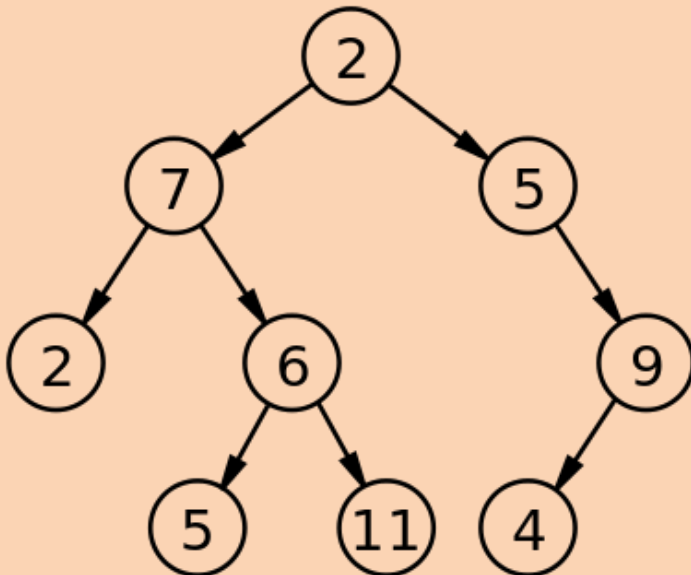




# ساختارهای داده

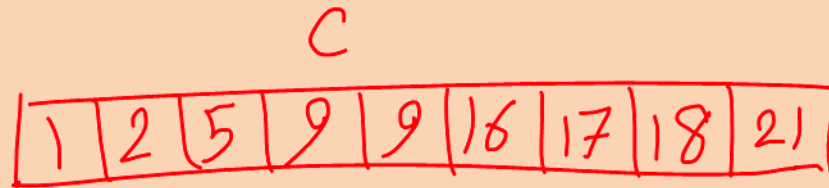
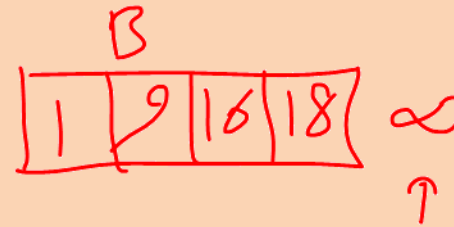
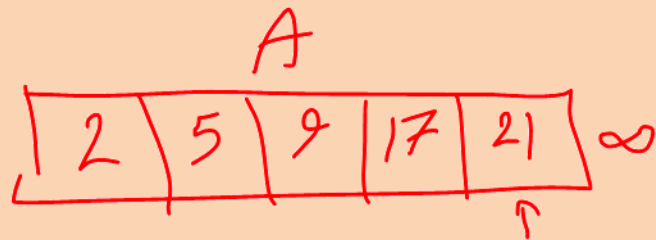
## مرتب‌ساز ادغامی *Merge Sort*



مدرس:

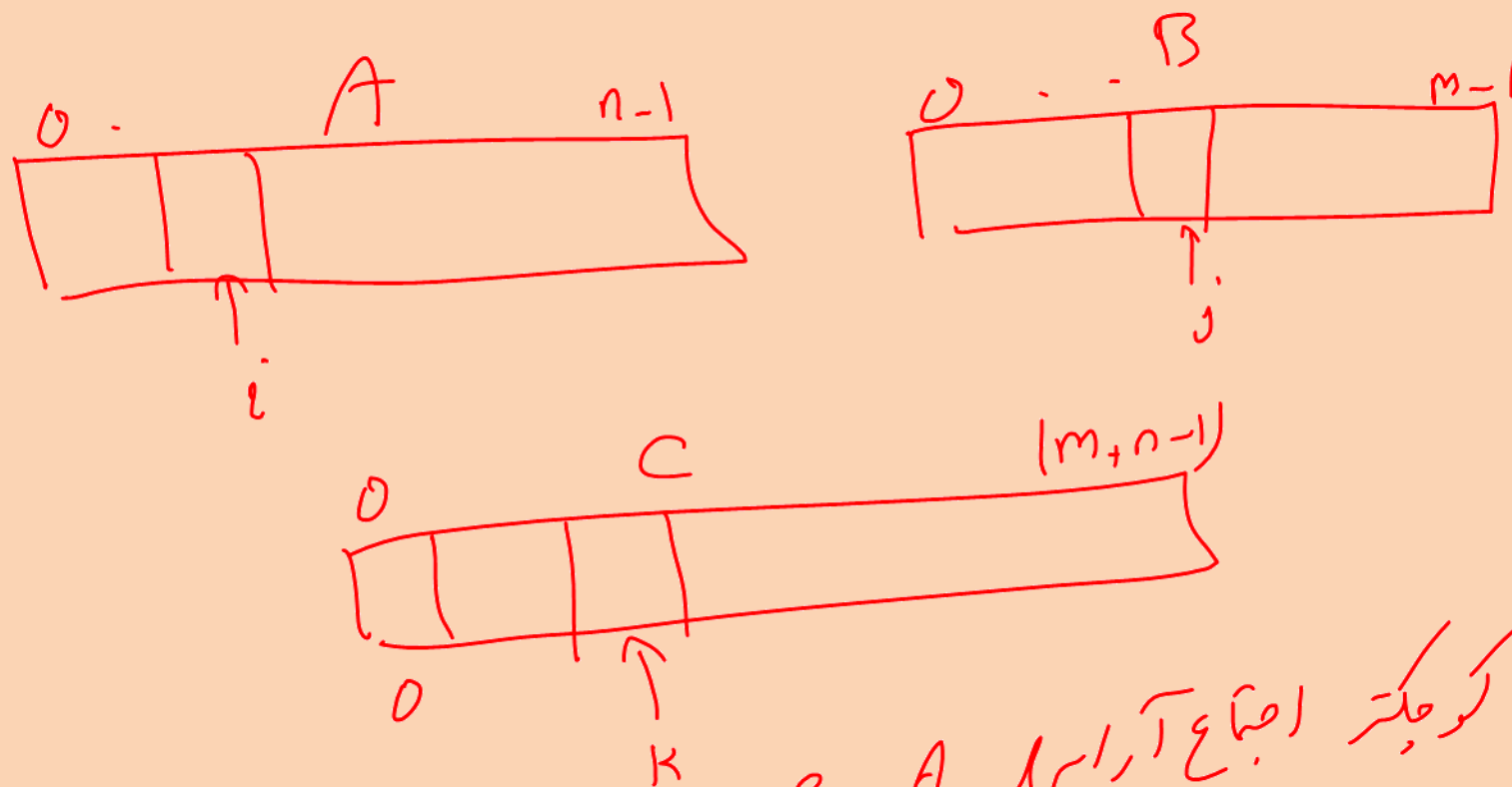
سید کمال الدین غیاثی شیرازی

# ایده اصلی: ادغام دو لیست مرتب



در یادگیری  
به صورت  
template  
درجایته کل می شود

# الگوریتم ادغام (Merge)



همواره  $k$  عنصر کوچکتر اجتماع آرایه‌های  $A$  و  $B$  در عناصر  $0$  تا  $i$  آرایه  $A$  و  $0$  تا  $j-1$  آرایه  $B$  هستند که به صورت مرتب در  $k$  عنصر اول آرایه  $C$  قرار گرفته‌اند.

# کد الگوریتم ادغام (Merge)

$i=0; j=0; k=0; \text{ // ( } m:\text{size } B[], n:\text{size } A[], \text{ size of } C \text{ is } n+m)$

$\text{while } (i < n \ \&\& \ j < m)\{$

$\text{if } (A[i] \leq B[j])$

$\{C[k] = A[i]; i++; k++;\}$

$\text{else}$

$\{C[k] = B[j]; j++; k++;\}$

$\}$

$\text{while } (i < n)$

$\{C[k] = A[i]; i++; k++;\}$

$\text{while } (j < m)$

$\{C[k] = B[j]; j++; k++;\}$

$$T(p \wedge q) \equiv T p \vee T q$$

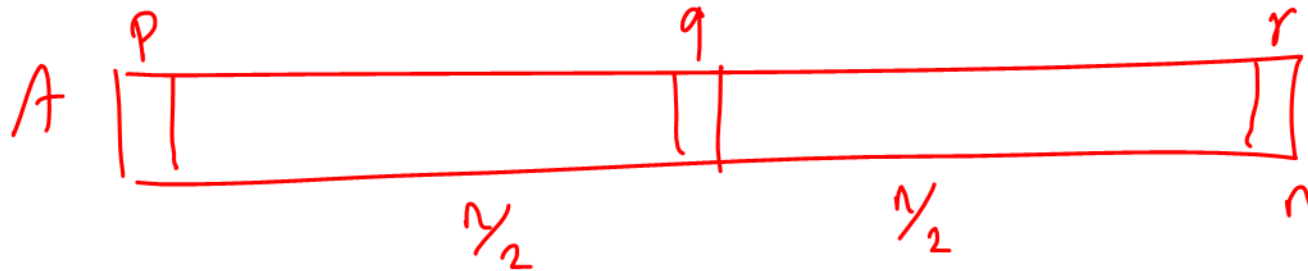
صحیح زبان الگوریتم ادغام A با n عنصر  
B با m عنصر

زبان اجرا = تعداد گام = تعداد دفعات  
انتراسی  $n$

$$\Rightarrow \text{زبان اجرا} = O(n + m)$$

# Merge Sort

$$q = \left\lfloor \frac{p+r}{2} \right\rfloor$$



if  $r > p$

$$\left\{ \begin{array}{l} q = \left\lfloor \frac{p+r}{2} \right\rfloor \end{array} \right.$$

MergeSort( $A, p, q$ )

MergeSort( $A, q+1, r$ )

Merge( $A, p, q, r$ )

$A[p \dots q]$   
 $A[q+1 \dots r]$

}

// else  
آرایه منفرجه یک عنصری  
خود به خود مرتب است

کار استقرای

# الگوریتم بازگشتی مرتب‌سازی ادغامی

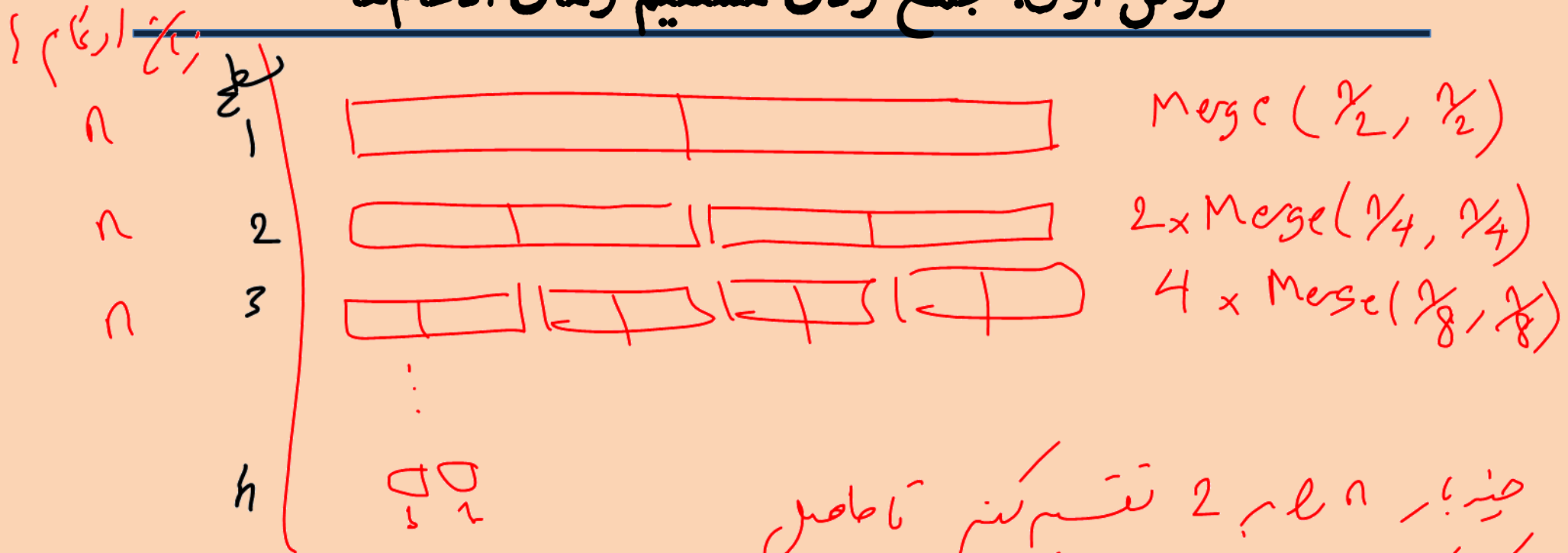
---

*int\* A*

```
void MergeSort(int A[], int p, int r) {  
    if (r > p) {  
        int q = (p + r) / 2;  
        MergeSort (A, p, q);  
        MergeSort (A, q+1, r);  
        Merge(A, p, q, r);  
    }  
}
```

# محاسبه پیچیدگی زمانی MergeSort

روش اول: جمع زدن مستقیم زمان ادغامها



چون  $n$  به  $2$  تقسیم می‌شود تا حاصل  
کوچک‌تر از  $1$  شود.

$$1, 2, 4, 8, \dots, 2^h \geq n$$

$$2^{h-1} < n \leq 2^h$$

$$h-1 < \log_2 n \leq h \quad h = \lceil \log_2 n \rceil$$

$$\begin{aligned} h \times n &= \text{تعداد ادغامها} \\ &\in \Theta(n \log n) \end{aligned}$$

# به دست آوردن رابطه بازگشتی پیچیدگی زمانی MergeSort

$$n = r - p + 1$$

$$T(n) = \begin{cases} 1 & n \leq 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \theta(n) & n > 1 \end{cases}$$

Recursion tree for MergeSort:

```
graph TD
    A["MergeSort(A, p, r)"] --> B["if p < r"]
    B --> C["q = floor((p+r)/2)"]
    B --> D["MergeSort(A, p, q)"]
    B --> E["MergeSort(A, q+1, r)"]
    C --> F["Merge(A, p, q, r)"]
    D --> F
    E --> F
    F --> G[""]
```

Complexity analysis of the recursion tree:

- Level 0:  $\theta(1)$  (circled)
- Level 1:  $\theta(n)$  (circled)
- Level 2:  $\theta(n)$  (circled)



# محاسبه پیچیدگی زمانی MergeSort

روش دوم: حل رابطه بازگشتی

$$T(n) = \begin{cases} 1 & n \leq 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & n > 1 \end{cases}$$

فرض کنیم  $n = 2^m$  و تقریباً رکشم

$$a_m = T(2^m)$$

$$a_m = T(2^m) = \begin{cases} 1 & m = 0 \\ a_{m-1} + a_{m-1} + 2^m & m > 0 \end{cases}$$

$$a_m = 2a_{m-1} + 2^m$$

$$a_m = 2a_{m-1} + \underbrace{2^m}$$

$$r-2=0$$

حاله سلف

$$a_m = (A_m + B)2^m$$

نرمالیزاسیون

$$(\cancel{A_m} + \cancel{B})2^{\cancel{m}} = (\cancel{A(m-1)} + \cancel{B})2^{\cancel{m}} + \underbrace{2^{\cancel{m}}}_1$$

$$-A + 1 = 0 \Rightarrow \boxed{A=1}$$

$$a_0 = 1 \quad (A \cdot 0 + B)2^0 = 1 \Rightarrow \boxed{B=1}$$

$$a_m = (m+1) \frac{2^m}{n} = n (\log_2 n + 1) \in \Theta(n \log n)$$

## قضیه اصلی (Master Method)

- فرض کنیم زمان اجرای یک تابع بازگشتی برابر  $T(n)$  است و در رابطه زیر صدق می کند:

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

$d$        $\log_b a$

- در این صورت داریم:

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

لازم است قضیه اصلی را حفظ باشید

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

$$\leq 2T(n/2) + O(n)$$

$$d = \log_b a$$

$$T(n) \in O(n \log n)$$

# مبنای لگاریتم در تابع $n \log(n)$

$$n \log n$$

10, 2, e, ...

---

> 1

$$\log_a n = \frac{\log_b n}{\log_b a}$$

فرض ثابت  $b$  = ثابت

$$\log_a n = \log_b n \cdot \log_a b$$

فرض ثابت  $b$  = ثابت

# محاسبه پیچیدگی زمانی MergeSort

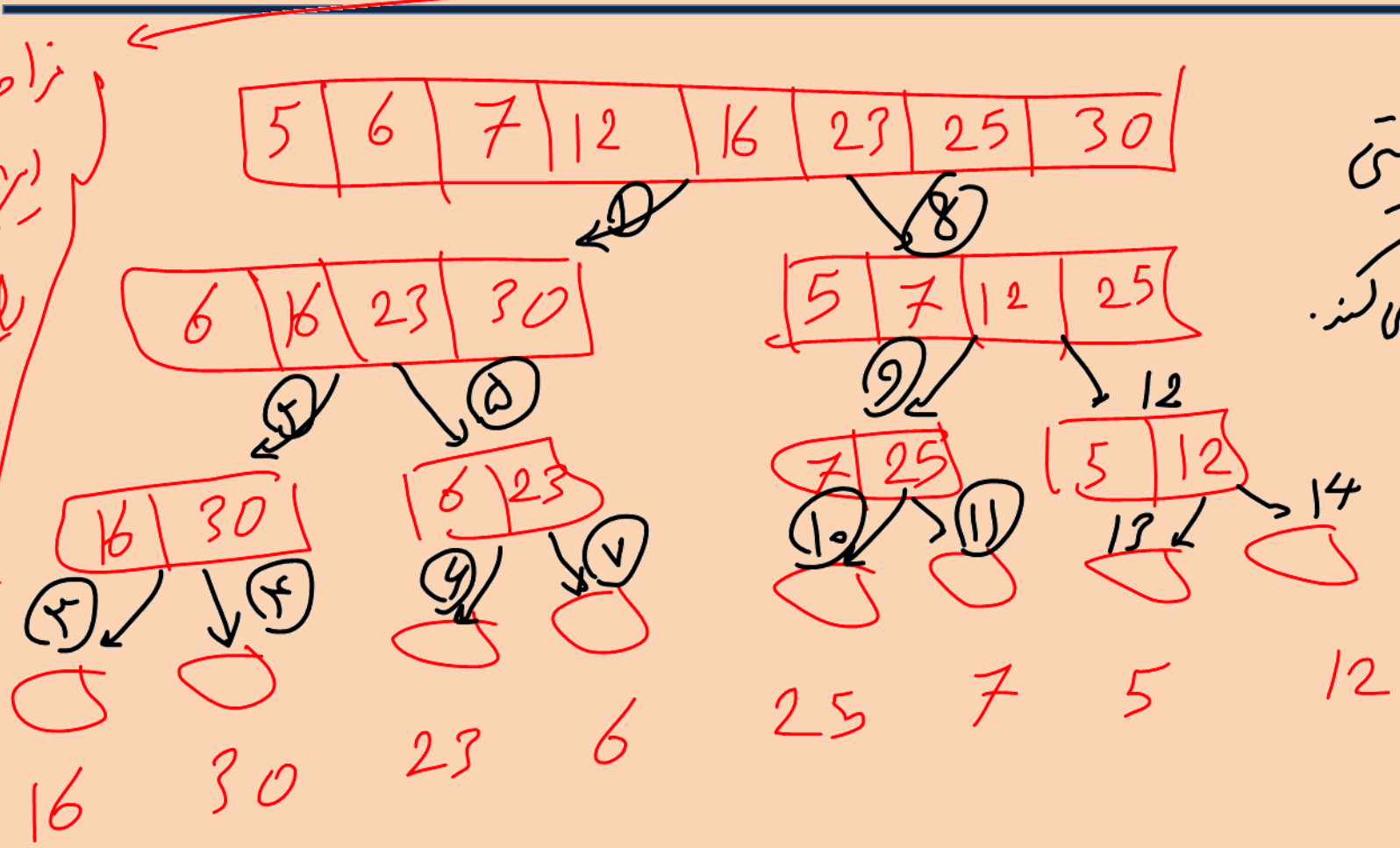
## روش سوم: استفاده از قضیه اصلی

---

# نسخه غیر بازگشتی مرتب سازی ادغامی

مراجعات  
ایستادن  
در منزل

نسخه بازگشتی  
درین عمل مرگند



ستاره! فقط فراخوانی MergeSort و بازگشت

$i=0$

$b=1$

16	30	23	6	25	7	5	12
----	----	----	---	----	---	---	----

Arrows point from indices 0, 2, 4, 6 to the corresponding elements in the array.

$i=0$

$b=2$

16	30	6	23	7	25	5	12
----	----	---	----	---	----	---	----

Brackets group the first four elements (16, 30, 6, 23) and the last four elements (7, 25, 5, 12).

$i=0$

$b=4$

6	16	23	30	5	7	12	25
---	----	----	----	---	---	----	----

A single bracket groups all eight elements of the array.

$b=8$

5	6	7	12	16	23	25	30
---	---	---	----	----	----	----	----

ارغام بزرگ 1  
باطل 1

ارغام بزرگ 2  
باطل 2

ارغام بزرگ 4  
باطل 4



# الگوریتم مرتب سازی ادغامی غیر بازگشتی

$\theta(\log(n))$

$f_{0-} (b=1; b < n; b*=2)$   $b=2^{h+1}$   $2^h < n \leq 2^{h+1}$

$\leftarrow f_{0-} (i=0; i < n; i+=2*b)$   $h+1 = \lceil \log_2 n \rceil$

$e_1 = \min(n, i+b)$

$e_2 = \min(n, i+2b)$

$\leftarrow \text{merge}(A, i, e_1, e_2)$   $A[i \dots e_1]$   
 $A[e_1+1 \dots e_2]$

$\log(n) \times \frac{n}{2^b} \times 2b = n \log(n)$

# پیچیدگی زمانی الگوریتم مرتب‌سازی ادغامی غیر بازگشتی

---

$$n \log(n)$$

هر اسلاید تبدیل لازم

## مرتب‌سازی ادغامی طبیعی (Natural MergeSort)

16, 30, 23, 6, 25, 7, 5, 12, 8, 14, 15, 21, 2, 24, 13, 22



به جای  $16 \times 4$  عمل فقط  $3 \times 16$  عمل انجام رد می‌دهیم.  
هزینه اضافی ۱۶

# C++ STL stable\_sort Function

---

- مرتب‌سازی ادغامی پایدار است.  $\theta(n \log n)$
- برای  $n$  های کوچک ( $n \leq 16$ ) مرتب‌سازی درجی سریع‌تر از مرتب‌سازی ادغامی است.  $\theta(n^2)$
- `stable_sort` در C++ STL برای  $n$  های کوچک به الگوریتم مرتب‌سازی درجی برمی‌گردد.