q
  then max ← min ← A[p]
  else if q − p = 1
         then if A[p] > A[q]
                then max ← A[p] ; min ← A[q]
                else max ← A[q] ; min ← A[p]
         else r ← ⌊(p+q)/2⌋
              min1, max1 ← MaxMin(A, p, r)
              min2, max2 ← MaxMin(A, r + 1, q)
              if max2 > max1
                then max ← max2
                else max ← max1
              if min2 < min1
                then min ← min2
                else min ← min1
```

کوچک‌ترین و بزرگ‌ترین عناصر: روش تقسیم‌وحل

$$T(n) = \begin{cases} ۰ & n = ۱ \\ ۱ & n = ۲ \\ T(\lfloor n/۲ \rfloor + T(\lceil n/۲ \rceil) + ۲ & n > ۲ \end{cases}$$

این راه‌حل چه‌گونه است؟

این راه حل نیز بهینه نیست!

$T(۶) = ۸$ در حالی که ۷ مقایسه $(۲ - \lceil \frac{۱۸}{۲} \rceil)$ کافی است.

برای $n = ۲^k$ بهینه است.

$$T(۲^k) = ۲[۳ \times ۲^{k-۲} - ۲] - ۲ = ۳ \times ۲^{k-۱} - ۲$$

کوچک‌ترین و بزرگ‌ترین عناصر: راه‌حل بهینه‌ی ۱

۱) با یک مقایسه بین هر دو عنصر متوالی Min ها و Max های هر زوج را پیدا کن

۲) کوچک‌ترین عنصر Min ها و حداکثر یک عنصر اضافی، کوچک‌ترین عنصر است

۳) بزرگ‌ترین عنصر Max ها و حداکثر یک عنصر اضافی، بزرگ‌ترین عنصر است.

کوچک‌ترین و بزرگ‌ترین عناصر: تحلیل راه‌حل بهینه‌ی ۱

- **بهینه برای $n = ۲k$ با $۳n/۲ - ۲ = (n/۲ - ۱) + (n/۲ - ۱) + n/۲$ مقایسه**
- **برای $n = ۲k + ۱$**

۱) k زوج دوتایی

۲) با k مقایسه k عدد Min و k عدد Max به‌دست می‌آید.

۳) با یک عنصر اضافی، k مقایسه برای یافتن عنصر کمینه و k تا برای عنصر بیشینه

$۳k = \lfloor \frac{۳n}{۲} \rfloor$ مقایسه لازم است. ولی، این بهینه است چون

$$\lceil \frac{۳(۲k+۱)}{۲} \rceil - ۲ = ۳k + \lceil \frac{۳}{۲} \rceil - ۲ = ۳k$$

کوچک‌ترین و بزرگ‌ترین عناصر: راه‌حل بهینه‌ی ۲

۱) تقسیم آرایه به ۲ و $n-۲$ عنصر

۲) پیدا کردن کوچکترین و بزرگترین عنصرهای هر قسمت با $T(n-۲)+۱$ مقایسه

۳) ۲ مقایسه‌ی دیگر

$$T(n)=\begin{cases} ۰ & n=۱ \\ ۱ & n=۲ \\ T(n-۲)+۳ & n>۲ \end{cases}$$

که جواب آن همان مقدار بهینه است. چرا؟

این الگوریتم همان راه‌حل بهینه‌ی اول است!

روش‌های حل رابطه‌ی بازگشتی: تکرار با جای‌گذاری

مثال:

$$T(n) = ۳T(\lfloor \frac{n}{۴} \rfloor) + n$$

تکرار با جای‌گذاری

مثال:

$$T(n) = ۳T(\lfloor \frac{n}{۴} \rfloor) + n$$

$$T(n) \; = ۳T(\lfloor \tfrac{n}{۴} \rfloor) + n$$

تکرار با جای‌گذاری

مثال:

$$T(n) = ۳T(\lfloor \frac{n}{۴} \rfloor) + n$$

$$\begin{aligned} T(n) &= ۳T(\lfloor \tfrac{n}{۴} \rfloor) + n \\ &= ۳^{۲}T(\lfloor \tfrac{\lfloor \frac{n}{۴} \rfloor}{۴} \rfloor) + ۳\lfloor \tfrac{n}{۴} \rfloor + n \end{aligned}$$

تکرار با جای‌گذاری

مثال:

$$T(n) = ۳T(\lfloor \frac{n}{۴} \rfloor) + n$$

$$\begin{aligned}
T(n) &= ۳T(\lfloor \tfrac{n}{۴} \rfloor) + n \\
&= ۳^۲T(\lfloor \tfrac{\lfloor \frac{n}{۴} \rfloor}{۴} \rfloor) + ۳\lfloor \tfrac{n}{۴} \rfloor + n \\
&= ۳^۲T(\lfloor \tfrac{n}{۴^۲} \rfloor) + ۳\lfloor \tfrac{n}{۴} \rfloor + n \\
&= ۳^۳T(\lfloor \tfrac{n}{۴^۳} \rfloor) + ۳^۲\lfloor \tfrac{n}{۴^۲} \rfloor + ۳\lfloor \tfrac{n}{۴} \rfloor + n \\
&= \ldots \\
&\leq ۳^iT(\tfrac{n}{۴^i}) + n\sum_{j=۰}^{i-۱} (\frac{۳}{۴})^j
\end{aligned}$$

داریم، $\frac{n}{۴^i} = ۱ \Rightarrow i = \log_۴ n$،

$$T(n) \le ۳^{\log_۴ n} * T(۱) + n \sum_{j=۰}^{\log_۴ n - ۱} (\frac{۳}{۴})^j$$

اما می‌دانیم که

$$\lim_{n\to\infty} \sum_{j=۰}^{\lg n - ۱} (\frac{۳}{۴})^j = ۴ \Rightarrow C_۲ < ۴$$

و داریم، $۳^{\log_۴ n} = n^{\log_۴ ۳}$.

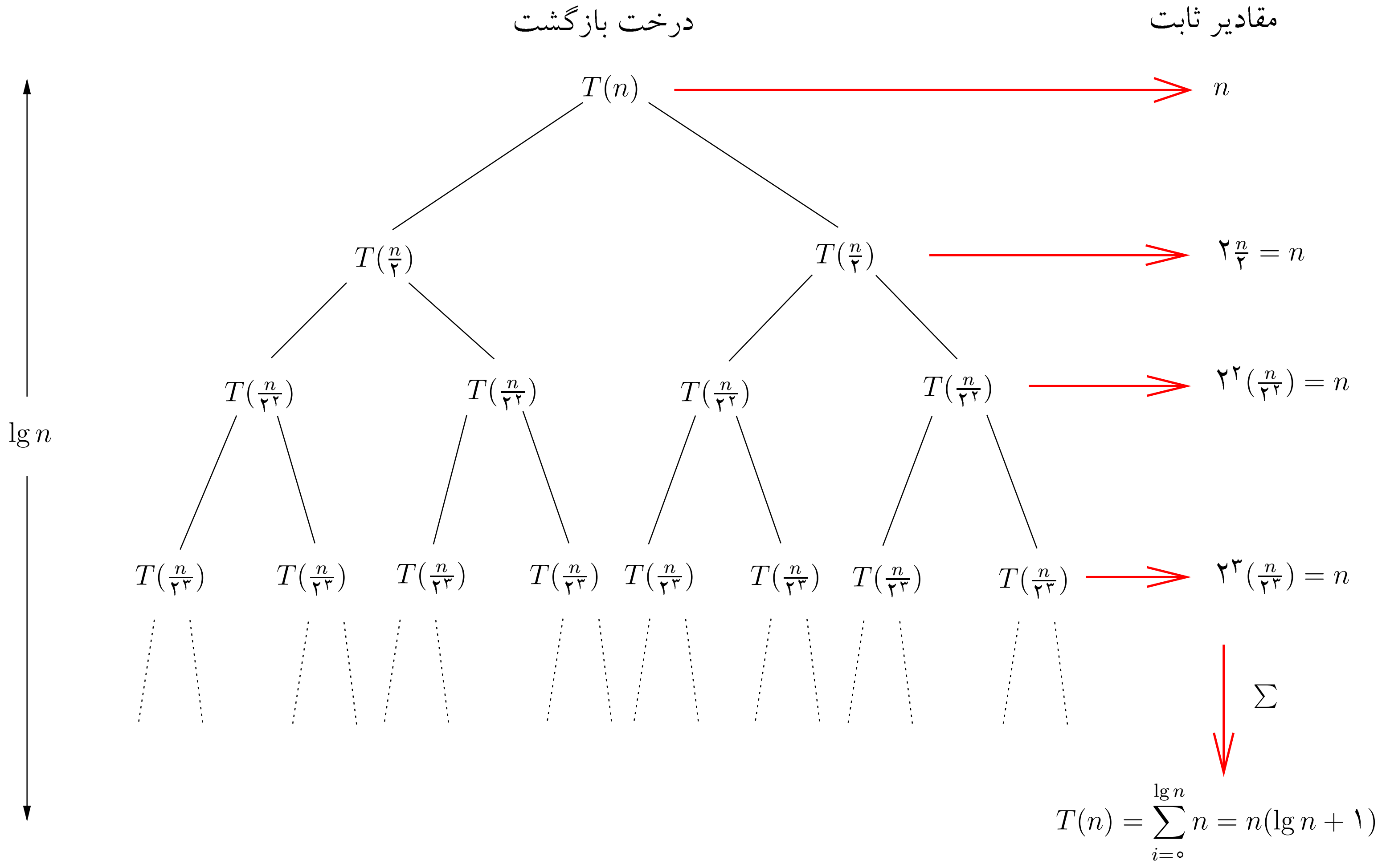
$$T(n) \le Cn^{\log_۴ ۳} + ۴n \Rightarrow T(n) = \mathcal{O}(n)$$

مثال: $F(n) = F(n-۱) + F(n-۲)$ و $F(۱) = F(۲) = ۱$.

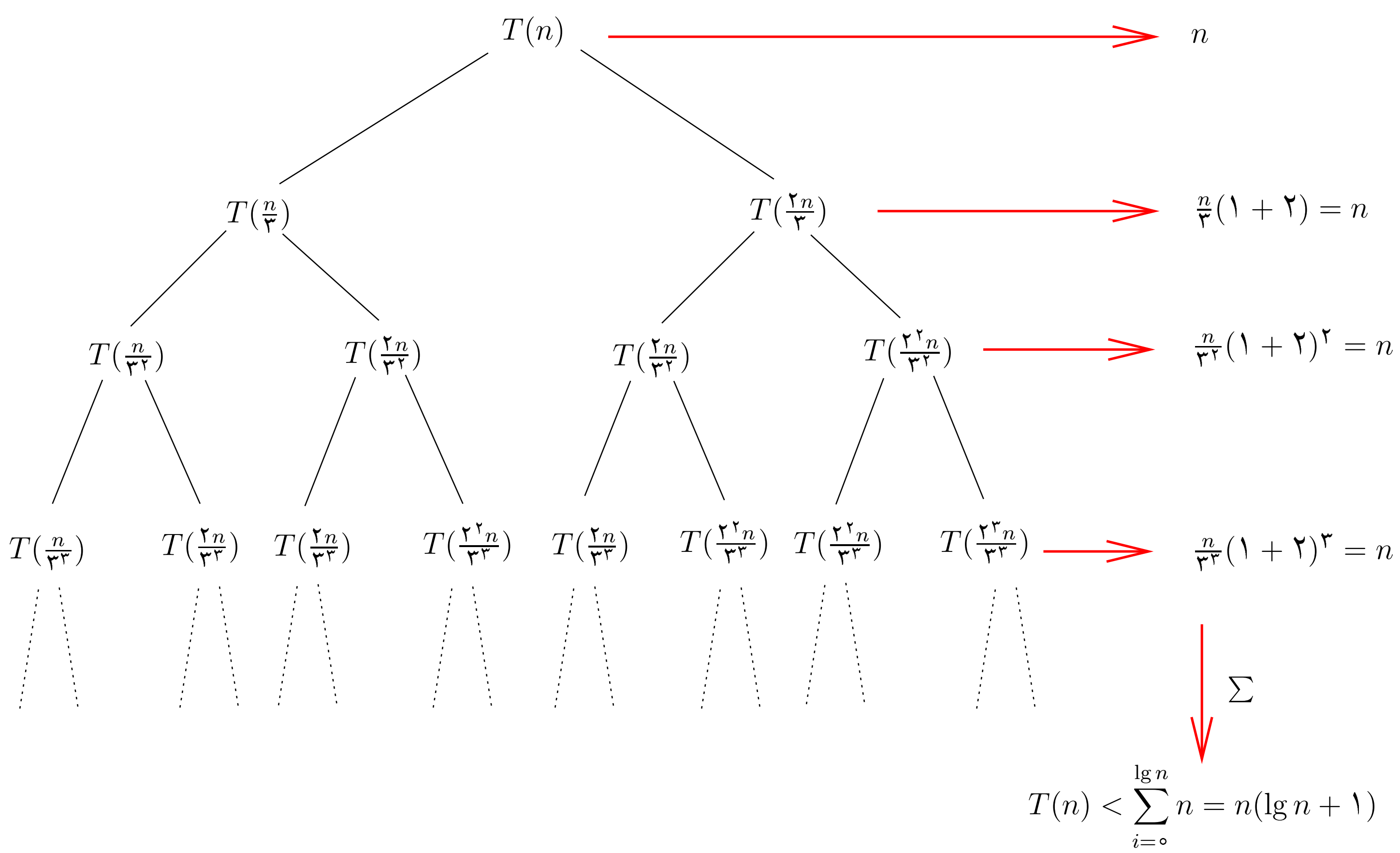
از روش حل رابطه‌های همگن استفاده می‌شود.

درخت بازگشت (Recursion Tree)

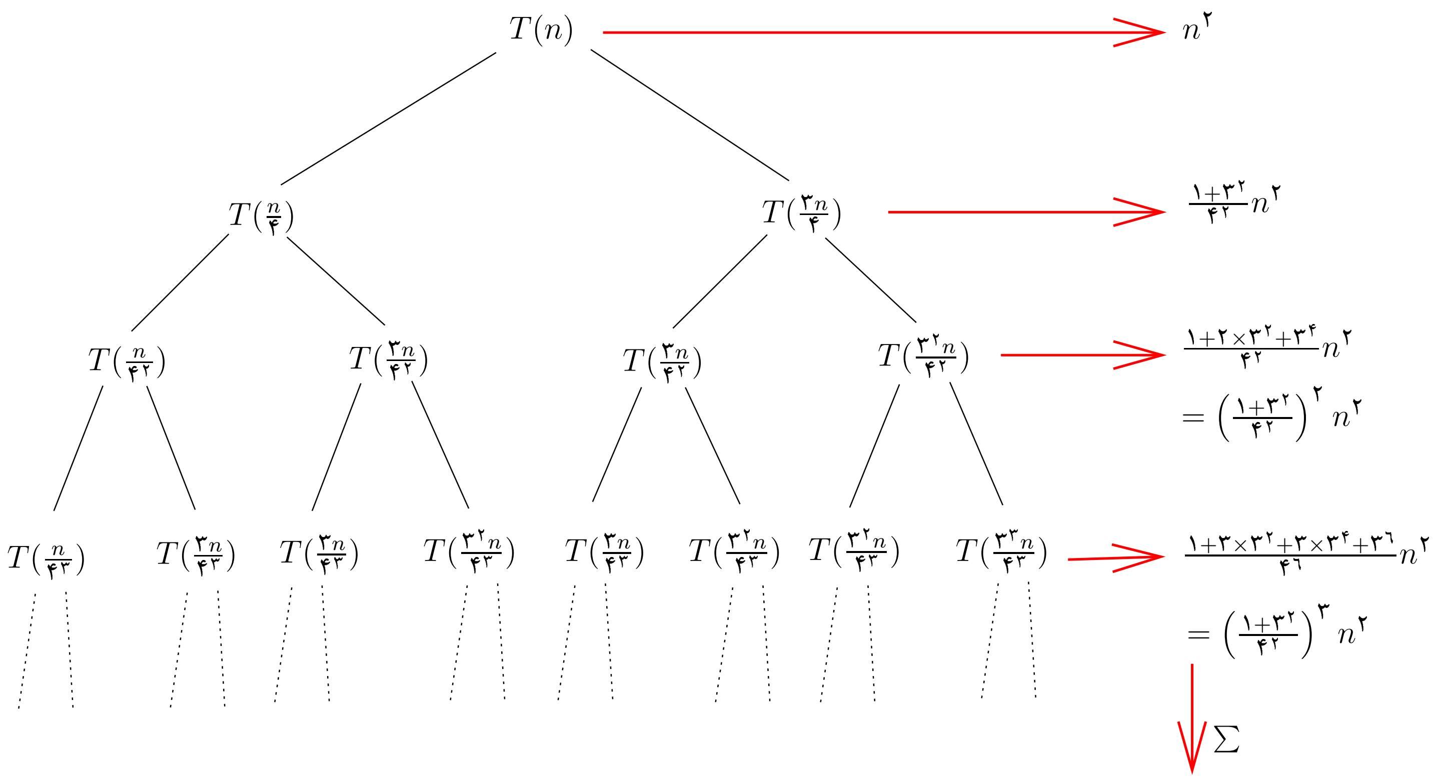
$$T(n) = ۲T(\frac{n}{۲}) + n$$



$$T(n) = T(\frac{n}{۳}) + T(\frac{۲n}{۳}) + n$$



$$T(n) = T(\frac{n}{4}) + T(\frac{3n}{4}) + n^2$$



$$T(n) < n^2 \sum_{i=0}^{\log_{\frac{4}{3}} n} \left(\frac{1+3^2}{4^2}\right)^i$$

$$< cn^2 = \mathcal{O}(n^2)$$

قضیه‌ی اصلی

برای $a \geq ۱, b > ۱$ و تابع $f(n)$ حل رابطه‌ی بازگشتی $T(n) = aT(\frac{n}{b}) + f(n)$ (که در آن $\frac{n}{b}$ می‌تواند به صورت کف یا سقف باشد) به قرار زیر است:

الف - اگر $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$، برای $\epsilon > ۰$ (یعنی رشد تابع $f(n)$ از تابع $n^{\lg a}$ به صورت چند جمله‌ای کمتر باشد)، در این صورت، $T(n) = \Theta(n^{\log_b a})$

ب - اگر $f(n) = \Theta(n^{\log_b a})$، دراین صورت، $T(n) = \Theta(n^{\log_b a} \log_۲ n)$

ج - اگر $f(n) = \Omega(n^{\log_b a + \epsilon})$، دراین صورت، $T(n) = \Theta(f(n))$.

اگر G **درجه‌ی رشد تابع** $g(n) = n^{\log_b a}$ **و** F **درجه‌ی رشد تابع** $f(n)$ **باشد، خواهیم داشت:**

۱) اگر $F > G$، $T(n) = \Theta(f(n))$

۲) اگر $G > F$، $T(n) = \Theta(g(n))$

۳) اگر $F = G$، $T(n) = \Theta(g(n) \lg n)$.

مثال. $T(n) = ۹T(\frac{n}{۳}) + n$

حل: داریم، $f(n) = n$ **و** $g(n) = n^{\log_۳ ۹} = n^۲$**. بدیهی است که درجه‌ی رشد** $n^۲$ **از** n **به‌صورت چند جمله‌ای بیشتر است. لذا،**

$$f(n) = \mathcal{O}(n^{۲-۱}) \Rightarrow T(n) = \Theta(n^۲)$$

مثال. $T(n) = T(\frac{2n}{3}) + 1$

حل: داریم، $g(n) = n^{\log_{\frac{3}{2}} 1} = n^{0} = 1$ **و** $f(n) = 1$. **پس،** $T(n) = \Theta(\lg n)$.

مثال. $T(n) = 3T(\frac{n}{4}) + n \lg n$

حل: چون، $f(n) = n \lg n$ **و** $g(n) = n^{\log_4 3} = n^{0.793}$ **و** $f(n) = \Omega(n^{0.793+0.1})$، **داریم** $T(n) = \Theta(n \lg n)$.

مثال. $T(n) = 2T(\frac{n}{2}) + n\lg n$

حل: برای این مثال، $n = n^{\lg 2} = g(n)$ و $f(n) = n\lg n = (n)$. ولی اختلاف به‌صورت نمایی نیست یعنی هیچ ϵ ای نمی‌توان پیدا کرد که برای کلیه‌ی n های بزرگ، رابطه‌ی $n\lg n > n^{1+\epsilon}$ برقرار باشد. بنابراین این مسئله را باید از روش دیگری، مثلاً استقرا حل کرد.

جواب $T(n) = \mathcal{O}(n \lg^۲ n)$ **است که به‌صورت زیر با استقرا اثبات می‌شود.**

پایه: $T(۲) \leq ۲c$

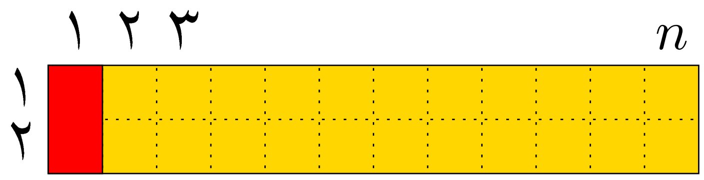
فرض: $T(k) \leq ck \lg^۲ k$ **برای** $k < n$.

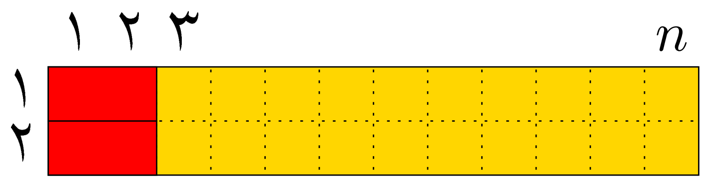
حکم:

$$\begin{aligned} T(n) &\leq ۲c\frac{n}{۲}\lg^۲ \frac{n}{۲} + n\lg n \\ &\leq cn(\lg n - ۱)^۲ + n\lg n \\ &\leq cn\lg^۲ n + n[c + (۱ - ۲c)\lg n] \end{aligned}$$

برای $c = ۱$ **و** $n > ۲$ **داریم** $c + (۱ - ۲c)\lg n < ۰$ **و حکم اثبات می‌شود.**

روابط بازگشتی همگن





به چند طریق می‌توان صفحه‌ای $۲ \times n$ را با موزاییک‌های ۲×۱ فرش کرد؟

حل: **اگر جدول $n \times ۲$ را بتوان با $f_{n+۱}$ روش مختلف فرش کرد، داریم:**

$$f_n = f_{n-۱} + f_{n-۲}$$

و $f_۰ = ۰$ و $f_۱ = ۱$ (دنباله‌ی فیبوناچی).

به چند طریق می‌توان صفحه‌ای $n \times ۳$ را با موزاییک‌های ۱×۲ فرش کرد؟

روابط بازگشتی همگن

اگر c_iهاعددهای حقیقی باشند، به رابطه‌ی بازگشتی زیر

رابطه‌ی بازگشتی همگن از درجه‌ی k

می‌گوییم:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$$

تابع $g(n)$ یک جواب دنباله‌ی بازگشتی فوق است، اگر دنباله‌ی $a_n = g(n)$ $(n \in \mathcal{N})$ در رابطه‌ی بازگشتی صدق کند.

قضیه: اگر $g_i(n)$ $(i = ۱, \ldots, r)$ هر یک جوابی برای

$$a_n = c_۱ a_{n-۱} + c_۲ a_{n-۲} + \cdots + c_k a_{n-k}$$

باشد، آن‌گاه هر ترکیب خطی از این r جواب یا

$$A_۱ g_۱(n) + A_۲ g_۲(n) + \cdots + A_r g_r(n)$$

که در آن A_iها اعدادی حقیقی‌اند، پاسخی برای رابطه‌ی بازگشتی است.

اثبات: فرض می‌کنیم،

$$h(n) = A_۱ g_۱(n) + A_۲ g_۲(n) + \cdots + A_r g_r(n)$$

، چون $g_i(n)$ **یک جواب** $a_n = c_۱ a_{n-۱} + c_۲ a_{n-۲} + \cdots + c_k a_{n-k}$ **است، پس داریم:**

$$g_i(n) = c_۱ g_i(n-۱) + c_۲ g_i(n-۲) + \cdots + c_k g_i(n-k)$$

پس نتیجه می‌شود:

$$h(n) = c_۱ h(n-۱) + c_۲ h(n-۲) + \cdots + c_k h(n-k)$$

بنابراین حکم قضیه ثابت است.

روش حل

اگر $g(n) = x^n$ جواب رابطه‌ی بازگشتی همگن باشد، داریم:

$$x^n - c_1 x^{n-1} - c_2 x^{n-2} - \cdots - c_k x^{n-k} = 0$$

یا به عبارت دیگر،

$$x^k - c_1 x^{k-1} - \cdots - c_k = 0$$

یعنی x جواب معادله درجه k فوق است.

این معادله را معادله‌ی مشخصه (characteristic equation) **برای رابطه‌ی بازگشتی می نامیم.**

اگر x_i، ریشه‌ی معادله‌ی مشخصه باشد، بدیهی است که $a_n = x_i^n$ یک جواب رابطه‌ی بازگشتی است و بنا بر قضیه‌ی قبل هر ترکیب خطی از x_i^n ها هم یک جواب رابطه‌ی

بازگشتی است.

به طور مثال ترکیب خطی:

$$a_n = t_۱ x_۱^n + t_۲ x_۲^n + \cdots + t_k x_k^n$$

در این رابطه‌ی بازگشتی باید مقادیر k عنصر اول این دنباله داده شده باشند، پس:

$$\begin{cases} a_۰ & = & t_۱ & + & t_۲ & + & \cdots & + & t_k \\ a_۱ & = & t_۱ x_۱ & + & t_۲ x_۲ & + & \cdots & + & t_k x_k \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{k-۱} & = & t_۱ x_۱^{k-۱} & + & t_۲ x_۲^{k-۱} & + & \cdots & + & t_k x_k^{k-۱} \end{cases}$$

k معادله و k مجهول می‌باشد.(مجهول‌ها $t_۱$ تا t_k هستند.)

اگر x_iها متمایز باشند این دستگاه معادلات یک جواب منحصربه‌فرد دارد، یعنی بادادن k عنصر اول دنباله،میتوان جوابی منحصربه‌فرد برای دنباله پیدا کرد.

مثال: دنباله‌ی فیبوناچی را حل کنید.

حل: معادله‌ی مشخصه: $x^۲ - x - ۱ = ۰$

ریشه‌های آن $x_۱ = \frac{۱+\sqrt{۵}}{۲}$ **و** $x_۲ = \frac{۱-\sqrt{۵}}{۲}$

$$f_n = t_۱ \left(\frac{۱ + \sqrt{۵}}{۲}\right)^n + t_۲ \left(\frac{۱ - \sqrt{۵}}{۲}\right)^n$$

و با توجه به مقادیر اولیه داریم:

$$\begin{cases} t_۱ + t_۲ = f_۰ = ۰ \\ t_۱(\frac{۱+\sqrt{۵}}{۲}) + t_۲(\frac{۱-\sqrt{۵}}{۲}) = f_۱ = ۱ \end{cases}$$

و از این معادله‌ها نتیجه میشود:

$$t_۱ = \frac{۱}{\sqrt{۵}}, t_۲ = -\frac{۱}{\sqrt{۵}}$$

$$f_n = \frac{۱}{\sqrt{۵}}\left(\left(\frac{۱+\sqrt{۵}}{۲}\right)^n - \left(\frac{۱-\sqrt{۵}}{۲}\right)^n\right)$$

جمله‌ی $(\frac{۱-\sqrt{۵}}{۲})^n$ بابزرگتر شدن n بسیار کوچک می‌شود و با توجه به اینکه f_n عددی حسابی است، اگر $\langle x\rangle$ را نزدیکترین عدد صحیح به x تعریف کنیم، داریم:

$$\langle x\rangle = \lfloor x + \frac{۱}{۲}\rfloor$$

و با این تعریف:

$$f_n = \langle\frac{۱}{\sqrt{۵}}\left(\frac{۱+\sqrt{۵}}{۲}\right)^n\rangle$$

اگر x_i ریشه‌ی مضاعف درجه‌ی ۲ معادله‌ی مشخصه باشد

$a_n = nx_i^n$ نیز یک جواب رابطه‌ی بازگشتی است (اثبات با مشتق‌گیری از معادله‌ی مشخصه است، به این وسیله که ریشه‌ی مضاعف، ریشه‌ی مشتق معادله‌ی مشخصه است.)

اگر x_i ریشه‌ی مضاعف درجه ۳ باشد

$n^2 x_i^n$ نیز یک جواب رابطه‌ی بازگشتی است.

در حالت کلی

اگر x_i ریشه‌ی مضاعف درجه p باشد،

$$g(n) = t_0 x^n + t_1 n x^n + t_2 n^2 x^n + \cdots + t_{p-1} n^{p-1} x^n$$

جوابی برای رابطه‌ی بازگشتی است.

مثال. رابطه‌ی بازگشتی زیر را حل کنید:

$$a_n = 5a_{n-1} - 8a_{n-2} + 4a_{n-3} \ (n \geq 3)$$

و داریم: $a_0 = 0$، $a_1 = 1$، و $a_2 = 2$.

حل:

معادله مشخصه:

$$x^۳ - ۵x^۲ + ۸x - ۴ = ۰ = (x - ۱)(x - ۲)^۲$$

بنابراین جواب کلی به این صورت است: $a_n = c_۱ ۱^n + c_۲ ۲^n + c_۳ n ۲^n$

که با اعمال مقادیر اولیه داریم

$$\begin{aligned} c_۱ + c_۲ &= ۰ \quad & n = ۰ \\ c_۱ + ۲c_۲ + ۲c_۳ &= ۱ \quad & n = ۱ \\ c_۱ + ۴c_۲ + ۸c_۳ &= ۲ \quad & n = ۲ \end{aligned}$$

که جواب آن $c_۱ = -۲$، $c_۲ = ۲$، $c_۳ = -۱/۲$ **است. پس**

$$a_n = ۲^{n+۱} - n۲^{n-۱} - ۲$$

تحلیل سرشکنی (Amortized Analysis)

- تحلیل بدترین حالت
- تحلیل در حالت متوسط

محیط: یک داده‌ساختار و دنباله‌ای از عملیات بر روی آن

هزینه‌ی سرشکن‌شده هر عمل: متوسط هزینه‌ی آن عمل (در بدترین حالت)

مثال ۱: پشته با عمل multiPoP

علاوه بر Push و Pop

```
Multipop(S, k)
  while  not Stack-Empty(S)  and k ≠ 0
        do Pop(S)
           k ← k − 1
```

هزینه‌ها در بدترین حالت:

- Push و Pop: $\Theta(1)$
- $\textsc{Multipop}(S, k)$: $\Theta(\min\{k, \textsc{Size}(S)\})$

پس اگر دنباله‌ای از n تا از این اعمال داشته باشیم،

جمع کل هزینه‌ها در بدترین حالت: $\Theta(n^2)$.

نشان می‌دهیم که هزینه‌ی سرشکن‌شده هر عمل $O(1)$ است.

مثال ۲: افزایش شمارنده‌ی دودویی

یک شمارنده‌ی دودویی k بیتی $A[0..k-1]$

(بیت ۰ کم‌ارزش‌ترین بیت)

```
INCREMENT(A)
i ← 0
while i < length[A] and A[i] = 1
      do A[i] ← 0
         i ← i + 1
if i < length[A]
  then A[i] ← 1
```

هزینه‌ی شمارنده در بدترین حالت

هر عمل افزایش در بدترین حالت $\mathcal{O}(k)$،

n عمل افزایش $\mathcal{O}(nk)$

نشان می‌دهیم که هزینه‌ی هر عمل افزایش به‌صورت سرشکن‌شده $\mathcal{O}(۱)$ است.

روش‌های تحلیل سرشکن‌شده

- روش انبوهه (aggregate): جمع هزینه‌های اعمال، تقسیم بر تعداد.
- روش حساب‌داری (accounting): استفاده از یک مخزن پول و پرداخت برای هر عمل
- روش تابع پتانسیل (potential): حالت کلی‌تر روش قبل

تحلیل پشته با روش مجموع

تحلیل پشته با روش مجموع

اگر n عمل داشته باشیم:

- حداکثر تعداد عناصر پشته n است.
- هر عنصر داخل پشته دقیقاً یک بار Push و حداکثر یک بار Pop می‌شود.
- یک عنصر یا مستقیماً Pop می‌شود و یا با Multipop
- هر عمل Push و Pop $\mathcal{O}(1)$ است.

جمع هزینه‌ها حداکثر $\Theta(2n)$ است

پس هزینه‌ی سرشکن‌شده‌ی هر عمل $\mathcal{O}(1)$ است.

تحلیل شمارنده با روش انبوهه

تحلیل شمارنده با روش انبوهه

- هر عمل افزایش حداکثر یک بیت را ۱ می‌کند و تعدادی را از ۱ به ۰ تغییر می‌دهد
- بیت ۰ با هر افزایش flip می‌شود (n بار)
- بیت ۱ یک در میان flip می‌شود ($n/۲$ بار)
- بیت ۲ هر ۴ بار یک دفعه تغییر می‌کند ($n/۴$)
- ...
- بیت i پس از ۲^i بار افزایش تغییر می‌کند (در مجموع $\lfloor \frac{n}{۲^i} \rfloor$ بار)

پس در مجموع

$$\sum_{i=۰}^{k-۱} \lfloor \frac{n}{۲^i} \rfloor \leq n \sum_{i=۰}^{\infty} \frac{۱}{۲^i} \leq ۲n$$

یعنی هزینه‌ی سرشکن‌شده‌ی هر عمل $\mathcal{O}(۱)$ است.

تحلیل پشته به‌روش حساب‌داری

- برای هر Push ۲ ریال خرج می‌کنیم.
- ۱ ریال صرف عمل Push می‌شود.
- ۱ ریال را بر روی عنصر داخل پشته می‌گذاریم.
- Pop ها همه مجانی خواهند بود.

کلاً $۲n$ ریال هزینه پس هزینه‌ی سرشکن‌شده $\mathcal{O}(۱)$ است.

تحلیل شمارنده به‌روش حساب‌داری

تحلیل شمارنده به‌روش حساب‌داری

- هر عمل افزایش ۲ ریال هزینه می‌کنیم.
- ۱ ریال صرف تبدیل یک بیت از ۰ به ۱ می‌شود.
- ۱ ریال برروی بیت ۱ قرار می‌دهیم.
- هزینه‌ی ۱ به ۰ مجانی خواهد بود.

روش تابع پتانسیل

- D_0 : داده‌ساختار اولیه
- D_i: داده‌ساختار پس از عمل i ام

$$D_0 \xrightarrow{\text{عمل } 1} D_1 \xrightarrow{\text{عمل } 2} D_2 \ldots \xrightarrow{\text{عمل } n} D_n$$

- c_i: هزینه‌ی واقعی عمل i ام
- تعریف می‌کنیم: $\Phi(D_i) =$ تابع پتانسیل که D_i به عدد حقیقی نگاشت می‌کند.

- **تعریف: $\hat{c}$ هزینه‌ی سرشکن‌شده‌ی عمل i ام**

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

- **پس** $\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$
- **اگر** $\Phi(D_0) = 0$،

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

یعنی $\sum_{i=1}^{n} \hat{c}_i$ **کران بالایی برای** $\sum_{i=1}^{n} c_i$ **است که می‌خواهیم به‌دست آوریم.**

تناظر با روش حساب‌داری

- $\Phi(D_i)$: مقدار پول موجود در مخزن پس از عمل i.
- $\hat{c}_i$: مقدار پولی که برای عمل i پرداخت می‌کنیم.
- c_i: مقدار هزینه‌ی صرف‌شده برای عمل i.
- اگر $c_i < \hat{c}_i$، مابه‌التفاوت به مخزن اضافه می‌شود: $\Phi(D_i) = \Phi(D_{i-۱}) + \hat{c}_i - c_i$
- اگر $c_i > \hat{c}_i$، برای انجام عمل i لازم است از مخزن به‌اندازه‌ی $c_i - \hat{c}_i$ برداریم تا بتوانیم این عمل را انجام دهیم. پس پول مخزن $\Phi(D_i) = \Phi(D_{i-۱}) - (c_i - \hat{c}_i)$ می‌شود.

تحلیل پشته با روش پتانسیل

- تعریف: $\Phi(D_i) =$ تعداد عناصر موجود در پشته
- در ابتدا $\Phi(D_0) = 0$
- عمل Push:
  - یک عنصر اضافه می‌شود: $\Phi(D_i) - \Phi(D_{i-1}) = 1$
  - هزینه‌ی واقعی $c_i = 1$،
  - پس $\hat{c}_i = 1 + 1 = 2$.

- عمل Pop:
  - یک عنصر کم می‌شود: $\Phi(D_i) - \Phi(D_{i-۱}) = -۱$
  - هزینه‌ی واقعی ۱ $= c_i$،
  - پس $\hat{c}_i = ۱ - ۱ = ۰$.
- عمل Multipop هم تعدادی Pop است. پس هزینه‌ی سرشکن‌شده‌ی آن هم صفر است.

$$\sum_{i=۱}^{n} c_i \leq \sum_{i=۱}^{n} \hat{c}_i = ۲n$$

پس هزینه‌ی سرشکن‌شده‌ی هر عمل $\mathcal{O}(۱)$ است.

تحلیل شمارنده به‌روش پتانسیل

- تعریف تابع پتانسیل: تعداد یک ها $\Phi(D_i) = A$
- داریم $\Phi(D_\circ) = \circ$ و $\Phi(D_i) \geq \circ$
- عمل «افزایش»:
  - t_i بیت از ۱ به صفر و حداکثر یک عدد صفر به ۱ تبدیل می‌شود.
  - پس

$$\left.\begin{array}{l}\hat{c}_i = \Phi(D_i) - \Phi(D_{i-۱}) + c_i \\ \Phi(D_i) - \Phi(D_{i-۱}) \leq -t_i + ۱ \\ c_i \leq t_i + ۱\end{array}\right\} \rightarrow \hat{c}_i \leq ۲$$

- پس هزینه‌ی سرشکن‌شده‌ی هر عمل افزایش $\mathcal{O}(۱)$ است.