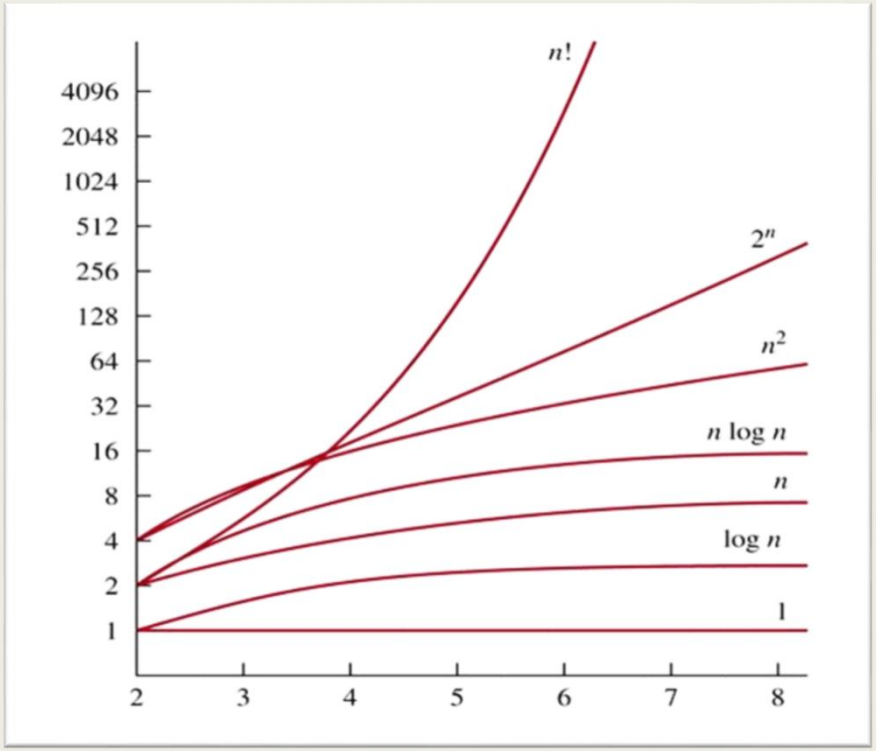
آشنایی بیشتر با مرتبه الگوریتم ها

برخی از گروه های پیچیدگی متداول

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



برنامه های بازگشتی(Recursive)

- **بعضی از مسائل ماهیت بازگشتی دارند**
- **دو ویژگی اصلی توابع بازگشتی**
  - ***هر تابع خودش را صدا می زند(با ورودی کوچکتر)***
  - ***یک شرط جهت توقف فراخوانی ها وجود دارد.***

برنامه های بازگشتی(Recursive)

- **مثال: محاسبه فاکتوریل یک عدد**

1! =1

2! = 1 × 2 =1! × 2

3! = 1 × 2 × 3 = 2! × 3 = 6

4! = 1 × 2 × 3 × 4 = 3! × 4 = 24

5! = 1 × 2 × 3 × 4 × 5 = 4! × 5 = 120

…

n! = 1 × 2 × 3 × … × n = (n-1)! × n

برنامه های بازگشتی(Recursive)

- **در تحلیل الگوریتم های بازگشتی، تابع زمانی معمولا به صورت یک رابطه بازگشتی ظاهر می شود.**
- **روش های حل روابط و معادلات بازگشتی**
  - ***جایگذاری با تکرار***
  - ***استفاده از سری مولد***
  - ***معادله مشخصه***

برنامه های بازگشتی– جایگذاری با تکرار

- **مثال: رابطه بازگشتی زیر را به روش جایگذاری حل کنید.**

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

$$= T(n-2) + c + c = T(n-2) + 2c$$

$$= T(n-3) + c + c + c = T(n-3) + 3c$$

$$\dots$$

$$= T(1) + \cdots + c + c = T(1) + nc$$

$$= 1 + \cdots + c + c = 1 + nc \xrightarrow{\text{مرتبه زمانی}} O(n)$$

برنامه های بازگشتی– جایگذاری با تکرار

- **مثال: رابطه بازگشتی زیر را به روش جایگذاری حل کنید.**

$$T(n) = \begin{cases} (n-1) + T(n-1) & n \geq 1 \\ 0 & n < 1 \end{cases}$$

$$\mathrm{T(n)} = (n-1) + T(n-1)$$

$$= (n-1) + (n-2) + T(n-2)$$

$$\dots$$

$$= (n-1) + (n-2) + \cdots + 3 + 2 + 1 + 0$$

$$\mathrm{T(n)} = \frac{n(n-1)}{2} \xrightarrow[\text{زمانی مرتبه}]{} O(n^2)$$

برنامه های بازگشتی– جایگذاری با تکرار

- **مثال: رابطه بازگشتی زیر را به روش جایگذاری حل کنید.**

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

$$= T(n-2) + c + c = T(n-2) + 2c$$

$$= T(n-3) + c + c + c = T(n-3) + 3c$$

$$\dots$$

$$= T(1) + \cdots + c + c = T(1) + nc$$

$$= 1 + \cdots + c + c = 1 + nc \rightarrow O(n)$$

روش های حل مساله

- تکنیک های مختلفی است که بهینه بودن آنها تا کنون به اثبات رسیده است و می توان از آنها برای حل مسائل مختلف استفاده نمود.
- با مسلط شدن بر این راهکارهای طراحی می توان به راحتی یکی از الگوریتم ها و روش های معروف و جدید را برای مسأله خود کشف نمود.

روش های حل مساله

- از جمله روش های حل مساله
  - روش تقسیم و حل(Divide & Conquer)
  - روش حریصانه (Greedy)
  - روش برنامه نویسی پویا (Dynamic Programming)
  - روش بازگشت به عقب (Back Tracking)

فصل دوم

روش تقسیم و حل

مقدمه

- یک روش بالا به پایین(Top-Down) است
- مساله به مسائل کوچکتر از همان نوع تجزیه می شود.
- با ترکیب حل مسائل کوچکتر، پاسخ مساله اصلی به دست می آید
- مثال:

برای مرتب سازی لیست اعداد، آن را به دو لیست تقسیم کرده سپس هر یک از لیست ها را مرتب می کنیم و با ادغام آنها یک لیست مرتب که حل مساله اصلی است به دست می آید.

ممکن است برای مرتب کرن دو لیست فوق، عمل تقسیم تکرار شود تا جایی که مرتب کردن لیست های حاصل از تقسیم به سادگی انجام شود

- مثالی دیگر: پیمایش درخت دودویی ؟؟؟

جستجوی دودویی(Binary Search)

- **الگوریتم را به صورت بازگشتی می نویسیم زیرا بازگشتی روش کل به جزء را نشان می دهد که در تقسیم و حل به کار می رود.**
  - ***کلید x را در یک آرایه ی مرتب شده از طریق مقایسه x با عنصر میانی آرایه، جستجو می کند.***
  - ***اگر مساوی بودند کار الگوریتم به پایان رسیده است***
  - ***اگر مساوی نبودند به دو زیر آرایه تقسیم می شود که یکی همه عناصر بزرگتر از میانه(سمت چپ) و دیگری همه عناصر کوچکتر(سمت راست) است.***
  - ***اگر x کوچکتر از میانه باشد، الگوریتم***

جستجوی دودویی(Binary Search)

- ***اگر مساوی نبودند به دو زیر آرایه تقسیم می شود که یکی همه عناصر بزرگتر از میانه(سمت راست) و دیگری همه عناصر کوچکتر(سمت چپ) است.***
- ***اگر x کوچکتر از میانه باشد، الگوریتم برای زیرآرایه چپ اجرا می شود در غیر اینصورت الگوریتم در زیرآرایه راستی اجرا خواهد شد.***
- ***این کار تکرار می شود تا x پیدا شود یا مشخص شود x در آرایه وجود ندارد.***

جستجوی دودویی– مراحل

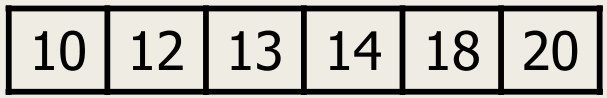
1. اگر x برابر عنصر میانی آرایه بود، کار تمام است. در غیر اینصورت:
2. آرایه را به دو زیر آرایه تقسیم کنید که هر یک حدود نصف آرایه اولیه باشد. اگر x کوچکتر از عنصر میانی است زیرآرایه سمت چپ و اگر بزرگتر است، زیرآرایه سمت راست را انتخاب کنید.
3. با تعیین اینکه آیا x در زیر آرایه است آن را حل کنید در غیراینصورت اگر زیرآرایه به اندازه کافی کوچک نیست برای غلبه از بازگشتی استفاده کنید.
4. حل مساله آرایه را از حل مساله زیرآرایه به دست بیاورید.

- نکته: جستجوی دودویی ساده ترین نوع الگوریتم تقسیم و حل است زیرا مساله تنها شکسته می شود و ترکیبی از خروجی ها وجود ندارد.

جستجوی دودویی– مثال

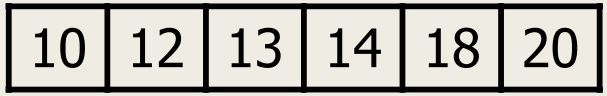
- فرض کنیم x=18 و آرایه زیر را داریم:

- آرایه به دو بخش تقسیم می شود؛ چون x<25، زیرآرایه چپ جستجو می شود:

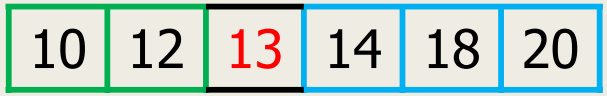


- با تعیین اینکه آیا x در زیرآرایه هست آن را تسخیر می کنیم: بله، x در آرایه هست.
- حل آرایه را از حل زیر آرایه به دست آورید.

جستجوی دودویی– مثال



- آرایه به دو بخش تقسیم می شود؛ چون x>13، زیرآرایه راست جستجو می شود:



- با تعیین اینکه آیا x در زیرآرایه هست آن را تسخیر می کنیم: بله، x در آرایه هست.
- حل آرایه را از حل زیر آرایه به دست آورید.

- چون x=18 کار تمام است.

مقدمه

- **هنگام طراحی یک الگوریتم بازگشتی ، باید:**

1. **راهی برای به دست آوردن حل یک نمونه از روی حل یک نمونه ازروی حل یک یا چند نمونه کوچک تر طراحی کنیم.**
2. **شرط(شرایط) نهایی نزدیک شدن به نمونه(های) کوچک تر را تعیین کنیم.**
3. **حل را در حالت شرط (شرایط)نهایی تعیین کنیم.**

جستجوی دودویی– الگوریتم بازگشتی

- **مسئله**
  - *تعیین کنید آیا X در آرایه مرتب شده S با اندازه n وجود دارد یا خیر*
- **ورودی**
  - *عدد صحیح و مثبت n، آرایه مرتب شده S با اندیس گذاری از ۱ تا n، کلید X*
- **خروجی**
  - *Location برابر موقعیت X در S(صفر اگر وجود نداشته باشد)*

جستجوی دودویی – الگوریتم بازگشتی

```
index location ( 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 location (low , mid – 1);
        else                              return   location (mid + 1, high);
        }
    }
```

جستجوی دودویی – تحلیل

- **تحلیل پیچیدگی زمانی دربدترین حالت**

عمل اصلی: مقایسه x با S [mid]

اندازه ورودی: n ، تعداد عناصر آرایه.

$$W(n) = W\left(\frac{n}{2}\right) + 1$$

برای n >1، n توانی از ۲ است

$$W(n) = W\left(\frac{n}{2}\right) + 1$$

$$W(1) = 1$$

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

مرتب سازی ادغامی(Merge Sort)

- **یک فرآیند مرتبط با مرتب سازی ادغام است.**
- **ادغام دوجانبه، به معنای ترکیب دو آرایه مرتب شده در یک آرایه مرتب شده است. با به کارگیری مکرر روال ادغام، می توان آرایه ای را مرتب کرد.**
- **مثال: جهت مرتب سازی آرایه ۱۶عنصری می توان آن را به دو زیرآرایه تقسیم کرد که تعداد هریک ۸ است سپس دو زیرآرایه های ۸ عنصری را به دو زیرآرایه ۴ عنصری تقسیم کرد و آنها را مرتب کرده ادغام نمود. به همین ترتیب ادامه می دهیم تا در نهایت همه زیرآرایه ها یک عنصری می شوند و آرایه ای به اندازه ۱ به خودی خود مرتب است.**

مرتب سازی ادغامی–مراحل

- تقسیم آرایه به دو زیر آرایه، هریک با نیمی از عناصر
- حل هر یک از زیرآرایه ها با مرتب سازی آن. اگر زیرآرایه به اندازه کافی کوچک نبود، برای حل آن از بازگشتی استفاده می کنیم.
- مرتب کردن حل های زیرآرایه ها از طریق ادغام آنها در یک آرایه مرتب شده

مرتب سازی ادغامی–مراحل

- **مثال: آرایه زیر را در نظر بگیرید:**

27	10	12	20	25	13	15	22

- **حل: ۱– تقسیم آرایه**

27	10	12	20	-	25	13	15	22

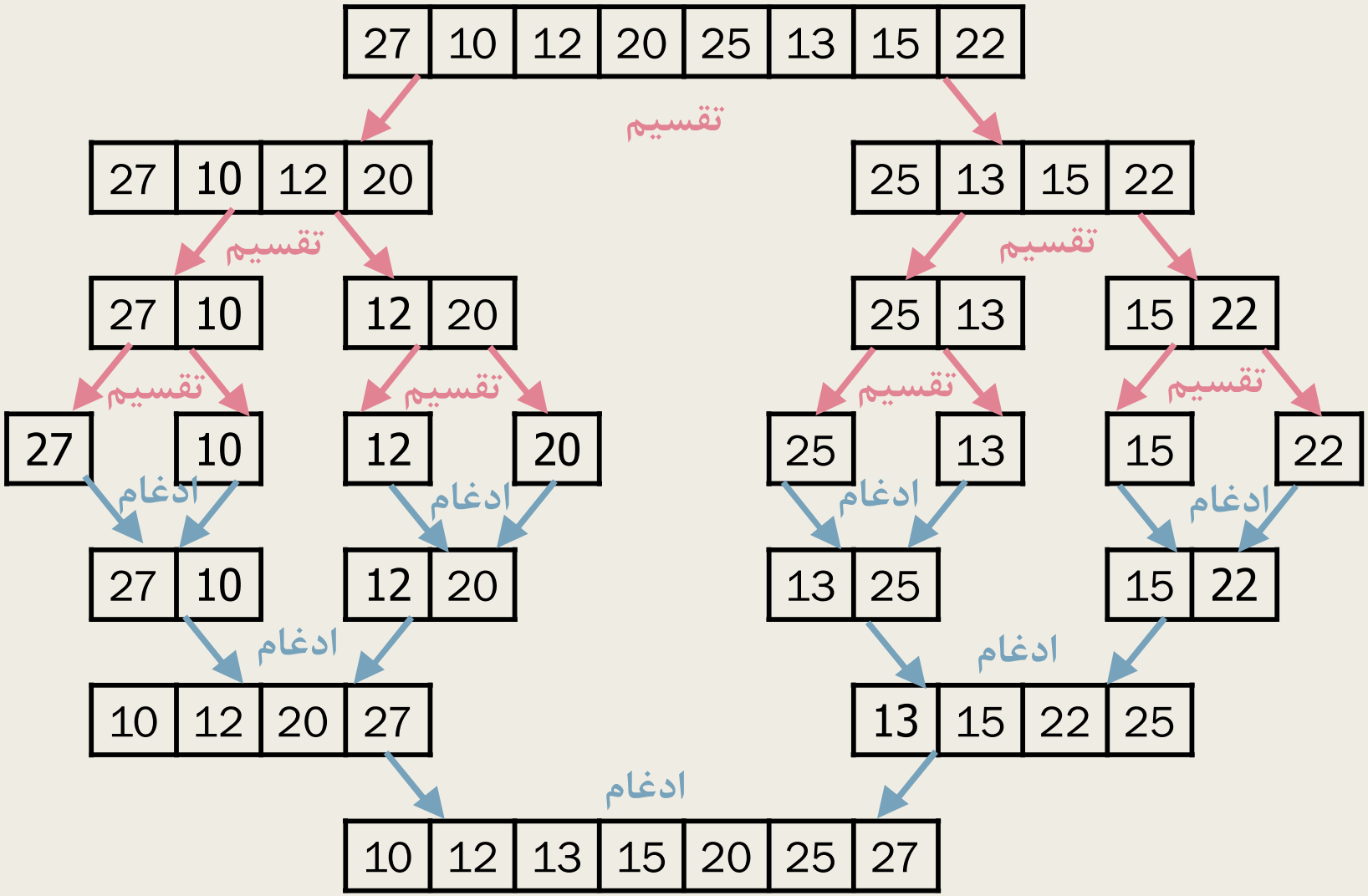
- **۲– مرتب سازی هز یک از زیرآرایه ها**

10	12	20	27	-	13	15	22	25

- **۳– ادغام دو زیر آرایه در هم**

10	12	13	15	20	22	25	27

مرتب سازی ادغامی – مثال



مرتب سازی ادغامی– الگوریتم بازگشتی

- **مسئله**
  - ***مرتب سازی n کلید به ترتیب صعودی***
- **ورودی**
  - ***عدد صحیح و مثبت n، آرایه ای از کلیدهایS با اندیس گذاری از ۱ تا n***
- **خروجی**
  - ***آرایه S حاوی کلیدها به ترتیب صعودی***

مرتب سازی ادغامی – الگوریتم بازگشتی

```
void  mergesort (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[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);
           }
    }
```

مرتب سازی ادغامی– الگوریتم 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+ + ;
```

مرتب سازی ادغامی– الگوریتم ادغام

```
}
    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 ]
 }
```

مرتب سازی ادغامی– تحلیل

- **تحلیل پیچیدگی زمانی ادغام دربدترین حالت**

عمل اصلی: مقایسه[i] U با [j]V

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

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

مرتب سازی ادغامی– تحلیل

- **تحلیل پیچیدگی زمانی مرتب سازی ادغامی دربدترین حالت**

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 سازیV | زمان لازم برای ادغام

برای n >1، n توانی از ۲ است

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

$$W(1) = 0$$

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

مرتب سازی درجا یا ادغامی۲

- مرتب سازی درجا: روشی که از فضایی بیشتر از آنچه مورد نیاز ورودی است استفاده نمی کند.
- الگوریتم قبلی درجا نیست زیرا از آرایه U و V به همراه آرایه ورودی S استفاده می کند.
- میزان حافظه اضافی:

$$n + n/2 + n/4 + \ldots \approx 2n$$

- کاهش مقدار حافظه اضافی به n ؟

مرتب سازی ادغامی۲– الگوریتم

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

مرتب سازی ادغامی ۲– الگوریتم ادغام۲

```
void  merge2 (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 + + ;
          }
```

مرتب سازی ادغامی ۲– الگوریتم ادغام۲

```
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]
}
```

مرتب سازی سریع (quicksort)

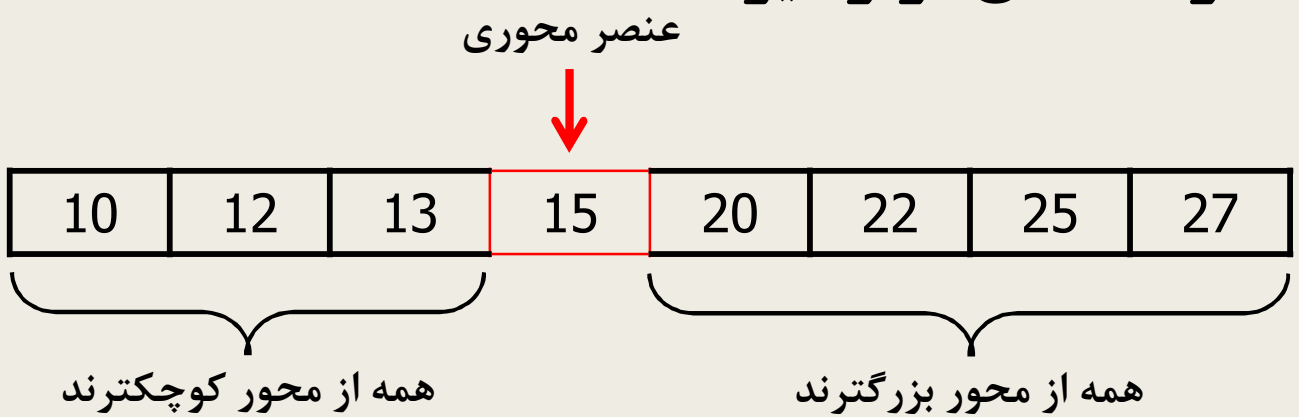
- در مرتب سازی سریع، ترتیب آنها از چگونگی افراز(Partition) آرایه ها ناشی می شود.
- همه عناصر کوچک تر از عنصر محوری در طرف چپ آن وهمه عناصر بزرگ تر، درطرف راست آن واقع هستند.
- عنصر محور می تواند هر عنصریباشد اما برای سادگی عنصر نخست را در نظر می گیریم.
- مرتب سازی سریع، به طور بازگشتی فراخوانی می شود تا هر یک از دوآرایه را مرتب کند، آن ها نیز افراز می شوند و این روال ادامه می یابد تا به آرایه ای با یک عنصربرسیم. چنین آرایه ای ذاتاً مرتب است.

مرتب سازی سریع – مثال

- آرایه زیر را در نظر بگیرید:

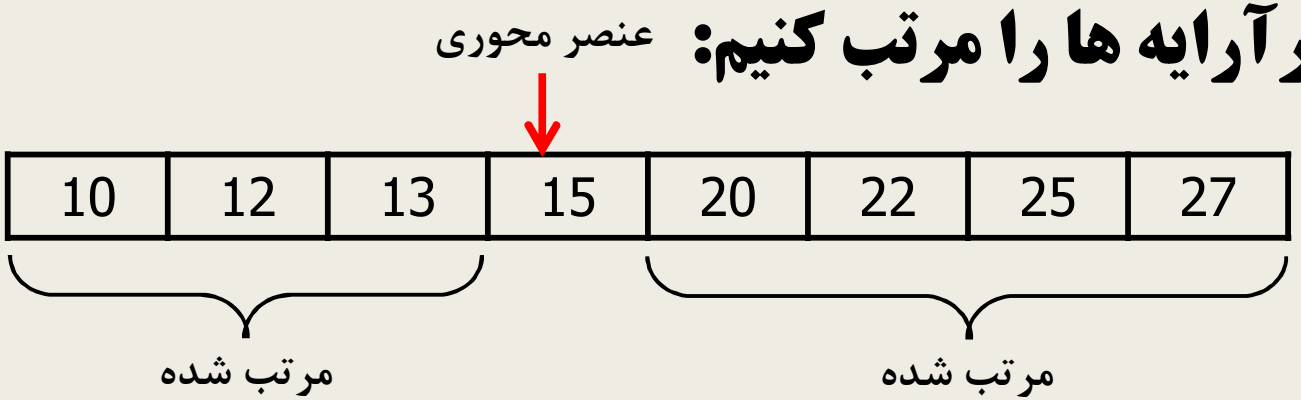
15	22	13	27	12	10	20	25

1. آرایه را چنان افراز می کنیم که همه عناصر کوچکتر از عنصر محوری در سمت چپ آن و همه عناصر بزرگتر در سمت راست آن قرار گیرند.



مرتب سازی سریع – مثال

2. **زیرآرایه ها را مرتب کنیم:**



3. **سپس مرتب سازی سریع به طور بازگشتی فراخوانی می شود تا هر یک از دو زیرآرایه را مرتب کند. آنها نیز افراز می شوند و این روال آنقدر ادامه می یابد تا به آرایه های یک عنصری برسیم که ذاتا مرتب است.**

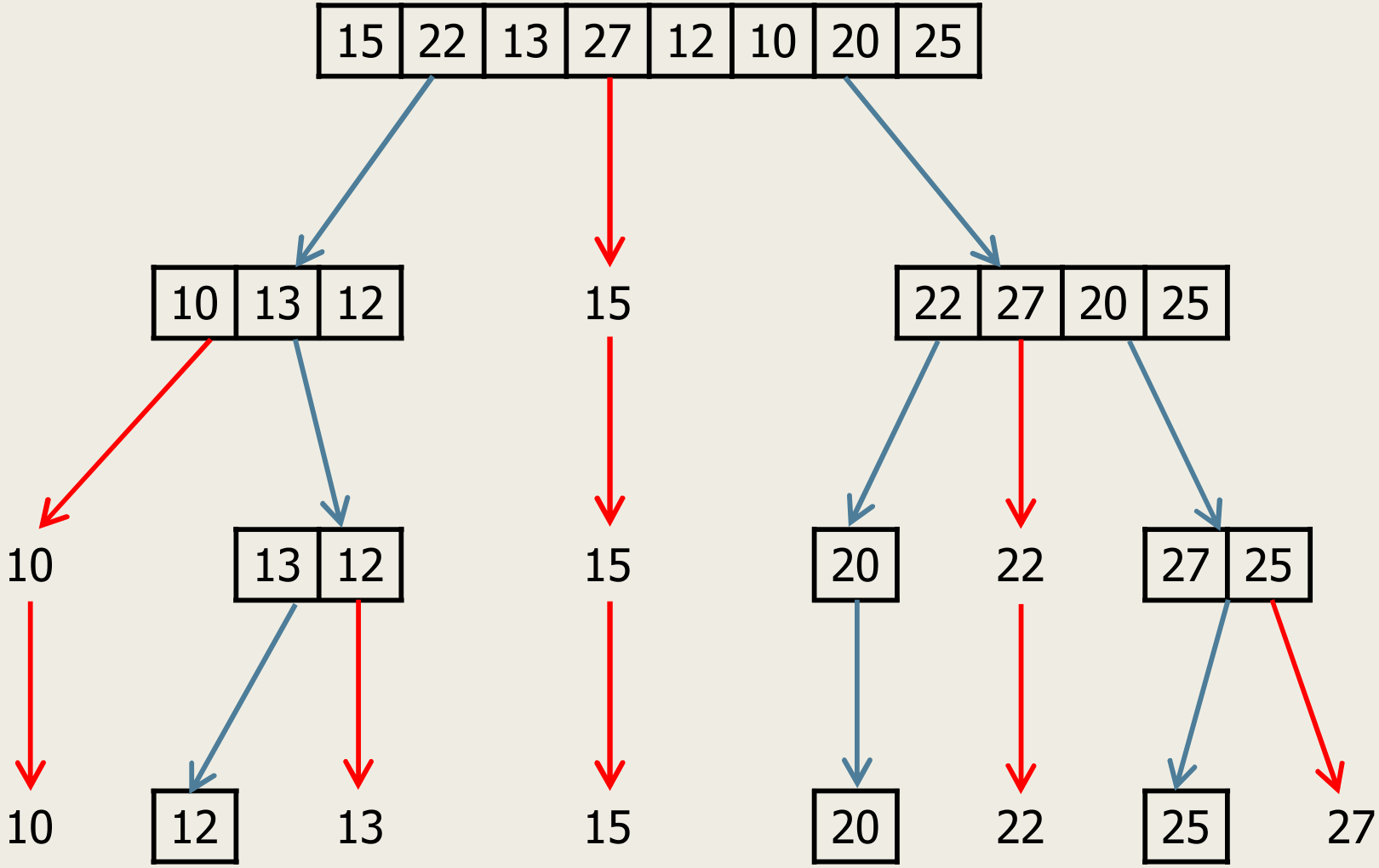
مرتب سازی سریع – الگوریتم

- **مسئله**
  - ***مرتب سازی n کلید به ترتیب صعودی***
- **ورودی**
  - ***عدد صحیح و مثبت n، آرایه ای از کلیدهایS با اندیس گذاری از ۱ تا n***
- **خروجی**
  - ***آرایه S حاوی کلیدها به ترتیب صعودی***

مرتب سازی سریع – الگوریتم

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

مرتب سازی سریع – مثال



مرتب سازی سریع - الگوریتم ادغام

- **مسئله**
  - *افراز آرایه S برای مرتب سازی سریع*
- **ورودی**
  - *دو اندیس Low و High، زیرآرایه S با اندیس کذاری از Low تا High*
- **خروجی**
  - *Pivot point، نقطه محوری زیرآرایه از low تا high*

مرتب سازی سریع – الگوریتم افراز

```
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];
      }
```

مرتب سازی سریع – الگوریتم افراز

i	j	S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]	S[8]
-	-	15	22	13	27	12	10	20	25
2	1	15	22	13	27	12	10	20	25
3	2	15	22	13	27	12	10	20	25
4	3	15	13	22	27	12	10	20	25
5	4	15	13	22	27	12	10	20	25
6	4	15	13	12	27	22	10	20	25
7	4	15	13	12	10	22	27	20	25
8	4	15	13	12	10	22	27	20	25
-	4	10	13	12	15	22	27	20	25

مرتب سازی سریع – تحلیل

- تحلیل پیچیدگی زمانی افراز در حالت میانگین

عمل اصلی: مقایسه S [i] با pivotitem

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

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

مرتب سازی سریع – تحلیل

- **تحلیل پیچیدگی زمانی مرتب سازی سریع در بدترین حالت**

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

اندازه ورودی: n ، تعداد عناصر موجود در آرایه 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 زیر آرایه طرف راست
- $n-1$: زمان لازم برای افراز

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

$T(0) = 0$

$$W(n) = \frac{n(n-1)}{2} \in \theta(n^2)$$

مرتب سازی سریع – تحلیل

- **تحلیل پیچیدگی زمانی مرتب سازی سریع در حالت میانگین**

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

اندازه ورودی: n ، تعداد عناصر موجود در آرایه S.

نیازمند انجام محاسبات ریاضی است که نتیجه برابررابطه زیر می شود

$$A(n) = 1.38\,(n+1)\,lg\,n \in \theta\,(n\,lg\,n)$$

ضرب ماتریس های استراسن(Strassen)

- در فصل قبل الگوریتم ضرب دو ماتریس بر اساس تعریف ریاضی، پیچیدگی زمانی $T(n)=n^3$ داشت.
- پیچیدگی زمانی تعداد جمع ها برابر $T(n)=n^3- n^2=n$ است.
- در سال ۱۹۶۹ استراسن الگوریتمی منتشر کرد که پیچیدگی زمانی آن چه از لحاظ ضرب و چه از لحاظ جمع بهتر از پیچیدگی درجه سوم است.

ضرب ماتریس های استراسن - تشریح

- فرض کنید Aو B دو ماتریس مربعی روی مجموعه اعداد حقیقی باشند. می خواهیم ماتریس حاصلضرب C به این طریق محاسبه می شود:

$$\mathbf{C} = \mathbf{AB} \qquad \mathbf{A}, \mathbf{B}, \mathbf{C} \in R^{2^n \times 2^n}$$

- A، Bو C را به ماتریس های بلوکی هم اندازه افراز می کنیم

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

ضرب ماتریس های استراسن - تشریح

- فرض کنید A و B دو ماتریس مربعی روی مجموعه اعداد حقیقی باشند. می خواهیم ماتریس حاصلضرب C به این طریق محاسبه می شود:

$$\mathbf{C} = \mathbf{AB} \qquad \mathbf{A}, \mathbf{B}, \mathbf{C} \in R^{2^n \times 2^n}$$

- A، Bو C را به ماتریس های بلوکی هم اندازه افراز می کنیم

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

که

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{n-1} \times 2^{n-1}}$$

ضرب ماتریس های استراسن - تشریح

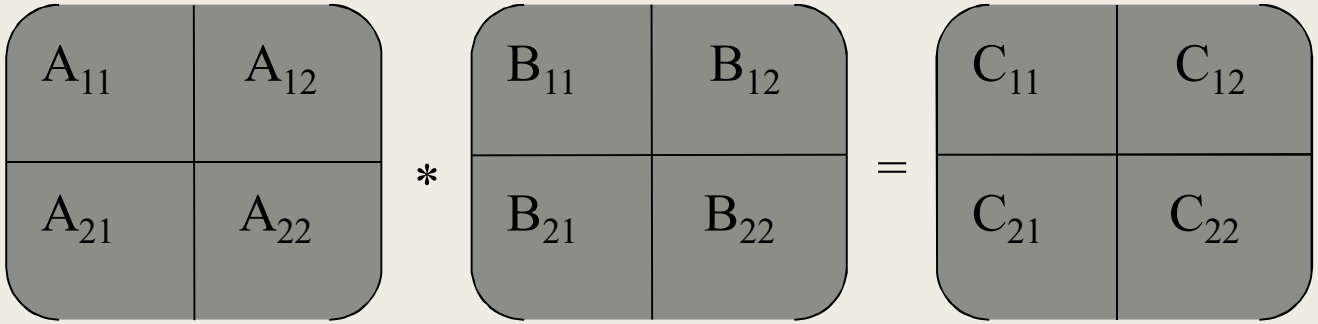
- طبق تعریف اولیه ضرب ماتریس ها داریم:

$$\mathbf{C}_{1,1} = \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{1,2} = \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2}$$
$$\mathbf{C}_{2,1} = \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{2,2} = \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}$$

که برای محاسبه هر عنصر آرایه حاصلضرب $C_{i,j}$، تعداد هشت عملیات ضرب نیاز است.

ضرب ماتریس های استراسن - تشریح

- **الگوریتم استراسن به جای آن ماتریس های جدیدی تعریف می کند:**



$$\mathbf{M}_1 := (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2})$$

$$\mathbf{M}_2 := (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1}$$

$$\mathbf{M}_3 := \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2})$$

$$\mathbf{M}_4 := \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1})$$

$$\mathbf{M}_5 := (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2}$$

$$\mathbf{M}_6 := (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2})$$

$$\mathbf{M}_7 := (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})$$

تنها ۷ عملیات ضرب (برای هر M_k) استفاده می شود.

ضرب ماتریس های استراسن - تشریح

- اکنون $C_{i,j}$ ها را با عبارات حاوی M_k بیان می کنیم:

$$\mathbf{C}_{1,1} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$

$$\mathbf{C}_{1,2} = \mathbf{M}_3 + \mathbf{M}_5$$

$$\mathbf{C}_{2,1} = \mathbf{M}_2 + \mathbf{M}_4$$

$$\mathbf{C}_{2,2} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

- فرآیند افراز را n مرتبه تکرار می کنیم تا هر ماتریس تنها شامل یک عدد شود.
- برای ضرب دو ماتریس ۲×۲ به ۷ عمل ضرب و ۱۸ عمل جمع و تفریق نیاز است.

ضرب ماتریس های استراسن – مثال

- **حاصلضرب دو ماتریس زیر را به روش استراسن محاسبه کنید.**

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{bmatrix} \quad B = \begin{bmatrix} 8 & 9 & 1 & 2 \\ 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 1 \\ 2 & 3 & 4 & 5 \end{bmatrix}$$

$$m_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22}) = \left(\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix}\right) \times \left(\begin{bmatrix} 8 & 9 \\ 3 & 9 \end{bmatrix} + \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 3 & 5 \\ 11 & 13 \end{bmatrix} \times \begin{bmatrix} 17 & 10 \\ 7 & 14 \end{bmatrix} = \begin{bmatrix} 3 \times 17 + 5 \times 7 & 3 \times 10 + 5 \times 14 \\ 11 \times 17 + 13 \times 7 & 11 \times 10 + 13 \times 14 \end{bmatrix}$$

$$= \begin{bmatrix} 86 & 100 \\ 278 & 292 \end{bmatrix}$$

ضرب ماتریس های استراسن - الگوریتم

- **مسئله**
  - ***تعیین حاصلضرب دو ماتریس $n \times n$ که در آن n توانی از ۲ است.***
- **ورودی**
  - ***عدد صحیح n که توانی از ۲ است و دو ماتریس $n \times n$ (A و B)***
- **خروجی**
  - ***ماتریس C حاصلضرب A و B***

ضرب ماتریس های استراسن – الگوریتم

```
void strassen ( 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  submatrices A11, A12 , A21,A22;
          partition  B into  four  submatrices B11, B12 , B21,B22;
          compute C = A × B using Strassen Method;
       }
  }
```

ضرب ماتریس های استراسن – الگوریتم

- تحلیل پیچیدگی زمانی تعداد ضرب ها در حالت میانگین

عمل اصلی: یک ضرب ساده.

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

به ازای $n > 1$ که n توانی از ۲ است

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

$$T(1) = 1$$

$$T(n) = 6n^{lg7} \approx 6n^{2.81} \in \theta(n^{2.81})$$

ضرب ماتریس های استراسن - الگوریتم

- **تحلیل پیچیدگی زمانی تعداد جمع و تفریق ها در حالت میانگین**

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 ۲ است

$$T(n) = 7T(n/2) + 18(n/2)^3$$

$$T(1) = 1$$

$$T(n) = 6n^{lg7} - 6n^2 \approx 6n^{2.81} - 6n^2 \in \theta(n^{2.81})$$

مقایسه دو الگوریتم ضرب ماتریس های n×n

عملیات	الگوریتم استاندارد	الگوریتم استراسن
ضرب ها	n^3	$n^{2.81}$
جمع و تفریق ها	$n^3 - n^2$	$6n^{2.81} - 6n^2$

دلایل استفاده از روش تقسیم و حل

- حل مساله به روش دیگری امکان پذیر نباشد.
- الگوریتم آن نسبت به روش های دیگر ساده تر باشد.
- مرتبه زمانی الگوریتم با این روش کاهش یابد.

موارد عدم استفاده از روش تقسیم و حل

- نمونه با اندازه n به دو یا چند نمونه تقسیم می شود که اندازه آنها نیز تقریبا n است.
- نمونه ای با اندازه n تقریبا به n نمونه با اندازه $\frac{n}{c}$ تقسیم می شود که c یک مقدار ثابت است.