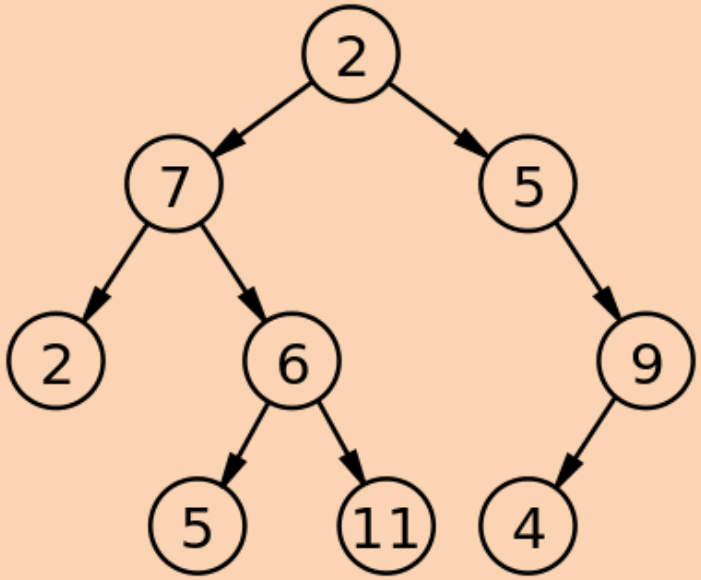
# ساختمان‌های داده

## مرتب‌سازی ادغامی

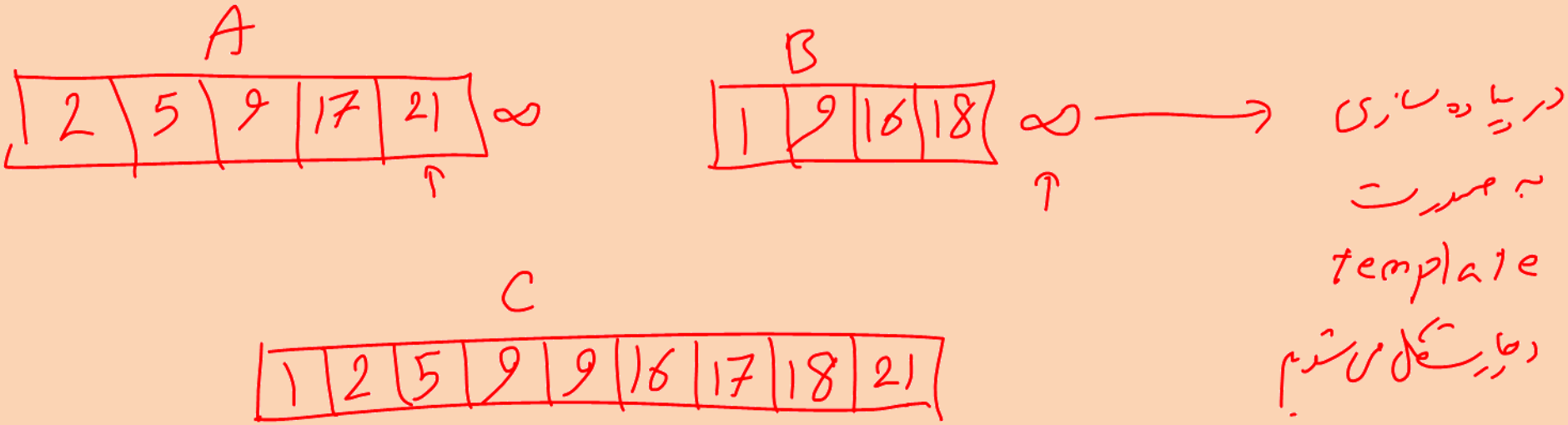
## MergeSort



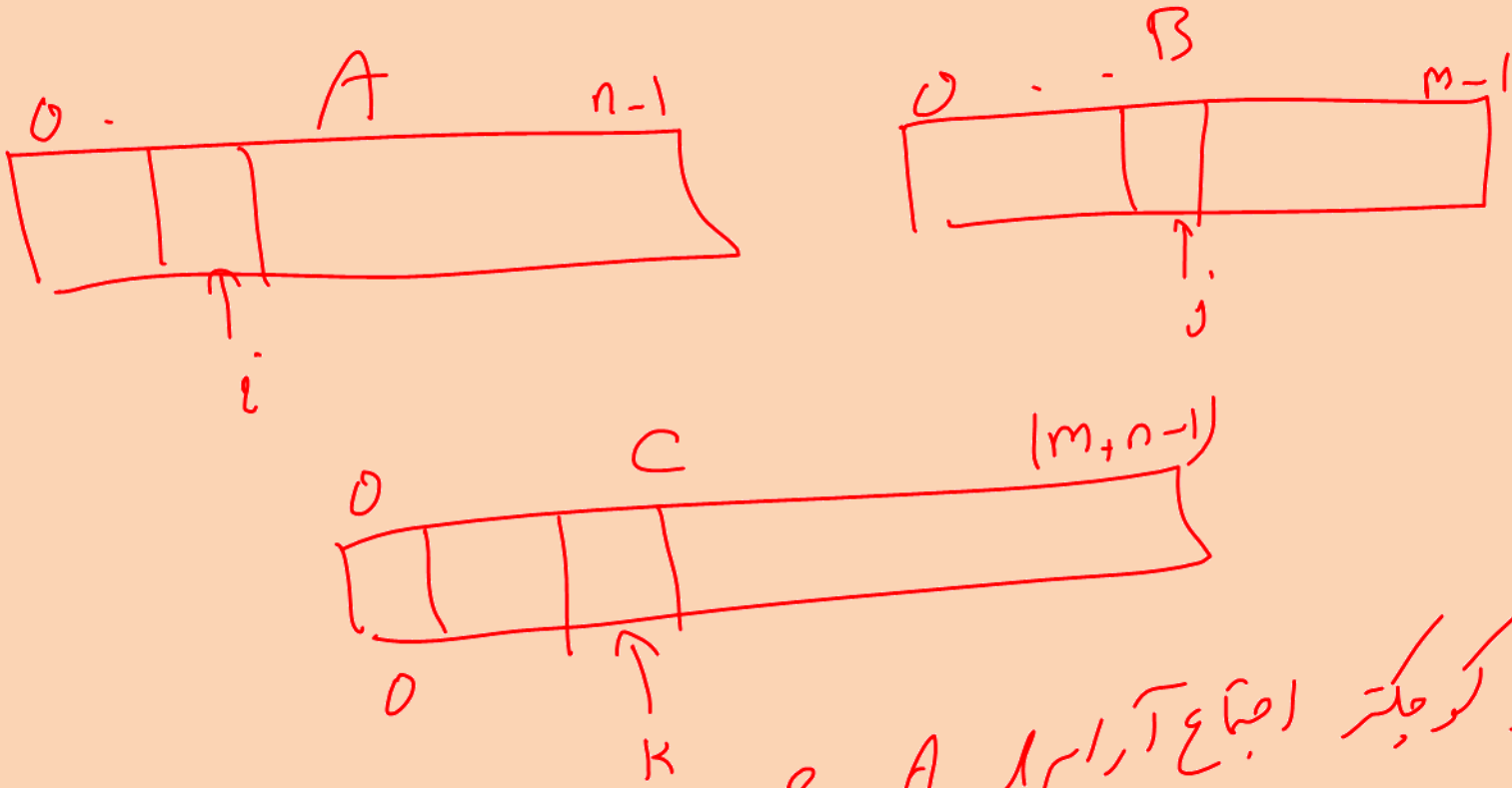

مدرس:

سید کمال الدین غیاثی شیرازی

# ایده اصلی: ادغام دو لیست مرتب



# الگوریتم ادغام (Merge)




همواره k عضو کوچکتر اجتماع آرایه‌های A و B در عناصر 0 تا n-1 آرایه A و 0 تا m-1 آرایه B هستند که به صورت مرتب در k عضو اول آرایه C قرار گرفته‌اند.

# کد الگوریتم ادغام (Merge)

```
i=0; j=0; k=0;  //( m:size B[], n:size A[], size of C is n+m)
while (i < n && j < m){
      if (A[i] <= B[j])
          {C[k] = A[i]; i++;k++;}
      else
          {C[k] = B[j];j++;k++;}
}
while (i < n)
          {C[k] = A[i];i++;k++;}
while (j < m)
          {C[k] = B[i];j++;k++;}
```

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

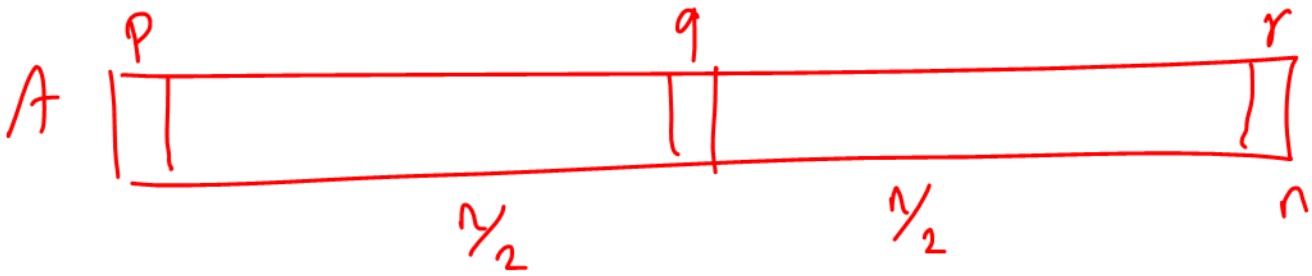
بهترین حالت زمانی الگوریتم ادغام A با n عنصر و B با m عنصر

زمان اجرا = تعداد گام ها = تعداد دفعات افزایش k

$\Rightarrow$ زمان اجرا $= \Theta(n+m)$

# Merge Sort

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




```
if r > p
{
    q = ⌊(p+r)/2⌋
    MergeSort(A, p, q)
    MergeSort(A, q+1, r)
    Merge(A, p, q, r)        A[p..q]  A[q+1..r]
}
// else
   آرایه صفر/یک عضوی
   خود به خود مرتب است
```

} بازگشتی

# 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


حینی که $n$ را $2$ تقسیم کنیم تا حاصل کوچک‌تر مساوی $1$ شود!

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

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

$h-1 < \log_2 n \leq h$

$h = \lceil \log_2 n \rceil$

$h \times n$ = زمان ادغام در هر سطح

$\in \Theta(n \log n)$

# به دست آوردن رابطه بازگشتی پیچیدگی زمانی MergeSort

$n = r - p + 1$

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

```
MergeSort(A, p, r)
if p < r
    q = ⌊(p+r)/2⌋          ← θ(1)
    MergeSort(A, p, q)     ⌊n/2⌋
    MergeSort(A, q+1, r)   ⌈n/2⌉
    Merge(A, p, q, r)      ← θ(n)
                           ← θ(n)
```

# محاسبه پیچیدگی زمانی MergeSort

## روش دوم: حل رابطه بازگشتی

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

فرض کنیم $n = 2^m$ و تعریف کنیم $a_m = T(2^m)$

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

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

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

معادله تحقیقی

$r - 2 = 0$

فرم کلی پاسخ:

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

$$(Am + B)2^m = (A(m-1) + B)2^m + 2^m$$

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

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

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

# قضیه اصلی (Master Method)

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

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

- در این صورت داریم:

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

لازم است قضیه اصلی را حفظ باشیم

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

$$\leq 2T(n/2) + O(n^1)$$

$$d = \log_b a$$

$$1 = 1$$

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

# مبنای لگاریتم در تابع $n\,log(n)$

$n \log n$

$10, 2, e$ ؟ نسبت $> 1$

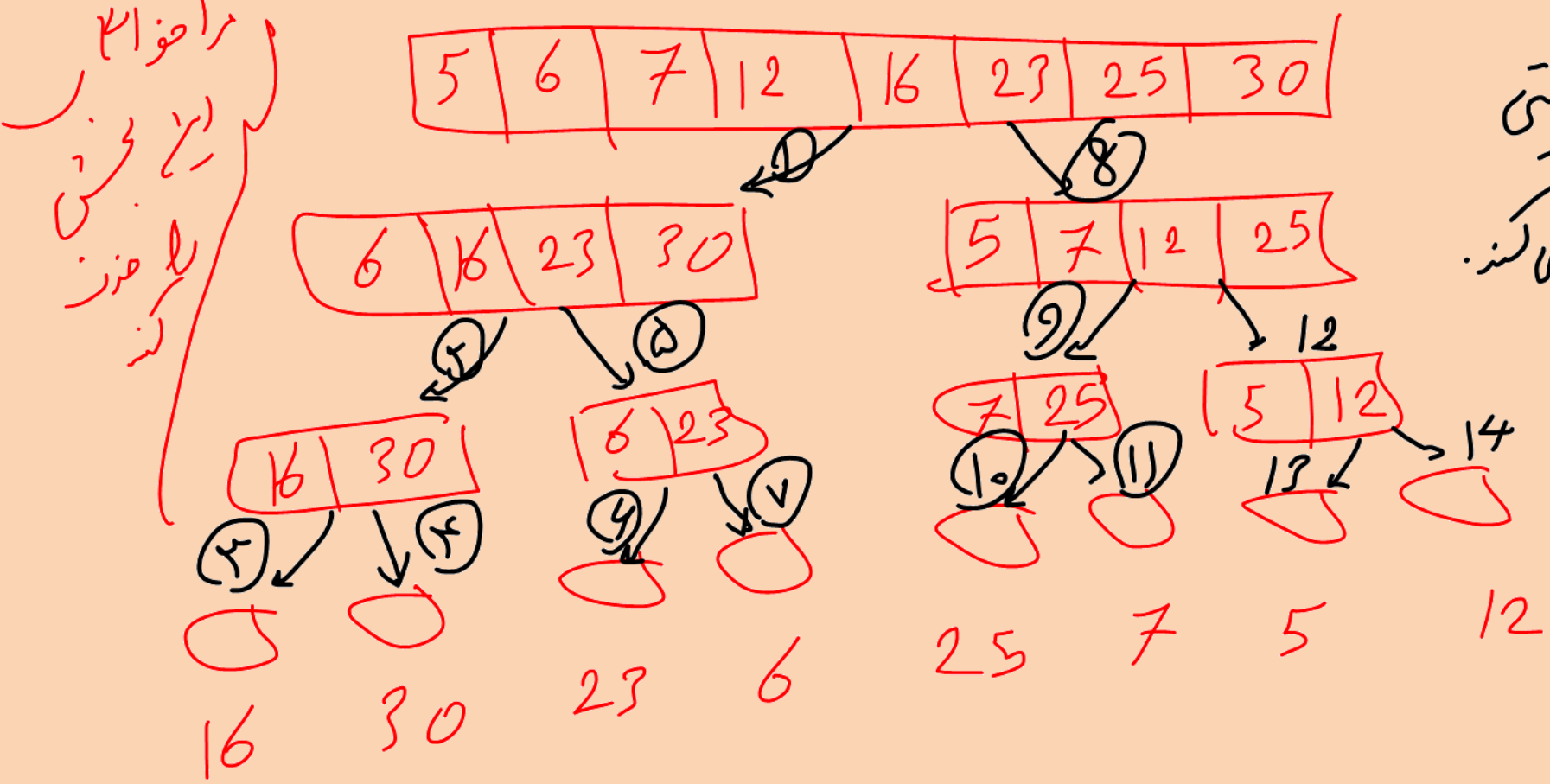
$\log_a n = \frac{\log_b n}{\log_b a} > 0$ = ضریب ثابت مثبت $\log_b n$

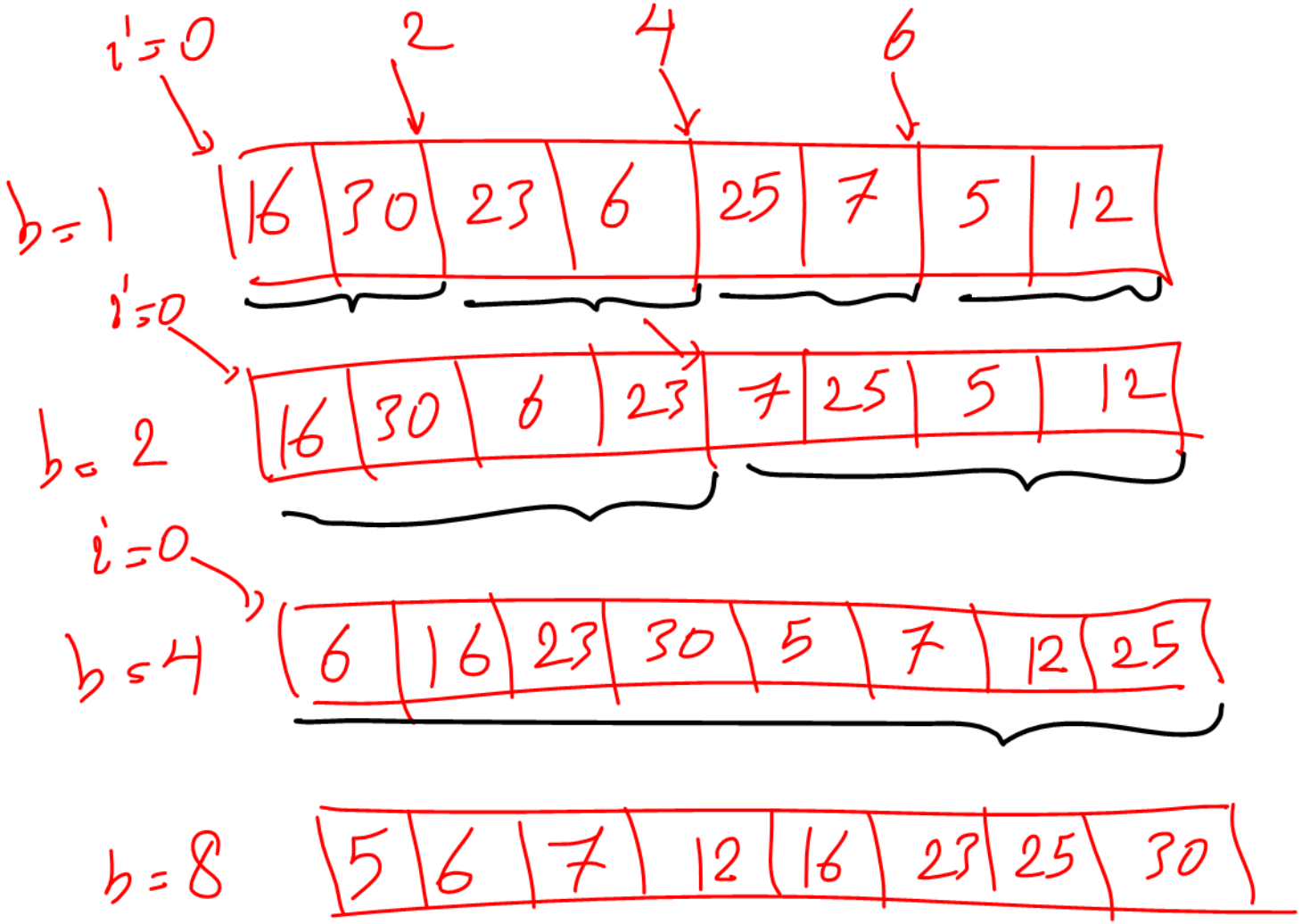
$\log_a n = \log_b n \; (\log_a b)$

ضریب ثابت مثبت

# محاسبه پیچیدگی زمانی MergeSort
## روش سوم: استفاده از قضیه اصلی

# نسخه غیر بازگشتی مرتب‌سازی ادغامی






ادغام بلاک ۱
با طول 1

ادغام بلاک ۲
با طول 2

ادغام بلاک ۳
با طول 4

# الگوریتم مرتب‌سازی ادغامی غیر بازگشتی

$\theta(\log(n))$

for (b = 1; b < n; b *= 2) ← $b = 2^{h+1}$ ; h: تعداد دفعات تکرار

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

$h+1 = \lceil \log_2 n \rceil$

for (i = 0; i < n; i += 2*b) ← $n/2b$: تعداد دفعات تکرار

{

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

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

merge(A, i, $e_1$, $e_2$) ⇐ برای دو آرایه با طول b

$A[i..e_1]$

$A[e_1+1..e_2]$

}

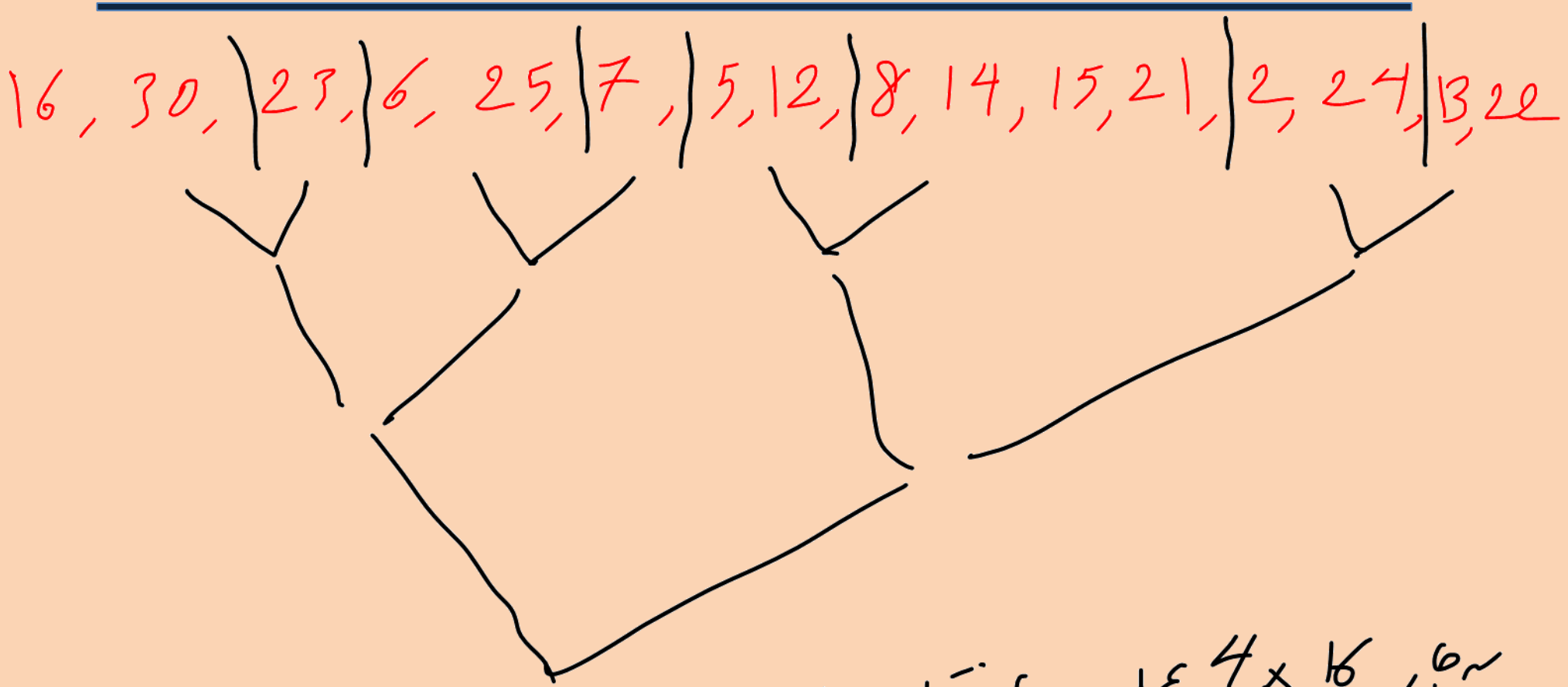
زمان کل: $\log(n) \times \frac{n}{2b} \times 2b = n\log(n)$

# پیچیدگی زمانی الگوریتم مرتب‌سازی ادغامی غیر بازگشتی

$n \log(n)$

مرا سلایر تبدیل خواهیم

# مرتب‌سازی ادغامی طبیعی (Natural MergeSort)




به جای 16 x 4 عمل تا فقط 16 x 3 عمل انجام دهیم.

هزینه اضافی، 16

# C++ STL stable_sort Function

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