

فصل دوم:

روش تقسیم و حل

نالپئون بنپارت



Napoleon Bonaparte

- Ability is nothing without opportunity
- A leader is a dealer in hope.
- Glory is fleeting, but obscurity is forever.
- I can no longer obey; I have tasted command, and I cannot give it up.
- I am sometimes a fox and sometimes a lion. The whole secret of government lies in knowing when to be the one or the other
- If you wish to be a success in the world, promise everything, deliver nothing.
- Let the path be open to talent.
- Never interrupt your enemy when he is making a mistake.
- There is no such thing as accident; it is fate misnamed.

- روش تقسیم و حل یک روش بالا به پایین است. یک نمونه از مسئله را به دو یا چند نمونه کوچکتر تقسیم می کنند به طوری که همین نمونه های کوچکتر هم نمونه های از مسئله اصلی هستند. اگر حل نمونه های کوچکتر به راحتی صورت گیرد نمونه اصلی با ترکیب این حل ها به دست می آید.
- حل یک نمونه سطح بالای مسئله با رفتن به جزء و بدست آوردن حل نمونه های کوچکتر و ترکیب آنها حاصل می شود.
- مانند الگوریتم های بازگشتی که در سطح حل مسئله فکر می شود و اجرا آن را بر عهده سیستم (با استفاده از پشته سیستم) می گذارند.

- هنگام پی ریزی یک الگوریتم بازگشتی ، باید:
 - ۱- راهی برای به دست آوردن حل یک نمونه از روی 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.

الگوریتم انتزاعی برای روش تقسیم و غلبه

Algorithm D&C (low, high)

if small(low , high) then return G(low, high)

else

{

mid = Divide (low, high)

return combine(D&C(low, high),D&C(mid+1, high))

}

Small(low, high) یک تابع بولی است که مشخص می کند آیا اندازه مساله به اندازه کافی کوچک هست که بتوان آن را بدون شکستن حل کرد.

اگر زمان اجرای الگوریتم combine در فراخوانی $D\&C(1,n)$ فرض کنیم $f(n)$ را برابر

برای n های بزرگ $T(n)=t([n/2]) + t([n/2]) + f(n) \Rightarrow t(n)=2t(n/2) + f(n)$

برای n های کوچک $T(n)=G(n)$

جست و جوی دودویی

Suppose $x = 18$ and we have the following array:

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

1. Divide the array: Because $x < 25$, we need to search

10 12 13 14 18 20.

2. Conquer the subarray by determining whether x is in the subarray. (This is accomplished by recursively dividing the subarray):

Yes, x is in the subarray.

3. Obtain the solution to the array from the solution to the subarray:

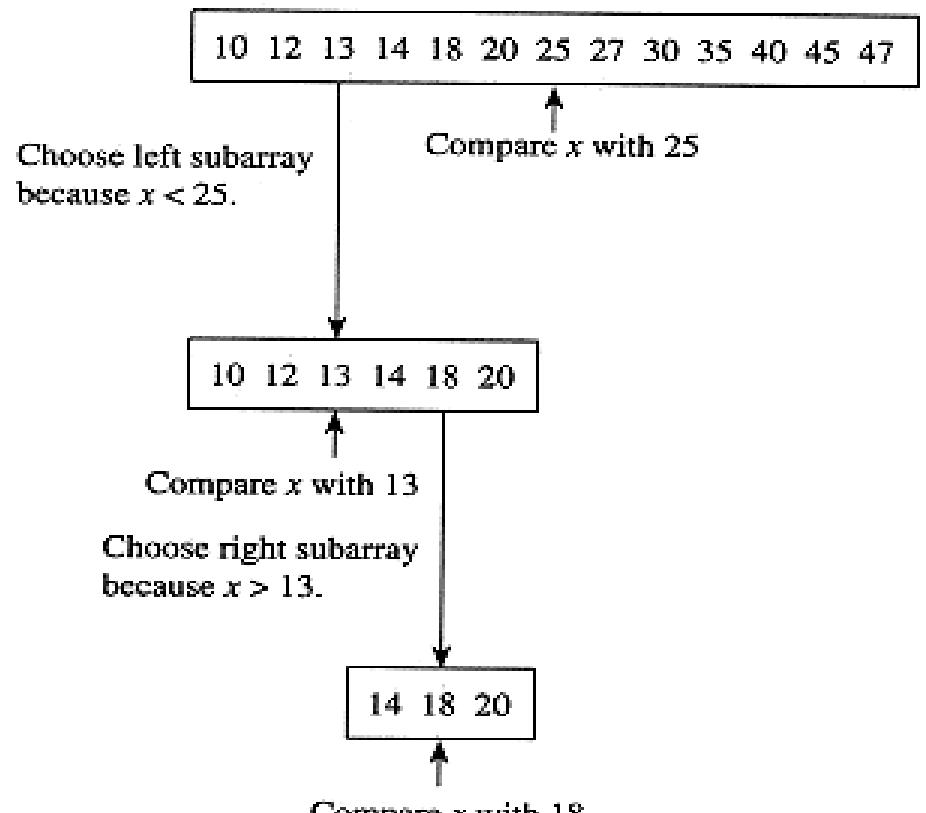
Yes, x is in the array.

- هنگام پی ریزی یک الگوریتم بازگشتی ، باید:
 - ۱- راهی برای به دست آوردن حل یک نمونه از روی 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.

الگوریتم ۱-۲: جست و جوی دودویی (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);
    }
}
```

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



1-location(1,14), mid=7

2-location(1,6), mid=3

3-location(4,6), mid=5

X=18 •

Determine that x is present
because $x = 18$.

```
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);
    }
}
```

1-location(1,14), mid=7
2-location(1,6), mid=3
3-location(4,6), mid=5
4-location(6,6), mid=6
5-location(6,5), x
X=19 •

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

آرگومان های تابع

- آرگومانهای تکراری ارسال نمی شود.
- روش های موجود
- تعریف متغیر های S و X و n به صورت عمومی
- استفاده از کلاس ها
- ارسال توسط آدرس
- ...

Location(1,n)

بازگشتی-----تکراری

```
void binsearch (int n,
                const keytype S[],
                keytype x,
                index& location)
{
    index low, high, mid;

    low = 1; high = n;
    location = 0;
    while (low <= high && location == 0) {
        ;
        if (x == S[mid])
            location = mid;
        else if (x < S[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
}
```

```
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);
    }
}
```

تحلیل پیچیدگی زمانی در بدترین حالت برای الگوریتم جست و جوی دودویی 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);
    }
}
```

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

بدترین حالت: X از همه عناصر بزرگتر و n توانی از ۲
اندازه: ۱-۲-۴-۸-۱۶

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

برای $n > 1$ ، n توانی از ۲ است

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

$$\times \times \times \times \times W(1) = 1$$

اگر n توانی از ۲ نباشد:

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

	بهترین حالت	حالت میانگین	بدترین حالت
جست و جوی موفق	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\log n)$
جست و جوی ناموفق	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$

نوعی از الگوریتم جست و جوی دودویی که به یک مقایسه نیاز دارد

index location (A, low, high, x)

{

if (low < high)

{

mid = $\lfloor (low + high) / 2 \rfloor$;

if (x < A [mid]) then return location (A, low , mid, x)

else

 return location (A, mid, high, x)

}

else

 if (x= A[low]) then return low

 else return 0

}

بزرگترین و کوچکترین عنصر یک آرایه

- مسئله: یافتن بزرگترین و کوچکترین عنصر یک آرایه n عضوی

Algorithm simple-max-min (A, n, max, min)

max=min=A[1]

for $i=2$ to n do

```
{ if ( A[i]> max) then max=A[i]  
    if ( A[i]< min) then min=A[i]  
}
```

زمان اجرا در همه حالات $2(n-1)$

اگر داشته باشیم:

if (A[i]> max) then max=A[i]

else if (A[i]< min) then min=A[i]

در بهترین حالت (آرایه صعودی) $n-1$ و در بدترین حالت (نزولی) $2(n-1)$

Algorithm D&C (low, high)

```
if small(low , high) then return G(low, high)
else{mid = Divide (low, high)
      return combine( D&C(low, high),D&C(mid+1, high))}
```

یک تابع بولی است که مشخص می کند آیا اندازه مساله به اندازه کافی کوچک هست که بتوان آن را بدون شکستن حل کرد.

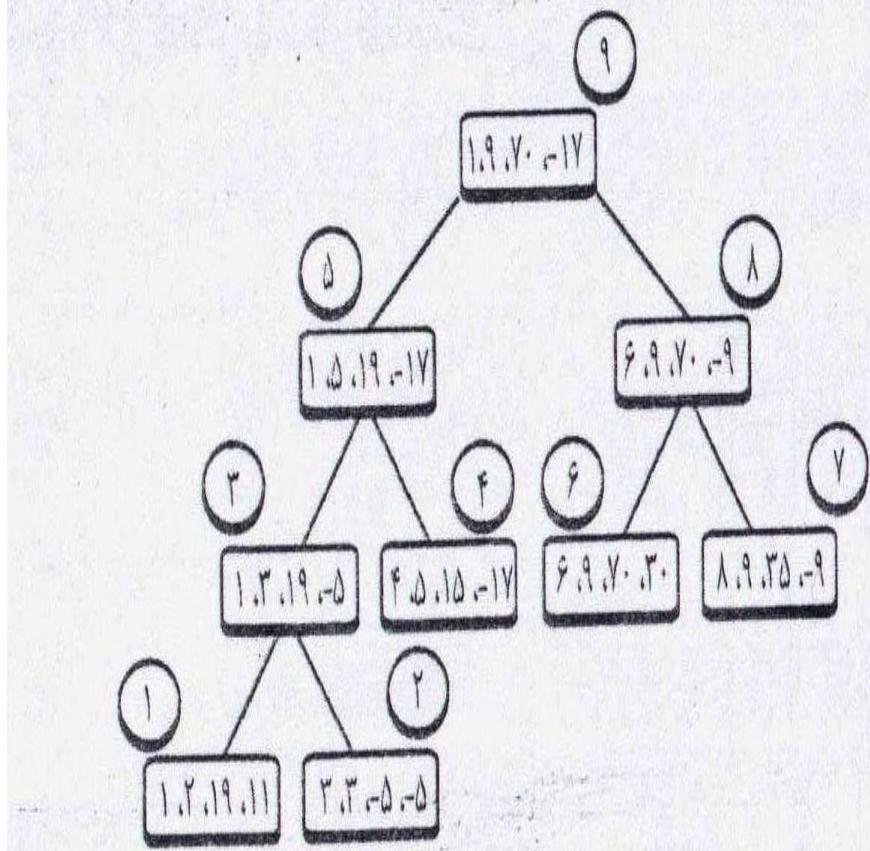
- اگر $n=1$ باشد:
- اگر $n=2$ باشد:

```

Algorithm Find _ Max _ Min (low, high, fmax, fmin)
if low = high then fmax = fmin = A[low]
    else if high = low + 1 then
    {
        if A[low] < A[high] then
        {
            fmax = A[high] ; fmin = A[low]
        }
        else
        {
            fmax = A[low] ; fmin = A[high]
        }
    }
else
{
    mid =  $\left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor$ 
    Find _ Max _ Min (low, mid, lmax, lmin)
    Find _ Max _ min (mid + 1, high, rmax, rmin)
    if lmax > rmax then fmax = lmax
        else fmax = rmax
    if lmin < rmin then fmin = lmin
        else fmin = rmin
}

```

i	1	2	3	4	5	6	7	8	9	10	11
A[i]	19	11	-8	-17	10	7	3	-9	10	1	2



Algorithm Find_Max_Min (low, high, fmax, fmin)

if low = high **then** fmax = fmin = A[low]

else if high = low + 1 **then**

{

if A[low] < A[high] **then**

{

fmax = A[high] ; fmin = A[low]

}

else

{

fmax = A[low] ; fmin = A[high]

}

else

{

$$\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor$$

Find_Max_Min (low, mid, lmax, lmin)

Find_Max_min (mid + 1, high, rmax, rmin)

if lmax > rmax **then** fmax = lmax

else fmax = rmax

if lmin < rmin **then** fmin = lmin

else fmin = rmin

}

- عمل اصلی: دستور مقایسه

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

$$T(2) = 1$$

$$T(1) = 0$$

$$T(n) = \theta(n)$$

2-2 مرتب سازی ادغامی

- ادغام یک فرآیند مرتبط با مرتب سازی است.
- ادغام دو طرفه به معنای ترکیب دو آرایه مرتب شده در یک آرایه‌ی مرتب است.
- آرایه $1 = 10, 12, 20, 27$
- آرایه $2 = 9, 13, 25, 30$
- ادغام دو طرفه $= 9, 10, 12, 13, 20, 25, 30$

- 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.

27 10 12 20 25 13 15 22.

1. Divide the array:

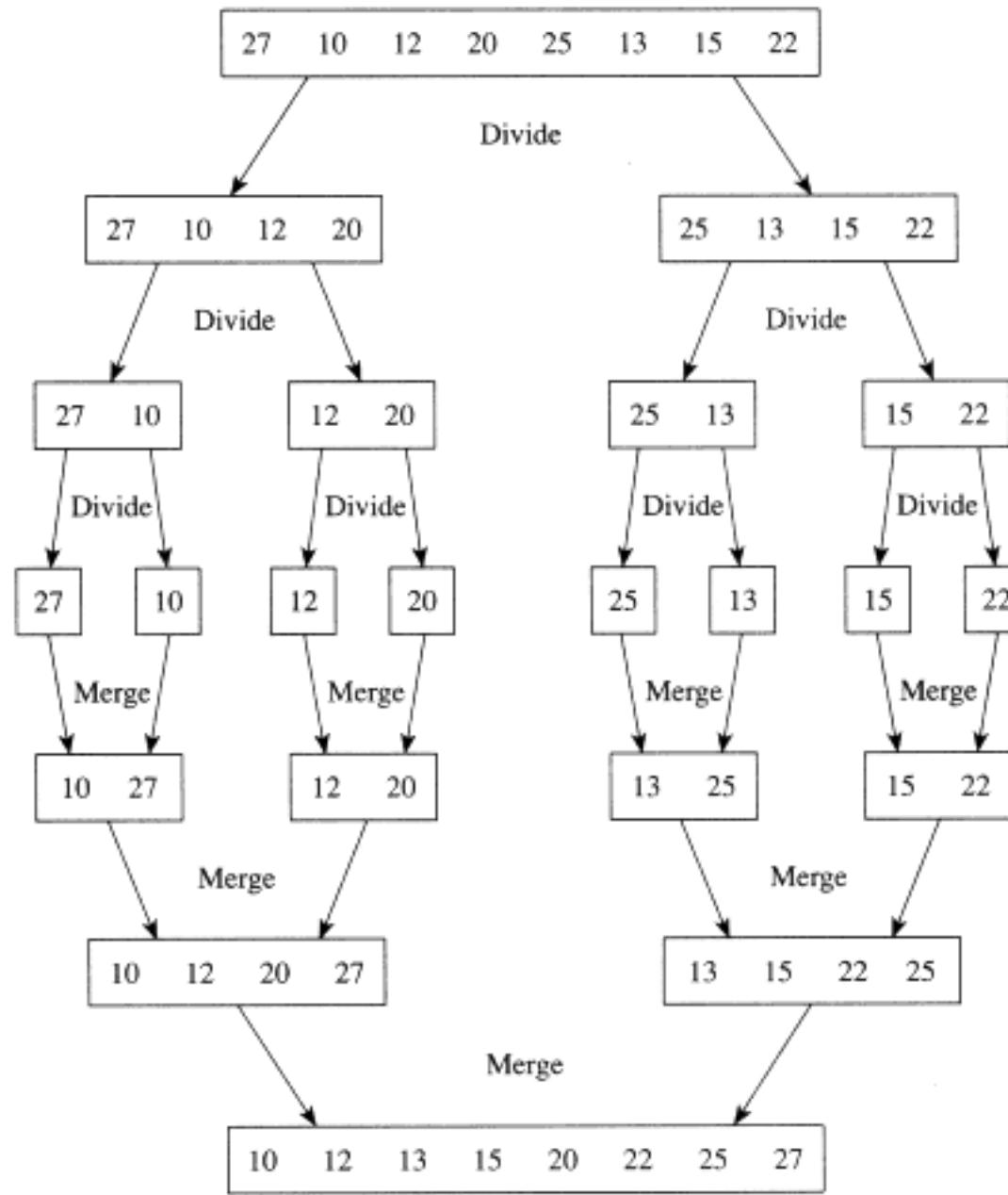
27 10 12 20 and 25 13 15 22.

2. Sort each subarray:

10 12 20 27 and 13 15 22 25.

3. Merge the subarrays:

10 12 13 15 20 22 25 27.



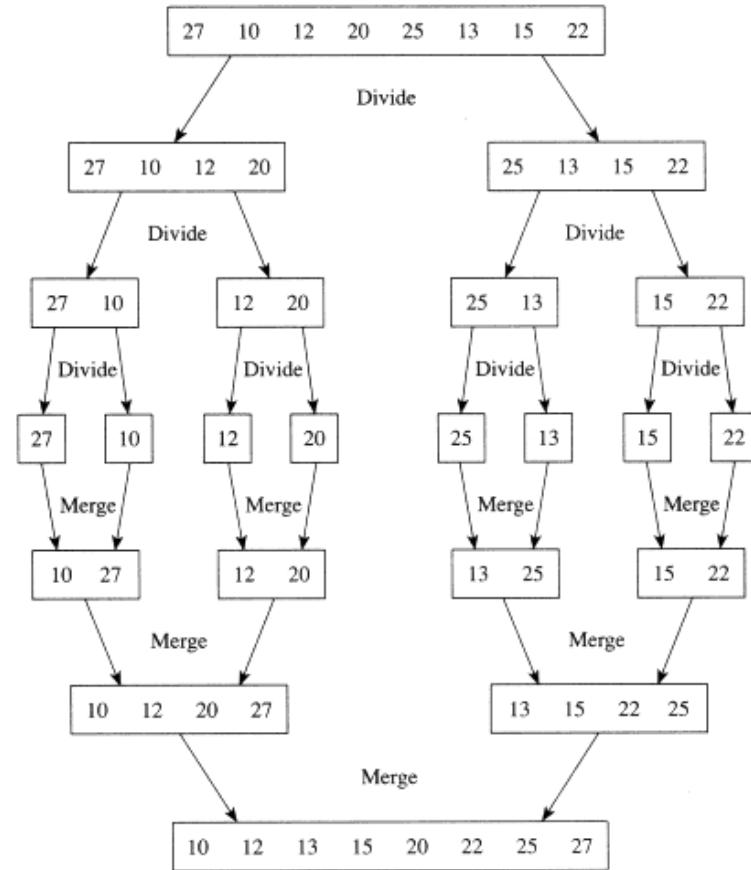
الگوریتم ۲-۲: مرتب سازی ادغامی

```
void mergesort (int n , keytype S [ ] )  
{  
    const int h =  $\lfloor \frac{n}{2} \rfloor$  , 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);  
    }  
}
```

```

void mergesort (int n , keytype S [ ])
{
    const int h = l n/2 r , 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);
    }
}

```



الگوريتم ٣-٢: ادغام

```
void merg ( 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 در یک آرایه S

```

void merg ( int h , int m, const keytype U[ ],
            const keytype V[ ],
            keytype S[] )
{
    index j , j , k;
    j = 1; j = 1 ; k = 1;
    while ( j <= h && j <= m) {
        if (U [ j ] < V [ j ]) { S [k] = U [ j ]
            j + + ;
        }
        else {S [k] = V [ j ];
            j + + ;
        }
        k + + ;
    }
    if ( j > h)
        copy V [ j ] through V [m] to S [k] through S [ h + m ]
    else
        copy U [ j ] through U [h] to S [k] through S [ h + m ]
}

```

k	U					V					S (Result)				
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

*The items compared are in boldface.

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

عمل اصلی: مقایسه $[i]U$ با $[j]V$.

اندازه ورودی: m و h ، تعداد عناصر موجود در هر یک از دو آرایه ورودی.
بدترین حالت زمانی است که یکی تمام شده است و دیگری یکی به آخر.

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

بهترین حالت زمانی رخ می دهد که کلیه عناصر یک آرایه از کلیه عناصر آرایه دیگر بزرگتر باشد.

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

عمل اصلی: مقایسه ای که در ادغام صورت می‌پذیرد.

اندازه ورودی: n ، تعداد عناصر آرایه S .

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

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

برای $n > 1$ که n توانی از ۲ است

$$W(1) = 0$$

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

مرتب سازی درجا

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

الگوريتم ۴-۲: مرتب سازی ادغامی (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)
    }
}
```

الگوریتم ۵-۲: ادغام

مسئله: ادغام دو آرایه‌ی مرتب S که در mergesort ایجاد شده‌اند.

```
void mrgre2 (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]

}

۳-۲- روش تقسیم و حل

راهبرد طراحی تقسیم و حل شامل مراحل زیر است:

- ۱- تقسیم نمونه ای از یک 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.

۲-۴ مرتب سازی سریع (Quicksort)

- در مرتب سازی سریع، ترتیب آن ها از چگونگی افراز آرایه ها ناشی می شود.
- همه عناصر کوچک تر آز عنصر محوری در طرف چپ آن و همه عناصر بزرگ تر، در طرف راست آن واقع هستند.

- مرتب سازی سریع، به طور بازگشتی فراخوانی می شود تا هر یک از دو آرایه را مرتب کند، آن ها نیز افزار می شوند و این روال ادامه می یابد تا به آرایه ای با یک عنصر بررسیم. چنین آرایه ای ذاتاً مرتب است.

Pivot item



15 22 13 27 12 10 20 25

1. Partition the array so that all items smaller than the pivot item are to the left of it and all items larger are to the right:

Pivot item



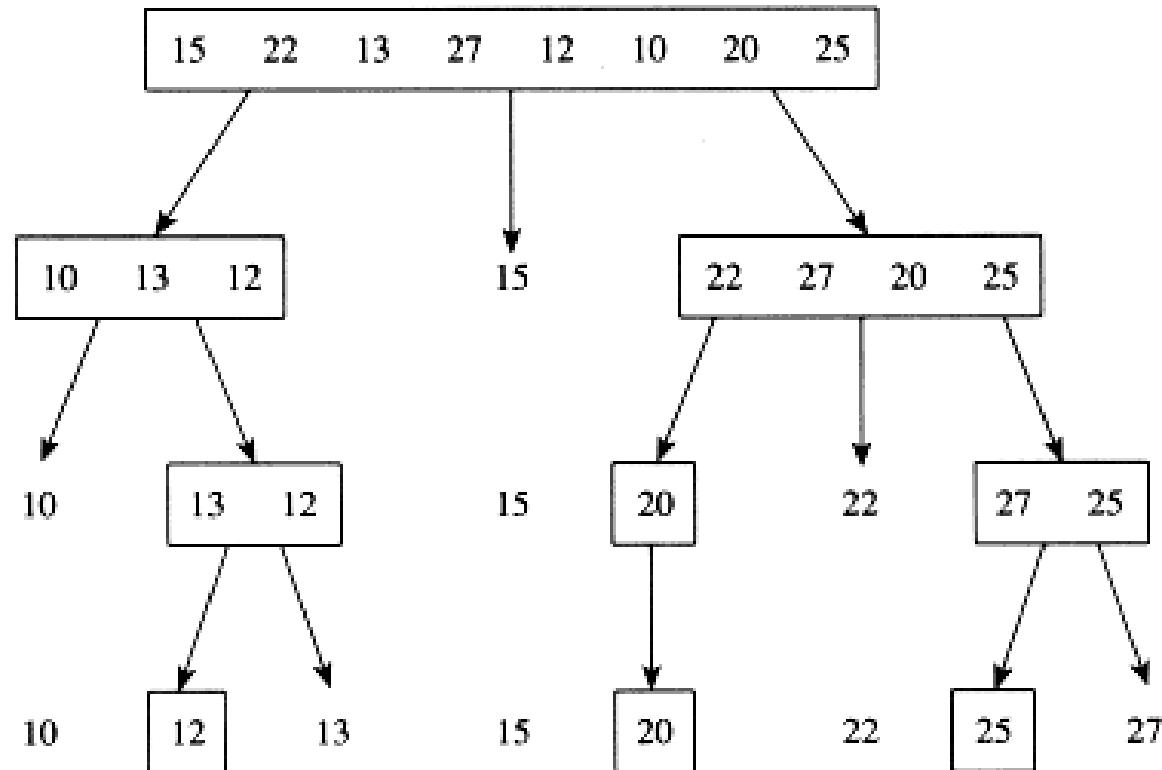
10 13 12 15 22 27 20 25
All smaller All larger

2. Sort the subarrays:

Pivot item



10 12 13 15 20 22 25 27
Sorted Sorted



الگوریتم ۲-۶: 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);
    }
}
```

الگوريتم 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];
}
```

```
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]; }
```

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

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

عمل اصلی: مقایسه $S[i]$ با pivotitem .
اندازه ورودی: n ، تعداد عناصر موجود در زیر آرایه.

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

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

. `partition`: مقایسه $[i]$ با `pivotitem` در روال `S` عمل اصلی است. `n`، تعداد عناصر موجود در آرایه `S` اندازه ورودی است.

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

\downarrow \downarrow \downarrow

زمان لازم برای افزایش زمان لازم برای مرتب سازی زیرآرایه طرف راست زمان لازم برای مرتب سازی زیرآرایه طرف چپ

$$T(n) = T(n-1) + n - 1 \quad n > 0 \text{ به ازای} \\ T(0) = 0$$

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

$$T(n) = aT(n-k) + b$$

$$T(n) = O(a^{n/k}) \quad \text{if } a \neq 1$$

$$T(n) = O(n) \quad \text{if } a = 1$$

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

عمل اصلی: مقایسه $S[i]$ با p در $\cdot \text{partition}$

اندازه ورودی: n ، تعداد عناو

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

↓
 Probability
 pivotpoint is p
 ↓
 Average time to
 sort subarrays when
 pivotpoint is p
Time to
 partition

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

- * تقسیم مسئله به تعدادی زیرمسئله با اندازه کوچکتر

- * حل و ادغام زیرمسئله‌ها برای حل مسئله اصلی

با فرض اینکه تعداد زیرمسئله‌ها a باشد، اندازه‌ی زیرمسئله‌ها با ضریب b کاهش یابد و پیچیدگی ادغام $O(n^d)$ باشد برای تحلیل الگوریتم داریم:

$$T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$$

به عنوان مثال برای الگوریتم مرتب‌سازی ادغامی رابطه‌ی زیر برقرار است:

$$T(n) \leq 2T\left(\frac{n}{4}\right) + O(n)$$

قضیه ۳ (قضیه اصلی): طبق تعاریف بالا اگر رابطه‌ی بازگشتی زیر را داشته باشیم:

$$T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$$

آنگاه:

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a < b^d \\ O(n^d) & \text{if } a = b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

ضرب ماتریس

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix},$$

their product $C = A \times B$ is given by

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j}.$$

For example,

$$\begin{bmatrix} 2 & 3 \\ 4 & 1 \end{bmatrix} \times \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix} = \begin{bmatrix} 2 \times 5 + 3 \times 6 & 2 \times 7 + 3 \times 8 \\ 4 \times 5 + 1 \times 6 & 4 \times 7 + 1 \times 8 \end{bmatrix} = \begin{bmatrix} 28 & 38 \\ 26 & 36 \end{bmatrix}.$$

In general, if we have two $n \times n$ matrices A and B , their product C is given by

$$c_{ij} = \sum_{k=1}^n a_{ik}b_{kj} \quad \text{for } 1 \leq i, j \leq n.$$

$$T(n) = \Theta(n^3)$$

- برای ضرب ماتریس 2×2 به ۸ عمل ضرب و ۵ عمل جمع تفریق نیاز دارد

```
void matrixmult (int n,
                  const number A[][],
                  const number B[][],
                  number C[][])
{
    index i, j, k;

    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++) {
            C[i][j] = 0;
            for (k = 1; k <= n; k++)
                C[i][j] = C[i][j] + A[i][k] * B[k][j];
        }
}
```

۵-۲-الگوریتم 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 2×2 ارزش چندانی ندارد.

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}. \quad \text{برای } 2 \times 2 \text{ تا ضرب و جمع و تفریق}$$

$$m_1 = (a_{11} + a_{22})(b_{11} + b_{22})$$

$$m_2 = (a_{21} + a_{22})b_{11}$$

$$m_3 = a_{11}(b_{12} - b_{22})$$

$$m_4 = a_{22}(b_{21} - b_{11})$$

$$m_5 = (a_{11} + a_{12})b_{22}$$

$$m_6 = (a_{21} - a_{11})(b_{11} + b_{12})$$

$$m_7 = (a_{12} - a_{22})(b_{21} + b_{22}),$$

the product C is given by

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

$$\downarrow \begin{matrix} n/2 \\ \hline C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{matrix} = \begin{matrix} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{matrix} \times \begin{matrix} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{matrix}$$

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

Using Strassen's Method, first we compute

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

$$\begin{matrix} \mathbf{C}_{11} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\ \mathbf{C}_{12} \quad \mathbf{C}_{21} \quad \mathbf{C}_{22} \end{matrix}$$

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

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

Figure 2.5 illustrates the partitioning in Strassen's Method. The computations proceed as follows:

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

When the matrices are sufficiently small, we multiply in the standard way. In this example, we do this when $n = 2$. Therefore,

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

After this, M_2 through M_7 are computed in the same way, and then the values of C_{11} , C_{12} , C_{21} , and C_{22} are computed. They are combined to yield C .

الگوريتم ۲-۸: استراسن

مسئله: تعين حاصلضرب دو ماتريص $n \times n$ 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;
        //example recursive call; strassen(n/2, A11 + A22, B11 + B22, M1)
    }
}
```

تحلیل پیچیدگی زمانی تعداد ضرب ها در الگوریتم ۲-۸(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 ۲ است

$$T(1) = 1$$

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

تحلیل پیچیدگی زمانی تعداد جمع ها و تفریق های الگوریتم (استراسن) در حالت معمول

عمل اصلی: یک جمع یا تفریق ساده.

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

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

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

بهترین الگوریتم برای ضرب ماتریس‌ها

بهترین الگوریتم موجود برای ضرب دو ماتریس $n \times n$ از مرتبه‌ی $O(n^{2.376})$ است که توسط کاپراسمیت و وینوگراد در سال ۱۹۸۷ ارایه شد. البته این نتیجه صرفاً از بعد نظری قابل توجه است و هنوز کاربرد عملی ندارد.

همچنان روشن نیست که آیا بهتر از این الگوریتم وجود دارد یا خیر. البته بدیهی که حد پایین این الگوریتم $\Theta(n^2)$ است.

اعمال محاسباتی روی اعداد صحیح بزرگ

- اعدادی که بزرگتر از حدی است که سخت افزار کامپیوتر قادر به نمایش آن ها است.
- یک راه استفاده از آرایه ای از اعداد صحیح است که در آن هر محل از آرایه یک رقم را نگهداری می نماید.
- برای نگهداری اعداد **مثبت و منفی** فقط کافی است که بالاترین محل آرایه را برای علامت عدد رزرو کنیم. برای نشان دادن عدد مثبت از صفر و برای منفی از یک استفاده نماییم.

الگوریتم های ساده

- جمع

- تفریق

- $U * (10^m)$

- $U \text{ divide } (10^m)$

- $U \text{ rem } (10^m)$

ضرب اعدا صحيح بزرگ

- $567,832 = 567 * (10^3) + 832$
- $9,423,723 = 9423 * (10^3) + 723$
- $u = x * (10^m) + y$
- $v = w * (10^m) + z$

- $u = x * (10^m) + y$
- $v = w * (10^m) + z$
- $uv = xw * (10^{2m}) + (xz + wy) * (10^m) + yz$
- $m \leq n/2$
- $y \leq n/2$
- $x \leq n/2$

الگوريتم ۹-۲: ضرب اعداد صحيح بزرگ

مسئله: ضرب دو عدد صحيح 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);
}
}
```

567,832 * 9,423,723

تحلیل پیچیدگی زمانی در بدترین حالت برای الگوریتم ۹-۲ (ضرب اعداد صحیح)

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

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

به ازای $s > n$ که n توانی از ۲ است

$$W(s) = 0$$

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

باید تعداد ضرب ها را کم کنیم.

ضرب دارد.

باید تعداد ضرب ها را کاهش دهیم.

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

در این صورت خواهیم داشت:

$xz + yw = r - xw - yz$
یعنی فقط کافی است که ۳ مقدار زیر را بدست آوریم:

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

بر همین اساس الگوریتم دوم ارائه می شود

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 ;
    }
}
```

تحليل پیچیدگی زمانی در بدترین حالت برای الگوریتم ۱-۲ (ضرب اعداد صحیح^۲)

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

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

به ازای $s > n$ که n توانی از ۲ است

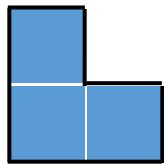
$$3W(n/2) + c n \leq W(n) \leq 3W(n/2 + 1) + c n$$
$$W(s) = 0$$

$$W(n) = \theta(n^{1.58})$$

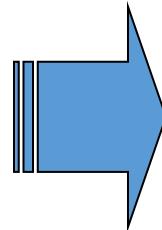
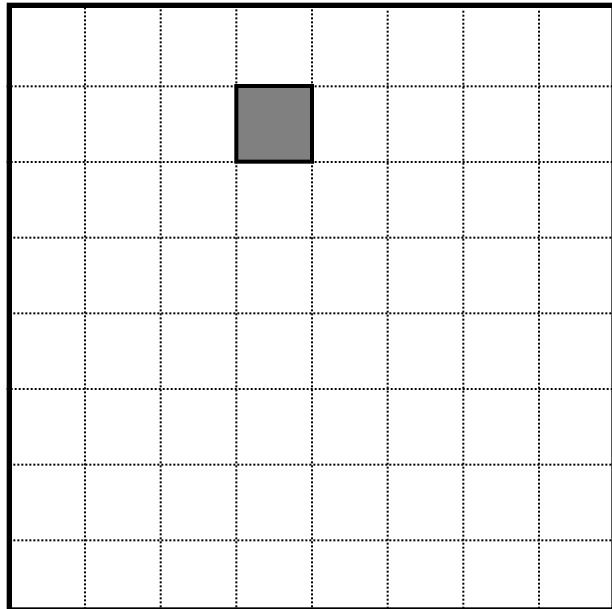
بوردین و مانرو با استفاده از تبدیلات فوریه یک الگوریتم $(n \log n)^2$ برای ضرب اعداد صحیح بزرگ ابداع کرده است.

Tromino Tiling

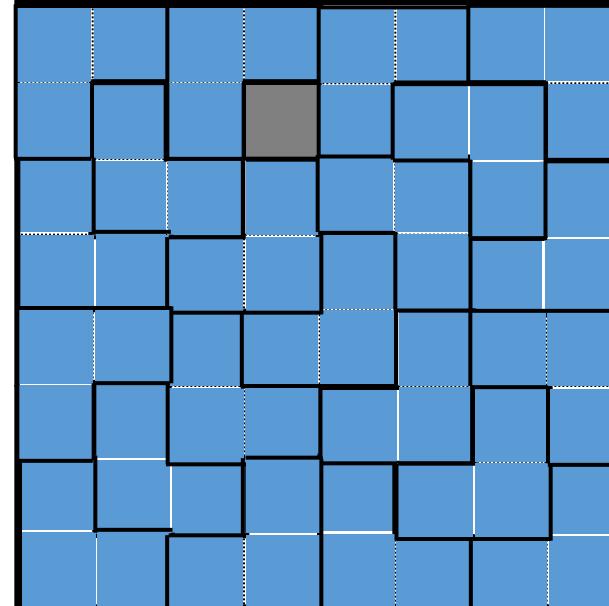
A tromino tile:



And a $2^n \times 2^n$ board
with a hole:

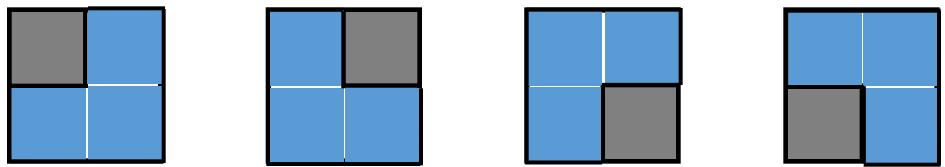


A tiling of the board
with trominos:



Tiling: Trivial Case ($n = 1$)

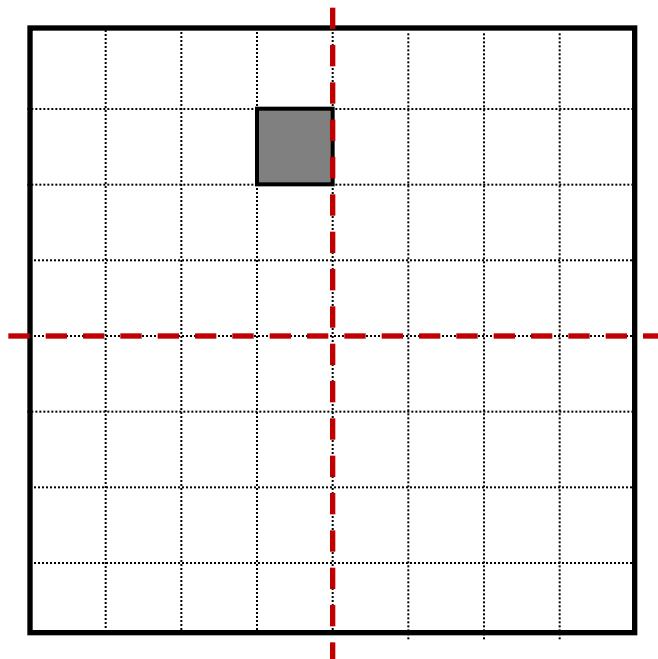
Trivial case ($n = 1$): tiling a 2×2 board with a hole:



Idea – try somehow to reduce the size of the original problem, so that we eventually get to the 2×2 boards which we know how to solve... ■

Tiling: Dividing the Problem

To get smaller square boards let's divide the original board into four boards

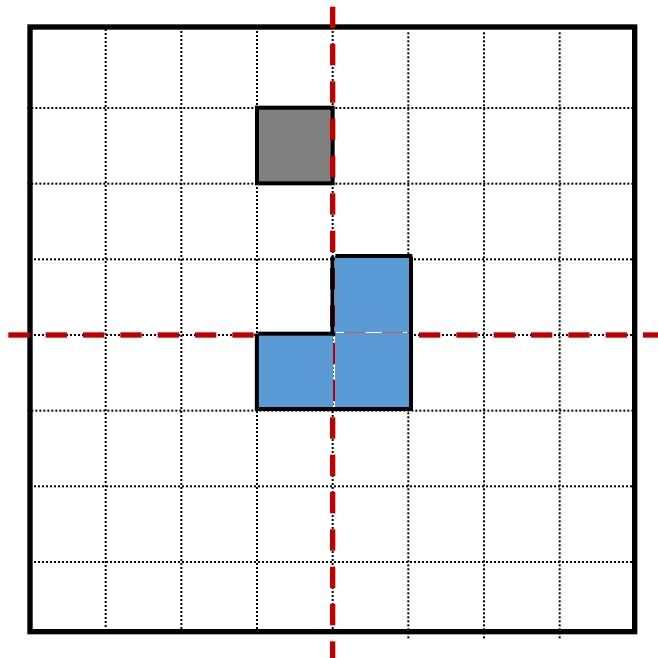


Great! We have one problem
of the size $2^{n-1} \times 2^{n-1}$!

But: The other three
problems are not similar to
the original problems – they
do not have holes!

Tiling: Dividing the Problem

Idea: insert one tromino at the center to get three holes •
in each of the three smaller boards



- Now we have four boards with holes of the size $2^{n-1} \times 2^{n-1}$. ■
- Keep doing this division, until we get the 2x2 boards with holes – we know how to tile those ■

Tiling: Algorithm

INPUT: n – the board size ($2^n \times 2^n$ board), L – location of the hole.

OUTPUT: tiling of the board

Tile(n , L)

if $n = 1$ **then**

Trivial case

 Tile with one tromino

return

 Divide the board into four equal-sized boards

 Place one tromino at the center to cut out 3 additional holes

 Let L_1, L_2, L_3, L_4 denote the positions of the 4 holes

Tile($n-1$, L_1)

Tile($n-1$, L_2)

Tile($n-1$, L_3)

Tile($n-1$, L_4)

Divide and Conquer

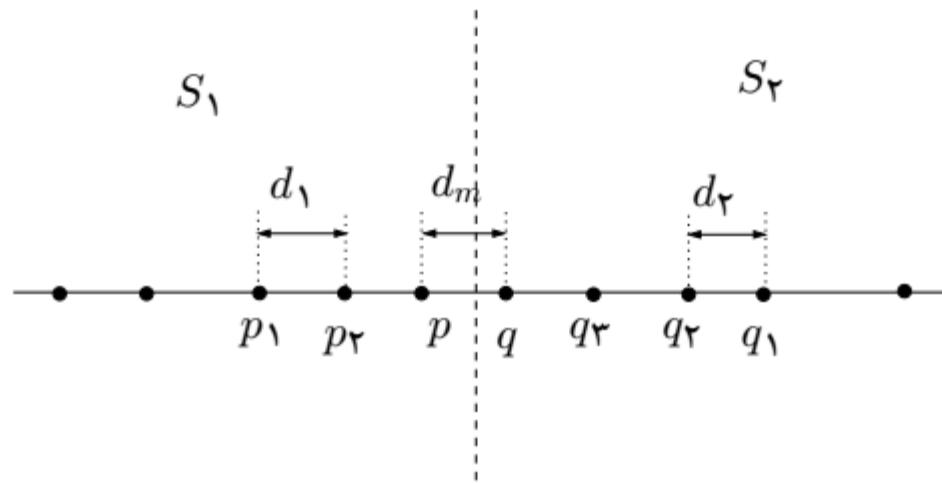
- *Divide-and-conquer* method for algorithm design:
 - If the problem size is small enough to solve it in a straightforward manner, solve it. Else:
 - **Divide:** Divide the problem into two or more disjoint *subproblems*
 - **Conquer:** Use divide-and-conquer recursively to solve the subproblems
 - **Combine:** Take the solutions to the subproblems and combine these solutions into a solution for the original problem

Tiling: Divide-and-Conquer

- Tiling is a divide-and-conquer algorithm:
 - Just do it trivially if the board is 2x2, else:
 - **Divide** the board into four smaller boards (introduce holes at the corners of the three smaller boards to make them look like original problems)
 - **Conquer** using the same algorithm recursively
 - **Combine** by placing a single tromino in the center to cover the three introduced holes

نزدیک ترین زوج نقاط

یک بعدی

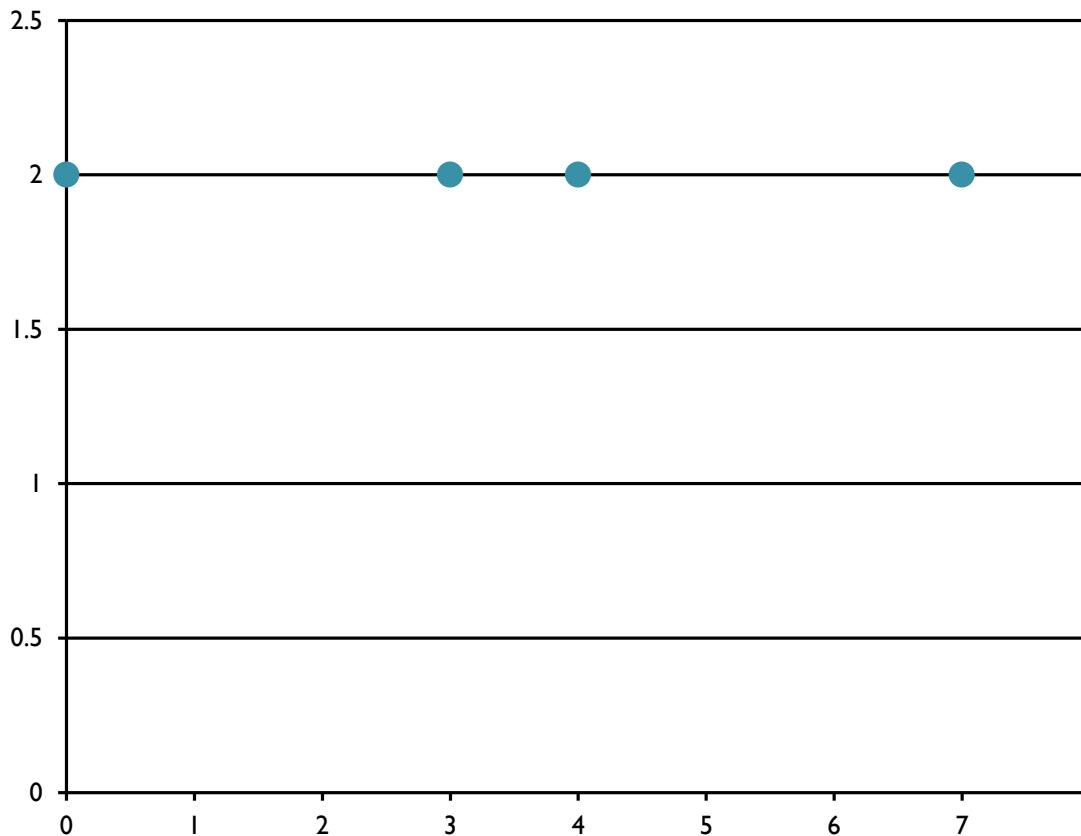


CLOSET-PAIR-1D(S)

```
1  if  $|S| = 2$ 
2  then return  $d = |p_1 - p_2|$ 
3  else divide  $S$  from the mid-point and Construct  $S_1$  and  $S_2$ 
4       $d_1 \leftarrow$  CLOSET-PAIR-1D( $S_1$ )
5       $d_2 \leftarrow$  CLOSET-PAIR-1D( $S_2$ )
6       $p \leftarrow \max(S_1)$ 
7       $q \leftarrow \min(S_2)$ 
8       $d \leftarrow \min \{d_1, d_2, q - p\}$ 
```

Finding the Closest Pair of Points

- Problem:
 - Given n ordered pairs $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, find the distance between the two points in the set that are closest together.



Closest-Points Problem

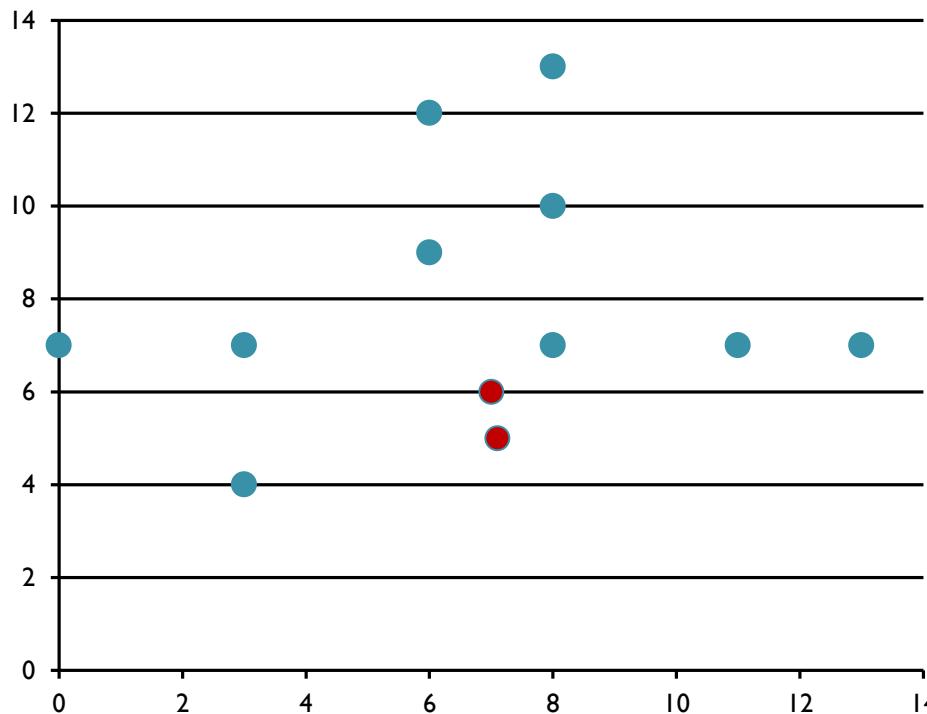
- Brute Force Algorithm
 - Iterate through all possible pairs of points, calculating the distance between each of these pairs. Any time you see a distance shorter than the shortest distance seen, update the shortest distance seen.

Since computing the distance between two points takes $O(1)$ time,

And there are a total of $n(n-1)/2 = \theta(n^2)$ distinct pairs of points,

It follows that the running time of this algorithm is $\theta(n^2)$.

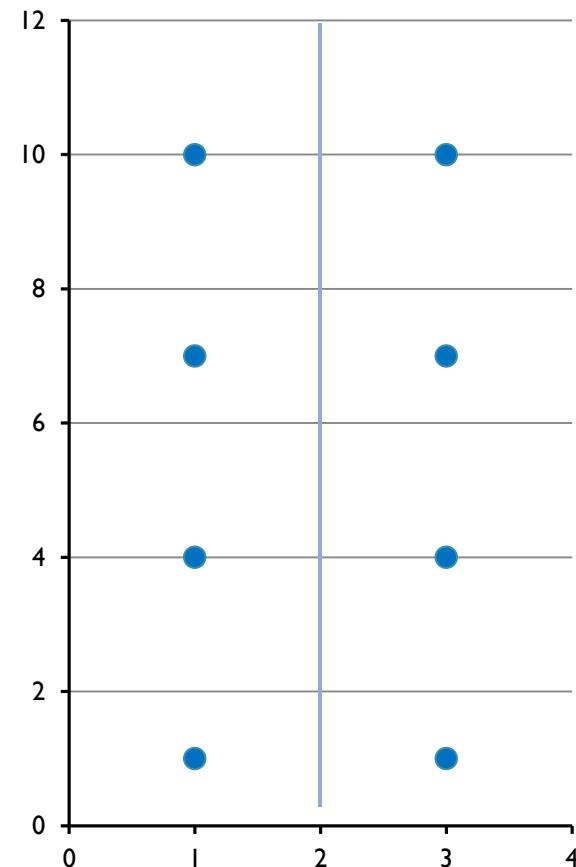
Can we do better?

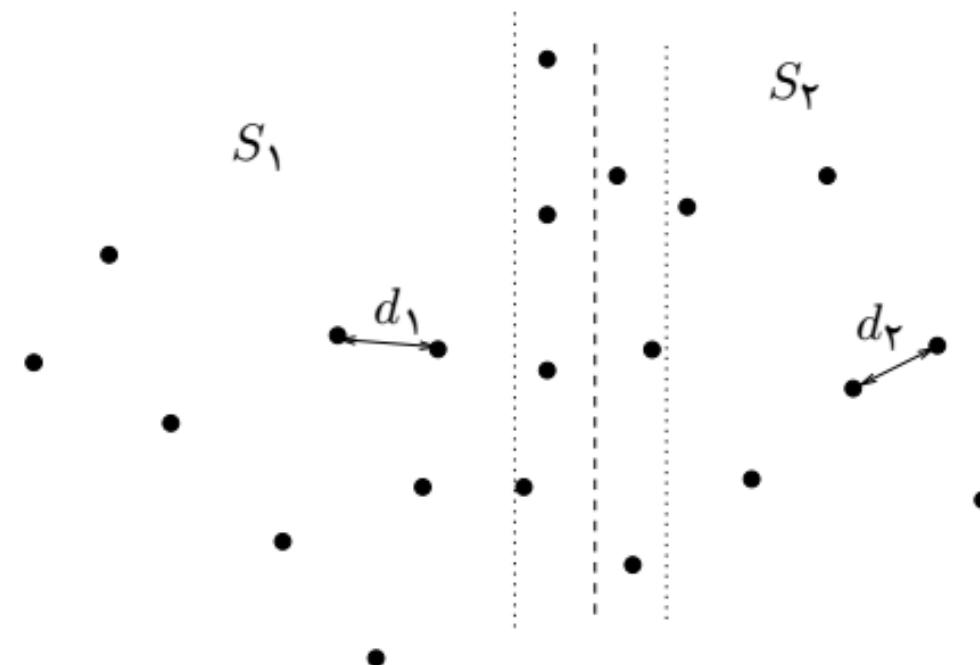


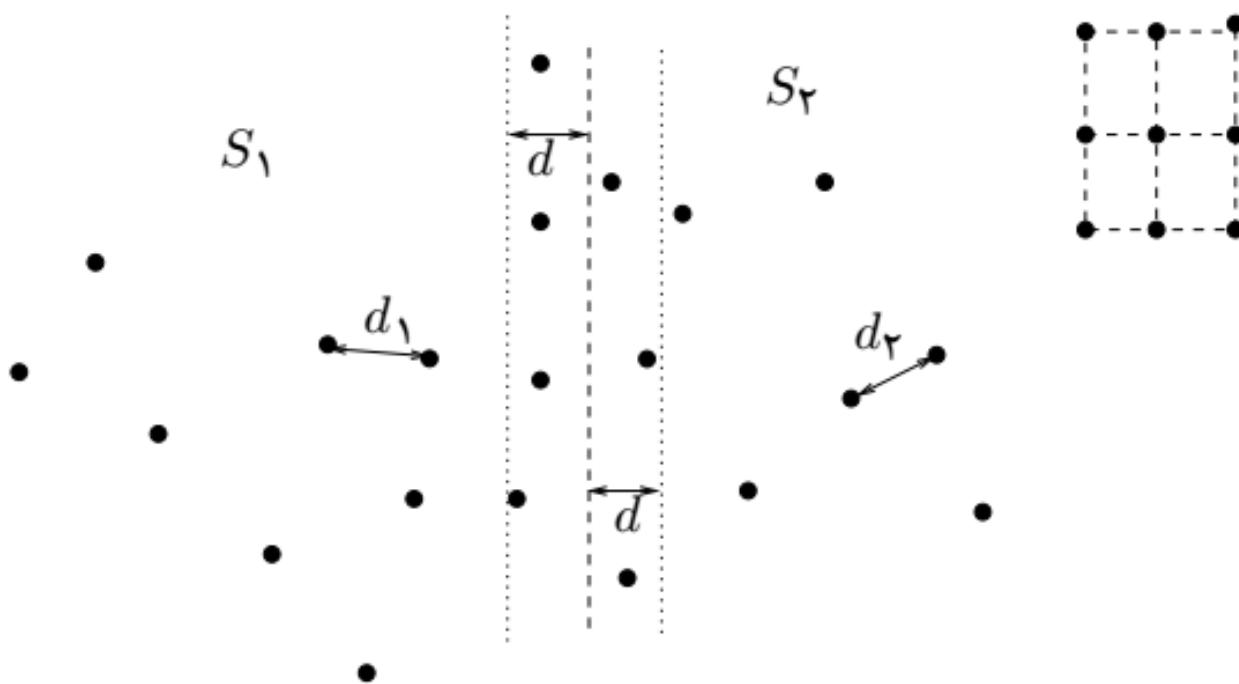
Closest Points Problem

- However, one could construct a case where ALL the points on each side are within δ of the vertical line:

- So, this case is as bad as our original idea where we'd have to compare each pair of points to one another from the different groups.
- But, wait!! Is it really necessary to compare each point on one side with every other point on every other side???

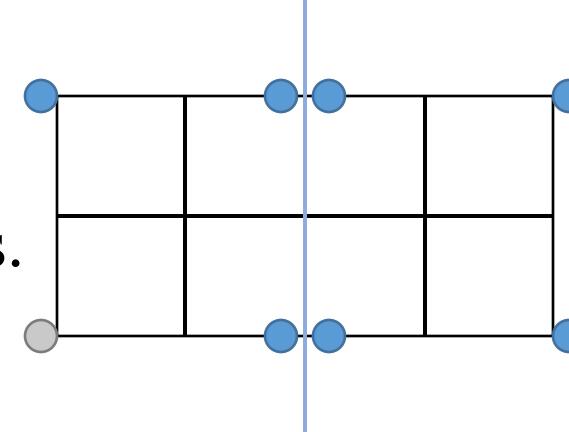






Closest Points Problem

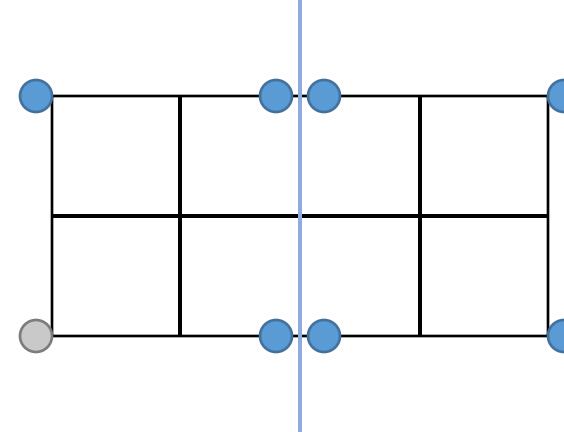
- Consider the following rectangle around the dividing line that is constructed by eight $\delta/2 \times \delta/2$ squares.



- Note that the diagonal of each square is $\delta/\sqrt{2}$, which is less than δ .
 - Since each square lies on a single side of the dividing line, at most one point lies in each box
 - Because if 2 points were within a single box the distance between those 2 points would be less than δ .
 - Therefore, there are at MOST 7 other points that could possibly be a distance of less than δ apart from a given point, that have a greater y coordinate than that point.
 - (We assume that our point is on the bottom row of this grid; we draw the grid that way.)

Closest Points Problem

- Now we have the issue of how do we know ***which 7 points*** to compare a given point with?
- The idea is:
 - As you are processing the points recursively, ***SORT*** them based on the **y-coordinate**.
 - Then for a given point within the strip, you only need to compare with the **next 7 points**.



Closest Points Problem

- Now the Recurrence relation for the runtime of this problem is:
 - $T(n) = T(n/2) + O(n)$
 - Which is the same as Mergesort, which we've shown to be $O(n \log n)$.

ClosestPair(ptsByX, ptsByY, n)

if ($n = 1$) return 1

if ($n = 2$) return distance(ptsByX[0], ptsByX[1])

// Divide into two subproblems

mid $\leftarrow n/2 - 1$

copy ptsByX[0 . . . mid] into new array XL *in x order.*

copy ptsByX[mid+1 . . . n - 1] into new array XR

copy ptsByY into arrays Y L and Y R *in y order,* s.t.

XL and Y L refer to same points, as do XR, Y R.

// Conquer

distL \leftarrow ClosestPair(XL, Y L, n/2)

distR \leftarrow ClosestPair(XR, Y R, n/2)

// Combine

midPoint ptsByX[mid]

lrDist min(distL, distR)

Construct array yStrip, in increasing y order,

of all points p in ptsByY s.t.

$|p.x - \text{midPoint}.x| < \text{lrDist}$

// Check yStrip

minDist \leftarrow lrDist

for ($j \leftarrow 0; j \leq \text{yStrip.length} - 2; j++$) {

 k $\leftarrow j + 1$

 while ($k < \text{yStrip.length} - 1$ and

$\text{yStrip}[k].y - \text{yStrip}[j].y < \text{lrDist}$) {

 d \leftarrow distance(yStrip[j], yStrip[k])

 minDist $\leftarrow \min(\text{minDist}, d)$

 k++

}

}

return minDist

```
closest_pair(p) {
    mergesort(p, 1, n) // n is number of points
    return rec_cl_pair(p, 1, 2)
}
```

```
rec_cl_pair(p, i, j) {
    if (j - i < 3) { \\ If there are three points or less...
        mergesort(p, i, j) // based on y coordinate
        return shortest_distance(p[i], p[i+1], p[i+2])
}
```

```
xval = p[(i+j)/2].x
deltaL = rec_cl_pair(p, i, (i+j)/2)
deltaR = rec_cl_pair(p, (i+j)/2+1, j)
delta = min(deltaL, deltaR)
merge(p, i, j) // merge points based on y coordinate
```

```
v = vert_strip(p, xval, delta)

for k=1 to size(v)-1
    for s = (k+1) to min(t, k+7)
        delta = min(delta, dist(v[k], v[s]))
    return delta
}
```

مسئله‌ی برج‌ها

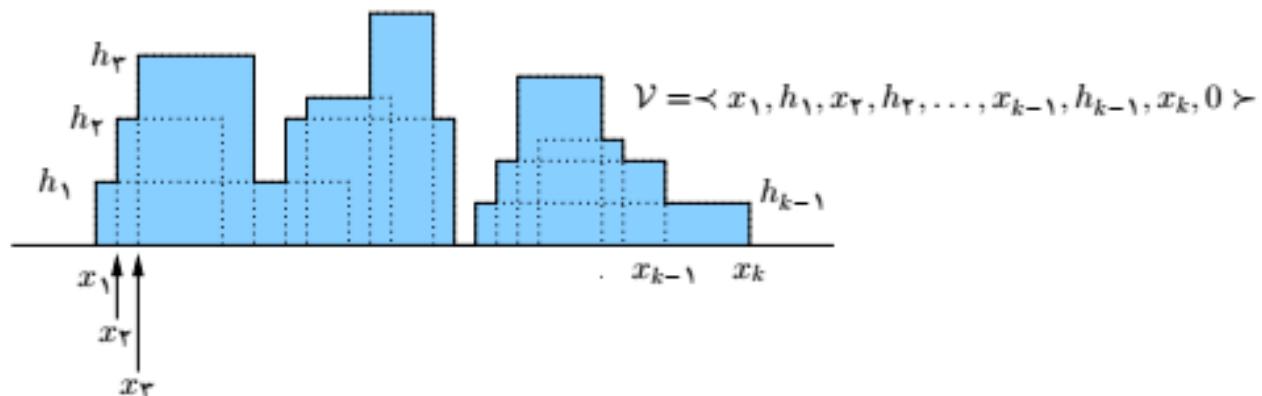
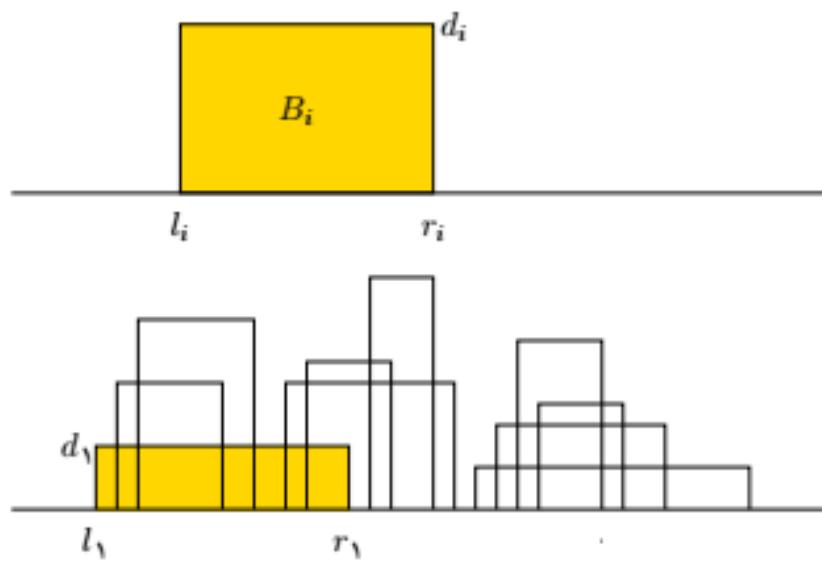
n تا برج بر روی محور x

برج B_i با سه عدد $\langle l_i, r_i, d_i \rangle$

می‌خواهیم «نمای برج‌ها» را به دست بیاوریم.

نمای: $\langle x_1, h_1, x_2, h_2, \dots, x_{k-1}, h_{k-1}, x_k, \circ \rangle$

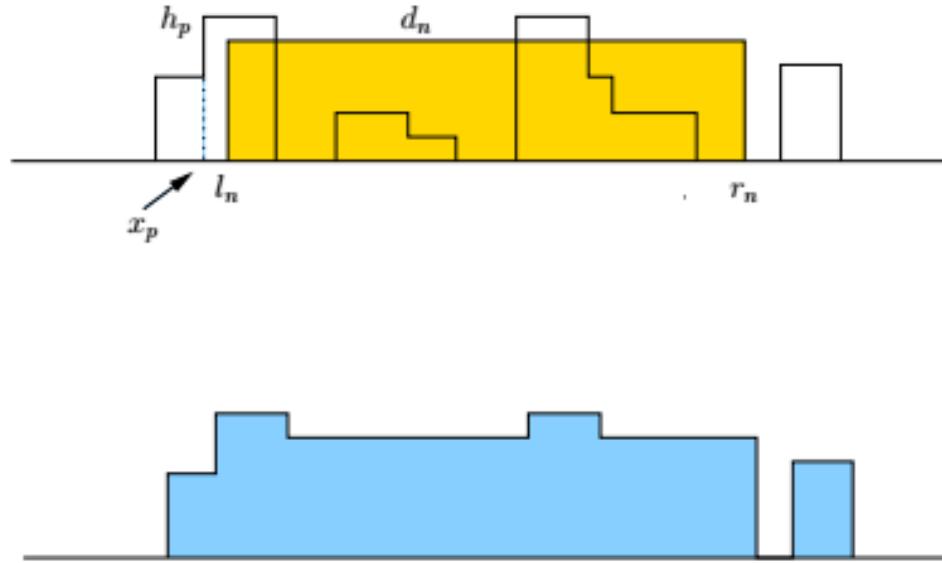
طراحی و تحلیل الگوریتم‌ها



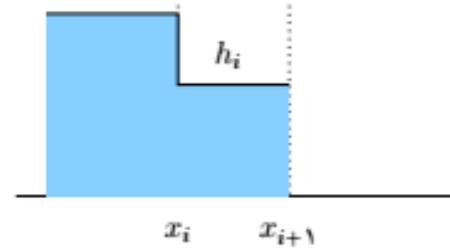
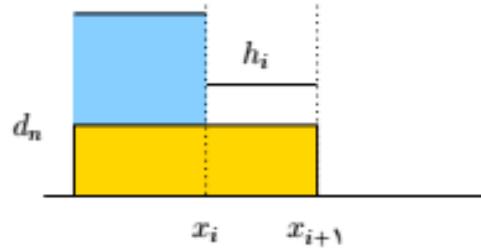
راه حل استقرایی

- برای $1 = n$ نما همان برج
- نما را برای $1 - n$ برج به صورت بازگشته به دست می‌آوریم
- برج B_n را اضافه می‌کنیم و نمای جدید را به دست می‌آوریم.

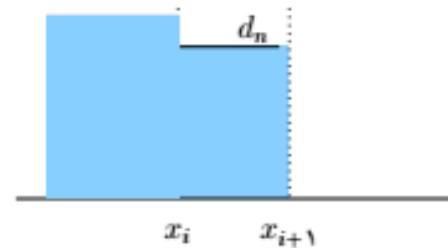
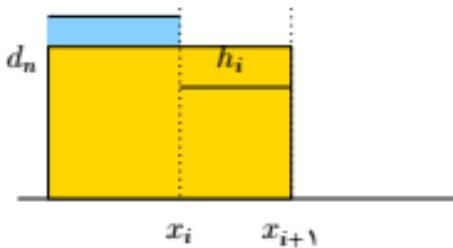
طراحی و تحلیل الگوریتم‌ها



طراحی و تحلیل الگوریتم‌ها

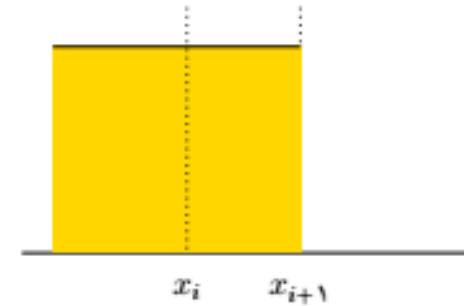
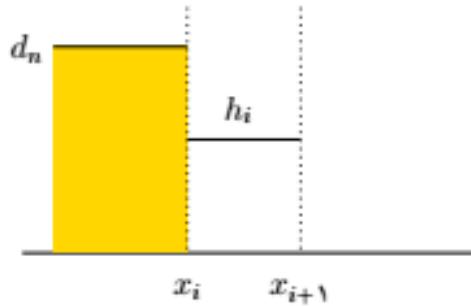


last-height $> d_n$, $h_i \geq d_n$

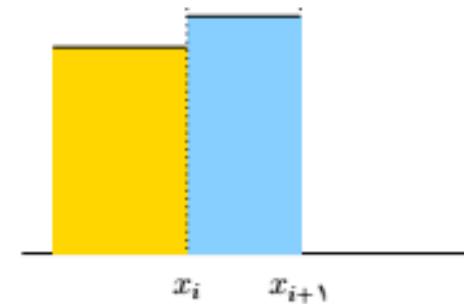
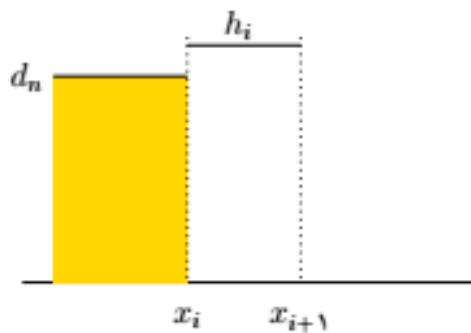


last-height $> d_n$, $h_i \leq d_n$

طراحی و تحلیل الگوریتم‌ها



last-height = d_n , $h_i < d_n$



last-height = d_n , $h_i < d_n$

```

ADDBORI( $\mathcal{V}_{n-1}, B_n$ )
    ▷ add  $B_n$  to view  $\mathcal{V}_{i-1}$ 
    ▷ Let  $\mathcal{V}_{n-1} = \langle x_1, h_1, x_2, h_2, \dots, x_{k-1}, h_{k-1}, x_k, 0 \rangle$ 
    ▷ Let  $B_n = \langle l_n, r_n, d_n \rangle$ 
1   Find  $x_p < l_n < x_{p+1}$  using binary search
2    $\mathcal{V} \leftarrow \langle x_1, h_1, \dots, x_p, h_p \rangle$ 
3   if  $d_n > h_p$ 
4       then Add  $\langle l_n, d_n \rangle$  to  $\mathcal{V}$ ; last-height  $\leftarrow d_n$ 
5       else last-height  $\leftarrow h_p$ 
6    $i \leftarrow p + 1$  while  $x_i < r_n$ 
7       do if last-height  $> d_n$  and  $h_i \geq d_n$ 
8           then Add  $\langle x_i, d_i \rangle$  to  $\mathcal{V}$ 
9           last-height  $\leftarrow d_i$ 
10      if last-height  $> d_n$  and  $h_i \leq d_n$ 
11          then Add  $\langle x_i, d_n \rangle$  to  $\mathcal{V}$ 
12          last-height  $\leftarrow d_n$ 
13      if last-height  $= d_n$  and  $h_i > d_n$ 
14          then Add  $\langle x_i, d_i \rangle$  to  $\mathcal{V}$ 
15          last-height  $\leftarrow d_i$ 
16       $i \leftarrow i + 1$ 
17      if last-height  $= d_n$ 
18          then Add  $\langle r_n, h_{i-1} \rangle$  to  $\mathcal{V}$ 
19  return  $\mathcal{V}$ 

```

روش استقرایی: پیچیدگی

پیدا کردن l_n : $\Theta(\log k)$

تصحیح ارتفاع‌ها می‌تواند حداقل $\Theta(k)$ باشد. چون $n \leq k$ داریم:

$$T(n) = T(n - 1) + \Theta(n) \implies T(n) = \Theta(n^2)$$

روش تقسیم و حل

تقسیم مسئله‌ی n را به دو مسئله با اندازه‌های $\lceil n/2 \rceil$ هر دو زیرمسئله را به همین روش حل می‌کنیم.

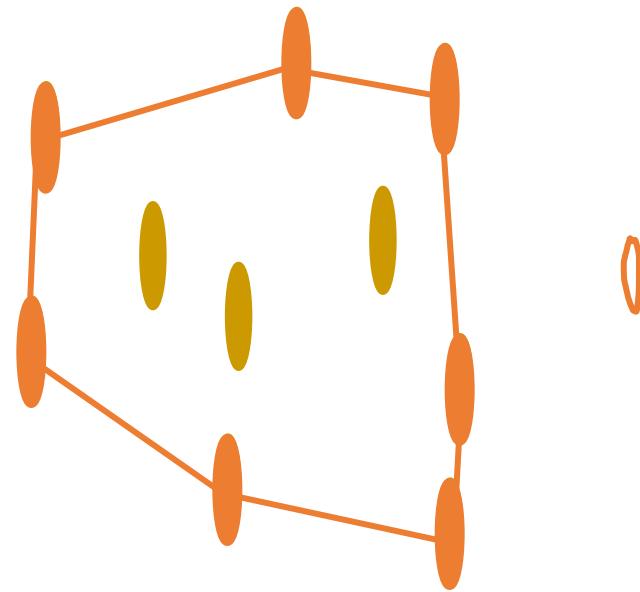
نمای اول: $V_1 = \langle x_1, h_1, x_2, h_2, \dots, x_{k-1}, h_{k-1}, x_k, \circ \rangle$

و نمای دوم: $V_2 = \langle x_r, h_r, x_{r+1}, h_{r+1}, \dots, x_{p-1}, h_{p-1}, x_p, \circ \rangle$ باشد.

ادغام V_1 و V_2 درست مانند عمل ادغام دو آرایه‌ی مرتب در MergeSort است:

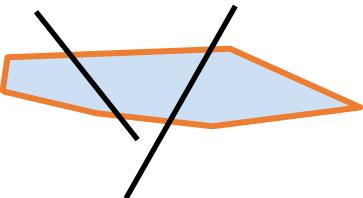
Convex Hull Problem

- Given a set of pins on a pinboard and a rubber band around them.
How does the rubber band look when it snaps tight?
- The convex hull of a point set is one of the simplest shape approximations for a set of points.

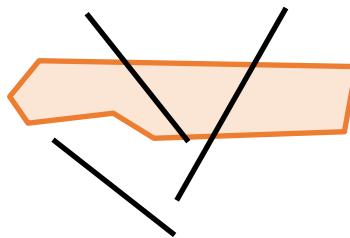


Convexity

- A set $C \subseteq \mathbf{R}^2$ is *convex* if for all two points $p, q \in C$ the line segment \overline{pq} is fully contained in C .



convex

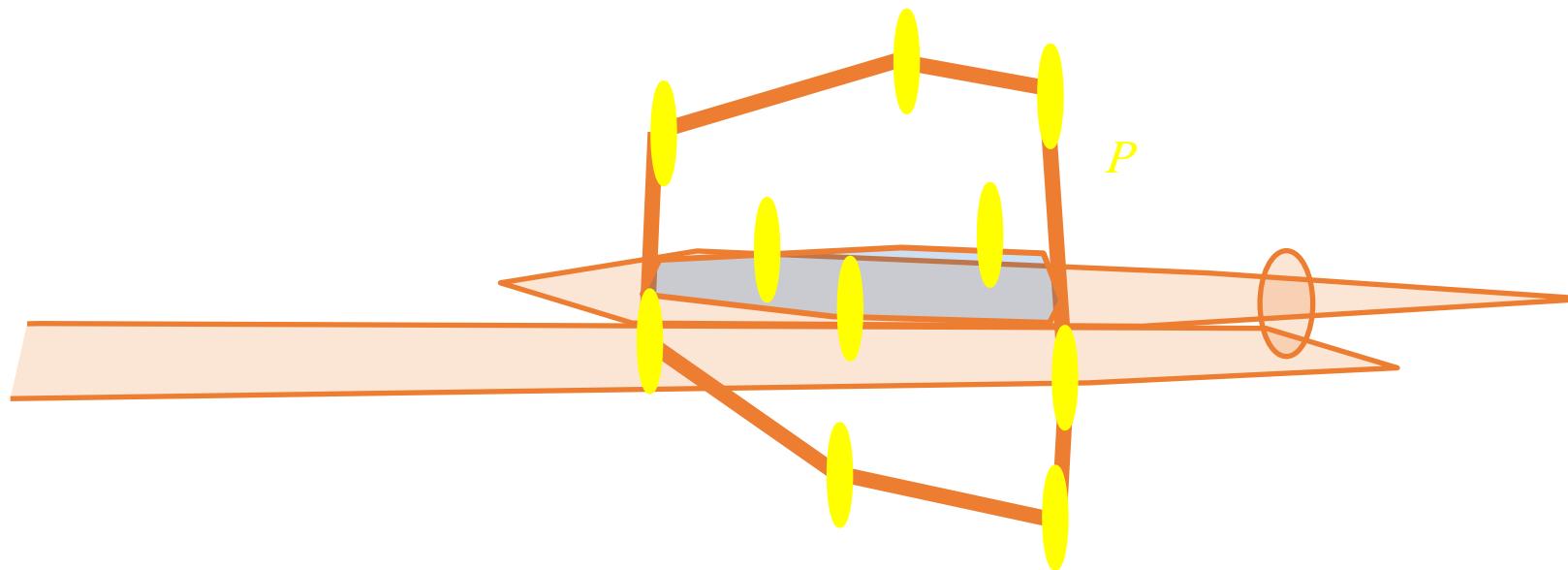


non-convex

Convex Hull

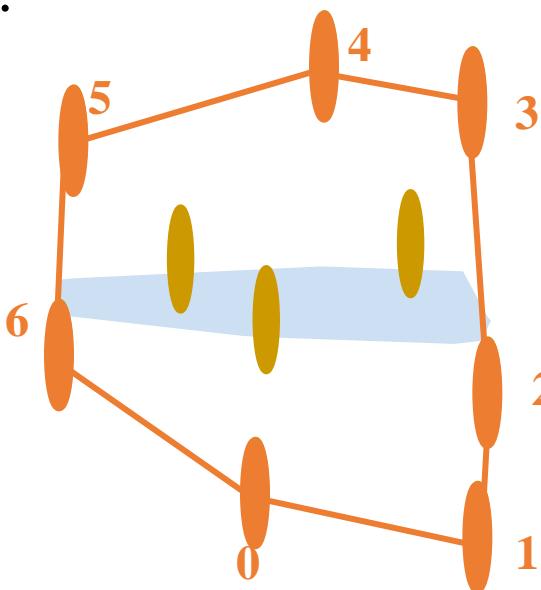
- The convex hull $CH(P)$ of a point set $P \subseteq \mathbf{R}^2$ is the smallest convex set $C \supseteq P$. In other words $CH(P) = \cap C$.

$$\begin{matrix} C \supseteq P \\ C \text{ convex} \end{matrix}$$



Convex Hull

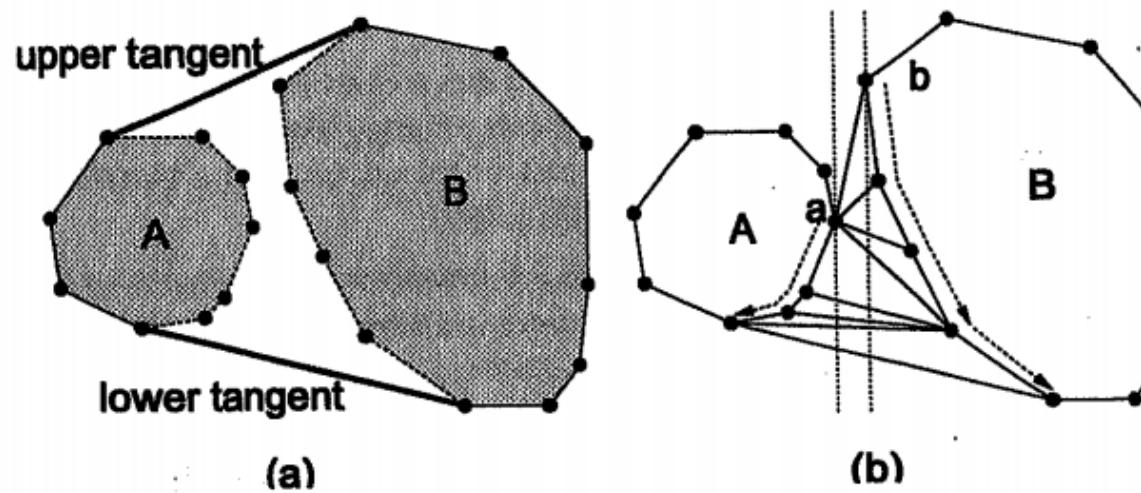
- **Observation:** $\text{CH}(P)$ is the unique convex polygon whose vertices are points of P and which contains all points of P .
- We represent the convex hull as the sequence of points on the convex hull polygon (the boundary of the convex hull), in counter-clockwise order.



Convex Hull D&C

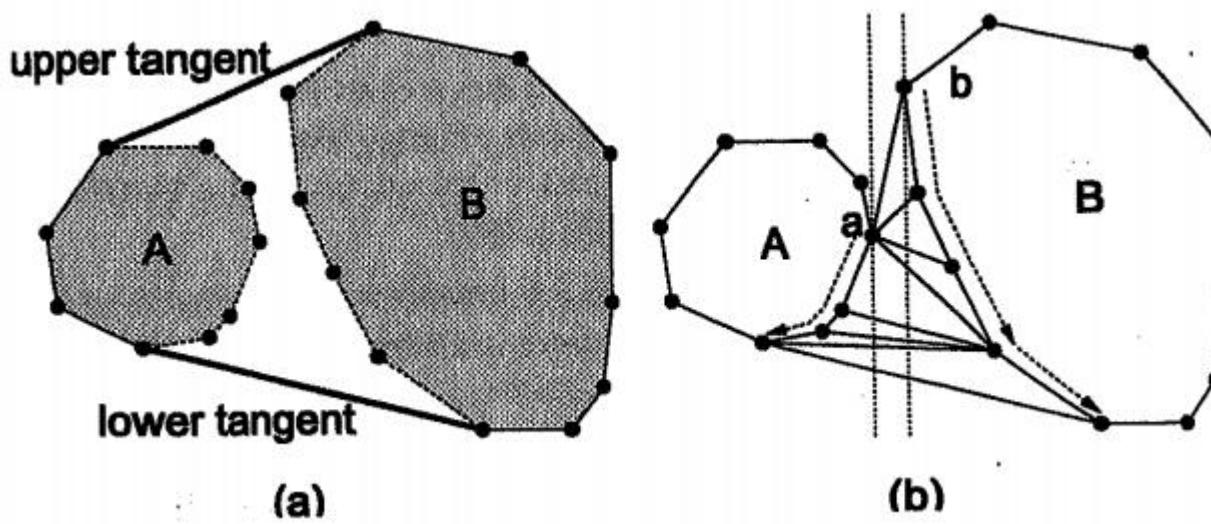
Divide-and-Conquer Convex Hull

-
- (1) If $|P| \leq 3$, then compute the convex hull by brute force in $O(1)$ time and return.
 - (2) Otherwise, partition the point set P into two sets A and B , where A consists of half the points with the lowest x -coordinates and B consists of half of the points with the highest x -coordinates.
 - (3) Recursively compute $H_A = \mathcal{CH}(A)$ and $H_B = \mathcal{CH}(B)$.
 - (4) Merge the two hulls into a common convex hull, H , by computing the upper and lower tangents for H_A and H_B and discarding all the points lying between these two tangents.
-



LowerTangent(H_A, H_B) :

- (1) Let a be the rightmost point of H_A .
- (2) Let b be the leftmost point of H_B .
- (3) While ab is not a lower tangent for H_A and H_B do
 - (a) While ab is not a lower tangent to H_A do $a = a - 1$ (move a clockwise).
 - (b) While ab is not a lower tangent to H_B do $b = b + 1$ (move b counterclockwise).
- (4) Return ab .



زمان‌بندی دوره‌ی بازی‌ها

n تیم در یک بازی دوره‌ای برنامه‌ی بازی‌ها با حداقل مدت را طوری طراحی کنید که در آن

- هر تیم با بقیه‌ی تیم‌ها بازی کند،
- هر تیم در هر روز بیش از یک بازی انجام ندهد

تعداد کل بازی‌ها: $n(n - 1)/2$

در هر روز حداقل تعداد بازی‌ها: $\lfloor \frac{n}{2} \rfloor$

↔ حد پایین تعداد روزهای بازی:

اگر n زوج باشد برابر $1 - n$ و اگر n فرد باشد برابر n روز است.

تعیین مقدار آستانه

- سربرار کامپیوتروی فرایнд بازگشتی
- فرض $n=8$ آیا این سربرار ارزش این را دارد که به جای الگوریتم n^2 از الگوریتم بازگشتی $n^* \log n$ استفاده کنیم.
- یک مقدار بهینه از ورودی تعیین می کنیم که به ازای همه نمونه های ورودی کوچکتر از آن به جای استفاده از الگوریتم بازگشتی از یک الگوریتم دیگر استفاده کنیم.

تعیین حد آستانه به روش غلط مرتب سازی ادغامی و تعویضی

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

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

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

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

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

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

The solution is

$$n < 257.$$

تعیین حد آستانه به روش درست مرتب سازی ادغامی و تعویضی

$$T=128 \bullet$$

$$T=128.008 \bullet$$

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

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

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

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

حد آستانه دو الگوریتم تقسیم غلبه و \mathcal{O} به توان ۲

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

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

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

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

If we substitute an even value for t (by setting $\lceil t/2 \rceil = t/2$) and solve, we get

$$t = 64.$$

If we substitute an odd value for t (by setting $\lceil t/2 \rceil = (t+1)/2$) and solve, we get

$$t = 70.04.$$

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

کجا نمی توان از الگوریتم تقسیم و غلبه استفاده کرد؟

- نمونه ای با اندازه n به دو یا چند نمونه با اندازه های تقریبا n تقسیم شود.
- نمونه ای با اندازه n به n نمونه با اندازه های C یا مقدار ثابت تقسیم شود.