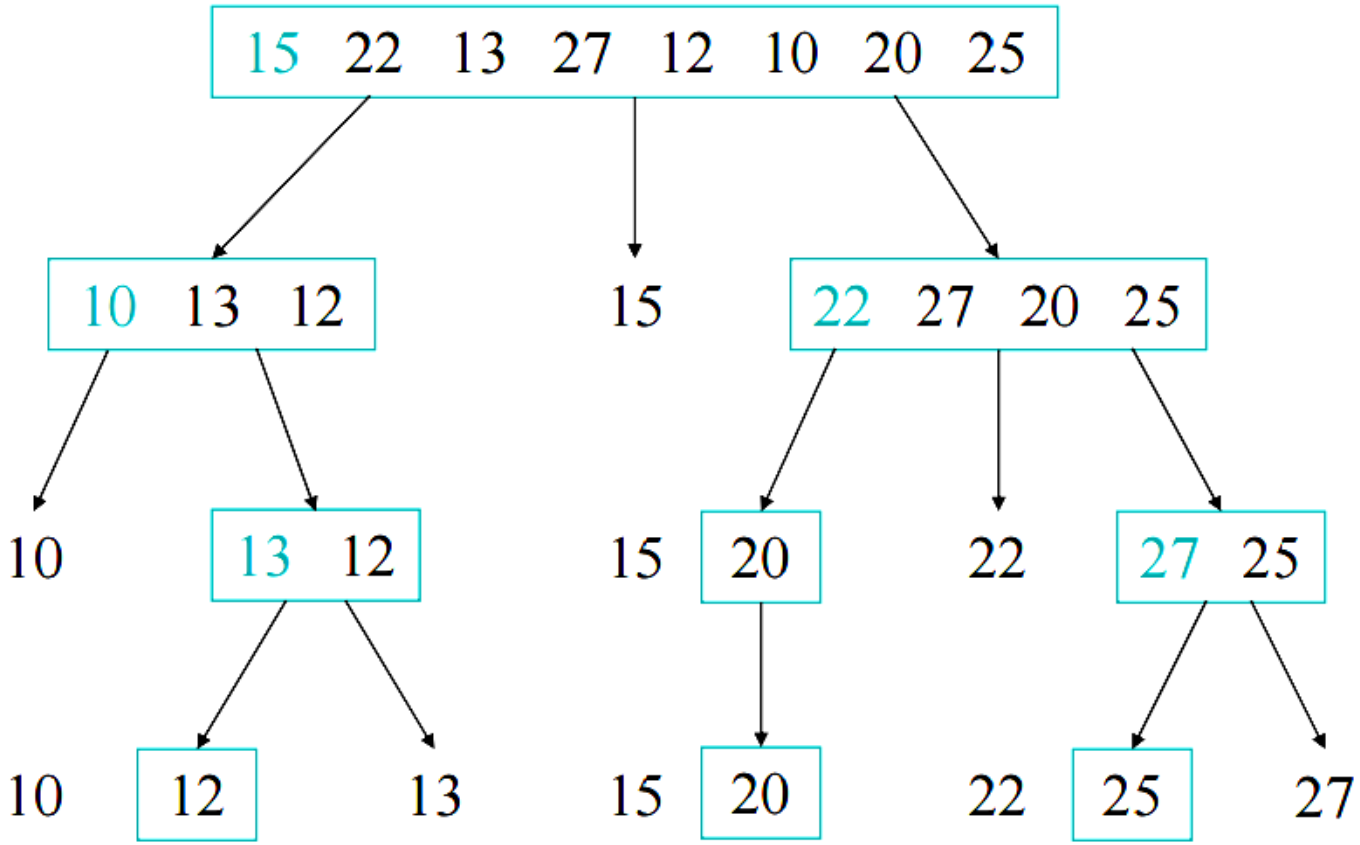
مرتب سازی سریع

- مراحل:
  - انتخاب **عنصر محوری** (معمولا عنصر اول)
  - تقسیم آرایه به دو بخش به طوری که عناصر کوچکتر از عنصر محوری در سمت چپ و عناصر بزرگتر از آن در سمت راست آن قرار بگیرند.
  - مرتب سازی هر بخش به صورت بازگشتی

روش تقسیم و حل



روش تقسیم و حل

Quick Sort

```
procedure Partition (Var x: Arraylist; left, right : integer; Var pivotpoint : integer
Var i,j,pivot : integer;
begin
        i := left ;     j := right+1;     pivot := x[left];
        repeat
                repeat
                        i := i + 1;
                until   x[i] >= pivot;
                repeat
                        j := j - 1;
                until  x[j] <= pivot;
                if i<j   then   swap (x[i], x[j]);
        until   i>= j;
       swap (x[left], x[j]);
       pivotpoint := j;
end;
{ * * * * * * * * * * * * * * * * * * * * }
procedure Quicksort (Var  x: Arraylist; left, right : integer) ;
var    pivotpoint : integer;
begin
        if (left  < right)    then   begin
                partition (x , left, right, pivotpoint); {سر لولا در محل واقعی خود قرار می‌گیرد
                QuickSort(x, left, pivotpoint - 1); {مرتب‌سازی زیر لیست چپ}
                QuickSort (x, pivotpoint+1, right); { مرتب‌سازی زیر لیست راست }
       end;    {if}
end;
```

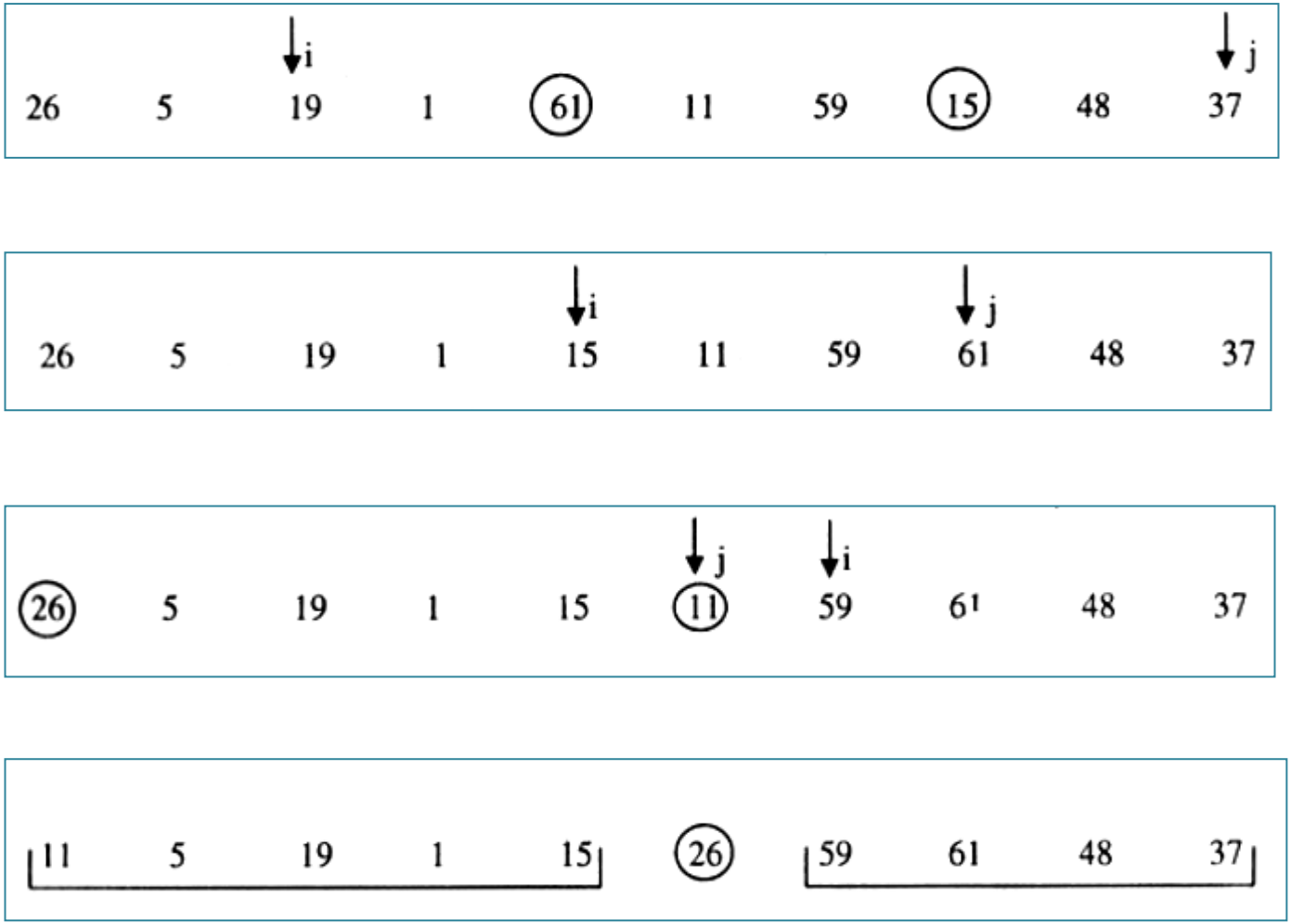
روش تقسیم و حل

پردازه Partition آرایه را به گونه‌ای افراز می‌کند که عنصر محوری (اولین عنصر سمت چپ آرایه) در مکان درست خود قرار گیرد و همچنین مکان قرارگیری آن در آرایه توسط متغیر خروجی pivotpoint برگردانده می‌شود.

فرض کنید آرایه اولیه به صورت زیر باشد انجام یک مرحله از الگوریتم فوق به صورت زیر است:

x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	x[9]	x[10]
26	5	37	1	61	11	59	15	48	19

روش تقسیم و حل



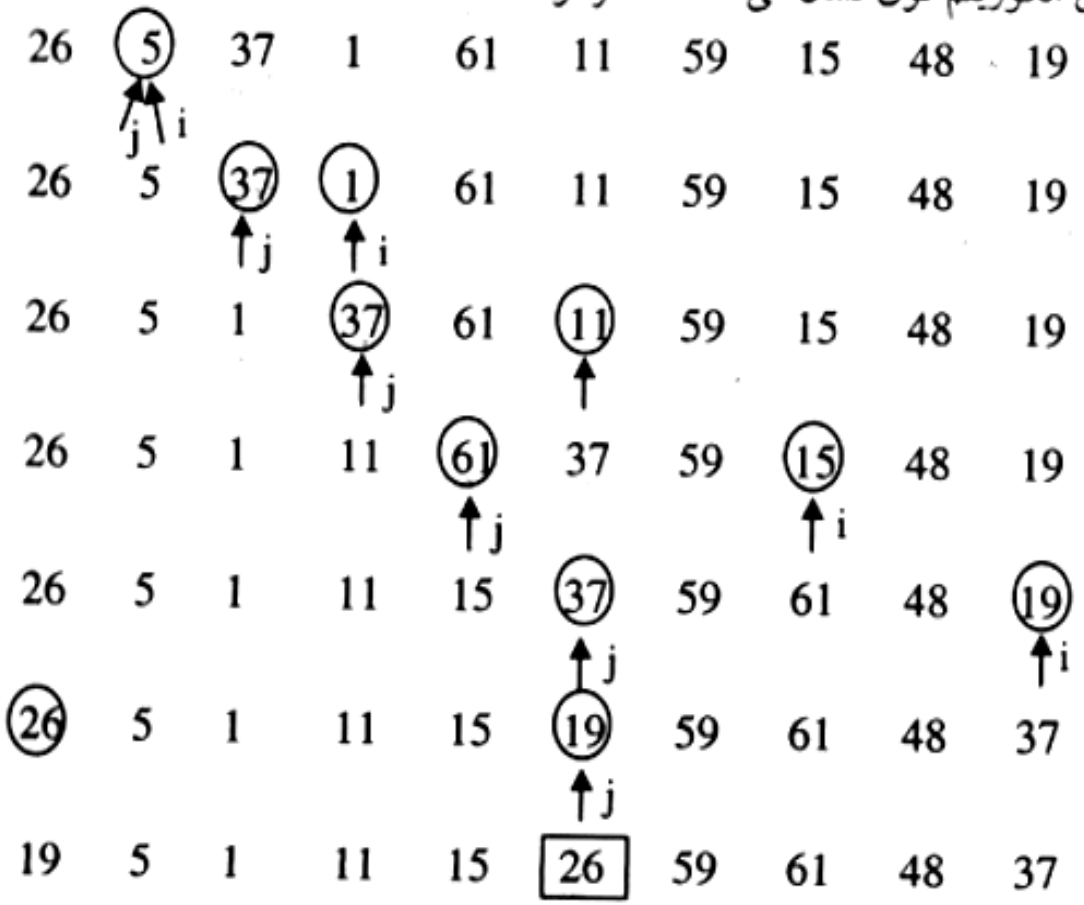
روش تقسیم و حل

[26	5	37	1	61	11	59	15	48	19
[11	5	19	1	15]	(26)	[59	61	48	37
[1	5]	(11)	[19	15]	26	[59	61	48	37
(1)	(5)	11	[19	15]	26	[59	61	48	37]
1	5	11	(15)	(19)	26	[59	61	48	37]
1	5	11	15	19	26	[48	37]	(59)	[61
1	5	11	15	19	26	37	48	59	[61
1	5	11	15	19	26	37	48	59	61

```
procedure partition (Var  x: Arraylist; left, right : integer; Var pivotpoint : integer)
Var  i,j, pivot : integer;
begin
   j := left;       pivot := x[left];
   for  i := left +1    to   right   do
      if  x[i] < pivot  then  begin
            j := j + 1 ;
            swap (s[i], s[j]);
        end;      -
  swap (x[left], x[j]);
  pivotpoint := j;
end;
```

روش تقسیم و حل

نمونه زیر مراحل کار افراز را مطابق الگوریتم فوق نشان می‌دهد. عنصر لولا عدد 26 است.



روش تقسیم و حل

26	5	37	1	61	11	59	15	48	19
19	5	37	1	61	11	59	15	48	26
19	5	26	1	61	11	59	15	48	37
19	5	15	1	61	11	59	26	48	37
19	5	15	1	26	11	59	61	48	37
19	5	15	1	11	26	59	61	48	37

پیچیدگی زمانی زمان بخشی (همه حالات)

- عمل اصلی: مقایسه $S[i]$ با $pivotitem$
- اندازه ورودی: $n = high - low + 1$ (اندازه زیر آرایه)
- پیچیدگی زمانی: از آنجا که هر یک از عناصر (به جز اولی) یک بار مقایسه می شوند:

$$T(n) = n - 1$$

پیچیدگی زمانی (بدترین حالات)

- عمل اصلی: مقایسه $S[i]$ با $pivotitem$ در رویه $partition$
- اندازه ورودی: n، اندازه آرایه S
- پیچیدگی زمانی:

$$T(n) = \underbrace{T(0)}_{\substack{Time\ to\ sort \\ left\ subarray}} + \underbrace{T(n-1)}_{\substack{Time\ to\ sort \\ right\ subarray}} + \underbrace{n-1}_{\substack{Time\ to \\ partition}}$$

$$\Rightarrow \begin{cases} T(n) = T(n-1) + n - 1 & for \quad n > 0 \\ T(0) = 0 \end{cases}$$

- حل:

$$T(n) = n(n-1)/2 \in \Theta(n^2)$$

پیچیدگی زمانی (حالت متوسط)

- عمل اصلی: مقایسه $S[i]$ با $pivotitem$ در رویه $partition$
- اندازه ورودی: n، اندازه آرایه S
- پیچیدگی زمانی:

$$A(n) = \sum_{p=1}^{n} \overbrace{\frac{1}{n}}^{\Pr obability\ pivotpoint\ is\ p} \underbrace{[A(p-1) + A(n-p)]}_{\substack{Average\ sort\ pivotpoint \\ time\ subarrays\ is \\ to\ when\ p}} + \underbrace{n-1}_{\substack{Time\ to \\ partition}}$$

- حل:

$$A(n) \approx 1.38(n+1)\lg n \in \Theta(n\lg n)$$

	بهترین حالت	حالت متوسط	بدترین حالت
مرتب‌سازی سریع	O (n logn)	O (n logn)	$O(n^2)$
مرتب‌سازی ادغام	O (n logn)	O (n logn)	O (n logn)

الگوریتم ضرب ماتریس ها به روش استراسن

- ضرب ماتریس ها طبق تعریف:
  - تعداد ضرب ها: $T(n) = n^3$
  - تعداد جمع ها: $T(n) = n^3 - n^2$
- الگوریتم استراسن برای ضرب ماتریس ها (1969)
  - پیچیدگی بهتر از درجه سوم (جمع و ضرب)

روش تقسیم و حل

- برای محاسبه:

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- تعریف می کنیم:

$$m_1 = (a_{11} + a_{22})(b_{11} + b_{22})$$
$$m_2 = (a_{21} + a_{22})b_{11}$$
$$m_3 = a_{11}(b_{12} - b_{22})$$
$$m_4 = a_{22}(b_{21} - b_{11})$$
$$m_5 = (a_{11} + a_{12})b_{22}$$
$$m_6 = (a_{21} - a_{11})(b_{11} + b_{22})$$
$$m_7 = (a_{12} - a_{22})(b_{21} + b_{22})$$

- آنگاه :

$$C = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

روش تقسیم و حل

- تقسیم ماتریس ها:

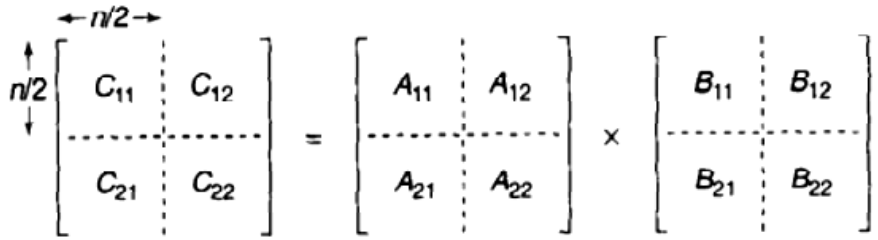


Figure 2.4 • The partitioning into submatrices in Strassen's algorithm.

- محاسبه M ها، مثلا:

$$M_1 = (A_{11} + A_{22})\,(B_{11} + B_{22})$$

- وقتی $n = 4$

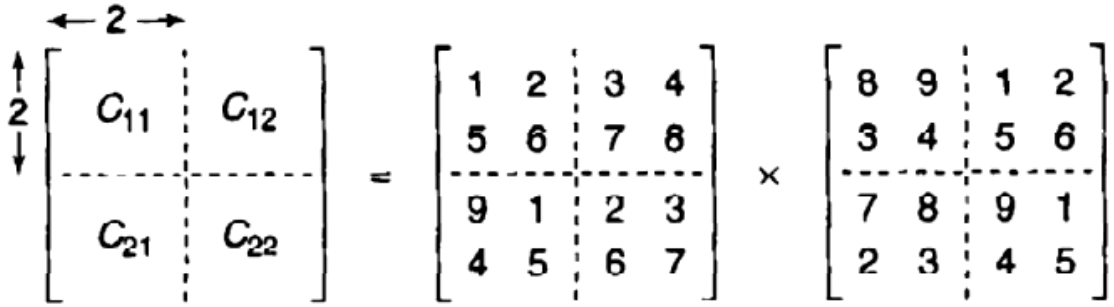


Figure 2.5 ● The partitioning in Strassen's algorithm with $n = 4$ and values given to the matrices.

- محاسبه M ها، مثلا:

$$M_1 = \begin{bmatrix} 3 & 5 \\ 11 & 13 \end{bmatrix} \times \begin{bmatrix} 17 & 10 \\ 7 & 9 \end{bmatrix} = \begin{bmatrix} 86 & 75 \\ 278 & 227 \end{bmatrix}$$

پیچیدگی زمانی (همه حالات - ضرب)

- عمل اصلی: یک ضرب ساده
- اندازه ورودی: n، تعداد سطرها و ستون های ماتریس ها
- پیچیدگی زمانی:

$$\begin{cases} T(n) = 7T(\frac{n}{2}) & for \quad n>1, \quad n \quad a \quad power \quad of \quad 2 \\ T(1) = 1 \end{cases}$$

- حل:

$$T(n) = n^{\lg 7} \approx n^{2.81} \in \Theta(n^{2.81})$$

پیچیدگی زمانی (همه حالات - جمع و تفریق)

- عمل اصلی: یک جمع یا تفریق ساده
- اندازه ورودی: n، تعداد سطرها و ستون های ماتریس ها
- پیچیدگی زمانی:

$$\begin{cases} T(n) = 7T(\frac{n}{2}) + 18(\frac{n}{2})^2 & for \quad n>1, \quad n \quad a \quad power \quad of \quad 2 \\ T(1) = 0 \end{cases}$$

- حل:

$$T(n) = 6n^{\lg 7} - 6n^2 \approx 6n^{2.81} - 6n^2 \in \Theta(n^{2.81})$$

عمل	روش استاندارد	روش استراسن
ضرب	n^3	$n^{2.81}$
جمع / تفریق	$n^3 - n^2$	$6n^{2.81} - 6n^2$

محاسبه با اعداد صحیح بزرگ

- نمایش اعداد صحیح بزرگ: جمع و عملیات خطی دیگر
  - استفاده از آرایه ای از اعداد صحیح، در هر خانه یک رقم
  - ذخیره علامت در بالاترین محل آرایه
  - الگوریتم های زمان خطی:
    - جمع
    - تفریق
    - $u \times 10^m$
    - u divide 10^m
    - u rem 10^m

ضرب اعداد صحیح بزرگ

- تقسیم یک عدد صحیح n-رقمی به دو عدد صحیح که هر کدام تقریبا دارای $n/2$ رقم می باشند. مثلا:

  - $567{,}832 = 567 \times 10^3 + 832$
  - $9{,}423{,}723 = 9423 \times 10^3 + 723$

- به طور کلی:

$$\underbrace{u}_{n \text{ digits}} = \underbrace{x}_{\lceil n/2 \rceil \text{ digits}} \times 10^m + \underbrace{y}_{\lfloor n/2 \rfloor \text{ digits}}$$

که $m = \lfloor n/2 \rfloor$

الگوریتم ضرب اعداد صحیح بزرگ

```
Large_int   prod (Large_int   u, Large_int   v)
  {
  Large_int   x,y,w,z;
   int  n,m ;
   n = Maximum (number of digits in u, number of digits in v)
   if (u == 0 || v == 0)    return  0;
   else  if(n <= threshold)
        return    u × v;
    else   {
         m = ⌊n / 2⌋ ;
         x = u   divide   10^m ;     y = u   rem   10^m ;
        w = v  divide   10^m ;  z = v  rem  10^m;
        return   prod (x,w) × 10^2m + (prod(x,z) + prod(w,y)) × 10^m + prod (y,z);
    }
}
```

روش تقسیم و حل

- برای ضرب نمودن دو عدد صحیح n-رقمی

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

- حاصل ضرب برابر است با:

$$uv = xw \times 10^{2m} + (xz + wy) \times 10^m + yz$$

- مثال:

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

روش تقسیم و حل

زمانی که نباید از تقسیم و حل استفاده کرد.

- یک نمونه به اندازه n به دو یا چند نمونه تقسیم شود به طوری که اندازه هر یک از این نمونه ها تقریبا برابر اندازه نمونه اصلی باشد. ← نمایی
  - مثال: دنباله فیبوناچی
  - استثناء : مساله برج های هانوی (تمرین ۱۷)

- یک نمونه به اندازه n تقریبا به n نمونه با اندازه های n/c تقسیم شود به طوری که c یک عدد ثابت باشد. ← $\Theta(n^{\lg n})$

فصل سوم

برنامه نویسی پویا

علت ناکارآمدی تقسیم و حل

- بعد از تقسیم ...
  - نمونه های کوچکتر غیر مرتبط هستند، مانند مرتب سازی ادغامی
  - نمونه های کوچکتر مرتبط هستند، مانند فیبونانچی
    - حل نمونه های مشترک به طور مکرر
- برنامه نویسی پویا
  - رهیافت پایین به بالا (bottom-up)
  - با استفاده از یک آرایه (جدول) برای ذخیره کردن راه حل نمونه های کوچکتر

مراحل

- ارایه یک خاصیت بازگشتی برای حل نمونه ای از مساله
- حل نمونه ای از مساله به روش پایین به بالا با حل نمونه های کوچکتر در ابتدا

مثال : رابطه بازگشتی $T(n) = T(n-1) + 3$ با حالت خاص $T(1) = 1$ را حل کنید.

روش اول (بالا به پائین) :

$$T(n) = T(n-1) + 3 = (T(n-2)+3) + 3 = T(n-2) + 2 \times 3$$
$$= T(n-3) + 3 \times 3 = T(n-4) + 4 \times 3 = \ldots$$
$$= T(n - (n-1)) + (n-1) \times 3 = T(1) + (n-1) \times 3 = 1 + (n-1) \times 3$$
$$= 3n - 2$$

روش دوم (پائین به بالا) :

$$T(1) = 1$$
$$T(2) = T(1) + 3 = 1 + 3 = 4$$
$$T(3) = T(2) + 3 = 4 + 3 = 7$$
$$T(4) = T(3) + 3 = 7 + 3 = 10$$
$$T(5) = T(4) + 3 = 10 + 3 = 13$$
$$\vdots$$
$$T(n) = 3n - 2$$

مثال : سری فیبوناچی

روش پویا

```
int   fib(int  n)
{
  int i, f(0 .. n];
   f[0] = 0;
   if (n > 0)  {
        f[1] = 1;
        for (i=2; i <= n ; i++)
              f[i] = f[i-1] + f[i-2];
    }
   return f[n];
}
```

روش تقسیم و غلبه

```
int   fib (int   n)
{
  if (n <=1)
        return   n;
   else
      return  fib(n-1) + fib(n-2);
}
```

مثلاً الگوریتم پویای فوق را می‌توان به صورت زیر نیز انجام داد :

```
int   fib (int n)
{
   int  f1, f2, f;
   f1 = 0;     f2 = 1;
        if(n == 0)  return  f1;
        else  if(n ==1) return  f2;
   for (i=2;  i<= n; i++)
   {
        f = f1 + f2;
        f1 = f2;
        f2 = f;
   }
   return f;
}
```

پس مراحل ایجاد یک الگوریتم پویا شامل مراحل زیر است :

۱ـ ایجاد یک خاصیت بازگشتی برای حل نمونه‌ای از مسأله

۲ـ حل نمونه‌ای از مسأله با روش جزء به کل از طریق حل نمونه‌های کوچکتر

مثال : ضریب دو جمله ای

- تعریف

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \qquad for \quad 0 \le k \le n$$

- تعریف بازگشتی

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 \quad k = 0 \quad or \quad k = n \end{cases}$$

پیاده سازی فرمول قبل با تقسیم و حل

```
int  bin(int  n, int k)
{
   if (k ==0  || n == k)   return 1 ;
   else  return  bin (n-1 , k-1) + bin (n-1, k);
}
```

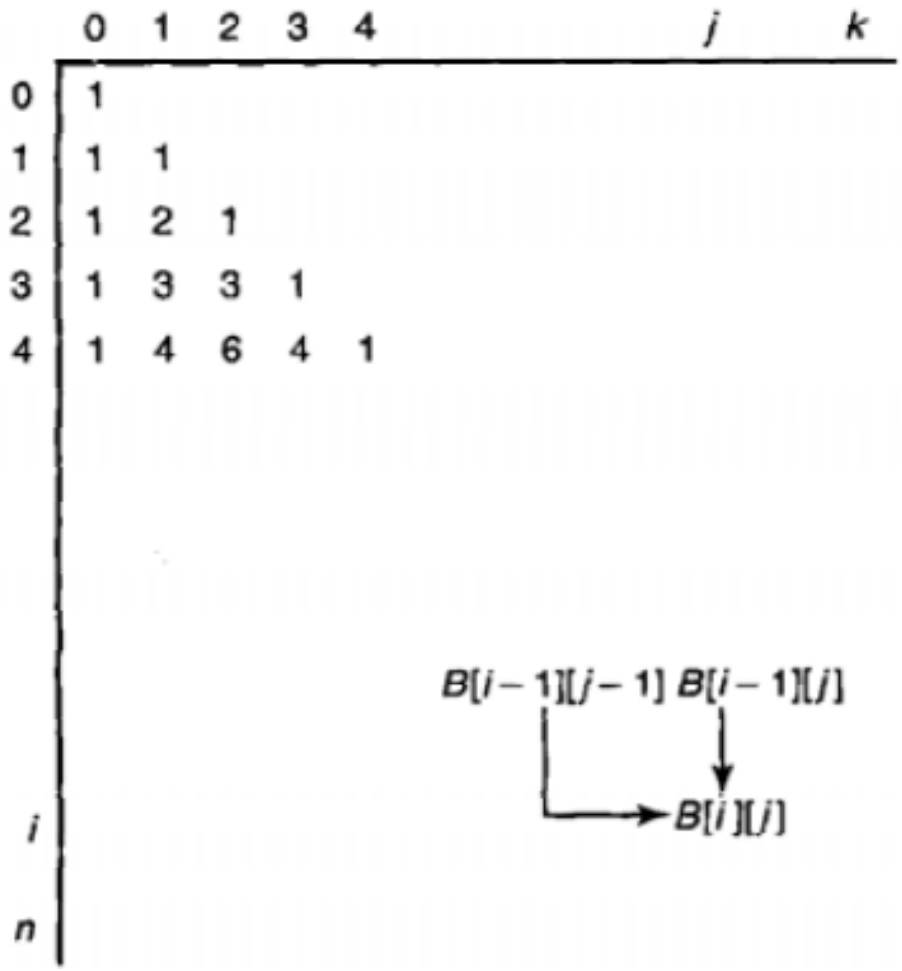
$$2\binom{n}{k}-1$$

استفاده از برنامه نویسی پویا

- استفاده از یک آرایه B به منظور ذخیره کردن ضرایب
- مراحل:
  - بنا نهادن یک خاصیت بازگشتی

$$B[i][j]=\begin{cases} B[i-1][j-1]+B[i-1][j] & 0<j<i \\ 1 \quad j=0 \quad or \quad j=i \end{cases}$$

  - حل یک نمونه مساله به روش پایین به بالا با محاسبه نمودن سطرهای B به طور متوالی با شروع از سطر اول


0 1 2 3 4 j k
0 1
1 1 1
2 1 2 1
3 1 3 3 1
4 1 4 6 4 1
i
n
B[i − 1][j − 1] B[i − 1][j]
B[i][j]

منظور از $B[i][j]$ عبارت $\binom{i}{j}$ و $B[n][k]=\binom{n}{k}$ است. برنامه‌ای که ماتریس فوق را ایجاد می‌کند به صورت زیر است:

```
int   bin2 (int  n ,  int  k)
{
   int  i,j ;   int  B[0 .. n] [0 .. k];
   for (i=0;  i <= n ; i ++)
     for (j=0; j <=minimum(i,k); j++)
     {
          if (j == 0 || j == i)
                  B[i][j] = 1 ;
          else  B[i][j] = B[i-1][j-1] + B[i-1][j];
     }
  return B[n][k];
}
```