3.3 Sorting

- Problem:
  - Arrange the elements of a set in an ascending (or descending) order
  - [2, 1, 4, 3, 7, 5, 6] ➔ [1, 2, 3, 4, 5, 6, 7]
- Sorting algorithms
  - O (n^2) algorithms for pairwise comparison
    - Bubble sort
    - Insertion sort
    - Selection sort
    - ……
  - Can it be faster? → Use divide & conquer
    - Merge sort
    - Quick sort

3.3 Sorting

- Merge sort
  - Given a sequent of n elements { a_1, a_2, …, a_n }, split them into two set.
  - Sort each set individually and merge them to produce a single sorted sequence of n elements.
  - Three steps
    - Divide
      - Split the list into two halves
    - Conquer
      - Recursively sort each half
    - Combine
      - Merge the two sorted sublists

- Merge sort (Divide & Conquer)

```
void msort( int s, int e, int A[] )
{
    if ( s == e )
        return;
    m ← (s+e)/2;
    msort ( s, m, A );
    msort ( m+1, e, A );
    merge ( s, m, m+1, e, A );
}
```

- Merge sort (Divide & Conquer)

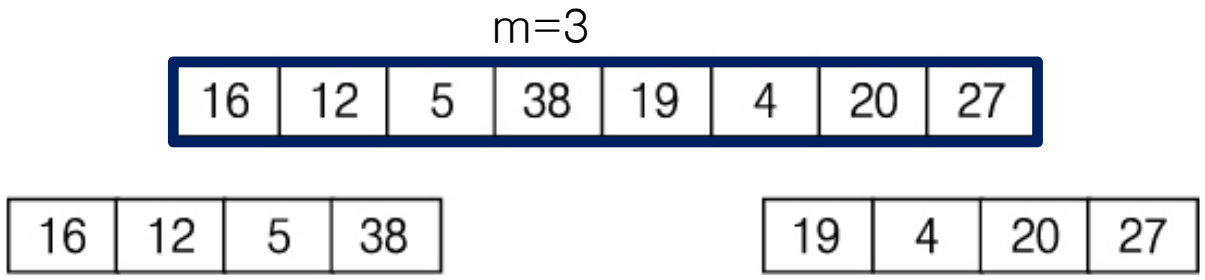
```
void merge( int ls, int le, int rs, int re, int A[] )
{
    int lptr = ls, rptr = rs, bptr = 0;
    int *B = (int *) calloc ((le – ls)+(re – rs)+2, sizeof(int));

    while ( lptr <= le && rptr <= re ) {
        if ( A[lptr] < A[rptr] )
            B[bptr++] = A[lptr++];
        else
            B[bptr++] = A[rptr++];
    }
    if ( lptr > le )
        for ( int i = rptr; i <= re; i++ )
            B[bptr++] = A[i];
    if ( rptr > re )
        for ( int i = lptr; i <= le; i++ )
            B[bptr++] = A[i];
    A ← B;
}
```

- Merge sort (Check three points)

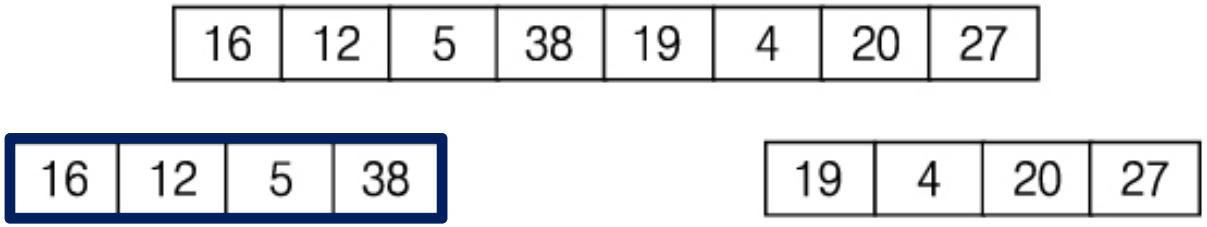
```
void msort( int s, int e, int A[] )
{
    if ( s == e )
        return;
    m ← (s+e)/2;
    msort ( s, m, A );
    msort ( m+1, e, A );
    merge ( s, m, m+1, e, A );
}
```

- Merge sort (Example)

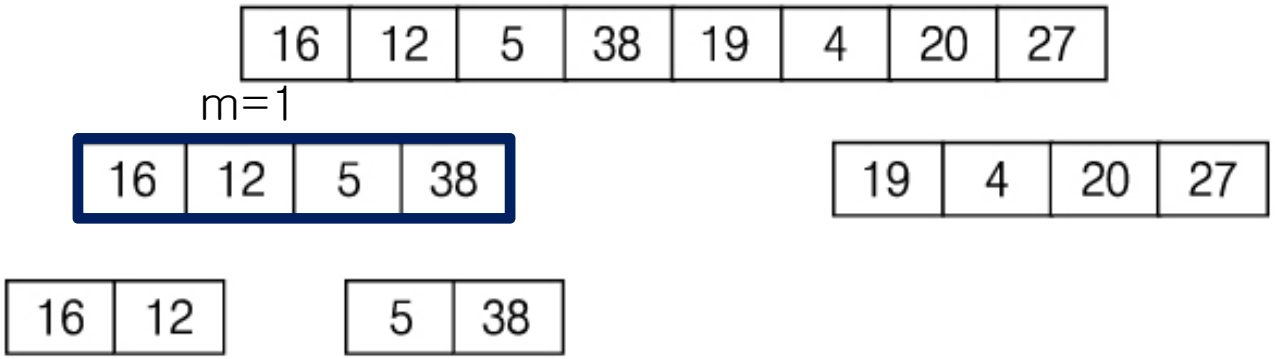
```
merge_sort ( 0, 7, a );
    m = 3;
```



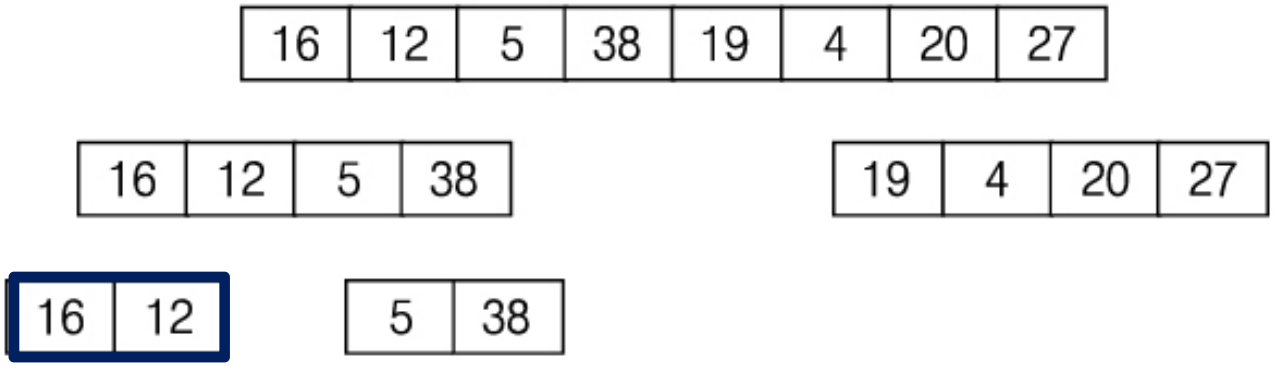
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
```



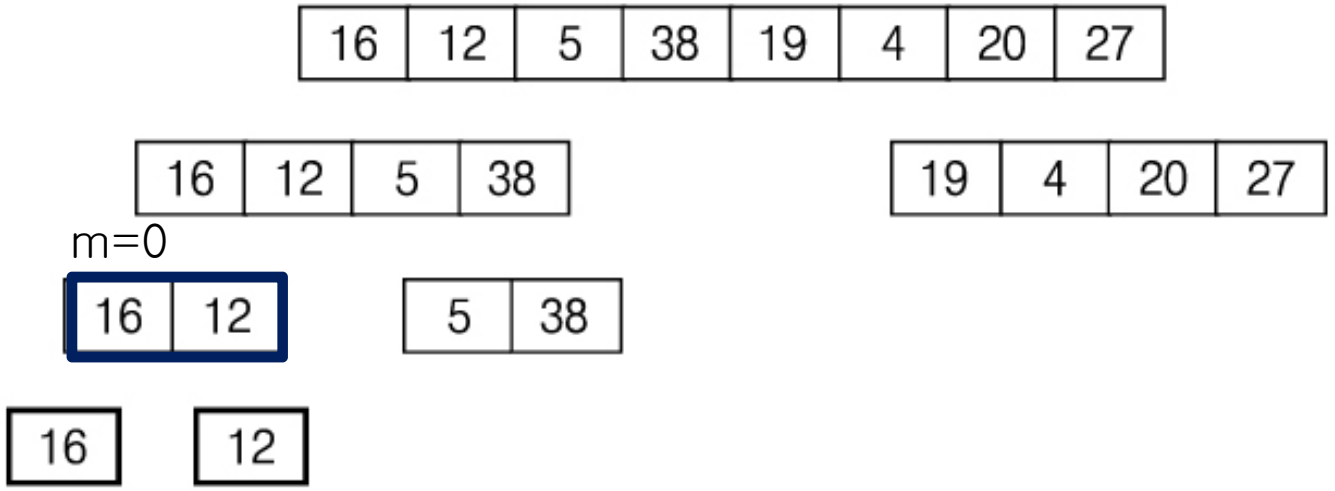
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
```



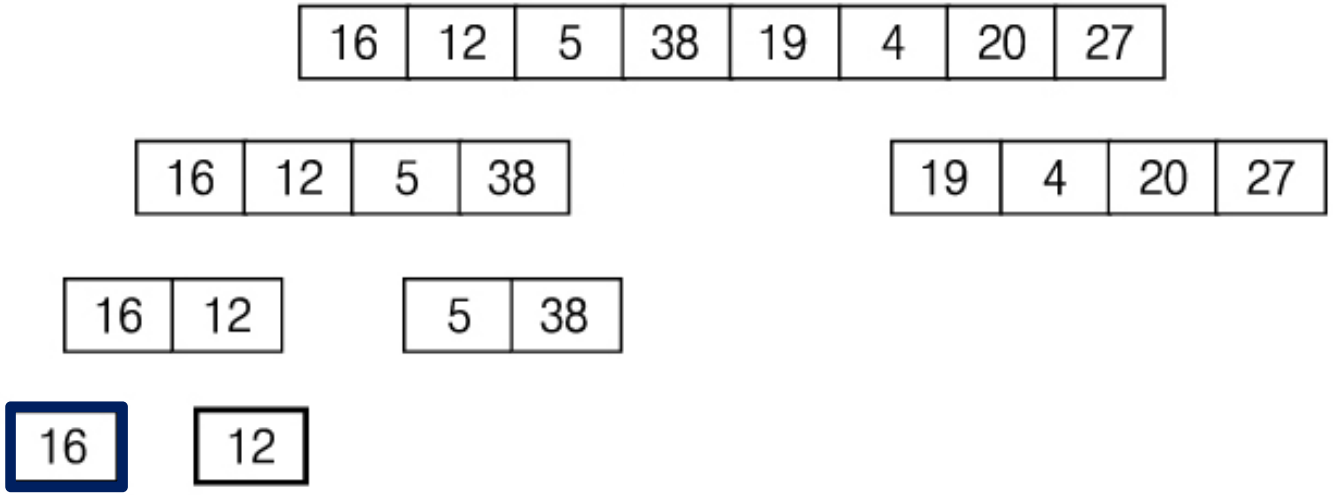
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a );
```



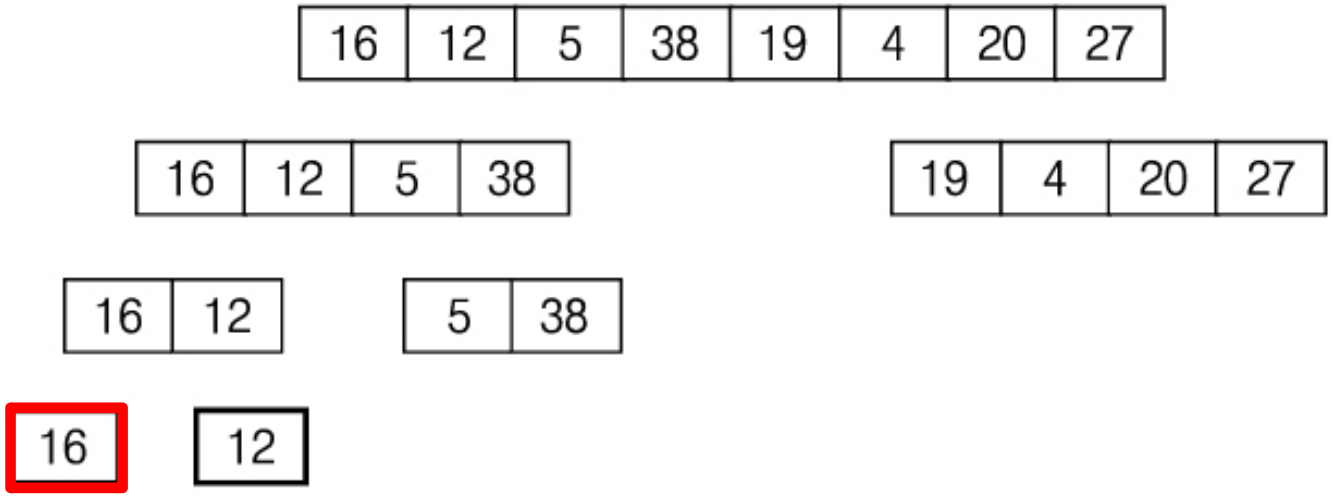
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a );
            m = 0;
```



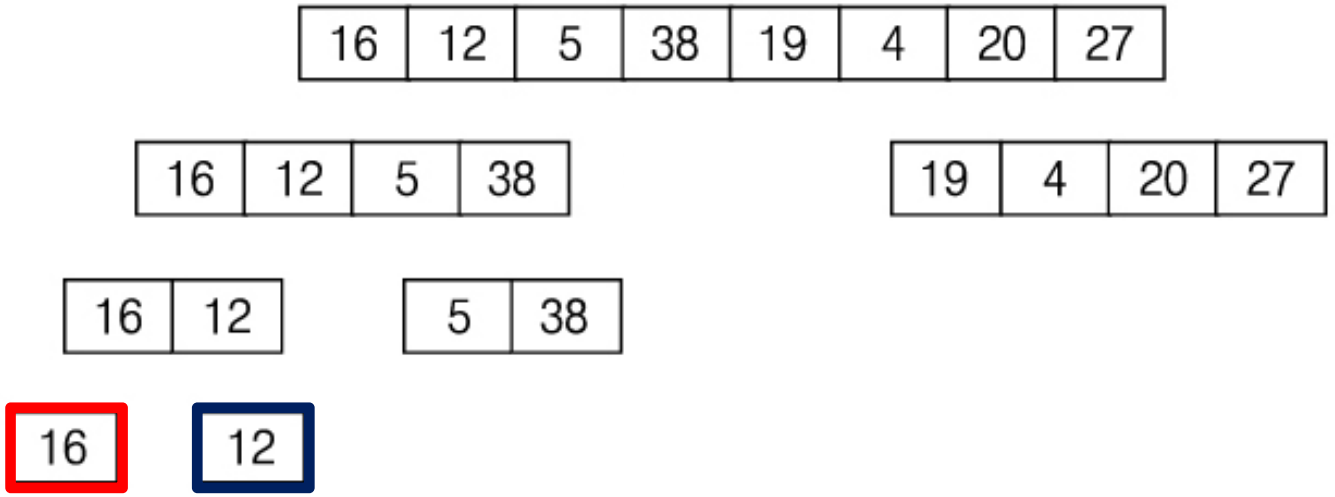
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a );
            m = 0;
            merge_sort ( 0, 0, a );
```



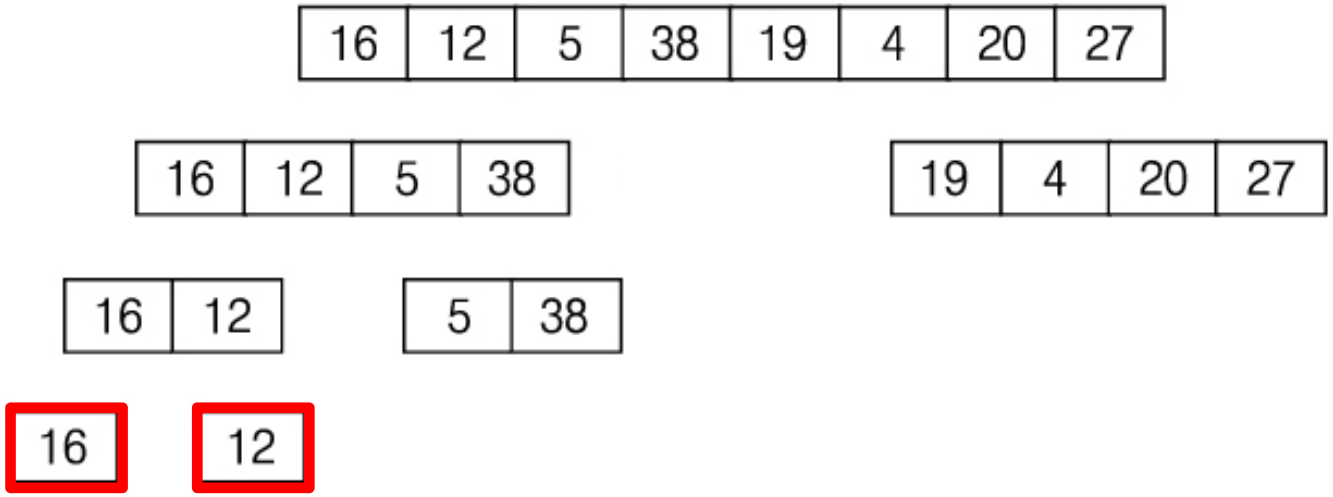
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a );
            m = 0;
            merge_sort ( 0, 0, a ); → sorted
```



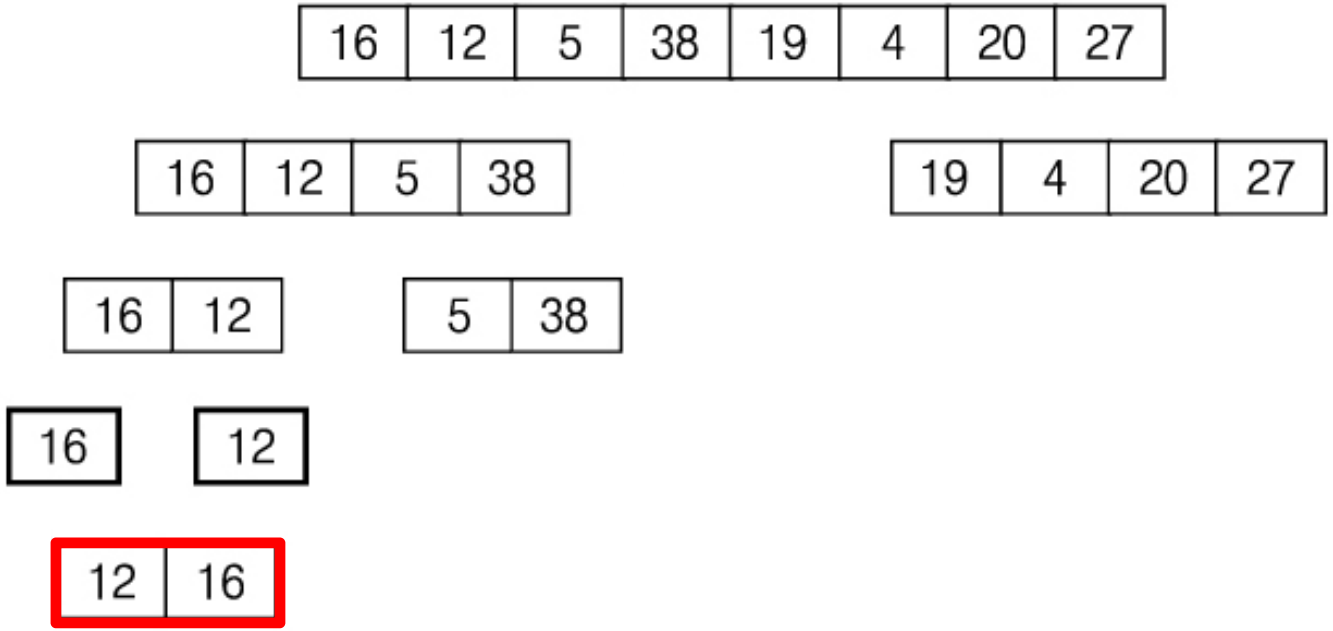
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a );
            m = 0;
            merge_sort ( 0, 0, a ); → sorted
            merge_sort ( 1, 1, a );
```



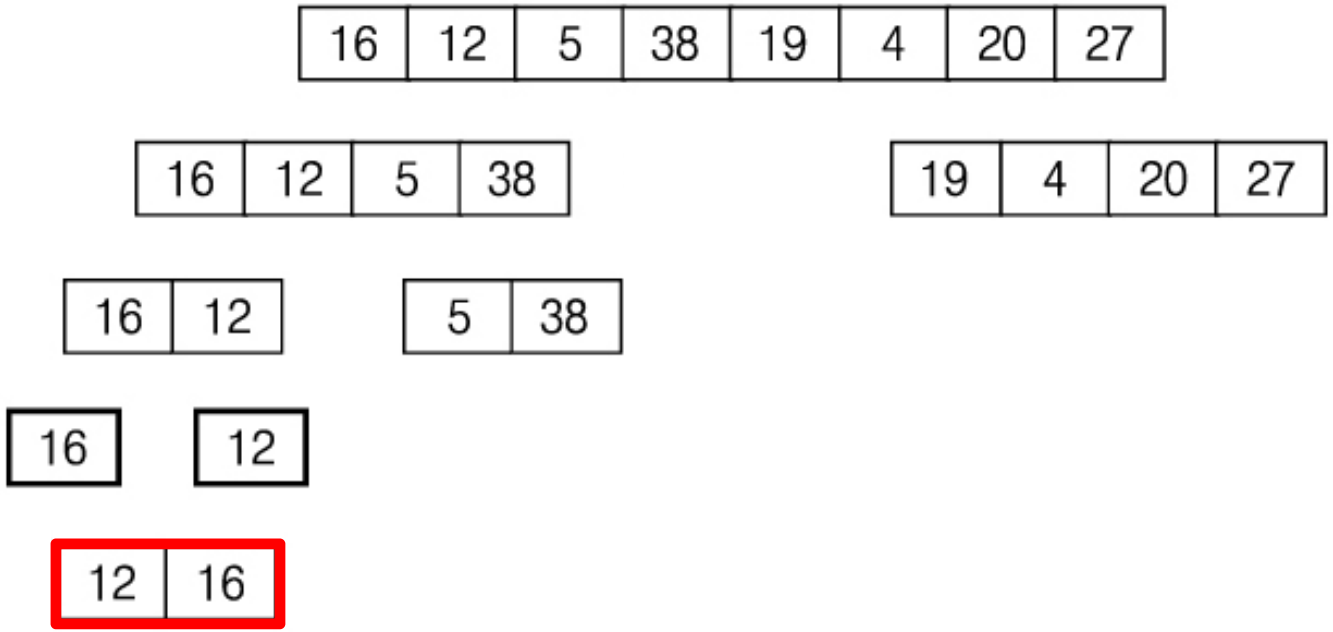
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a );
            m = 0;
            merge_sort ( 0, 0, a ); → sorted
            merge_sort ( 1, 1, a ); → sorted
```



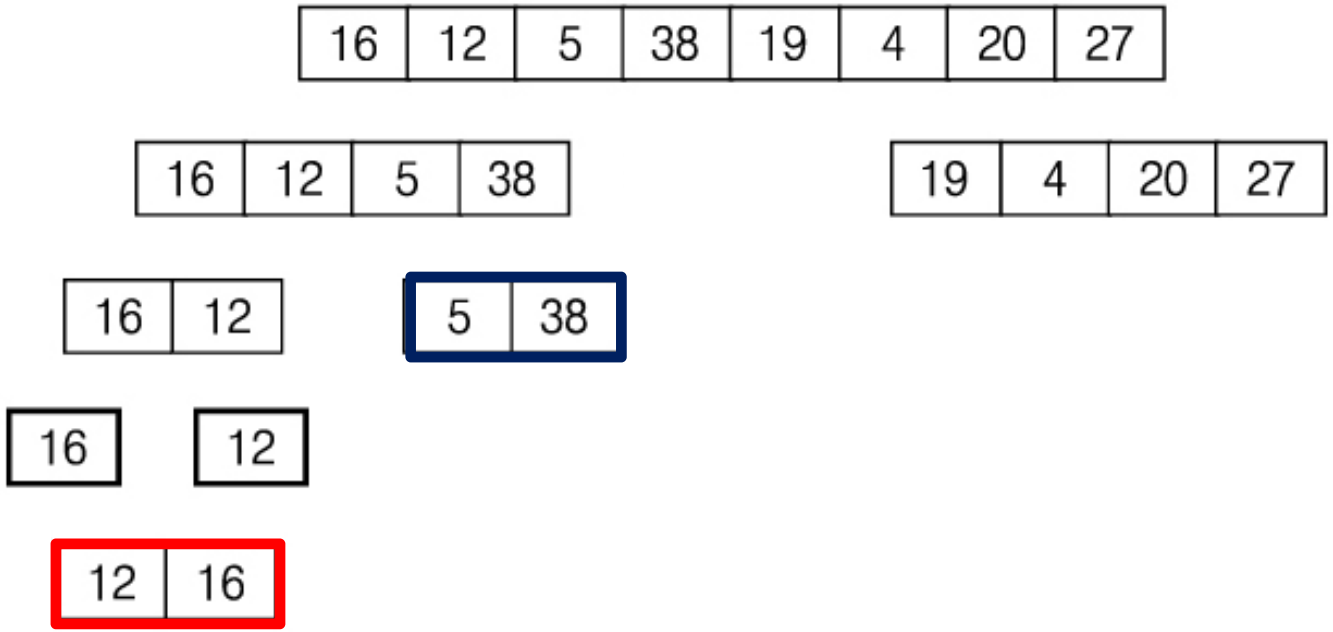
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a );
            m = 0;
            merge_sort ( 0, 0, a ); → sorted
            merge_sort ( 1, 1, a ); → sorted
            merge ( 0, 0, 1, 1, a );
```



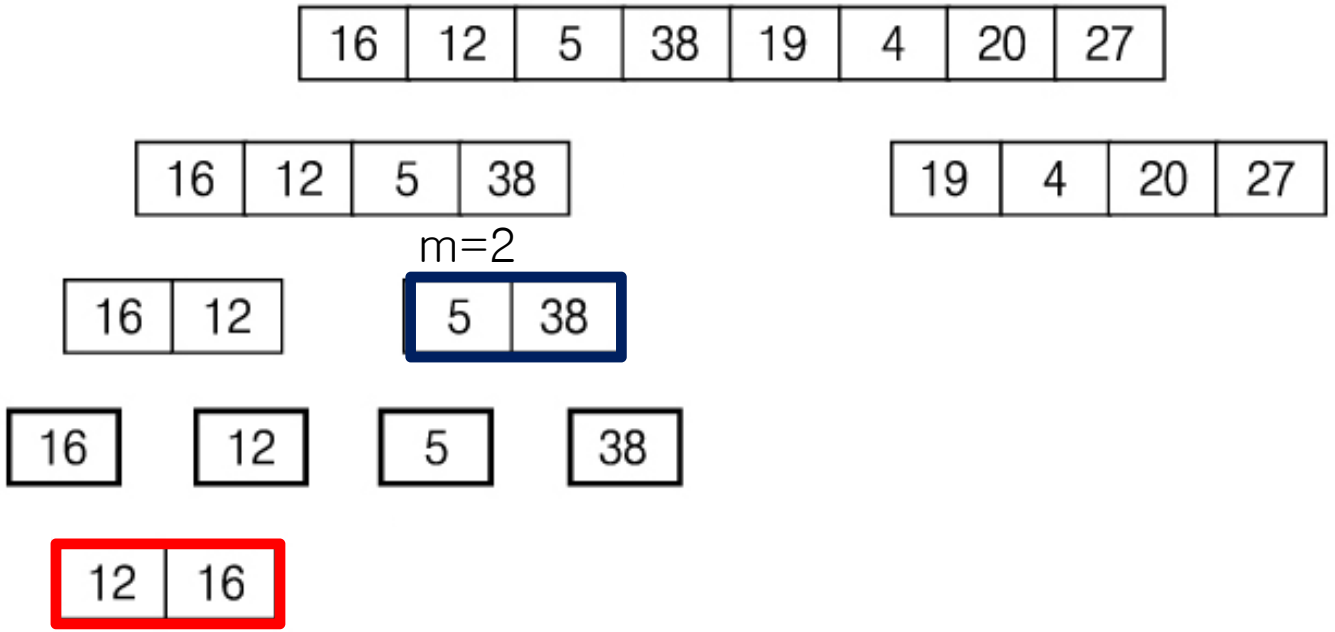
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a ); → sorted
```



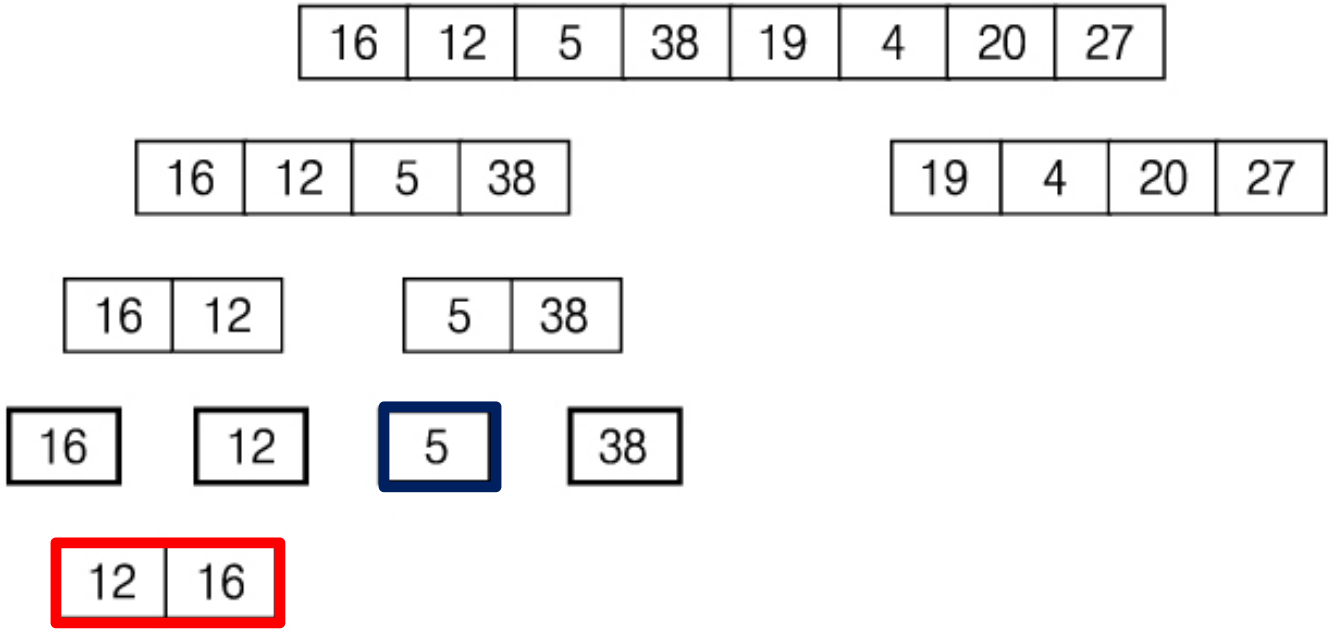
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a ); → sorted
        merge_sort ( 2, 3, a );
```



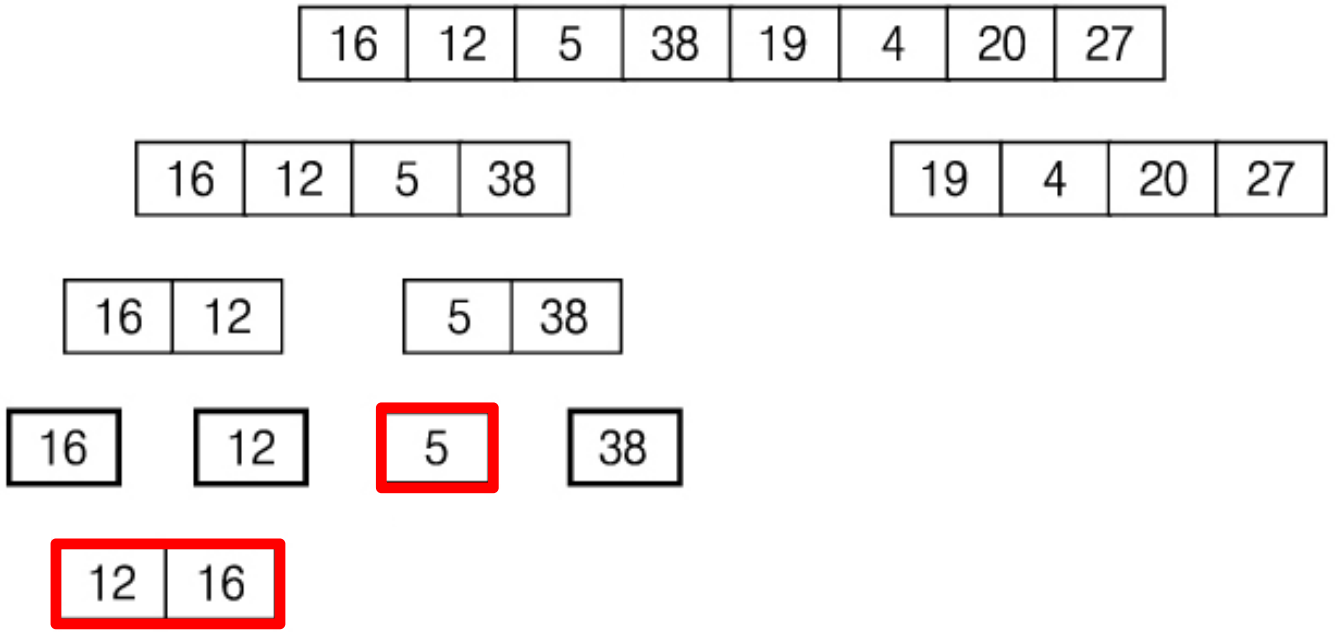
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a ); → sorted
        merge_sort ( 2, 3, a );
            m = 2;
```



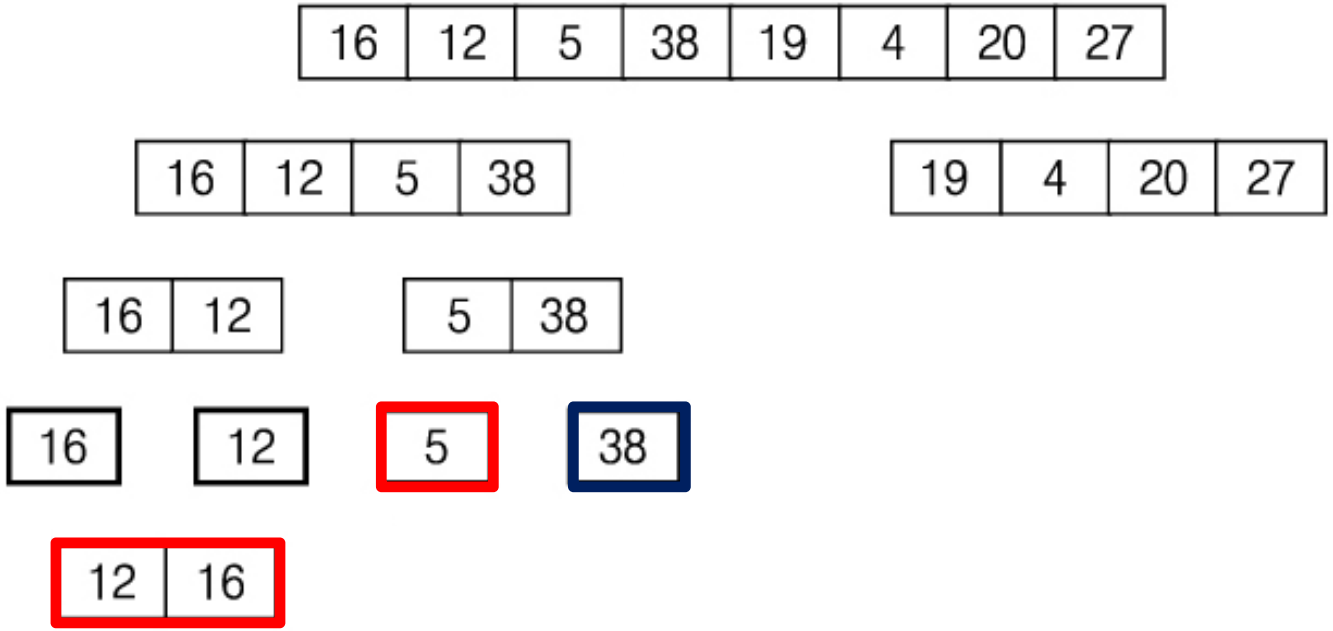
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a ); → sorted
        merge_sort ( 2, 3, a );
            m = 2;
            merge_sort ( 2, 2, a );
```



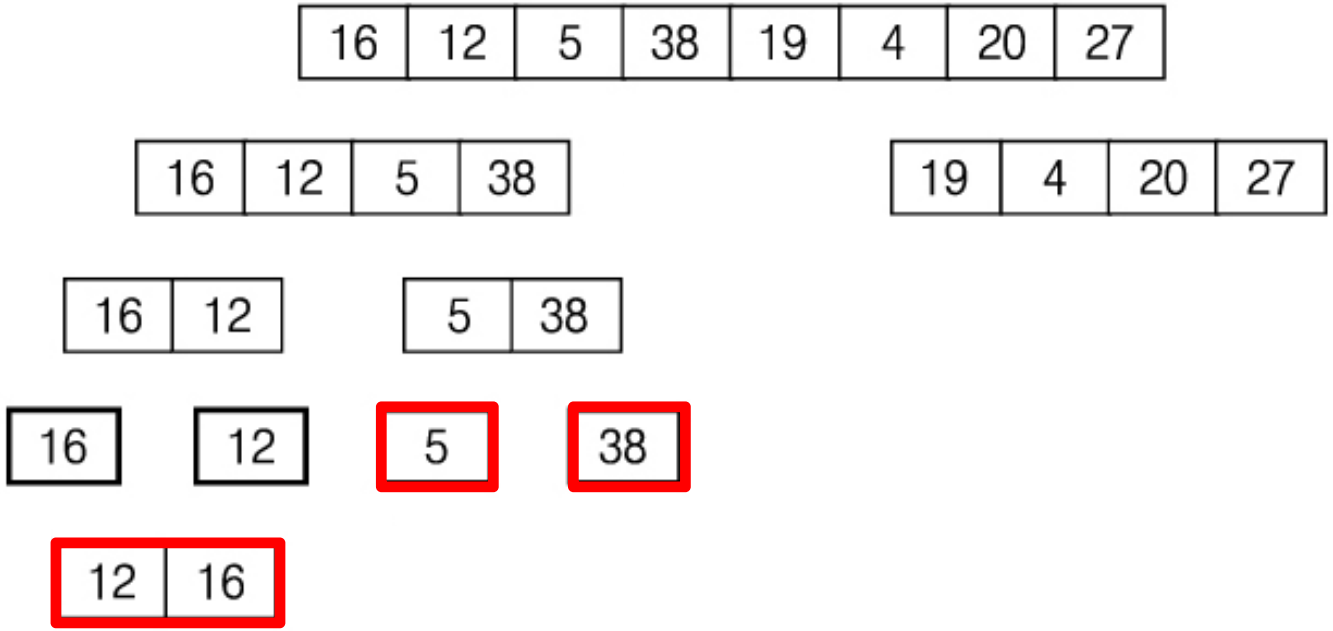
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a ); → sorted
        merge_sort ( 2, 3, a );
            m = 2;
            merge_sort ( 2, 2, a ); → sorted
```



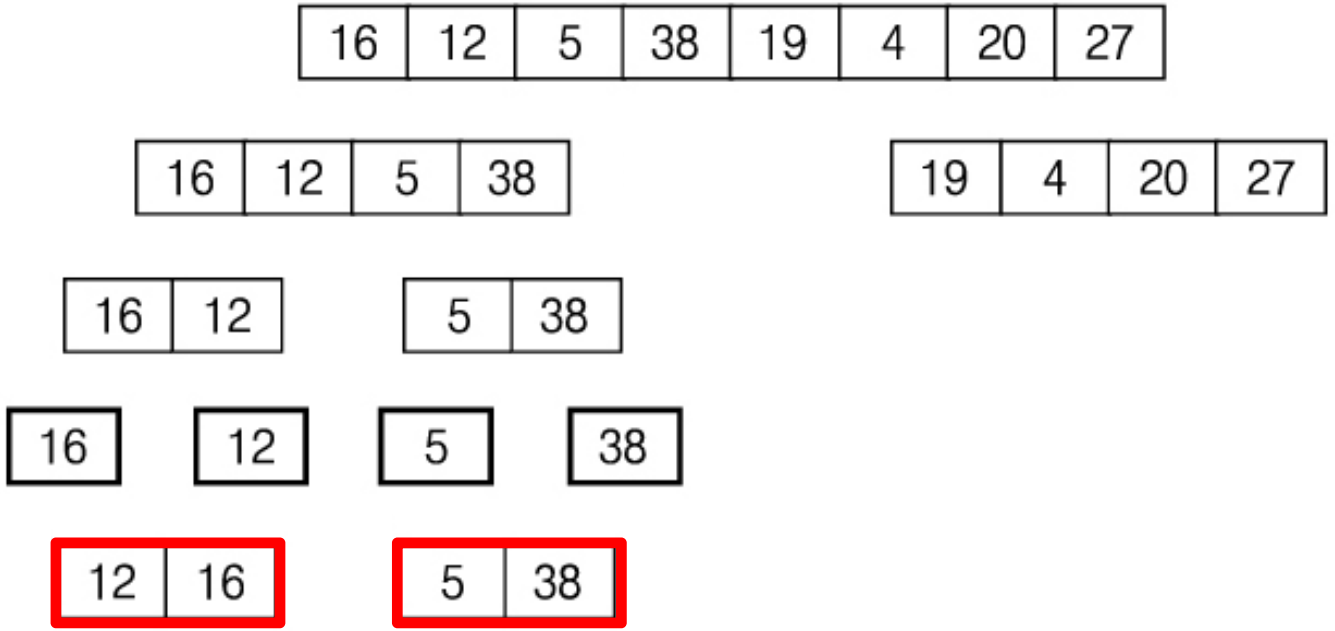
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a ); → sorted
        merge_sort ( 2, 3, a );
            m = 2;
            merge_sort ( 2, 2, a ); → sorted
            merge_sort ( 3, 3, a );
```



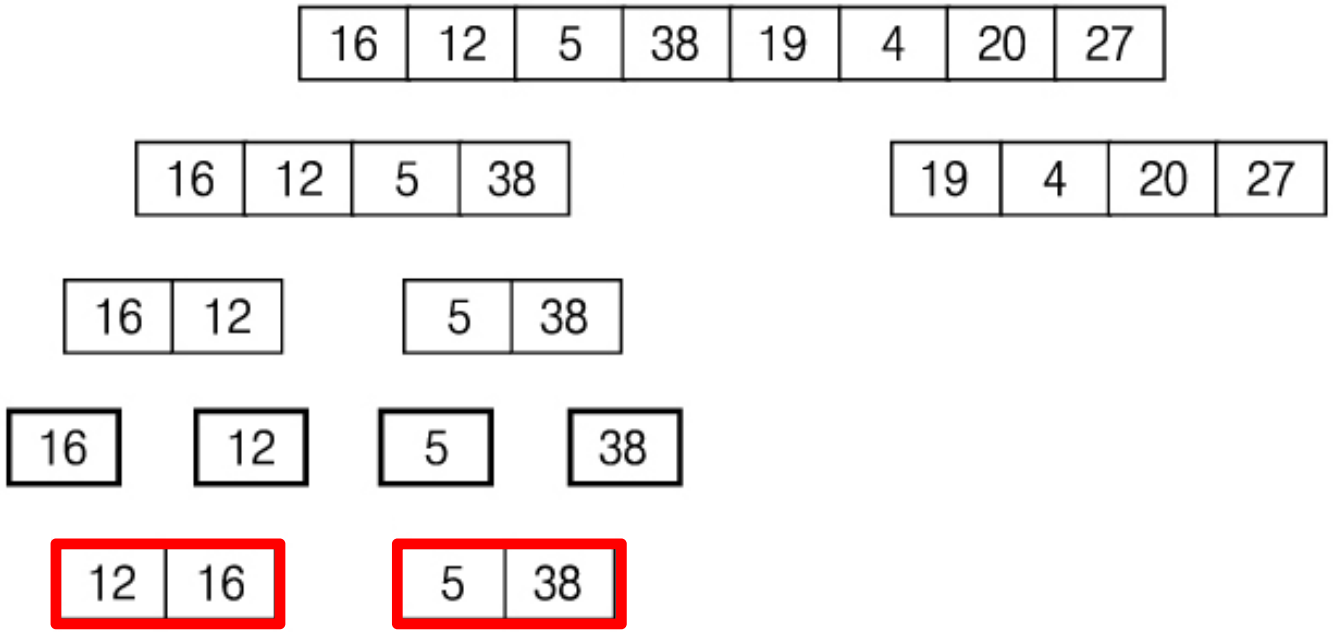
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a ); → sorted
        merge_sort ( 2, 3, a );
            m = 2;
            merge_sort ( 2, 2, a ); → sorted
            merge_sort ( 3, 3, a ); → sorted
```



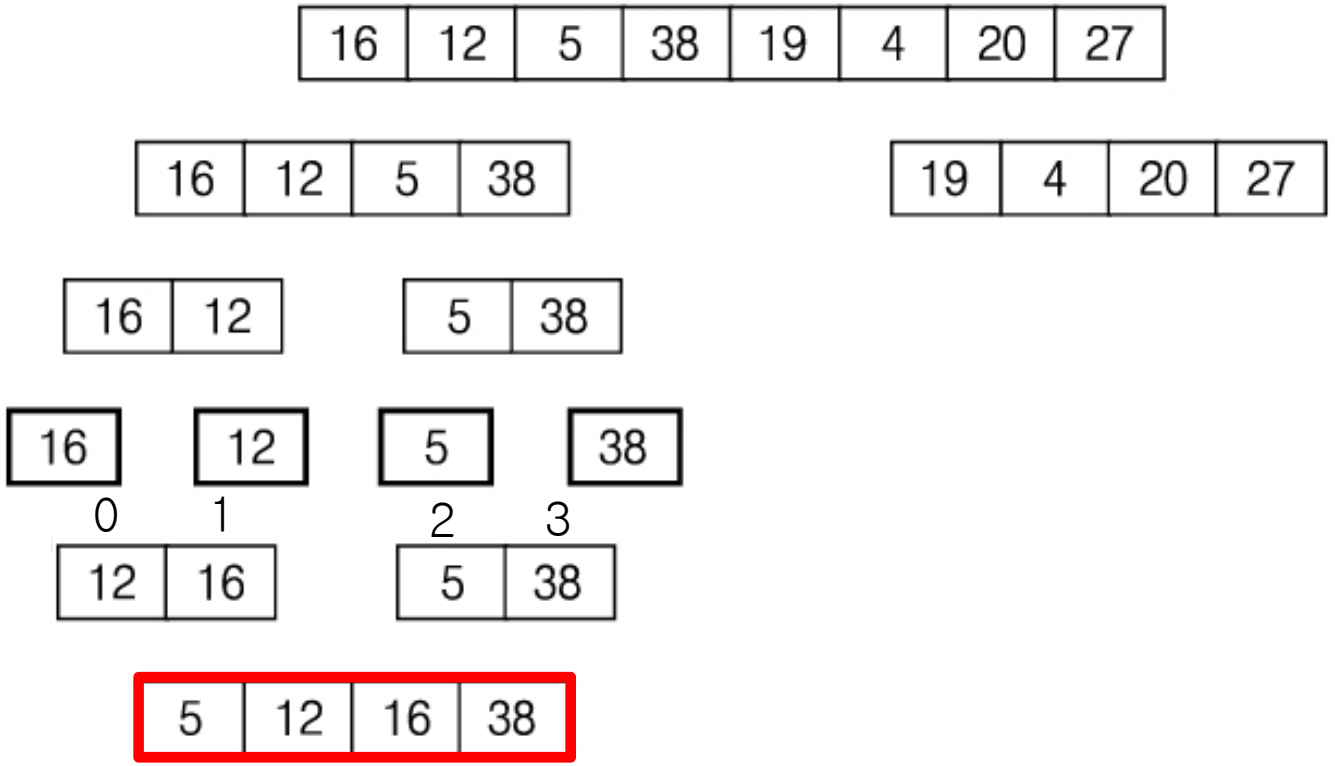
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a ); → sorted
        merge_sort ( 2, 3, a );
            m = 2;
            merge_sort ( 2, 2, a ); → sorted
            merge_sort ( 3, 3, a ); → sorted
            merge ( 2, 2, 3, 3, a );
```



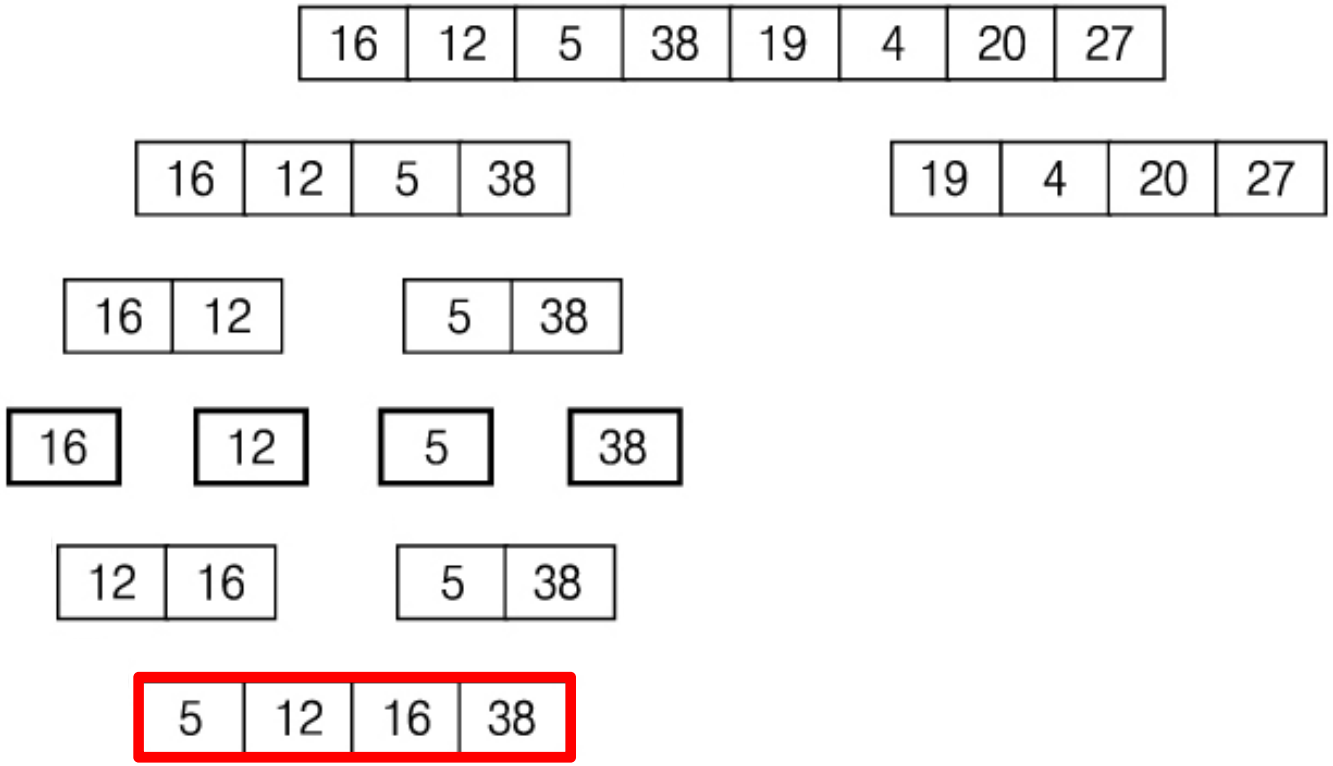
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a ); → sorted
        merge_sort ( 2, 3, a ); → sorted
```



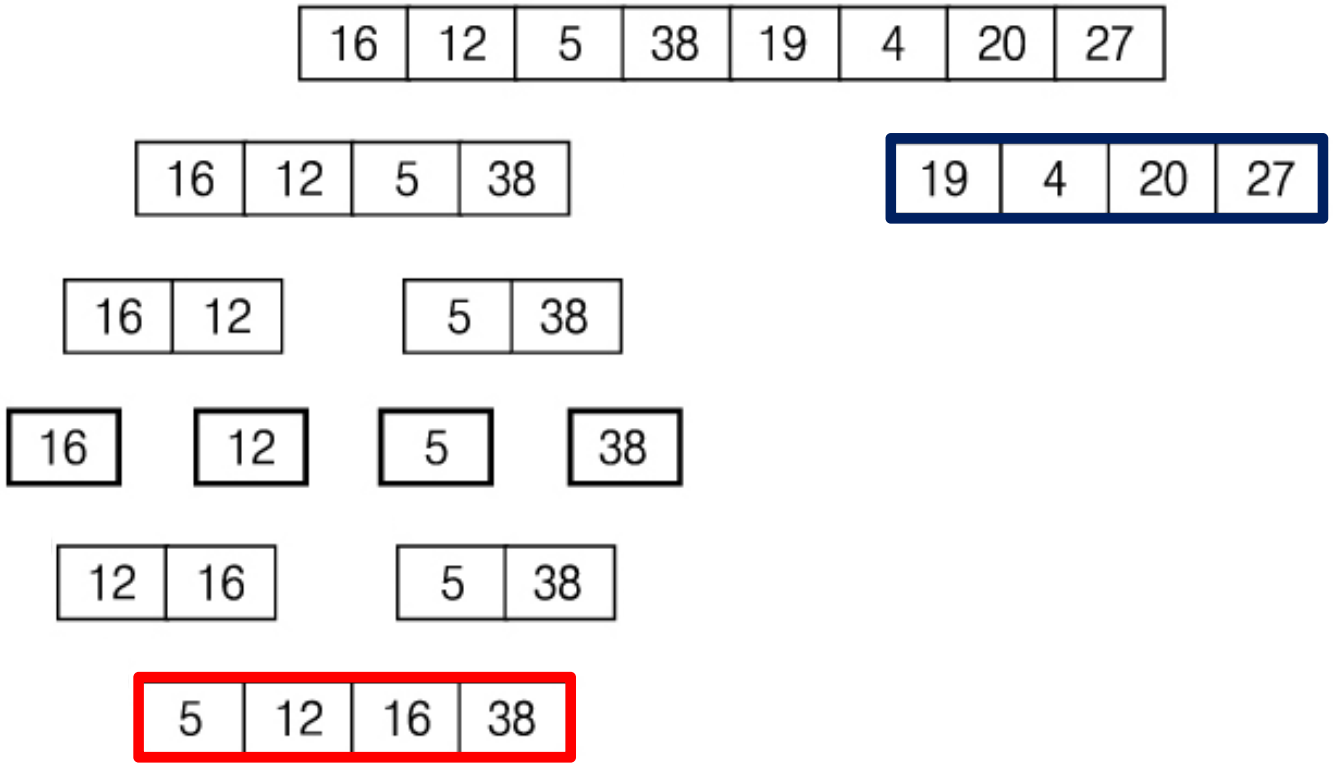
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a );
        m = 1;
        merge_sort ( 0, 1, a ); → sorted
        merge_sort ( 2, 3, a ); → sorted
        merge ( 0, 1, 2, 3, a );
```



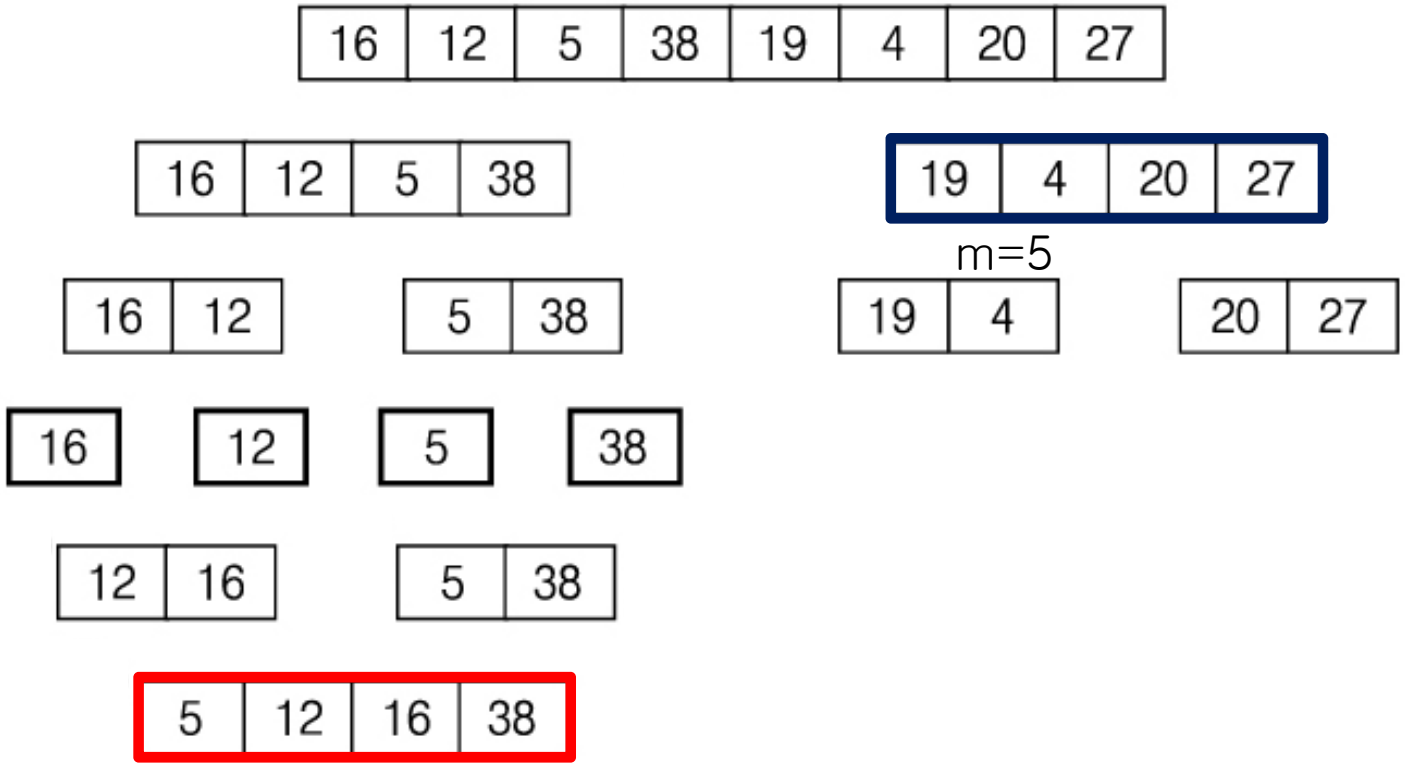
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
```



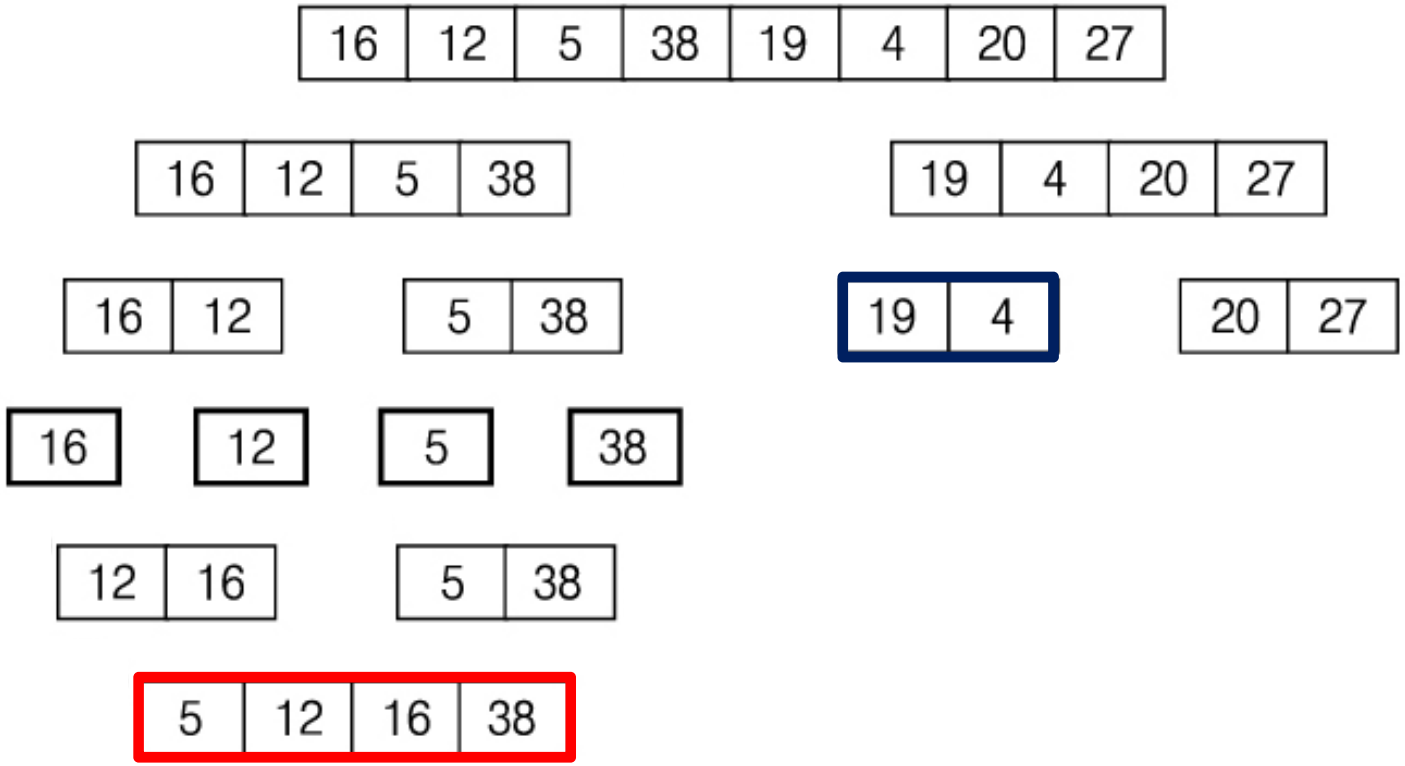
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
```



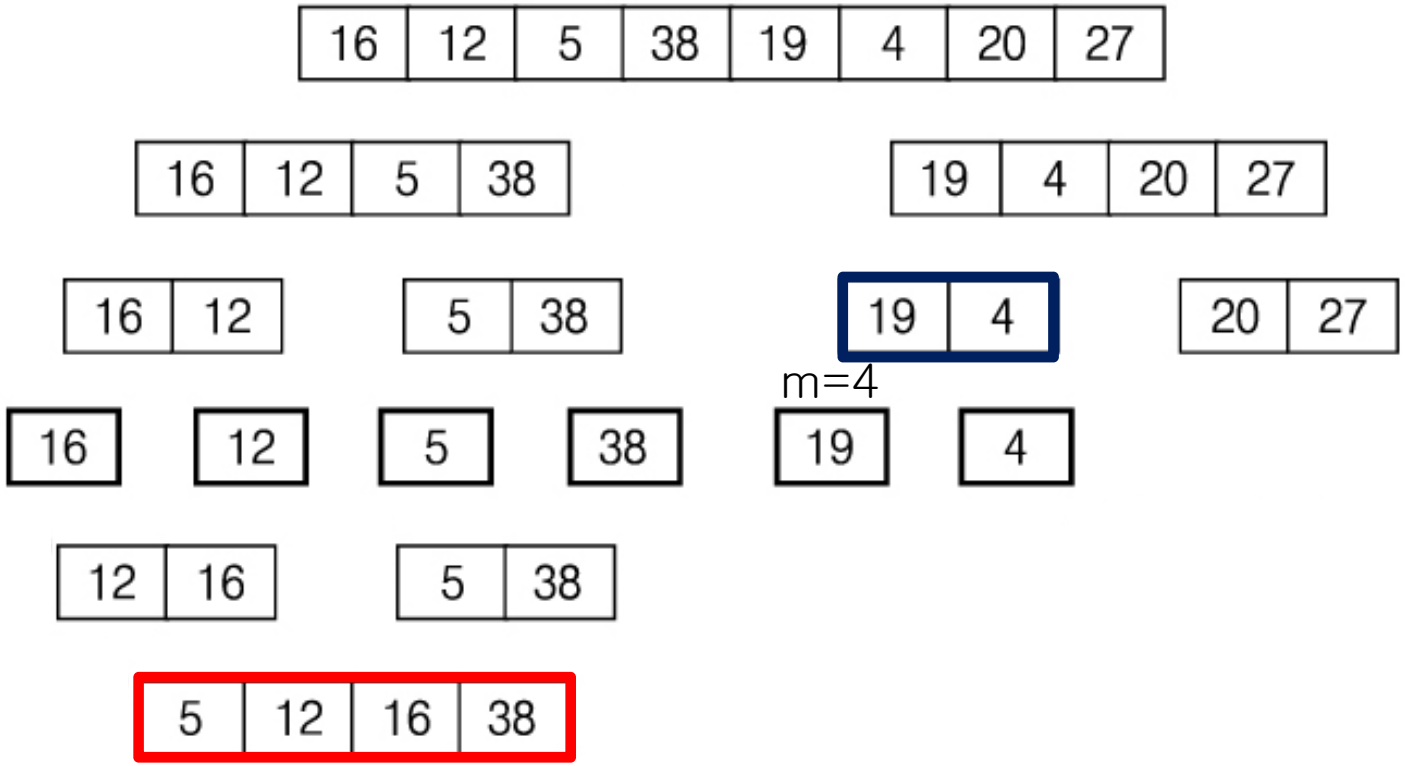
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
```



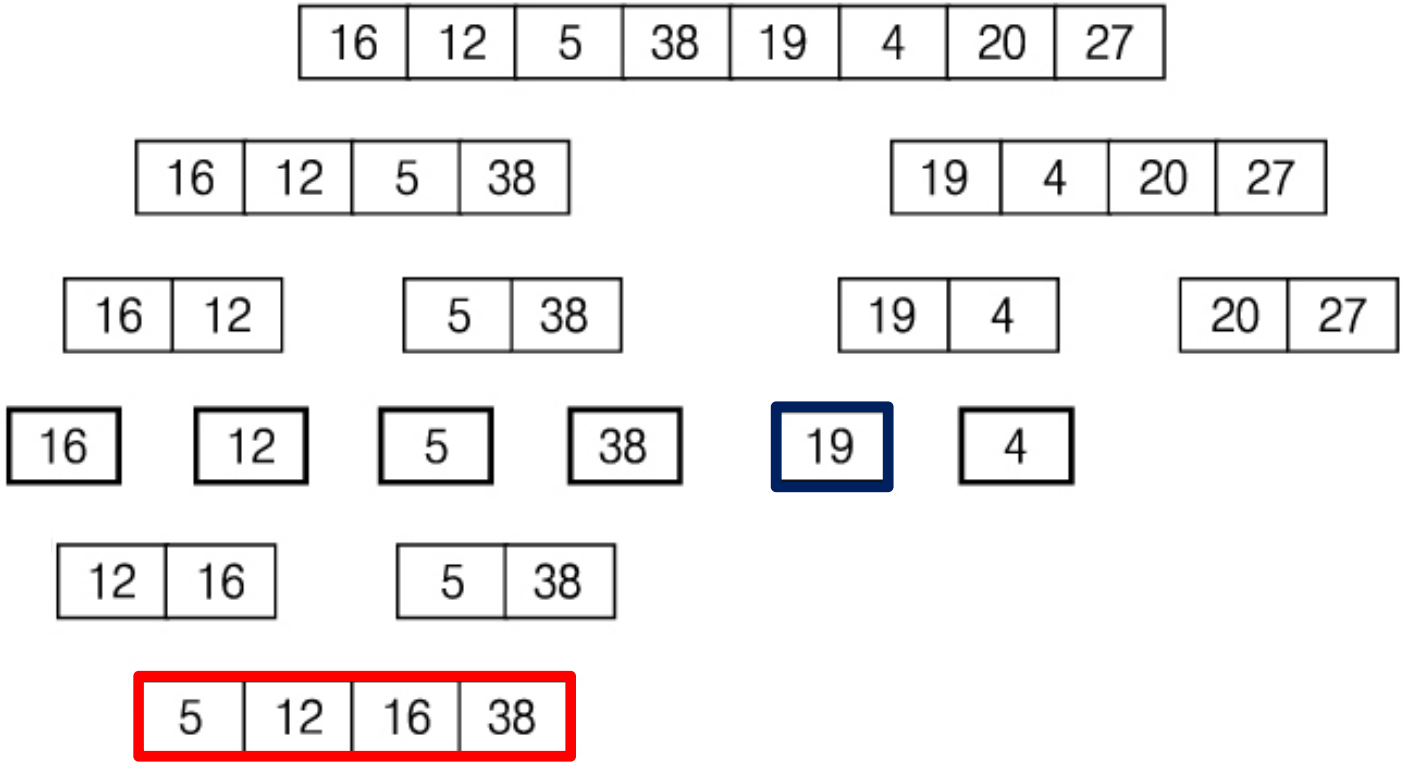
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a );
```



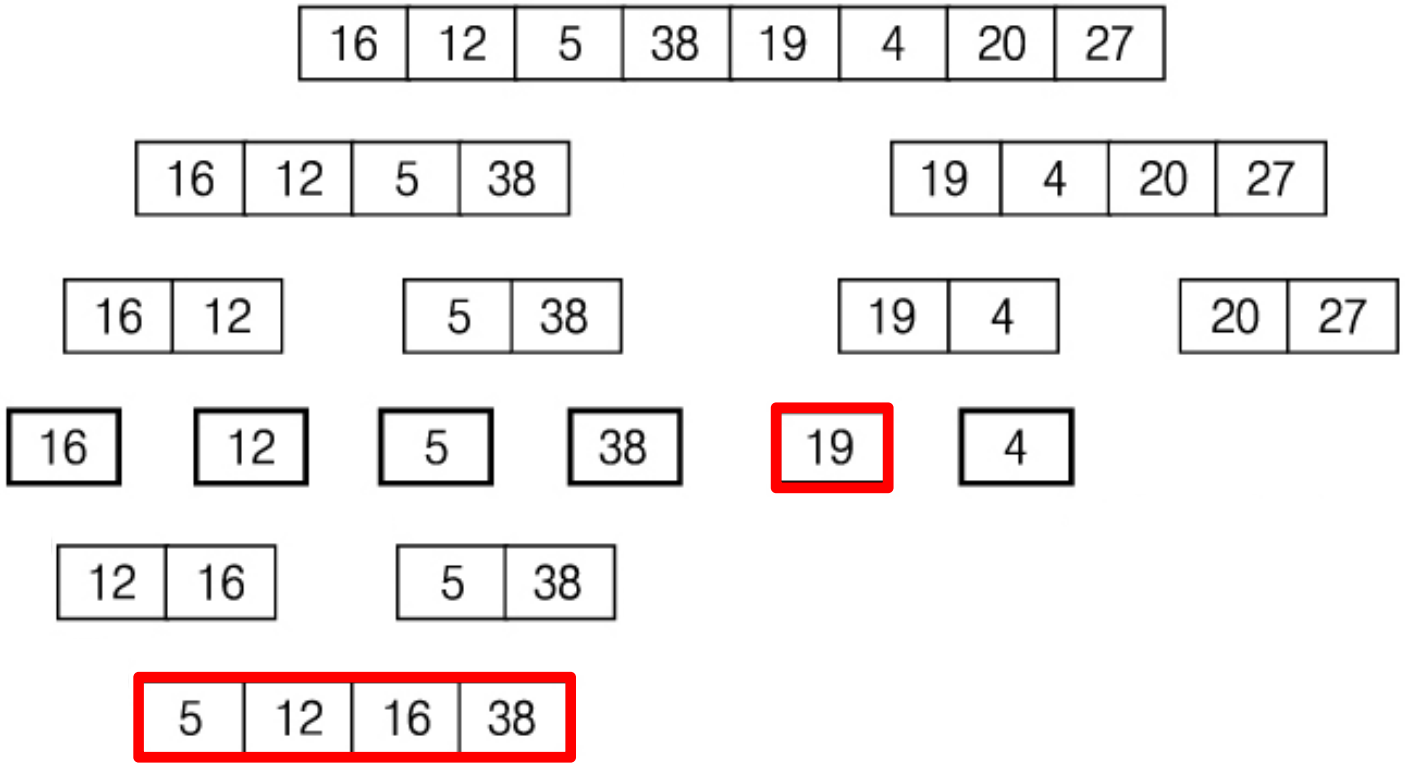
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a );
            m = 4;
```



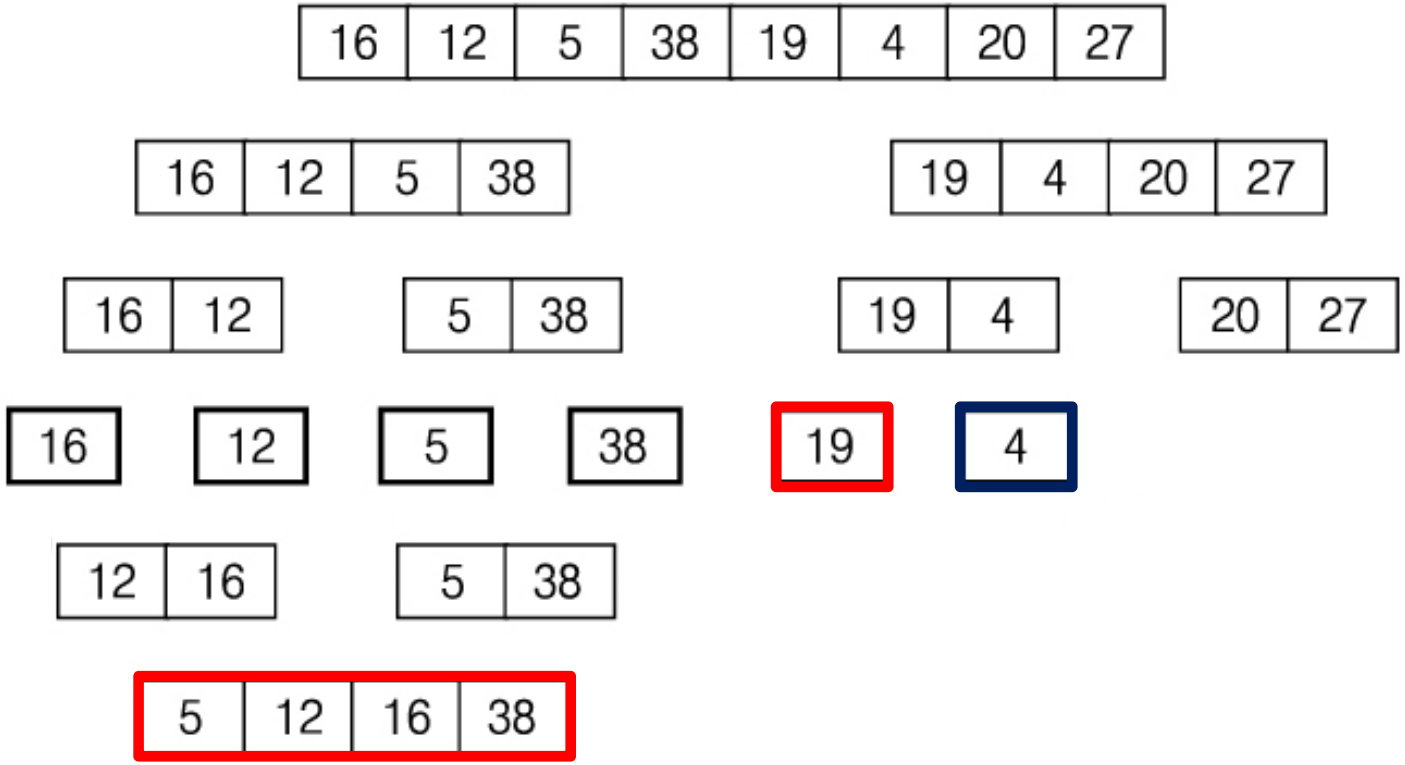
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a );
            m = 4;
            merge_sort ( 4, 4, a );
```



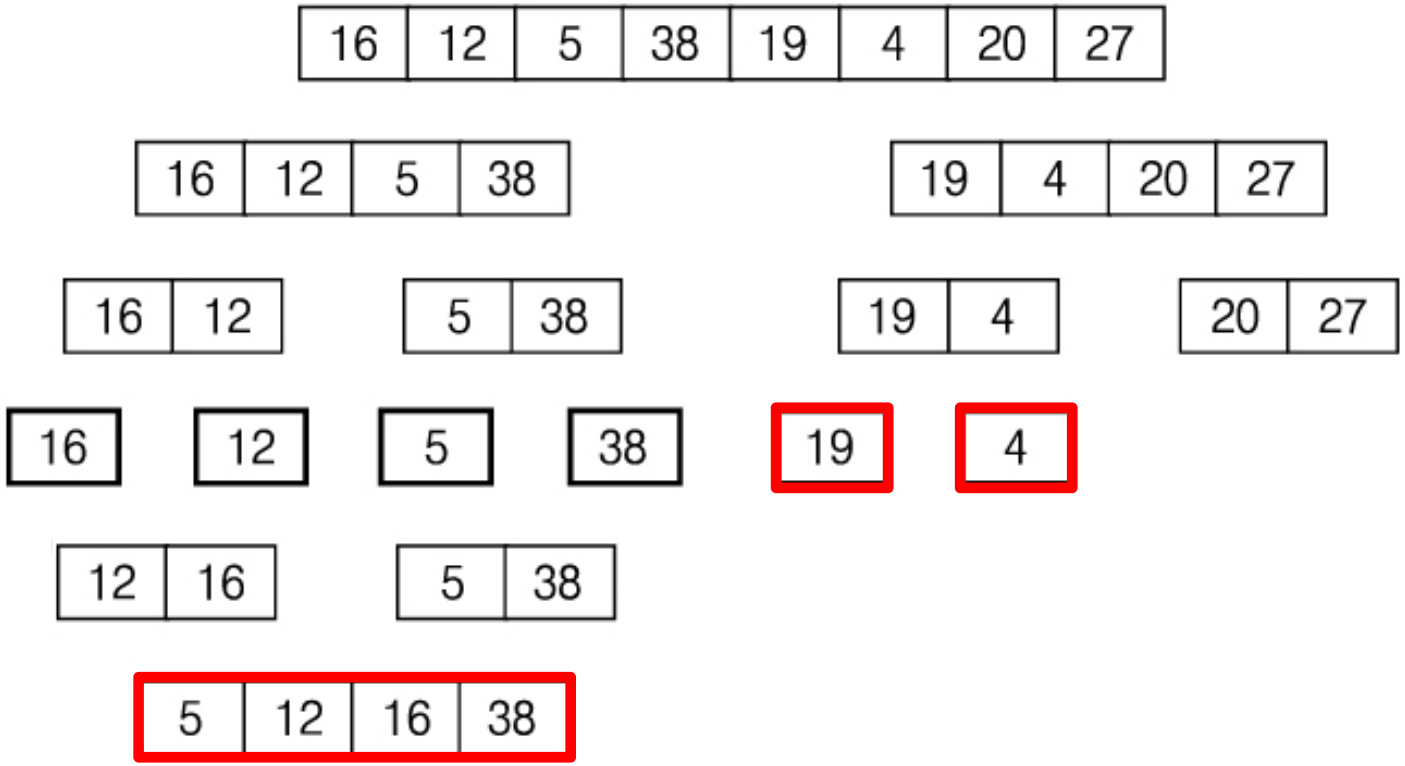
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a );
            m = 4;
            merge_sort ( 4, 4, a ); → sorted
```



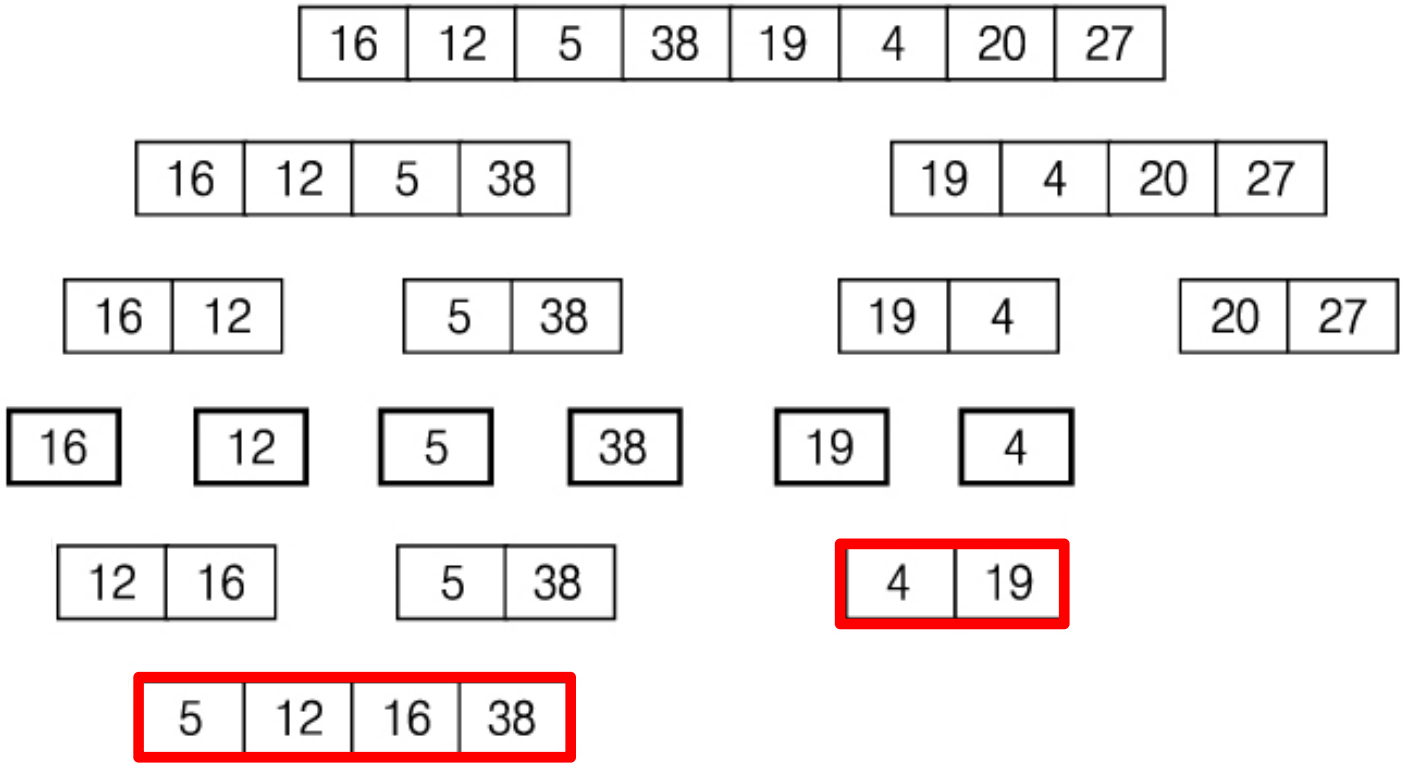
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a );
            m = 4;
            merge_sort ( 4, 4, a ); → sorted
            merge_sort ( 5, 5, a );
```



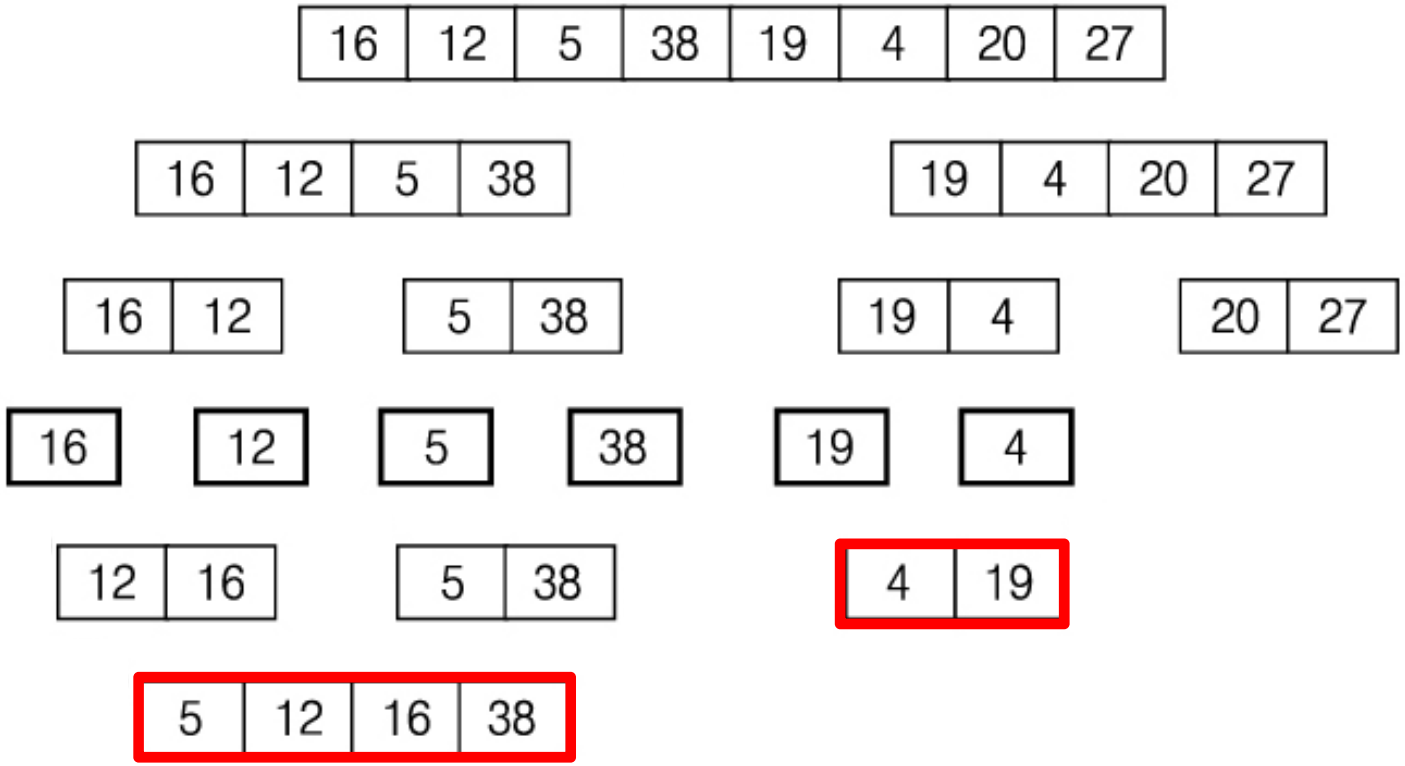
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a );
            m = 4;
            merge_sort ( 4, 4, a ); → sorted
            merge_sort ( 5, 5, a ); → sorted
```



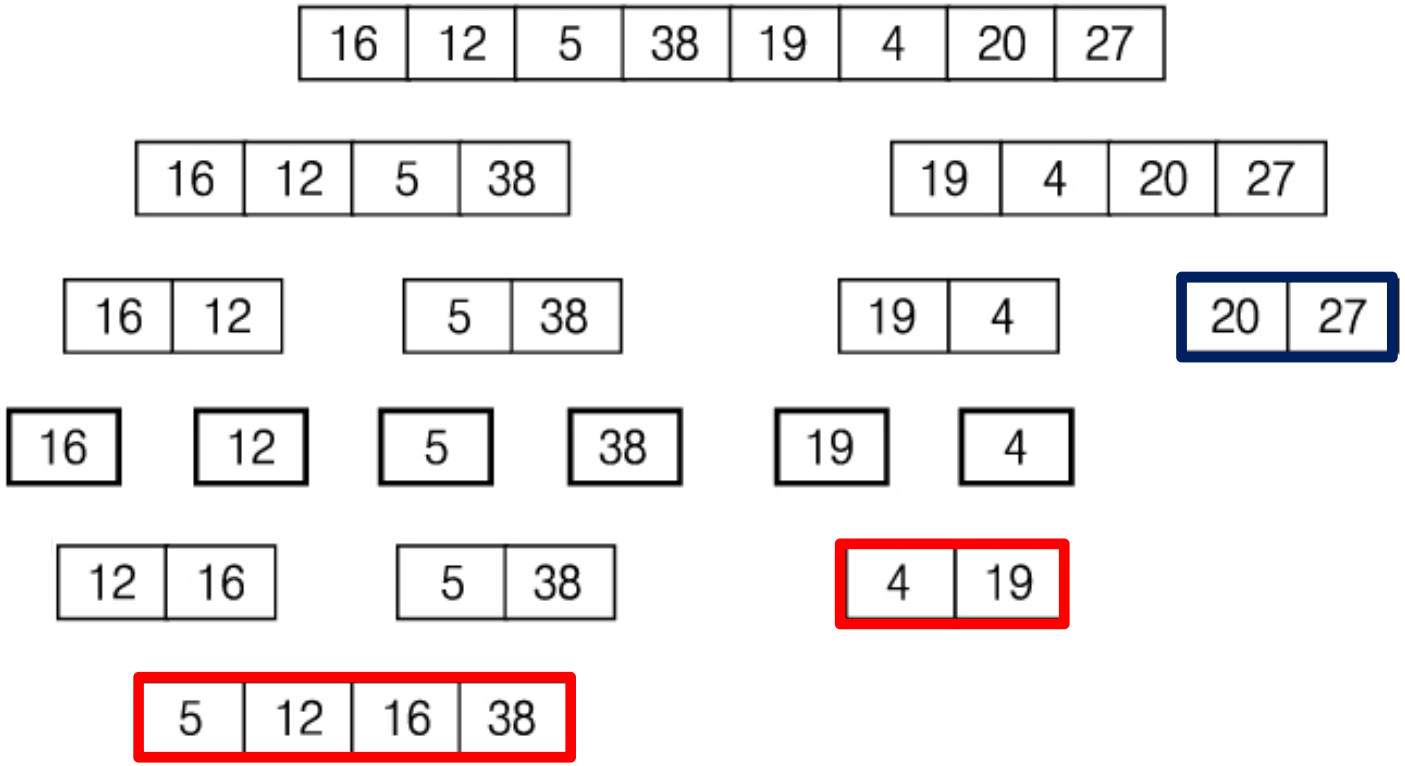
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a );
            m = 4;
            merge_sort ( 4, 4, a ); → sorted
            merge_sort ( 5, 5, a ); → sorted
            merge ( 4, 4, 5, 5, a );
```



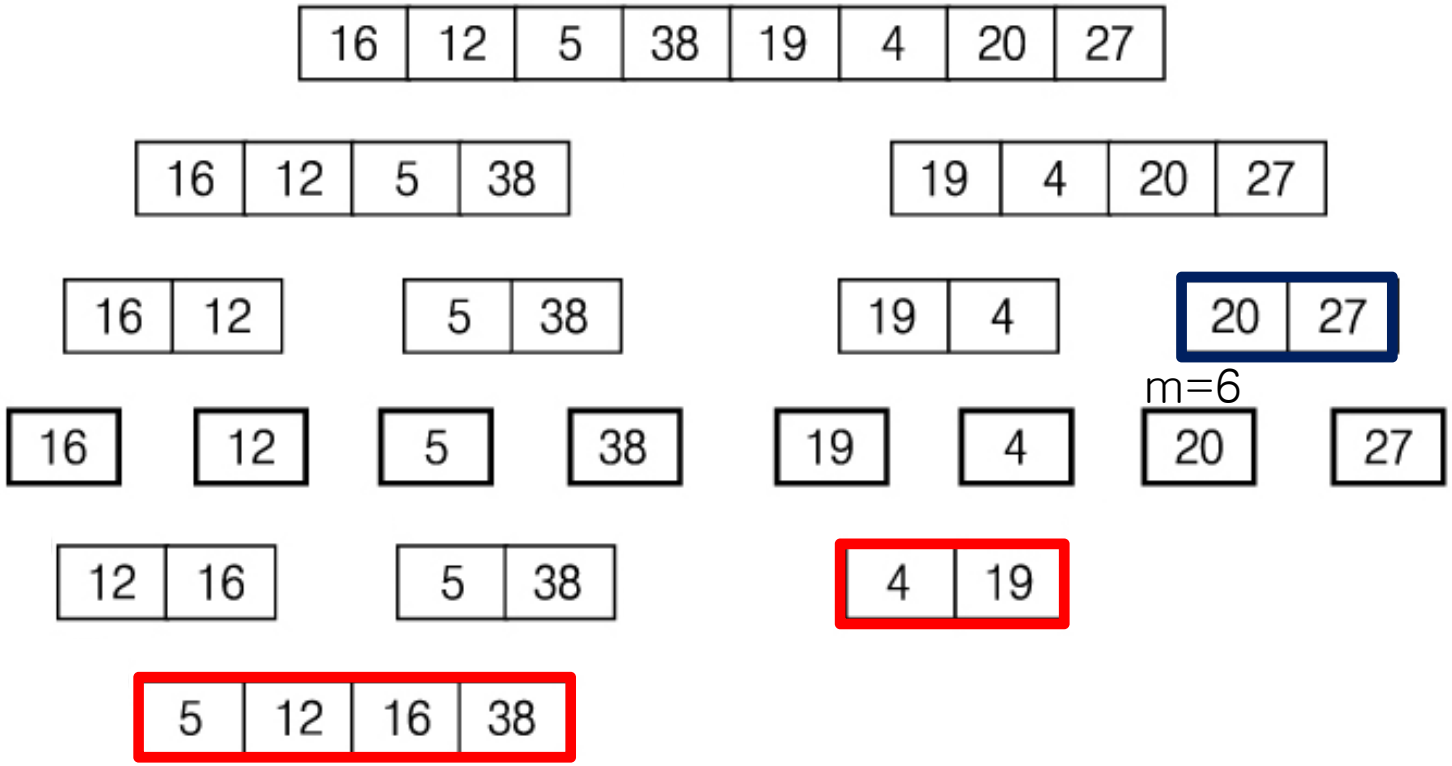
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a ); → sorted
```



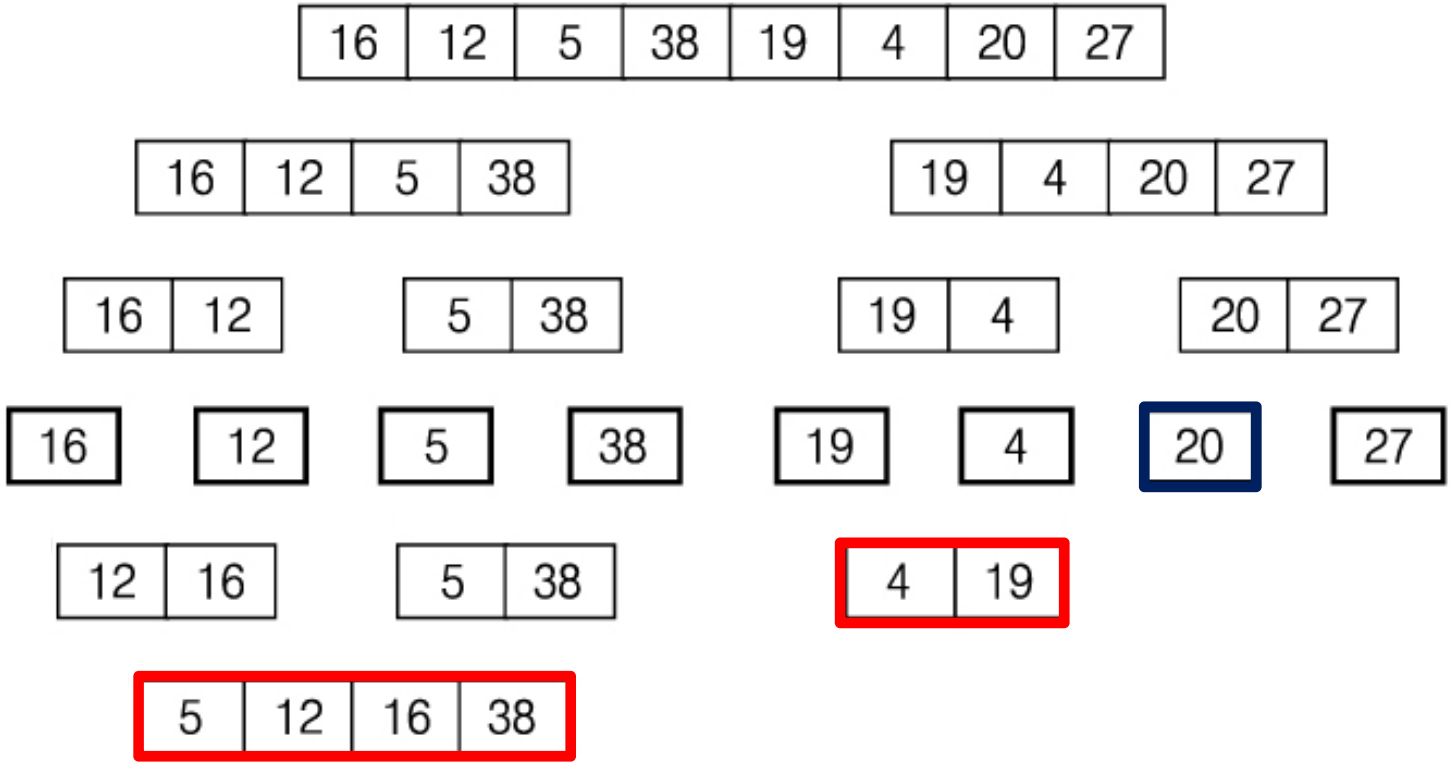
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a ); → sorted
        merge_sort ( 6, 7, a );
```



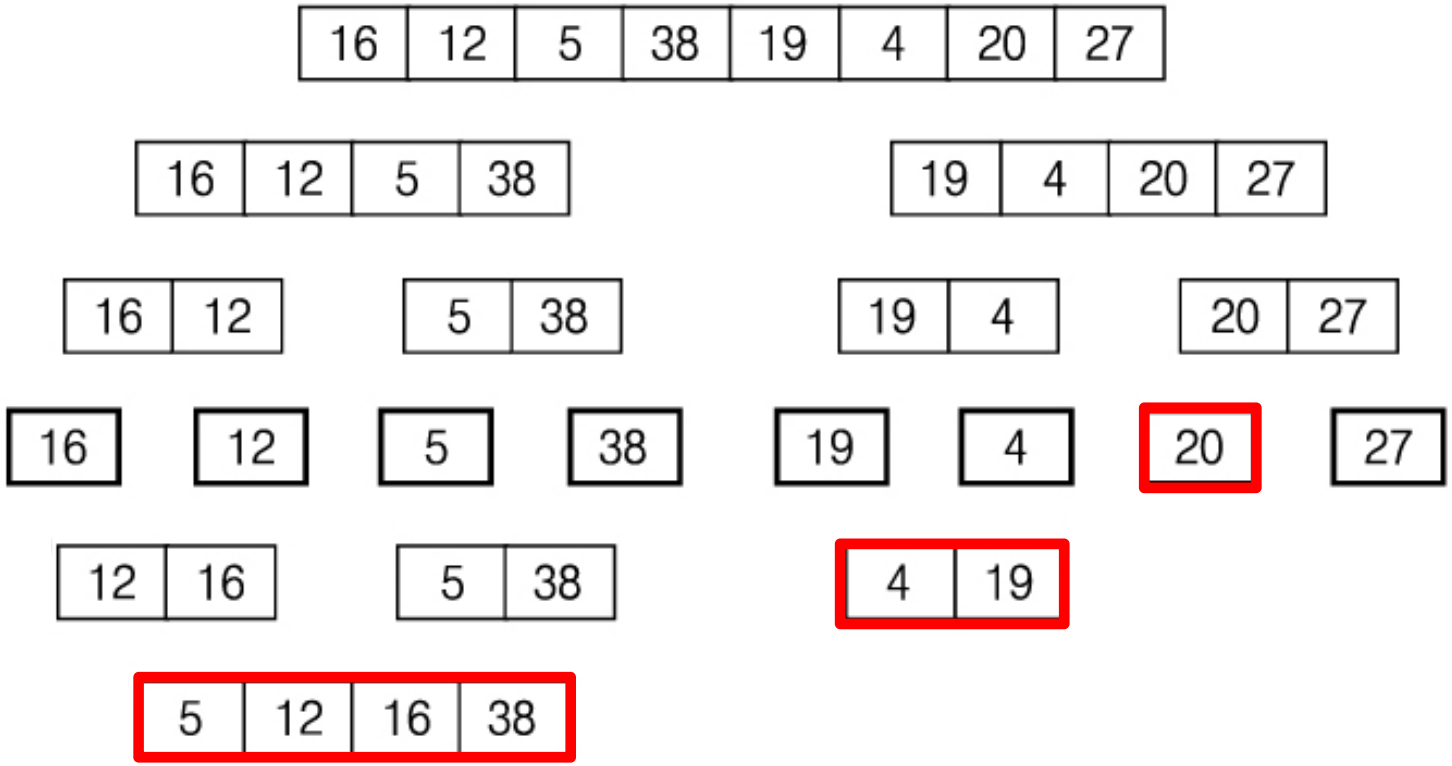
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a ); → sorted
        merge_sort ( 6, 7, a );
            m = 6;
```



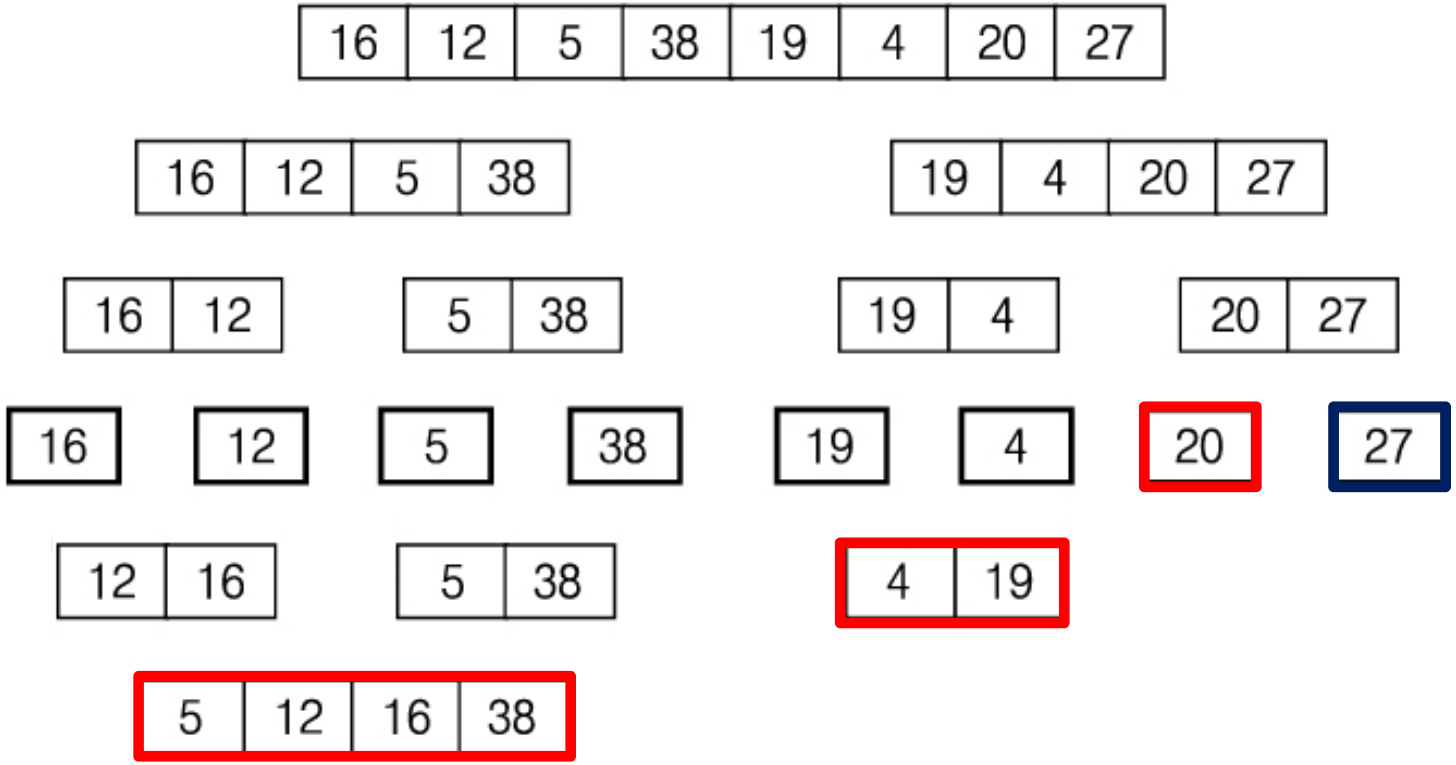
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a ); → sorted
        merge_sort ( 6, 7, a );
            m = 6;
            merge_sort ( 6, 6, a );
```



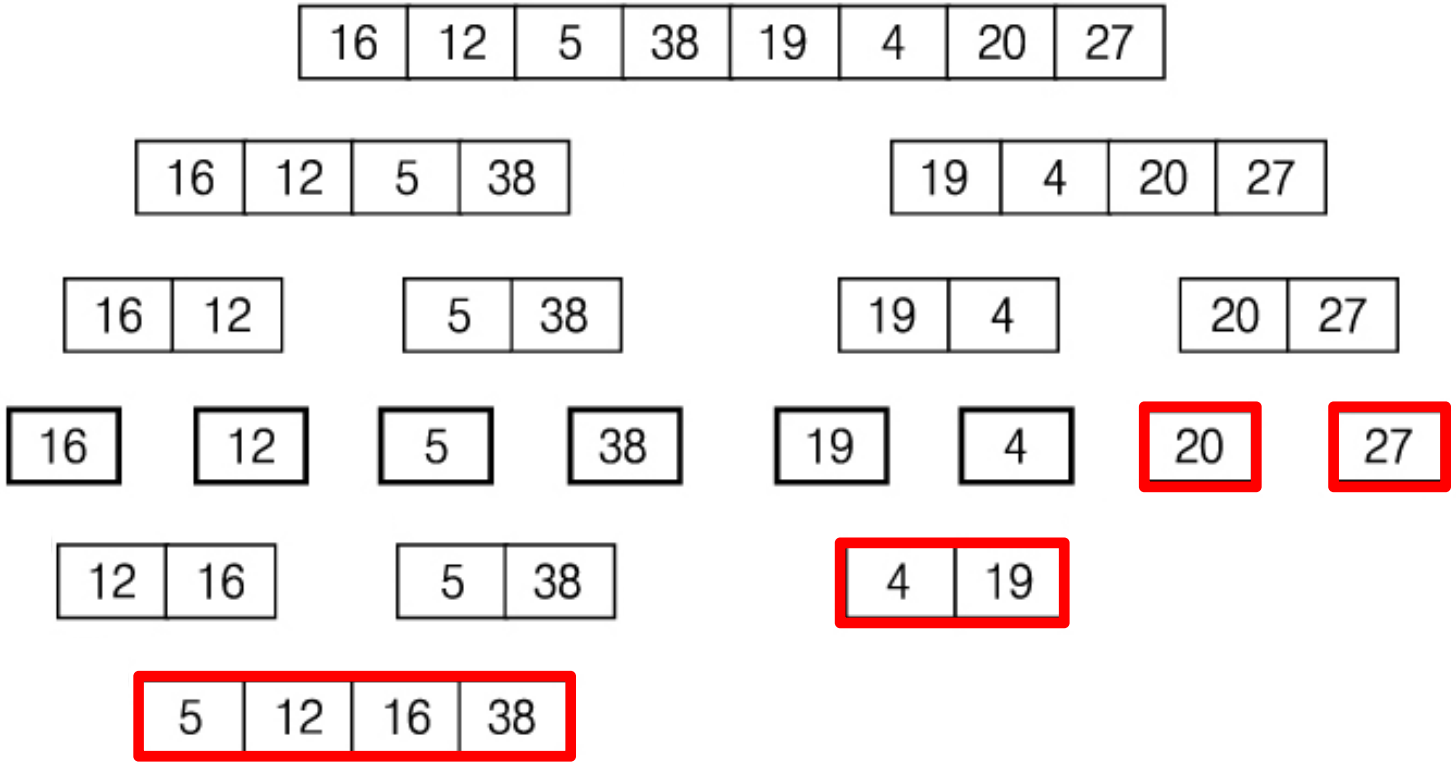
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a ); → sorted
        merge_sort ( 6, 7, a );
            m = 6;
            merge_sort ( 6, 6, a ); → sorted
```



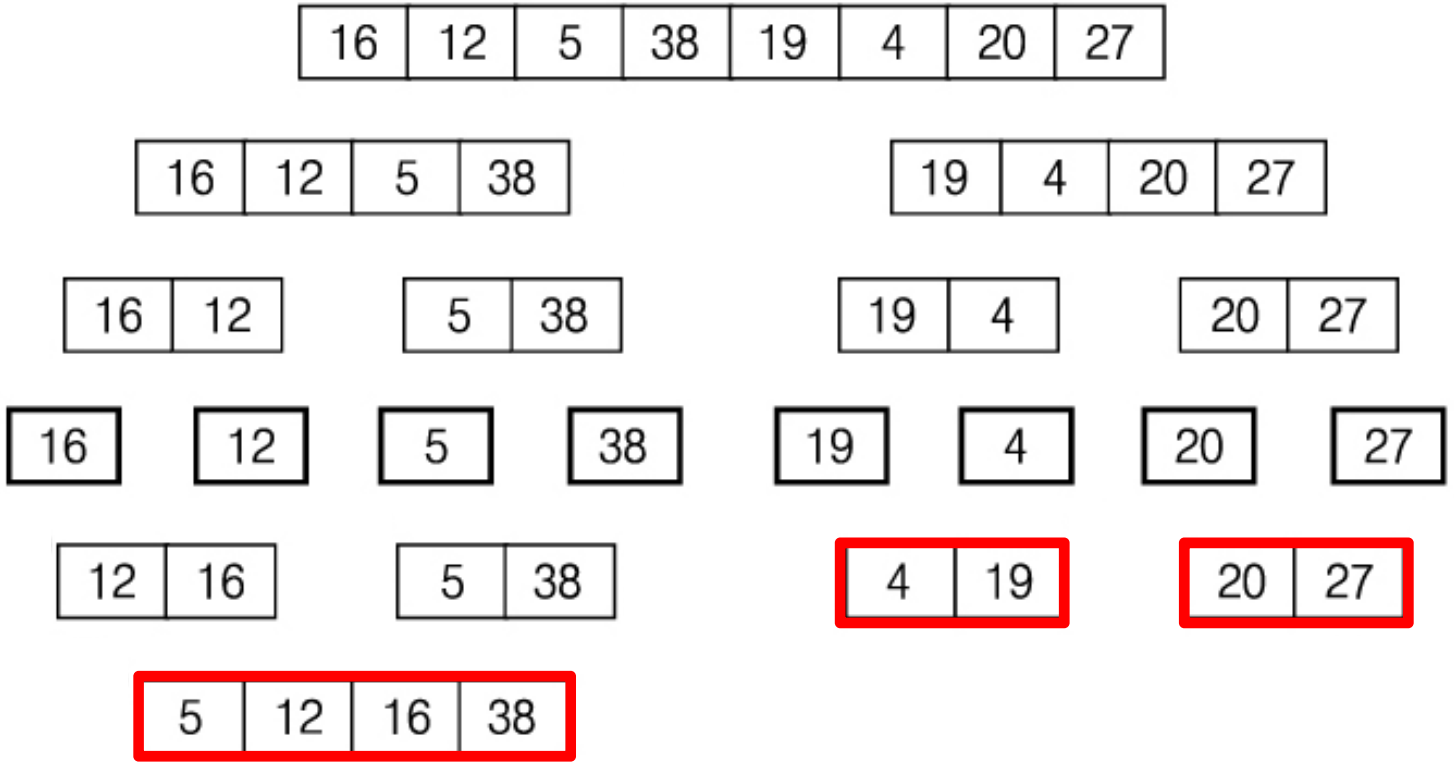
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a ); → sorted
        merge_sort ( 6, 7, a );
            m = 6;
            merge_sort ( 6, 6, a ); → sorted
            merge_sort ( 7, 7, a );
```



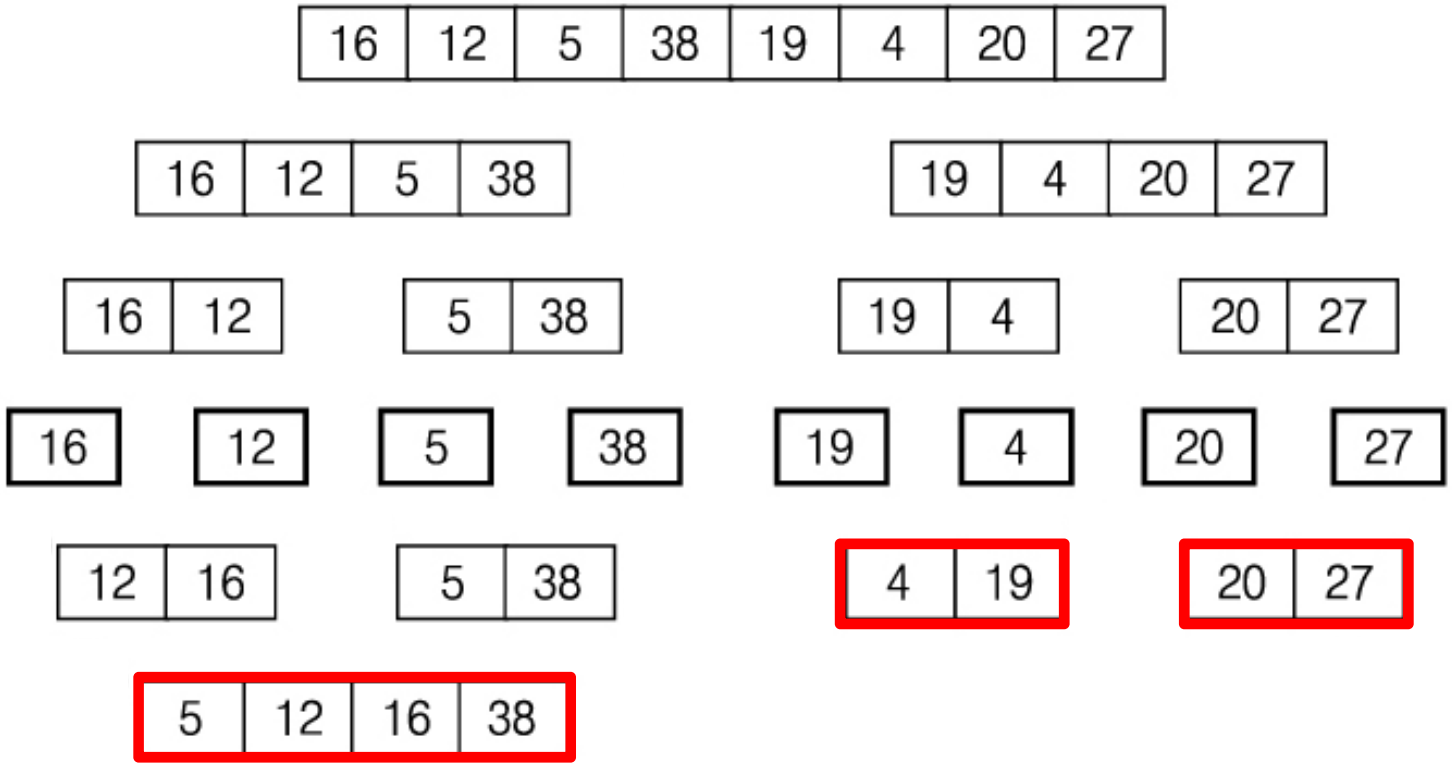
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a ); → sorted
        merge_sort ( 6, 7, a );
            m = 6;
            merge_sort ( 6, 6, a ); → sorted
            merge_sort ( 7, 7, a ); → sorted
```



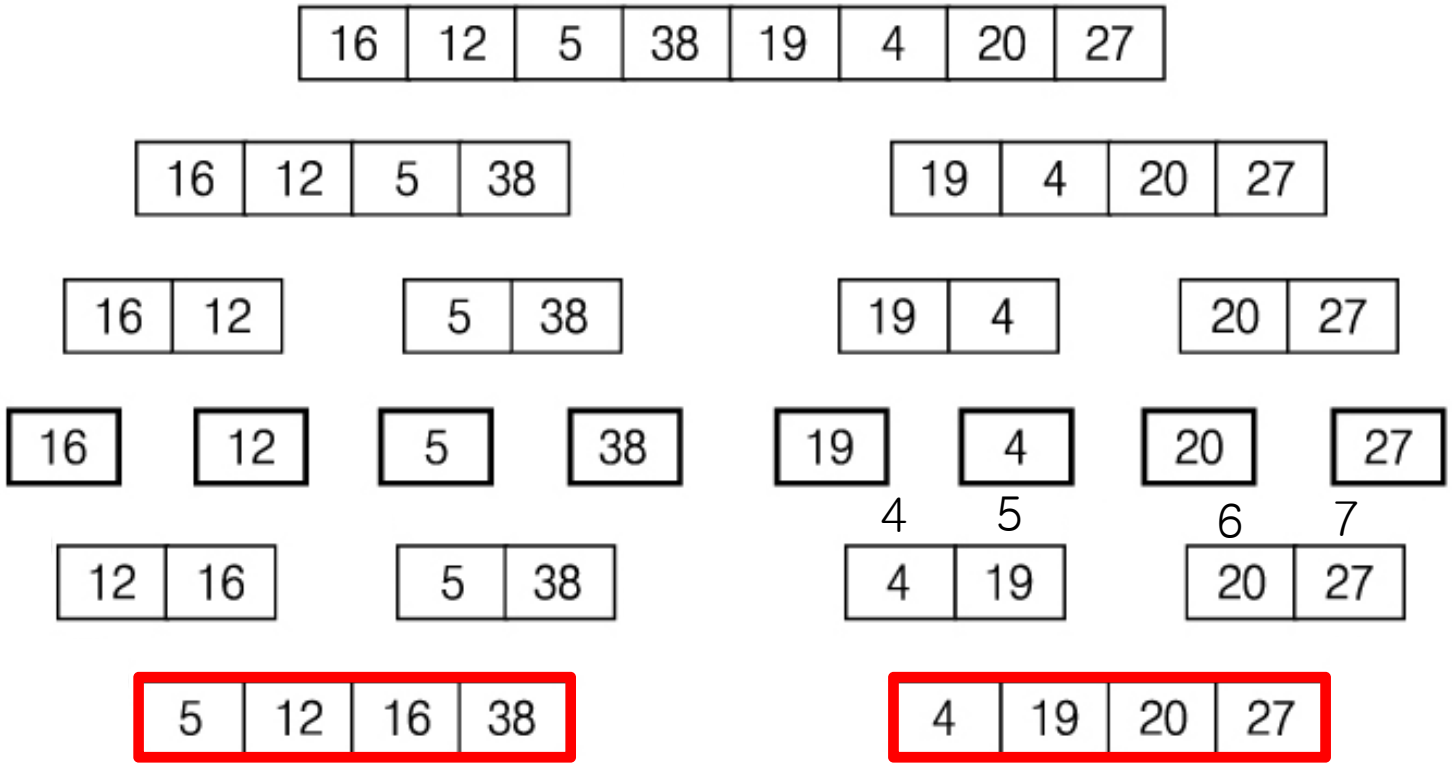
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a ); → sorted
        merge_sort ( 6, 7, a );
            m = 6;
            merge_sort ( 6, 6, a ); → sorted
            merge_sort ( 7, 7, a ); → sorted
            merge ( 6, 6, 7, 7, a );
```



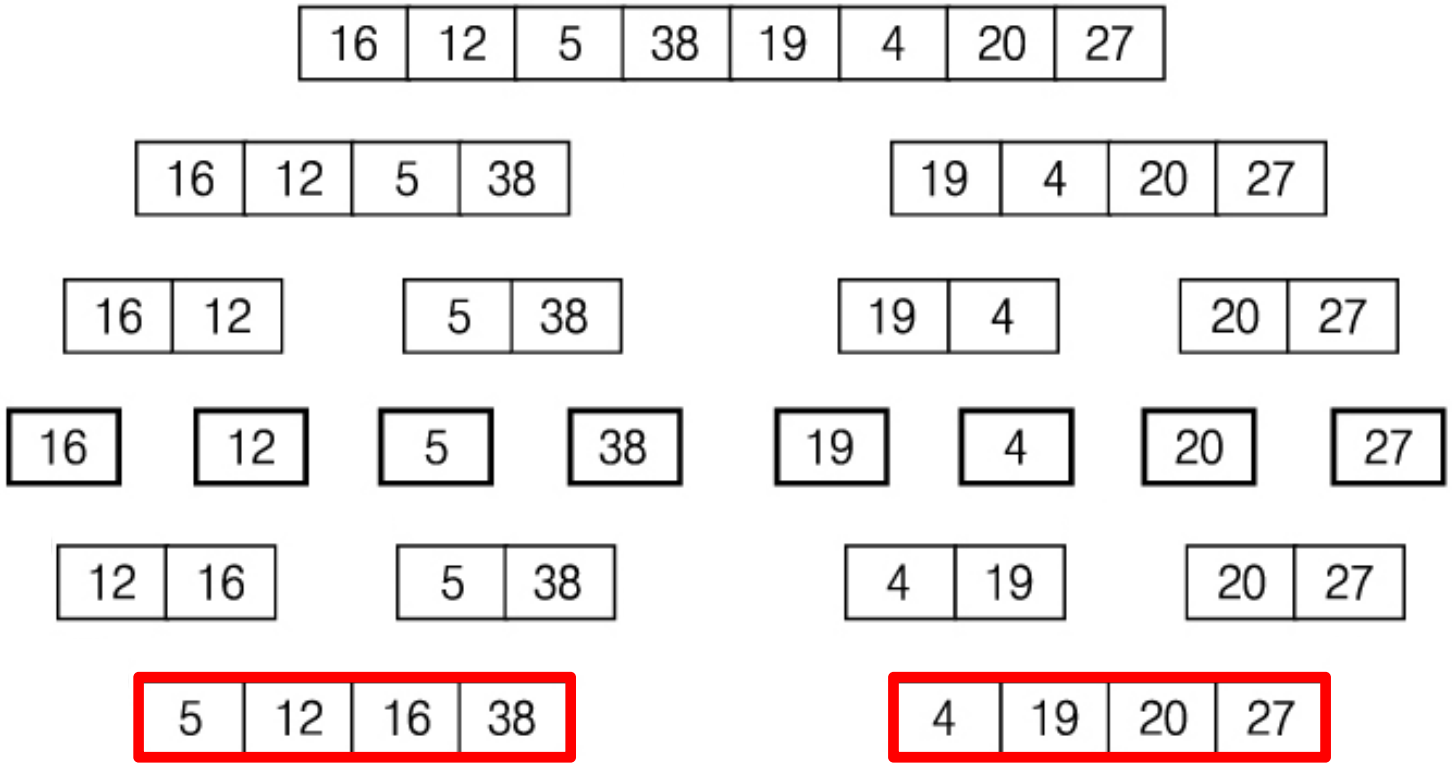
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a ); → sorted
        merge_sort ( 6, 7, a ); → sorted
```



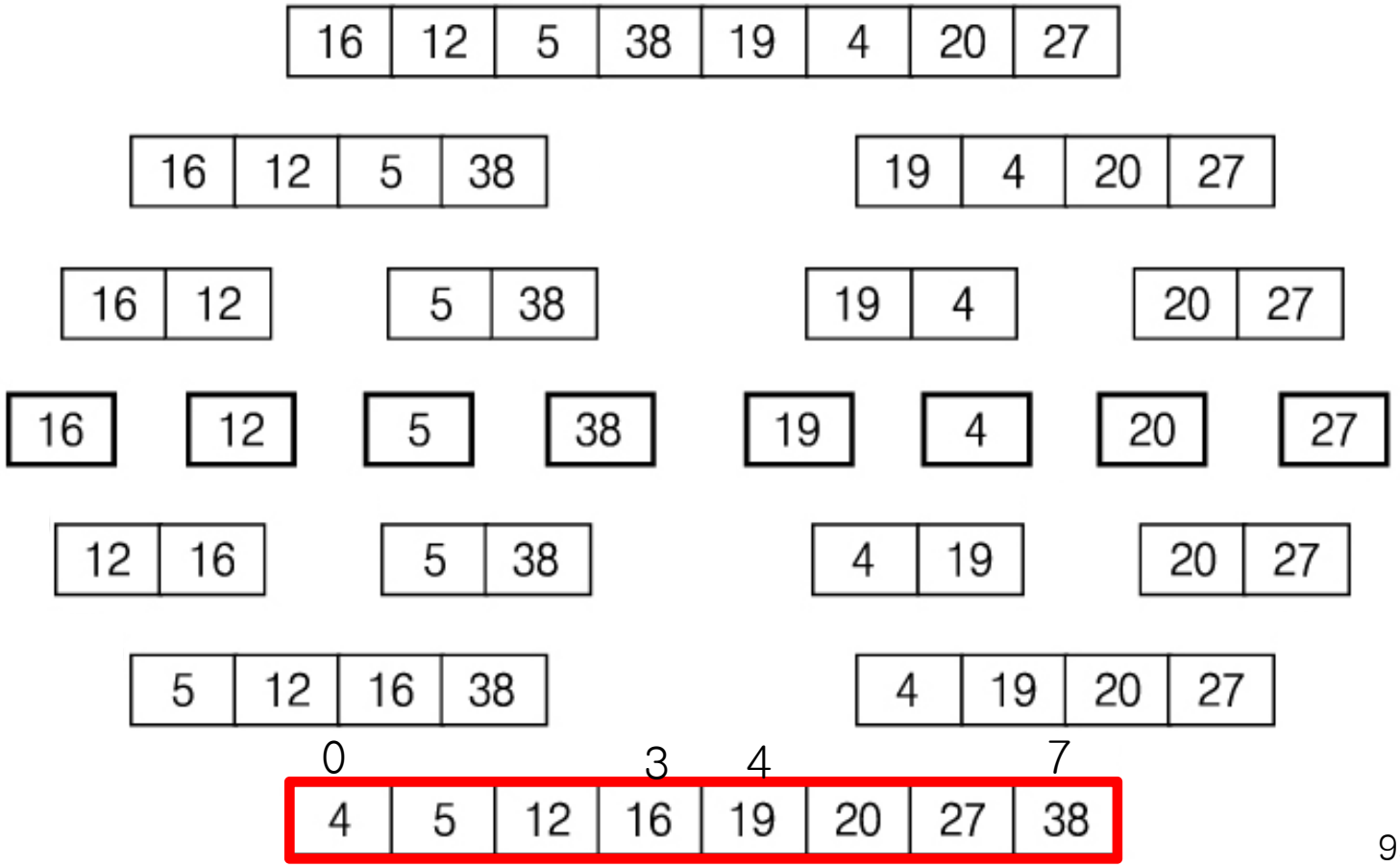
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a );
        m = 5;
        merge_sort ( 4, 5, a ); → sorted
        merge_sort ( 6, 7, a ); → sorted
        merge ( 4, 5, 6, 7, a );
```



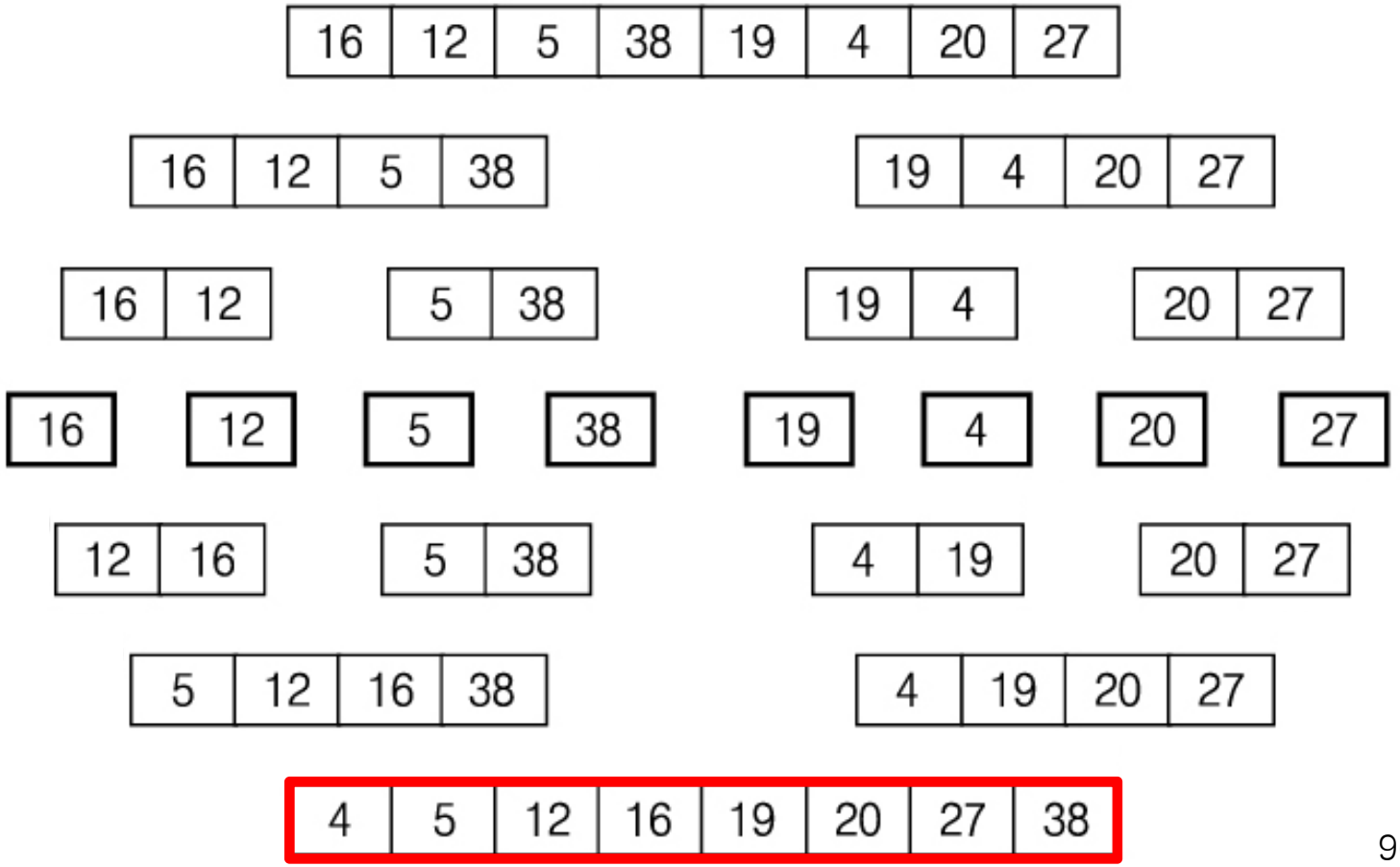
```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a ); → sorted
```



```
merge_sort ( 0, 7, a );
    m = 3;
    merge_sort ( 0, 3, a ); → sorted
    merge_sort ( 4, 7, a ); → sorted
    merge ( 0, 3, 4, 7, a );
```



`merge_sort ( 0, 7, a );` → **sorted**



- Merge sort (Performance analysis)
  - Recurrence relation

$$T(n) = \begin{cases} 2T(n/2) + cn, & n > 1 \\ a, & n = 1 \end{cases}$$

$$T(n) = an + cn \log n$$

- Iterative merge sort
  - A merge sort without recursive call

```
int *iterative_mergesort ( int n, int *a )
{
    Q <- empty Queue;

    for ( int i = 0; i < n; i++ )
        Q.push ( a[i] )

    while ( Q.count ( ) > 1 ) {
        Q.push ( merge ( Q.pop ( ), Q.pop ( ) ) );

    return Q.pop ( );
}
```

- Comparison

	degenerate case	divide	conquer	combine	performance
tournament	n = 1 (s = e)	m = (s+e)/2	champ (s,m); champ (m+1,e);	win (LW, RW);	2T(n/2) + O(1) = O(n)
binary search	n = 1 (s = e)	m = (s+e)/2	bs (s, m-1); or bs (m+1, e);	-	T(n/2) + O(1) = O(log n)

integer multiplication	n = 1	s = n/2
		w = u div 2^s;
		……
			mult (w+x, y+z);
			mult (w, y);
			mult (x, z);
				p 2^n +
				(r – p – q) 2^s +
				q;
					3T(n/2) + O(n)
					=O($n^{\log\_2 3}$)

merge sort	n = 1 (s = e)	m = (s+e)/2	ms (s, m); ms (m+1, e);	merge (s, m, e);	2T(n/2) + O(n) = O(n log n)
quick sort					
median					
matrix multiplication					

퀴즈 3

- 다음은 합병 정렬에 대한 설명이다. 올바른 문장을 모두 고르시오.

  (a) 합병 정렬은 최선의 경우와 최악의 경우에 모두 O(n log n)이 걸리는 알고리즘이다.

  (b) 합병 정렬은 분할-정복-결합의 3단계를 모두 요구한다.

  (c) 합병 정렬의 분할은 O(1)이, 결합은 O(n)의 시간 복잡도가 요구된다.

  (d) merge_sort (0, 7, a)를 완료하기 위해서는 8번의 merge 연산을 수행한다.