درس طراحی الگوریتم ها

(با شبه کد های c ++)

منبع : کتاب طراحی الگوریتمها

مترجم : جعفر نژاد قمی

ارائه دهنده : دکتر پدیداران مقدم

۴-۱مرتبه الگوریتم

- الگوریتم ها یی با پیچیدگی زمانی ازقبیل n و100n را الگوریتم های زمانی خطی می گویند.

- مجموعه کامل توابع پیچیدگی را که با توابع درجه دوم محض قابل دسته بندی باشند، $\theta(n^2)$ می گویند.

- مجموعه ای ازتوابع پیچیدگی که با توابع درجه سوم محض قابل دسته بندی باشند، $\theta(n^3)$ نامیده می شوند.

- برخی از گروه های پیچیدگی متداول در زیر داده شده است:

$$\theta(\lg n) < \theta(n) < \theta(n \lg n) < \theta(n^2) < \theta(n^3) < \theta(2^n)$$

۲-۴-۱آشنایی بیشتر با مرتبه الگوریتم ها

- برای یک تابع پیچیدگی مفروض $f(n)$، O” O (f (n) بزرگ“

مجموعه ای از توابع پیچیدگی g (n) است که برای آن ها یک ثابت حقیقی مثبت c و یک عدد صحیح غیر منفی N وجود دارد به قسمی که به ازای همه ی N =< n داریم:

$$g(n) <= c \times f(n)$$

اوی بزرگ مجانب بالایی g(n) هست

- برای یک تابع پیچیدگی مفروض $f(n)$ ، $\Omega(f(n))$ مجموعه ای از توابع پیچیدگی $g(n)$ است که برای آن ها یک عدد ثابت حقیقی مثبت c و یک عدد صحیح غیر منفی N وجود دارد به قسمی که به ازای همه ی $n >= N$ داریم:

$$g(n) => c \times f(n)$$

برای یک تابع پیچیدگی مفروض $f(n)$، داریم:

$$\theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

یعنی $\theta(f(n))$ مجموعه ای از توابع پیچیدگی $g(n)$ است که برای آن ها ثابت های حقیقی مثبت c و d و عدد صحیح غیر منفی N وجود دارد به قسمی که :

$$c \times f(n) <= g(n) <= d \times f(n)$$

- برای یک تابع پیچیدگی $f(n)$ مفروض،$o(f(n))$ "o کوچک" عبارت ازمجموعه کلیه توابع پیچیدگی$g(n)$ است که این شرط را برآورده می سازند : به ازای هرثابت حقیقی مثبت c ،یک عدد صحیح غیر منفی N وجود دارد به قسمی که به ازای همه ی $n >= N$ داریم:

$$g(n) < c \times f(n)$$

ویژگی های مرتبه

۱ – $g(n) \in O(f(n))$ اگروفقط اگر $f(n) \in \Omega(g(n))$.

۲– $g(n) \in \theta(f(n))$ اگروفقط اگر $f(n) \in \theta(g(n))$.

۳– اگر $b > 1$ و $a > 1$، در آن صورت:

- $\log_a n \in \theta(\log_b n)$

۴– اگر $b > a > 0$،در آن صورت:

$a^n \in o(b^n)$

۵- به ازای همه ی مقادیر $a > 0$ داریم :

$$a^n \in o(n!)$$

۶- اگر $c >= 0$، $d > 0$ ، $g(n) \in o(f(n))$ و $h(n) \in \theta(f(n))$ باشد، در آن صورت:

$$c \times g(n) + d \times h(n) \in \theta(f(n))$$

۷– ترتیب دسته های پیچیدگی زیر را در نظربگیرید:

θ (lg n) θ (n) θ(n lg n) θ(n²) θ(n^j) θ (n^k) θ (aⁿ) θ (bⁿ) θ (n!)

که در آن $k > j > 2$ و $b > a > 1$ است. اگر تابع پیچیدگی $g(n)$ در دسته ای واقع در طرف چپ دسته ی حاوی $f(n)$ باشد، در آن صورت:

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

فصل دوم:

روش تقسیم و حل

- روش تقسیم و حل یک روش بالا به پایین است.

- حل یک نمونه سطح بالای مسئله با رفتن به جزء و بدست آوردن حل نمونه های کوچکتر حاصل می شود.

۳-۲روش تقسیم و حل

راهبرد طراحی تقسیم و حل شامل مراحل زیر است:

۱ – تقسیم نمونه ای ازیک مسئله به یک یا چند نمونه کوچکتر.

۲ – حل هر نمونه کوچکتر. اگر نمونه های کوچک تر به قدر کافی کوچک نبودند، برای این منظور از بازگشت استفاده کنید.

۳– در صورت نیاز، حل نمونه های کوچک تر را ترکیب کنید تا حل نمونه اولیه به دست آید.

❖ هنگام پی ریزی یک الگوریتم بازگشتی ، باید:

۱ – راهی برای به دست آوردن حل یک نمونه ازروی حل یک یا چند نمونه کوچک تر طراحی کنیم.

۲ – شرط(شرایط) نهایی نزدیک شدن به نمونه(های) کوچک تر را تعیین کنیم.

۳– حل را در حالت شرط (شرایط)نهایی تعیین کنیم.

الگوریتم۱–۲: جست و جوی دودویی (بازگشتی)

```
index B_Search ( index low, index high )
{
    index mid;
    if  (low > high )
        return  0;
    else  {
        mid = ⌊ (low + high) /2 ⌋ ;
        if (x = = S [mid])
            return  mid;
 else if ( x < S [mid])
                return B_Search (low , mid – 1);
        else
                return   B_Search (mid + 1, high);
         }
     }
```

تحلیل پیچیدگی زمانی دربدترین حالت برای الگوریتم جست و جوی دودویی بازگشتی

عمل اصلی: مقایسه x با S [mid].

اندازه ورودی: n ، تعداد عناصر آرایه.

برای n >1 ، n توانی از 2 است

$$W(n) = W(n/2) + 1$$

$$W(1) = 1$$

$$W(n) = \lfloor \lg n \rfloor + 1 \in \theta(\lg n)$$

2-2 مرتب سازی ادغامی

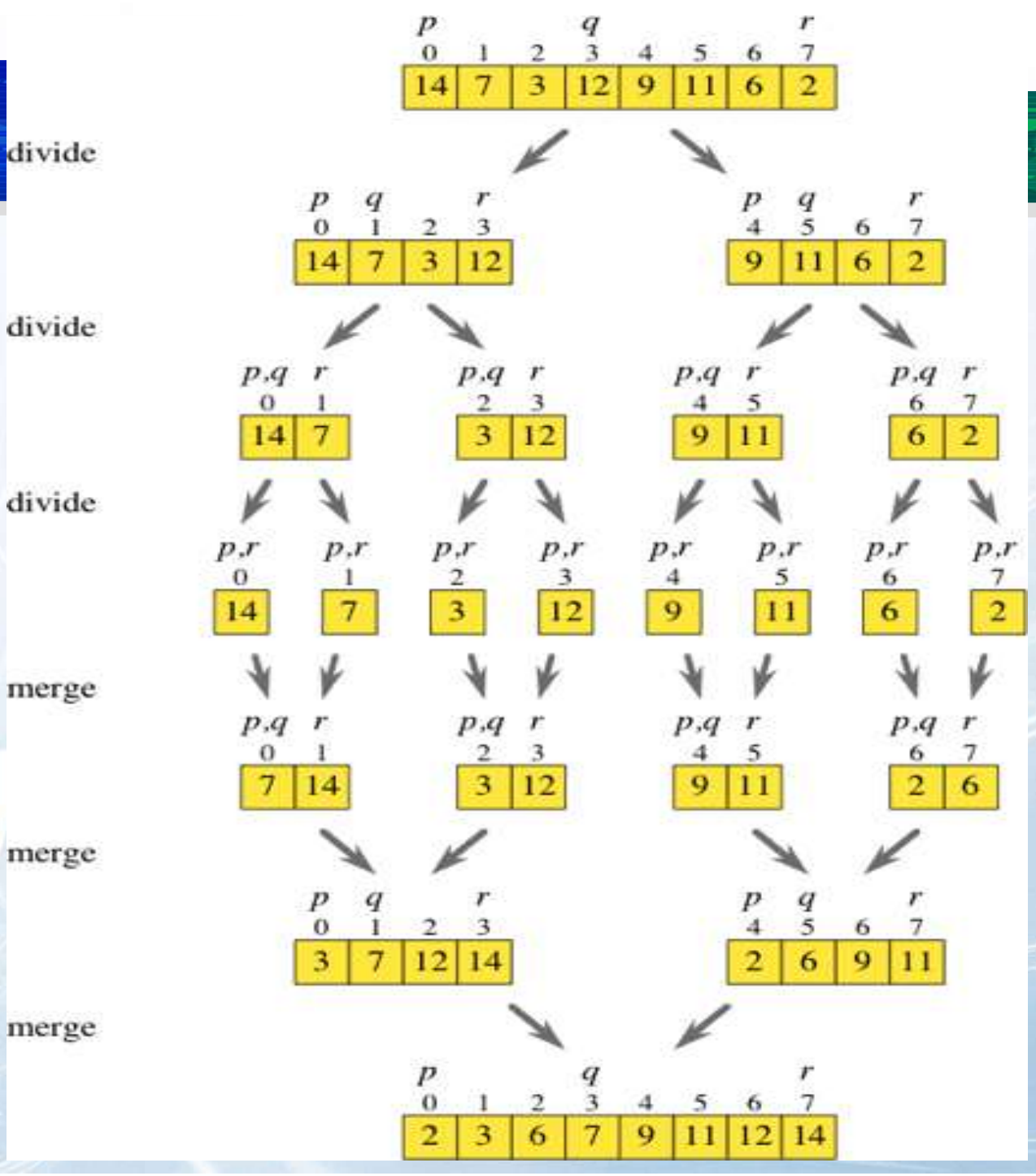
- ادغام یک فرآیند مرتبط با مرتب سازی است.
- ادغام دوطرفه به معنای ترکیب دو آرایه مرتب شده در یک آرایه ی مرتب است.

- مرتب سازی ادغامی شامل مراحل زیر می شود:

۱ – تقسیم آرایه به دو زیر آرایه، هر یک با n/2 عنصر.

۲ – حل هر زیر آرایه با مرتب سازی آن.

۳– ترکیب حل های زیر آرایه ها از طریق ادغام آن ها در یک آرایه مرتب.


divide
divide
divide
merge
merge
merge
p q r
0 1 2 3 4 5 6 7
14 7 3 12 9 11 6 2
p,q r
p,r
2 3
4 5
6 7
3 12
9 11
6 2
7 14
2 6
3 7 12 14
2 3 6 7 9 11 12 14

الگوریتم۲-۲: مرتب سازی ادغامی

```
void  mergsort (int n , keytype S [ ])
  {
       const int  h = ⌊ n/2 ⌋ , m = n – h;
       keytype U [1...h],V [1..m];
       if (n >1)  {
          copy S[1] through  S[h] to U[1] through U[h];
          copy S [h + 1] through S[n] to V[1] through V[m];
          mergesort(h, U);
          mergesort(m,V);
          merge (h , m , U,V,S);
         }
   }
```

الگوریتم ۳–۲: ادغام

```
void  merge ( int h , int m, const keytype U[ ],
                                const keytype V[ ],
                                      keytype  S[ ] )
{
    index i , j , k;
    i = 1; j = 1 ; k = 1;
    while (i <= h && j <= m) {
        if (U [i] < V [j]) {
            S [k] = U [i]
            i+ + ;
}
```

```
        }
    else {
        S [k] = V [j];
         j+ +;
      }
      k+ +;
}
if ( i > h)
     copy V [j] through V [m] to S [k]  through  S [ h + m ]
else
     copy U [i] through U [h] to S [k]  through  S [ h + m ]
}
```

تحلیل پیچیدگی زمانی دربدترین حالت برای الگوریتم ۳-۲(ادغام)

عمل اصلی: مقایسه[i] U با [j]V .

اندازه ورودی: h وm ،تعداد عناصر موجود در هر یک از دو آرایه ورودی.

$$W(h, m) = h + m - 1$$

تحلیل پیچیدگی زمانی دربدترین حالت برای الگوریتم ۲-۲(مرتب سازی ادغامی)

عمل اصلی: مقایسه ای که درادغام صورت می پذیرد.

اندازه ورودی: n ، تعداد عناصر آرایه S.

$$W(n) = W(h) + W(m) + h + m - 1$$

↓ زمان لازم برای مرتب سازی U ↓ زمان لازم برای مرتب سازی V ↓ زمان لازم برای ادغام

برای $n > 1$ که n توانی از 2 است

$$W(n) = 2\,W(n/2) + n - 1$$
$$W(1) = 0$$

$$W(n) \in \theta(n \lg n)$$

الگوریتم ۴-۲: مرتب سازی ادغامی ۲(mergesort 2)

```
void mergesort2  (index low, index high)
 {
     index mid;
     if  (low  < high)  {
              mid = ⌊ ( low + high) / 2 ⌋;
            mergesort 2 (low, mid);
            mergesort 2 (mid +1, high);
            merge2(low,mid,high)
        }
     }
```

الگوریتم۵-۲:ادغام۲

مسئله: ادغام دو آرایه ی مرتب S که درmergesort ایجاد شده اند.

```
void  mrge2 (index low, index mid, index high)
{
   index i, j , k;
   keytype U [ low..high]
   i = low; j = mid +1 ; k = low;
   while ( i <= mid && j <=  high)  {
         if ( S [i]  < S [j] )  {
               U [k] = S [i];
               i + + ;
         }
```

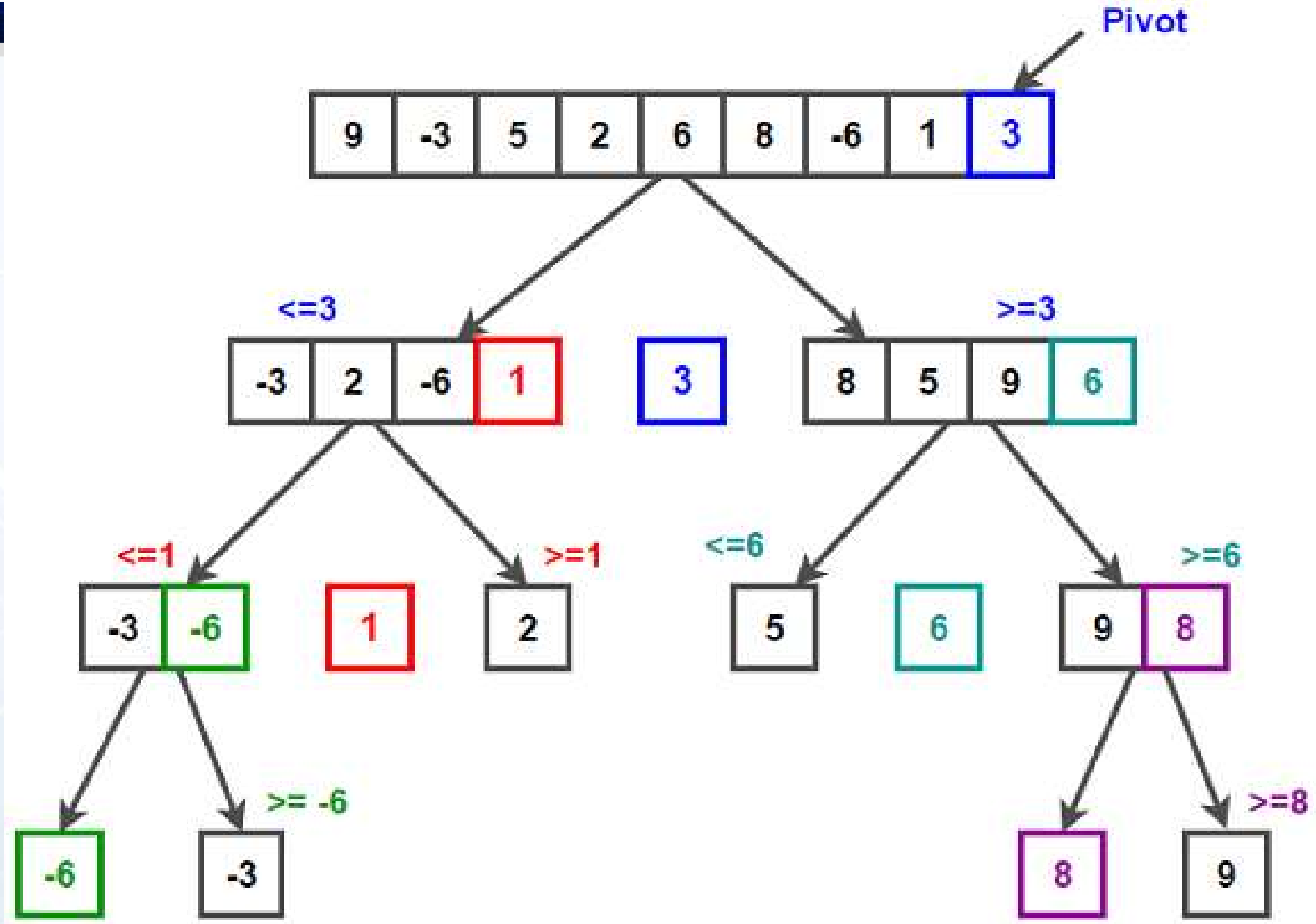
```
    else  {
          U [k] = S [j]
          j ++;
       }
       k ++;
    }
    if ( i >  mid )
          move S [j]  through S [high] to U [k]  through U [high]
    else
           move S [i]  through S [mid] to U [k]  through U [high]
    move U [low] through U [high] to S [low] through S [high]
}
```

۴-۲ مرتب سازی سریع (quicksort)

- در مرتب سازی سریع، ترتیب آنها از چگونگی افراز آرایه ها ناشی می شود.

- همه عناصر کوچک تر از عنصر محوری در طرف چپ آن وهمه عناصربزرگ تر، درطرف راست آن واقع هستند.

- مرتب سازی سریع، به طور بازگشتی فراخوانی می شود تا
  هر یک از دوآرایه را مرتب کند، آن ها نیز افراز می شوند
  واین روال ادامه می یابد تا به آرایه ای با یک عنصربرسیم.
  چنین آرایه ای ذاتاً مرتب است.


Pivot
9 -3 5 2 6 8 -6 1 3
<=3
-3 2 -6 1
3
8 5 9 6
>=3
<=1
-3 -6
1
2
>=1
<=6
5
6
9 8
>=6
-6
-3
>= -6
8
9
>=8

الگوریتم۶-۲ :مرتب سازی سریع

مسئله: مرتب سازی n کلید با ترتیب غیر نزولی.

```
void quicksort  (index low , index high)
 {
        index  pivotpoint;
        if   ( high >  low) {
                partition (low , high , pivotpoint)
                quicksort (low , pivotpoint – 1)
                quicksort (pivotpoint + 1 , high);
          }
  }
```

الگوریتم ۷–۲: افراز آرایه

مسئله: افراز آرایه S برای مرتب سازی سریع.

```
void partition (index low, index high,
                index & pivotpoint)
 {
      index i , j;
      keytype pivotitem;
      pivotitem = S [low];
      j = low
      for ( i = low +1 ; i <= high; i ++)
if ( S [i] < pivotitem )   {
            j++;
            exchange S [i]  and S [j];
      }
   pivotpoint  = j;
   exchange S [low] and S [ pivotpoint];
}
```

تحلیل پیچیدگی زمانی در حالت معمول برای الگوریتم ۷-۲(افراز)

عمل اصلی: مقایسه S [i] با pivotitem .

اندازه ورودی: n = high – low +1 ، تعداد عناصرموجود در زیر آرایه.

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

تحلیل پیچیدگی زمانی در بدترین حالت برای الگوریتم ۶-۲(مرتب سازی سریع)

عمل اصلی: مقایسه S [i] با pivotitem در روال partition .

اندازه ورودی: n ، تعداد عناصر موجود درآرایه S.

$$T(n) = T(0) + T(n-1) + n-1$$

- $T(0)$: زمان لازم برای مرتب سازی زیر آرایه طرف چپ
- $T(n-1)$: زمان لازم برای مرتب سازی زیرآرایه طرف راست
- $n-1$: زمان لازم برای افراز

$T(n) = T(n-1) + n - 1$ به ازای $n > 0$

$T(0) = 0$

$$W(n) = n(n-1)/2 \in \theta(n^2)$$

تحلیل پیچیدگی زمانی در حالت میانگین برای الگوریتم ۶-۲(مرتب سازی سریع)

عمل اصلی: مقایسه S [i] با pivotitem در partition .

اندازه ورودی: n ، تعداد عناصر موجود در S.

$$A(n) \in \theta(n \lg n)$$

الگوریتم‌های مرتب‌سازی را می‌توان به دو دسته درجا (inplace) و برون از جا (outplace) تقسیم‌بندی کرد. در روش‌های درجا فضای مورد استفاده وابسته به اندازه آرایه ورودی نبوده و از مرتبه O(1) است ولی در الگوریتم‌های برون از جا فضای مورد استفاده (مانند مرتب‌سازی ادغامی) تابعی از اندازه ورودی است و هر چقدر تعداد عناصر آرایه (n) بیشتر شود، فضای کمکی مورد نیاز نیز افزایش می‌یابد.

۵-۲الگوریتم ضرب ماتریس استراسن

- پیچیدگی این الگوریتم از لحاظ ضرب، جمع و تفریق بهتر از پیچیدگی درجه سوم است.
- روش استراسن در مورد ضرب ماتریس های ۲×۲ ارزش چندانی ندارد.

الگوریتم ۸-۲: استراسن

مسئله : تعیین حاصلضرب دو ماتریس n× n که در آن n توانی از ۲ است.

```
void starssen ( int n
                n × n _ matrix A,
                n × n _ matrix B,
                n × n _ matrix & C)
{
   if ( n <= threshold)
      compute C = A × B using the standard algorithm;
```

```
    else {
        partition A into four submatrics A11, A12 , A21,A22;
        partition B into four submatrics B11, B12 , B21,B22;
        compute C = A × B using Starssen's Method;
    }
}
```

Strassen's Matrix Multiplication

Basic Matrix Multiplication

Suppose we want to multiply two matrices of size N x N: for example A x $B = C$.

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$$C_{11} = a_{11}b_{11} + a_{12}b_{21}$$

$$C_{12} = a_{11}b_{12} + a_{12}b_{22}$$

$$C_{21} = a_{21}b_{11} + a_{22}b_{21}$$

$$C_{22} = a_{21}b_{12} + a_{22}b_{22}$$

2x2 matrix multiplication can be accomplished in 8 multiplication.($2^{\log_2 8} = 2^3$)

Basic Matrix Multiplication

```
void matrix_mult (){
        for (i = 1; i <= N; i++) {
    for (j = 1; j <= N; j++) {
        compute C_{i,j};
                    }
 }}
```

algorithm

Time analysis

$$C_{i,j} = \sum_{k=1}^{N} a_{i,k} b_{k,j}$$

$$\text{Thus } T(N) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} c = cN^3 = O(N^3)$$

Strassens's Matrix Multiplication

- Strassen showed that 2x2 matrix multiplication can be accomplished in 7 multiplication and 18 additions or subtractions. .($2^{\log_2 7} = 2^{2.807}$)

- This reduce can be done by Divide and Conquer Approach.

Divide-and-Conquer

- Divide-and conquer is a general algorithm design paradigm:
  - Divide: divide the input data S in two or more disjoint subsets S_1, S_2, …
  - Recur: solve the subproblems recursively
  - Conquer: combine the solutions for S_1, S_2, …, into a solution for S
- The base case for the recursion are subproblems of constant size
- Analysis can be done using **recurrence equations**

Divide and Conquer Matrix Multiply

$$A \times B = R$$

$$\begin{bmatrix} A_0 & A_1 \\ A_2 & A_3 \end{bmatrix} \times \begin{bmatrix} B_0 & B_1 \\ B_2 & B_3 \end{bmatrix} = \begin{bmatrix} A_0 \times B_0 + A_1 \times B_2 & A_0 \times B_1 + A_1 \times B_3 \\ A_2 \times B_0 + A_3 \times B_2 & A_2 \times B_1 + A_3 \times B_3 \end{bmatrix}$$

- Divide matrices into sub-matrices: A_0 , A_1, A_2 etc
- Use blocked matrix multiply equations
- Recursively multiply sub-matrices

Divide and Conquer Matrix Multiply

$$A \times B = R$$

$$\boxed{a_0} \times \boxed{b_0} = \boxed{a_0 \times b_0}$$

- Terminate recursion with a simple base case

Strassens's Matrix Multiplication

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
$P_2 = (A_{21} + A_{22}) * B_{11}$
$P_3 = A_{11} * (B_{12} - B_{22})$
$P_4 = A_{22} * (B_{21} - B_{11})$
$P_5 = (A_{11} + A_{12}) * B_{22}$
$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$
$P_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$C_{11} = P_1 + P_4 - P_5 + P_7$
$C_{12} = P_3 + P_5$
$C_{21} = P_2 + P_4$
$C_{22} = P_1 + P_3 - P_2 + P_6$

Comparison

$$C_{11} = P_1 + P_4 - P_5 + P_7$$
$$= (A_{11} + A_{22})(B_{11} + B_{22}) + A_{22} * (B_{21} - B_{11}) - (A_{11} + A_{12}) * B_{22} + (A_{12} - A_{22}) * (B_{21} + B_{22})$$
$$= A_{11} B_{11} + A_{11} B_{22} + A_{22} B_{11} + A_{22} B_{22} + A_{22} B_{21} - A_{22} B_{11} - A_{11} B_{22} - A_{12} B_{22} + A_{12} B_{21} + A_{12} B_{22} - A_{22} B_{21} - A_{22} B_{22}$$
$$= A_{11} B_{11} + A_{12} B_{21}$$

Strassen Algorithm

```
void matmul(int *A, int *B, int *R, int n) {
 if (n == 1) {
    (*R) += (*A) * (*B);
 } else {
    matmul(A, B, R, n/4);
    matmul(A, B+(n/4), R+(n/4), n/4);
    matmul(A+2*(n/4), B, R+2*(n/4), n/4);
    matmul(A+2*(n/4), B+(n/4), R+3*(n/4), n/4);
    matmul(A+(n/4), B+2*(n/4), R, n/4);
    matmul(A+(n/4), B+3*(n/4), R+(n/4), n/4);
    matmul(A+3*(n/4), B+2*(n/4), R+2*(n/4), n/4);
    matmul(A+3*(n/4), B+3*(n/4), R+3*(n/4), n/4);
 }
```

Divide matrices in sub-matrices and recursively multiply sub-matrices

Time Analysis

$$T(1) = 1 \qquad (\text{assume } N = 2^k)$$

$$T(N) = 7T(N/2)$$

$$T(N) = 7^k T(N/2^k) = 7^k$$

$$T(N) = 7^{\log N} = N^{\log 7} = N^{2.81}$$

تحلیل پیچیدگی زمانی تعداد ضرب ها در الگوریتم ۸-۲(استرسن)در حالت معمول

عمل اصلی: یک ضرب ساده.

اندازه ورودی: n ، تعداد سطرها و ستون ها در ماتریس.

به ازای $n > 1$ که n توانی از 2است $T(n) = 7\,T(n/2)$

$T(1) = 1$

$$T(n) \in \theta(n^{2.81})$$

تحلیل پیچیدگی زمانی تعدادجمع هاو تفریقهای الگوریتم (استرسن)درحالت معمول

عمل اصلی: یک جمع یا تفریق ساده.

اندازه ورودی: n ، تعداد سطرها و ستون ها در ماتریس.

به ازای $n > 1$ که n توانی از 2است $T(n) = 7T(n/2) + 18(n/2)^2$

$T(1) = 1$

$$T(n) \in \theta(n^{2.81})$$

الگوریتم۹–۲: ضرب اعداد صحیح بزرگ

مسئله: ضرب دو عدد صحیح بزرگ u و v

```
large _ integer prod ( large_integer u, large_integer v)
{
  large_inreger 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 obtained in the usual way;
else {
      m = ⌊ n / 2 ⌋;
       x = u divide 10 ^ m ;  y = rem 10 ^ m;
      w = v divide 10 ^ m ;  z = rem 10 ^ m;
       return prod (x ,w) × 10 ^2m +  ( prod ( x, z) + prod
(w, y )) × 10 ^ m + prod ( y, z);
      }
}
```

تحلیل پیچیدگی زمانی در بدترین حالت برای الگوریتم۹-۲(ضرب اعداد صحیح)

عمل اصلی: دستکاری یک رقم دهدهی در یک عدد صحیح بزرگ در هنگام جمع کردن ، تفریق کردن، یا انجام اعمال **divide 10 ^ m**، **rem 10 ^m** یا **x10 ^ m**. هر یک از این اعمال را m بار انجام می دهد.

اندازه ورودی: n ، تعداد ارقام هر یک از دو عدد صحیح.

به ازای $n > s$ که n توانی از 2است $W(n) = 4\,W(n/2) + cn$

$W(s) = 0$

$$W(n) \in \theta(n^2)$$

الگوریتم ۱۰–۲: ضرب اعداد صحیح بزرگ ۲

```
large_integer prod2 (large_integer u , large_ integer v)
{
 large_integer x , y , w , z , r , p , q;
 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 obtained in the  usual way;
```

```
    else {
        m = ⌊ n / 2 ⌋;
          x = u divide  10 ^ m  ;  y = rem 10 ^ m;
          w = v divide  10 ^ m  ;  z = rem 10 ^ m;
           r =  prod2 (x + y, w + z );
           p = prod2 ( x , w )
           q = prod2 ( y , z );
           return  p ×10 ^ 2m + ( r – p – q ) × 10 ^ m +q ;
}
     }
```

تحلیل پیچیدگی زمانی در بدترین حالت برای الگوریتم۱۰-۲(ضرب اعداد صحیح۲)

عمل اصلی: دستکاری یک رقم دهدهی در یک عدد صحیح بزرگ در هنگام جمع کردن ، تفریق کردن، یا انجام اعمال **divide 10 ^ m**، **rem 10 ^m** یا **x10 ^ m**. هر یک از این اعمال را m بار انجام می دهد.

اندازه ورودی: n ، تعداد ارقام هر یک از دو عدد صحیح.

به ازای $n > s$ که n توانی از 2است

$$3W(n/2)+ c\ n <= W(n) <= 3W(n / 2 +1) + c\ n$$

$$W(s) = 0$$

$$\mathbf{W(n) = \theta(n \wedge 1.58)}$$

الگوریتم ۶-۱: جمله n ام فیبوناچی (بازگشتی)

مسئله : جمله n ام از دنباله فیبوناچی را تعیین کنید.

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

الگوریتم ۷–۱:جمله nام فیبوناچی (تکراری)

```
int fib2 (int n)
 {
     index i;
     int 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];
}
```

تحلیل پیچیدگی زمانی برای حالت معمول برای الگوریتم(جمع کردن عناصرآرایه)

عمل اصلی: افزودن یک عنصر از آرایه به sum.

اندازه ورودی: n، تعداد عناصر آرایه.

$$T(n) = n$$

کجا نباید از تقسیم و غلبه استفاده کرد

- در صورت امکان در موارد زیر ازروش تقسیم و غلبه استفاده نمی کنیم چرا که مرتبه الگوریتم نمایی می شود :

۱ – هنگامی که نمونه ای با اندازه nبه دو یا چند نمونه تسیم می شود که اندازه آن ها نیز تقریباً برابر با nاست .مثلاً :

$$T(n) = 2T(n-1) \Rightarrow T(n) = \theta(2^n)$$

۲– هنگامی که نمونه ای با اندازه , nتقریباً به nنمونه با اندازه n/c تقسیم شود که cیک مقدار ثابت است

- توجه کنید که همه مسائل را نمی توان به نمونه های کوچک تر تقسیم کرد و سپس نتایج حاصله را با هم ترکیب نمود. مثلاً اگر بخواهیم بزرگ ترین عدد بین ۲۰ عدد را پیدا کنیم می توان آن را به دو دسته ۱۰ تایی تقسیم کرد , سپس ماکزیمم هر دسته را یافته و با مقایسه این دو بزرگ ترین را پیدا کرد.ولی مثلا در مورد یک دستگاه ۲۰ معادله ۲۰ مجهولی نمی توان آن را به دو دستگاه کوچک تر ۱۰ معادله و ۱۰ مجهولی تقسیم کرد

کجا نباید از تقسیم و غلبه استفاده کرد

در صورت امکان در موارد زیر از روش تقسیم و غلبه استفاده نمی‌کنیم چرا که مرتبه الگوریتم نمایی می‌شود:

۱ـ هنگامی که نمونه‌ای با اندازه n به دو یا چند نمونه تقسیم شود که اندازه آنها نیز تقریباً برابر n است، مثلاً :

$$T(n) = 2T(n-1) \Rightarrow T(n) = \theta(2^n)$$

۲ـ هنگامی که نمونه‌ای با اندازه n ، تقریباً به n نمونه با اندازه $\frac{n}{c}$ تقسیم شود که c یک مقدار ثابت است.