طراحی و تحلیل الگوریتم‌ها

مبحث چهارم

روش‌های طراحی الگوریتم

روش تقسیم و غلبه

Methods of Algorithm Design: Divide and Conquer

کاظم فولادی

دانشکده مهندسی برق و کامپیوتر

دانشگاه تهران

http://courses.fouladi.ir/algorithm

طراحی الگوریتم با روش تقسیم و غلبه

برای حل یک مسئله با روش تقسیم و غلبه:

- **تقسیم** نمونه‌ای از یک مسئله به یک یا چند نمونه‌ی کوچک‌تر
- **حل** نمونه‌های کوچکتر
  - *اگر نمونه‌های کوچک‌تر به اندازه‌ی کافی کوچک نبودند، از **بازگشت** استفاده می‌کنیم.*
  - *اگر نمونه‌های کوچک‌تر به اندازه‌ی کافی کوچک بودند، آنها را **مستقیماً** حل می‌کنیم.*
- **ترکیب** راه حل نمونه‌های کوچک‌تر برای یافتن راه حل نمونه‌ی اولیه

مسائلی با این روش قابل حل هستند که **قابل تقسیم** باشند
و بتوان راه‌حل مسئله را از **حل زیرمسئله‌های آن** پیدا کرد.

قضیه‌ی اصلی

$$f(n) = af\left(\frac{n}{b}\right) + c\,n^p$$

$$f(n) \in \begin{cases} \Theta(n^{\log_b a}) & ,a > b^p \\ \Theta(n^p \log_b n) & ,a = b^p \\ \Theta(n^p) & ,a < b^p \end{cases}$$

قضیه‌ی اصلی برای حد بالا

$$f(n) \leq af\left(\frac{n}{b}\right) + c\,n^p$$

$$f(n) \in \begin{cases} O(n^{\log_b a}) & , a > b^p \\ O(n^p \log_b n) & , a = b^p \\ O(n^p) & , a < b^p \end{cases}$$

قضیه‌ی اصلی برای حد پایین

$$f(n) \geq af\left(\frac{n}{b}\right) + c\,n^p$$

$$f(n) \in \begin{cases} \Omega(n^{\log_b a}) & , a > b^p \\ \Omega(n^p \log_b n) & , a = b^p \\ \Omega(n^p) & , a < b^p \end{cases}$$

مثال

1) $T(n) = 2T(\frac{n}{3}) + n, \quad T(1) = 0$

2) $T(n) = 10T(\frac{n}{5}) + n^2, \quad T(1) = 0$

3) $T(n) = 2T(\frac{n}{5}) + 6n^3, \quad T(1) = 6$

4) $T(n) = 40T(\frac{n}{3}) + 2n^3, \quad T(1) = 5$

5) $T(n) = 16T(\frac{n}{2}) + 7n^4, \quad T(1) = 1$

6) $T(n) = 4T(\frac{n}{4}) + 2n^2, \quad T(16) = 50$

7) $T(n) = 2T(\frac{n}{3}) + \log_3 n, \quad T(1) = 0$

8) $T(n) = 3T(\frac{n}{2}) + n, \quad T(1) = 0.5$

9) $T(n) = T(\frac{n}{3}) + 1, \quad T(1) = 0$

10) $T(n) = 2T(\frac{n}{2}) + 1, \quad T(1) = 0$

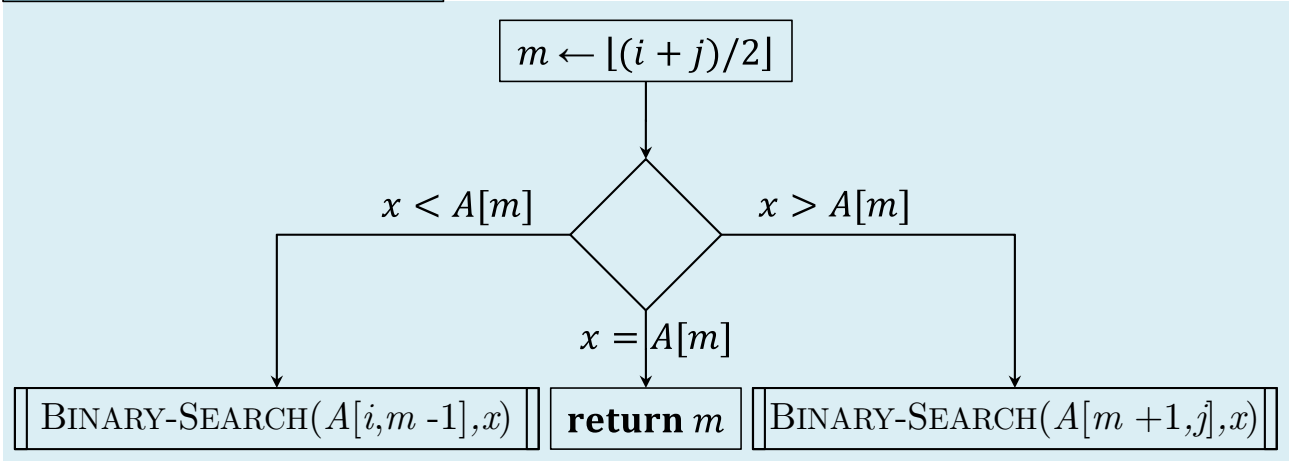
جستجوی دودویی

BINARY SEARCH

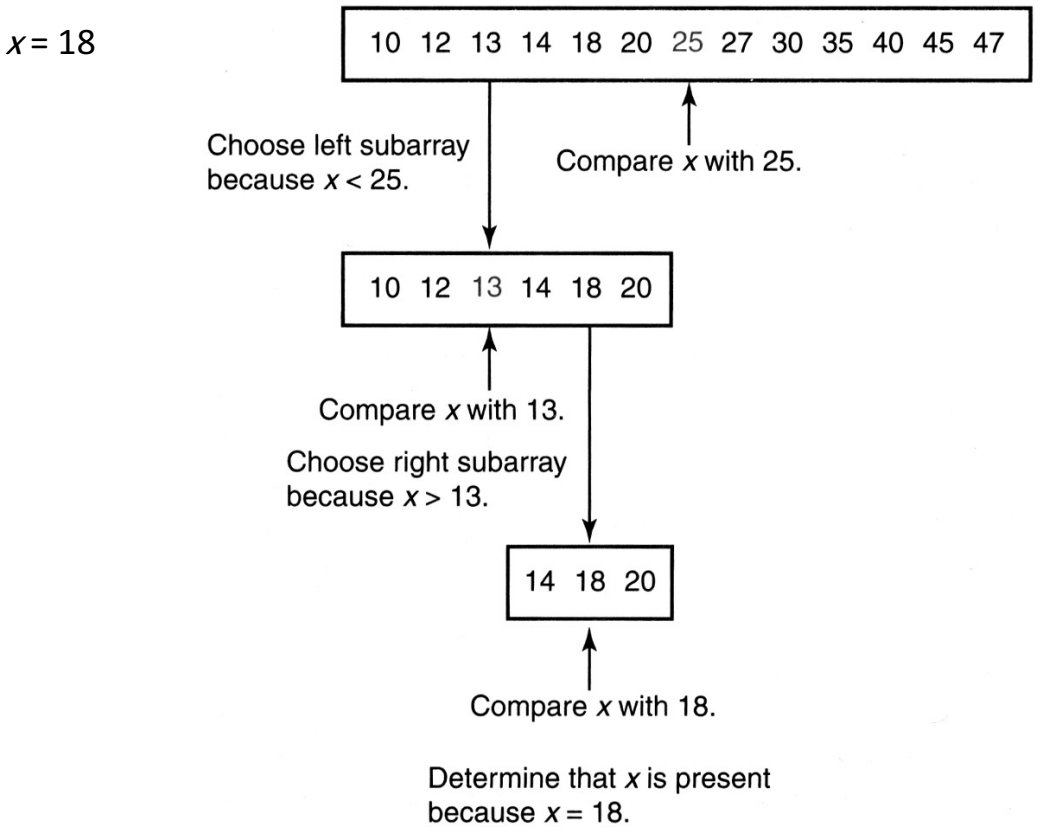
ورودی: آرایه‌ی مرتب‌شده‌ی A (صعودی) و کلید x
خروجی: اندیس کلید در آرایه، در صورت عدم وجود: صفر

راه حل:

BINARY-SEARCH$(A[i,j],x)$ $\quad i \le j$



جستجوی دودویی: مثال



جستجوی دودویی

ورودی: آرایه‌ی مرتب‌شده‌ی A (صعودی) و کلید x
خروجی: اندیس کلید در آرایه، در صورت عدم وجود: صفر

```
BINARY-SEARCH(A, i, j, x)
    if i ≤ j then
        m ← ⌊(i+j)/2⌋
        if x = A[m] then
            return m
        else if x < A[m]
            return BINARY-SEARCH(A, i, m -1, x)
        else ▷ x > A[m]
            return BINARY-SEARCH(A, m +1, j, x)
    else
        return 0
```

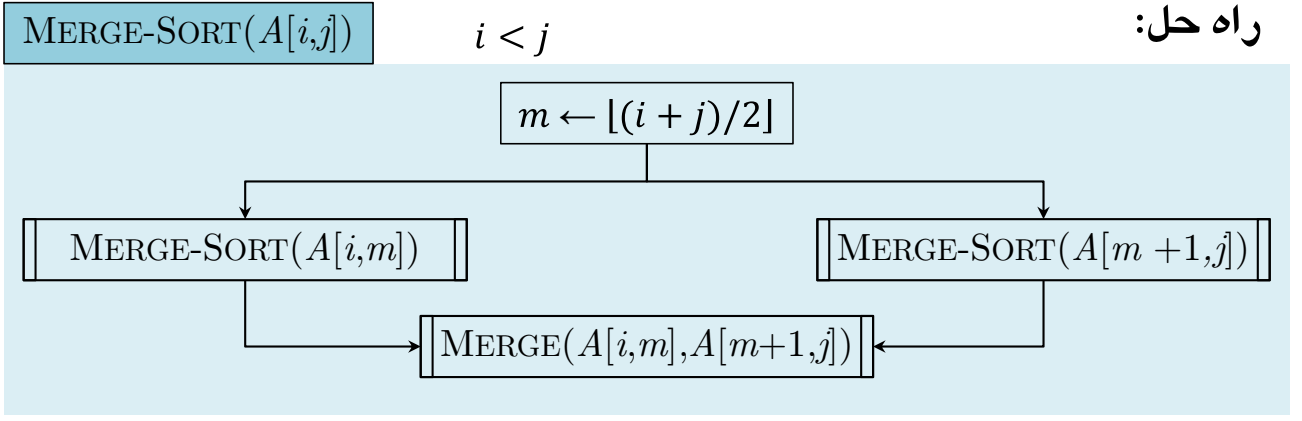
زمان اجرا: بر حسب تعداد اجرای عمل اصلی (مقایسه): $\Theta(\log_2 n)$

مرتب‌سازی ادغامی

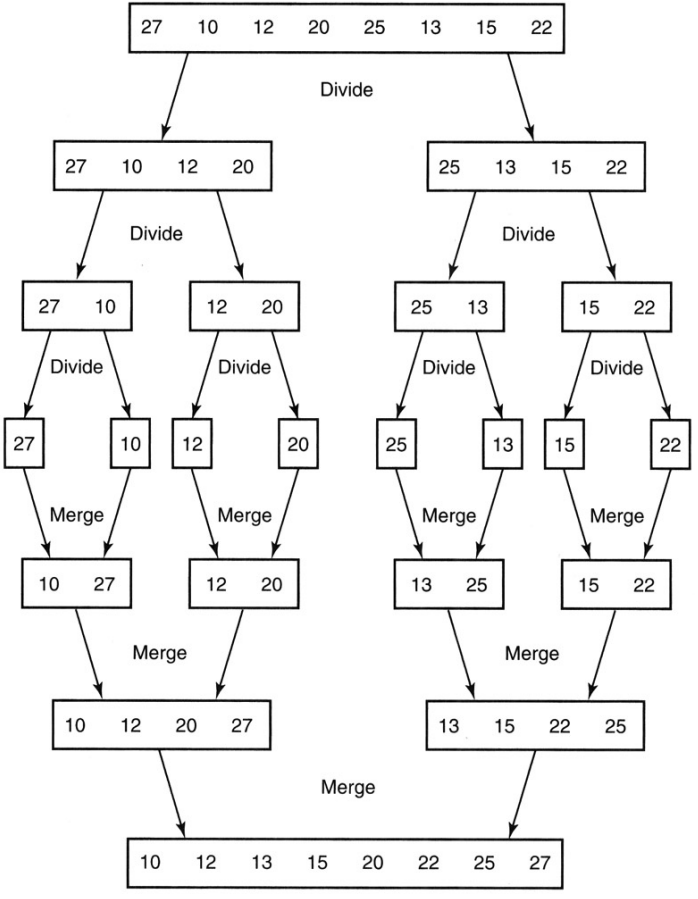
MERGE SORT

هدف: مرتب‌سازی آرایه‌ی $A[1..n]$ به صورت صعودی

راه حل:



مرتب‌سازی ادغامی: مثال



مرتب‌سازی ادغامی

MERGE SORT

هدف: مرتب‌سازی آرایه‌ی $A[1..n]$ به صورت صعودی

```
MERGE-SORT(A, i, j)
    if i < j then
        m ← ⌊(i+j)/2⌋
        MERGE-SORT(A, i, m)
        MERGE-SORT(A, m+1, j)
        MERGE(A, i, m, j)
```

زمان اجرا: بر حسب تعداد اجرای عمل اصلی (مقایسه): $\Theta(n\log_2 n)$

الگوریتم ادغام دو آرایه‌ی مرتب

MERGE ALGORITHM

هدف: ایجاد یک آرایه‌ی مرتب از اجتماع دو آرایه‌ی مرتب کوچکتر X و Y

$$\text{MERGE}(X[1..n], Y[1..m]) =$$
$$\begin{cases} [\text{MERGE}(X[1..n-1], Y[1..m-1]), X[n], Y[m]] & , X[n] = Y[m] \\ [\text{MERGE}(X[1..n-1], Y[1..m]), X[n]] & , X[n] > Y[m] \\ [\text{MERGE}(X[1..n], Y[1..m-1]), Y[m]] & , X[n] < Y[m] \end{cases}$$
$$\text{MERGE}(X, [\,]) = X$$

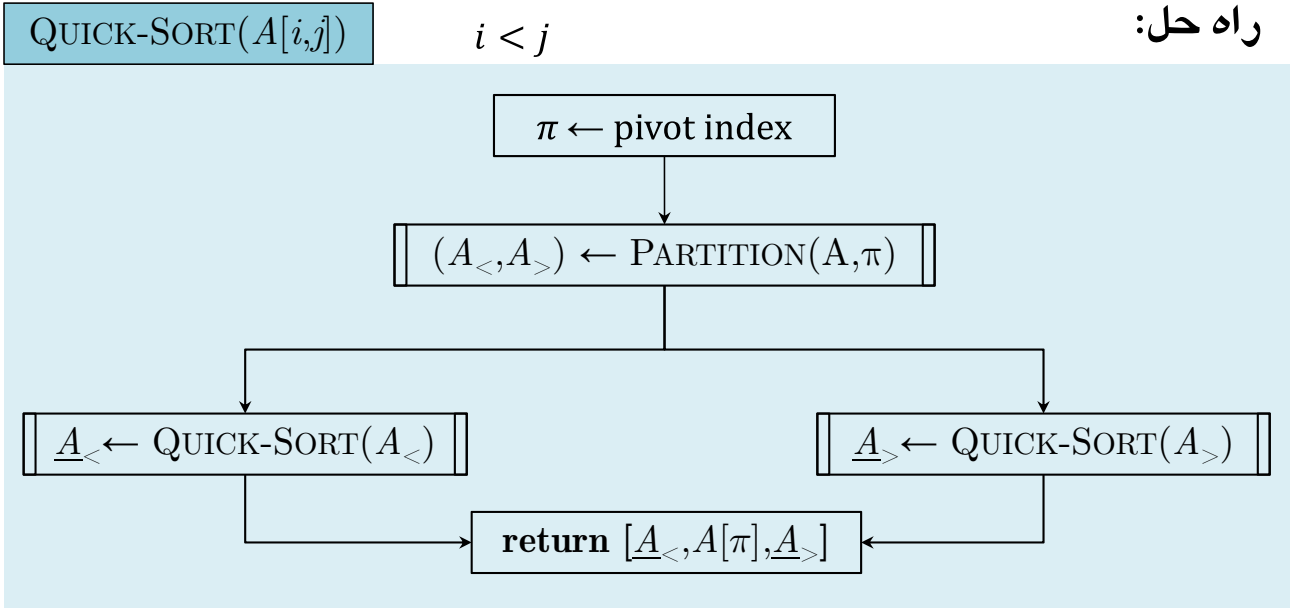
زمان اجرا: بر حسب تعداد اجرای عمل اصلی (مقایسه): حداکثر $n + m - 1$
زمان اجرا: بر حسب تعداد اجرای عمل اصلی (جابجایی): $n + m$

مرتب‌سازی سریع

QUICK SORT

هدف: مرتب‌سازی آرایه‌ی $A[1..n]$ به صورت صعودی

راه حل:



الگوریتم افراز یک آرایه‌ی بر اساس عنصر محوری

PARTITION ALGORITHM BASED ON PIVOT ITEM

هدف: افراز آرایه به دو زیرآرایه از عناصر کوچکتر و بزرگتر از عنصر محوری

$$\text{QUICK-SORT}(X[1..n]) = (\text{QUICK-SORT}(X_<), X[\pi], \text{QUICK-SORT}(X_>))$$
$$\text{QUICK-SORT}([\,]) = [\,]$$

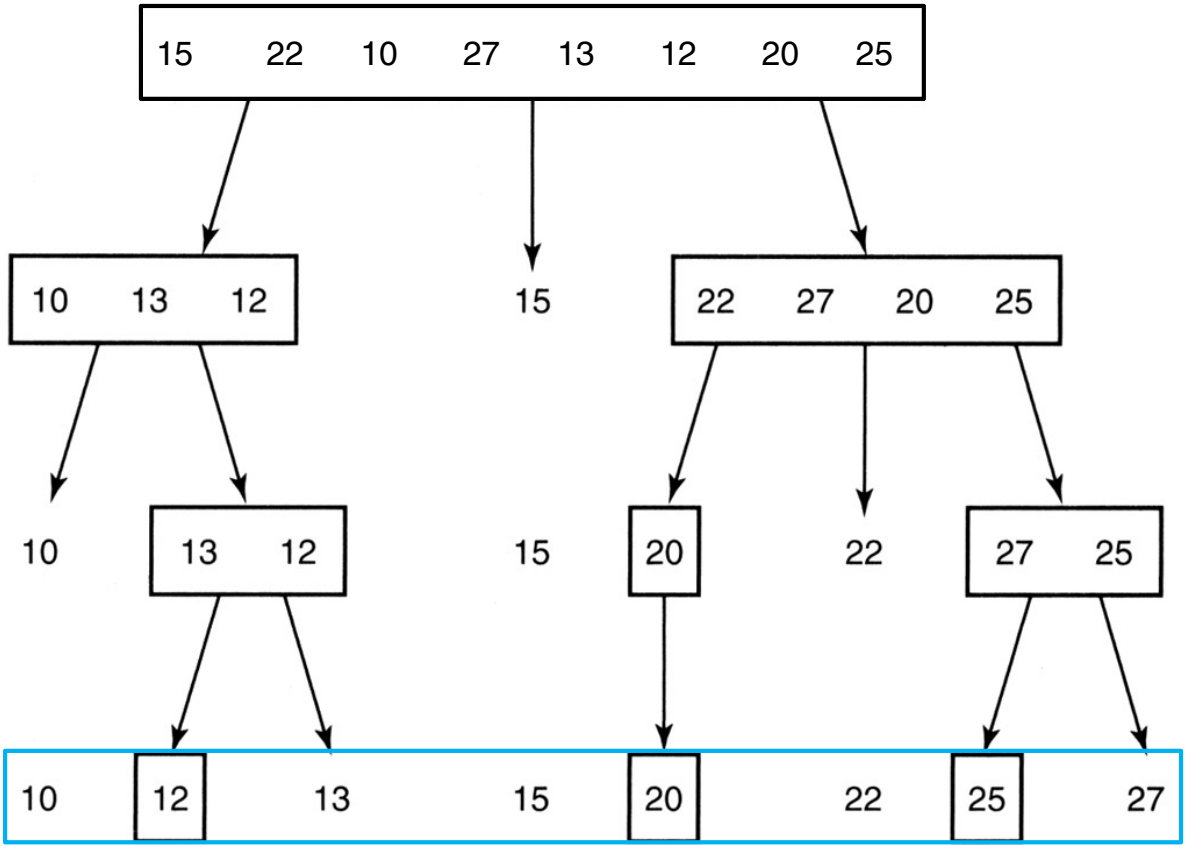
افراز:

$X_<$: آرایه‌ی X پس از حذف عناصر بزرگ‌تر از یا مساوی با $X[\pi]$

$X_>$: آرایه‌ی X پس از حذف عناصر کوچک‌تر از یا مساوی با $X[\pi]$

زمان اجرا: بر حسب تعداد اجرای عمل اصلی (مقایسه): $n-1$
(بجز عنصر محوری، همه‌ی عنصرها باید با عنصر محوری مقایسه شوند)

مرتب‌سازی سریع: مثال



مرتب‌سازی سریع

QUICK SORT

هدف: مرتب‌سازی آرایه‌ی $A[1..n]$ به صورت صعودی

```
QUICK-SORT(A, i, j)
    π ← 1
    if i < j then
        π ← PARTITION(A, i, π, j)
        QUICK-SORT(A, i, π-1)
        QUICK-SORT(A, π+1, j)
```

زمان اجرا بدترین حالت: بر حسب عمل اصلی (مقایسه): $W(n) = O(n^2)$

زمان اجرا حالت متوسط: عمل اصلی (مقایسه): $A(n) = \Theta(n\log_2 n)$

انتخاب نام مرتب‌سازی سریع، به دلیل عملکرد الگوریتم در حالت متوسط است!

روش استراسن برای ضرب دو ماتریس

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \qquad B = \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]$$

$$A \times B = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

$$\begin{aligned} M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ M_2 &= (A_{21} + A_{22})B_{11} \\ M_3 &= A_{11}(B_{12} - B_{22}) \\ M_4 &= A_{22}(B_{21} - B_{11}) \\ M_5 &= (A_{11} + A_{12})B_{22} \\ M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

زمان اجرا (الگوریتم معمولی):
بر حسب عمل اصلی (ضرب): $\theta(n^3)$

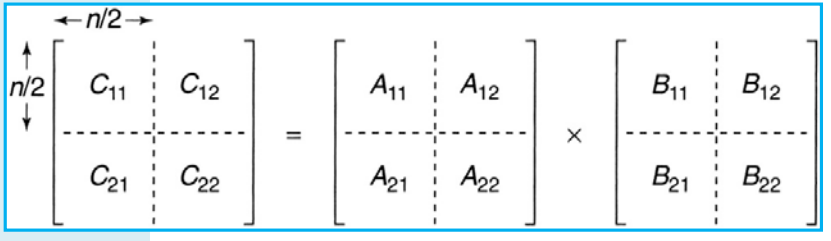
روش استراسن برای ضرب دو ماتریس

```
STRASSEN(A, B, n)
    if n > 1 then
        A11, A12, A21, A22 ← divide A
        B11, B12, B21, B22 ← divide B
        M1 ← STRASSEN(A11 + A22, B11 + B22, n/2)
        M2 ← STRASSEN(A21 + A22, B11, n/2)
        M3 ← STRASSEN(A11, B12 − B22, n/2)
        M4 ← STRASSEN(A22, B21 − B11, n/2)
        M5 ← STRASSEN(A11 + A12, B22, n/2)
        M6 ← STRASSEN(A21 − A11, B11 + B12, n/2)
        M7 ← STRASSEN(A12 − A22, B21 + B22, n/2)
        C11 ← M1 + M4 − M5 + M7
        C12 ← M3 + M5
        C21 ← M2 + M4
        C22 ← M1 + M3 − M2 + M6
        C ← [C11 C12; C21 C22]
        return C
    else return A * B
```

زمان اجرا (الگوریتم استراسن):
بر حسب عمل اصلی (ضرب): $\theta(n^{\log_2 7})$

$$\begin{cases} T(n) = 7T\left(\frac{n}{2}\right) \\ T(1) = 1 \end{cases}$$

$$\Rightarrow T(n) = n^{\log_2 7} = n^{2.81}$$



ضرب اعداد صحیح بزرگ

مثال

هر رقم در یک خانه‌ی آرایه:

$\frac{5}{S[6]}$	$\frac{4}{S[5]}$	$\frac{3}{S[4]}$	$\frac{1}{S[3]}$	$\frac{2}{S[2]}$	$\frac{7}{S[1]}$

$$\underbrace{567,832}_{6\ \text{digits}} = \underbrace{567}_{3\ \text{digits}} \times 10^3 + \underbrace{832}_{3\ \text{digits}}$$

$$\underbrace{9,423,723}_{7\ \text{digits}} = \underbrace{9423}_{4\ \text{digits}} \times 10^3 + \underbrace{723}_{3\ \text{digits}}$$

$$\begin{aligned} 567,832 \times 9,423,723 &= (567 \times 10^3 + 832) \times (9423 \times 10^3 + 723) \\ &= (567 \times 9423) \times 10^6 \\ &\quad + (567 \times 723 + 9423 \times 832) \times 10^3 \\ &\quad + (832 \times 723) \end{aligned}$$

ضرب اعداد صحیح بزرگ

حالت کلی: الگوریتم استاندارد

$$u = \boxed{x \mid y}$$

$\underbrace{\quad}_{\lfloor \frac{n}{2} \rfloor \text{ digits}}$ $\underbrace{\quad}_{\lceil \frac{n}{2} \rceil \text{ digits}}$

$$v = \boxed{w \mid z}$$

$\underbrace{\quad}_{\lfloor \frac{n}{2} \rfloor \text{ digits}}$ $\underbrace{\quad}_{\lceil \frac{n}{2} \rceil \text{ digits}}$

$$u = x \times 10^m + y$$
$$v = w \times 10^m + z$$

$$u \times v = (x \times 10^m + y)(w \times 10^m + z)$$
$$= wx \times 10^{2m} + (xz + yw) \times 10^m + yz$$

$W(n) = 4W\left(\frac{n}{2}\right) + cn$
$W(s) = 0$

زمان اجرا (الگوریتم معمولی):
بر حسب عمل اصلی (ضرب): $\theta(n^2)$

$$m = \left\lceil \frac{n}{2} \right\rceil$$

ضرب اعداد صحیح بزرگ

حالت کلی: الگوریتم بهبودیافته

$$u = \boxed{\underbrace{x}_{\lfloor \frac{n}{2} \rfloor \text{ digits}} \mid \underbrace{y}_{\lceil \frac{n}{2} \rceil \text{ digits}}}$$

$$v = \boxed{\underbrace{w}_{\lfloor \frac{n}{2} \rfloor \text{ digits}} \mid \underbrace{z}_{\lceil \frac{n}{2} \rceil \text{ digits}}}$$

مقادیر زیر باید تعیین شوند:

$$xw, \qquad xz + yw, \qquad yz$$

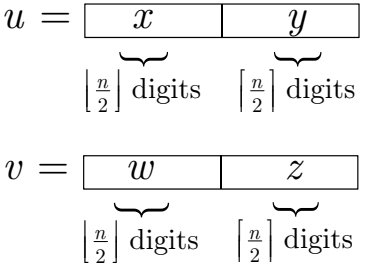
$$\begin{aligned} r &= (x + y)(w + z) \\ &= xw + (xz + yw) + yz \end{aligned}$$

$$\Rightarrow$$

با این محاسبه تعداد ضرب‌ها از چهار به سه کاهش می‌یابد:

$$xz + yw = r - xw - yz$$

ضرب اعداد صحیح بزرگ



زمان اجرا (الگوریتم بهبود یافته):
بر حسب عمل اصلی (ضرب): $\theta\left(n^{\log_2 3}\right)$

$$\begin{cases} T(n) = 3T\left(\frac{n}{2}\right) \\ T(1) = 1 \end{cases} \Rightarrow T(n) = n^{\log_2 3} = n^{1.58}$$

LARGE-NUMBERS-MULTIPLY$(u,\ v,\ n)$
  if $n > 1$ **then**
    $(x,y) \leftarrow$ divide$(u, \lfloor n/2 \rfloor)$
    $(w,z) \leftarrow$ divide$(v, \lfloor n/2 \rfloor)$
    $xw \leftarrow$ LARGE-NUMBERS-MULTIPLY$(x,\ w, \lfloor n/2 \rfloor)$
    $yz \leftarrow$ LARGE-NUMBERS-MULTIPLY$(y,\ z, \lfloor n/2 \rfloor)$
    $r \leftarrow$ LARGE-NUMBERS-MULTIPLY$(x+y,\ w+z, \lfloor n/2 \rfloor)$
    return $wx \times 10^{2m} + (r - xw - yz) \times 10^{m} + yz$
  else
    return $u \times v$

ضرب در توان‌های ۱۰ با شیفت به چپ آرایه انجام می‌شود.
جمع و تفریق دو عدد n رقمی هم در زمان خطی n انجام می‌شود.

تورنمت بازی‌ها

در یک جام حذفی، n تیم شرکت می‌کنند. در دور نخست، $n/2$ بازی برگزار می‌شود و $n/2$ برنده در دور بعدی بازی می‌کنند (مساوی نداریم) و همین طور تا آخر:

$$\boxed{n = 2^k}$$

- تعداد بازی‌ها در کل چند تاست؟
- تعداد دورهای بازی چند تاست؟

$$\begin{cases} f(n) = f(\frac{n}{2}) + \frac{n}{2} \\ f(2) = 1 \end{cases} \Rightarrow f(n) = n - 1$$

تعداد بازی‌ها

$$\begin{cases} g(n) = g(\frac{n}{2}) + 1 \\ g(2) = 1 \end{cases} \Rightarrow g(n) = \log_2 n$$

تعداد دورها

کجا نباید از روش تقسیم و غلبه استفاده کرد؟

وقتی زیرمسئله‌ها با هم همپوشانی پیدا می‌کنند:

- وقتی نمونه‌ای به اندازه‌ی n به دو یا چند نمونه تقسیم می‌شود که اندازه‌ی آنها نیز تقریباً n است.
  - در این صورت زمان اجرا نمایی می‌شود.

- وقتی نمونه‌ای به اندازه‌ی n به تقریباً n نمونه با اندازه‌ی n/c تقسیم شود که $c>1$ باشد.
  - در این صورت زمان اجرا بدتر از نمایی (فاکتوریل) می‌شود.

❋ اگر مسئله‌ای **ذاتاً** نیاز به زمان‌های نمایی و بدتر از نمایی داشته باشد، استفاده از روش تقسیم و غلبه منطقی است.

تعیین مقادیر آستانه

THRESHOLDS

تعیین مقداری از اندازه‌ی ورودی که به ازای مقادیر کمتر از آن فراخوانی بازگشتی کارایی کمتری دارد:

در اینجا بهتر است از الگوریتم دیگری استفاده شود.

* مقدار آستانه‌ی بهینه ممکن است منحصر به فرد نباشد!