فصل دوم
رهیافت تقسیم و حل

سید ناصر رضوی
E-mail: razavi@Comp.iust.ac.ir

۱۳۸۵

رهیافت بالا به پایین
(Top – Down)

- استراتژی بکار رفته توسط ناپلئون
- نمونه ای از یک مساله را به صورت بازگشتی به تعدادی نمونه کوچکتر تقسیم کن تا زمانی که راه حل نمونه های کوچکتر به سادگی قابل تعیین باشند.
- رهیافت بالا به پایین که توسط روتین های بازگشتی به کار می رود.

جستجوی دودویی

اگر x با عنصر وسط برابر است خارج شو، در غیر این صورت:

۱- **تقسیم** آرایه به دو زیر آرایه با اندازه ای تقریبا برابر نصف اندازه آرایه اولیه. اگر x کوچکتر از عنصر وسط می باشد، آرایه سمت چپ را انتخاب کن. اگر x بزرگتر از عنصر وسط می باشد، آرایه سمت راست را انتخاب کن.

۲- **حل** زیر آرایه به صورت بازگشتی با تعیین این که آیا x در آن زیر آرایه قرار دارد یا خیر، مگر این که اندازه زیر آرایه به اندازه کافی کوچک باشد.

۳- راه حل آرایه را با توجه به راه حل زیر آرایه تعیین کن.

یک مثال

- فرض کنید $x = 18$ و آرایه به صورت زیر باشد:

10 12 13 14 18 20 25 27 30 35 40 45 47

↑

عنصر وسط

مرحله بعدی؟ (شکل 2.1)

کل فرآیند جستجو

10 12 13 14 18 20 25 27 30 35 40 45 47

x را با 25 مقایسه کن

زیر آرایه سمت چپ را
انتخاب کن زیرا $x < 25$

10 12 13 14 18 20

x را با 13 مقایسه کن

زیر آرایه سمت راست را
انتخاب کن زیرا $x > 13$

14 18 20

x را با 18 مقایسه کن

چون $x = 18$ پس x موجود است

توسعه یک الگوریتم بازگشتی

- توسعه روشی برای بدست آوردن راه حل یک نمونه از روی راه حل یک یا چند نمونه کوچکتر
- تعیین شرط (شرایط) نهایی نزدیک شدن به نمونه (های) کوچکتر
- تعیین راه حل در شرط (شرایط) نهایی

جستجوی دودویی

◄ الگوریتم ۲-۱ جستجوی دودویی (بازگشتی)

- مساله: تعیین کنید که آیا x در آرایه مرتب شده S به اندازه n وجود دارد یا خیر.
- ورودی ها: عدد صحیح و مثبت n، آرایه مرتب S که از ۱ تا n اندیس گذاری شده است، کلید x.
- خروجی ها: $location$، موقعیت x در S (اگر x در S نباشد برابر صفر می باشد)

الگوریتم جستجوی دودویی

```
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);
    }
}
```

پیچیدگی زمانی: بدترین حالت

- عمل اصلی: مقایسه x با $S[mid]$
- اندازه ورودی: تعداد عناصر آرایه، n
- پیچیدگی زمانی:

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

- حل: $W(n) = \lg n + 1$

و اگر n به توانی از دو محدود نباشد:

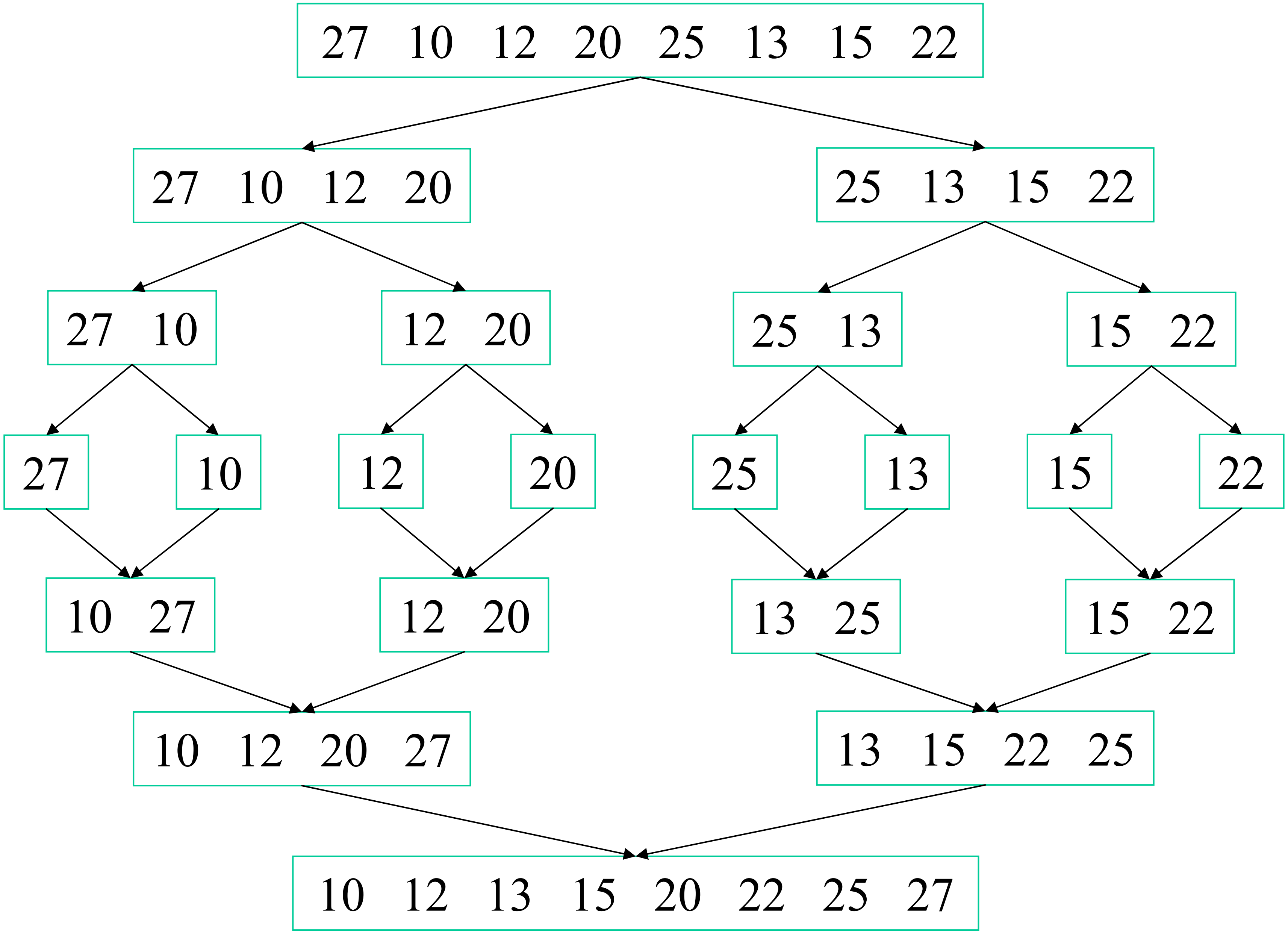
$$W(n) = \lfloor \lg n \rfloor + 1 \in \Theta(\lg n)$$

مرتب سازی ادغامی

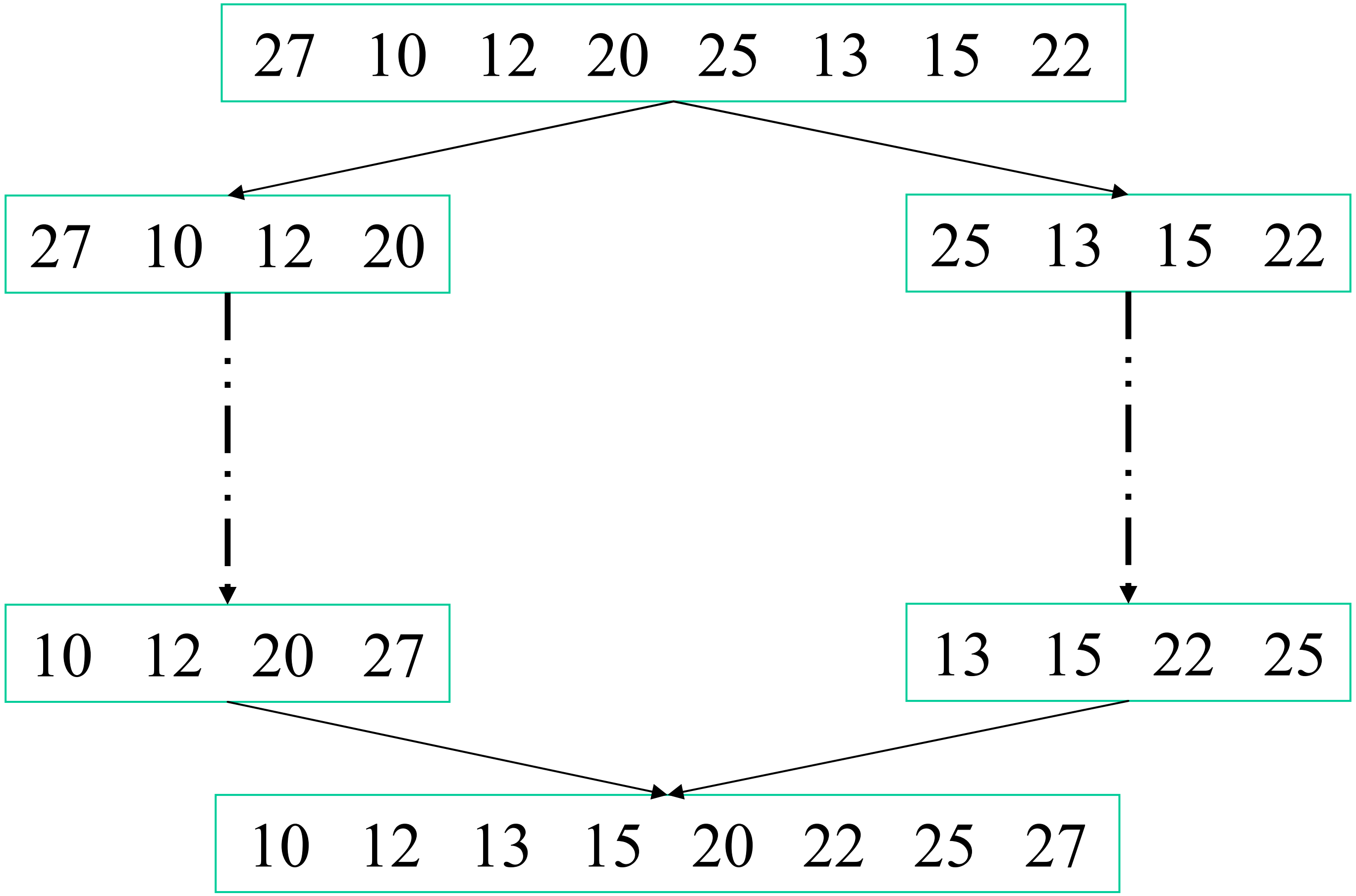
- **تقسیم** آرایه به دو زیر آرایه به اندازه $n/2$ عنصر.

- **حل** هر یک از زیر آرایه ها با مرتب سازی آن. اگر زیر آرایه به اندازه کافی کوچک نبود، برای حل آن به روش بازگشتی عمل می کنیم.

- **ترکیب** راه حل های زیر آرایه ها با ادغام آن ها در یک آرایه مرتب.


27 10 12 20 25 13 15 22
27 10 12 20
25 13 15 22
27 10
12 20
25 13
15 22
27
10
12
20
25
13
15
22
10 27
12 20
13 25
15 22
10 12 20 27
13 15 22 25
10 12 13 15 20 22 25 27

مثال ۲-۲


27 10 12 20 25 13 15 22
27 10 12 20
25 13 15 22
10 12 20 27
13 15 22 25
10 12 13 15 20 22 25 27

الگوریتم مرتب سازی ادغامی

► Algorithm ۲.۲

Merge sort

Problem: Sort n keys in nondecreasing order.

Inputs: positive integer n, array of keys S indexed from 1 to n.

Outputs: the array S containing the keys in nondecreasing order.

```
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[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 ادغام

► Algorithm 2.3 **Merge**

Problem: merge two sorted array into one sorted array.
Inputs: positive integer *h and m*, array of sorted keys *U* indexed from 1 to *h,*
array of sorted keys *V* indexed from 1 to *m.*
Outputs: the array *S* containing the keys in nondecreasing order.

```
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++;
   } // end of while
   if ( i > h)
      copy V[j] thtough V[m] to S[k] through S[h + m];
   else
      copy U[i] thtough U[h] to S[k] through S[h + m];
}
```

پیچیدگی زمانی ادغام: بدترین حالت

- عمل اصلی: مقایسه $U[i]$ با $V[j]$
- اندازه ورودی: h و m، تعداد عناصر موجود در هر یک از دو آرایه ورودی
- پیچیدگی زمانی:

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

- یعنی زمانی که هنگام خروج از حلقه به دلیل مقایسه تمام عناصر یکی از آرایه ها، آرایه دیگر فقط یک عنصر مقایسه نشده داشته باشد.

پیچیدگی زمانی مرتب سازی ادغامی: بدترین حالت

- عمل اصلی: مقایسه ای که در $merge$ انجام می شود.
- اندازه ورودی: n، تعداد عناصر آرایه S.
- پیچیدگی زمانی:

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

- اگر n توانی از ۲ باشد:

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

- حل: (مثال ۱۹ از پیوست ۲)

$$W(n) = n\lg n - (n - 1) \in \Theta(n\lg n)$$

- و اگر n به توانی از ۲ محدود نباشد:

$$W(n) = W(\left\lfloor \frac{n}{2} \right\rfloor) + W(\left\lceil \frac{n}{2} \right\rceil) + n - 1 \Rightarrow W(n) = \theta(n \lg n)$$

پیچیدگی حافظه

- پیچیدگی حافظه الگوریتم ۲-۲:

$$n(1 + 1/2 + 1/4 + \ldots) = 2n$$

- برای کاهش پیچیدگی حافظه به n:

▶ Algorithm 2.4 Mergesort 2

Problem: Sort n keys in nondecreasing sequence.

Inputs: positive integer n, array of keys S indexed from 1 to n.

Outputs: the array S containing the keys in nondecreasing order.

```
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);
  }
}
```

الگوریتم ادغام ۲

▶ Algorithm 2.5 **Merge 2**

Problem: Merge the two sorted subarrays of S created in Mergesort 2.

Inputs: indices *low*, *mid*, and *high*, and the subarray of S indexed from *low* to *high*. The keys in array slots from *low* to *mid* are already sorted in nondecreasing order, as are the keys in array slots from $mid + 1$ to *high*.

Outputs: the subarray of S indexed from *low* to *high* containing the keys in nondecreasing order.

```
void merge2 (index low, index mid, index high)
{
  index i, j, k;
  keytype U[low..high];    // A local array needed for the
                           // merging
  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];
}
```

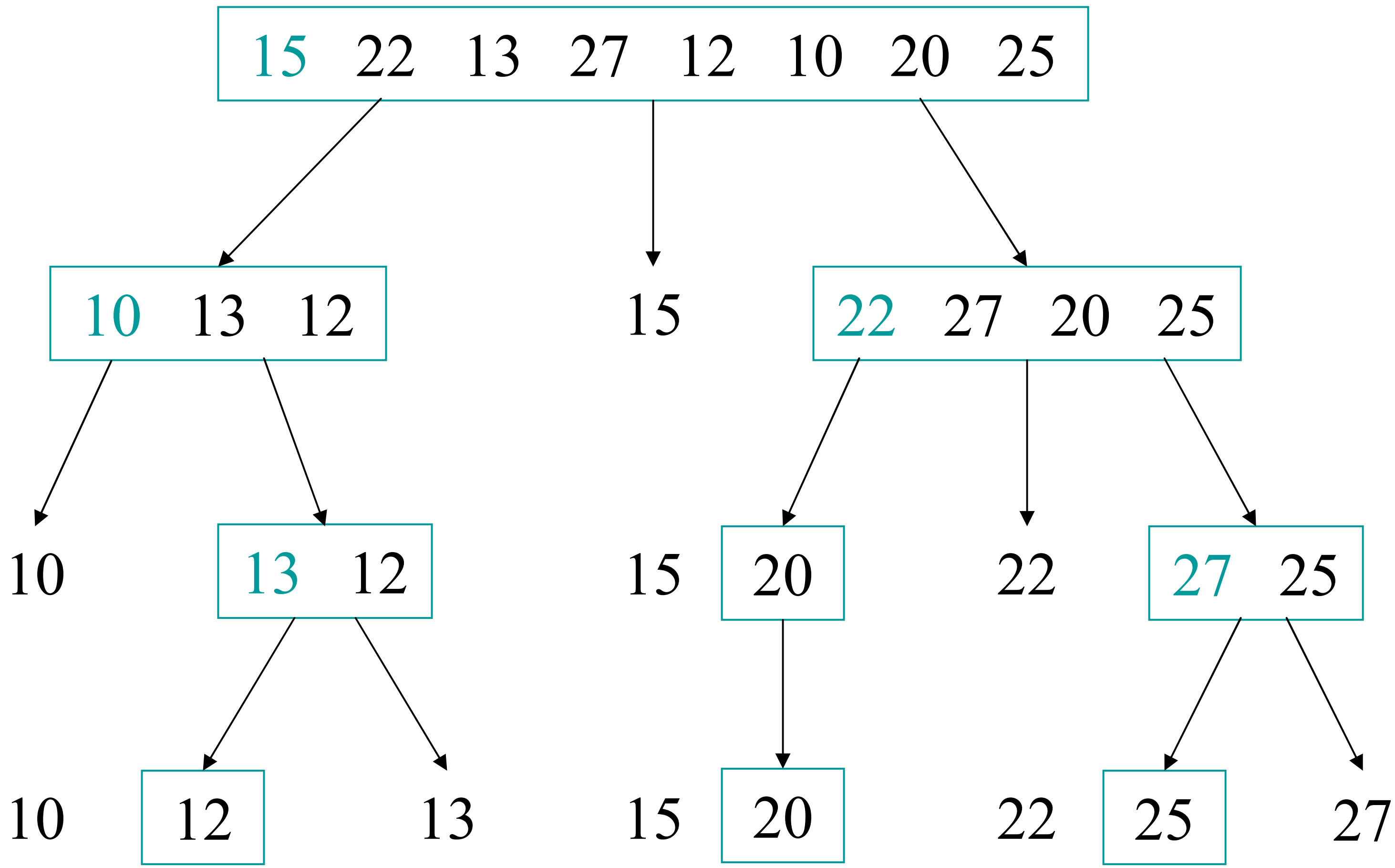
رهیافت تقسیم و حل

- **تقسیم** یک نمونه از مساله به یک یا چند نمونه کوچکتر.
- **حل** هر یک از نمونه های کوچکتر. اگر اندازه نمونه به اندازه کافی کوچک نیست، به روش بازگشتی عمل می کنیم.
- **ترکیب** راه حل های نمونه های کوچکتر برای بدست آوردن راه حل نمونه اصلی در صورت لزوم.

مرتب سازی سریع

- توسعه یافته توسط Hoare (1962)
- 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

مثال: مرتب سازی سریع



الگوریتم مرتب سازی سریع

▶ Algorithm 2.6 **Quicksort**

Problem: Sort n keys in nondecreasing order.

Inputs: positive integer n, array of keys S indexed from 1 to n.

Outputs: the array S containing the keys in nondecreasing order.

```
void quicksort (index low, index high)
{
  index pivotpoint;

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

الگوریتم بخش بندی

▶ Algorithm 2.7 Partition

Problem: Partition the array S for Quicksort.

Inputs: two indices, *low* and *high*, and the subarray of S indexed from *low* to *high*.

Outputs: *pivotpoint*, the pivot point for the subarray indexed from *low* to *high*.

```
void partition (index low, index high,
                index& pivotpoint)
  {
     index i, j;
     keytype pivotitem;

     pivotitem = S[low];                      // Choose first item for
     j = low;                                 // pivotitem.
     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];       // Put pivotitem at pivotpoint.
}
```

یک مثال از رویه بخش بندی

Table 2.2 An example of procedure *partition**

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	← Initial values
2	1	**15**	**22**	13	27	12	10	20	25	
3	2	**15**	22	**13**	27	12	10	20	25	
4	2	**15**	$\boxed{13}$	$\boxed{22}$	**27**	12	10	20	25	
5	3	**15**	13	22	27	**12**	10	20	25	
6	4	**15**	13	$\boxed{12}$	27	$\boxed{22}$	**10**	20	25	
7	4	**15**	13	12	$\boxed{10}$	22	$\boxed{27}$	**20**	25	
8	4	**15**	13	12	10	22	27	20	**25**	
—	4	$\boxed{10}$	13	12	$\boxed{15}$	22	27	20	25	← Final values

*Items compared are in boldface. Items just exchanged appear in squares.

پیچیدگی زمانی بخش بندی: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 مرتب سازی سریع: بدترین حالت

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

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

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

- حل:

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

پیچیدگی زمانی مرتب سازی سریع: حالت متوسط

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

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

- حل:

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

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 استراسن

- 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:

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

- آنگاه :

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

برای ماتریس های بزرگ ...

- تقسیم ماتریس ها:

$$\begin{array}{cc} & \overleftrightarrow{n/2} \\ \updownarrow n/2 & \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right] \end{array} = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \times \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]$$

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

- محاسبه M ها، مثلا:

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

یک مثال

- وقتی $n = 4$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{array}\right] \times \left[\begin{array}{cc|cc} 8 & 9 & 1 & 2 \\ 3 & 4 & 5 & 6 \\ \hline 7 & 8 & 9 & 1 \\ 2 & 3 & 4 & 5 \end{array}\right]$$

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

- محاسبه M ها، مثلا:

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

الگوریتم استراسن

▶ Algorithm 2.8 **Strassen**

Problem: Determine the product of two $n \times n$ matrices where n is a power of 2.

Inputs: an integer n that is a power of 2, and two $n \times n$ matrices A and B.

Outputs: the product C of A and B.

```
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's method;
        // example recursive call;
        // strassen(n/2, A11 + A22 , B11 + B22 , M1)
  }
}
```

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:

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

مقایسه

Table 2.3 A comparison of two algorithms that multiply $n \times n$ matrices

	Standard Algorithm	Strassen's Algorithm
Multiplications	n^3	$n^{2.81}$
Additions/Subtractions	$n^3 - n^2$	$6n^{2.81} - 6n^2$

محاسبات 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 rem10^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:

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

که $m = \lfloor n/2 \rfloor$

ضرب

- 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:

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

الگوریتم

▶ Algorithm 2.9 **Large Integer Multiplication**

Problem: Multiply two large integers, u and v.

Inputs: large integers u and v.

Outputs: *prod*, the product of u and v.

```
large_integer prod (large_integer u, large_integer v)
{
  large_integer 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 = u rem 10^m;
      w = v divide 10^m; z = v rem 10^m;
      return prod(x,w) × 10^2m + (prod(w,y)) × 10^m + prod(y,z);
  }
}
```

پیچیدگی زمانی: بدترین حالت

- عمل اصلی: دستکاری یک رقم دهدهی در یک عدد صحیح بزرگ هنگام جمع کردن، تفریق کردن، یا انجام اعمال تقسیم بر 10^m ، ضرب در 10^m و محاسبه باقیمانده بر 10^m
- اندازه ورودی: n، تعداد ارقام هر یک از دو عدد
- پیچیدگی زمانی:

$$\begin{cases} W(n) = 4W(\frac{n}{2}) + cn & for \quad n > s, \quad n \quad a \quad power \quad of \quad 2 \\ W(s) = 0 \end{cases}$$

- حل:

$$W(n) \in \Theta(n^{\lg 4}) = \Theta(n^2)$$

کاهش تعداد ضرب ها

- رویه $prod$ باید موارد زیر را محاسبه کند:

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

- رویه $prod$ برای محاسبه موارد مذکور 4 مرتبه فراخوانی می شود.
- اگر $r = (x + y)(w + z) = xw + (xz + yw) + yz$
آنگاه $xz + yw = r - xw - yz$
- بنابراین تنها باید سه حاصل ضرب زیر را محاسبه کنیم:

$$r = (x + y)(w + z),\ xw,\ yz$$

الگوریتم جدید

▶ Algorithm 2.10 **Large Integer Multiplication 2**

Problem: Multiply two large integers, u and v.

Inputs: large integers u and v.

Outputs: *prod2*, the product of u and v.

```
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 = u rem 10^m;
     w = v divide 10^m; z = v 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;
  }
}
```

پیچیدگی زمانی: بدترین حالت

- عمل اصلی: دستکاری یک رقم دهدهی در یک عدد صحیح بزرگ هنگام جمع کردن، تفریق کردن، یا انجام اعمال تقسیم بر 10^m ، ضرب در 10^m و محاسبه باقیمانده بر 10^m
- اندازه ورودی: n، تعداد ارقام هر یک از دو عدد
- پیچیدگی زمانی:

$$\begin{cases} 3W(\frac{n}{2}) + cn \le W(n) \le 3W(\frac{n}{2}+1) + cn & for \quad n > s, \quad n \quad a \quad power \quad of \quad 2 \\ W(s) = 0 \end{cases}$$

- حل:

$$W(n) \in \Theta(n^{\lg 3}) = \Theta(n^{1.58})$$

تعیین مقادیر آستانه

- فرآیند بازگشتی از لحاظ زمانی به قدری سر بار نیاز دارد
- مثال: ممکن است برای مرتب سازی ۸ کلید، الگوریتم مرتب سازی تعویضی $(\Theta(n^2))$ سریعتر از الگوریتم مرتب سازی سریع $(\Theta(n\lg n))$ باشد.

- مساله: تعیین مقدار آستانه بهینه در الگوریتم های تقسیم و حل:
  - یعنی، عمل تقسیم نمونه تا کی انجام شود

- عوامل موثر در مقدار آستانه:
  - الگوریتم تقسیم و حل
  - الگوریتم جانشین
  - کامپیوتر اجراکننده الگوریتم ها

مثال: مرتب سازی ادغامی

- پیچیدگی زمانی در بدترین حالت: (بهینه سازی در بدترین حالت)

$$W(n) = W\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + W\left(\left\lceil \frac{n}{2} \right\rceil\right) + n - 1$$

- اگر زمان لازم برای تقسیم و ترکیب یک نمونه به اندازه n در یک کامپیوتر مفروض برابر با $32n$ میکروثانیه باشد، آنگاه:

$$W(n) = W\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + W\left(\left\lceil \frac{n}{2} \right\rceil\right) + 32n\ \mu s$$

- بنابراین (اگر n توانی از ۲ باشد):

$$W(n) = 2W(n/2) + 32n\ \mu s$$

$$W(1) = 0$$

مثال ۲-۷: تعیین مقدار آستانه برای مرتب سازی ادغامی

- تعیین t

$$W(n)=\begin{cases}\dfrac{n(n-1)}{2}\mu s & n\le t\\ W\left(\left\lfloor \dfrac{n}{2}\right\rfloor\right)+W\left(\left\lceil \dfrac{n}{2}\right\rceil\right)+32n\ \mu s & n>t\end{cases}$$

- حل معادله روبرو

$$W\left(\left\lfloor \frac{t}{2}\right\rfloor\right)+W\left(\left\lceil \frac{t}{2}\right\rceil\right)+32t=\frac{t(t-1)}{2}$$

$$W\left(\left\lfloor \frac{t}{2}\right\rfloor\right)=\frac{\lfloor t/2\rfloor(\lfloor t/2\rfloor-1)}{2}$$

$$W\left(\left\lceil \frac{t}{2}\right\rceil\right)=\frac{\lceil t/2\rceil(\lceil t/2\rceil-1)}{2}$$

$$\frac{\lfloor t/2\rfloor(\lfloor t/2\rfloor-1)}{2}+\frac{\lceil t/2\rceil(\lceil t/2\rceil-1)}{2}+32t=\frac{t(t-1)}{2}$$

مثال ۲-۷: تعیین مقدار آستانه برای مرتب سازی ادغامی(ادامه)

$$\frac{\lfloor t/2 \rfloor(\lfloor t/2 \rfloor - 1)}{2} + \frac{\lceil t/2 \rceil(\lceil t/2 \rceil - 1)}{2} + 32t = \frac{t(t-1)}{2}$$

- اگر t زوج باشد، آنگاه $t = 128$
- اگر t فرد باشد؛ آنگاه $t = 128.008$
- بنابراین، مقدار آستانه بهینه برابر $t = 128$ می باشد

مثال ۲-۸

- پیچیدگی زمانی یک الگوریتم تقسیم و حل بر روی یک کامپیوتر خاص:

$$T(n) = 3T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 16n \ \ \mu s$$

- پیچیدگی زمانی یک الگوریتم تکراری برای حل نمونه ای به اندازه n:

$$T(n) = n^2 \ \ \mu s$$

- تعیین مقدار آستانه بهینه t:

$$3T\left(\left\lceil \frac{t}{2} \right\rceil\right) + 16t = t^2 \ \ \& \ \ T\left(\left\lceil \frac{T}{2} \right\rceil\right) = \left\lceil \frac{t}{2} \right\rceil^2 \Rightarrow$$

$$3\left\lceil \frac{t}{2} \right\rceil^2 + 16t = t^2$$

مثال ۲-۸ (ادامه)

- برای تعیین t باید معادله زیر را حل نمود:

- الف) اگر t زوج باشد، آنگاه $t = 64$
- ب) اگر t فرد باشد، آنگاه $t = 70.04$

- چون دو مقدار برابر نیستند، آستانه بهینه وجود ندارد؛ یعنی:
  - اگر n یک عدد صحیح زوج بین ۶۴ و ۷۰ باشد، بهتر است یک بار دیگر تقسیم شود
  - اگر n یک عدد صحیح فرد بین ۶۴ و ۷۰ باشد، فراخوانی الگوریتم جانشین کارآیی بیشتری دارد
  - اگر n کوچکتر از ۶۴ باشد، همواره فراخوانی الگوریتم جانشین کارآیی بیشتری دارد
  - اگر n بزرگتر از ۷۰ باشد، همواره تقسیم دوباره نمونه کارآتر خواهد بود.

مثال ۲-۸ (ادامه)

- مقایسه کارآیی الگوریتم بازگشتی و جانشین به ازاء مقادیر مختلف n:

n	n^2	$3\left\lceil \frac{n}{2} \right\rceil^2 + 16n$
62	3844	3875
63	3969	4080
64	4096	4096
65	4225	4307
68	4624	4556
69	4761	4779
70	4900	4795
71	5041	5024

مواقعی که نباید از تقسیم و حل استفاده کنیم

- 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. ← $\Theta(n^{\lg n})$

تمرین ها

- بخش 2.1
  - ۴، ۶
- بخش 2.2
  - ۱۰، ۱۳
- بخش 2.3
  - ۱۴، ۱۵، ۱۶، ۱۷، ۱۸
- بخش 2.4
  - ۲۴
- بخش 2.5
  - ۲۶، ۲۹
- بخش 2.7
  - ۳۶
- بخش 2.8
  - ۳۷، ۳۸
- تمرینات اضافی
  - ۳۹ و ۴۰و ۴۱