به نام خدا

فصل دوم

تقسیم و حل

Divide-and-Conquer

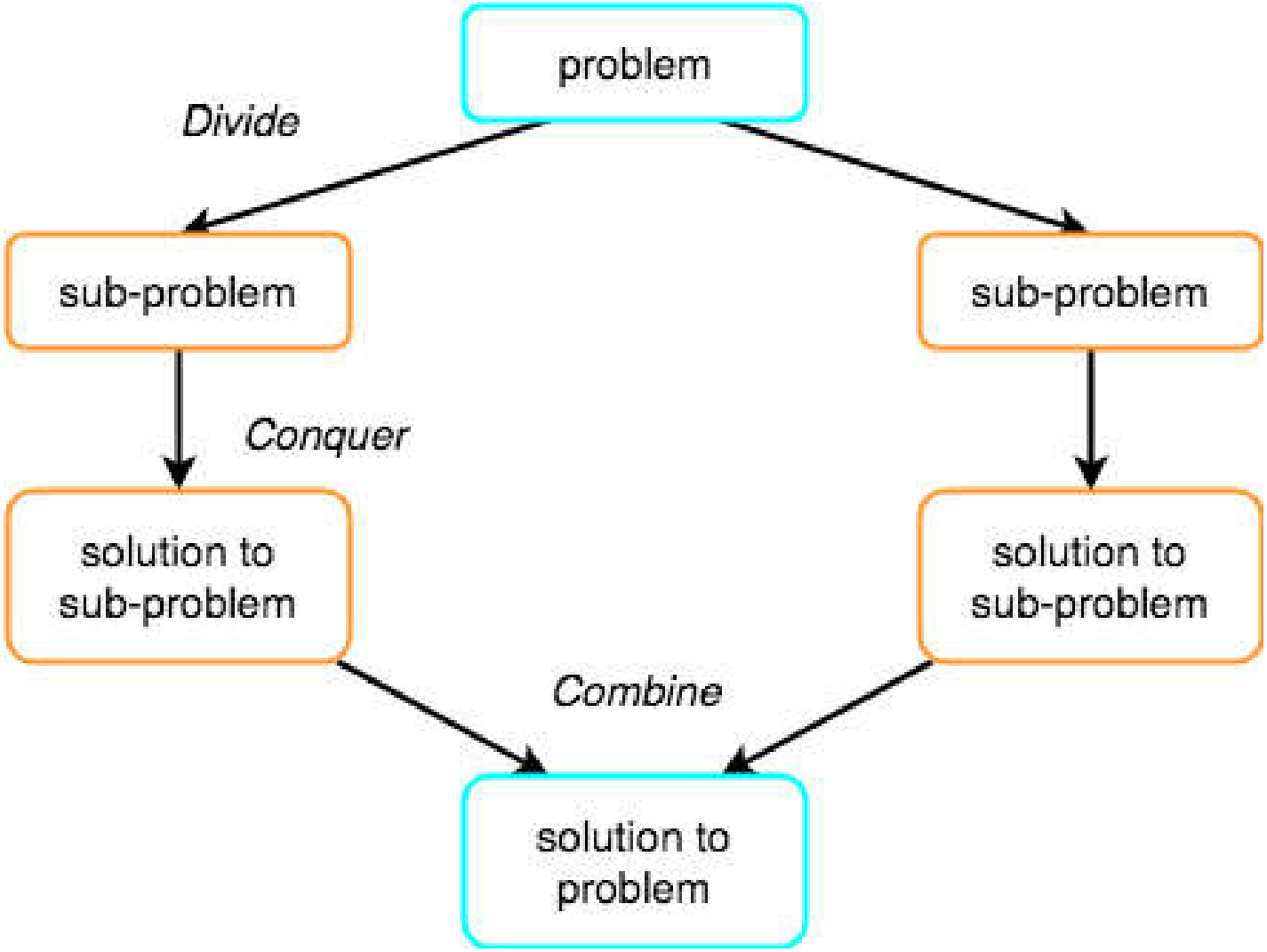
دانشکده مهندسی کامپیوتر - دانشگاه اصفهان
دکتر مرجان کائدی

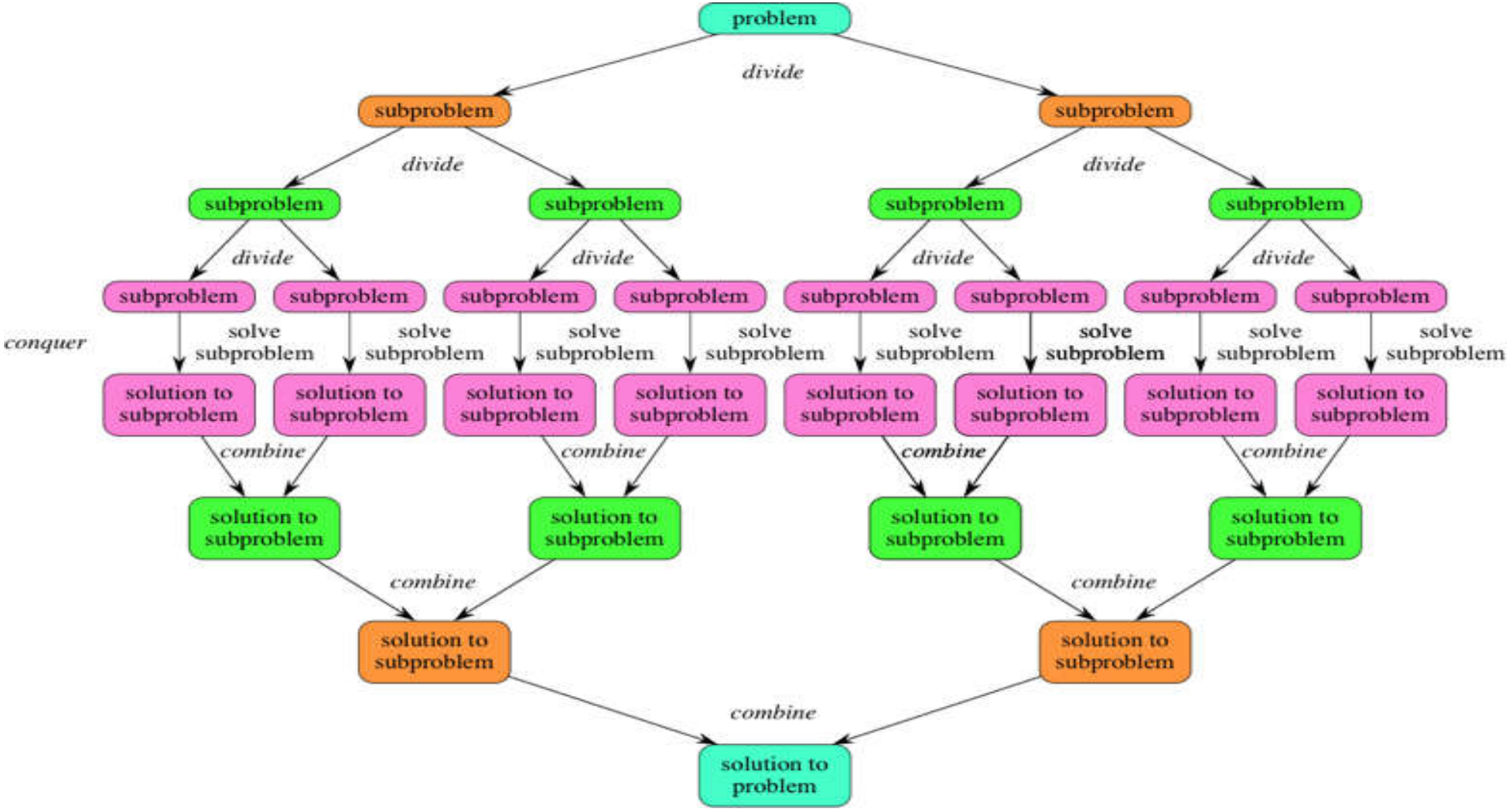
روش تقسیم و حل

- روش تقسیم و حل، نمونه ای از مساله را به دو یا چند نمونه کوچکتر تقسیم می کند.
- نمونه های کوچکتر معمولا نمونه هایی از مساله اصلی هستند.
- اگر حل نمونه های کوچک تر به دست آید، حل مساله اصلی با ترکیب این جواب ها به دست خواهد آمد.

- اگر نمونه های کوچکتر، باز هم کوچکتر از آن باشند که به راحتی حل شوند، می توان آنها را به نمونه های کوچکتری تقسیم کرد.
- این فرآیند تقسیم کردن ادامه می یابد تا به مساله هایی برسیم که به راحتی حل می شوند.

- روش تقسیم و حل یک روش بالا به پایین (top down) است.


problem
Divide
sub-problem
sub-problem
Conquer
solution to
sub-problem
solution to
sub-problem
Combine
solution to
problem


problem
divide
subproblem
subproblem
divide
divide
subproblem
subproblem
subproblem
subproblem
divide
divide
divide
divide
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
subproblem
conquer
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solve subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
combine
combine
combine
combine
solution to subproblem
solution to subproblem
solution to subproblem
solution to subproblem
combine
combine
solution to subproblem
solution to subproblem
combine
solution to problem

جستجوی دودویی

- جستجوی دودویی، ساده ترین نوع الگوریتم تقسیم و حل است. زیرا نمونه مساله تنها به یک نمونه کوچکتر شکسته می شود و بنابراین نیازی به ترکیب خروجی ها نیست.

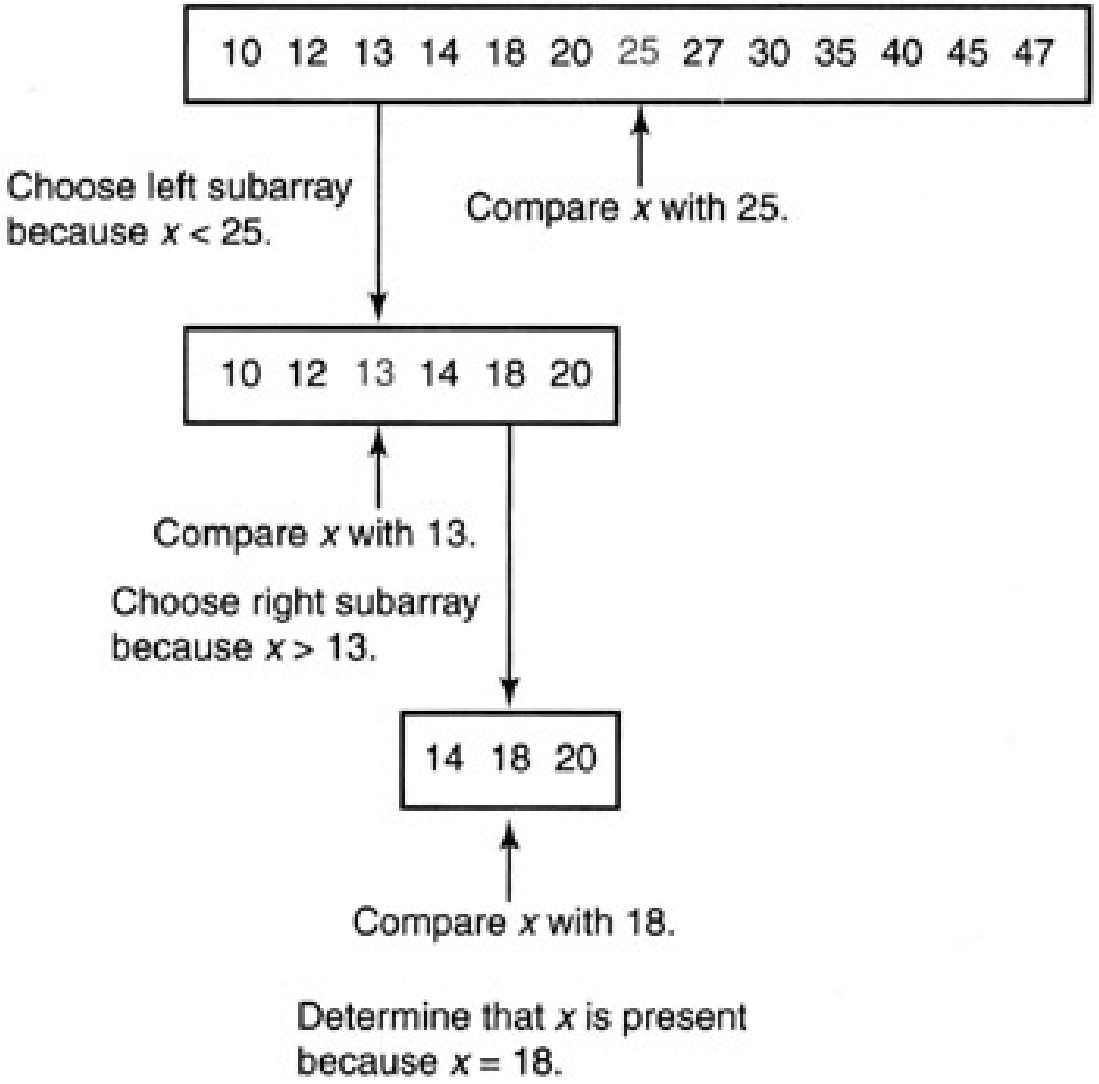
10 12 13 14 18 20 25 27 30 35 40 45 47.

↑

Middle item

10 12 13 14 18 20.

جستجوی دودویی



Problem: Determine whether x is in the sorted array S of size n.
Inputs: positive integer n, sorted (nondecreasing order) array of keys S indexed from 1 to n, a key x.
Outputs: location, the location of x in S (0 if x is not in S).

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);
  }
 }
```

- توجه شود که n و S و x پارامترهای تابع location نیستند
- بلکه چون بدون تغییر هستند، به صورت سراسری تعریف شده اند

- اولین فراخوانی:

$$locationout = location\ (1\ ,\ n)\ ;$$

جستجوی دودویی

- جستجوی دودویی، به صورت بازگشت انتهایی نوشته شده است (یعنی هیچ عملی پس از فراخوانی بازگشتی انجام نمی شود).
- بنابراین نوشتن نسخه غیربازگشتی آن کار را آسان می کند.

- مزیت نوشتن الگوریتم به صورت غیربازگشتی:
  - حذف پشته
  - صرفه جویی در حافظه و زمان

پیچیدگی زمانی در بدترین حالت برای جستجوی بازگشتی

$$W(n) = \underbrace{W\left(\frac{n}{2}\right)}_{\text{Comparisons in recursive call}} + \underbrace{1}_{\text{Comparison at top level}}$$

$$W(n) = W\left(\frac{n}{2}\right) + 1 \quad \text{for } n > 1,\ n \text{ a power of } 2$$
$$W(1) = 1$$

$$W(n) = \lg n + 1.$$

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

- عمل اصلی: مقایسه x با S[mid]
- اندازه ورودی: n یعنی تعداد عناصر آرایه
- اگر n به توانی از ۲ محدود نباشد:

توسعه رابطه بازگشتی اسلاید قبل

$$W(n) = W\left(\frac{n}{2}\right) + 1 \qquad \text{for } n > 1,\ n \text{ a power of } 2$$
$$W(1) = 1$$

$$w(n) = w(n/2) + 1$$
$$= [w(n/4)+1]+1 = w(n/2^2) + 2$$
$$= [w(n/8+1]+ 2 = w(n/2^3)+ 3$$
$$\ldots \quad = w(n/2^k)+k$$

روش کلی برای حل یک مساله به روش تقسیم و حل:

- راهی برای به دست آوردن حل یک نمونه از روی حل یک یا چند نمونه کوچکتر طراحی می کنیم.
- شرط (یا شرایط) پایانی نزدیک شدن به نمونه های کوچکتر را تعیین می کنیم.
- حل را در حالت (شرایط) پایانی تعیین می کنیم.

مرتب سازی ادغامی

Merge sort

مرتب سازی ادغامی

- برای مرتب سازی یک آرایه ۱۶ عضوی، می توان ابتدا آن را به دو آرایه ۸ عضوی تقسیم کرد.
- سپس دو زیرآرایه را مرتب کرده و با هم ادغام کنیم تا آرایه ۱۶ تایی مرتب شده به دست آید.
- برای مرتب کردن زیرآرایه های ۸ تایی نیز می توان همین رویه را در پیش گرفت و ....

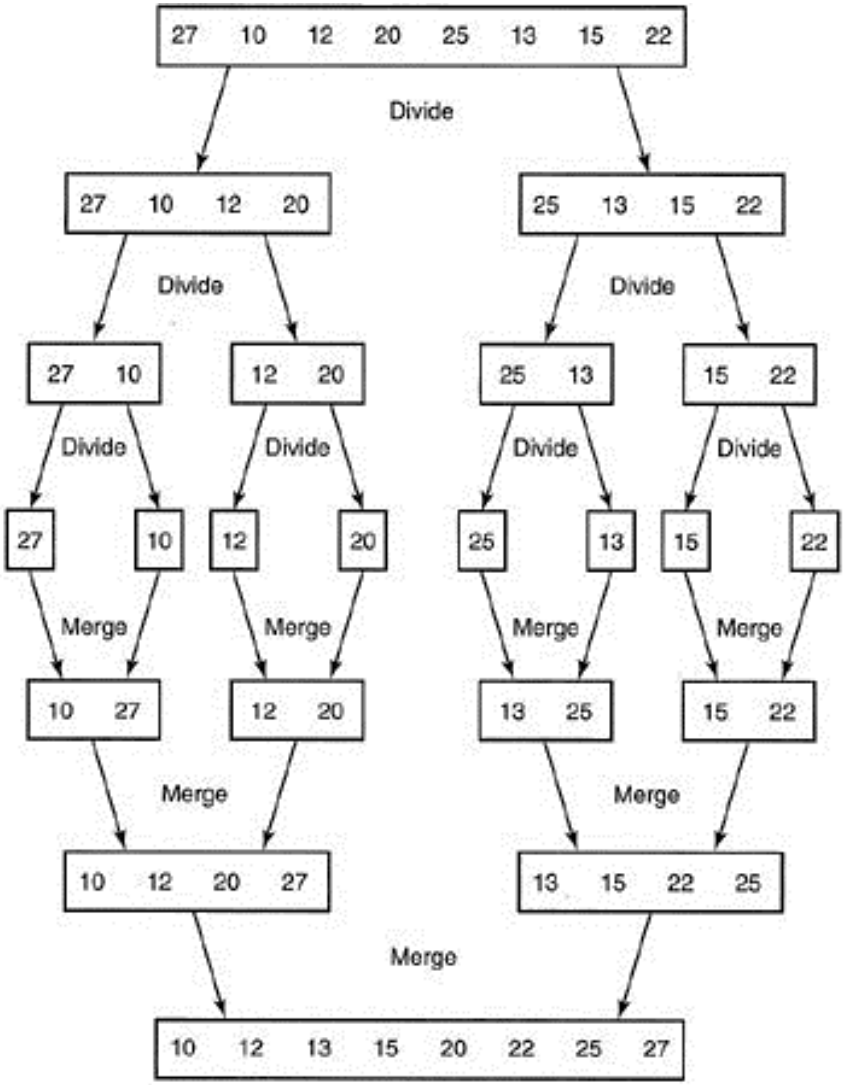
27 10 12 20 25 13 15 22.

27 10 12 20 and 25 13 15 22.

10 12 20 27 and 13 15 22 25.

10 12 13 15 20 22 25 27.

مرتب سازی ادغامی



ادغام دو آرایه مرتب

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.

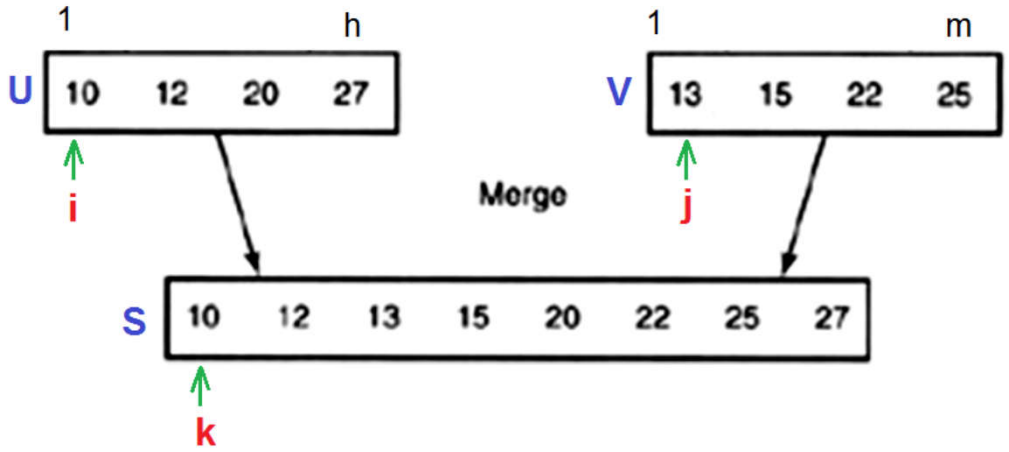
مرتب سازی ادغامی (غیرنزولی)

```
void mergesort (int n, keytype S[])
{
    if (n>1) {
        const int h=[n/2], m = n - h;
        keytype U[1 ..h], V[1 ..m];

        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);
    }
}
```

Problem: Merge two sorted arrays into one sorted array.
Inputs: positive integers h and m, array of sorted keys U indexed from 1 to h, array of sorted keys V indexed from 1 to m.
Outputs: an array S indexed from 1 to h + m containing the keys in U and V in a single sorted array.

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++;
      }
      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];
}
```



مثالی از ادغام دو آرایه U و V

Table 2.1: An example of merging two arrays *U* and *V* into one array *S*[*]

k	*U*	*V*	*S* (Resutl)
1	**10** 12 20 27	**13** 15 22 25	10
2	10 **12** 20 27	**13** 15 22 25	10 12
3	10 12 **20** 27	**13** 15 22 25	10 12 13
4	10 12 **20** 27	13 **15** 22 25	10 12 13 15
5	10 12 **20** 27	13 15 **22** 25	10 12 13 15 20
6	10 12 20 **27**	13 15 **22** 25	10 12 13 15 20 22
7	10 12 20 **27**	13 15 22 **25**	10 12 13 15 20 22 25
—	10 12 20 27	13 15 22 25	10 12 13 15 20 22 25 27 ← Final values

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

پیچیدگی زمانی در بدترین حالت برای الگوریتم ادغام

- عمل اصلی: مقایسه U[i] و V[i]

- اندازه ورودی: h و m (اندازه آرایه های ورودی)

- بدترین حالت زمانی است که یکی از اندیس ها به نقطه خروج خود رسیده است و اندیس دیگر به یک واحد کمتر از خروج خود رسیده است.

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

پیچیدگی زمانی در بدترین حالت برای الگوریتم مرتب سازی ادغامی

- عمل اصلی: مقایسه ای که در ادغام صورت می گیرد.
- اندازه ورودی: n یعنی تعداد اعضای آرایه S
- ابتدا حالتی را بررسی می کنیم که n توانی از ۲ است:

$$W(n) = \underbrace{W(h)}_{\text{Time to sort } U} + \underbrace{W(m)}_{\text{Time to sort } V} + \underbrace{h+m-1}_{\text{Time to merge}}$$

$$h = \lfloor n/2 \rfloor = \frac{n}{2}$$

$$m = n - h = n - \frac{n}{2} = \frac{n}{2}$$

$$h + m = \frac{n}{2} + \frac{n}{2} = n.$$

- پس:

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

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

پیچیدگی زمانی در بدترین حالت برای الگوریتم مرتب سازی ادغامی

- هنگامیکه اندازه ورودی ۱ باشد، شرط پایانی برآورده می شود و ادغامی صورت نمی گیرد.
- بنابراین:

$$W(n) = 2W\left(\frac{n}{2}\right) + n - 1 \qquad \text{for } n > 1,\ n \text{ a power of } 2$$
$$W(1) = 0$$

- حل رابطه بازگشتی:

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

- اگر n توانی از ۲ نباشد:

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

$$W(n) \in \Theta(n \lg n).$$

مرتب سازی درجا

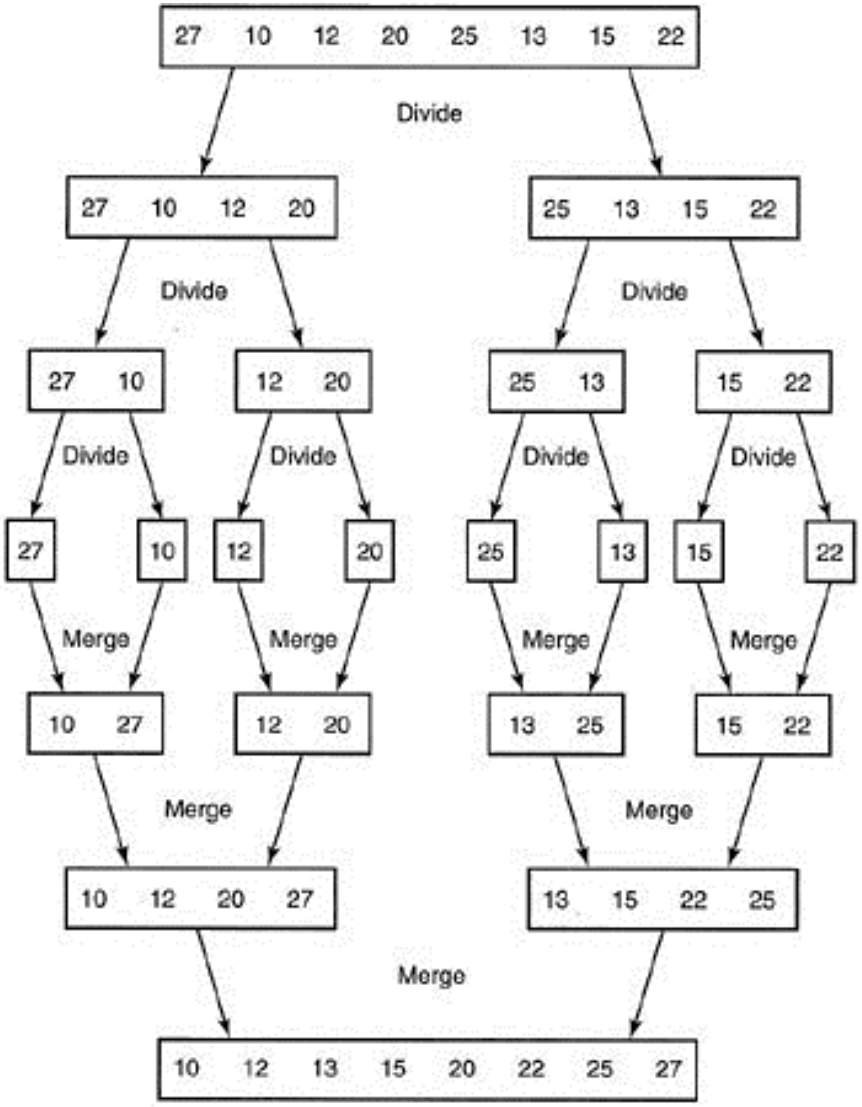
- مرتب سازی درجا، نوعی الگوریتم مرتب سازی است که فضایی بیشتر از آنچه مورد نیاز نگهداری ورودی است را استفاده نمی کند.

- الگوریتم مرتب سازی ادغامی که تا اینجا گفته شد، درجا نبود.
- زیرا با هر بار فراخوانی mergeSort، آرایه های جدید U و V ایجاد خواهند شد.

- تعداد کل عناصر آرایه های اضافی تولید شده، حدودا به صورت زیر است:

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

مرتب سازی ادغامی



الگوریتم مرتب سازی ادغامی ۲ (درجا)

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);
  }
}
```

- اولین فراخوانی: `mergesort2(1, n);`

الگوریتم ادغام ۲

```
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];
}
```

روش تقسیم و حل

- راهبرد تقسیم و حل شامل مراحل زیر است:

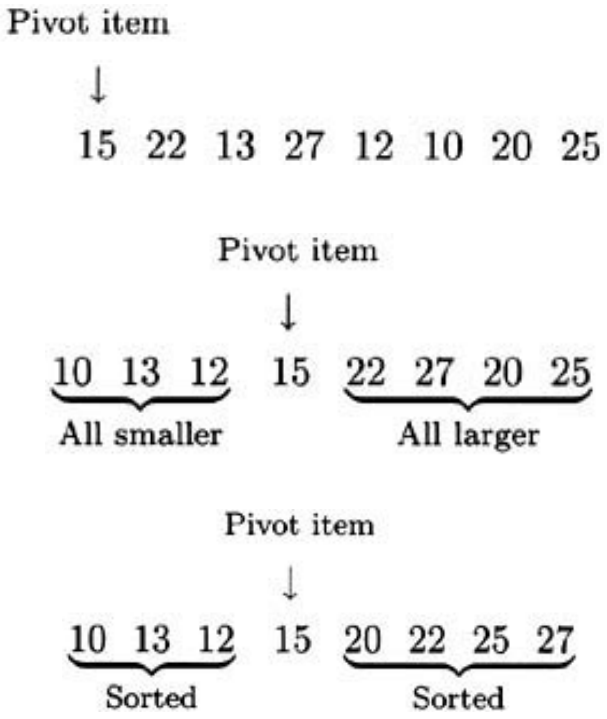
- تقسیم نمونه ای از مساله به یک یا چند نمونه کوچکتر
- حل هر نمونه کوچکتر. اگر نمونه های کوچکتر به اندازه کافی کوچک نبودند، برای حل آنها از بازگشت استفاده می کنیم.
- در صورت نیاز، حل نمونه های کوچکتر را ترکیب می کنیم تا حل نمونه اولیه به دست آید.

مرتب سازی سریع

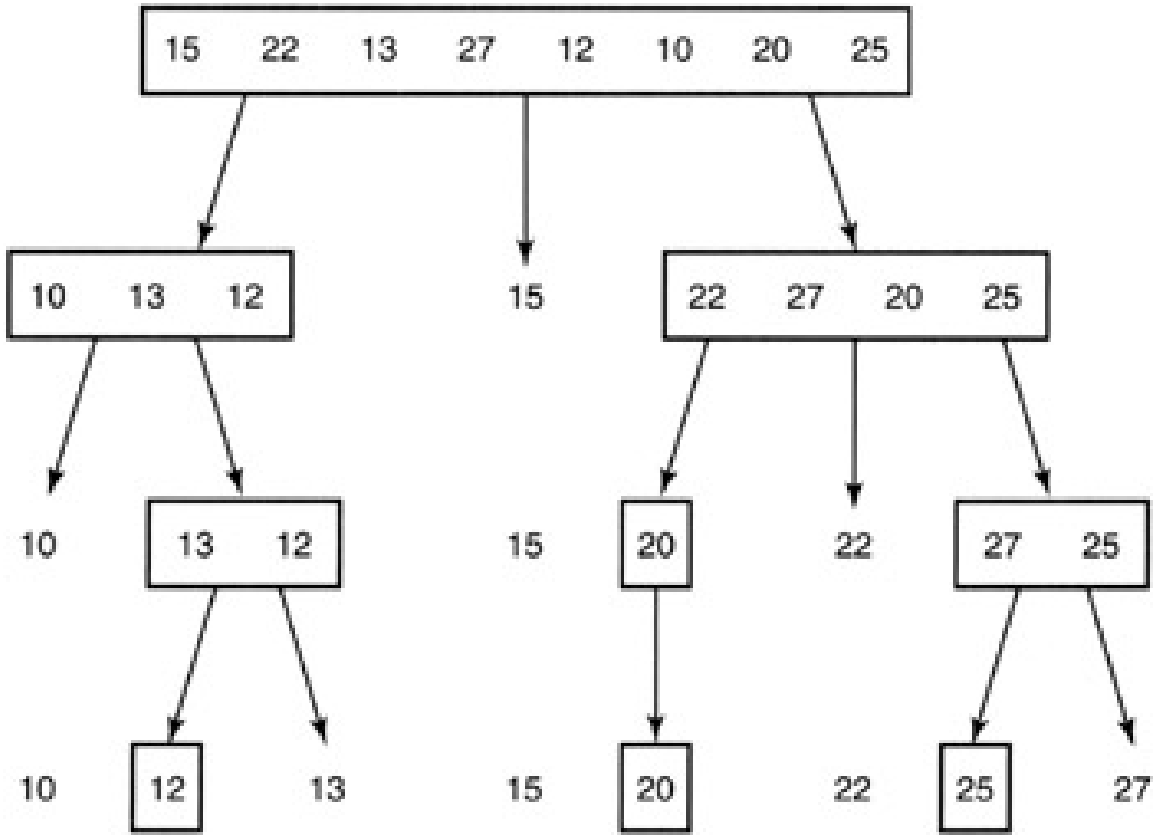
Quick sort

مرتب سازی سریع

- در این روش یک عنصر به عنوان عضو محور انتخاب می شود و عناصر کوچکتر از محور در یک آرایه و عناصر بزرگتر از محور در آرایه دیگر قرار می گیرند.
- این روال به صورت بازگشتی آنقدر ادامه می یابد تا به آرایه ای با یک عنصر برسیم.
- عنصر محوری می تواند هر عنصری باشد.
- برای سهولت، عضو اول را در نظر می گیریم.



مثالی از مرتب سازی سریع



الگوریتم مرتب سازی سریع

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);
  }
}
```

- اولین فراخوانی: quicksort(1, n);

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.

افراز آرایه

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

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

مثالی از روال افراز

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**	13	22	**27**	12	10	20	25	
5	3	**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	← Final values

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

تحلیل پیچیدگی الگوریتم افراز در حالت معمول

- عمل اصلی: مقایسه S[i] با pivotitem
- اندازه ورودی: تعداد عناصر موجود در آرایه یعنی $n = high - low + 1$

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

تحلیل پیچیدگی الگوریتم مرتب سازی سریع در بدترین حالت

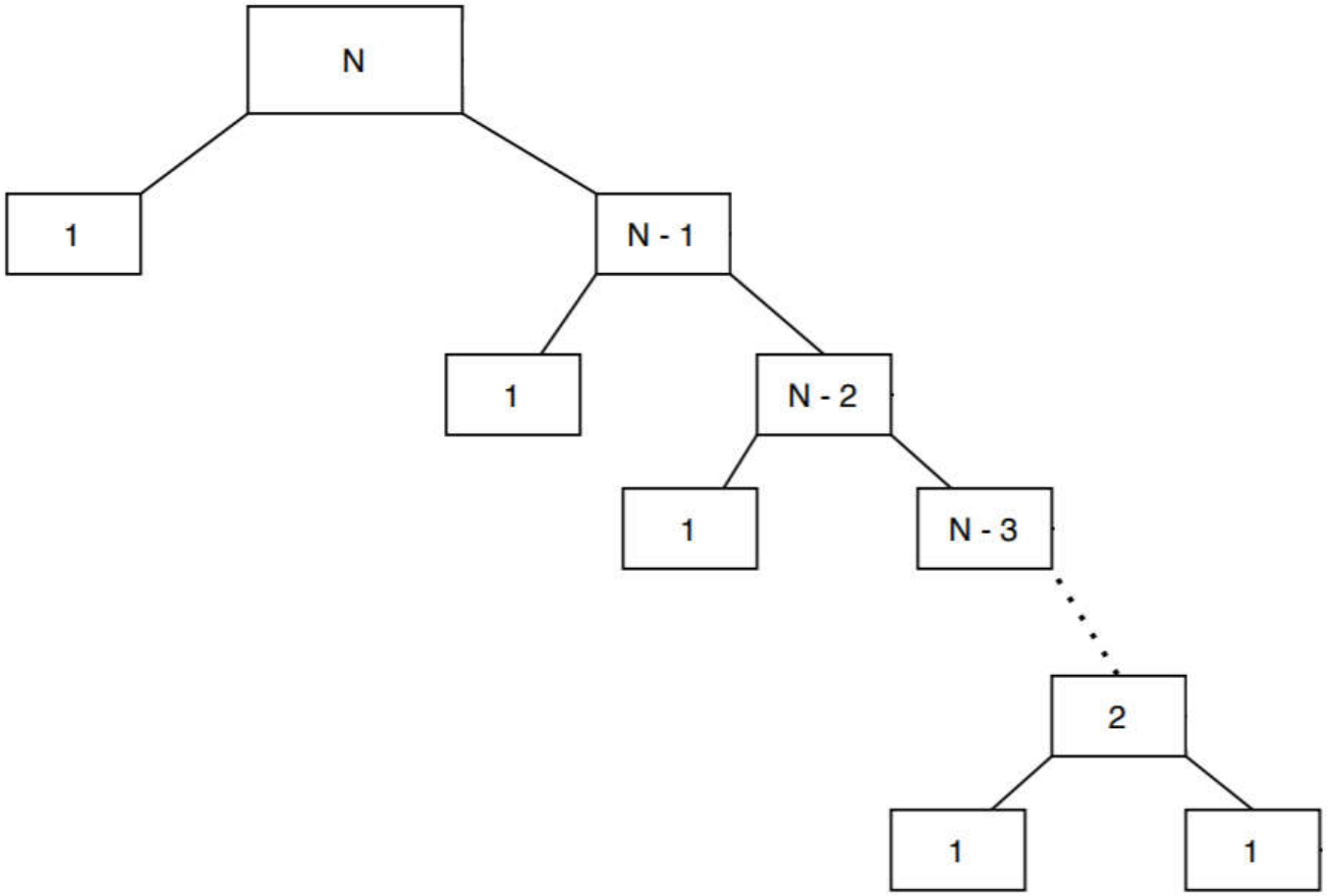
- بدترین حالت این الگوریتم هنگامی رخ می دهد که آرایه از قبل به ترتیب غیرنزولی مرتب شده باشد.
- عمل اصلی: مقایسه S[i] با pivotitem در روال partition
- اندازه ورودی: n یعنی تعداد عناصر آرایه S

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

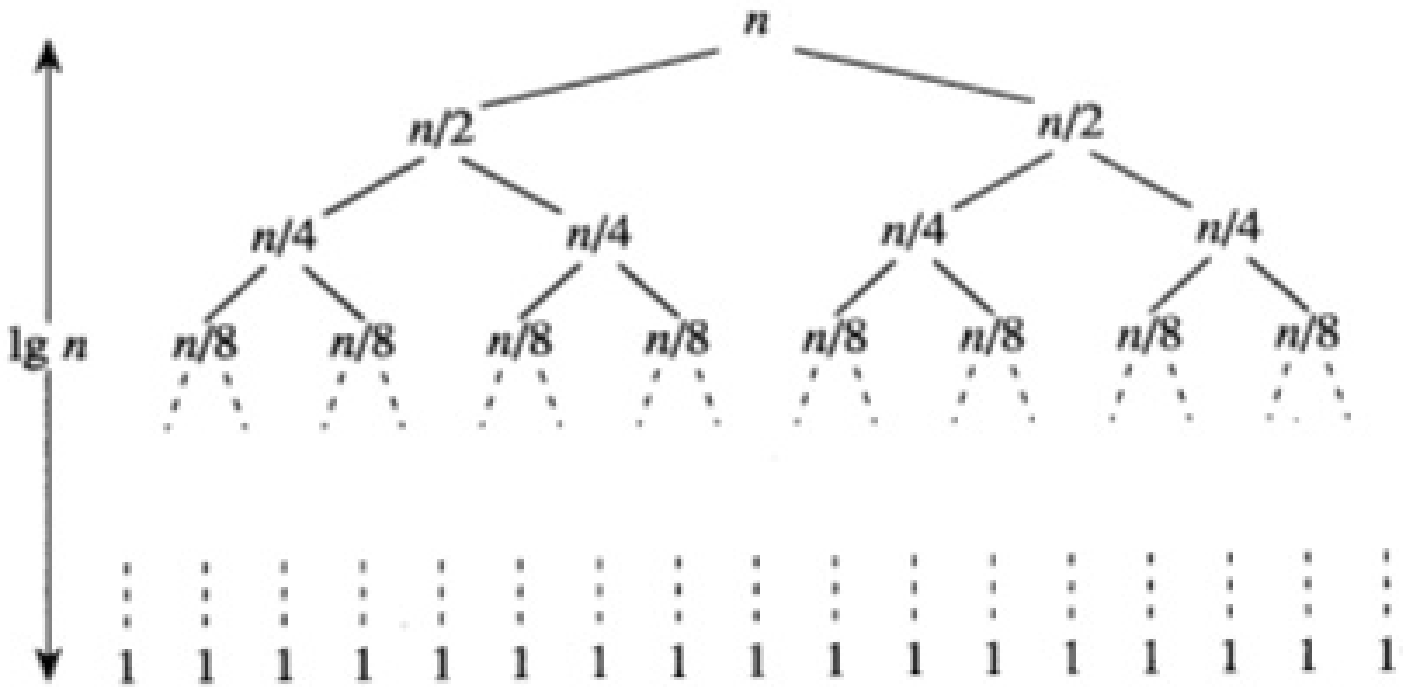
$$\begin{aligned} &T(n) = T(n-1) + n - 1 \qquad \text{for } n > 0 \\ &T(0) = 0. \end{aligned}$$

$$T(n) = \frac{n(n-1)}{2}.$$

الگوریتم مرتب سازی سریع در بدترین حالت



الگوریتم مرتب سازی سریع در بهترین حالت



انتخاب محور

- اگر بدانیم که آرایه تقریبا مرتب است، انتخاب اولین عنصر به عنوان محور، خوب نیست.
- روش های دیگر انتخاب عنصر محوری هم وجود دارند.
- در صورت استفاده از آنها، بدترین حالت در هنگامی که آرایه مرتب باشد رخ نمی دهد.
- ولی پیچیدگی بدترین حالت الگوریتم، هنوز هم $n(n-1)/2$ است.

تحلیل پیچیدگی زمانی مرتب سازی سریع در حالت میانگین

- عمل اصلی: مقایسه S[i] با pivotitem در روال partition
- اندازه ورودی: n یعنی تعداد عناصر آرایه S
- احتمال اینکه pivotpoint برگردانده شده توسط روال افراز، هر یک از اعداد ۱ تا n باشد، یکسان است.

$$A(n) = \sum_{p=1}^{n} \underbrace{\frac{1}{n}}_{\substack{\text{Probability} \\ \textit{pivotpoint} \text{ is } p}} \underbrace{[A(p-1) + A(n-p)]}_{\substack{\text{Average time to} \\ \text{sort subarrays when} \\ \textit{pivotpoint} \text{ is } p}} + \underbrace{n-1}_{\substack{\text{Time to} \\ \text{partition}}} \qquad (2.1)$$

- با بسط دادن رابطه بالا می توان داشت:

$$\sum_{p=1}^{n} [A(p-1) + A(n-p)] = 2 \sum_{p=1}^{n} A(p-1).$$

تحلیل پیچیدگی زمانی مرتب سازی سریع در حالت میانگین (ادامه)

$$A(n) = \frac{2}{n}\sum_{p=1}^{n} A(p-1) + n - 1.$$

- با ضرب کردن در n داریم:

$$nA(n) = 2\sum_{p=1}^{n} A(p-1) + n(n-1). \qquad (2.2)$$

- قرار دادن n-1 به جای n :

$$(n-1)A(n-1) = 2\sum_{p=1}^{n-1} A(p-1) + (n-1)(n-2). \qquad (2.3)$$

- تفریق دو معادله:

$$nA(n) - (n-1)A(n-1) = 2A(n-1) + 2(n-1),$$

$$\frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2(n-1)}{n(n+1)}.$$

تحلیل پیچیدگی زمانی مرتب سازی سریع در حالت میانگین (ادامه)

- اگر قرار دهیم:

$$a_n = \frac{A(n)}{n+1},$$

- این عبارت بازگشتی به دست می آید:

$$a_n = a_{n-1} + \frac{2(n-1)}{n(n+1)} \qquad \text{for } n > 0$$
$$a_0 = 0.$$

- حل عبارت بازگشتی:

$$a_n \approx 2\ln n,$$

- که به این معنا است:

$$A(n) \approx (n+1)\,2\ln n = (n+1)\,2\,(\ln 2)\,(\lg n)$$
$$\approx 1.38\,(n+1)\lg n \in \Theta(n \lg n).$$

- ضرب ماتریس ها به روش استراسن (Strassen)

ضرب ماتریس ها به روش استراسن

- در ضرب ماتریس ها به روش عادی (تکرارشونده)، پیچیدگی زمانی تعداد ضرب ها به صورت $T(n)=n^3$ است.

- پیچیدگی زمانی تعداد جمع ها به صورت $T(n)=n^3-n^2$ است.

این پیچیدگی ها هر دو از مرتبه زمانی $\Theta(n^3)$ هستند که چندان عملی نیست.

استراسن در سال ۱۹۶۹ الگوریتمی را منتشر کرد که پیچیدگی آن چه از لحاظ ضرب و چه از لحاظ جمع و تفریق بهتر از درجه ۳ است.

ضرب ماتریس ها به روش استراسن

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

- اگر فرض کنیم:

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

آنگاه:

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

ضرب ماتریس ها به روش استراسن

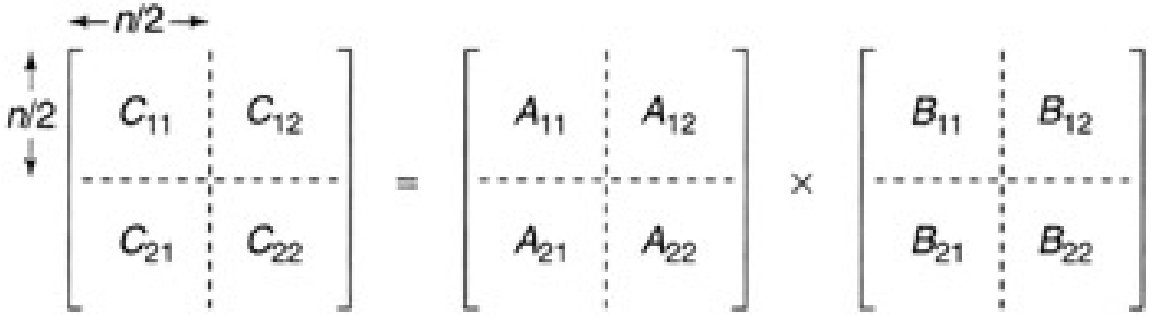
- روش استراسن برای ضرب دو ماتریس ۲ در ۲، به ۷ ضرب و ۱۸ جمع و تفریق نیاز دارد.
- ولی روش عادی به ۸ ضرب و ۴ جمع و تفریق نیاز دارد.

- پس یک عمل در ضرب در ازای ۱۴ عمل جمع و تفریق اضافی صرفه جویی می شود.

- این چندان جالب نیست و ضرب استراسن در مورد ضرب ماتریس های ۲ در ۲ ارزش چندانی ندارد.

- کاربی ضرب استراسن وقتی مشخص می شود که به ماتریس هایی با ابعاد بالاتر توسعه داده شود و ماتریس ها هر کدام به چهار زیرماتریس تقسیم شوند.

ضرب ماتریس ها به روش استراسن



$$A_{11} = \begin{bmatrix} a_{11} & a_{12} \cdots a_{1,n/2} \\ a_{21} & a_{22} \cdots a_{2,n/2} \\ & \vdots \\ a_{n/2,1} & \cdots a_{n/2,n/2} \end{bmatrix}.$$

- فرض کنیم که n توانی از ۲ باشد.

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

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

$$C_{11} = M_1 + M_4 - M_5 + M_7$$

- و به همین ترتیب تا آخر....

مثال

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

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

$$\begin{aligned} M_1 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ &= \left(\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix}\right) \times \left(\begin{bmatrix} 8 & 9 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix}\right) \\ &= \begin{bmatrix} 3 & 5 \\ 11 & 13 \end{bmatrix} \times \begin{bmatrix} 17 & 10 \\ 7 & 9 \end{bmatrix}. \end{aligned}$$

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

- هنگامی که ماتریس ها به قدر کافی کوچک باشند، آنها را به شیوه استاندارد ضرب می کنیم. در این مثال این کار را وقتی انجام می دهیم که n=2 باشد.

الگوریتم ضرب استراسن

Problem: Determine the product of two n x n matrices where n is a power of 2.
Inputs: an integer n that is a power of 2, and two n x 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)
  }
}
```

- مقدار threshold مقدار آستانه ای است که در آن احساس می شود استفاده از ضرب استاندارد بهتر از فراخوانی بازگشتی استراسن باشد.

تحلیل پیچیدگی تعداد ضرب های الگوریتم استراسن در حالت معمول

- عمل اصلی: یک ضرب ساده
- اندازه ورودی: n یعنی تعداد سطرها و ستون ها در ماتریس
- فرض می کنیم تا زمانی که به دو ماتریس ۱ در ۱ برسیم، کار ادامه می یابد.

$$T(n) = 7T\left(\frac{n}{2}\right) \qquad \text{for } n > 1,\ n \text{ a power of } 2$$
$$T(1) = 1.$$

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

تحلیل پیچیدگی تعداد جمع و تفریق های الگوریتم استراسن در حالت معمول

- عمل اصلی: یک جمع ساده
- اندازه ورودی: n یعنی تعداد سطرها و ستون ها

$$T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 \qquad \text{for } n > 1,\ n \text{ a power of } 2$$
$$T(1) = 0.$$

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

هنگامی که n توانی از ۲ نباشد ...

- هنگامی که n توانی از ۲ نباشد، باید الگوریتم را مقداری اصلاح کنیم.
- یک راه ساده اصلاح آن، افزودن تعداد کافی سطر و ستون صفر به ماتریس های اولیه است تا به ابعادی برسیم که توانی از ۲ هستند.

- به طریق دیگر، در فراخوانی های بازگشتی، هرگاه تعداد سطرها و ستون ها فرد شد، می توانیم فقط سطر و ستون اضافی صفر، اضافه کنیم.

مقایسه تعداد عملیات در ضرب استراسن و عادی

- با چشم پوشی از سربارهای ناشی از فراخوانی های بازگشتی، الگوریتم استراسن از لحاظ تعداد ضرب های انجام شده همواره کارایی بیشتری دارد و از لحاظ تعداد جمع و تفریق ها به ازای مقادیر بزرگ n کارایی آن بیشتر است.

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	$n^3 - n^2$	$6n^{2.81} - 6n^2$

- اعمال محاسباتی روی اعداد صحیح بزرگ

اعمال محاسباتی روی اعداد صحیح بزرگ

- می خواهیم اعمال محاسباتی را روی اعدد صحیحی انجام دهیم که بزرگتر از حدی هستند که سخت افزار کامپیوتر قادر به نمایش آنها باشد.

- فرض می کنیم اعداد در مبنای ۱۰ هستند ولی هر مبنای دیگر را نیز می توان در نظر گرفت.
- یک راه نمایش، استفاده از آرایه ای از اعداد صحیح است که هر محل آن یک رقم را نگهداری می کند.

$$\overset{5}{\underset{[6]}{}} \quad \overset{4}{\underset{S[5]}{}} \quad \overset{3}{\underset{S[4]}{}} \quad \overset{1}{\underset{S[3]}{}} \quad \overset{2}{\underset{S[2]}{}} \quad \overset{7}{\underset{S[1]}{}}.$$

- به راحتی می توان الگوریتم هایی با زمان خطی نوشت که عملیات زیر را انجام دهند:

$$u \times 10^m \qquad u \texttt{ divide } 10^m \qquad u \texttt{ rem } 10^m,$$

اعمال محاسباتی روی اعداد صحیح بزرگ

- یک الگوریتم با زمان درجه دو، برای ضرب اعداد صحیح بزرگ

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

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

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

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

$$m = \left\lfloor \frac{n}{2} \right\rfloor.$$

$$\begin{aligned} uv &= (x \times 10^m + y)(w \times 10^m + z) \\ &= xw \times 10^{2m} + (xz + wy) \times 10^m + yz. \end{aligned}$$

- می توانیم U و V را با چهار عمل ضرب روی اعداد صحیح با حدود نیمی از ارقام و به همراه اجرای عملیات با زمان خطی در هم ضرب کنیم.

مثال

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

- این فرآیند تقسیم آنقدر ادامه می یابد تا به یک مقدار آستانه برسیم که در آن، عمل ضرب از طریق استاندارد انجام می شود.

Problem: Multiply two large integers, u and v.
Inputs: large integers u and v.
Outputs: prod, the product of u and v.

الگوریتم ضرب اعداد صحیح بزرگ

```
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) x 10^2m + (prod(x,z)+prod(w,y)) x 10^m + prod(y,z);
 }
}
```

$$uv = (x \times 10^m + y)(w \times 10^m + z)$$
$$= xw \times 10^{2m} + (xz + wy) \times 10^m + yz.$$

- عمل های devide و rem و × نمایانگر توابع با زمان خطی هستند.

تحلیل پیچیدگی زمانی الگوریتم ضرب اعداد صحیح بزرگ در بدترین حالت

- عمل اصلی: دستکاری یک رقم دهدهی در یک عدد صحیح بزرگ در هنگام جمع کردن، تفریق کردن، یا انجام devide 10^m و $rem\ 10^m$ و $\times 10^m$.
- هر یک از عملیات 10^m و $rem\ 10^m$ و $\times 10^m$، عمل اصلی را از مرتبه n بار انجام می دهند.
- اندازه ورودی: n یعنی تعداد ارقام هر یک از دو عدد صحیح
- همه عملیات خطی را در یک جمله cn گروه بندی می کنیم که در آن، c یک ثابت مثبت است.
- بدترین حالت، زمانی است که هر دو عدد صحیح هیچ رقم صفری نداشته باشند. زیرا بازگشت، فقط هنگامی به پایان می رسد که به حد آستانه برسیم.
- مقدار واقعی S که در آن نمونه دیگر تقسیم نمی شود، کوچکتر یا مساوی مقدار آستانه و توانی از ۲ است.
- فرض شده اندازه ورودی ها توانی از ۲ هستند.

$$W(n) = 4W\left(\frac{n}{2}\right) + cn \qquad \text{for } n > s,\ n \text{ a power of } 2$$
$$W(s) = 0.$$

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

بهبود الگوریتم

- الگوریتمی که برای ضرب اعداد صحیح بزرگ ذکر شد، هنوز از درجه دوم است.
- مشکل اینجا است که این الگوریتم چهار عمل ضرب را روی اعداد صحیح با نیمی از ارقام اولیه انجام می دهد.
- اگر این تعداد ضرب را کاهش دهیم می توانیم الگوریتم بهتری به دست آوریم.

بهبود الگوریتم

- در الگوریتم قبل، باید موارد زیر را محاسبه کنیم:

$$xw, \quad xz + yw, \quad \text{and} \quad yz,$$

- و این کار را با چهار بار فراخوانی تابع prod برای موارد زیر انجام دادیم:

$$xw, \quad xz, \quad yw, \quad \text{and} \quad yz,$$

- اگر به جای آن، قرار دهیم:

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

- خواهیم داشت:

- و

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

- فقط سه عمل ضرب و چند جمع و تفریق اضافی $r = (x + y)(w + z), \quad xw, \quad \text{and} \quad yz.$

Problem: Multiply two large integers, u and v.
Inputs: large integers u and v.
Outputs: prod2, the product of u and v.

الگوریتم ضرب اعداد بزرگ صحیح 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 = 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;
  }
}
```

در نسخه قبلی الگوریتم:

```
return prod (x,w) x 10^2m + (prod(x,z)+prod(w,y)) x 10^m + prod(y,z);
```

تحلیل پیچیدگی زمانی الگوریتم ضرب اعداد صحیح بزرگ ۲ در بدترین حالت

- عمل اصلی: دستکاری یک رقم دهدهی در یک عدد صحیح بزرگ در هنگام جمع کردن، تفریق کردن، یا انجام devide 10^m و rem 10^m و $10^m \times$. هر یک از سه فراخوانی اخیر، عمل اصلی را از مرتبه n بار انجام می دهند.
- اندازه ورودی: n یعنی تعداد ارقام هر یک از دو عدد صحیح
- بدترین حالت، زمانی است که هر دوعدد صحیح هیچ رقم صفری نداشته باشند. زیرا بازگشت، فقط هنگامی به پایان می رسد که به حد آستانه برسیم.
- اگر n توانی از ۲ باشد، آنگاه x و y و w و z همگی n/2 رقم دارند. یعنی داریم:

$$\frac{n}{2} \leq \text{digits in } x+y \leq \frac{n}{2}+1.$$
$$\frac{n}{2} \leq \text{digits in } w+z \leq \frac{n}{2}+1.$$

مثال هایی از اعداد ارقام x+y در الگوریتم

Table 2.4: Examples of the number of digits in x + y in Algorithm 2.10

n	*x*	*y*	*x* + *z*	Number of Digits in *x* +*y*
4	10	10	20	$2 = \frac{n}{2}$
4	99	99	198	$3 = \frac{n}{2} + 1$
8	1000	1000	2000	$4 = \frac{n}{2}$
8	9999	9999	19,998	$5 = \frac{n}{2} + 1$

تحلیل پیچیدگی زمانی الگوریتم ضرب اعداد صحیح بزرگ ۲ در بدترین حالت (ادامه)

- اندازه ورودی ها برای فراخوان های توابع:

	Input Size
$prod2(x+y, w+z)$	$\frac{n}{2} \leq$ input size $\leq \frac{n}{2}+1$
$prod2(x, w)$	$\frac{n}{2}$
$prod2(y, z)$	$\frac{n}{2}$

- عملیات جمع، تفریق، devide 10^m و rem 10^m و $\times 10^m$ همگی دارای زمانی خطی بر حسب n هستند. بنابراین:

$$3W\left(\frac{n}{2}\right)+cn \leq W(n) \leq 3W\left(\frac{n}{2}+1\right)+cn \quad \text{for } n>s, n \text{ a power of } 2$$
$$W(s)=0,$$

- که s کوچکتر مساوی آستانه و توانی از ۲ است.

$$W(n) \in \Omega\left(n^{\log_2 3}\right).$$

تحلیل پیچیدگی زمانی الگوریتم ضرب اعداد صحیح بزرگ ۲ در بدترین حالت (ادامه)

- حال نشان می دهیم که:

$$W(n) \in O\left(n^{\log_2 3}\right).$$

$$W'(n) = W(n+2).$$

- فرض می کنیم که:
- با استفاده از نامساوی طرف راست بازگشتی داریم:

$$\begin{aligned} W'(n) = W(n+2) &\le 3W\left(\frac{n+2}{2}+1\right) + c[n+2] \\ &\le 3W\left(\frac{n}{2}+2\right) + cn + 2c \\ &\le 3W'\left(\frac{n}{2}\right) + cn + 2c. \end{aligned}$$

- چون W(n) غیرنزولی است،W′(n) هم غیرنزولی است. پس داریم:

$$W'(n) \in O\left(n^{\log_2 3}\right),$$

$$W(n) = W'(n-2) \in O\left(n^{\log_2 3}\right). \quad \Rightarrow \quad W(n) \in \Theta\left(n^{\log_2 3}\right) \approx \Theta\left(n^{1.58}\right).$$

- تعیین مقادیر آستانه

تعیین مقادیر آستانه

- سوال: الگوریتم به ازای چه مقادیری از n آنقدر سریع است که دیگر نیاز به تقسیم کردن نمونه نباشد؟

- این مقادیر به الگوریتم تقسیم و حل، الگوریتم جانشین، و کامپیوتری که این الگوریتم ها روی آن اجرا می شوند بستگی دارند.

- به طور ایده آل می خواهیم یک مقدار آستانه بهینه از n را بیابیم.

- البته آستانه بهینه همواره وجود ندارد.

مثال ۱: تعیین مقدار آستانه برای مرتب سازی ادغامی جهت فراخوانی مرتب سازی تعویضی (ادامه)

- پیچیدگی مرتب سازی ادغامی را درنظر بگیرید:

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

- فرض کنید روی کامپیوتر مورد نظر، زمانی که مرتب سازی ادغامی ۲ برای تقسیم و ترکیب دوباره نمونه به اندازه n نیاز دارد، 32n میکروثانیه است. یعنی:

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

- چون هنگامی که n=1 است فقط شرط نهایی چک می شود، فرض می کنیم W(1) صفر است. فرض می کنیم n توانی از ۲ است.

$$W(n) = 2W(n/2) + 32n\ \mu s \qquad \text{for } n > 1,\ n \text{ a power of } 2$$
$$W(1) = 0\ \mu s.$$

- حل معادله:

$$W(n) = 32n \lg n\ \mu s.$$

مثال ۱: تعیین مقدار آستانه برای مرتب سازی ادغامی جهت فراخوانی مرتب سازی تعویضی(ادامه)

- فرض می کنیم روی همان کامپیوتر، مرتب سازی تعویضی به زمان زیر برای مرتب سازی آرایه n تایی نیاز دارد:

$$\frac{n\,(n-1)}{2}\mu s$$

- به دنبال نقطه ای هستیم که در مرتب سازی ادغامی ۲ باید مرتب سازی تعویضی را فراخواند.
- **یک راه حل نادرست:**

$$\frac{n\,(n-1)}{2}\mu s < 32n \lg n\ \mu s.$$

- جواب:

$$n < 591.$$

مثال ۱: تعیین مقدار آستانه برای مرتب سازی ادغامی جهت فراخوانی مرتب سازی تعویضی

- **راه حل درست:** فرض کنید در مرتب سازی ادغامی ۲ وقتی n≤t شود، مرتب سازی تعویضی فراخوانی می شود:

$$W(n) = \begin{cases} \frac{n(n-1)}{2}\mu s & \text{for } n \le t \\ W\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + W\left(\left\lceil \frac{n}{2} \right\rceil\right) + 32n\ \mu s & \text{for } n > t \end{cases} \qquad (2.5)$$

- برای تعیین مقدار بهینه t باید معادله زیر را حل کنیم:

$$W\left(\left\lfloor \frac{t}{2} \right\rfloor\right) + W\left(\left\lceil \frac{t}{2} \right\rceil\right) + 32t = \frac{t(t-1)}{2}. \qquad (2.6)$$

- چون سقف و کف t/2 هر دو کوچکتر مساوی tهستند، داریم:

$$W\left(\left\lfloor \frac{t}{2} \right\rfloor\right) = \frac{\lfloor t/2 \rfloor (\lfloor t/2 \rfloor - 1)}{2} \quad \text{and} \quad W\left(\left\lceil \frac{t}{2} \right\rceil\right) = \frac{\lceil t/2 \rceil (\lceil t/2 \rceil - 1)}{2}.$$

مثال ۱: تعیین مقدار آستانه برای مرتب سازی ادغامی جهت فراخوانی مرتب سازی تعویضی (ادامه)

- با قرار دادن تساوی ها در معادله (۲-۶) داریم:

$$\frac{\lfloor t/2 \rfloor (\lfloor t/2 \rfloor - 1)}{2} + \frac{\lceil t/2 \rceil (\lceil t/2 \rceil - 1)}{2} + 32t = \frac{t(t-1)}{2}. \qquad (2.7)$$

- اگر یک مقدار زوج به جای t قرار دهیم از حل معادله داریم:

$$t = 128.$$

- اگر یک مقدار زوج به جای t قرار دهیم از حل معادله داریم:

$$t = 128.008.$$

- بنابراین مقدار آستانه بهینه ۱۲۸ است.

مثال ۲:

- فرض کنید برای یک الگوریتم تقسیم و حل روی کامپیوتر خاصی رابطه زیر برقرار است:

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

- که در آن، 16n میکروثانیه زمان تقسیم و ادغام دوباره نمونه ای به اندازه n است.
- فرض کنید روی همان کامپیوتر، یک الگوریتم تکرارشونده، برای پردازش نمونه ای به اندازه n، زمان n^2 میکروثانیه را صرف می کند. برای تعیین مقدار t که باید در آن، الگوریتم تکراری را فراخوانی کنیم باید معادله زیر حل شود:

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

- چون سقف t/2 کوچکتر مساوی t است داریم:

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

- اگر مقدار زوج به جای t قرار دهیم و آن را حل کنیم: $t = 64.$
- اگر مقدار فرد به جای t قرار دهیم و آن را حل کنیم: $t = 70.04.$

اندازه های گوناگون نمونه که نشان می دهند آستانه به ازای n های زوج برابر با ۶۴ و برای n های فرد برابر با ۷۰ است. یعنی آستانه بهینه واحدی وجود ندارد.

Table 2.5: Various instance sizes illustrating that the threshold is 64 for *n* even and 70 for *n* odd in Example 2.8

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

کجا نمی توان از روش تقسیم و حل استفاده کرد؟

- در موارد زیر باید از روش تقسیم و حل پرهیز کرد:

- نمونه ای با اندازه n به دو یا چند نمونه تقسیم می شود که اندازه آنها نیز تقریبا n است (در این حالت یک الگوریتم با زمان نمایی خواهیم داشت). مثل فیبوناچی.
- نمونه ای با اندازه n تقریبا به n نمونه با اندازه n/c تقسیم می شود که c یک مقدار ثابت است (در این حالت یک الگوریتم به پیچیدگی $\Theta(n^{\lg n})$ خواهیم داشت).

- از طرفی گاهی مساله ذاتا نیاز به حالت نمایی دارد و در چنین شرایطی نیاز نیست که از روش تقسیم و حل پرهیز کنیم.

- پایان فصل دوم