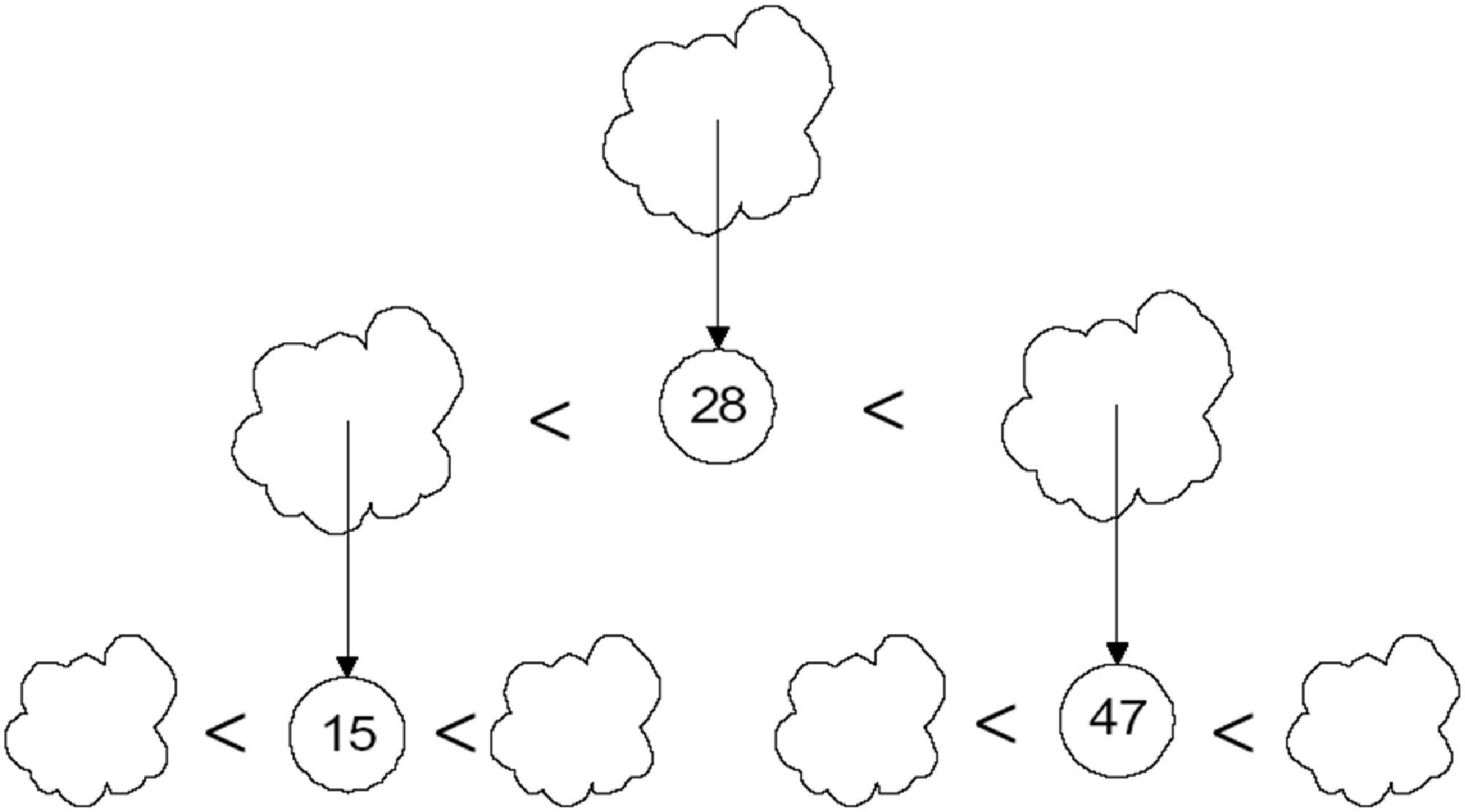
Quick Sort dan Merge Sort

Arna Fariza

Yuliana Setiowati

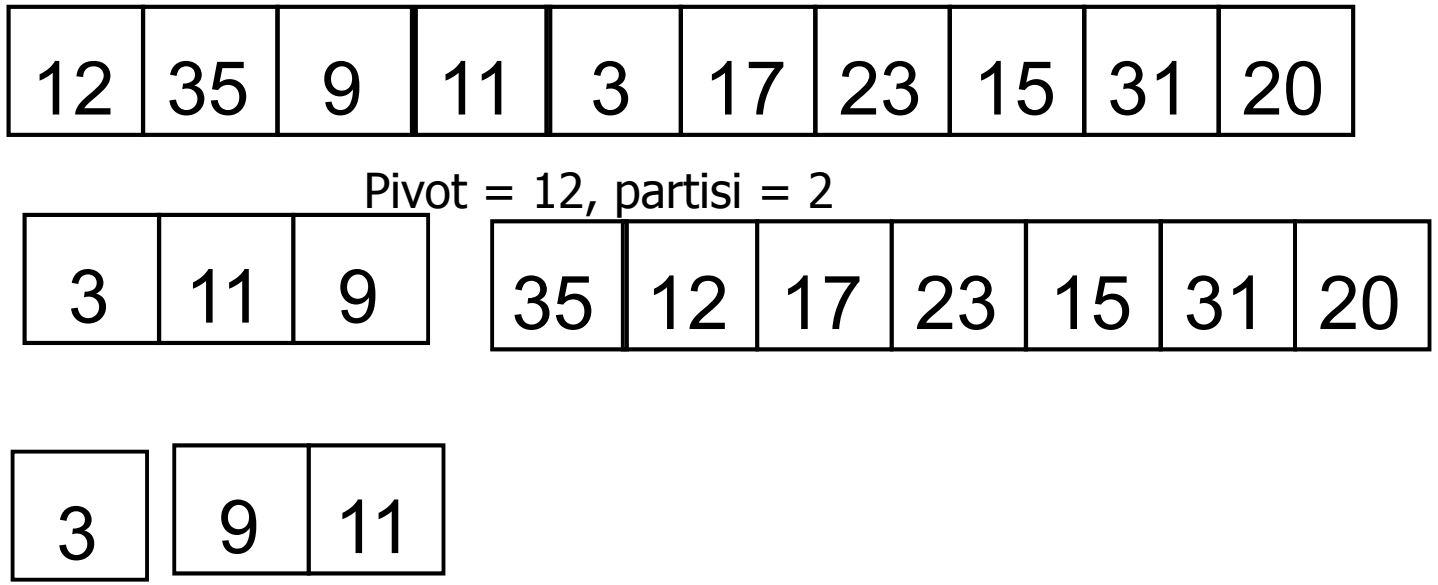
Ide Quicksort



Tentukan “pivot”. Bagi Data menjadi 2 Bagian yaitu Data kurang dari dan Data lebih besar dari pivot. Urutkan tiap bagian tersebut secara rekursif.

Quicksort

- Algoritma divide-and-conquer (membagi dan menyelesaikan)
  - array A[p..r] is *dipartisi* menjadi dua subarray yang tidak empty A[p..q] and A[q+1..r]
    - Invariant: Semua elemen pada A[p..q] lebih kecil dari semua elemen pada A[q+1..r]
  - Subarray diurutkan secara rekursif dengan memanggil quicksort


12 35 9 11 3 17 23 15 31 20
Pivot = 12, partisi = 2
3 11 9
35 12 17 23 15 31 20
3
9 11

Program Quicksort

```
Quicksort(A,p, r)
{
     if (p < r)
     {
          q = Partition(p, r);
          Quicksort(A, p, q);
          Quicksort(A, q+1, r);
     }
}
```

Partisi

- Jelas, semua kegiatan penting berada pada fungsi **`partition()`**
  - Menyusun kembali subarray
  - Hasil akhir :
    - Dua subarray
    - Semua elemen pada subarray pertama < semua nilai pada subarray kedua
  - Mengembalikan indeks pivot yang membagi subarray

Partisi

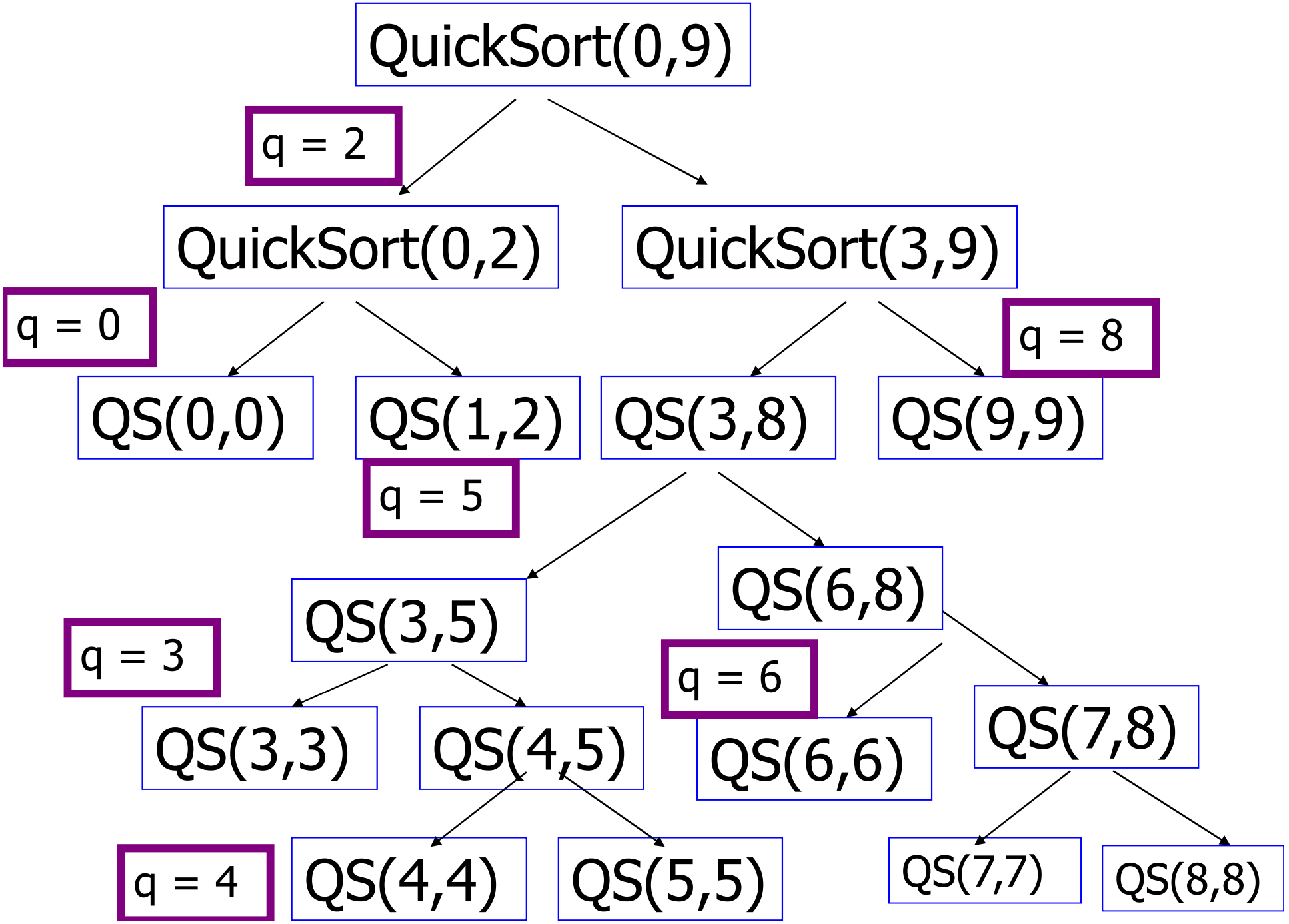
- Partition(A, p, r):
  - Pilih elemen sebagai “pivot”
  - Dua bagian A[p..i] and A[j..r]
    - Semua element pada A[p..i] <= pivot
    - Semua element pada A[j..r] >= pivot
  - Increment i sampai A[i] >= pivot
  - Decrement j sampai A[j] <= pivot
  - Swap A[i] dan A[j]
  - Repeat Until i >= j
  - Return j

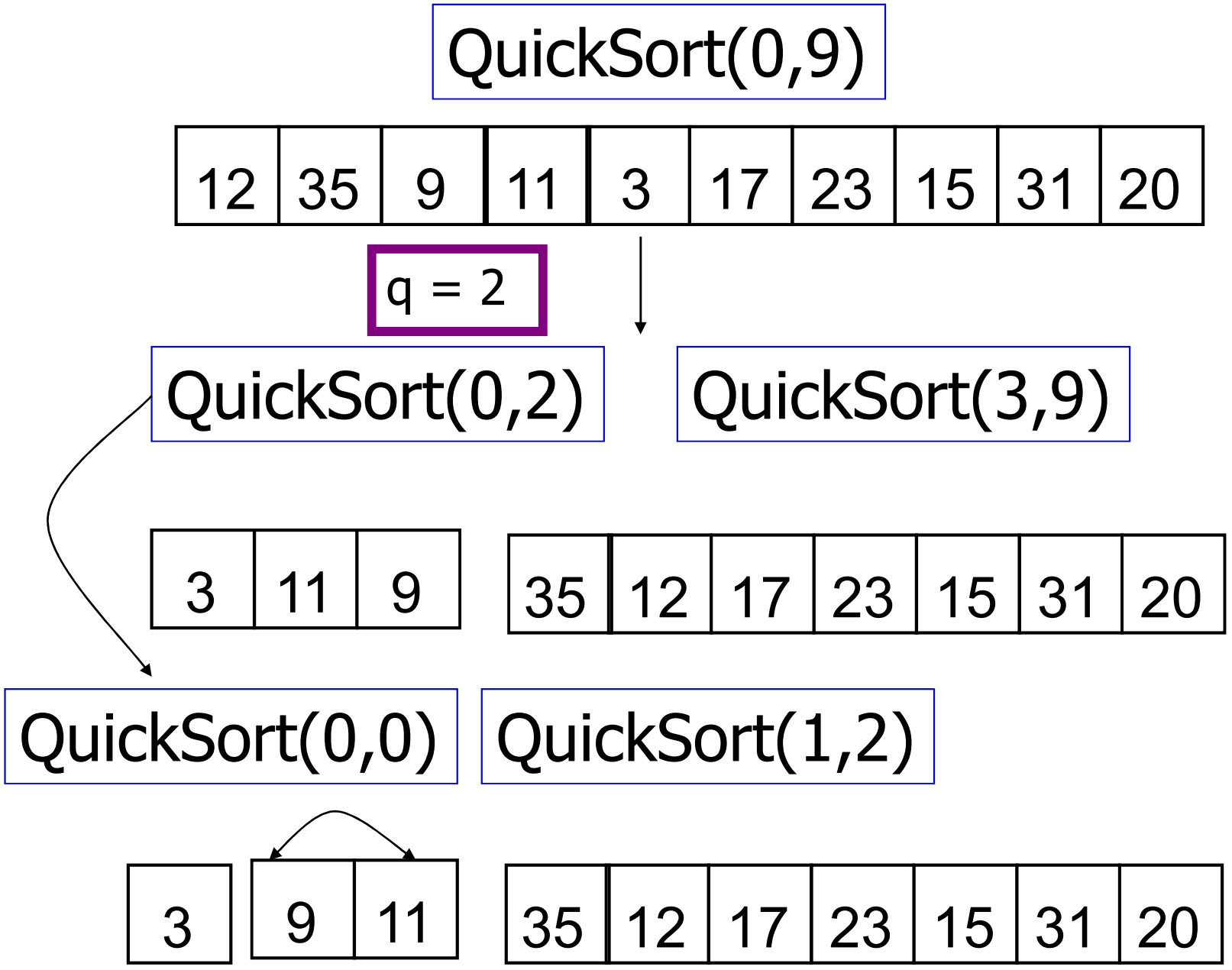
Partition Code

```
Partition(A, p, r)
    x = A[p];
    i = p - 1;
    j = r + 1;
    while (TRUE)
        repeat
            j--;
        until A[j] <= x;
        repeat
            i++;
        until A[i] >= x;
        if (i < j)
            Swap(A, i, j);
        else
            return j;
```

`Partition`(A,0,9)

12	35	9	11	3	17	23	15	31	20


QuickSort(0,9)
q = 2
QuickSort(0,2)
QuickSort(3,9)
q = 0
q = 8
QS(0,0)
QS(1,2)
QS(3,8)
QS(9,9)
q = 5
QS(3,5)
QS(6,8)
q = 3
q = 6
QS(3,3)
QS(4,5)
QS(6,6)
QS(7,8)
q = 4
QS(4,4)
QS(5,5)
QS(7,7)
QS(8,8)


QuickSort(0,9)
12 35 9 11 3 17 23 15 31 20
q = 2
QuickSort(0,2)
QuickSort(3,9)
3 11 9
35 12 17 23 15 31 20
QuickSort(0,0)
QuickSort(1,2)
3
9 11
35 12 17 23 15 31 20

12	35	9	11	3	17	23	15	31	20

QuickSort(0,9)

- X = PIVOT merupakan indeks ke –0
- PIVOT = 12
- terdapat variabel i dan j , i=0 , j=9
- variabel i untuk mencari bilangan yang lebih besar dari PIVOT. Cara kerjanya : selama Data[i] < PIVOT maka nilai i ditambah.
- variabel j untuk mencari bilangan yang lebih kecil dari PIVOT. Cara kerjanya : selama Data[j] > PIVOT maka nilai j dikurangi

q = Partition(0,9)

12	35	9	11	3	17	23	15	31	20

PIVOT = 12

i = 0 j = 4

i < j maka SWAP

SWAP

3	35	9	11	12	17	23	15	31	20

PIVOT = 12

i = 1 j = 3

i < j maka SWAP

SWAP

3	11	9	35	12	17	23	15	31	20

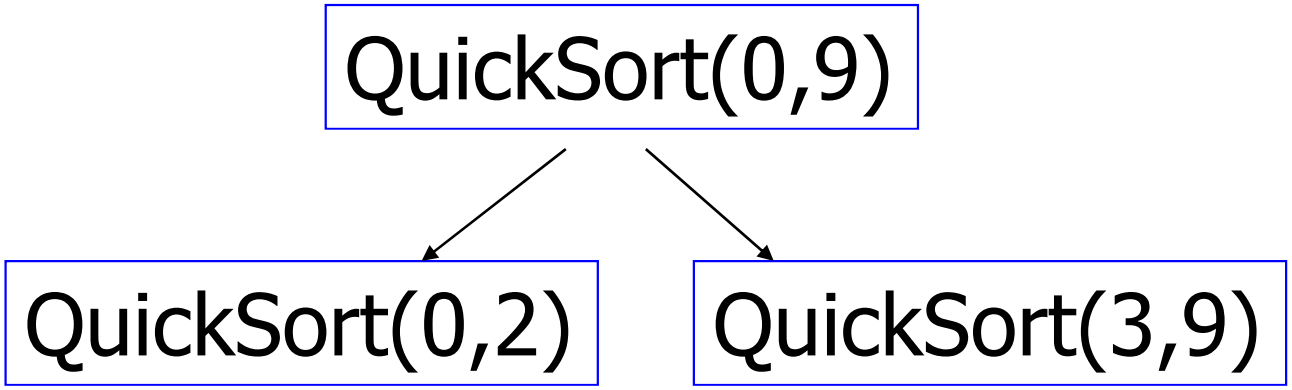
PIVOT = 12

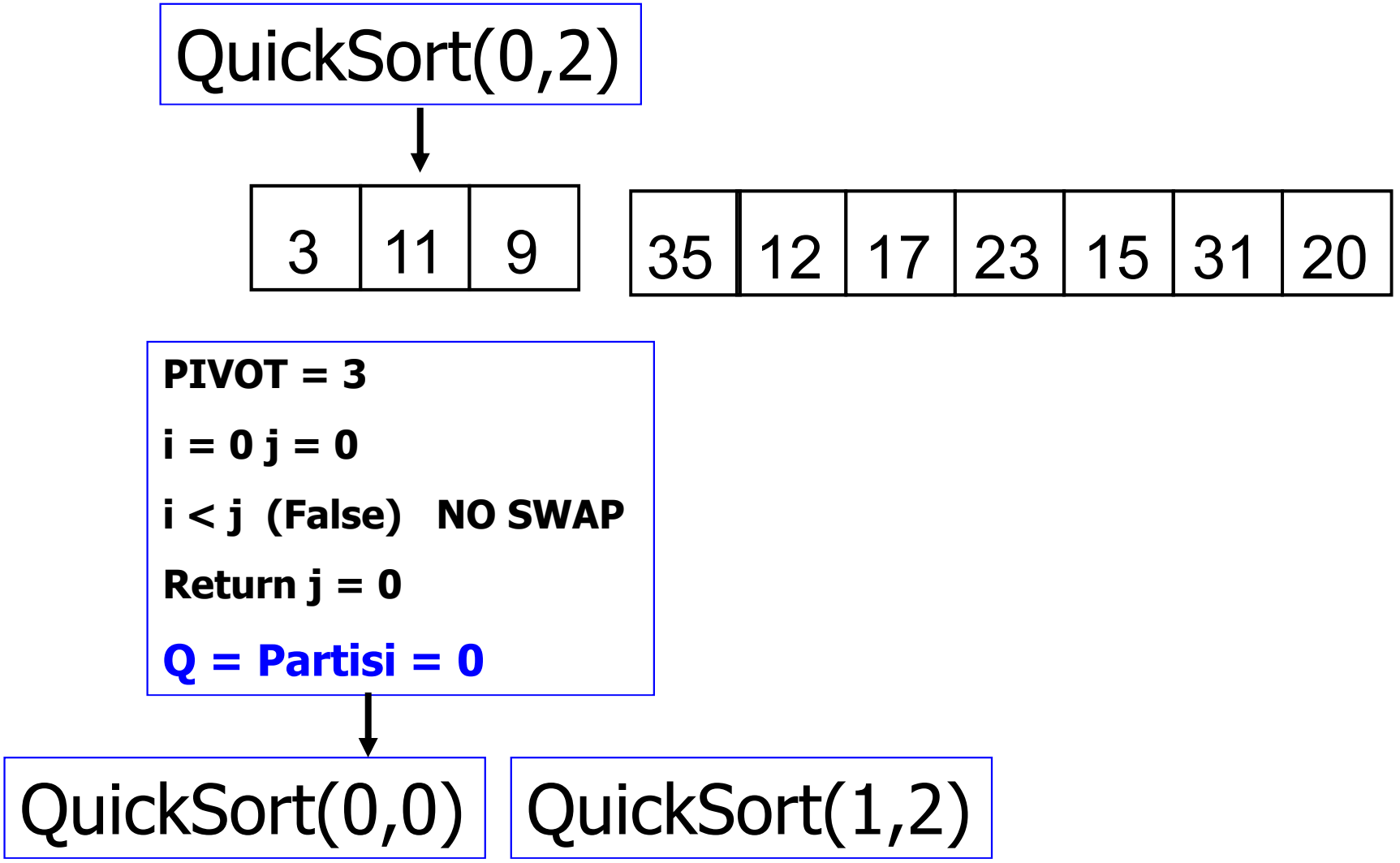
i = 3 j = 2

i < j (False) NO SWAP

Return j = 2

Q = Partisi = 2



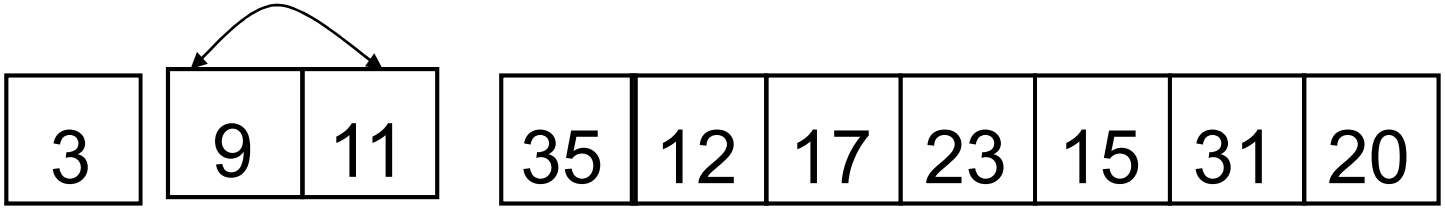

QuickSort(0,2)
3
11
9
35
12
17
23
15
31
20
PIVOT = 3
i = 0 j = 0
i < j (False) NO SWAP
Return j = 0
Q = Partisi = 0
QuickSort(0,0)
QuickSort(1,2)

QuickSort(1,2)

PIVOT = 11

i = 1 j = 2

i < j SWAP



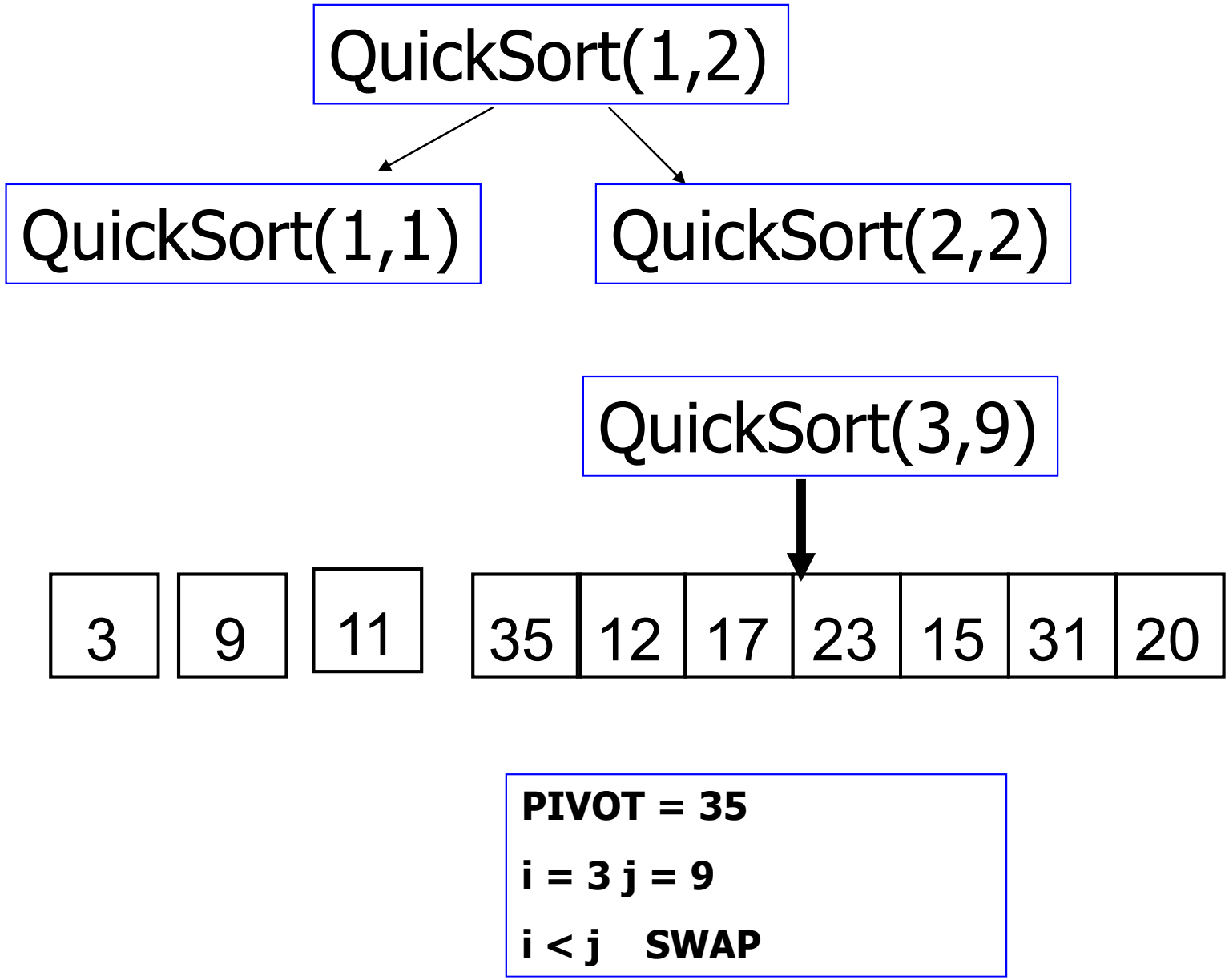
PIVOT = 11

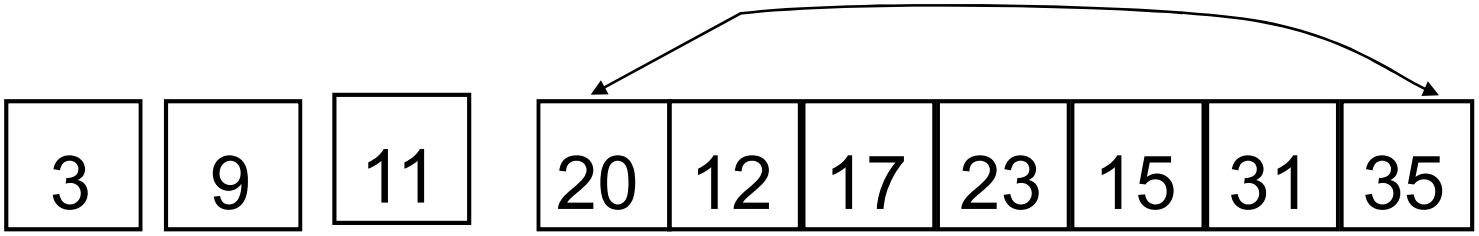
i = 2 j = 1

i < j NO SWAP

Return j = 1

Q = Partisi = 1


QuickSort(1,2)
QuickSort(1,1)
QuickSort(2,2)
QuickSort(3,9)
3
9
11
35
12
17
23
15
31
20
PIVOT = 35
i = 3 j = 9
i < j SWAP



PIVOT = 35

i = 9 j = 8

i < j NO SWAP

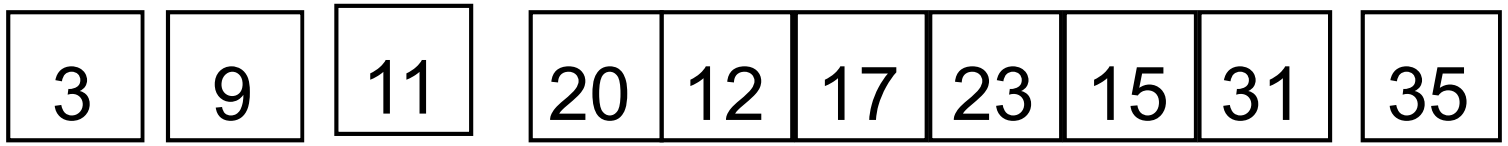
Return j = 8

Q = Partisi = 8

QuickSort(3,9)

QuickSort(3,8)

QuickSort(9,9)

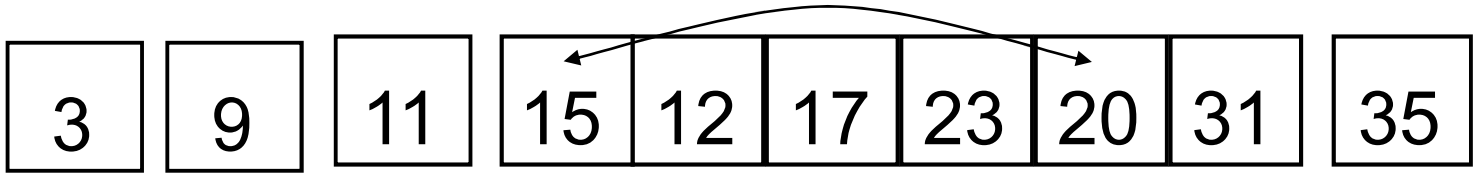


QuickSort(3,8)

PIVOT = 20

i = 3 j = 7

i < j SWAP



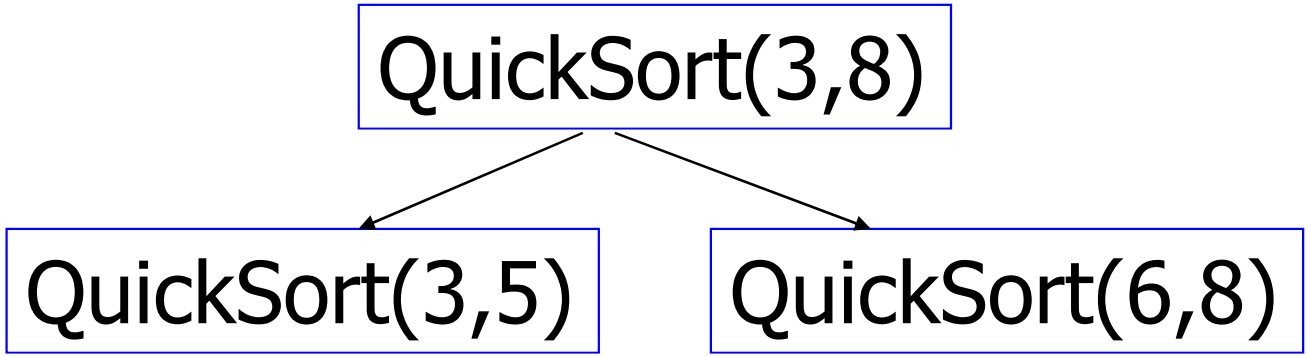
PIVOT = 20

i = 6 j = 5

i < j NO SWAP

Return j = 5

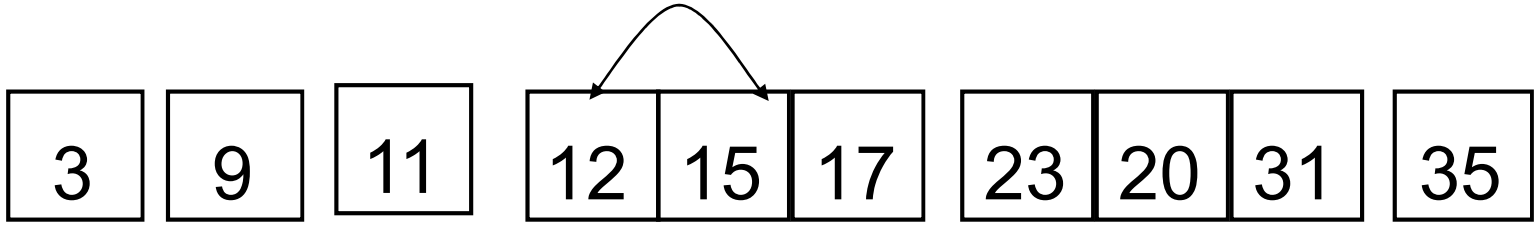
Q = Partisi = 5



PIVOT = 15

i = 3 j = 4

i < j SWAP



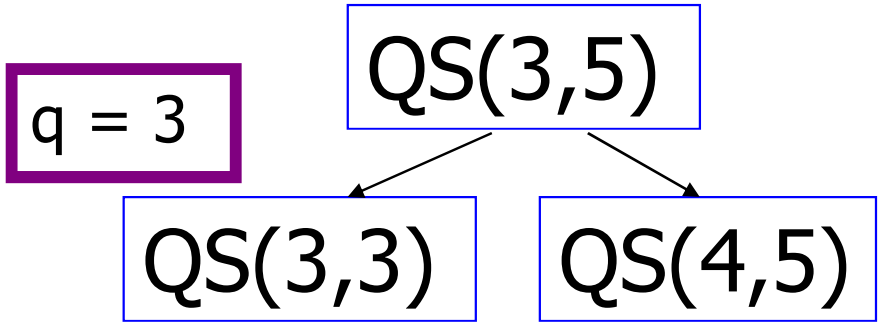
PIVOT = 15

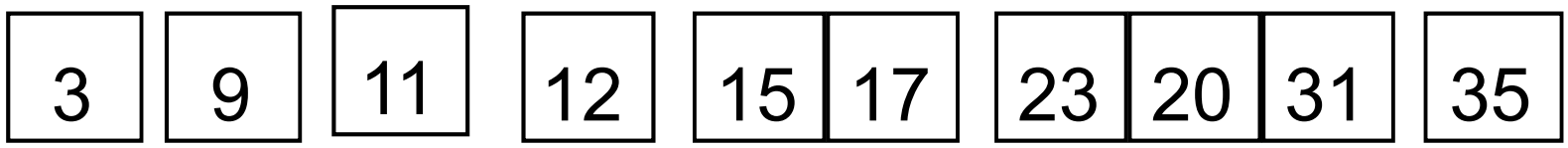
i = 4 j = 3

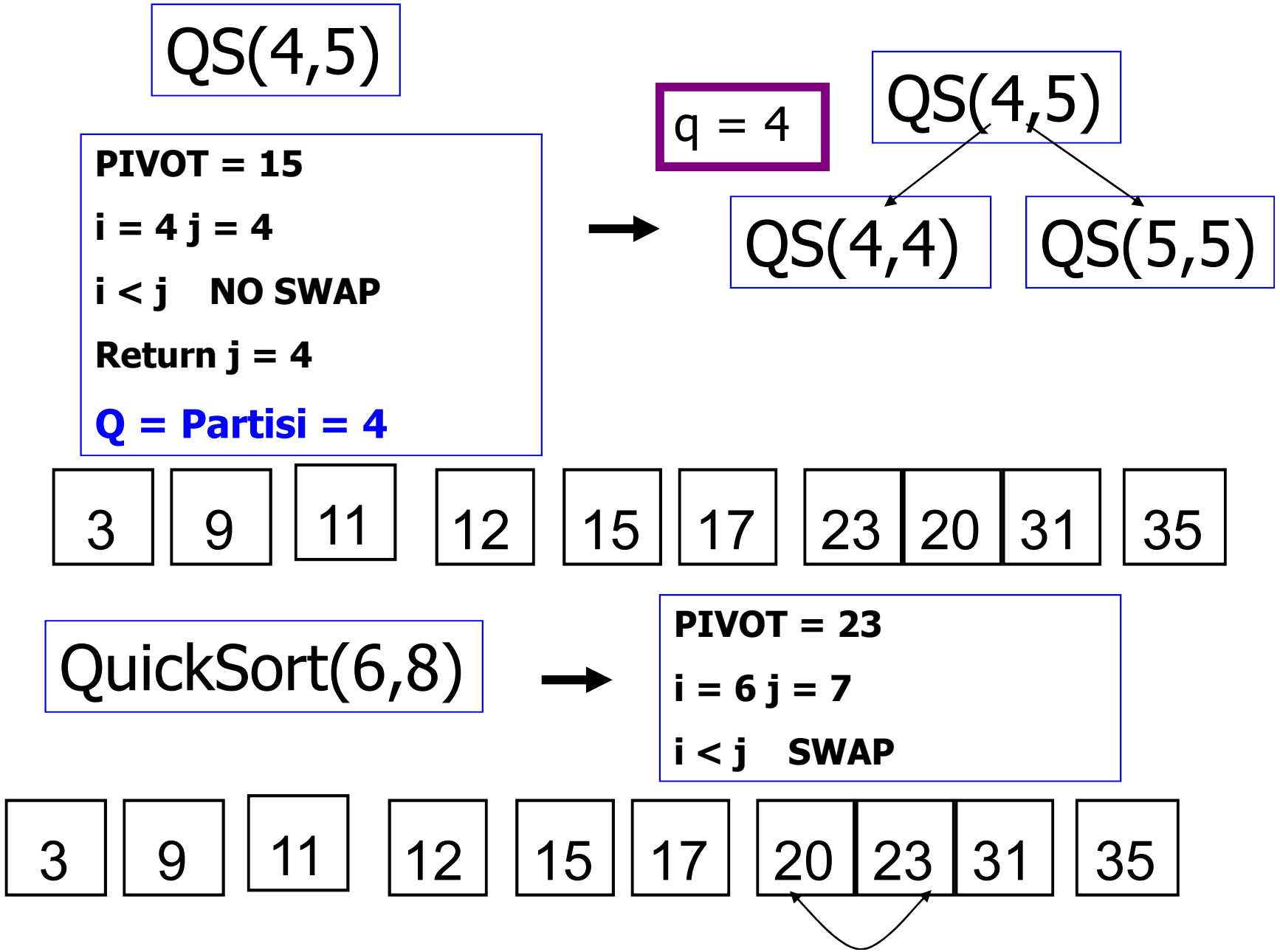
i < j NO SWAP

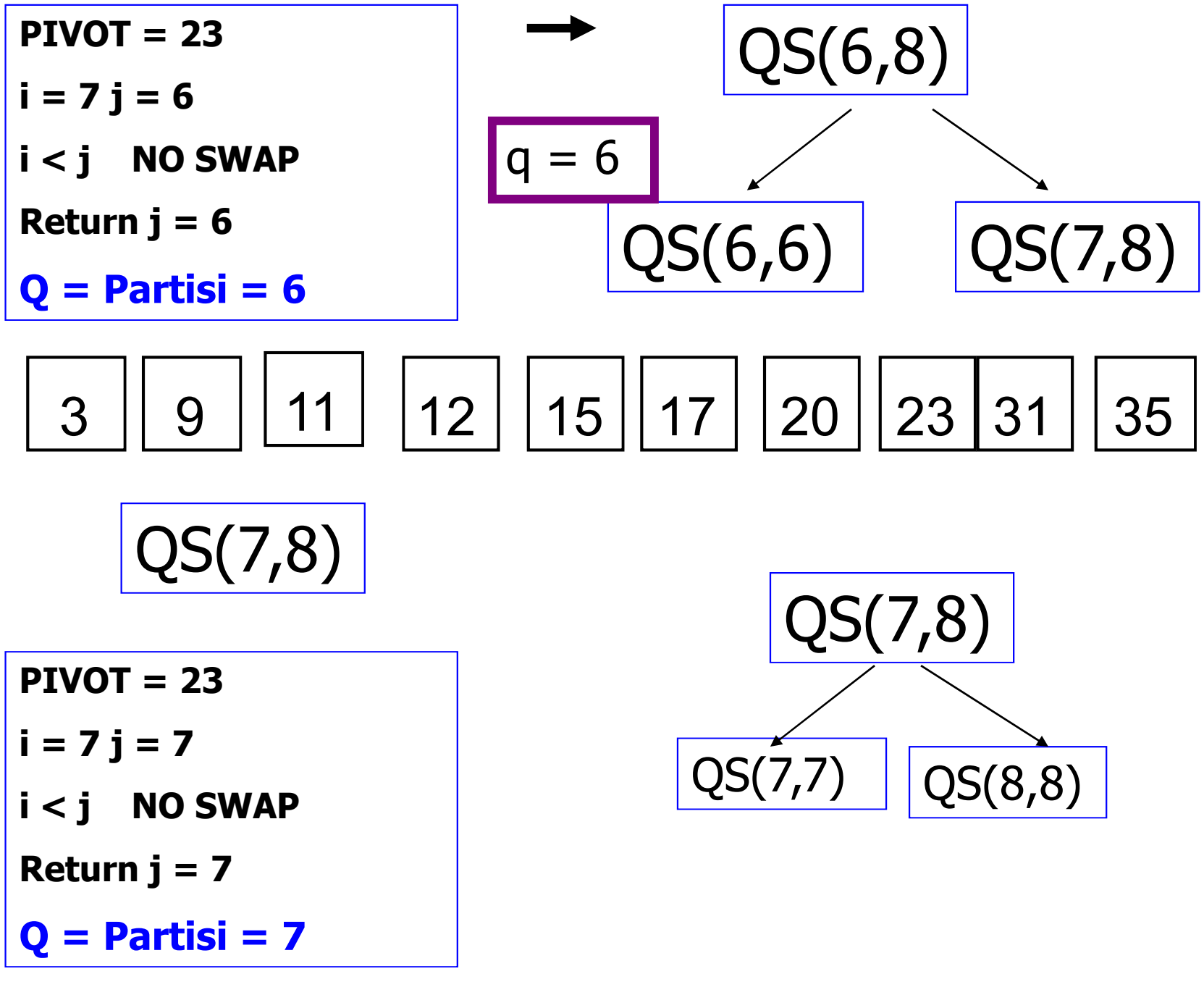
Return j = 3

Q = Partisi = 3






QS(4,5)
PIVOT = 15
i = 4 j = 4
i < j NO SWAP
Return j = 4
Q = Partisi = 4
q = 4
QS(4,5)
QS(4,4)
QS(5,5)
3
9
11
12
15
17
23
20
31
35
QuickSort(6,8)
PIVOT = 23
i = 6 j = 7
i < j SWAP
3
9
11
12
15
17
20
23
31
35


PIVOT = 23
i = 7 j = 6
i < j NO SWAP
Return j = 6
Q = Partisi = 6
QS(6,8)
q = 6
QS(6,6)
QS(7,8)
3
9
11
12
15
17
20
23
31
35
QS(7,8)
PIVOT = 23
i = 7 j = 7
i < j NO SWAP
Return j = 7
Q = Partisi = 7
QS(7,8)
QS(7,7)
QS(8,8)

Analisa Quicksort

- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?

Analisa Quicksort

- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?
  - Rekursif
    1. Partisi membagi array menjadi dua subarray dengan ukuran n/2
    2. Quicksort untuk tiap subarray

Quicksort Analysis

- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?
  - Rekursif
    1. Partisi membagi array menjadi dua subarray dengan ukuran n/2
    2. Quicksort untuk tiap subarray
  - Berapa Waktu Rekursif ?

Quicksort Analysis

- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?
  - Rekursif
    1. Partisi membagi array menjadi dua subarray dengan ukuran n/2
    2. Quicksort untuk tiap subarray
  - Berapa Waktu Rekursif ?$O(\log_2 n)$

Quicksort Analysis

- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?
  - Rekursif
    1. Partisi membagi array menjadi dua subarray dengan ukuran n/2
    2. Quicksort untuk tiap subarray

  - Berapa Waktu Rekursif ? $O(\log_2 n)$
  - Jumlah pengaksesan partisi ?

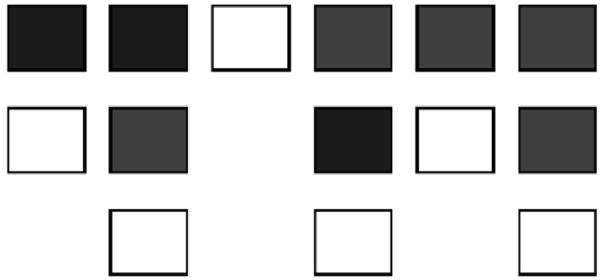
Quicksort Analysis

- Misalkan pivot dipilih secara random.
- Berapa hasil running time untuk Best Case ?
  - Rekursif
    1. Partisi membagi array menjadi dua subarray dengan ukuran n/2
    2. Quicksort untuk tiap subarray

  - Berapa Waktu Rekursif ? $O(\log_2 n)$
  - Jumlah pengaksesan partisi ? $O(n)$

Quicksort Analysis

- Diasumsikan bahwa pivot dipilih secara random
- **Running Time untuk Best case : $O(n \log_2 n)$**
  - Pivot selalu berada ditengah elemen
  - Tiap pemanggilan rekursif array dibagi menjadi dua subaarray dengan ukuran yang sama, Bagian sebelah kiri pivot : elemennya lebih kecil dari pivot, Bagian sebelah kanan pivot : elemennya lebih besar dari pivot



Quicksort Analysis

- Diasumsikan bahwa pivot dipilih secara random
- **Running Time untuk Best case : O(n $\log_2 n$)**
- Berapa running time untuk Worst case?

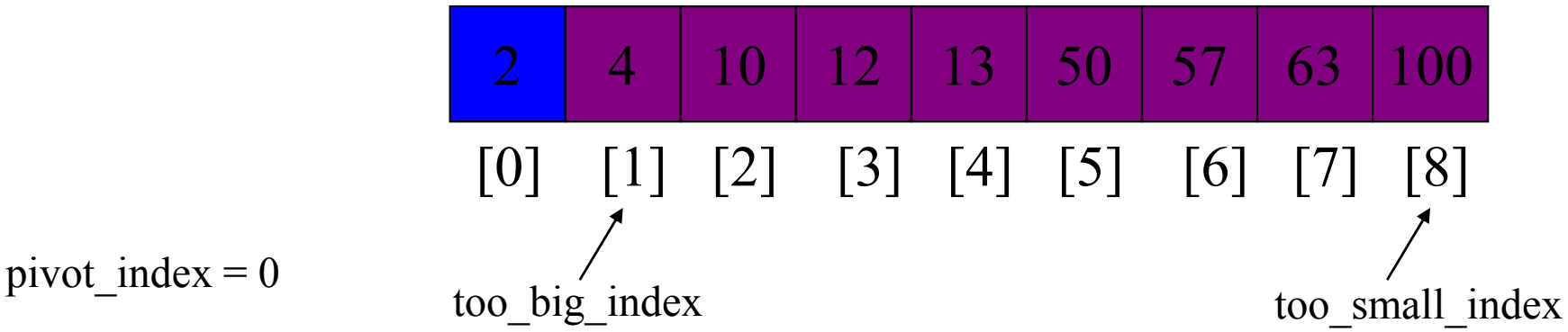
Quicksort Analysis

Worst case: $O(N^2)$

- Pivot merupakan elemen terbesar atau terkecil pada tiap pemanggilan rekursif, sehingga menjadi suatu bagian yang lebih kecil pivot, pivot dan bagian yang kosong

Quicksort: Worst Case

- Dimisalkan elemen pertama dipilih sebagai pivot
- Assume first element is chosen as pivot.
- Misalkan terdapat array yang sudah urut



Quicksort Analysis

- **Best** case running time: **$O(n \log_2 n)$**
- **Worst** case running time: **$O(n^2)$**
- **Average** case running time: **$O(n \log_2 n)$**

Kesimpulan Algoritma Sorting

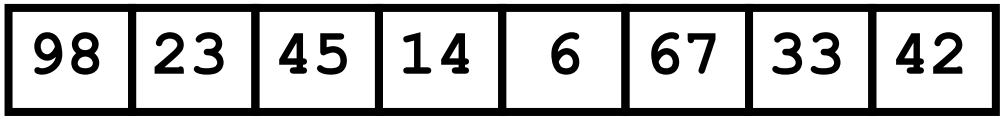
Algorithm	Time	Notes
selection-sort	$O(n^2)$	▪ in-place ▪ lambat (baik untuk input kecil)
insertion-sort	$O(n^2)$	▪ in-place ▪ lambat (baik untuk input kecil)
quick-sort	$O(n \log_2 n)$ expected	▪ in-place, randomized ▪ paling cepat (Bagus untuk data besar)
merge-sort	$O(n \log_2 n)$	▪ sequential data access ▪ cepat (Bagus untuk data besar)

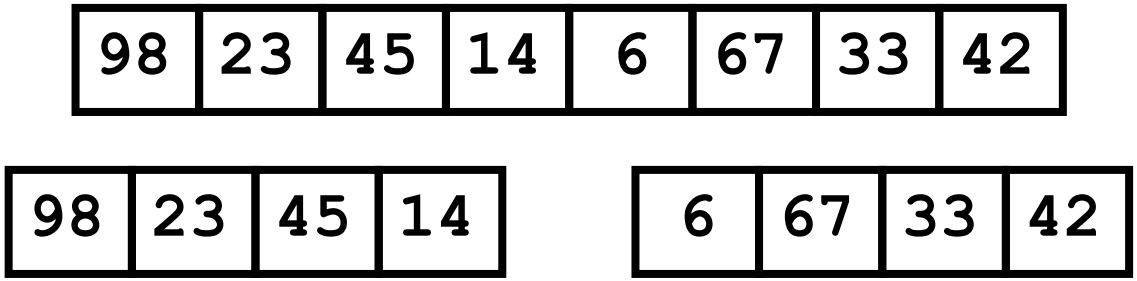
Sorting Algorithms – Merge Sort

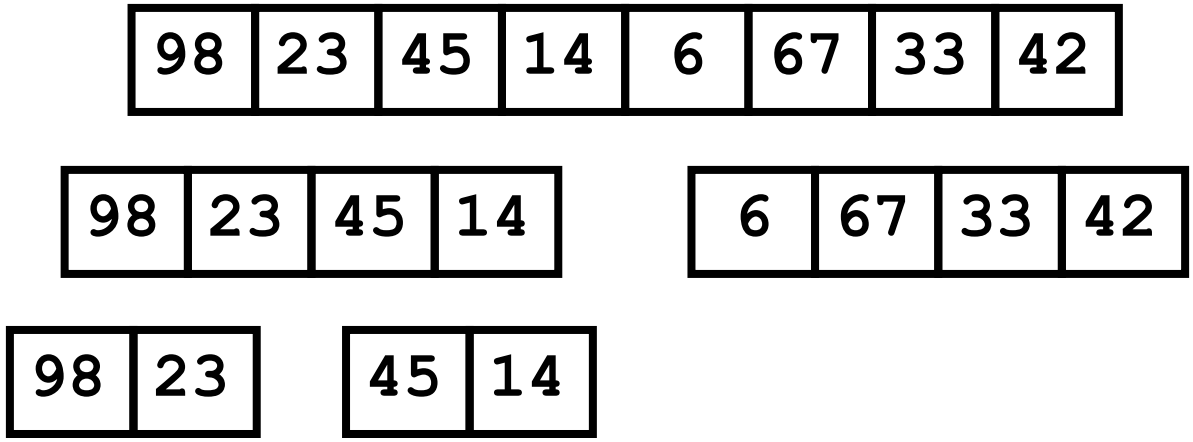
Mergesort

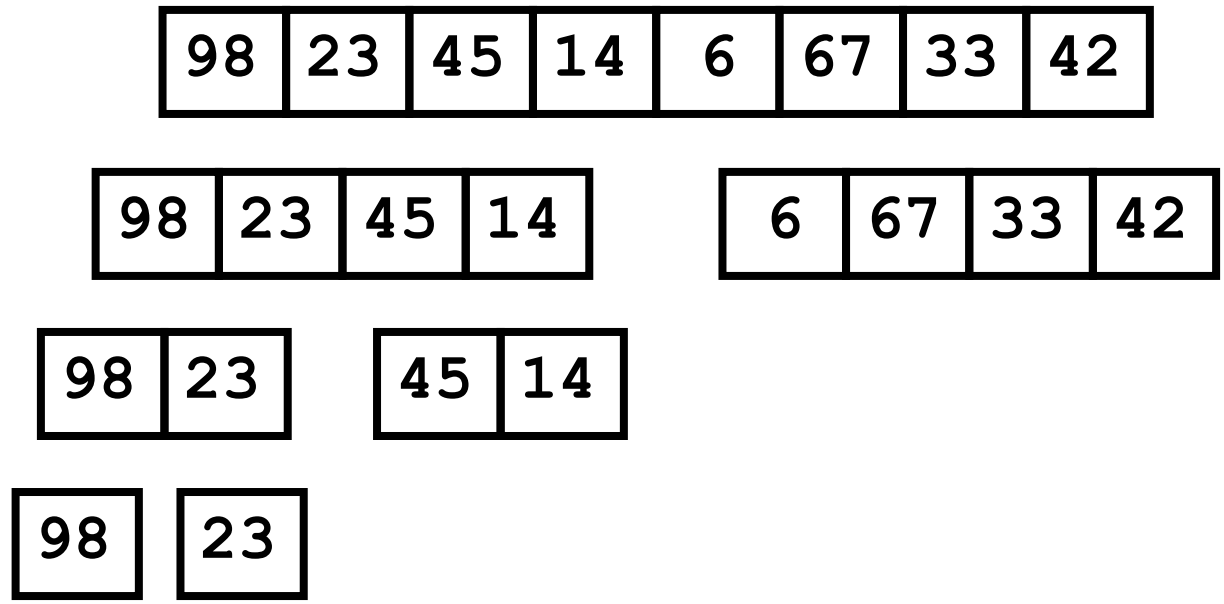
- **Merupakan algoritma divide-and-conquer (membagi dan menyelesaikan)**
- **Membagi array menjadi dua bagian sampai subarray hanya berisi satu elemen**
- **Mengabungkan solusi sub-problem :**
  - **Membandingkan elemen pertama subarray**
  - **Memindahkan elemen terkecil dan meletakkannya ke array hasil**
  - **Lanjutkan Proses sampai semua elemen berada pada array hasil**

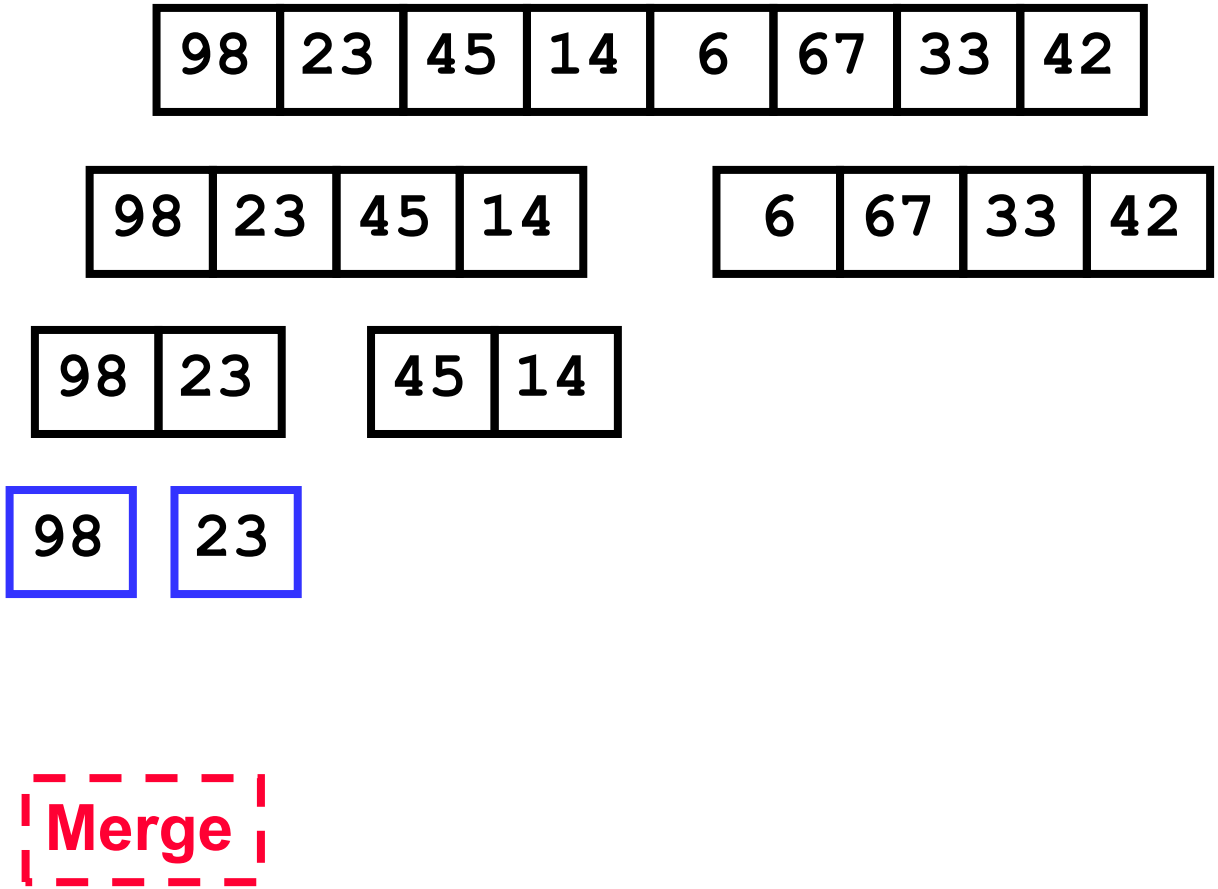
37	23	6	89	15	12	2	19

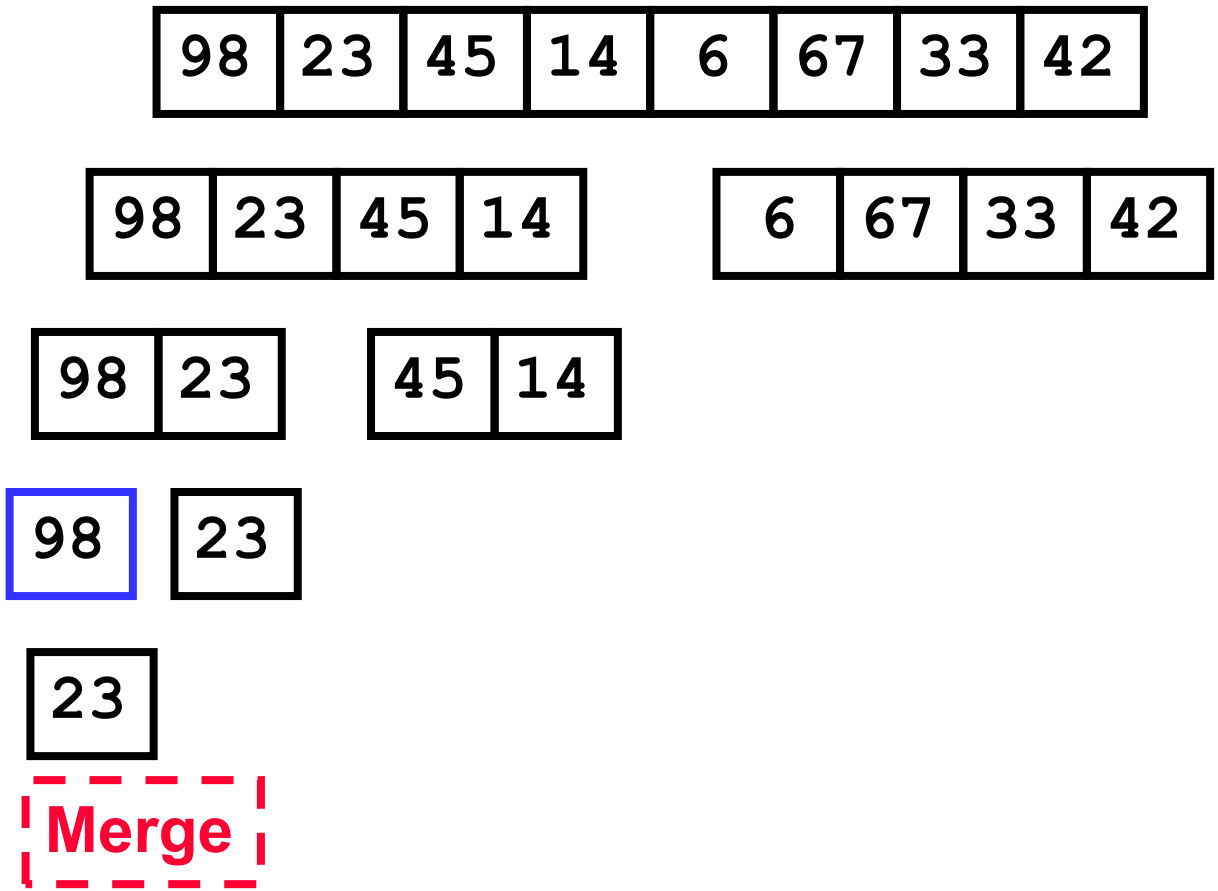

98
23
45
14
6
67
33
42

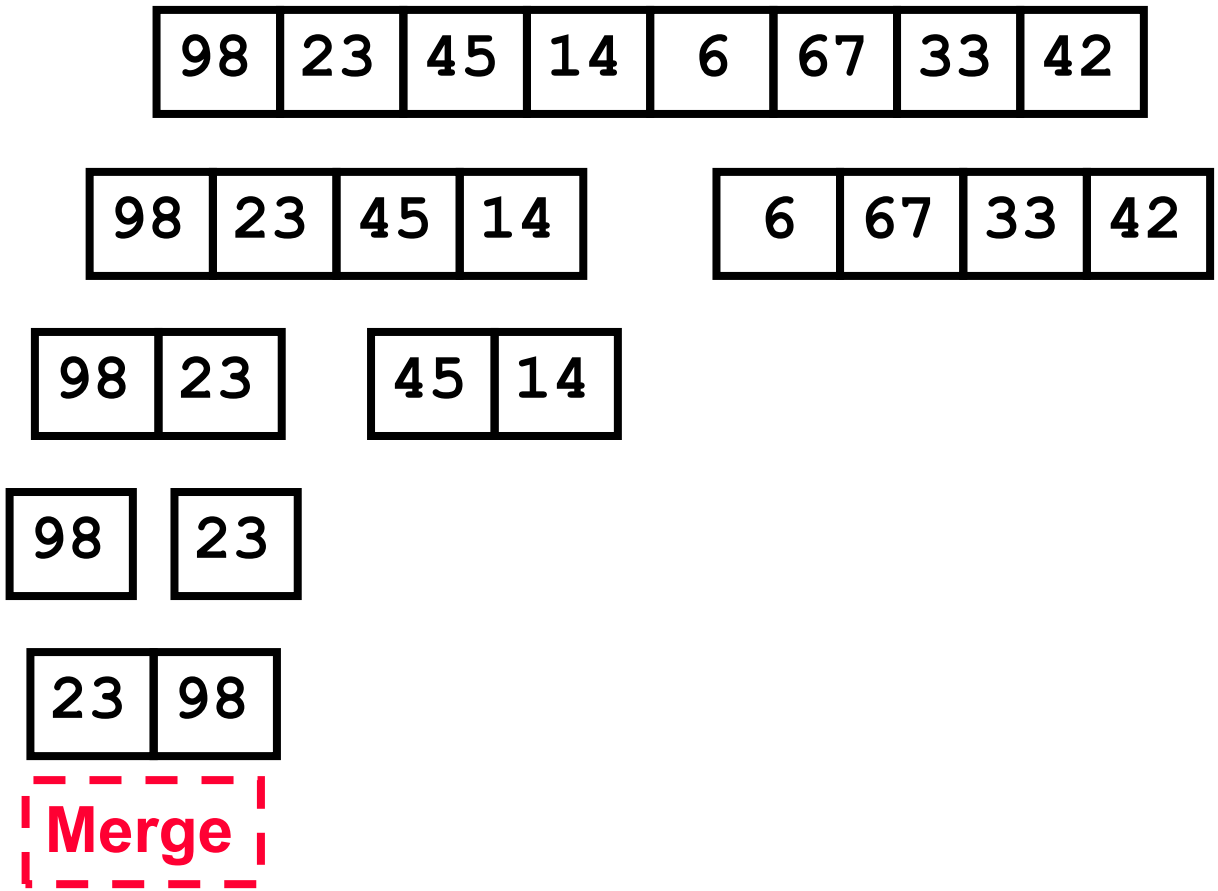

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42

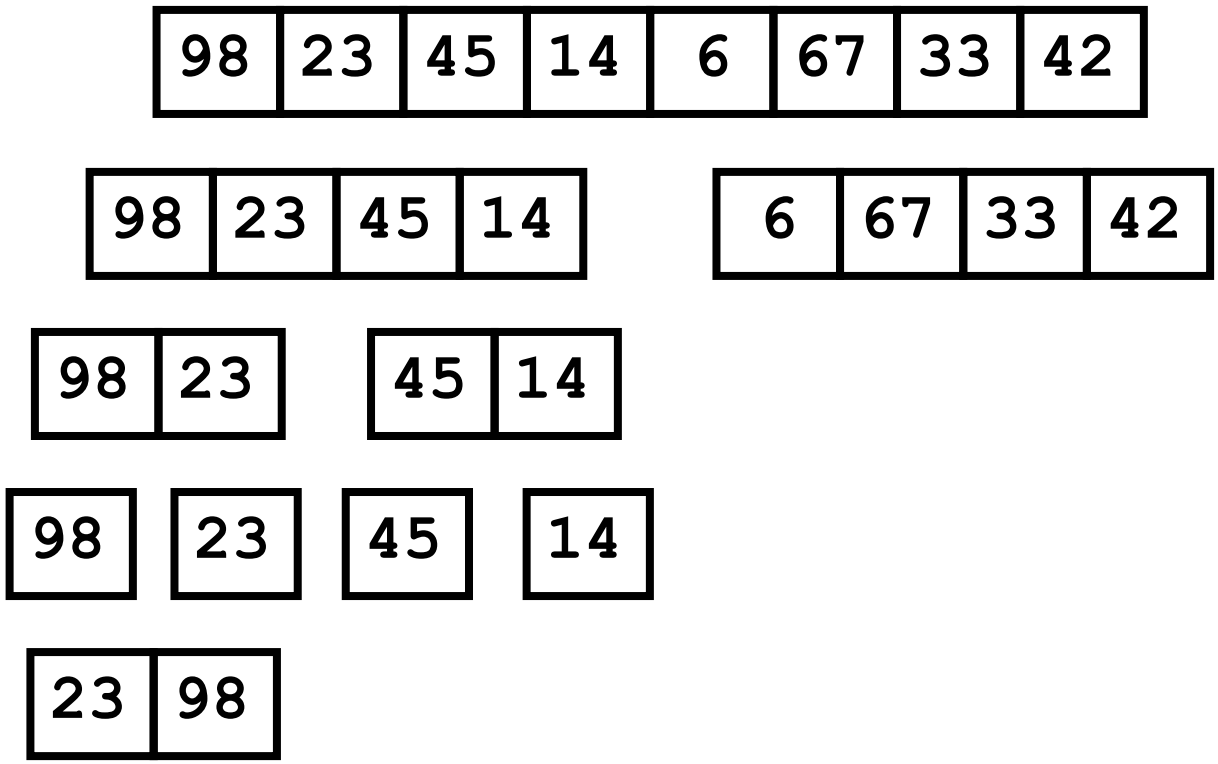

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14

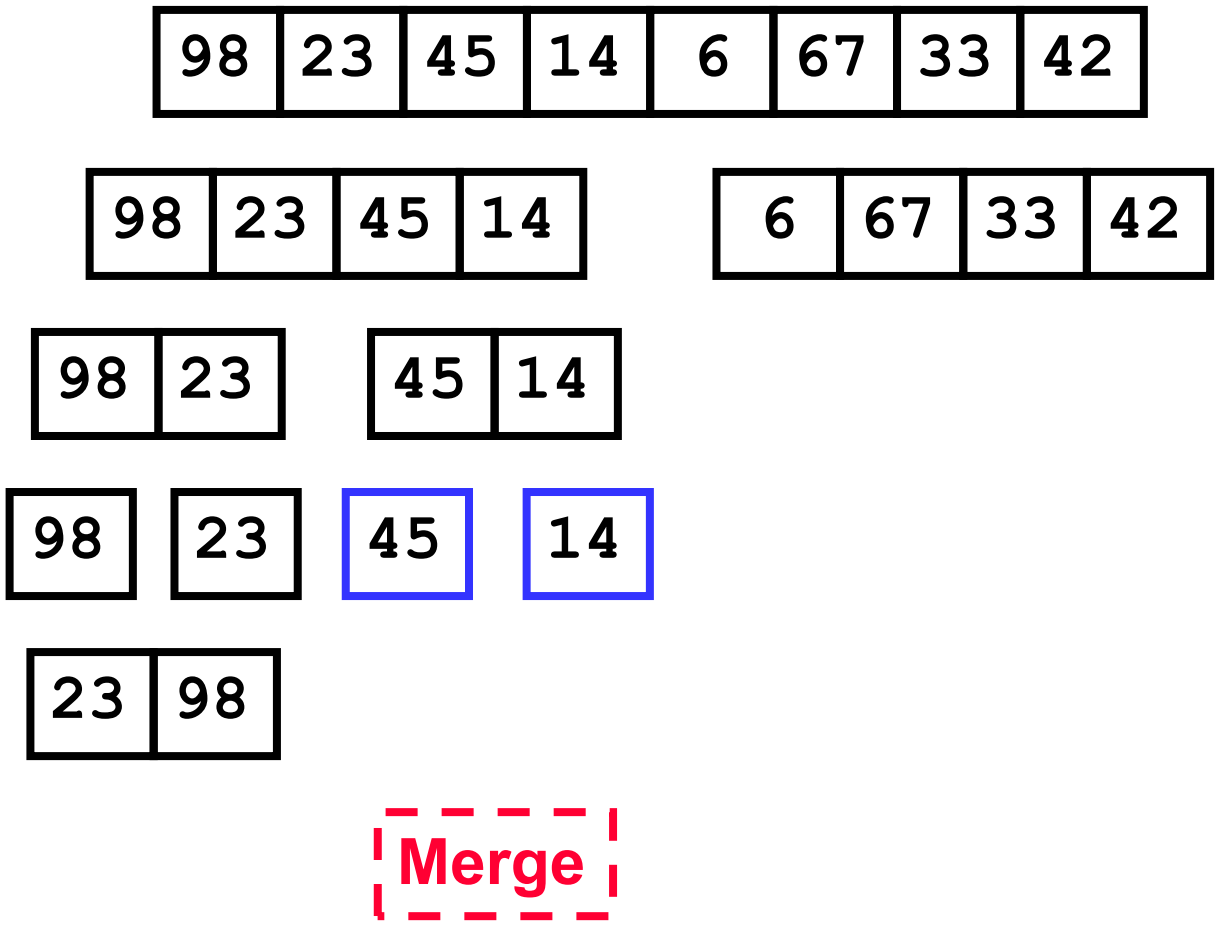

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23

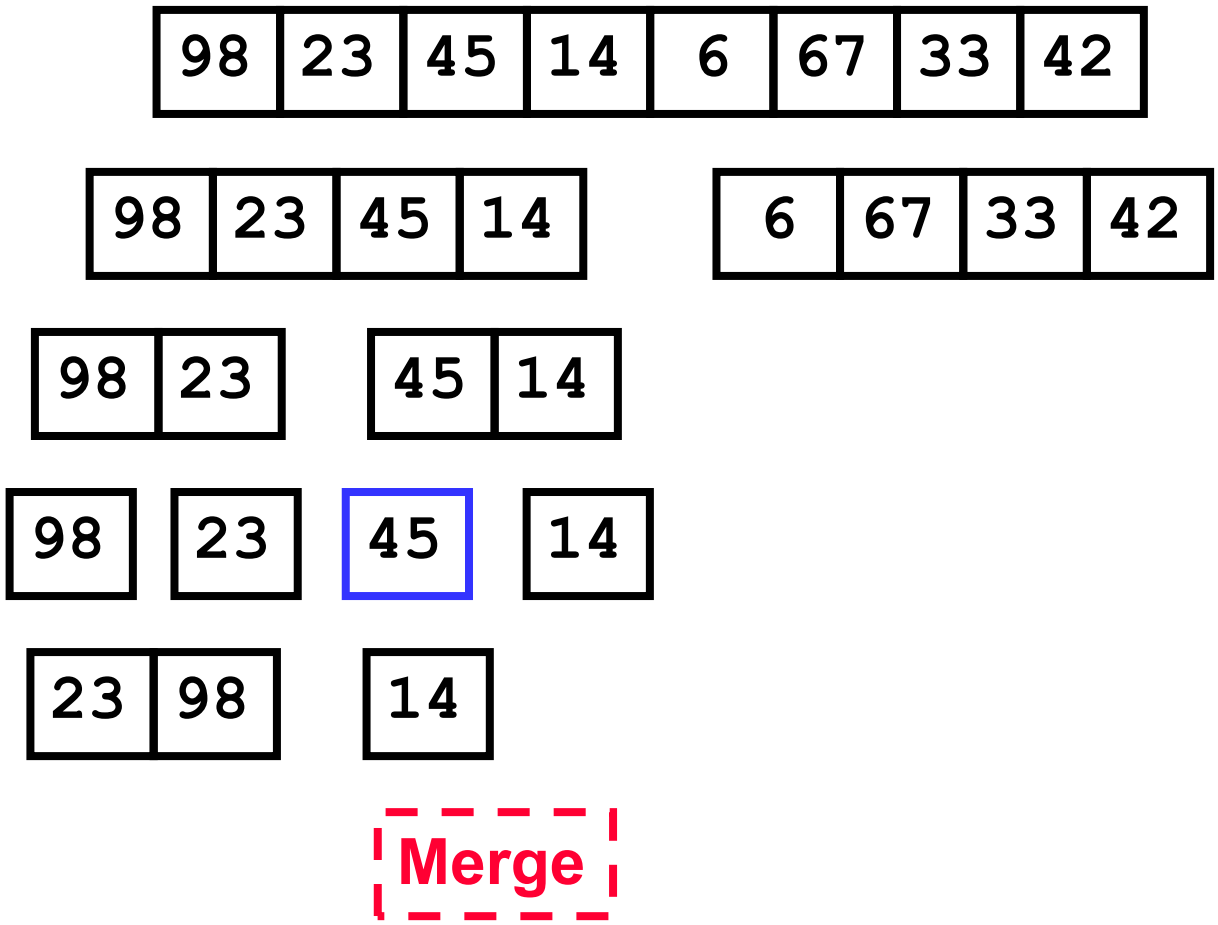

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
Merge

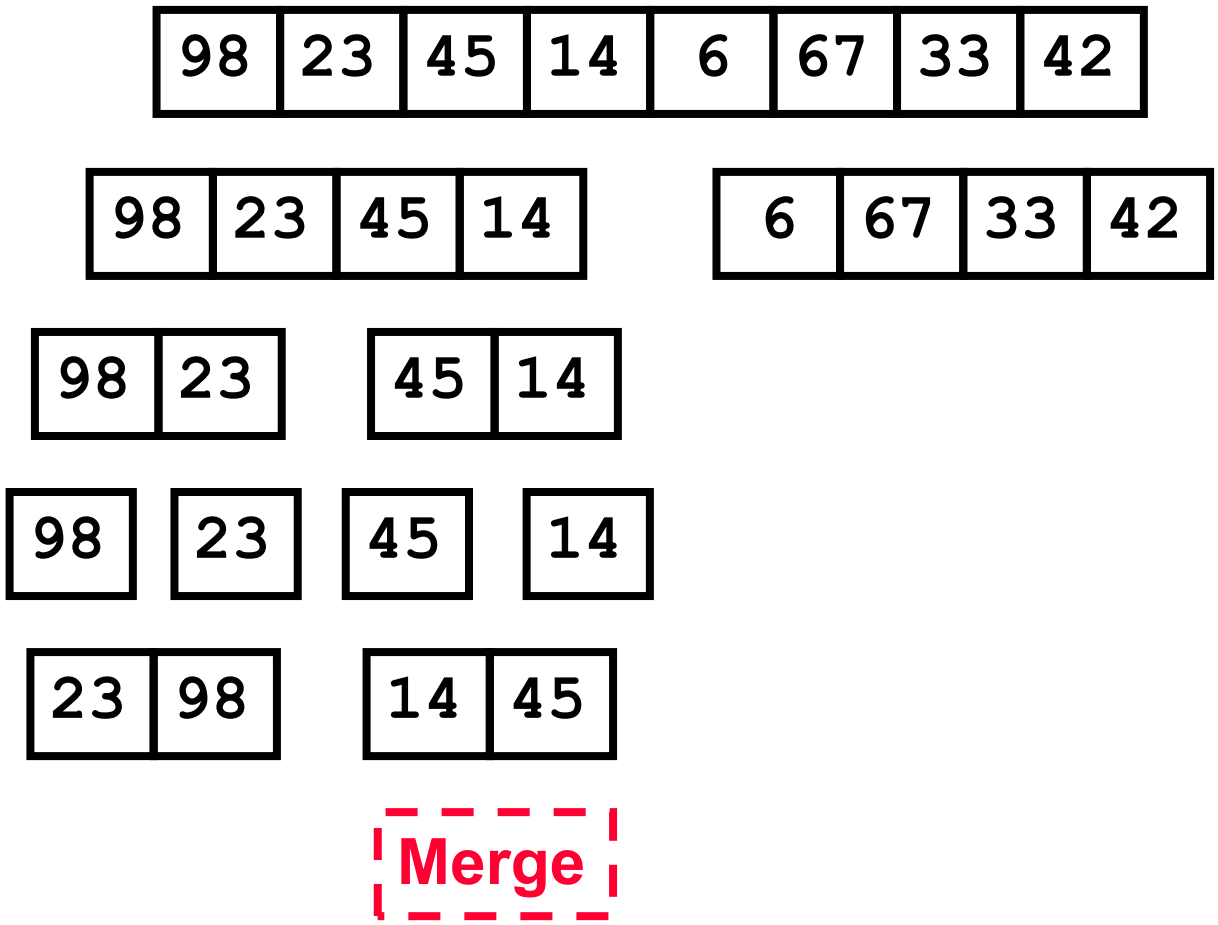

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
23
Merge

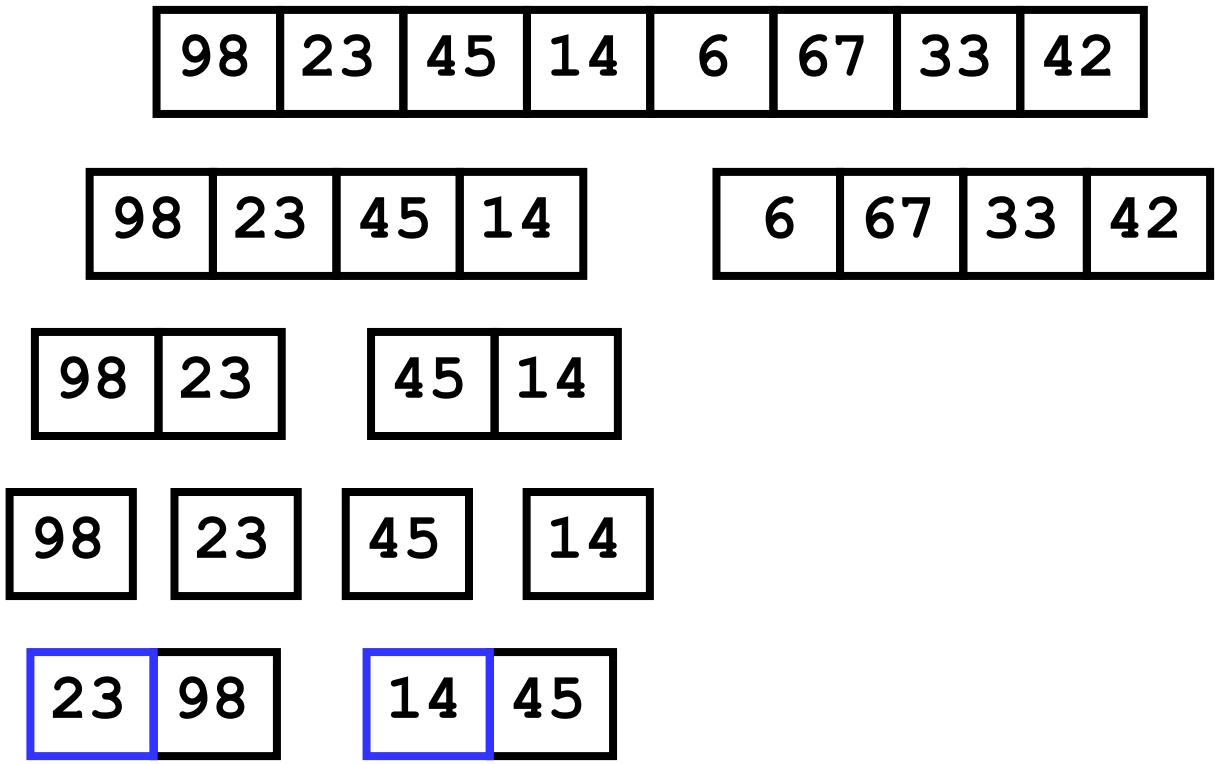

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
23 98
Merge

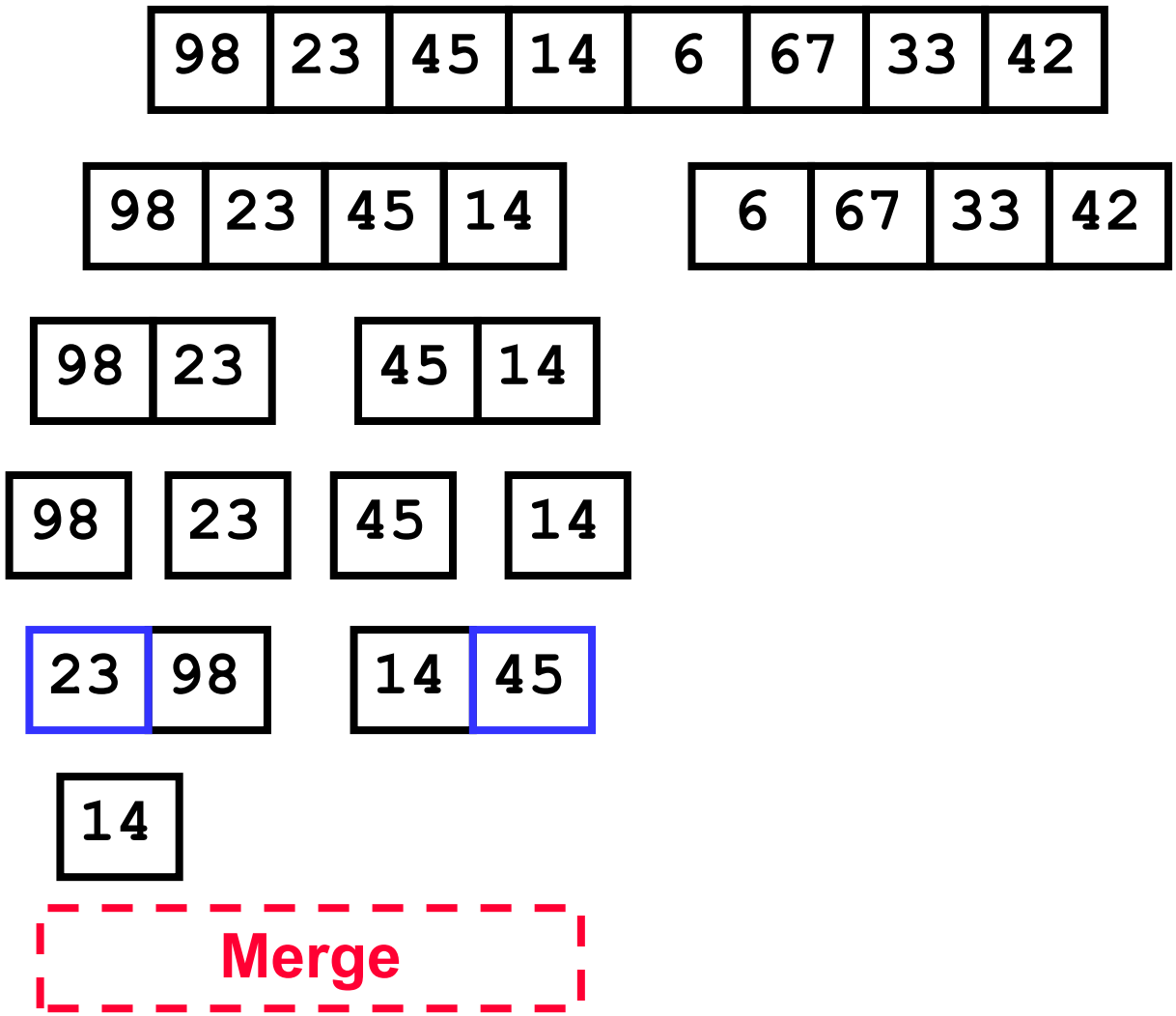

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98

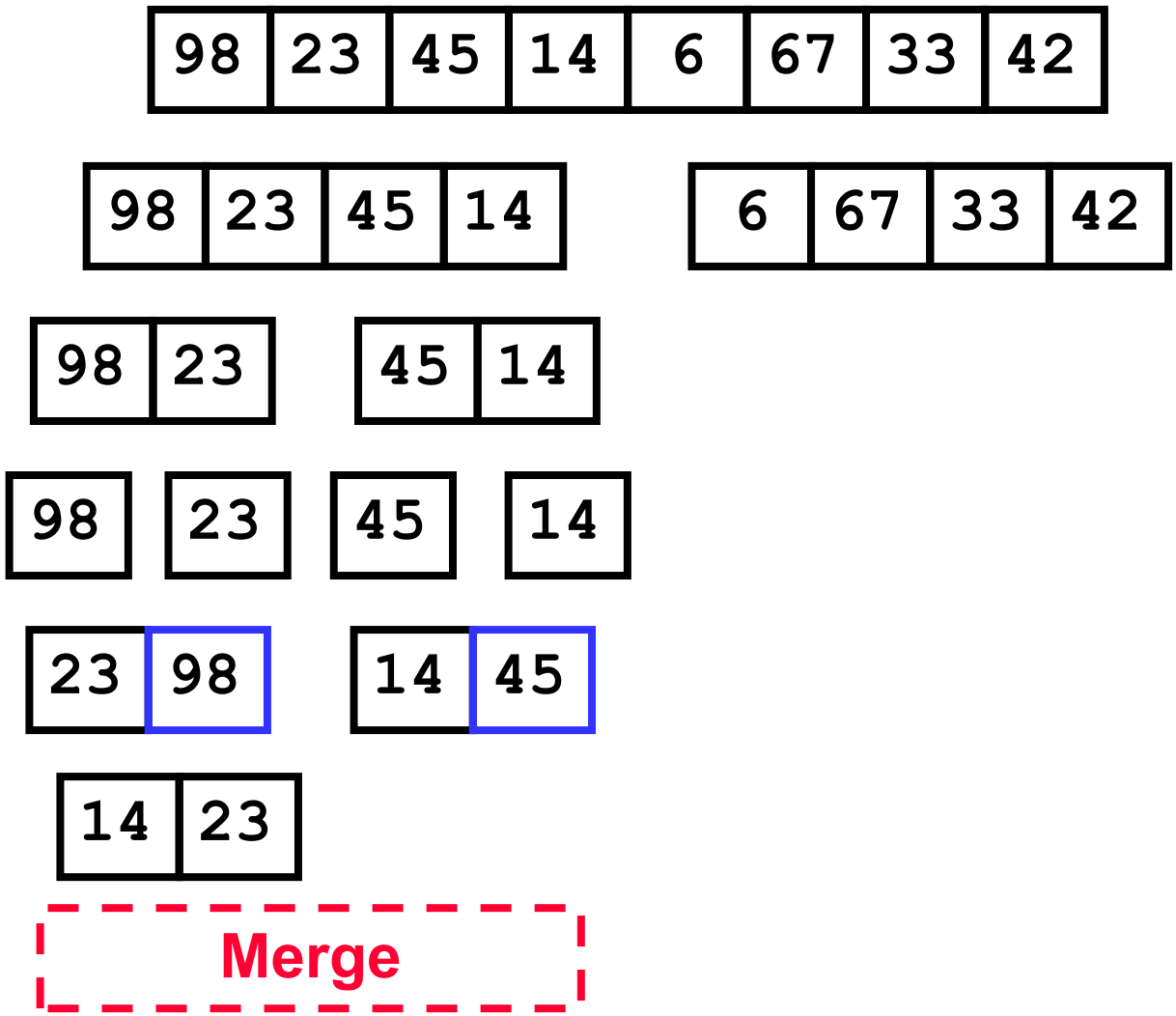

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
Merge

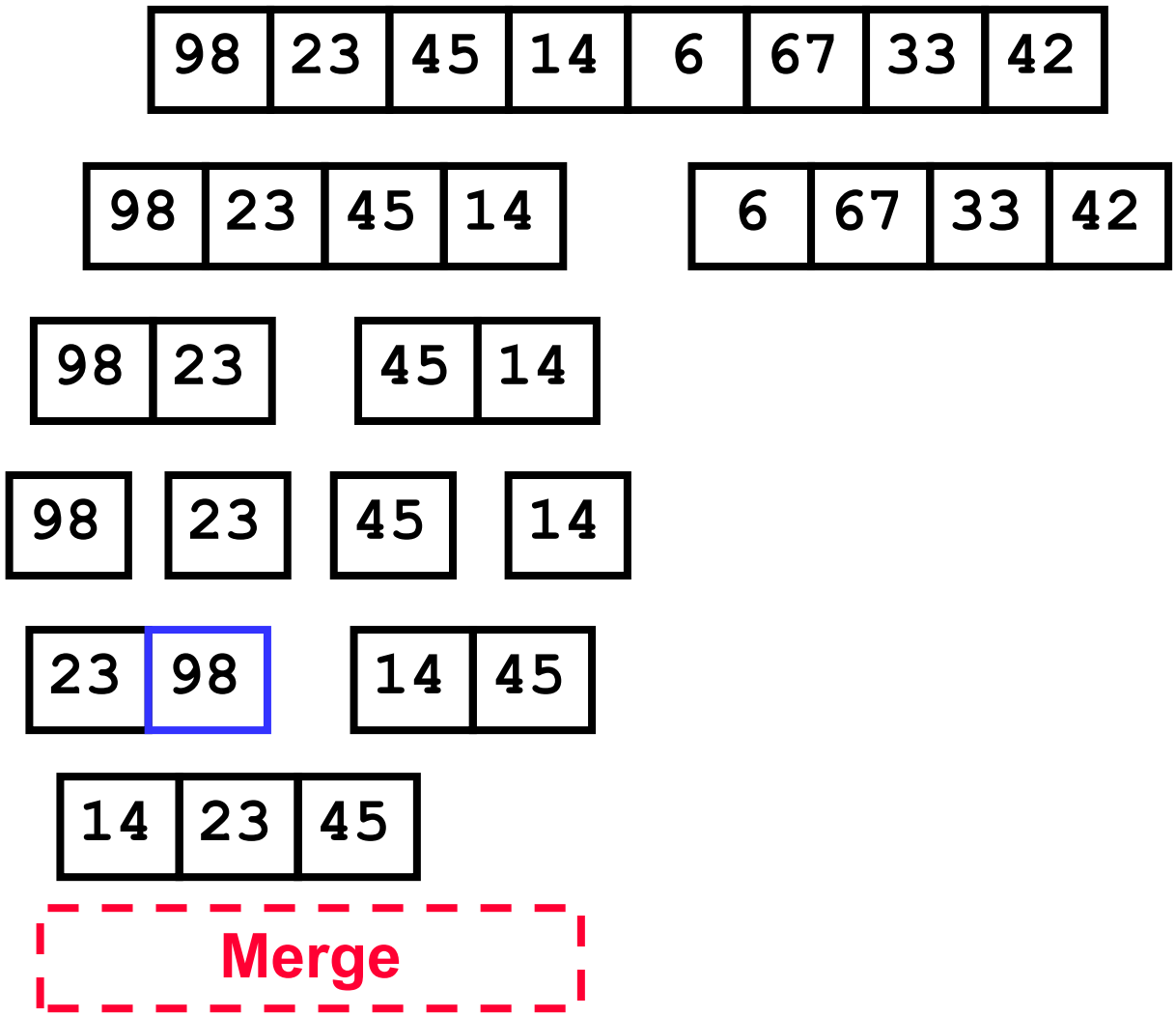

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14
Merge

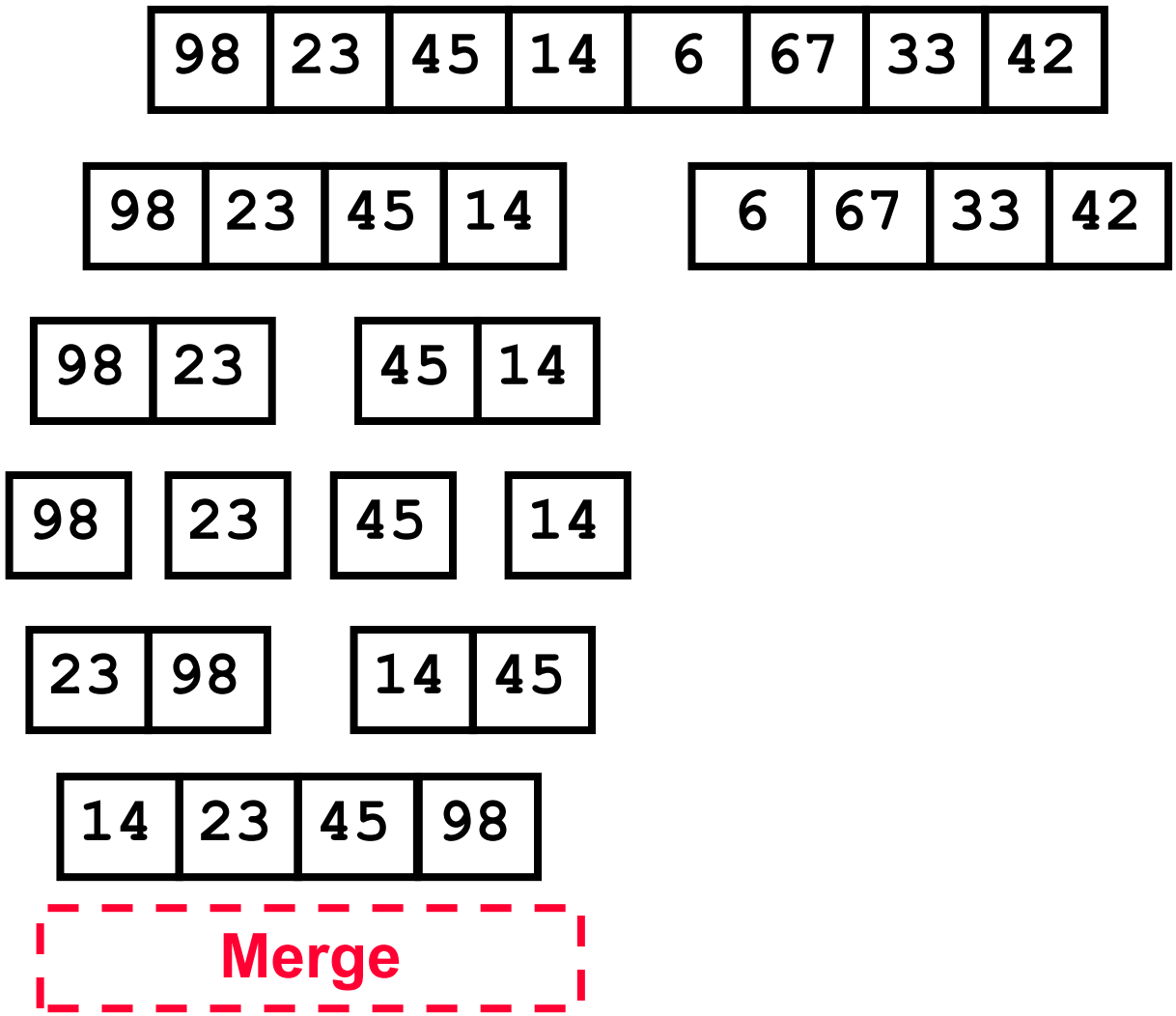

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
Merge

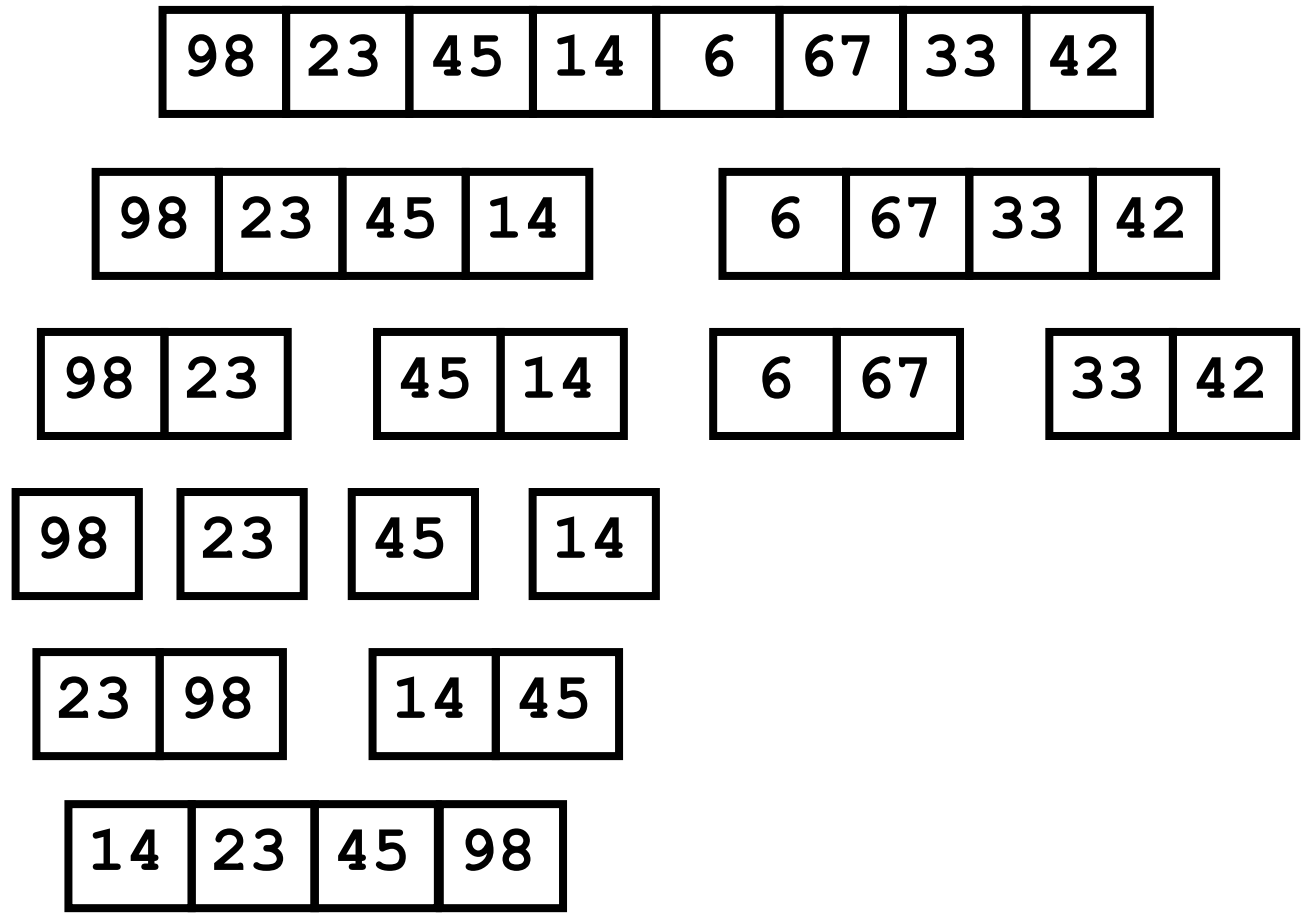

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
Merge

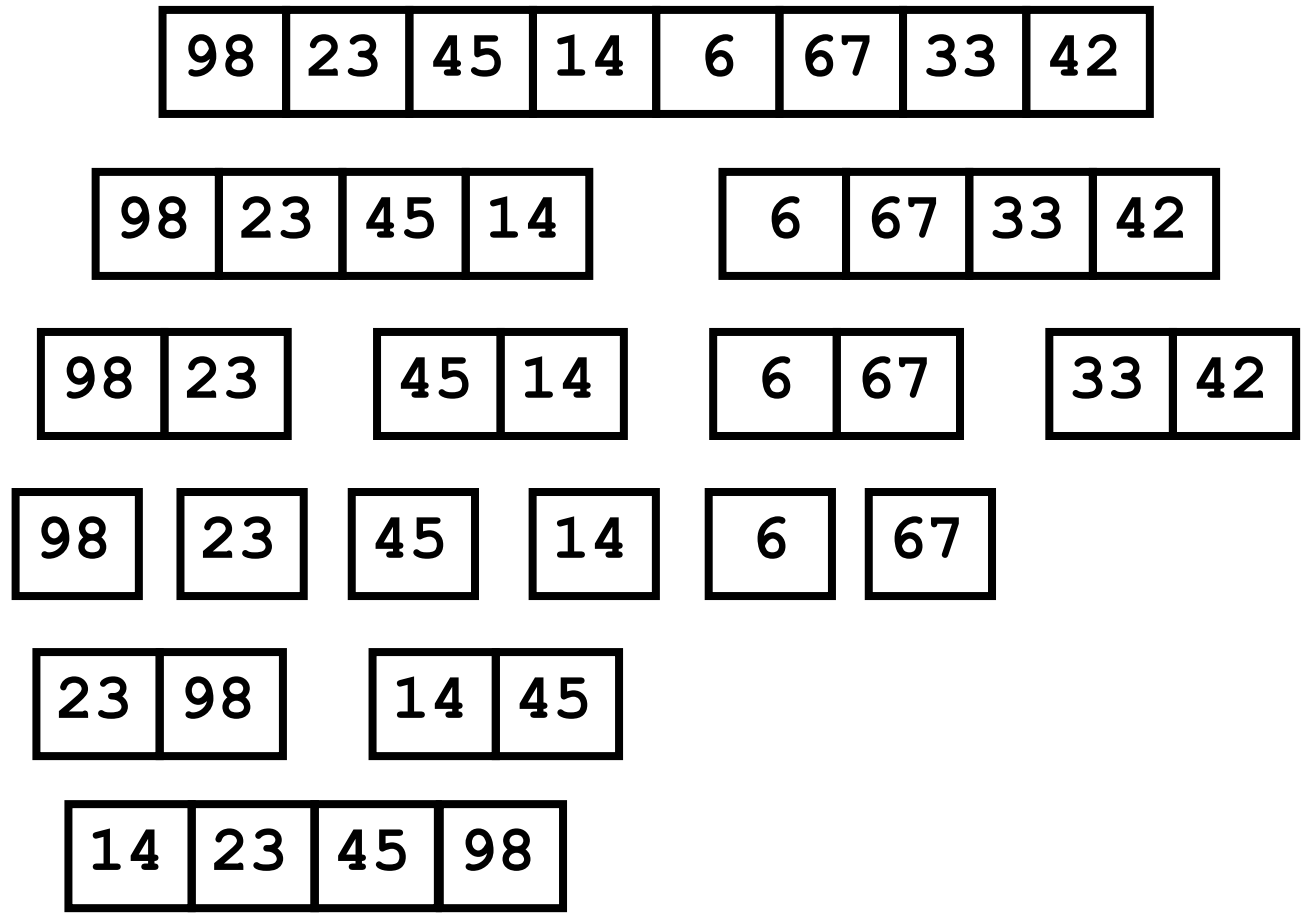

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14
Merge

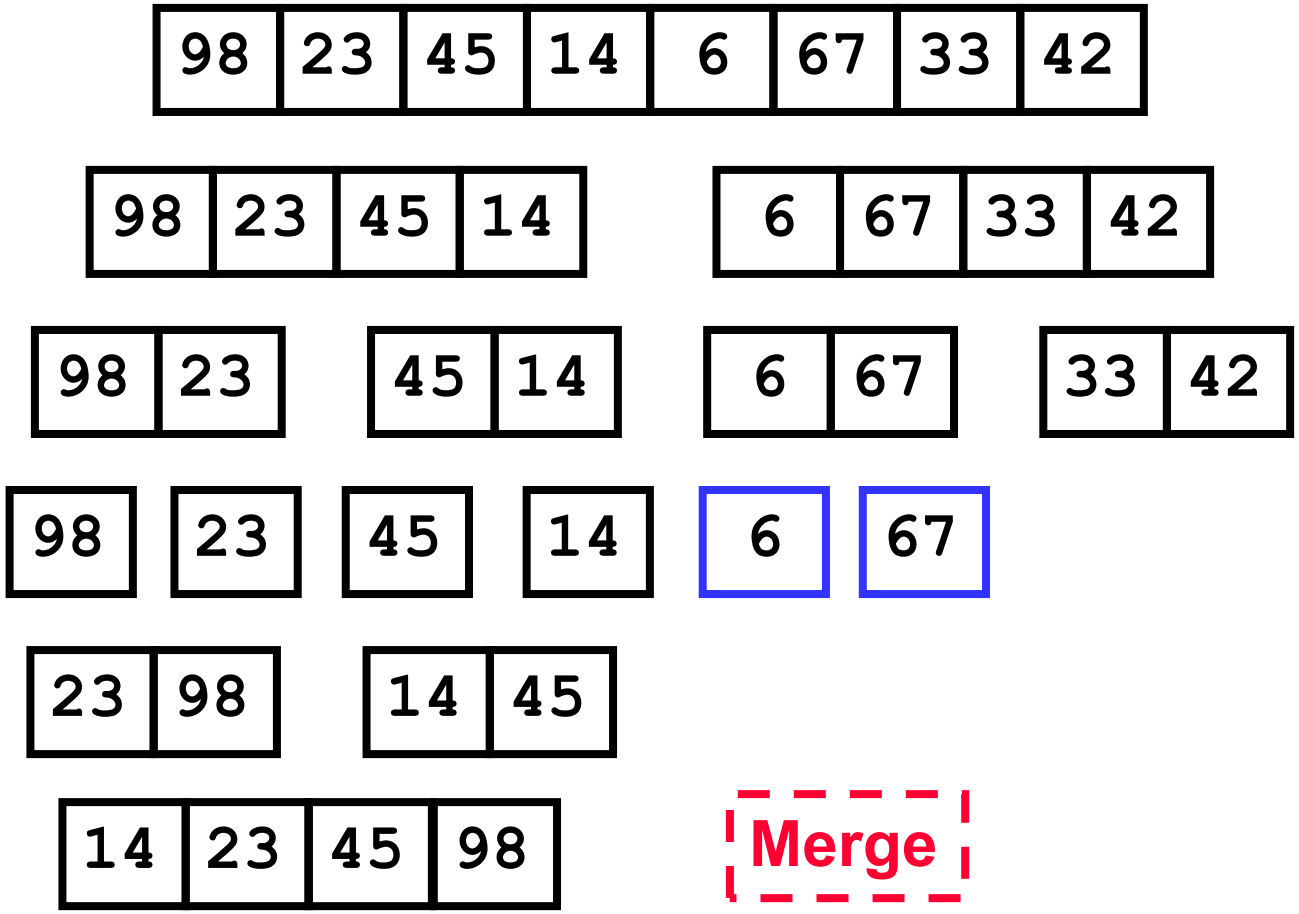

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23
Merge

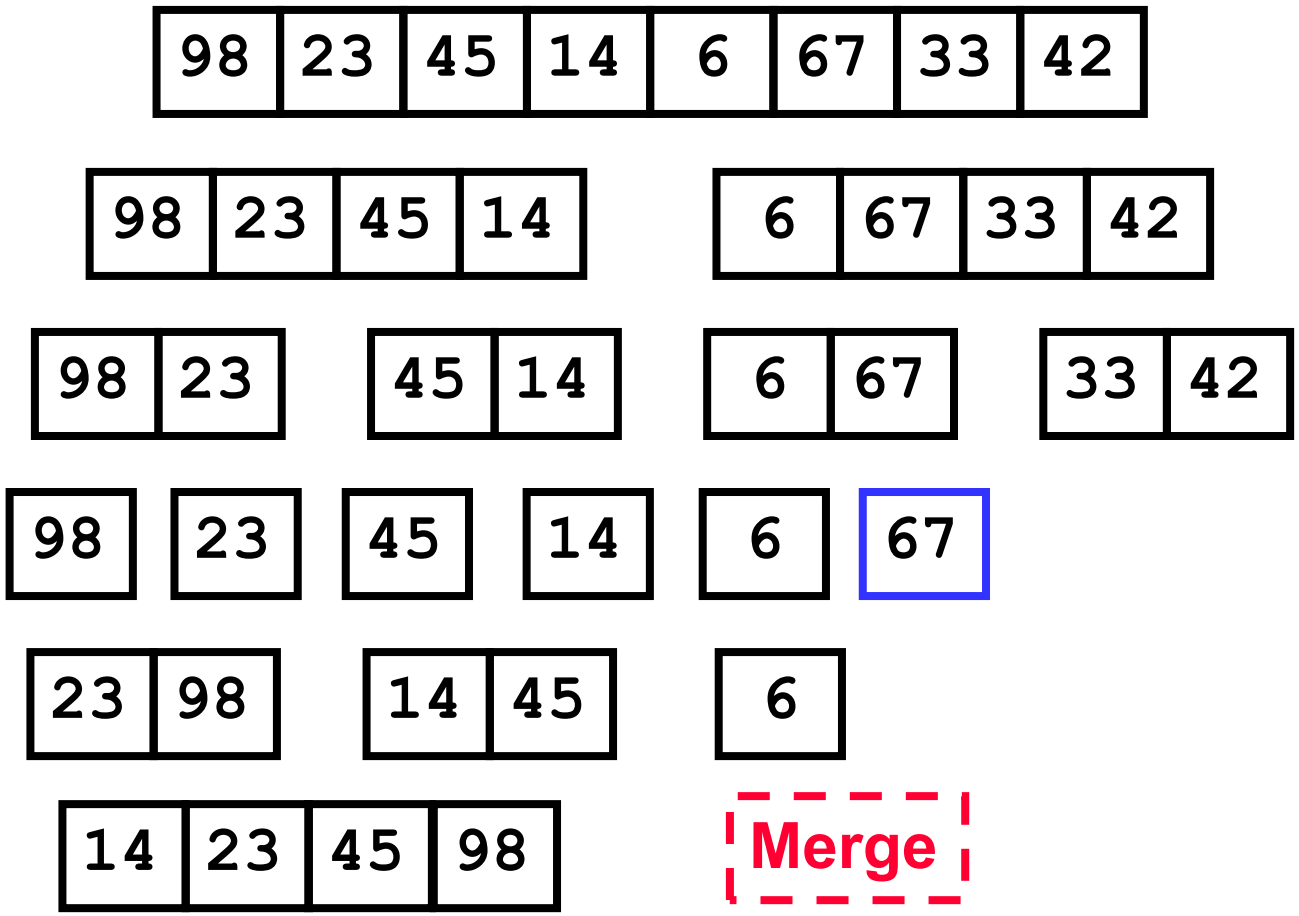

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23 45
Merge

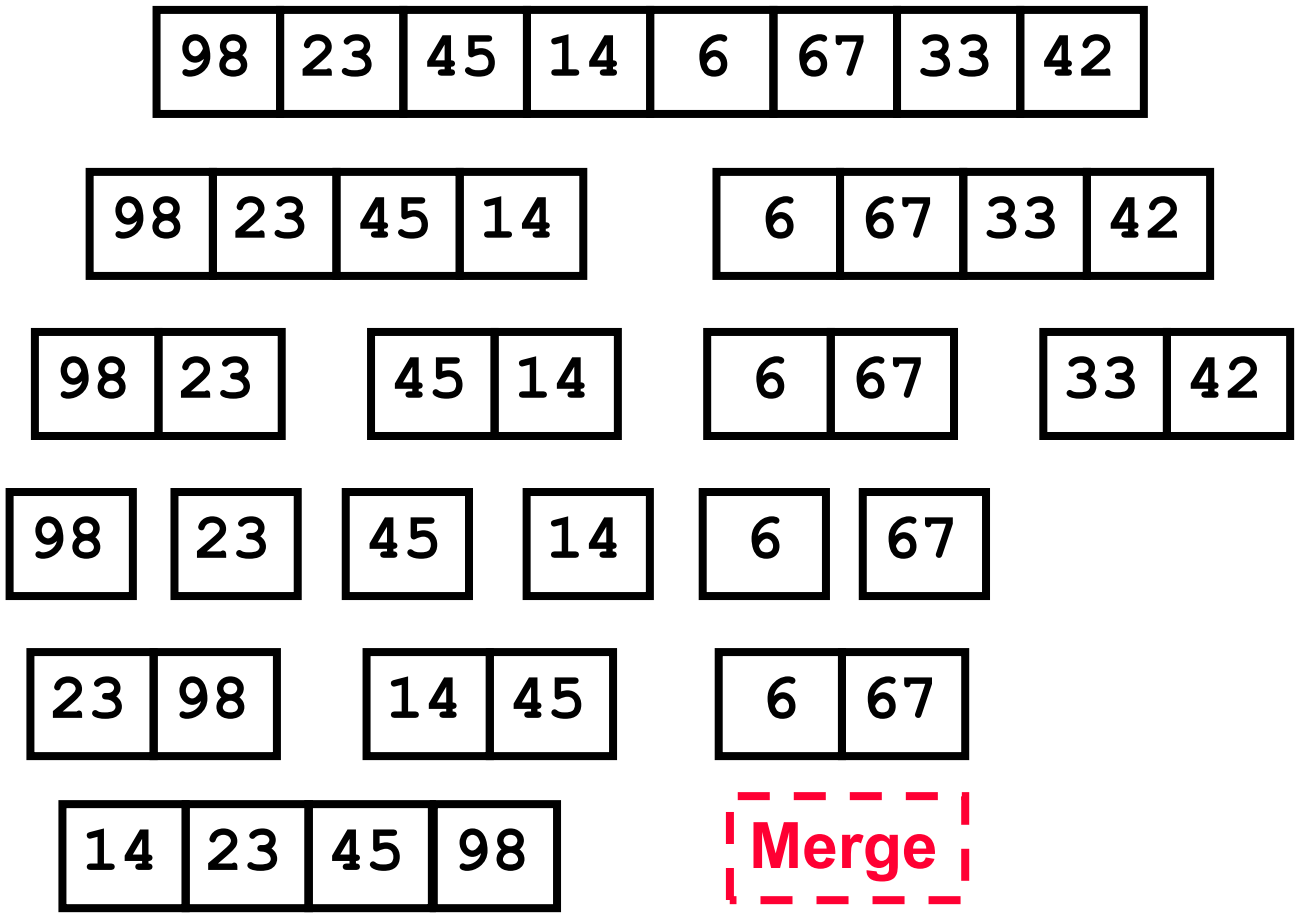

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23 45 98
Merge

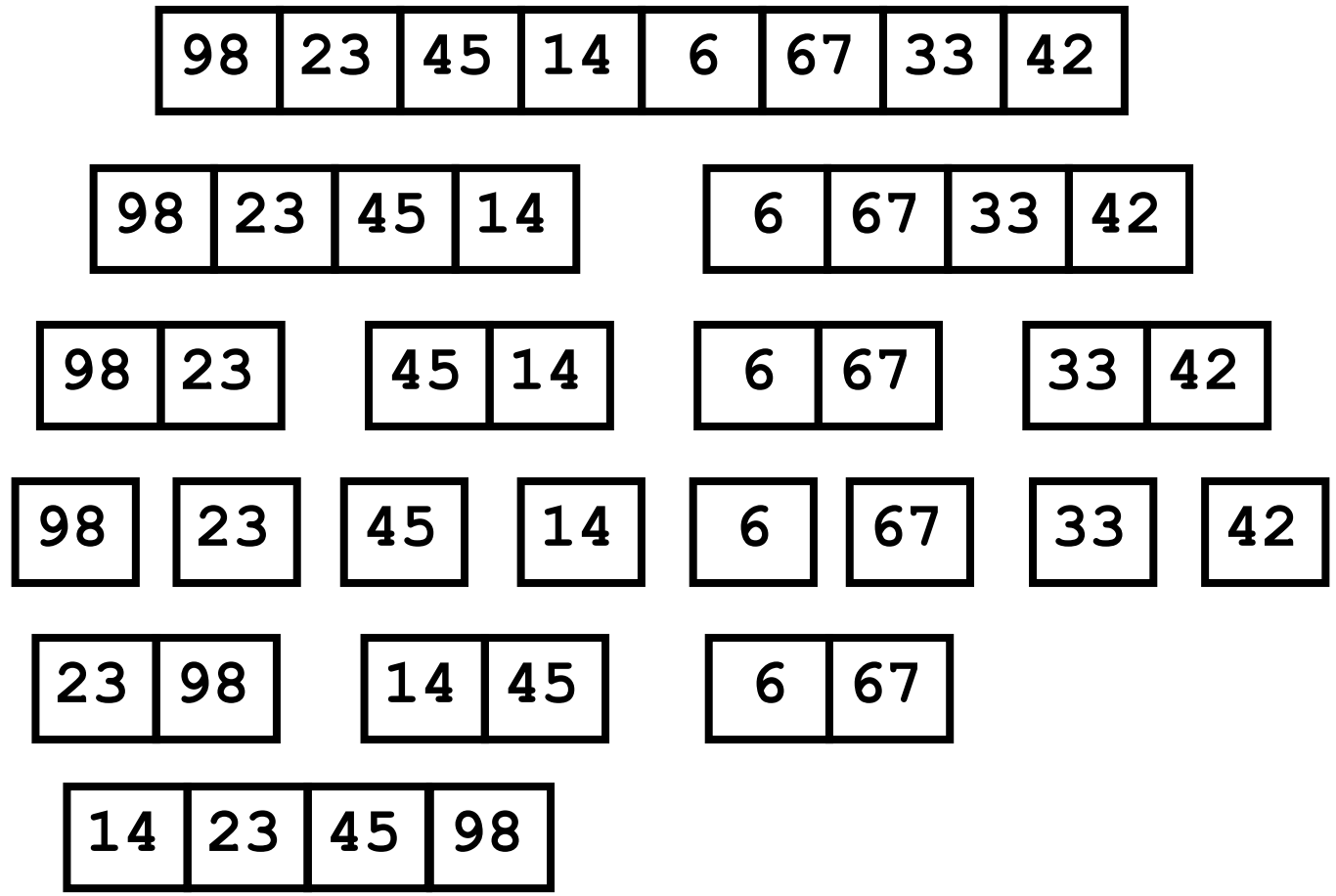

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
23 98
14 45
14 23 45 98

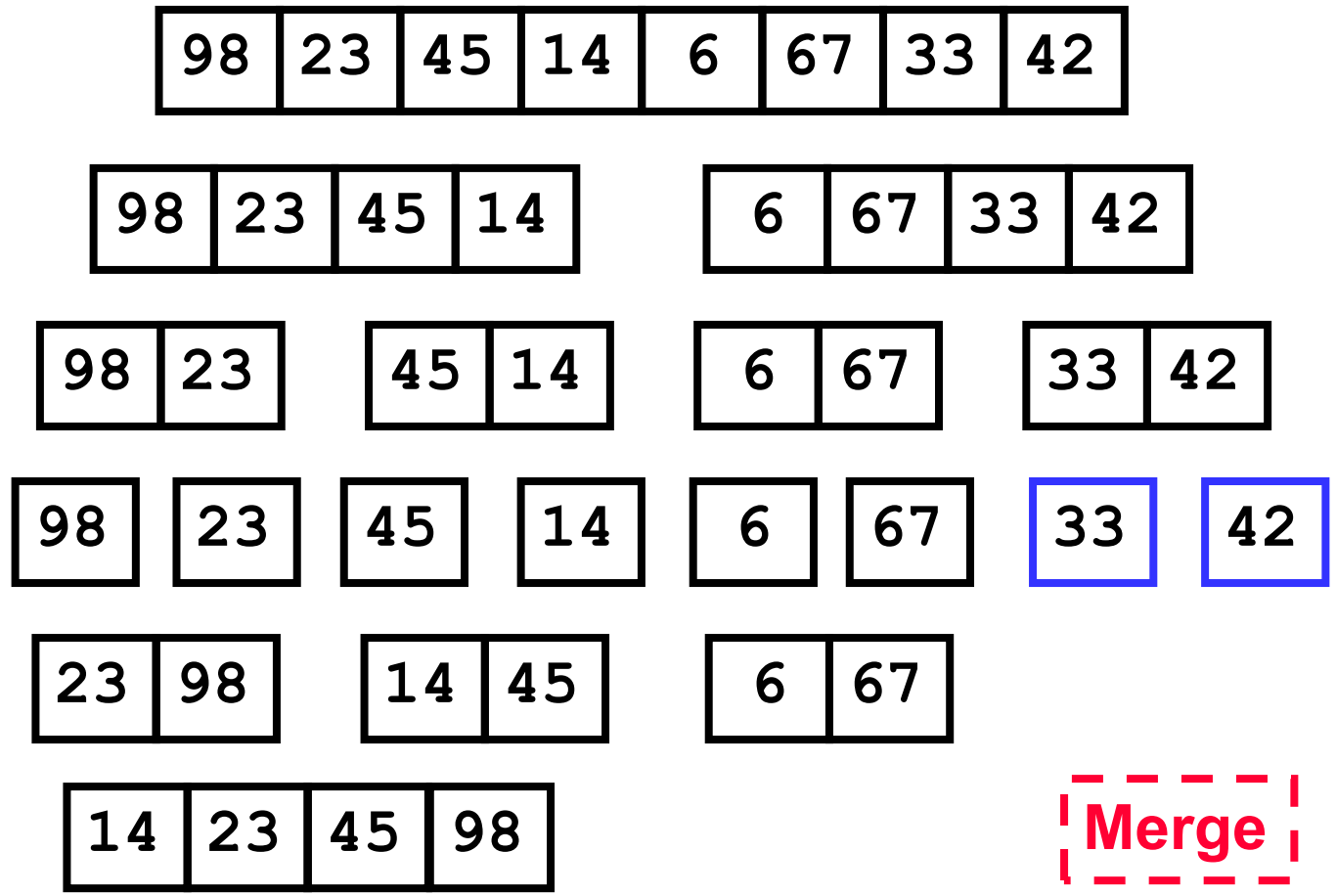

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
14 23 45 98

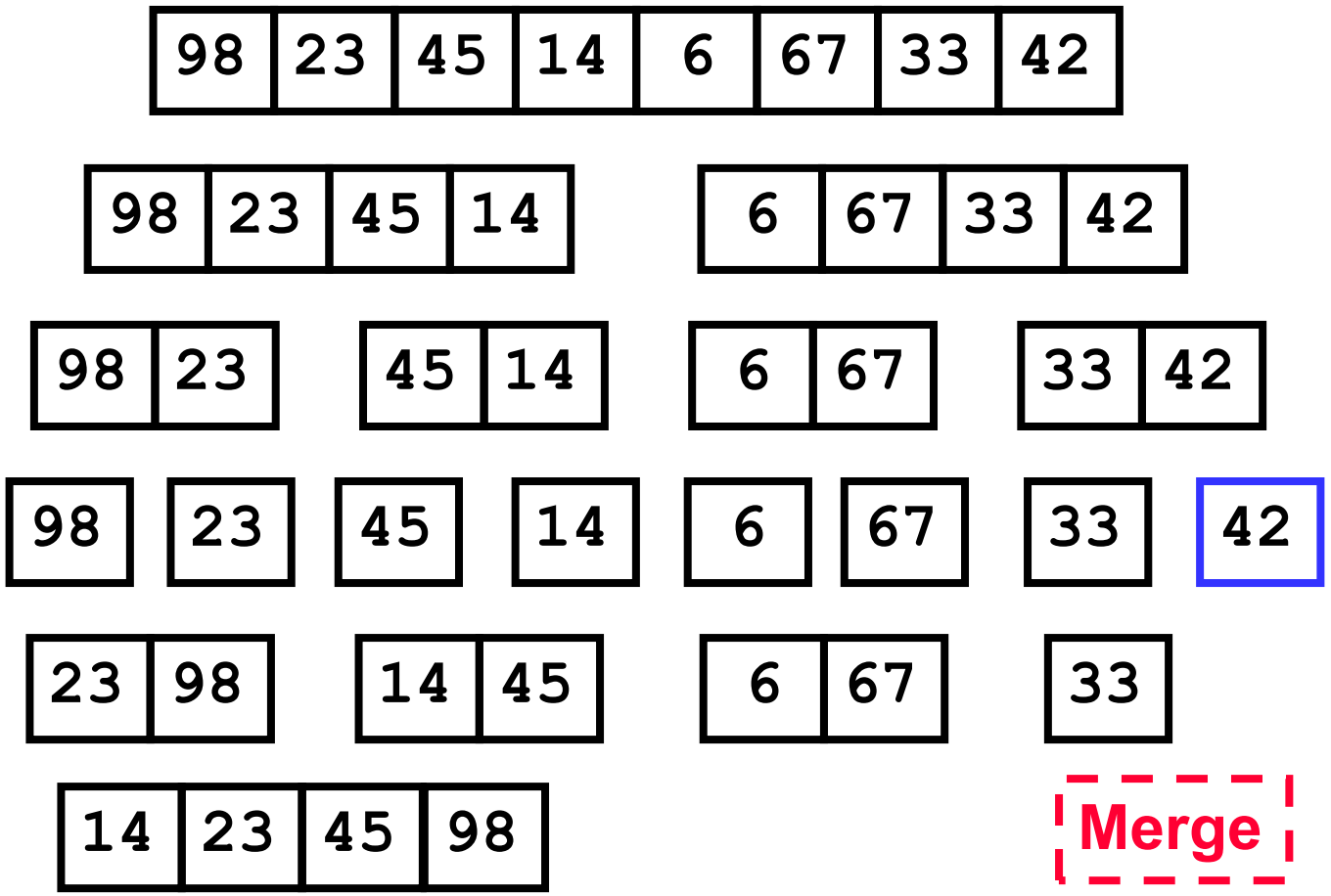

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
14 23 45 98
Merge

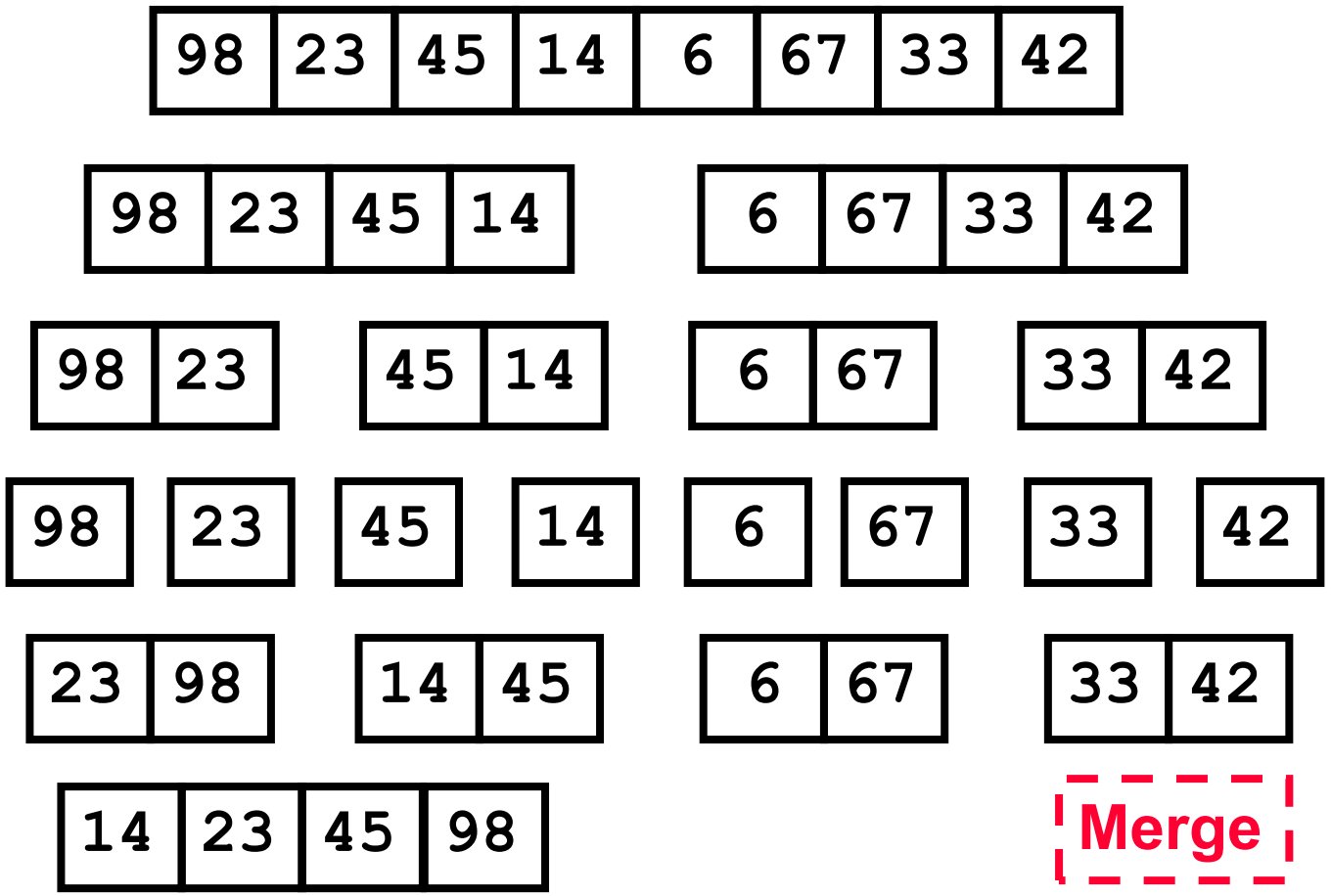

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
6
14 23 45 98
Merge

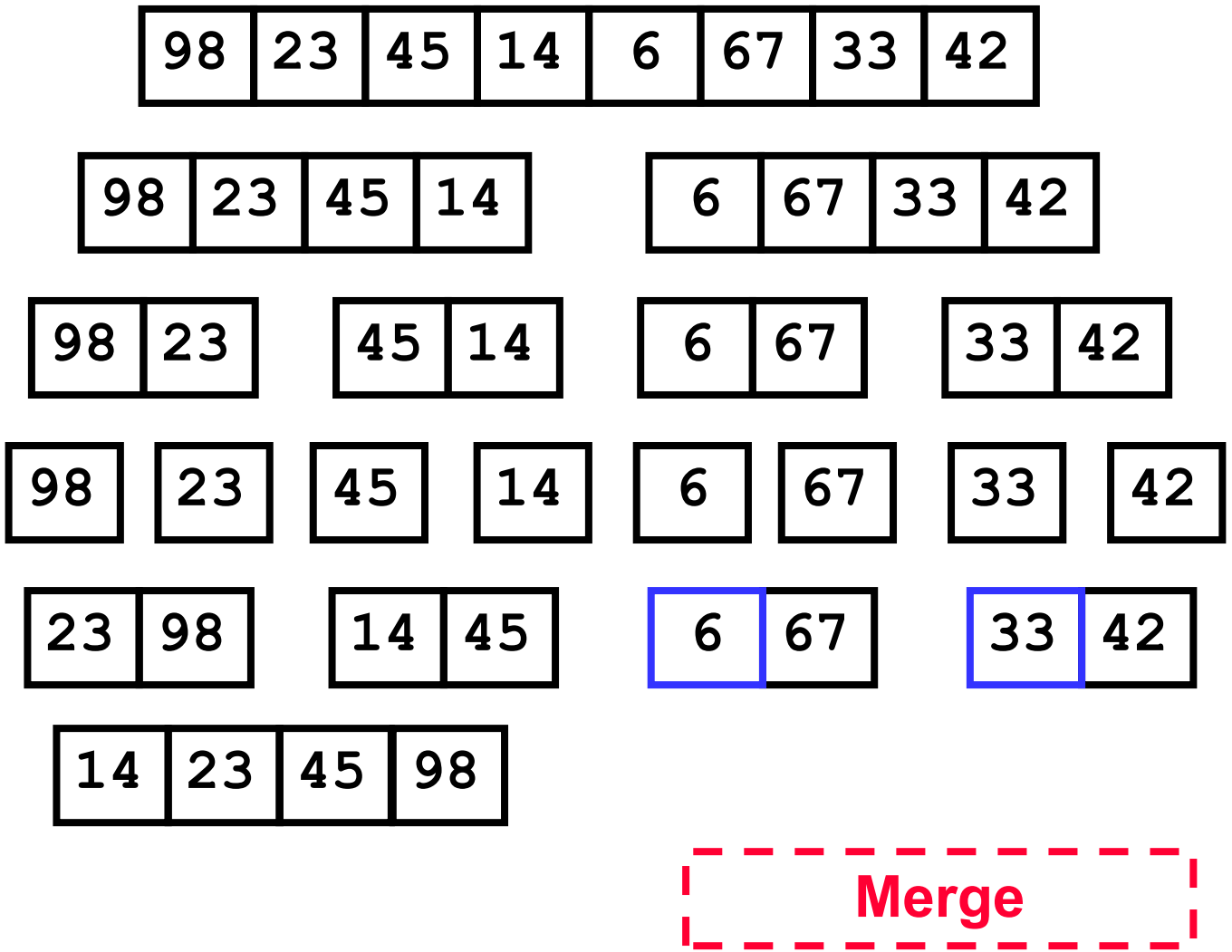

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
6 67
14 23 45 98
Merge

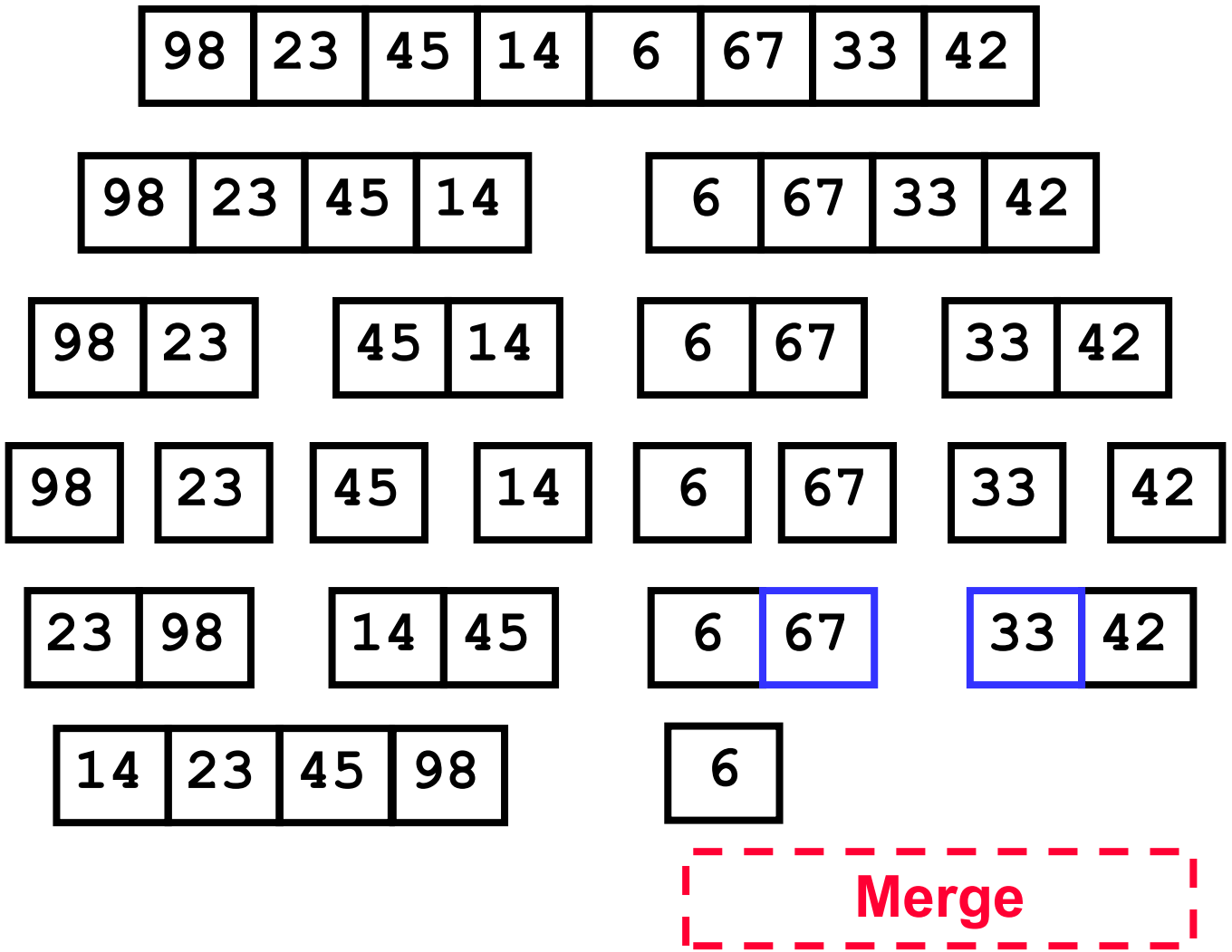

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
14 23 45 98

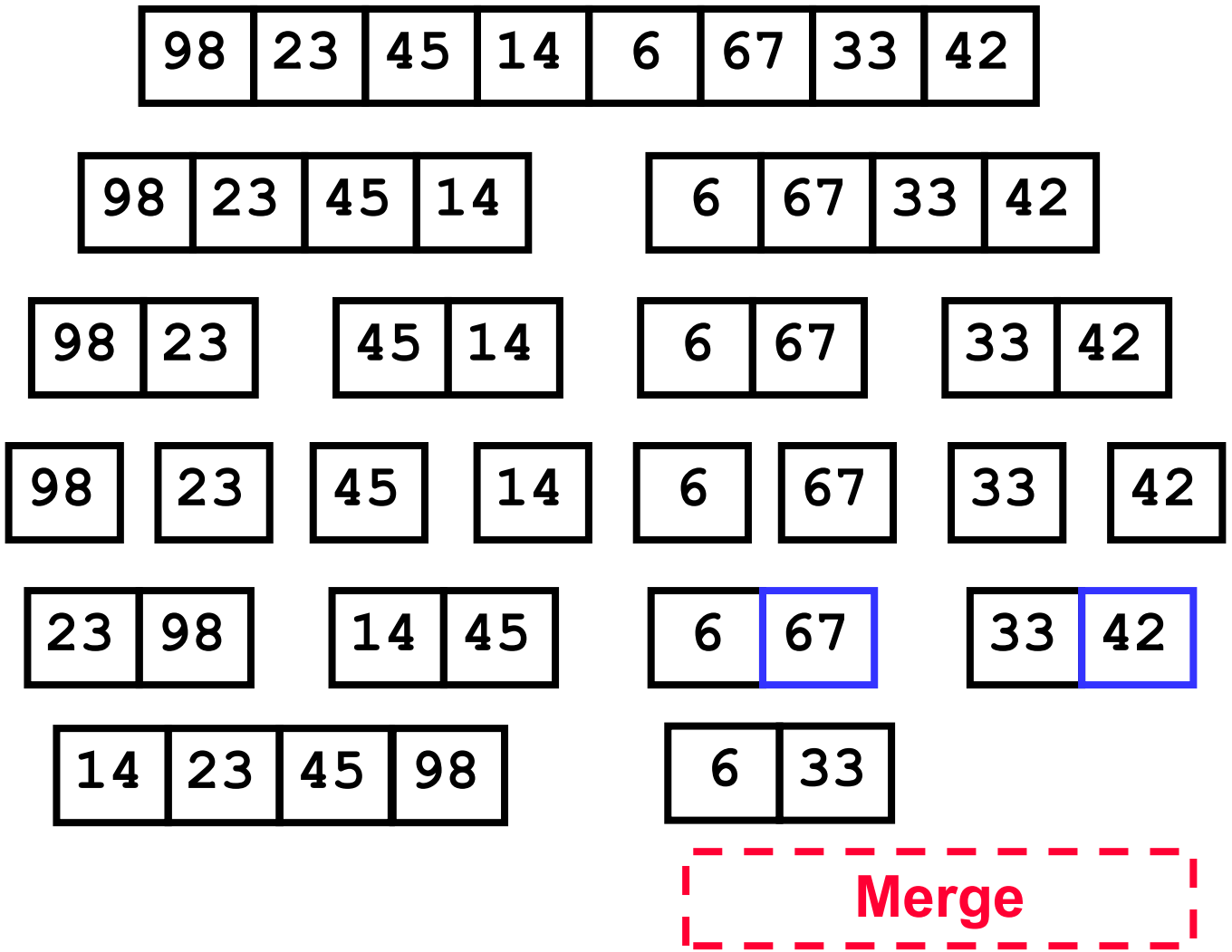

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
14 23 45 98
Merge

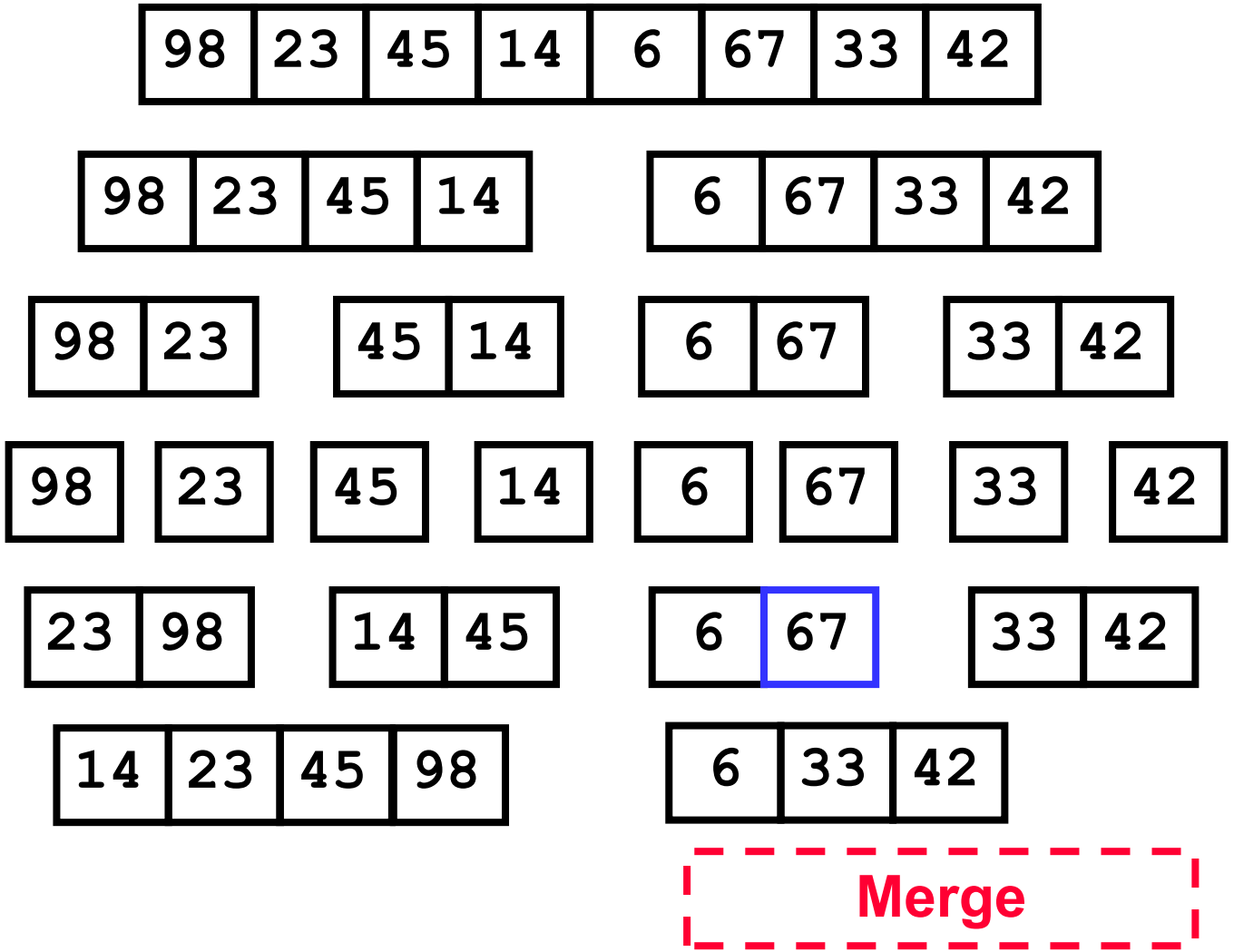

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33
14 23 45 98
Merge

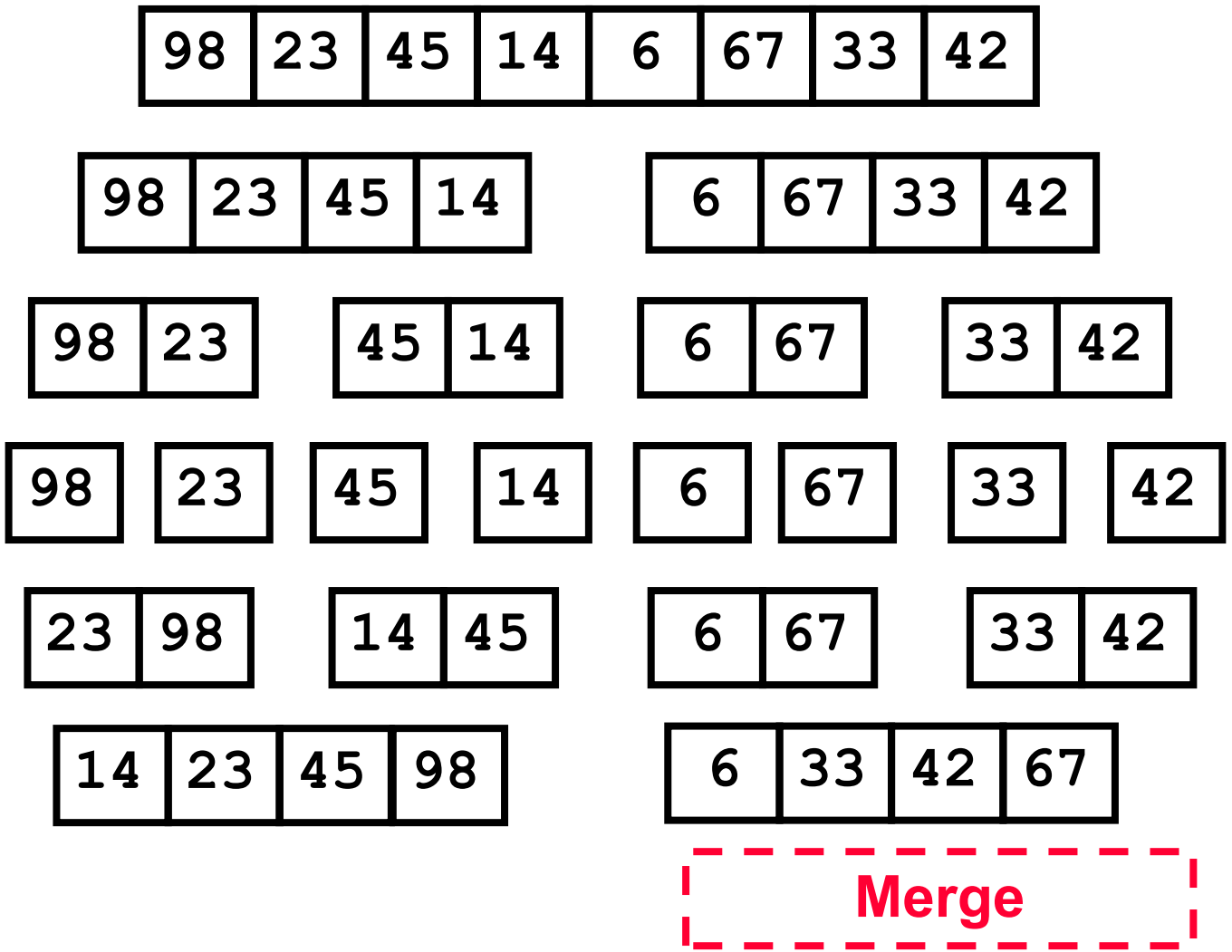

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
Merge

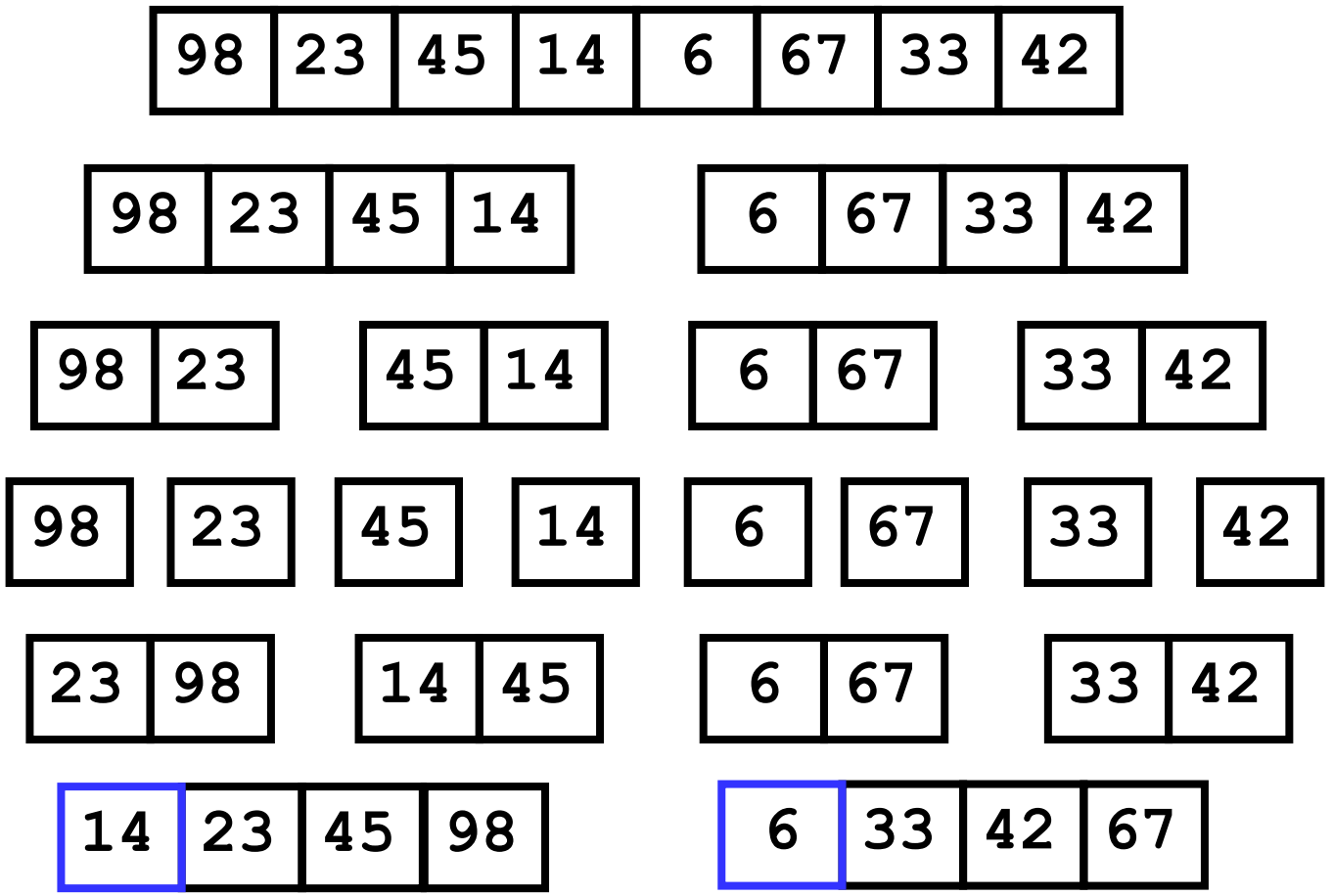

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
Merge

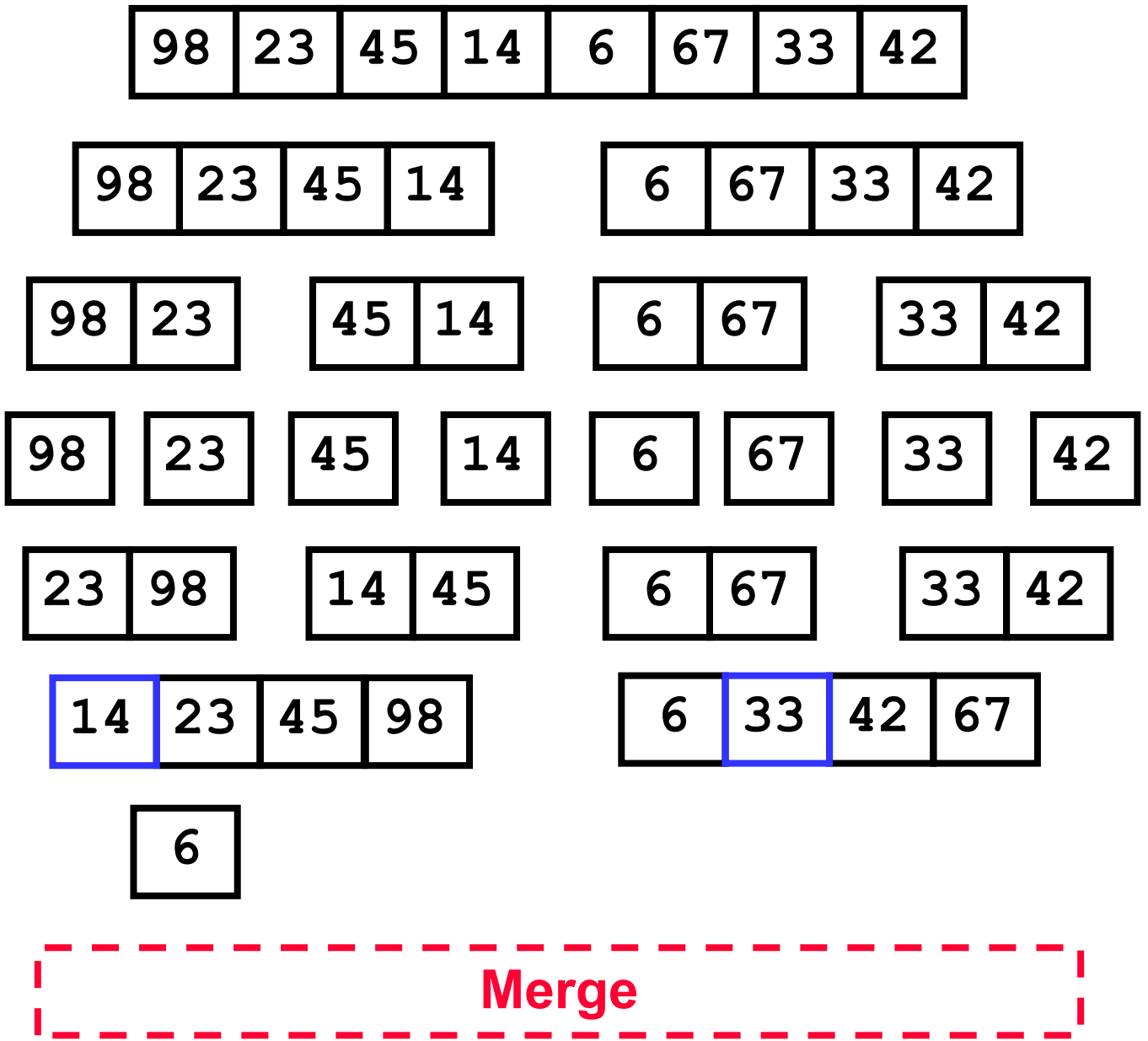

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6
Merge

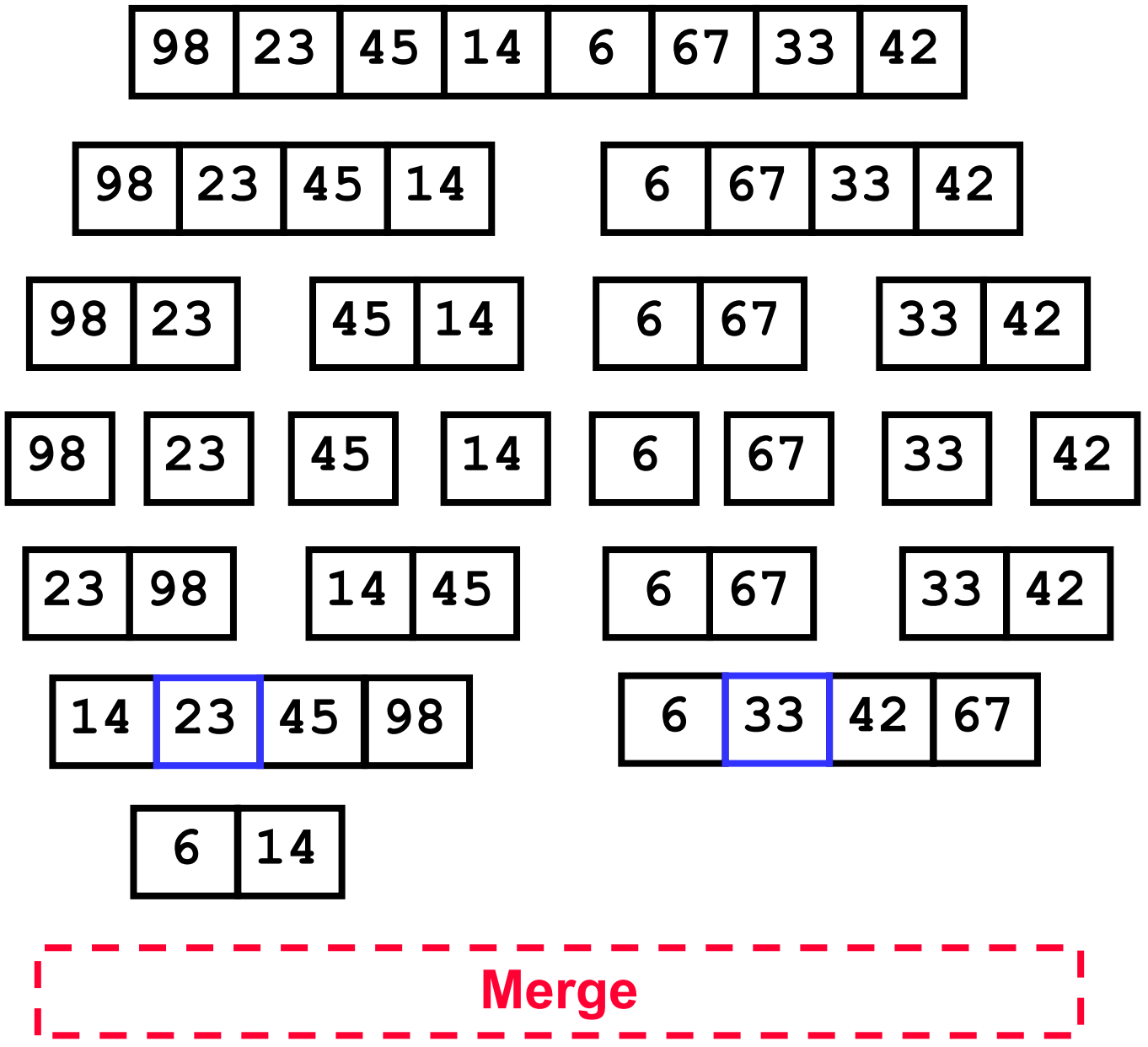

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33
Merge

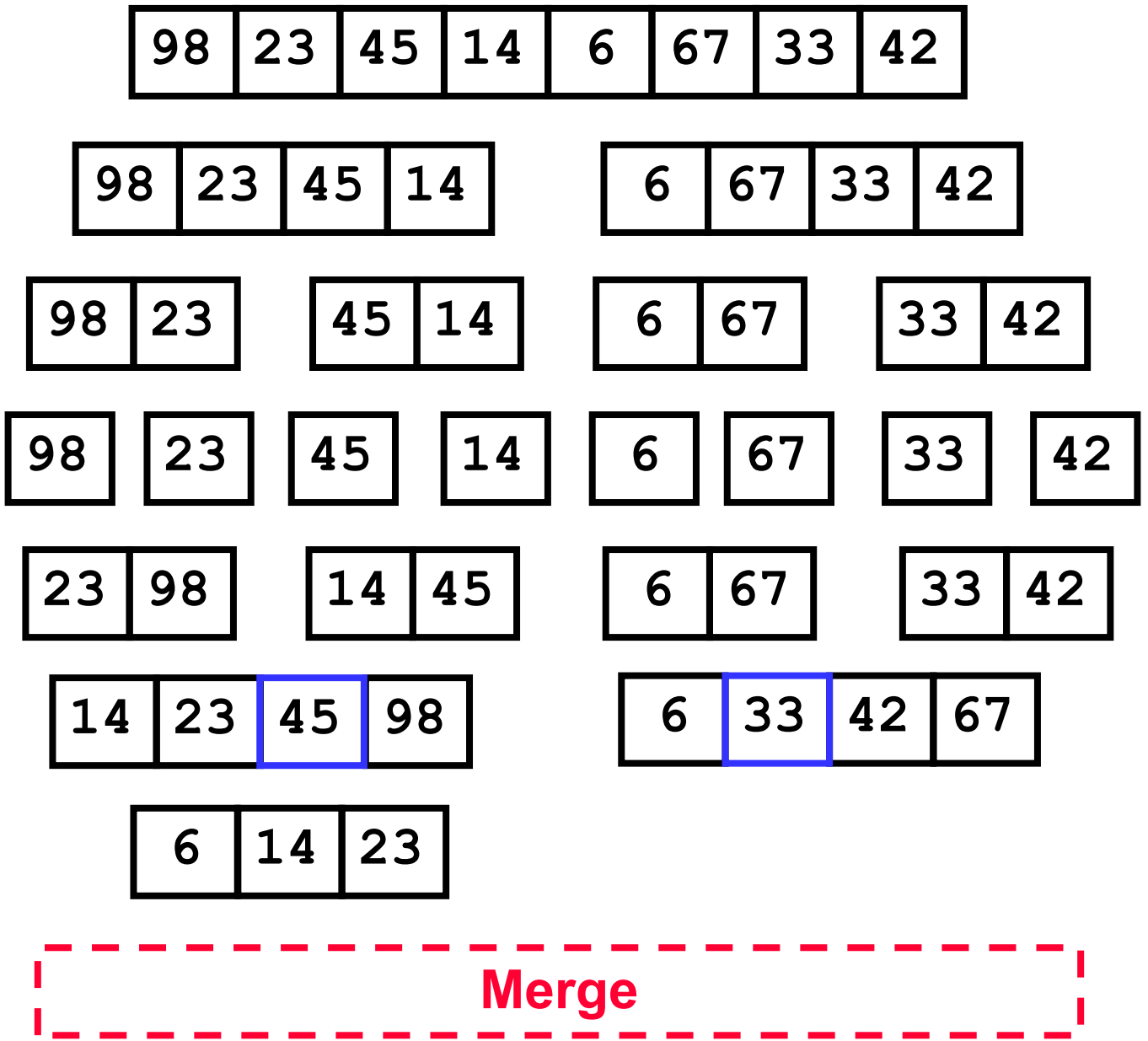

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42
Merge

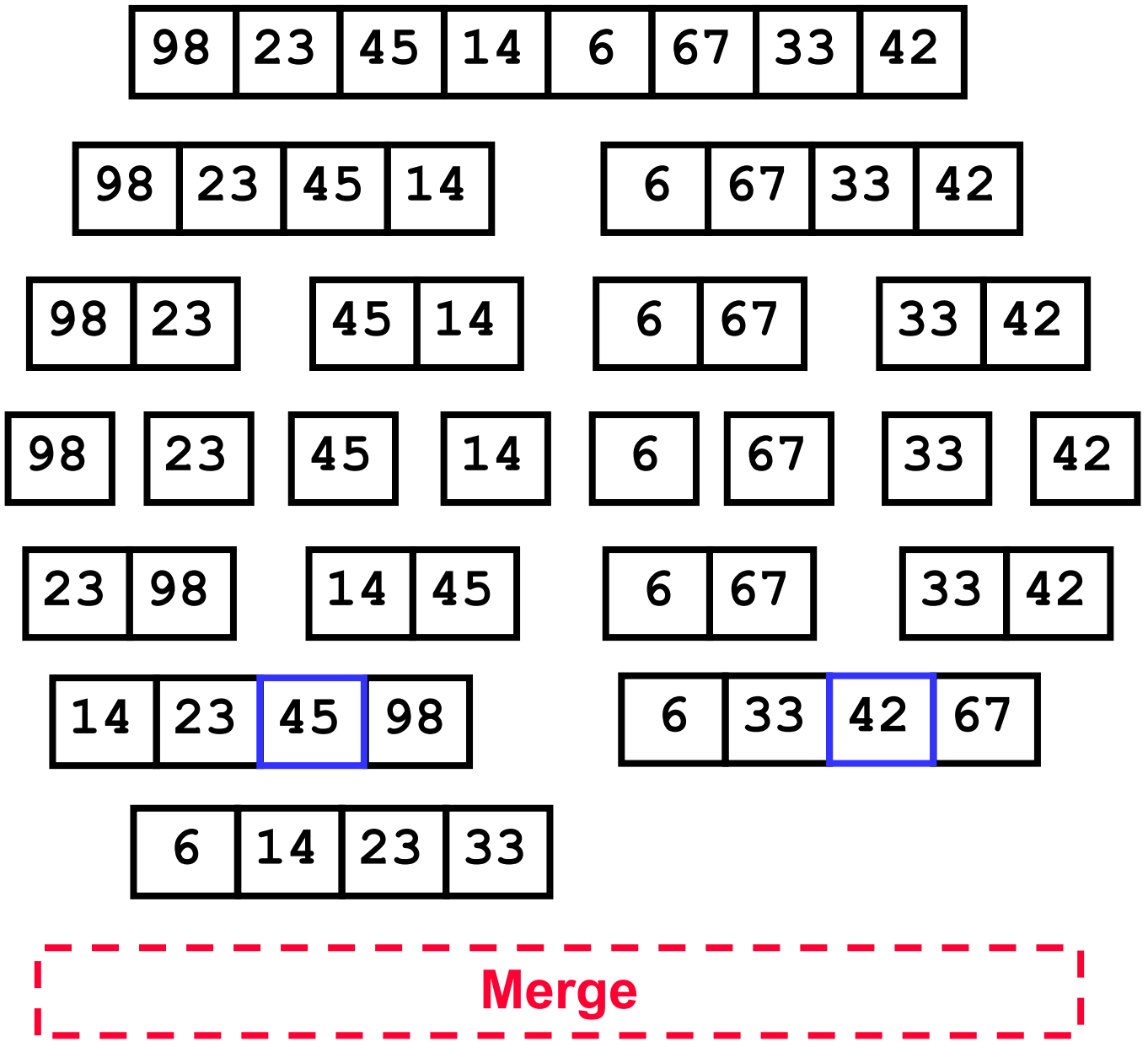

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
Merge

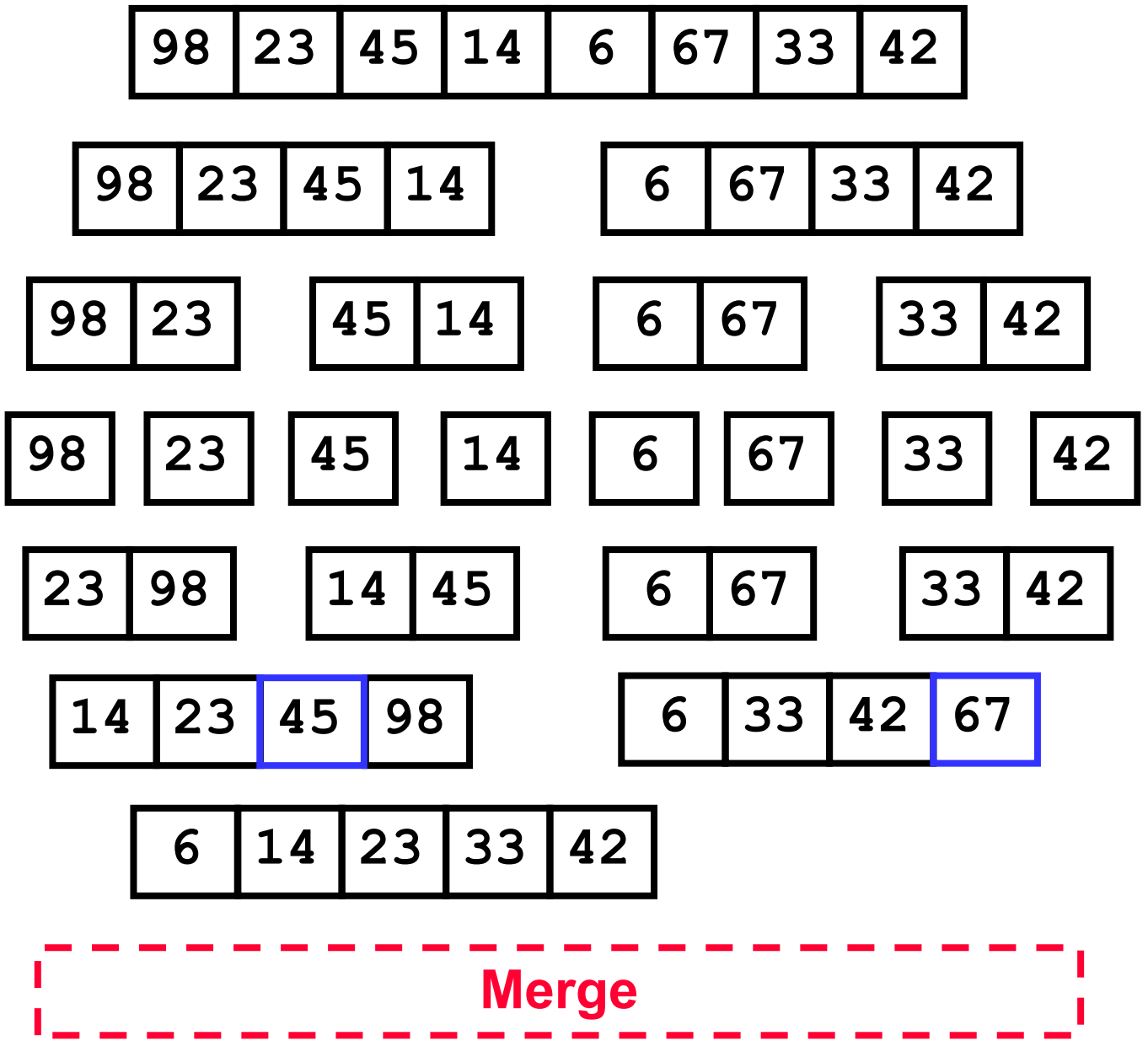

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
Merge

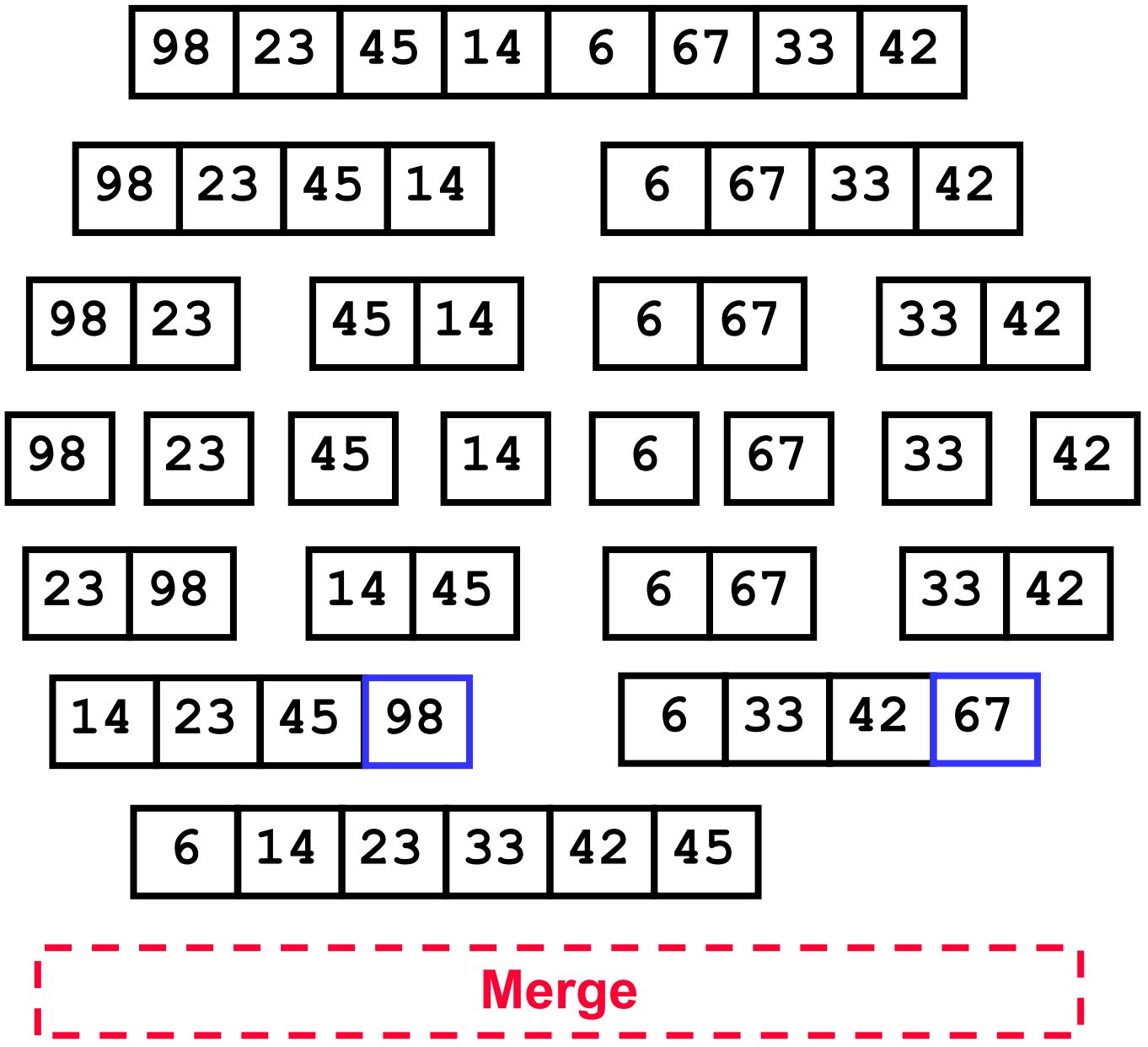

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6
Merge

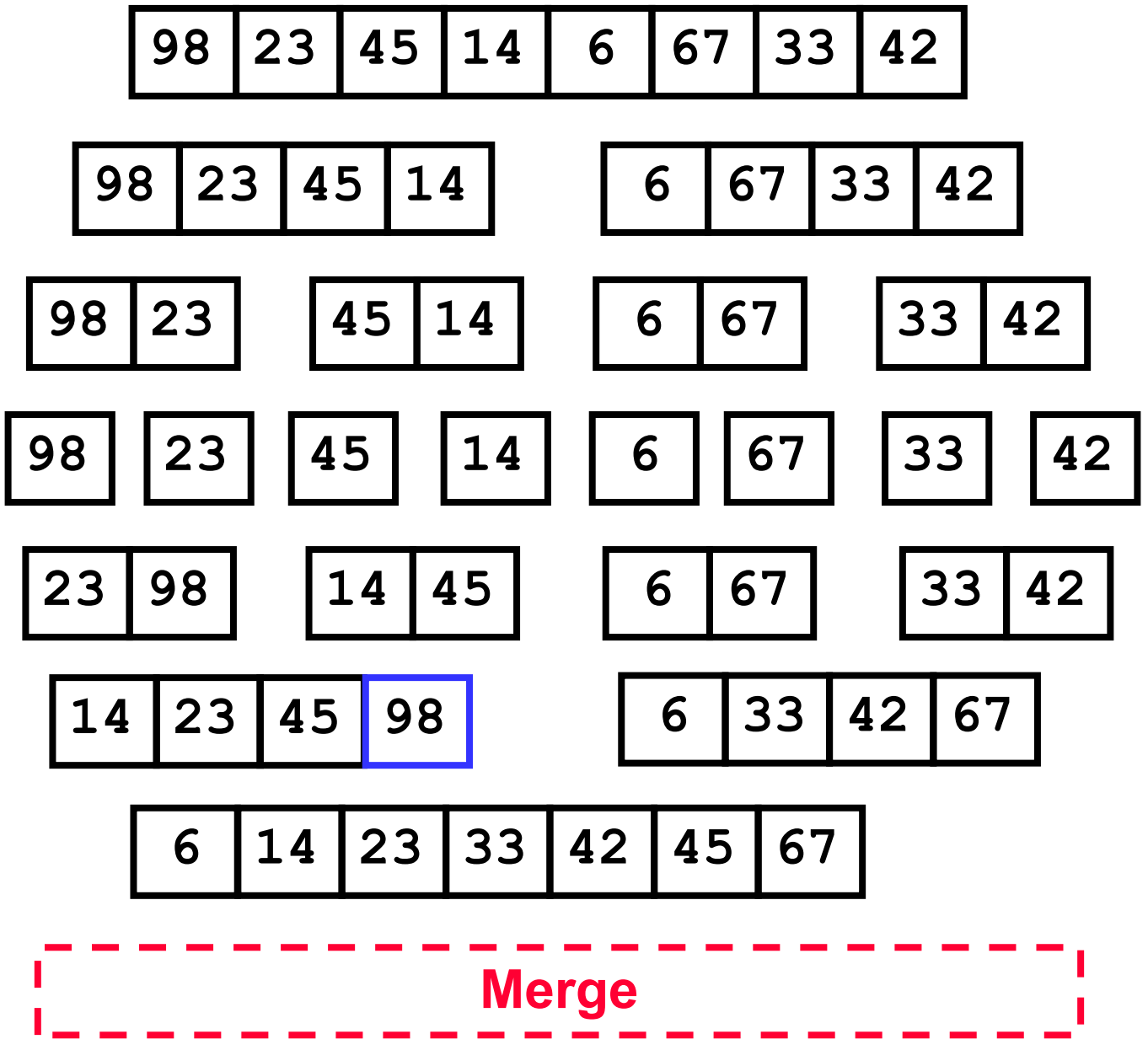

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14
Merge

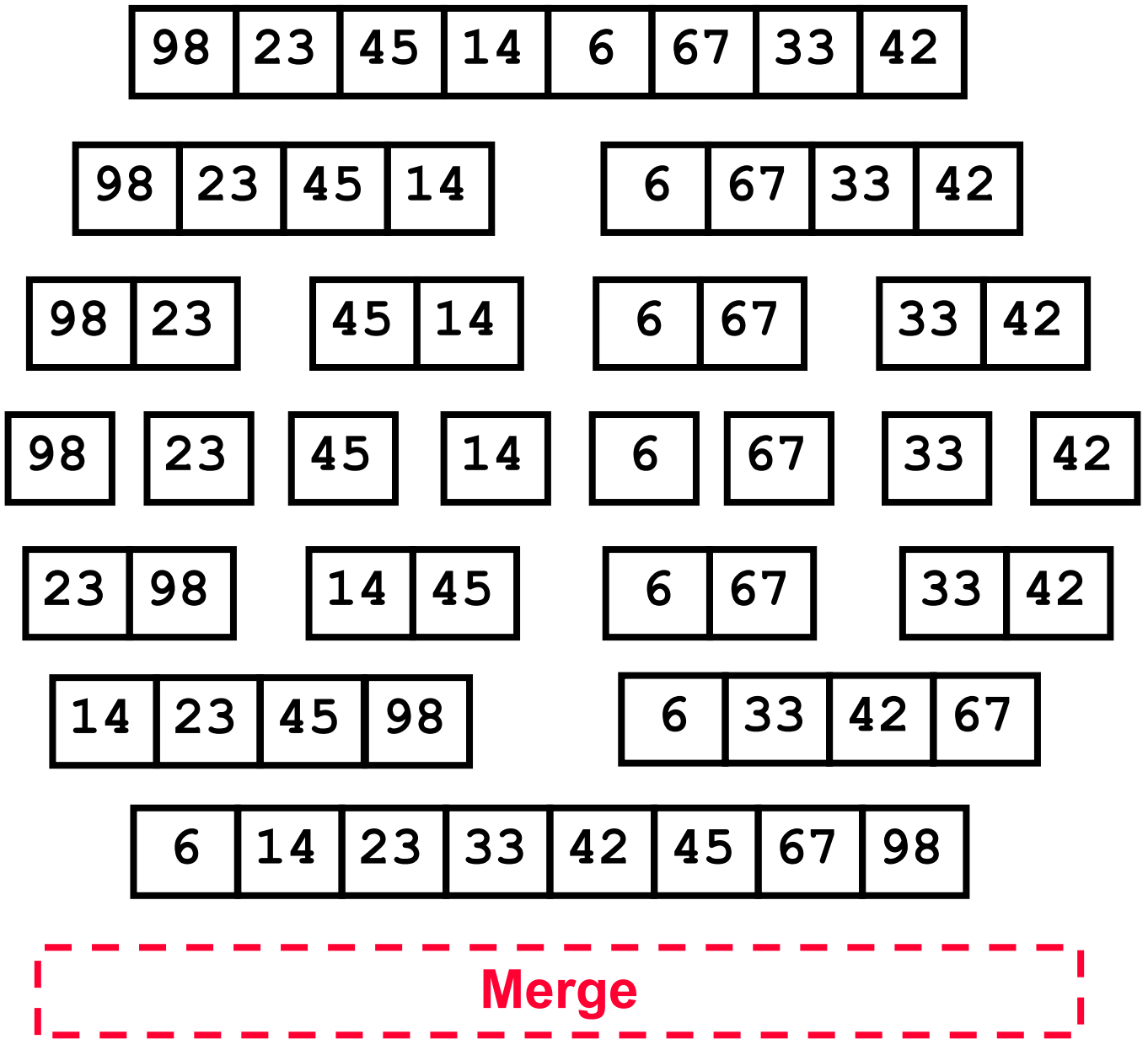

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23
Merge

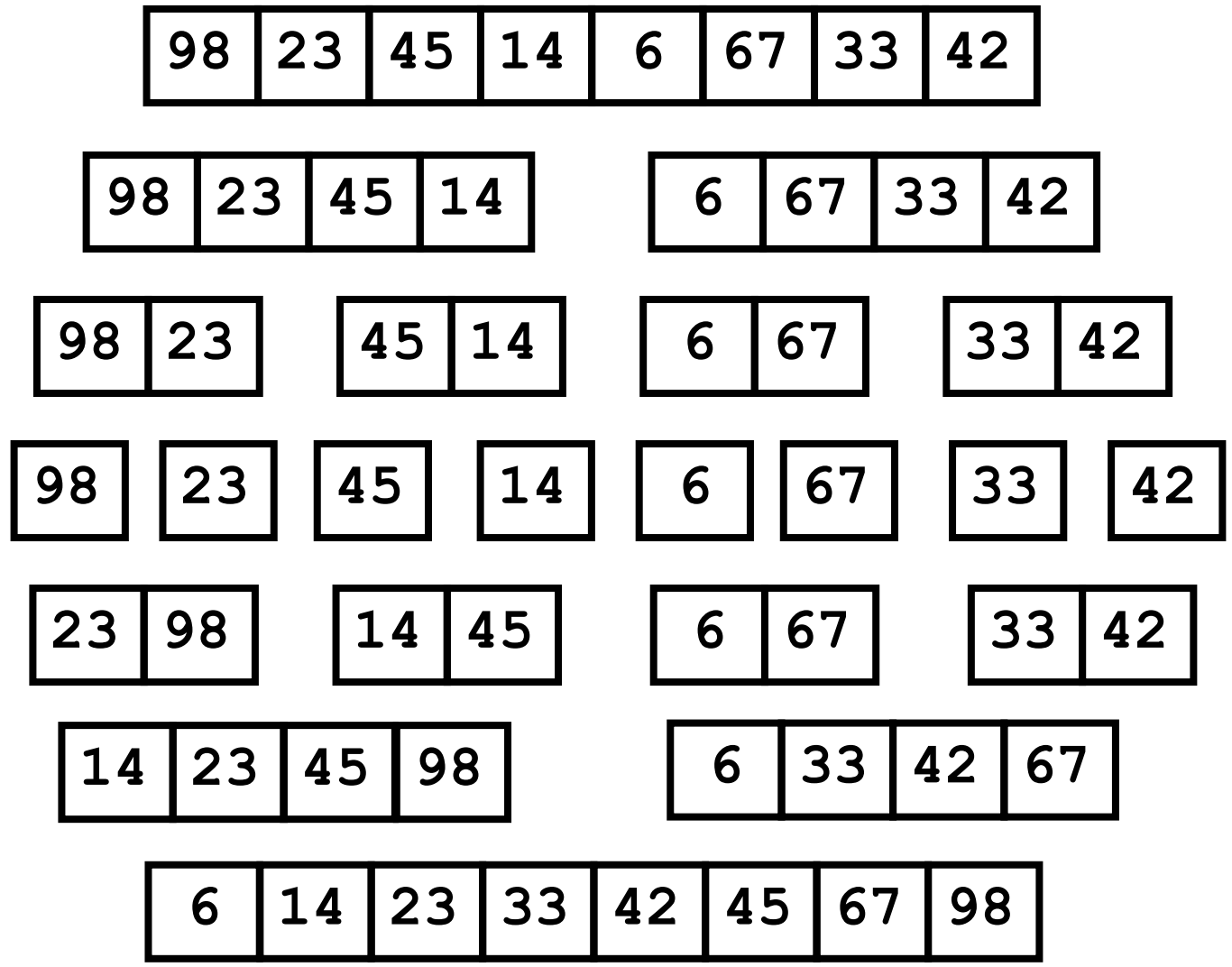

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33
Merge

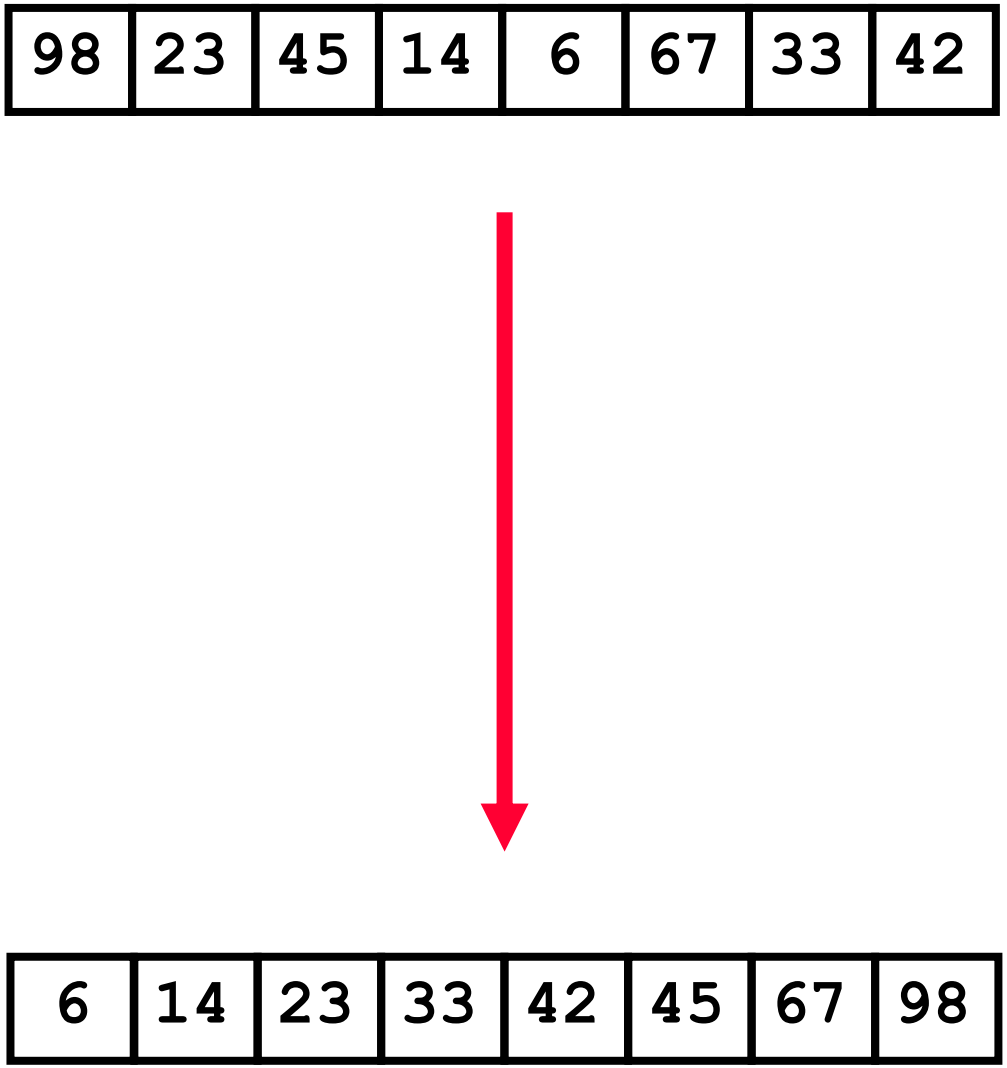

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45 67
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45 67 98
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45 67 98


98 23 45 14 6 67 33 42
6 14 23 33 42 45 67 98

Algoritma Merge Sort

```
1. void MergeSortRekursif(a,  b)
2.   jika (a<b) maka kerjakan baris 3-6
3.         tengah = (a+b) / 2 ;
4.         MergeSortRekursif(a,tengah);
5.         MergeSortRekursif(tengah+1,b);
6.         Merge(a,tengah,b);
```

Fungsi Merge

```
1. void Merge(int kiri, int tengah, int kanan
2.    l1 ← kiri
3.    u1 ← tengah
4.    l2 ← tengah+1
5.    u2 ← kanan
6.    k ← l1;
```

Fungsi Merge()

```
7.      selama  (l1<=u1 && l2<=u2) kerjakan baris 8-14
8.              jika (Data[l1] < Data[l2]) maka kerjakan 9-10
9.                      aux[k] ← Data[l1]
10.                     l1 ← l1 + 1
11.             jika tidak kerjakan baris 12-13
12.                     aux[k] = Data[l2]
13.                     l2 ← l2 + 1
14.             k ← k + 1

15.     selama (l1<=u1) kerjakan baris 16-18
16.             aux[k] = Data[l1]
17.             l1 ← l1 + 1
18.             k ← k + 1
```

```
19.  selama (l2<=u2) kerjakan baris 20-22
20.        aux[k] = Data[l2]
21.        l2 ← l2 + 1
22.        k ← k + 1

23.  k ← kiri
24.  selama (k <=kanan) kerjakan baris 25-26
25.        Data[k] = aux[k]
26.        k ← k + 1
```

Kesimpulan

- **Bagi array yang tidak terurut menjadi dua**
- **Sampai hanya subarray berisi satu elemen**
- **Selanjutnya gabungkan solusi sub problem bersama-sama**

Waktu Kompleksitas Mergesort

Kompleksitas waktu dari proses Rekursif.
T(n) adalah running time untuk worst-case untuk mengurutkan n data/bilangan.
Diasumsikan $n=2^k$, for some integer k.

```
void mergesort(vector<int> & A, int left, int right)
{
    if (left < right) {
        int center = (left + right)/2;
        mergesort(A,left,center);
        mergesort(A,center+1,right);
        merge(A,left,center+1,right);
    }
}
```

T(n/2)

T(n/2)

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

Terdapat 2 rekursif merge sort, kompleksitas waktunya T(n/2)

Proses Merge memerlukan waktu O(n) untuk menggabungkan Hasil dari merge sort rekursif

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

Waktu Kompleksitas Mergesort

$$T(n)$$ — **Recursive step**

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

$$= 2(2T(n/4) + O(n/2)) + O(n)$$ — **Recursive step**

$$= 4T(n/4) + 2 \cdot O(n/2)) + O(n)$$

$$= 4T(n/4) + O(2n/2)) + O(n)$$ — Collect terms

$$= 4T(n/4) + O(n) + O(n)$$

$$= 4(2T(n/8) + O(n/4)) + O(n) + O(n)$$ — **Recursive step**

$$= 8T(n/8) + 4 \cdot O(n/4) + O(n) + O(n)$$

$$= 8T(n/8) + O(4n/4) + O(n) + O(n)$$ — Collect terms

$$= 8T(n/8) + O(n) + O(n) + O(n)$$

$$T(n) = 2^k T(n/2^k) + k \cdot O(n)$$ — **Setelah level ke - k**

Waktu Kompleksitas Mergesort

$$T(n) = 2^k T(n/2^k) + k \cdot O(n)$$

Karena $n=2^k$, setelah level ke- k ($=\log_2 n$) pemanggilan rekursif, bertemu dengan (n=1)

Put $k= \log_2 n$, ($n=2^k$)
$T(n) = 2^k T(n/2^k) + kO(n) = nT(n/n) + kO(n)$
$= nT(1) + \log_2 n O(n) = nO(1) + O(n \log_2 n)$
$= O(n) + O(n \log_2 n)$
$= O(n \log_2 n) = O(n \log n)$

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

Perbandingan insertion sort dan merge sort (dalam detik)

n	Insertion sort	Merge sort	Ratio
100	0.01	0.01	1
1000	0.18	0.01	18
2000	0.76	0.04	19
3000	1.67	0.05	33
4000	2.90	0.07	41
5000	4.66	0.09	52
6000	6.75	0.10	67
7000	9.39	0.14	67
8000	11.93	0.14	85