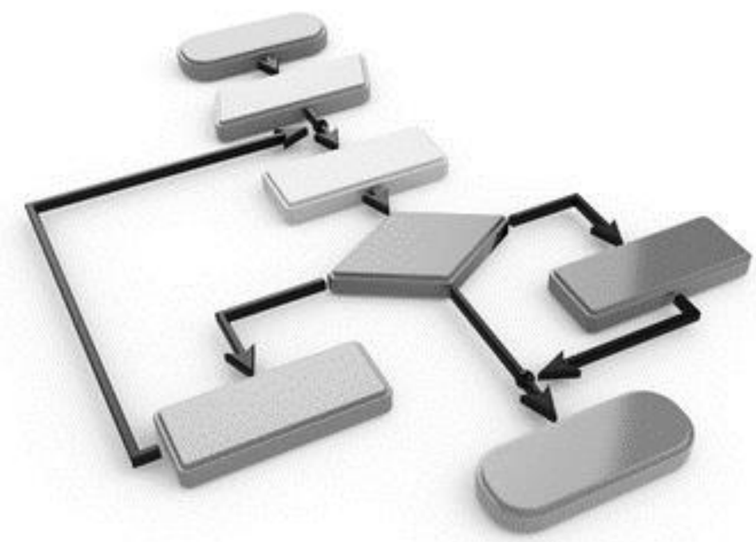


Lecture 02

Divide and Conquer (BinarySearch & Mergesort)

CSE373: Design and Analysis of Algorithms

A motivating Example of D&C Algorithm
Binary Search (recursive)

// Returns location of x in the **sorted** array A[first..last] if x is in A, otherwise returns -1

Algorithm BinarySearch(A, first, last, x)

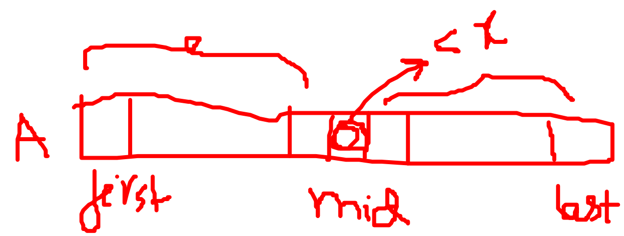
if last ≥ first **then**

mid = first + (last - first)/2

// If the element is present at the middle itself

if A[mid] = x **then**

return mid



// If element is smaller than mid, then it can only be present in left sub-array

else if A[mid] > x **then**

return BinarySearch(A, first, mid-1, x)

left

// Otherwise the element can only be present in the right sub-array

else

return BinarySearch(A, mid+1, last, x);

return -1 // We reach here when element is not present in A

Initial call: BinarySearch(A,1,n,key) where key is an user input which is to be sought in A

Retrieving an Item from Sorted List

- Find **84**

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Retrieving an Item from Sorted List

- Find **84**

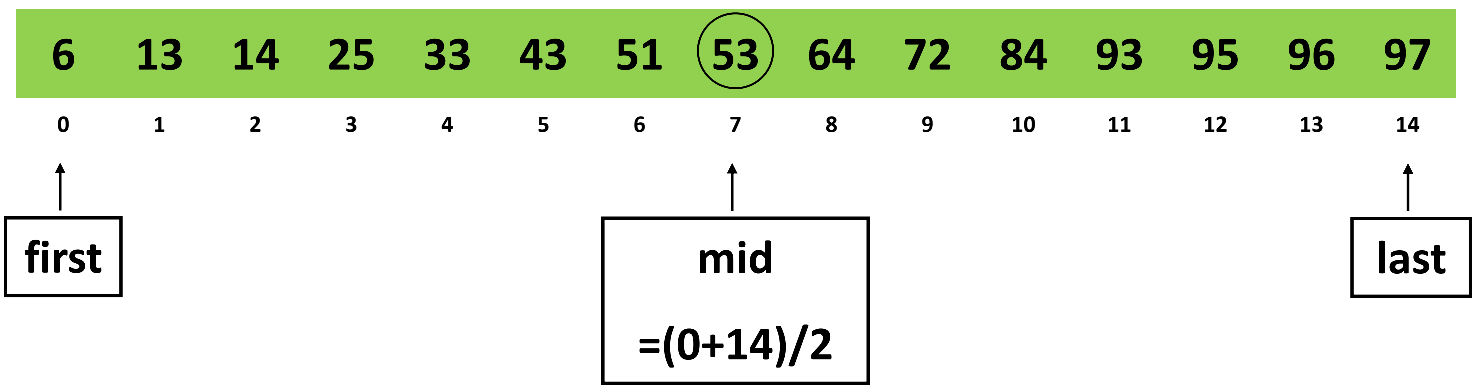
6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

first **last**

- **Step 1**

Retrieving an Item from Sorted List

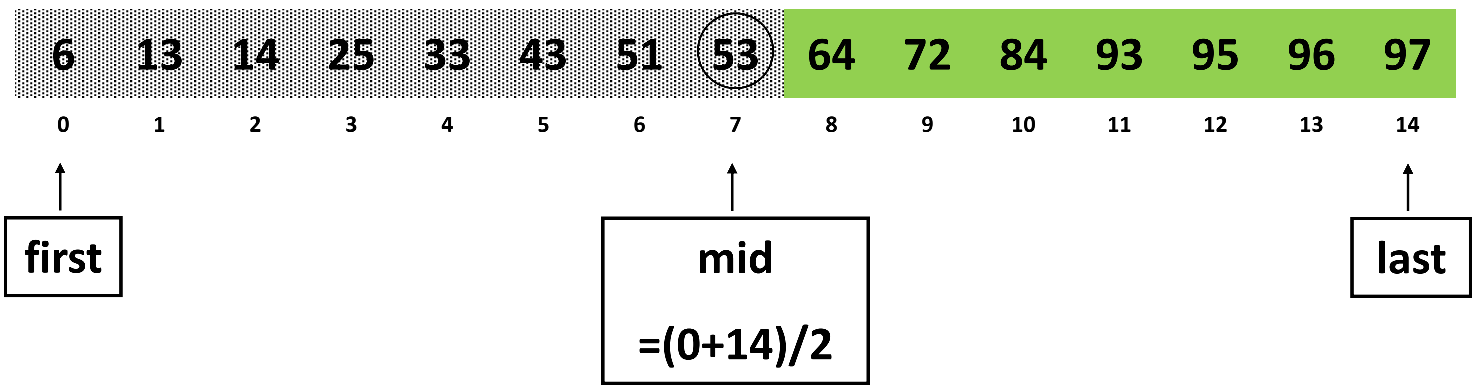
- Find **84**



- **Step 1**

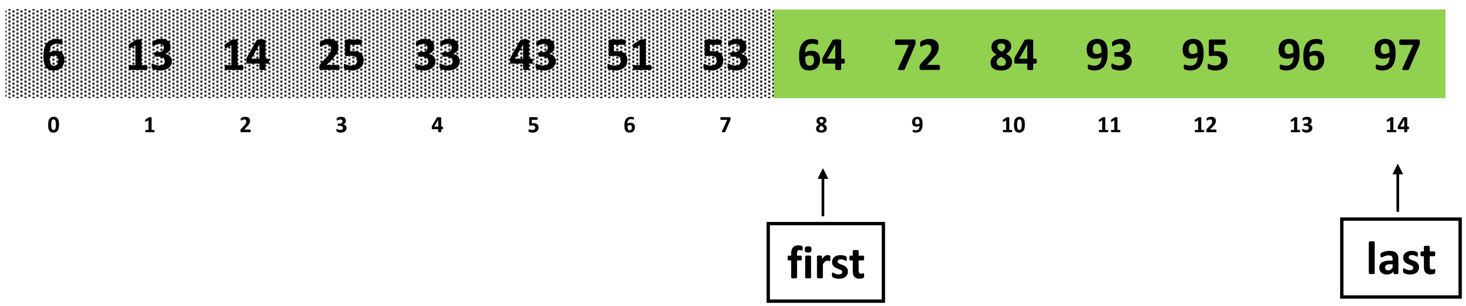
Retrieving an Item from Sorted List

- Find **84**



Retrieving an Item from Sorted List

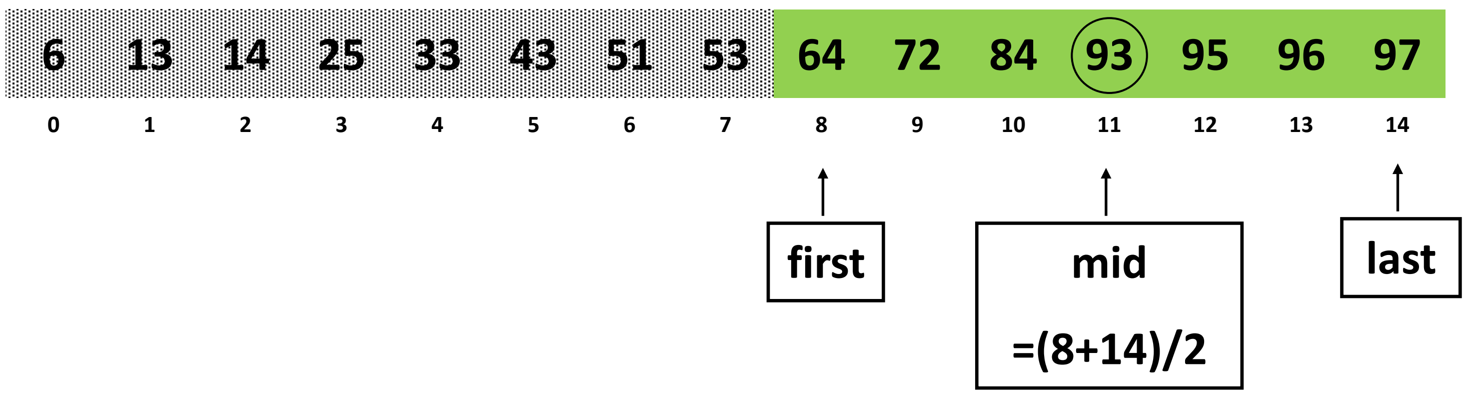
- Find **84**



- **Step 2**

Retrieving an Item from Sorted List

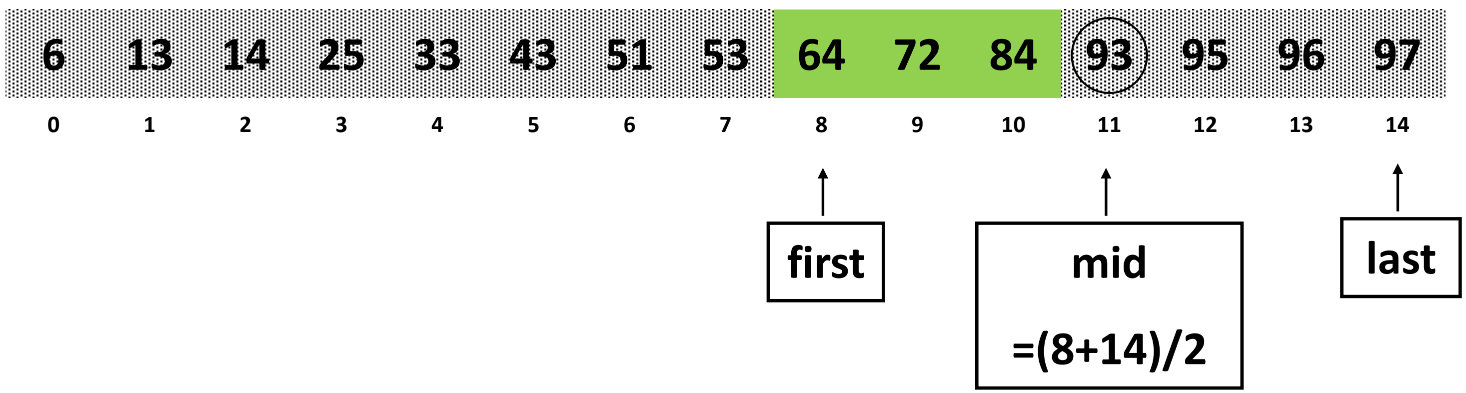
- Find **84**



- **Step 2**

Retrieving an Item from Sorted List

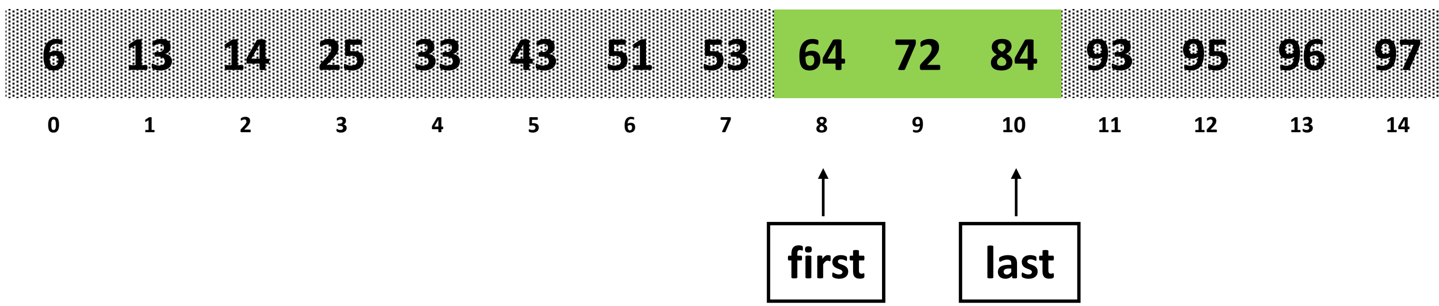
- Find **84**



- **Step 2**

Retrieving an Item from Sorted List

- Find **84**

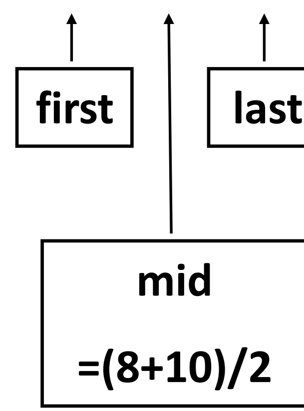


- **Step 3**

Retrieving an Item from Sorted List

- Find **84**

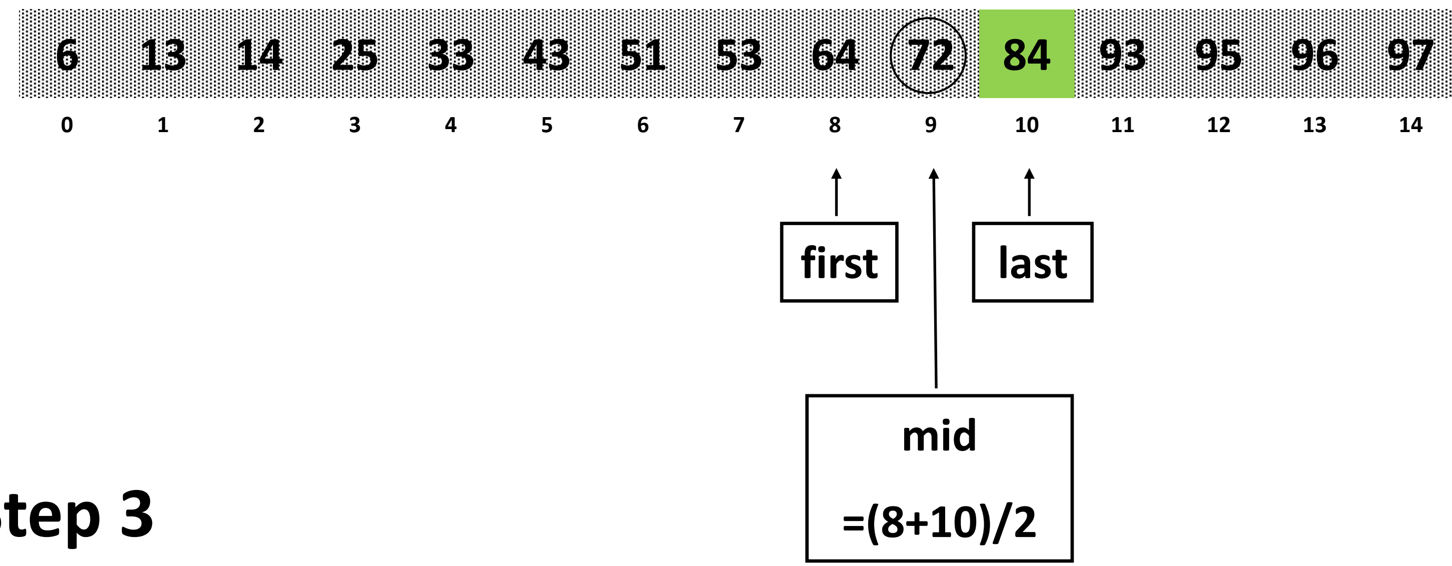
6	**13**	**14**	**25**	**33**	**43**	**51**	**53**	**64**	**72**	**84**	**93**	**95**	**96**	**97**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14



- **Step 3**

Retrieving an Item from Sorted List

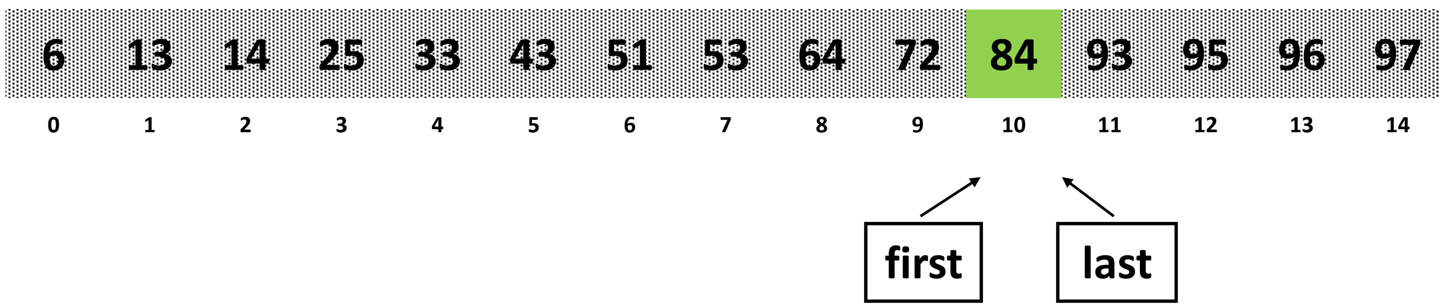
- Find **84**



- **Step 3**

Retrieving an Item from Sorted List

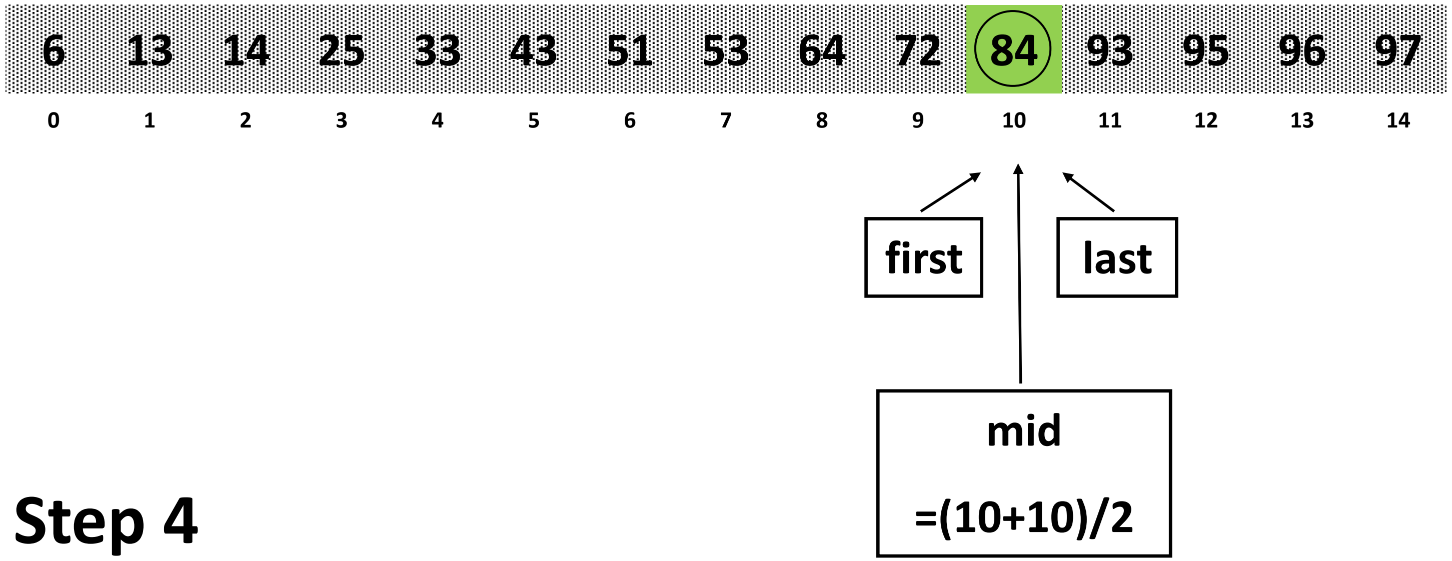
- Find **84**



- **Step 4**

Retrieving an Item from Sorted List

- Find **84**



- **Step 4**
- **84 found at the midpoint**

Binary Search (recursive) Algorithm

// Returns location of x in the **sorted** array A[first..last] if x is in A, otherwise returns -1

Algorithm BinarySearch(A, p, q, x)

  if last ≥ first **then**

    mid $\leftarrow \lfloor (p+q)/2 \rfloor$

    // If the element is present at the middle itself

    if A[mid] = x **then**

      return mid

O(1)

    // If element is smaller than mid, then it can only be present in left sub-array

    if A[mid] > x **then**

      return BinarySearch(A, p, mid-1, x)

    // Otherwise the element can only be present in the right sub-array

    else

      return BinarySearch(A, mid+1, q, x)

  return -1   // We reach here when element is not present in A

Initial call: BinarySearch(A,1,n,key) where key is an user input which is to be sought in A

Time: Θ(lg n), why?

for(i=n; i>=1; i/=2){— }

for(i=1; i<=n; i*=2){

search area	time
n	1
n/2	1
n/4	1
.	
.	
.	
$n/2^k$ = 1	1

Let $2^k = n$

T(n) = 1 + . . . + 1

= k+1

Divide and Conquer (D&C)

- In general, has 3 steps:
  - ***Divide*** the problem into independent sub-problems that are similar to the original but smaller in size
  - ***Conquer*** the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
  - ***Combine*** the solutions to create a solution to the original problem (this step may be empty)

D&C Algorithm Example: Binary Search

***Searching Problem*:** Search for item in a sorted sequence *A* of *n* elements

***Divide*:** Divide the *n*-element input array into two subarray of ≈ *n/2* elements each:

$$m \leftarrow \lfloor (p+q)/2 \rfloor$$

Conquer: Search either of the subarrays recursively by calling BinarySearch on the appropriate subarray:

```
if A[m] > x  then
    return BinarySearch(A, p, m-1, x)
else
    return BinarySearch(A, m+1, q, x)
```

***Combine*:** Nothing to be done

D&C Example: Merge Sort (Section 2.3)

Sorting Problem*:** Sort a sequence *A* of *n* elements into non-decreasing order: **MergeSort (*A[p..r]*) //sort *A[p..r]

***Divide*:** Divide the *n*-element input array into two subarray of ≈ *n/2* elements each [easy]:

$q \leftarrow \lfloor (p+r)/2 \rfloor$

Conquer: Sort the two subsequences recursively by calling merge sort on each subsequence [easy]:

MergeSort (*A[p .. q]*) // *A[p .. q]* becomes sorted after this call

MergeSort (*A[q+1 .. r]*) //*A[q+1..r]* becomes sorted after this call

***Combine*:** Merge the two sorted subsequences to produce the sorted sequence [how?]

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging two sorted subsequeces

Unsorted

6	18	56	62	1	9	15	43

Sorted **Sorted**

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging

TimSort

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging

1							

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging

1							

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subseqeuces

6	18	56	62	1	9	15	43

Merging

1	6						

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging

1	6						

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subseqeuces

6	18	56	62	1	9	15	43

Merging

1	6	9					

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging

1	6	9					

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subseqeuces

6	18	56	62	1	9	15	43

Merging

1	6	9	15				

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subseqeuces

6	18	56	62	1	9	15	43

Merging

1	6	9	15				

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging

1	6	9	15	18			

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging

1	6	9	15	18			

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging

1	6	9	15	18	43		

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging

1	6	9	15	18	43		

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subsequeces

6	18	56	62	1	9	15	43

Merging

1	6	9	15	18	43	56	62

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subsequeces

1	6	9	15	18	43	56	62

Merging

1	6	9	15	18	43	56	62

Left half

Right half

Minimum between first elements in both halves

Merging two sorted subsequeces

1	**6**	**9**	**15**	**18**	**43**	**56**	**62**

Merging two sorted subseqeces

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

```
n1 ← q − p + 1
n2 ← r − q
for i ← 1 to n1
    do L[i] ← A[p + i − 1]
for j ← 1 to n2
    do R[j] ← A[q + j]
L[n1+1] ← ∞
R[n2+1] ← ∞
i ← 1
j ← 1
for k ← p to r
    do if L[i] ≤ R[j]
        then A[k] ← L[i]
                i ← i + 1
        else A[k] ← R[j]
                j ← j + 1
```

Input: Array containing sorted subarrays A[p..q] and A[q+1..r].

Output: Merged sorted subarray in A[p..r].

***Sentinels**, to avoid having to check if either subarray is fully copied at each step.*

Time complexity of Merge

```
Merge(A, p, q, r)          //Let r-p+1 = n
n1 ← q − p + 1             //Θ(1)
n2 ← r − q                 //Θ(1)
    for i ← 1 to n1  //Θ(q-p+1)
        do L[i] ← A[p + i − 1]
    for j ← 1 to n2  //Θ(r-q)
        do R[j] ← A[q + j]
    L[n1+1] ← ∞
    R[n2+1] ← ∞
    i ← 1
    j ← 1
    for k ←p to r    //Θ(r-p+1) = Θ(n)
        do if L[i] ≤ R[j]
            then A[k] ← L[i]
                 i ← i + 1
            else A[k] ← R[j]
                 j ← j + 1
//Total time: Θ(n)
```

Input: Array containing sorted subarrays A[p..q] and A[q+1..r].

Output: Merged sorted subarray in A[p..r].

Merge Sort (recursive/D&C version)

MergeSort (A, p, r) *// sort A[p..r] via merge sort*

if $p < r$

  then $q \leftarrow \lfloor (p+r)/2 \rfloor$ *//divide*

    MergeSort (A, p, q) *//conquer*

    MergeSort (A, q+1, r) *//conquer*

    Merge (A, p, q, r) *//combine: merge A[p..q] with A[q+1..r]*

Initial Call: *MergeSort(A, 1, n)*

$p=1$ $\quad q=4$ $\quad r=n=8$

A

98	23	45	14	6	67	33	42

1 MS(A, 1, 8)

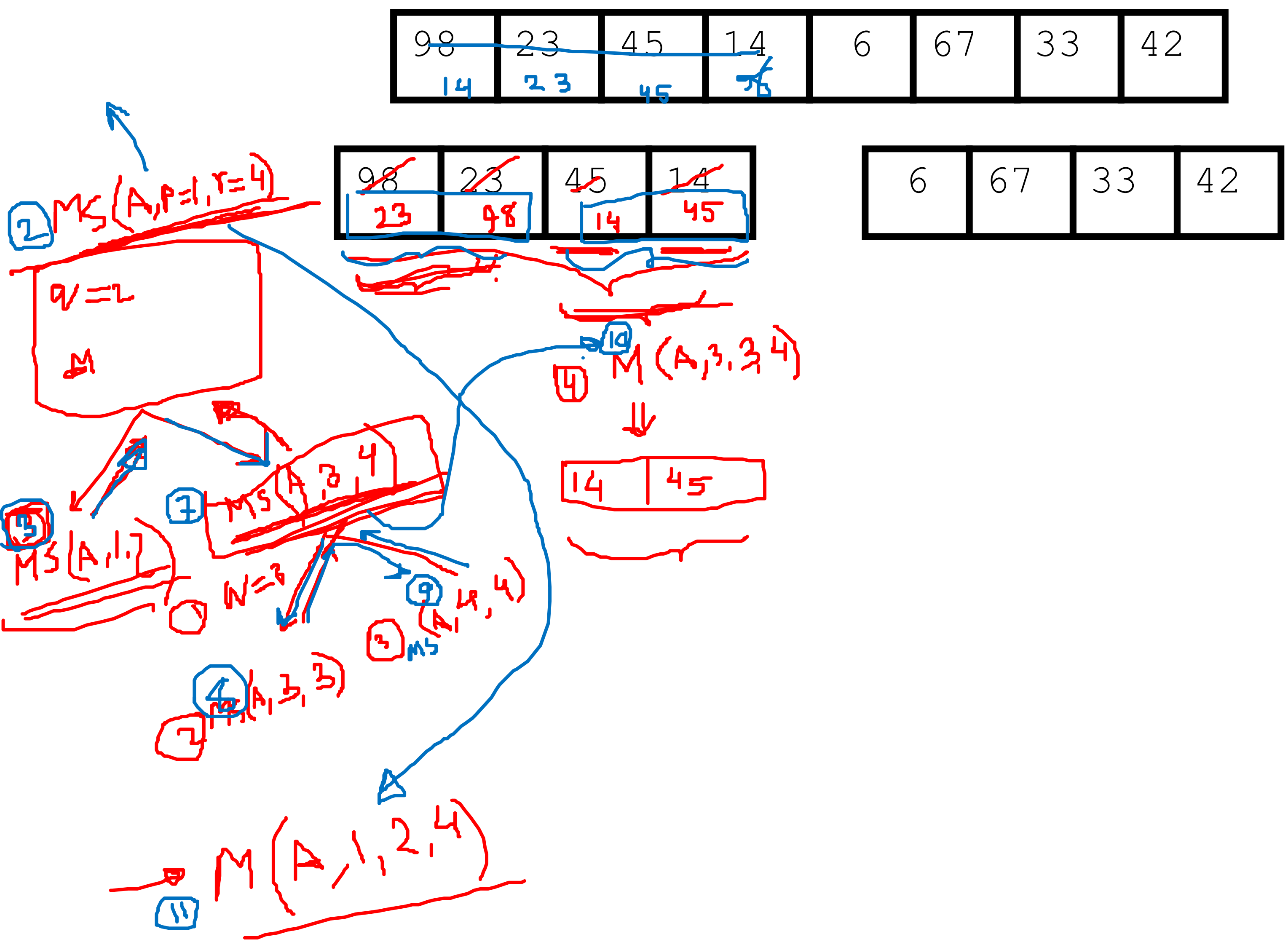
$q=4$
$p=1$
$r=8$

2 MS(A, 1, 4)

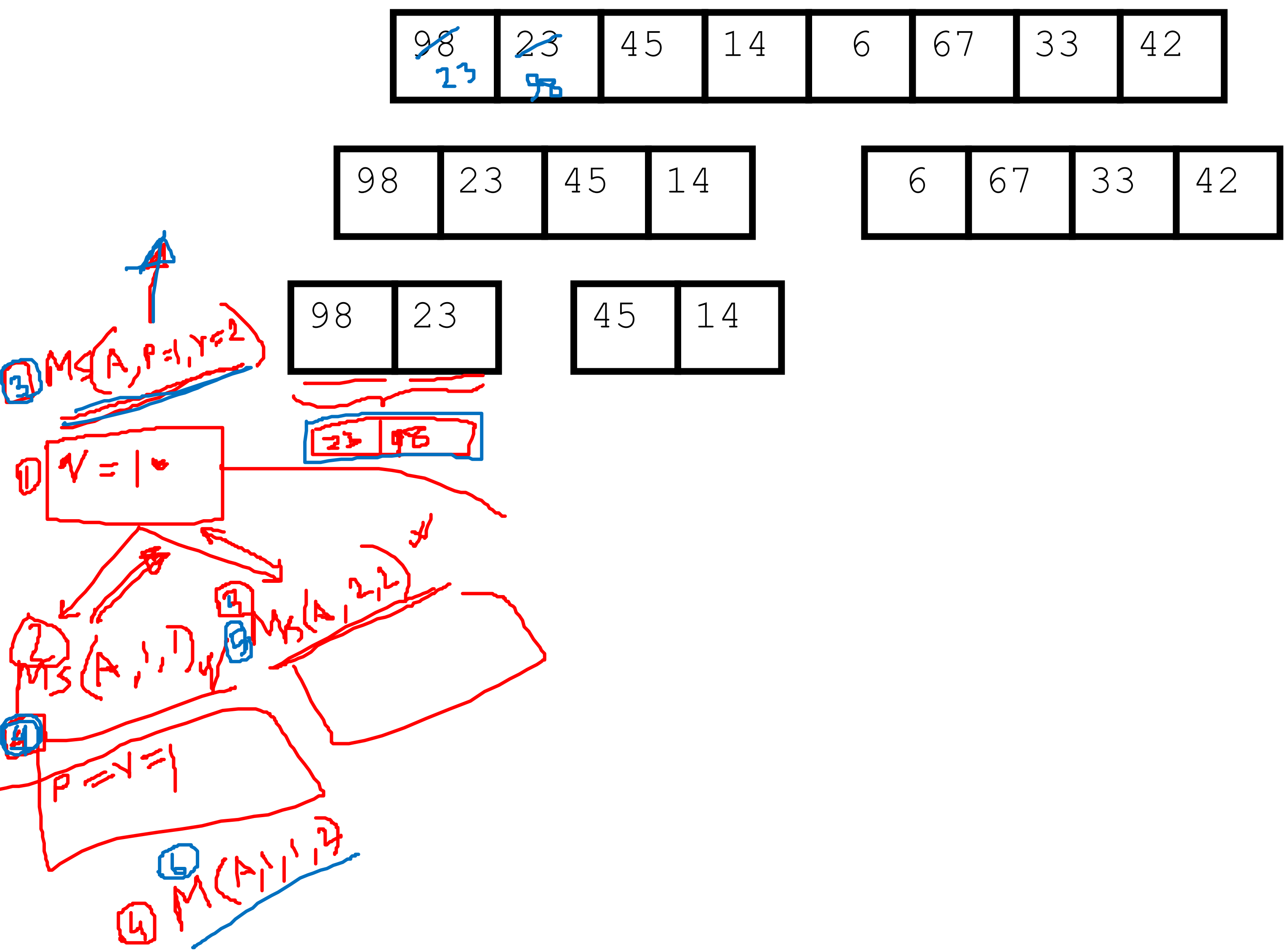
$q=2$

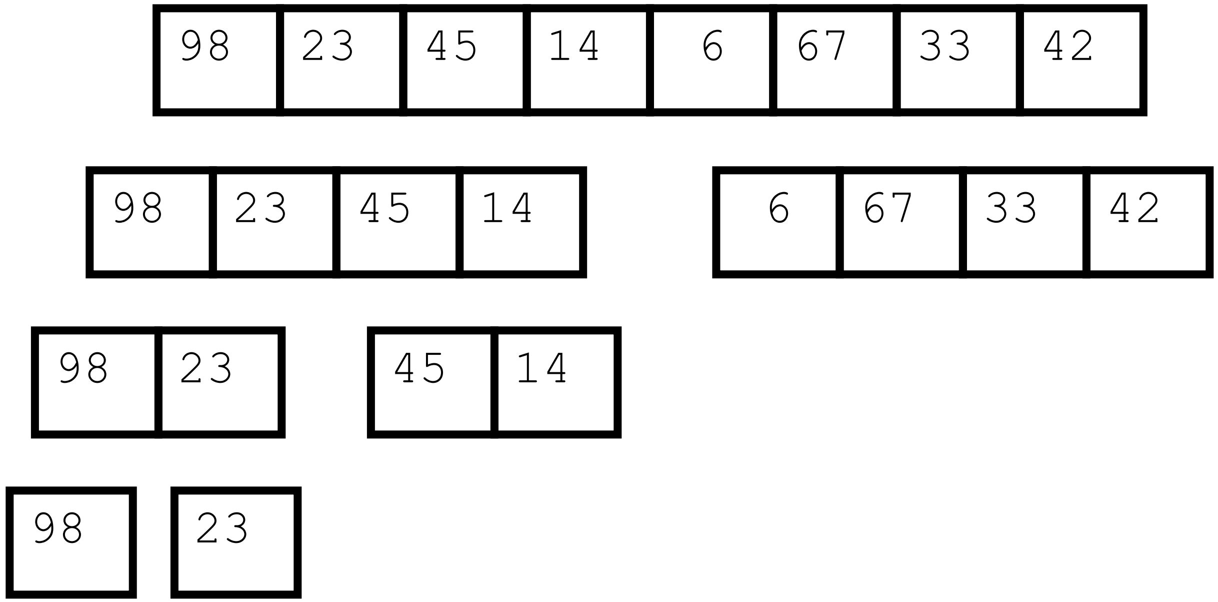
$p=1$
$r=4$

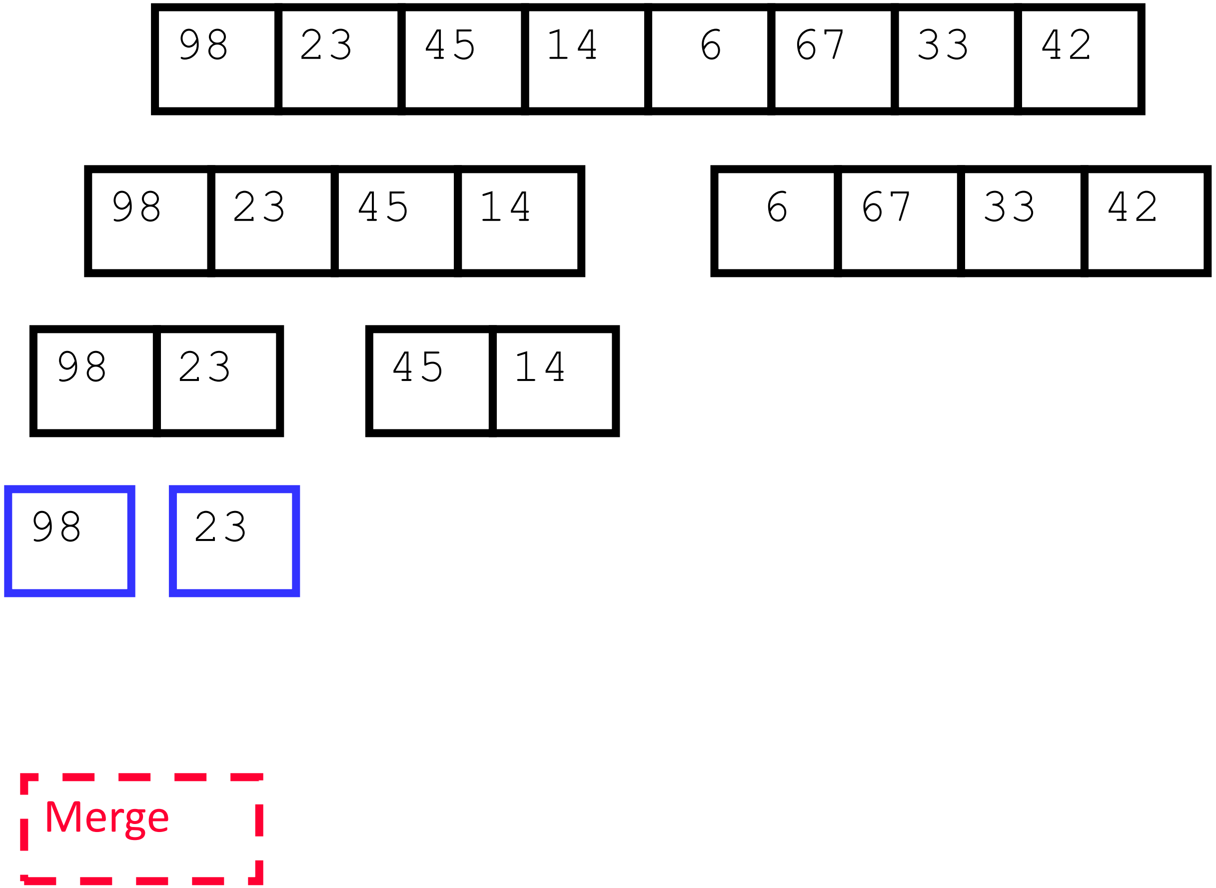
12 MS(A, 5, 8)

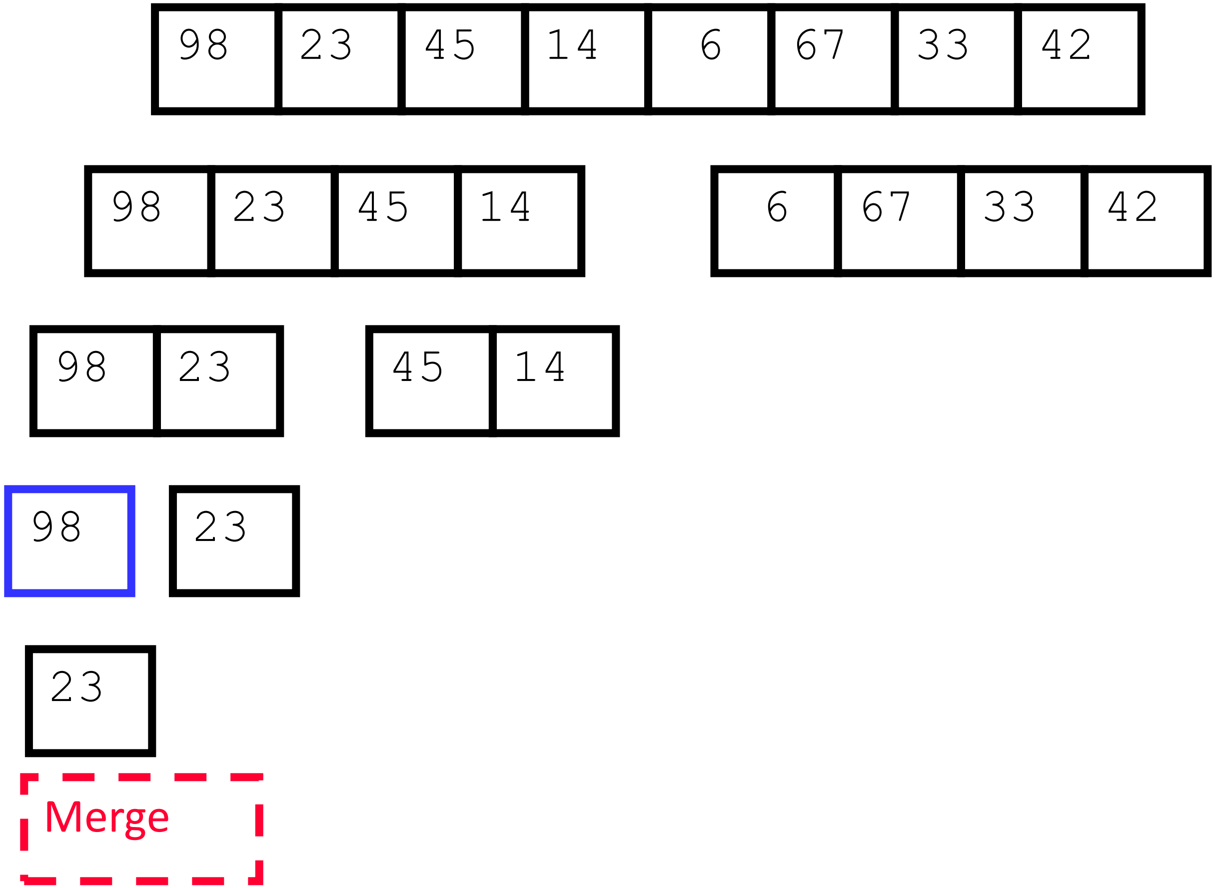


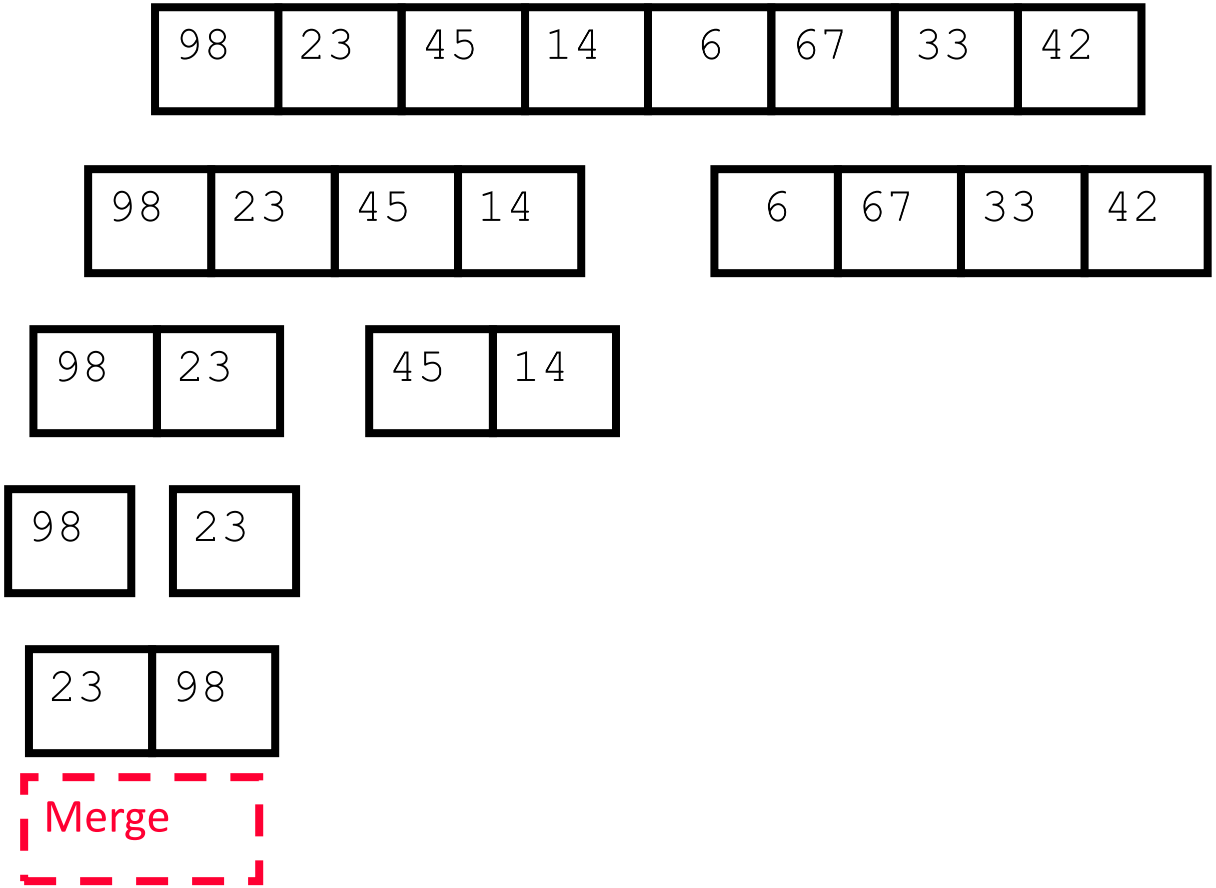
98 | 23 | 45 | 14 | 6 | 67 | 33 | 42
14 | 23 | 45
98 | 23 | 45 | 14
23 | 98 | 14 | 45
6 | 67 | 33 | 42
MS(A, p=1, r=4)
q/=2
MS(A,3,4)
MS(A,1,2)
M(A,3,3,4)
14 | 45
M(A,1,2,4)

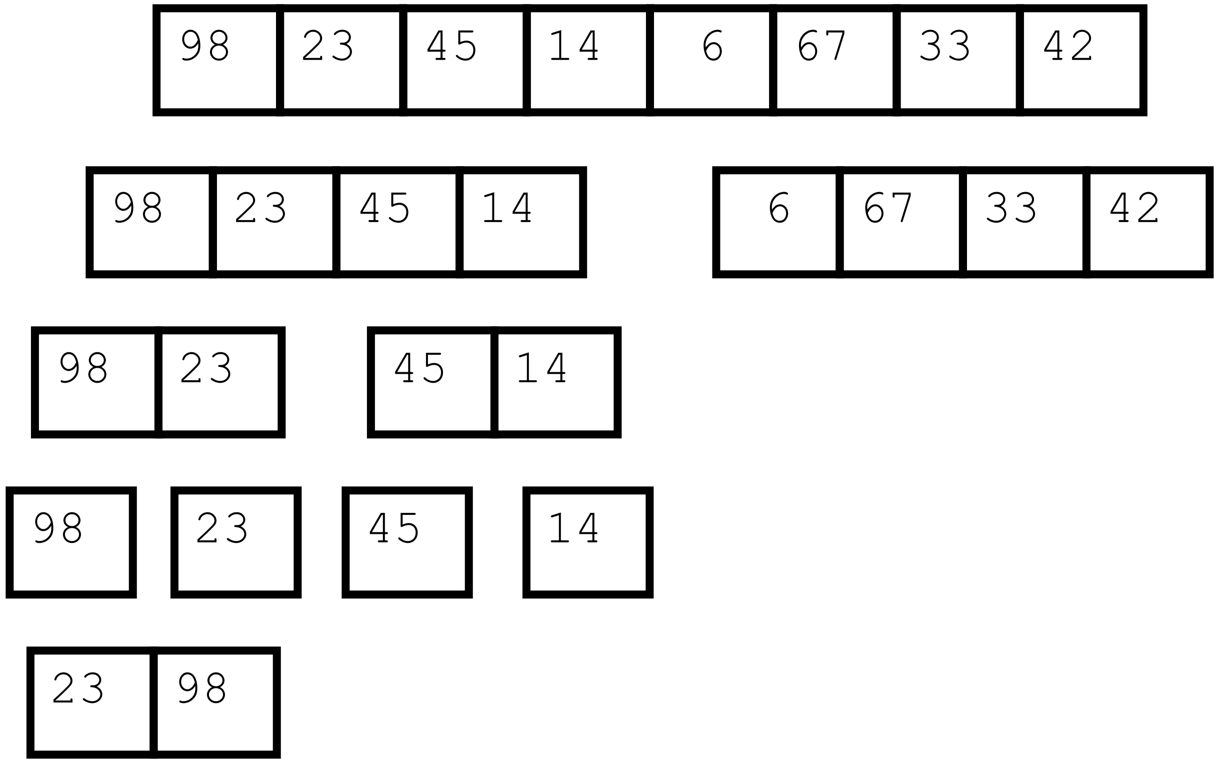

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
23 98
MS(A, p=1, r=2)
r = 1
MS(A,1,1)
MS(A,2,2)
p = r = 1
M(A,1,1,2)

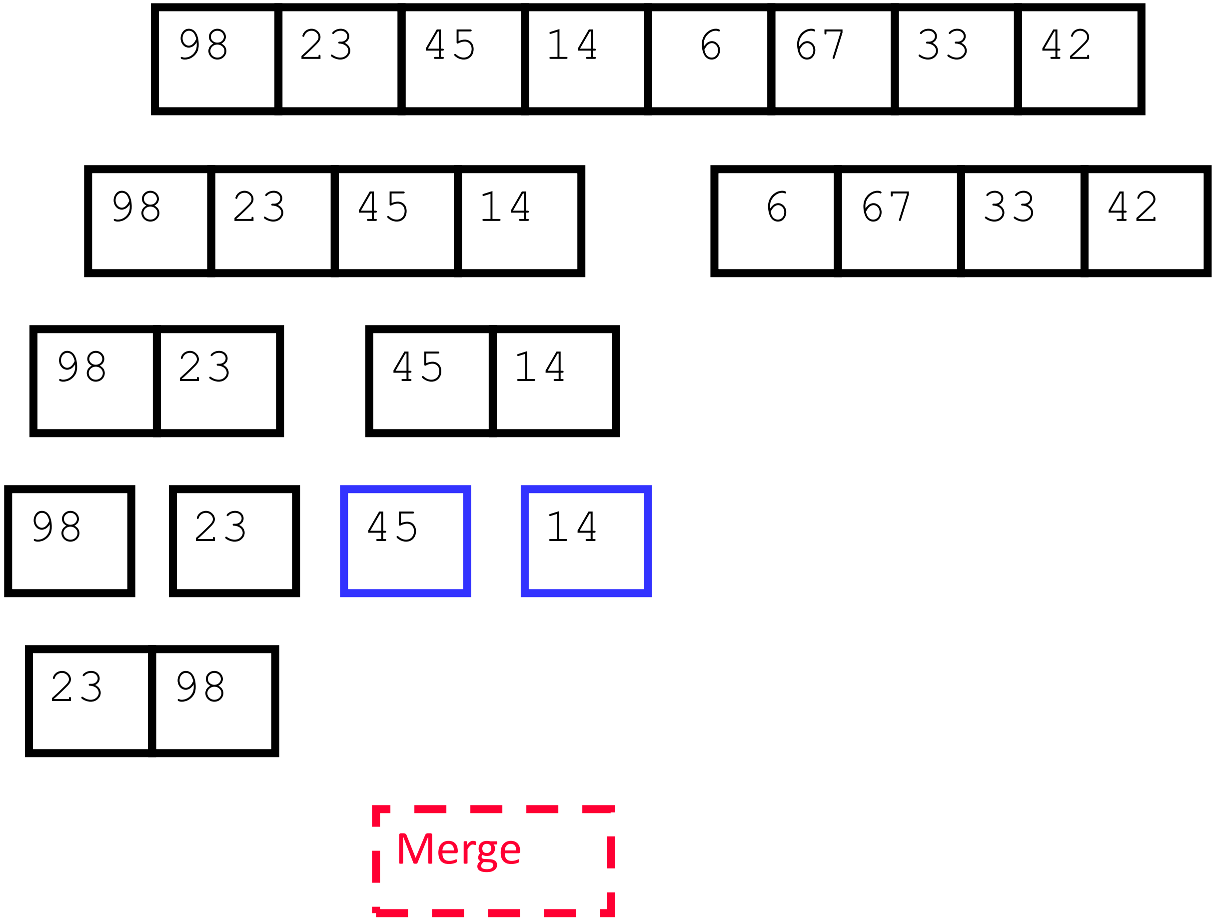

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23

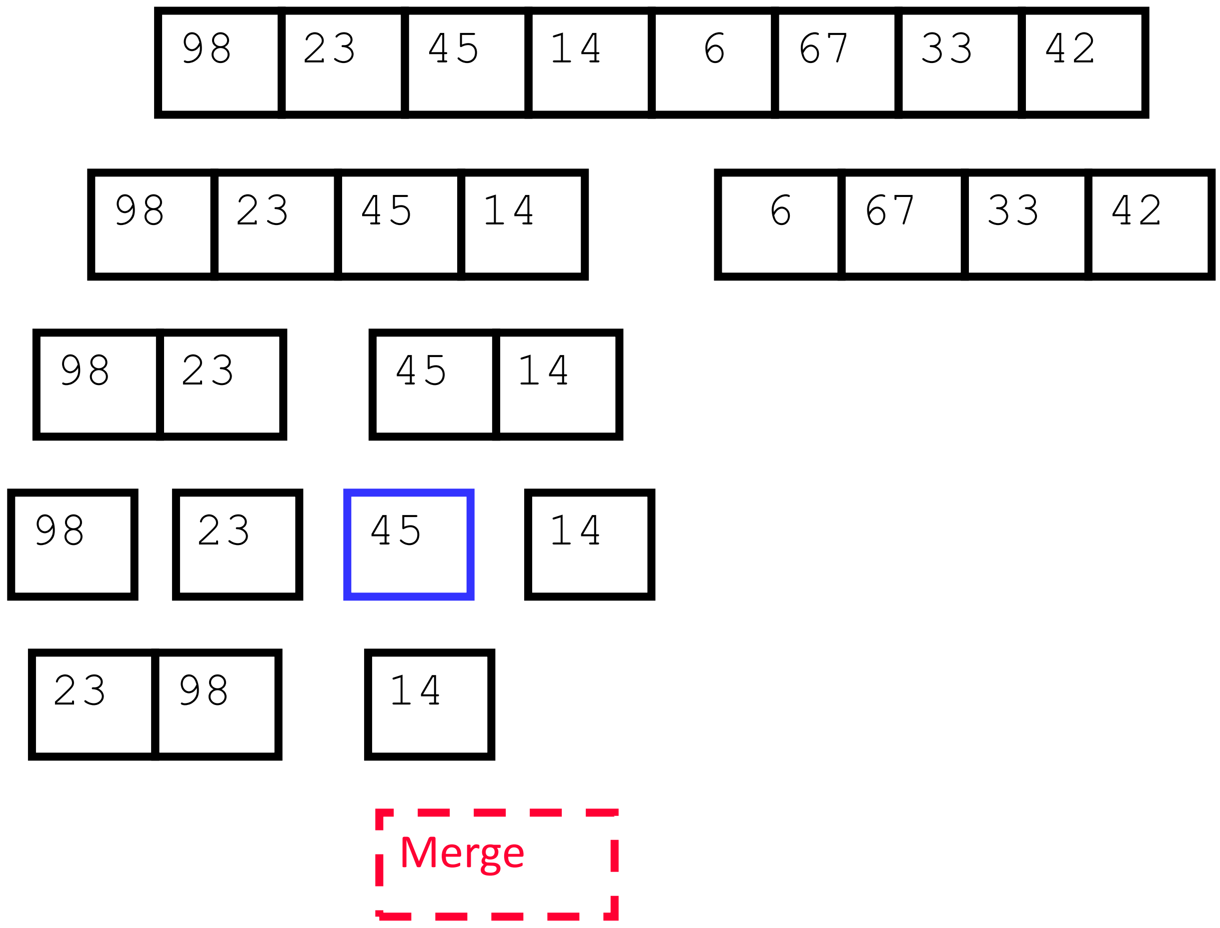

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
Merge


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

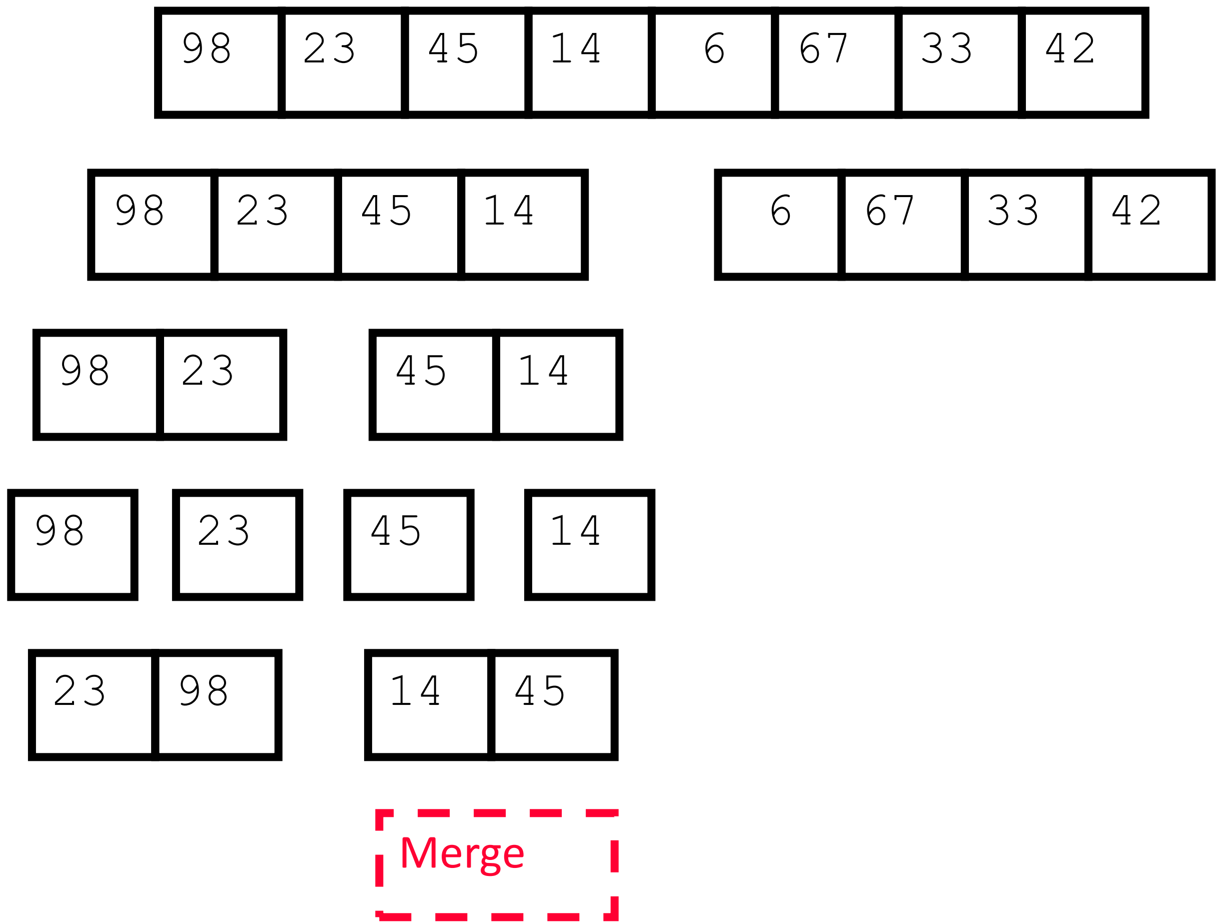

98
23
45
14
6
67
33
42
98
23
45
14
6
67
33
42
98
23
45
14
98
23
23
98
Merge


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

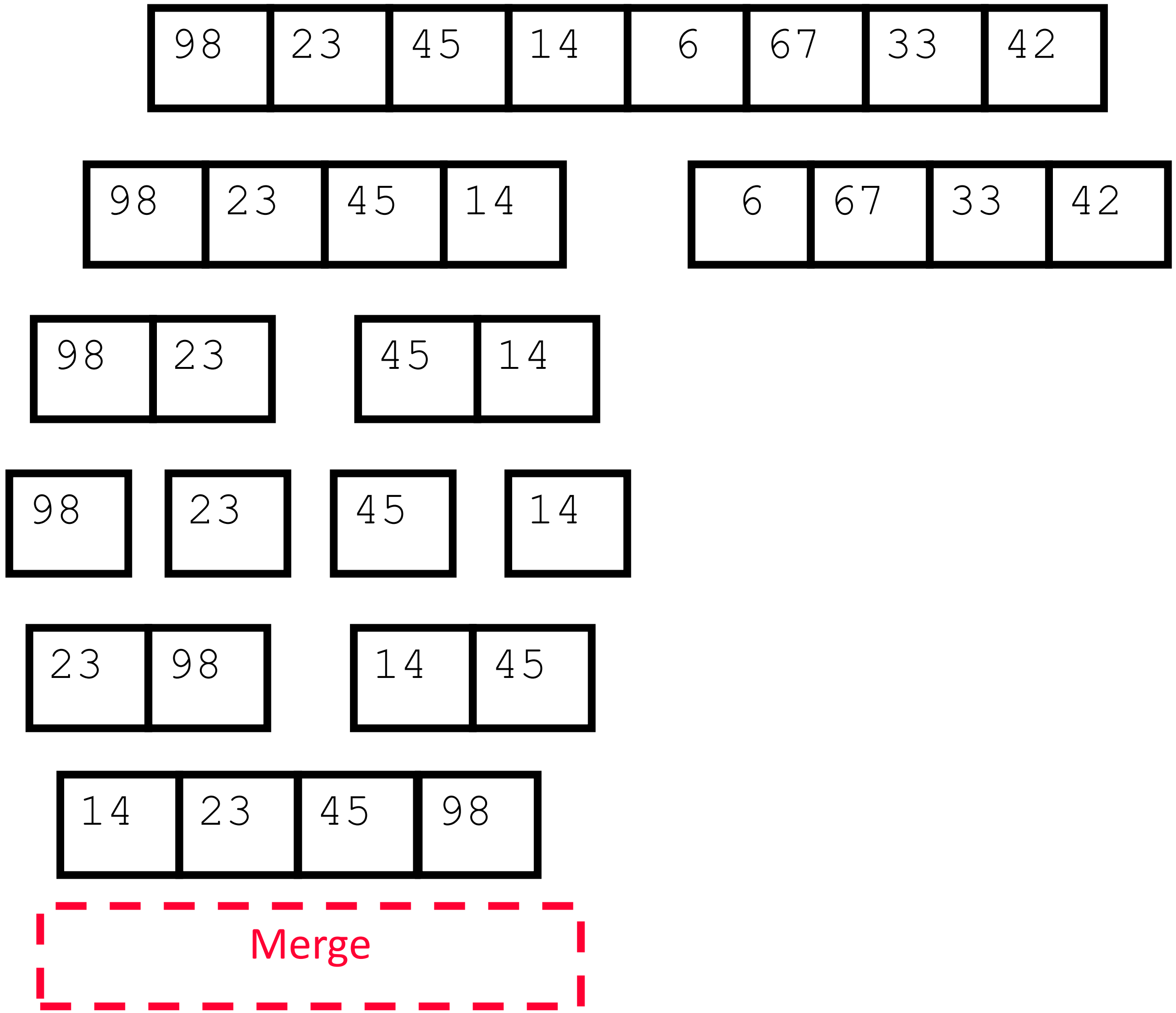

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
Merge

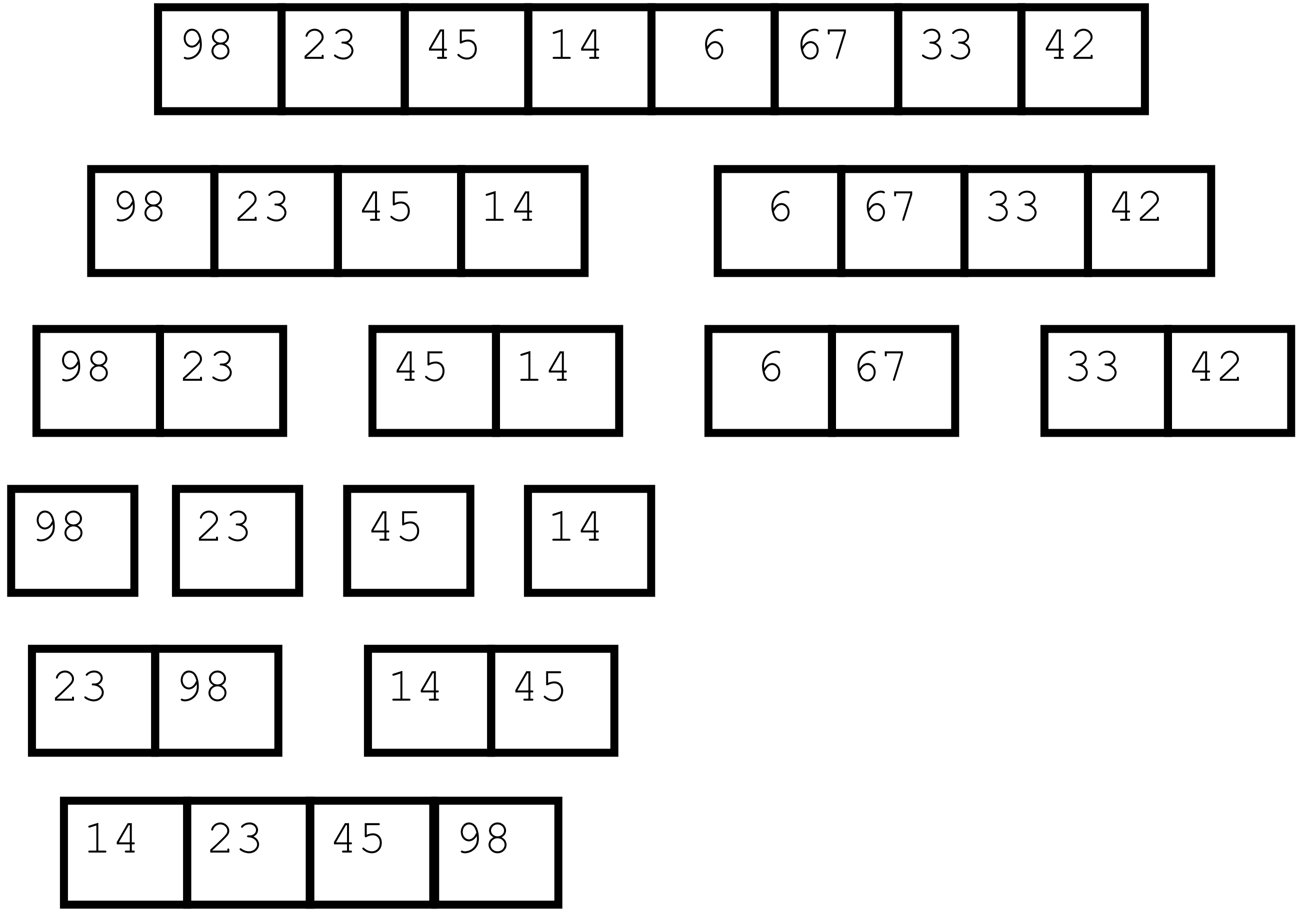

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
Merge

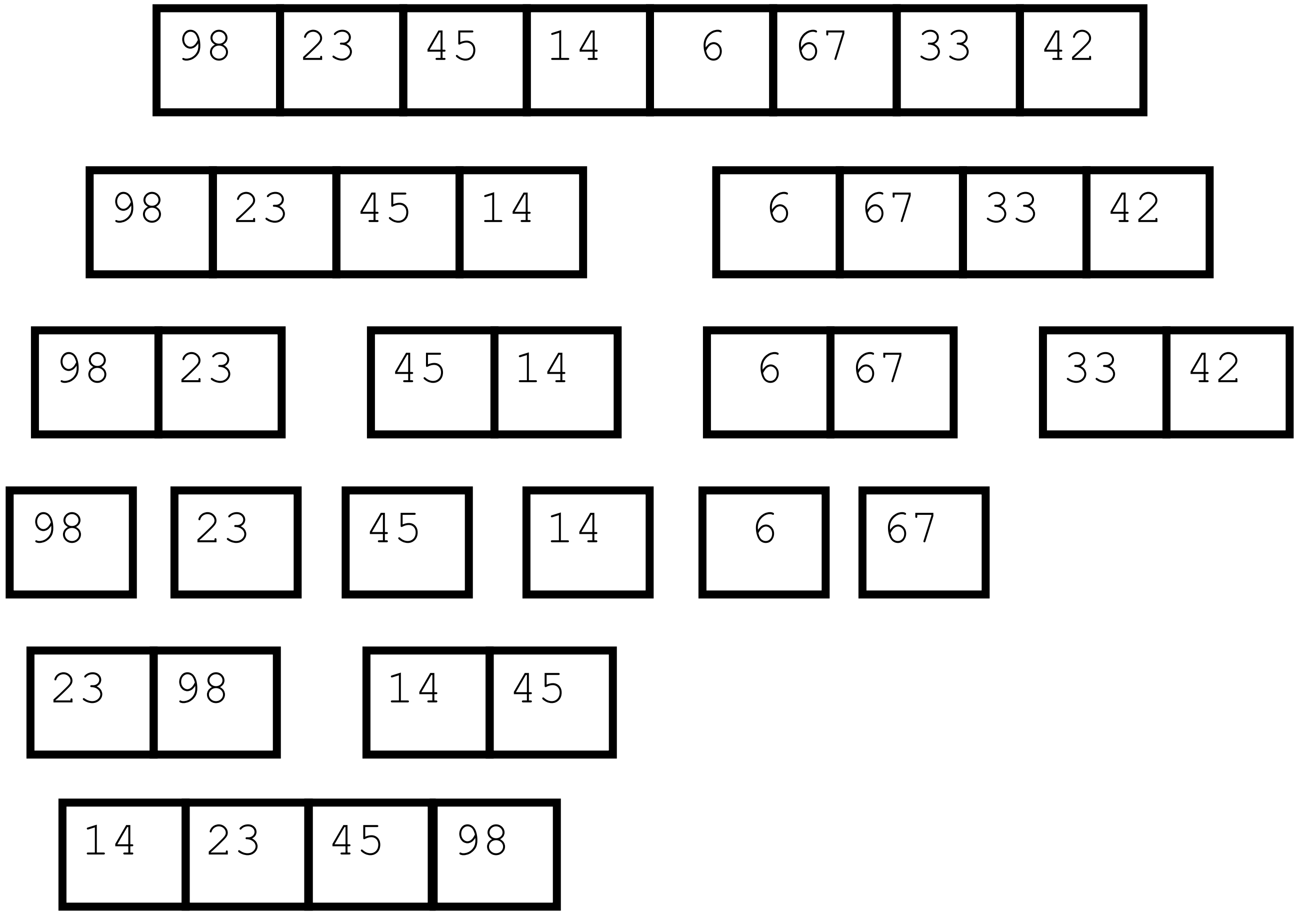

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14
Merge

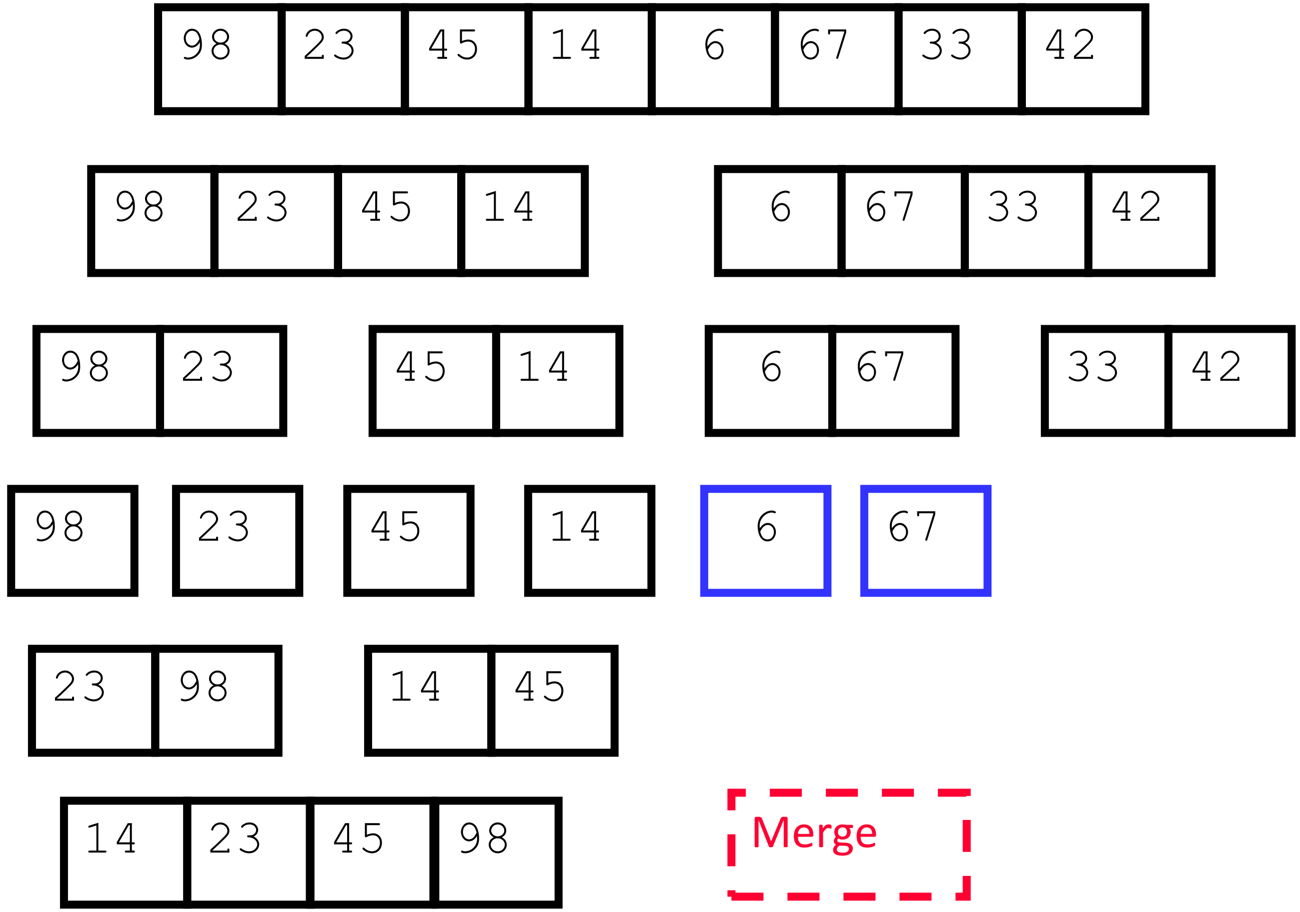

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23
Merge

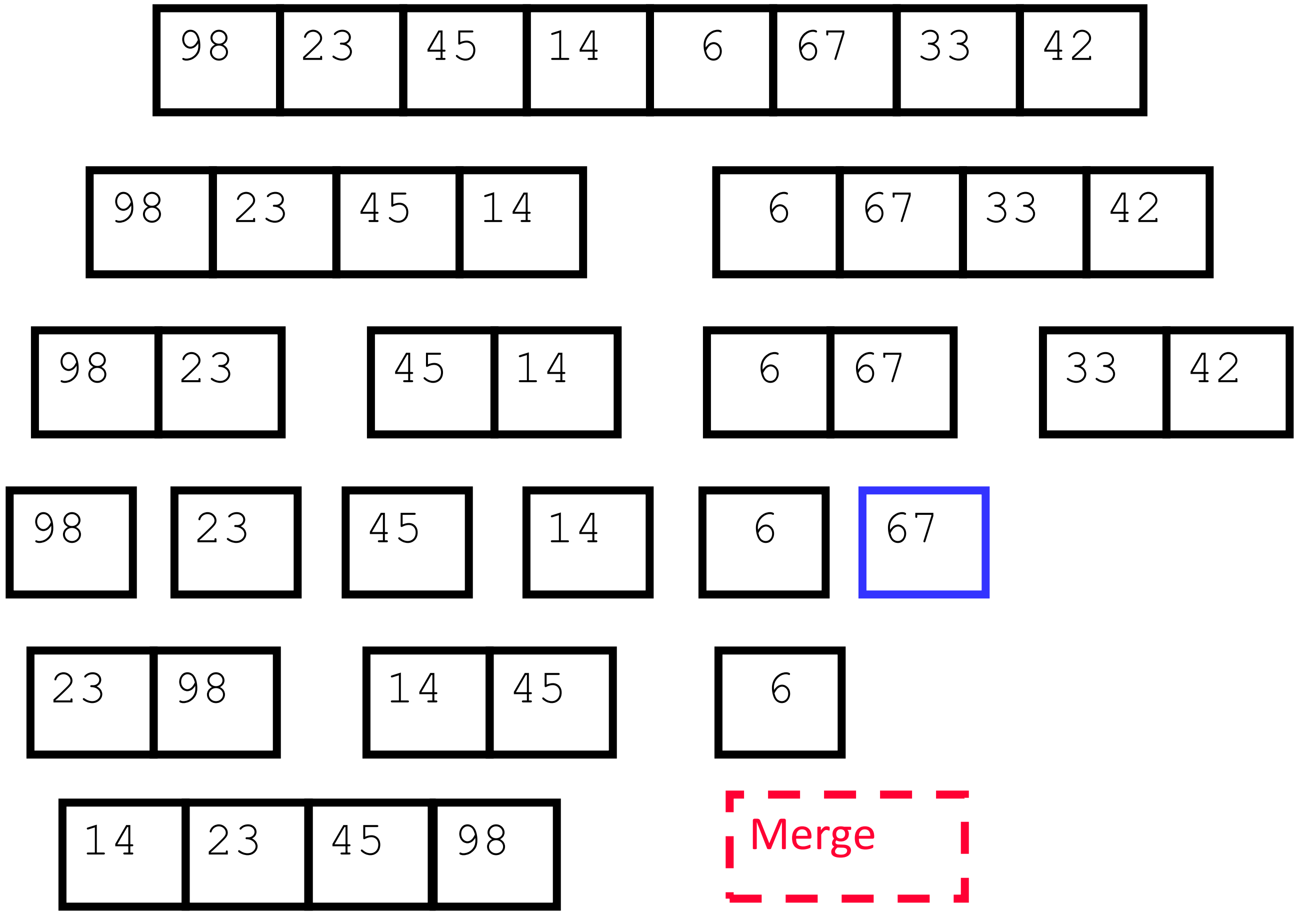
98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23 45
Merge

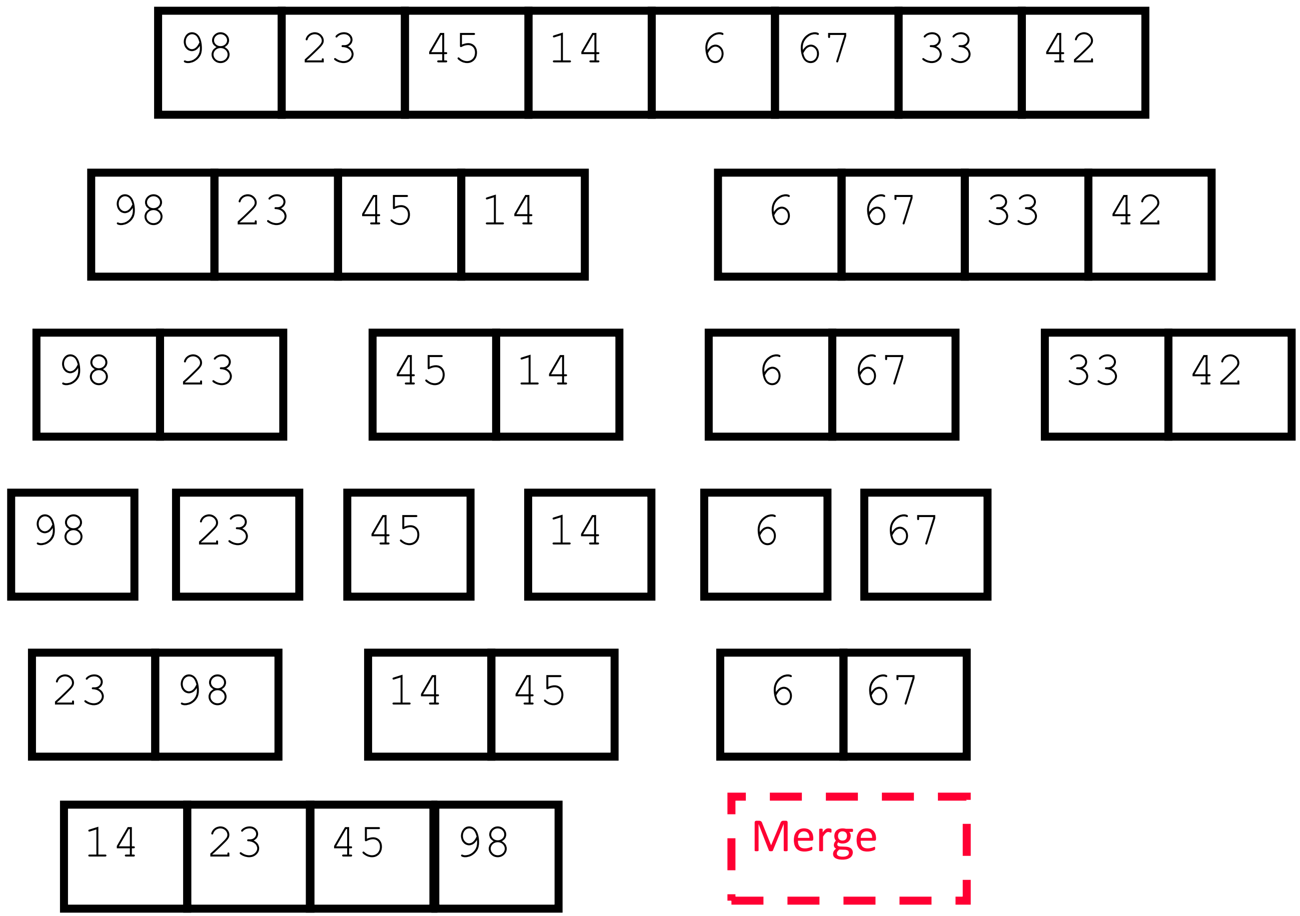

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23 45 98
Merge

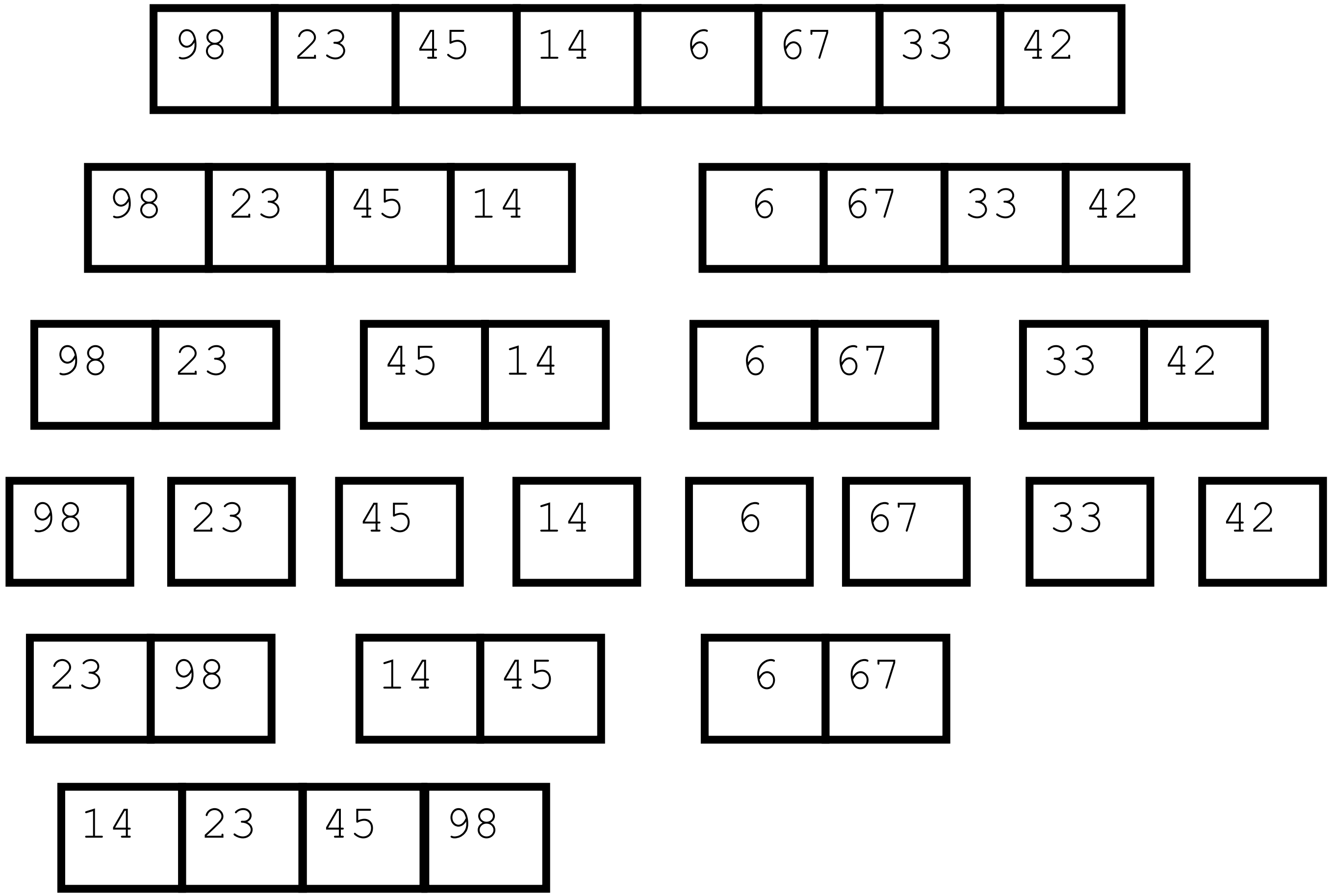

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
23 98
14 45
14 23 45 98

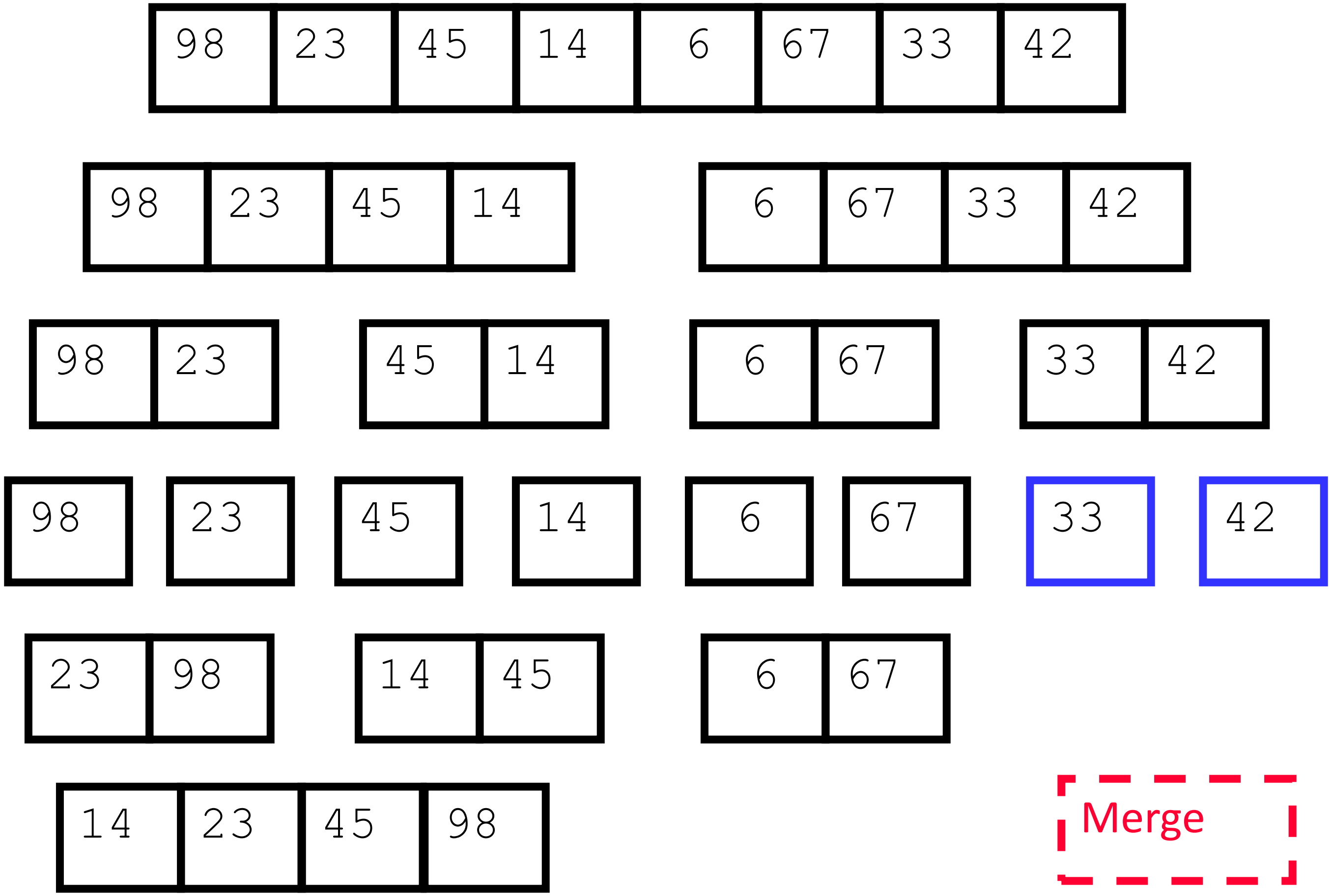

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
14 23 45 98

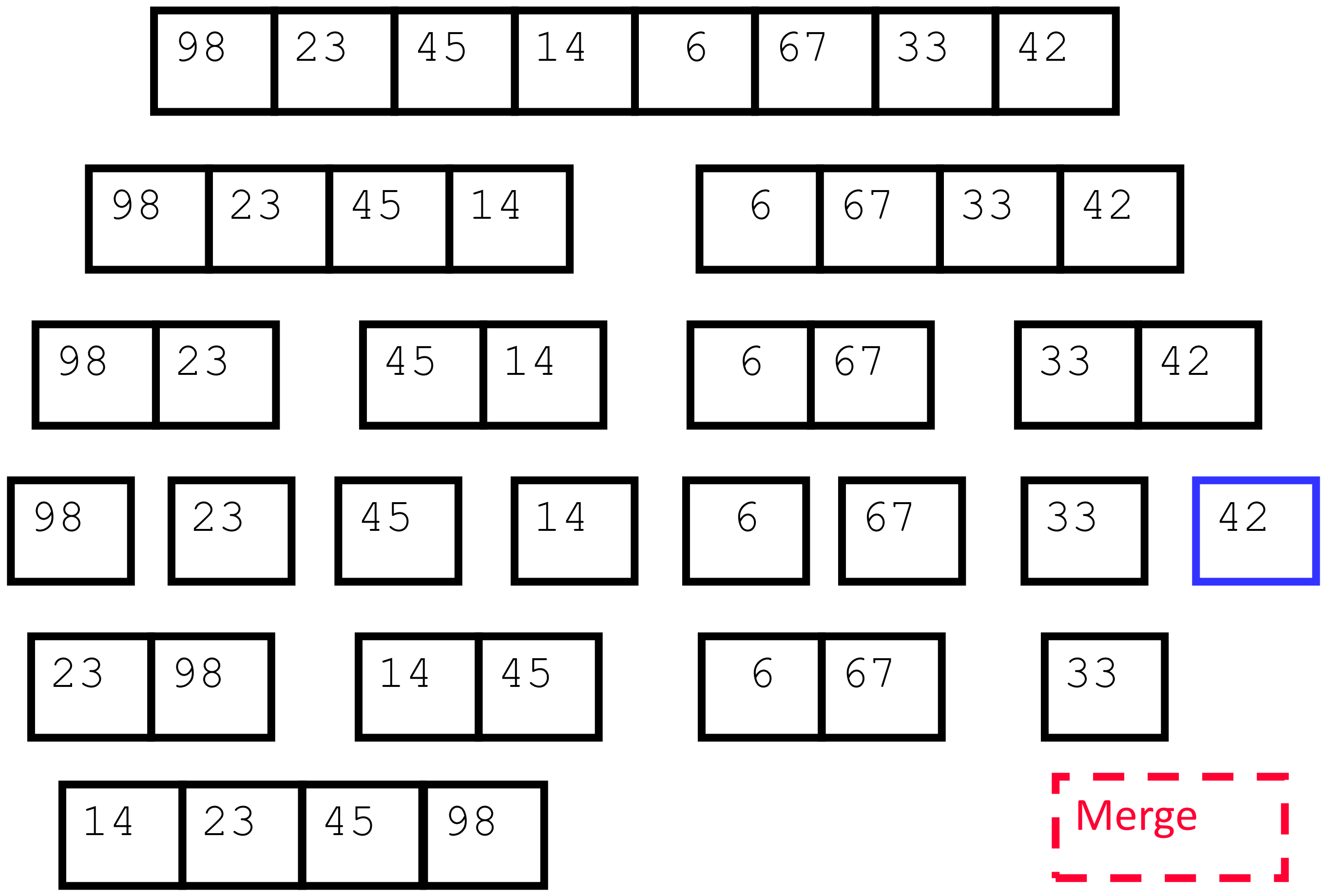

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
14 23 45 98
Merge

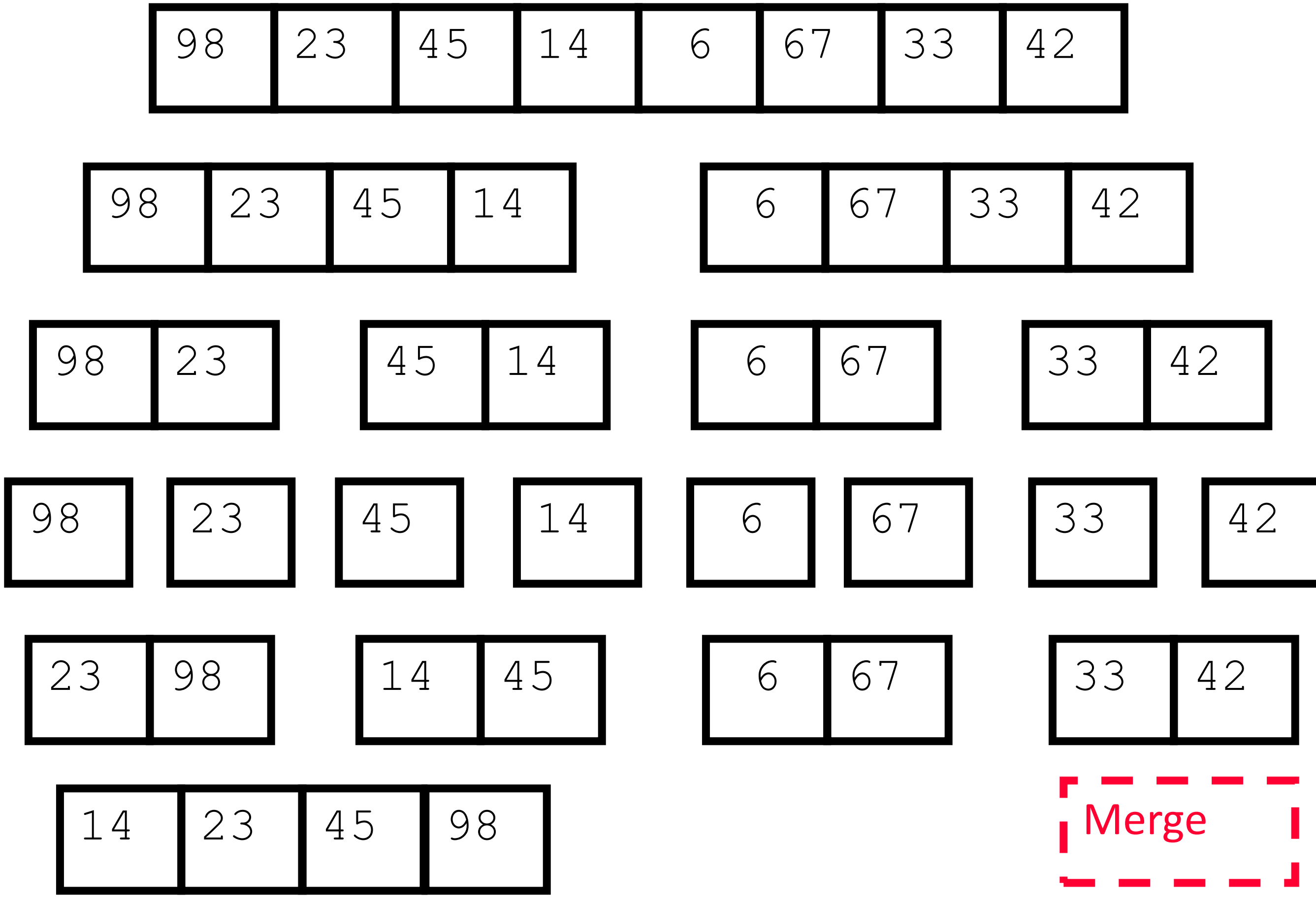

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
6
14 23 45 98
Merge

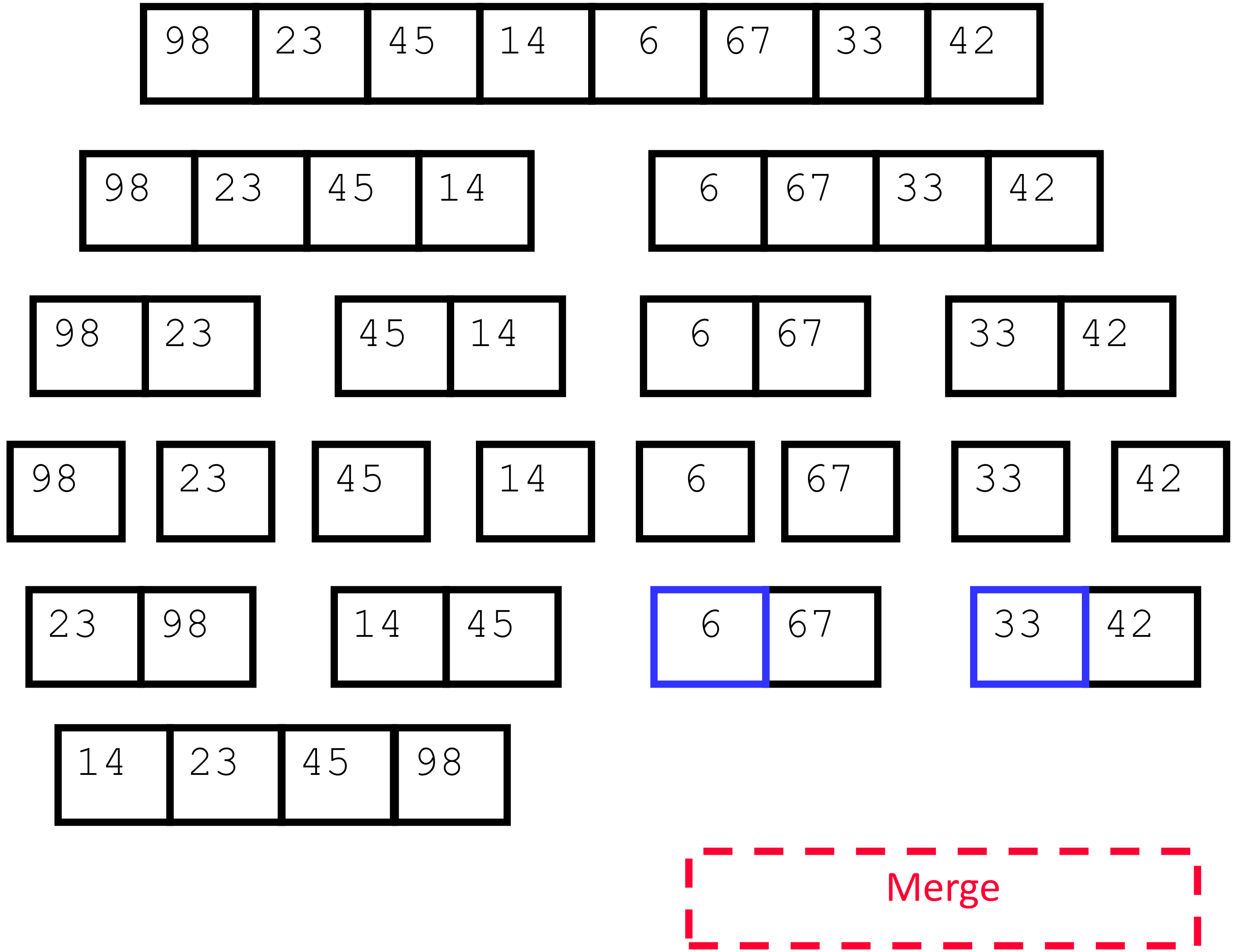

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
6 67
14 23 45 98
Merge

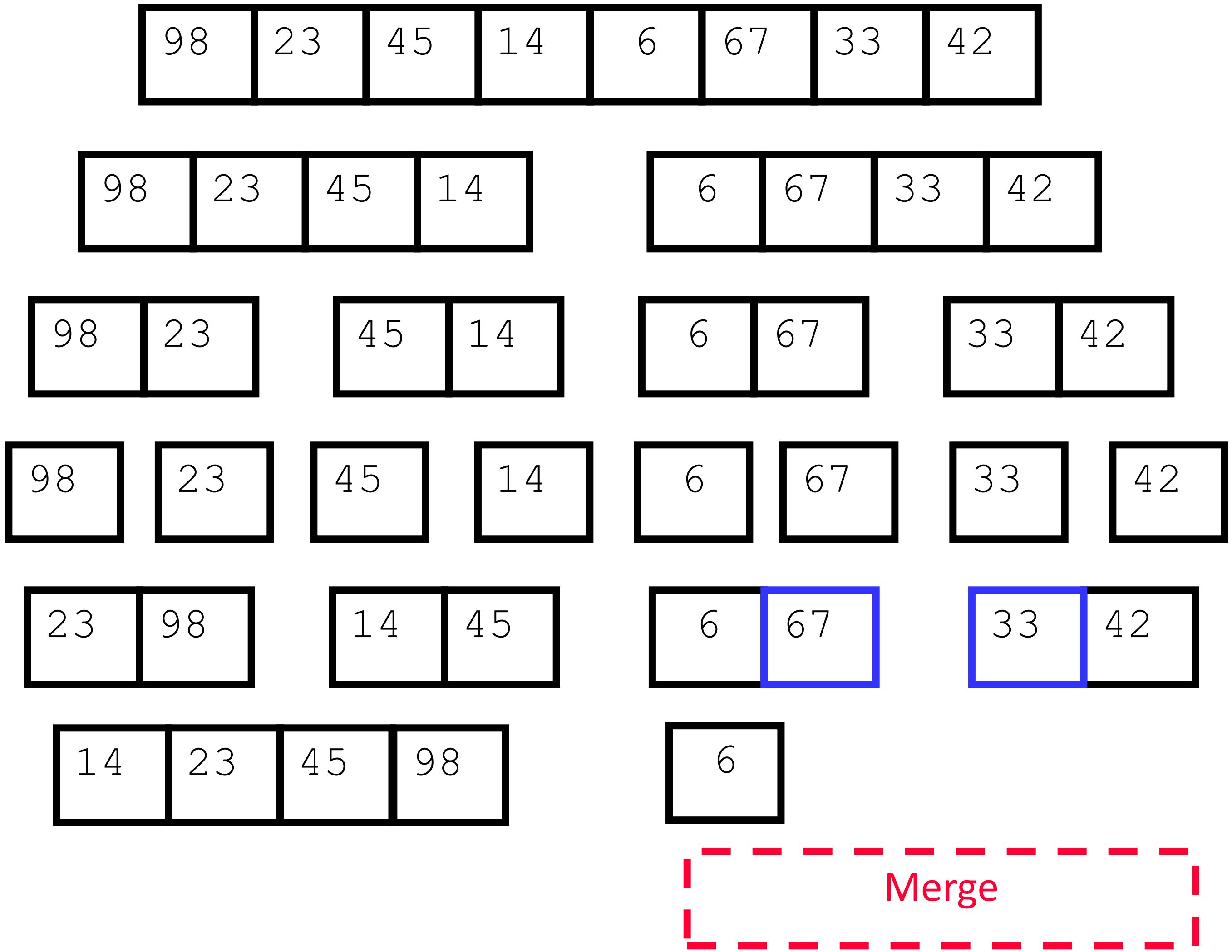

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
14 23 45 98

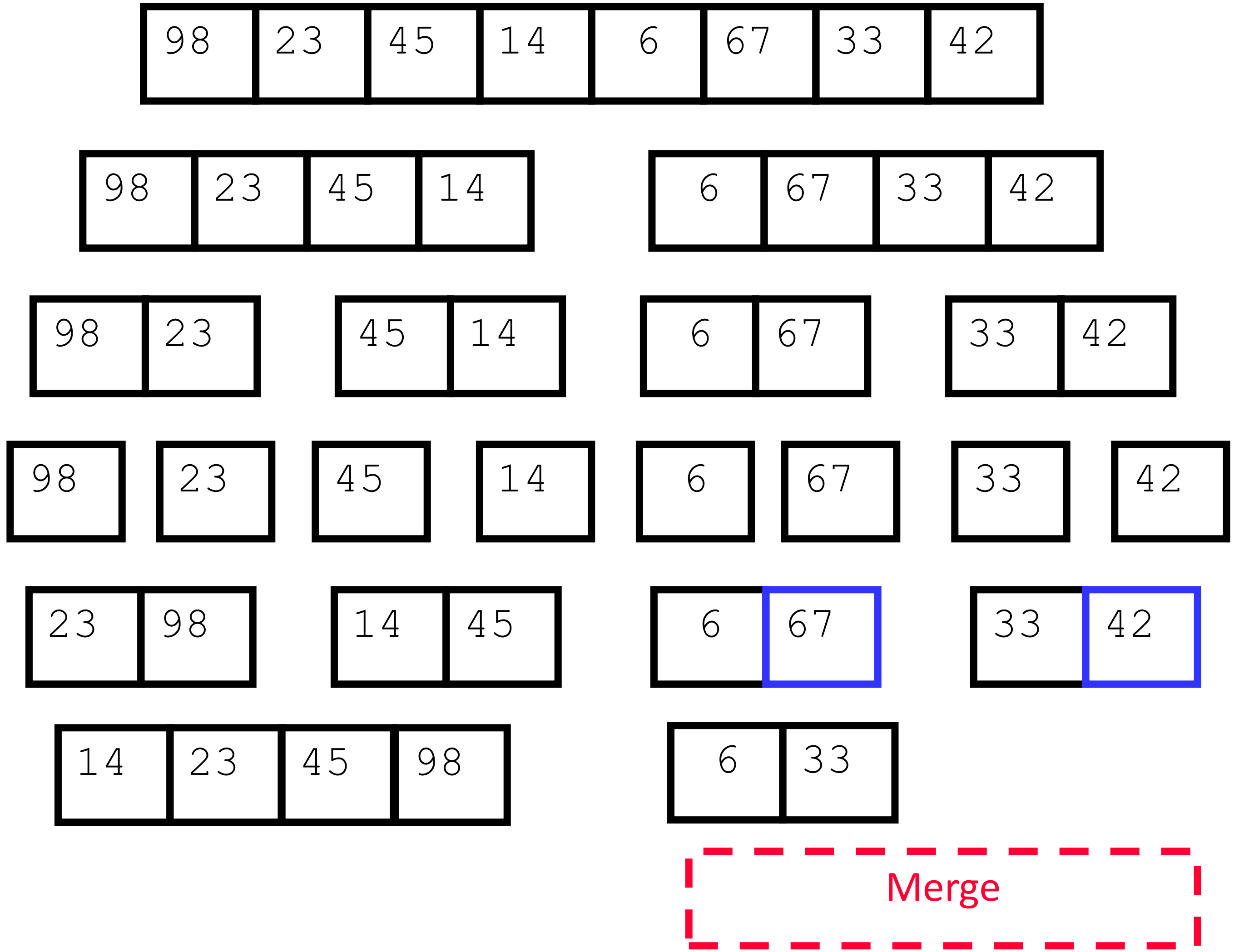

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
14 23 45 98
Merge

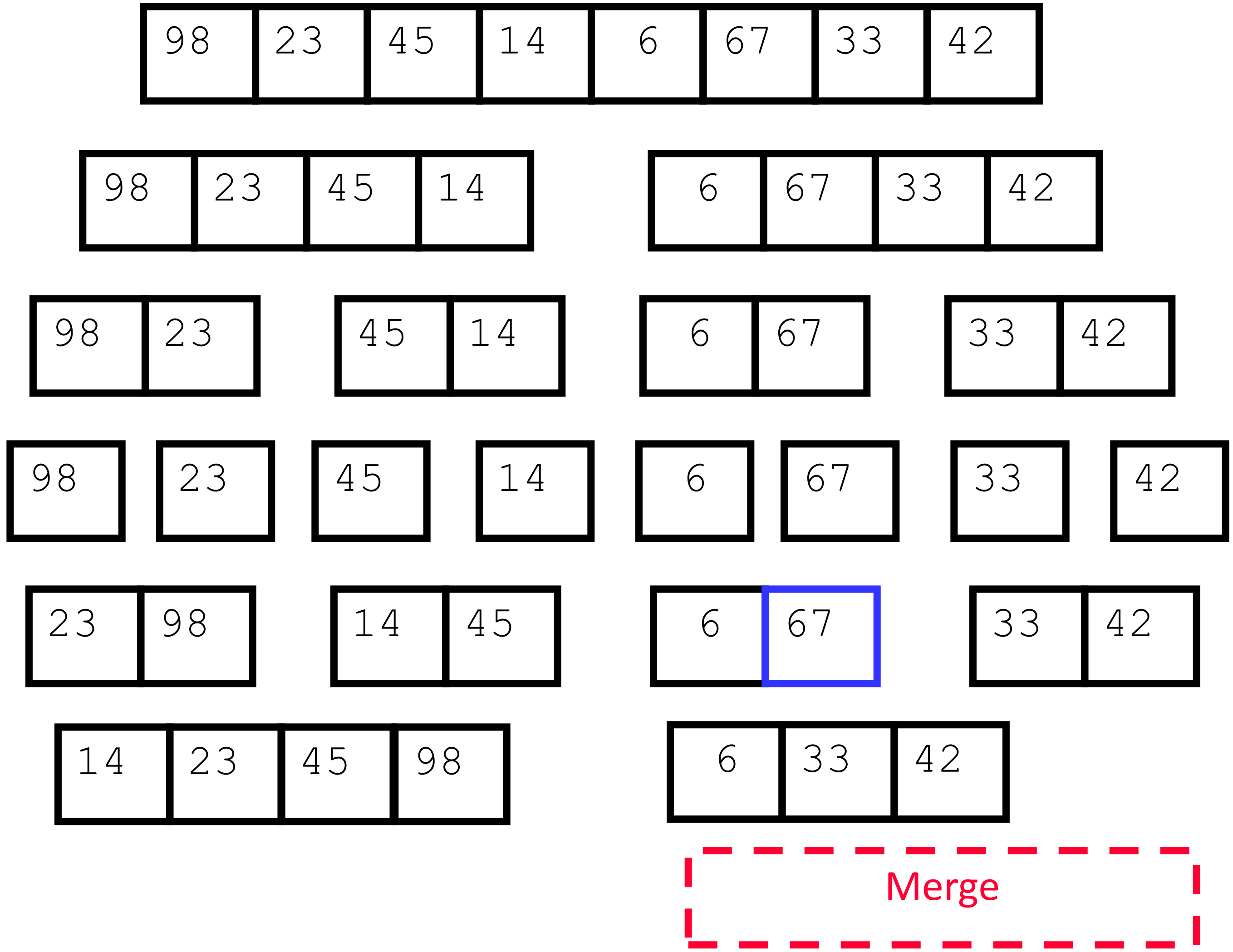

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33
14 23 45 98
Merge

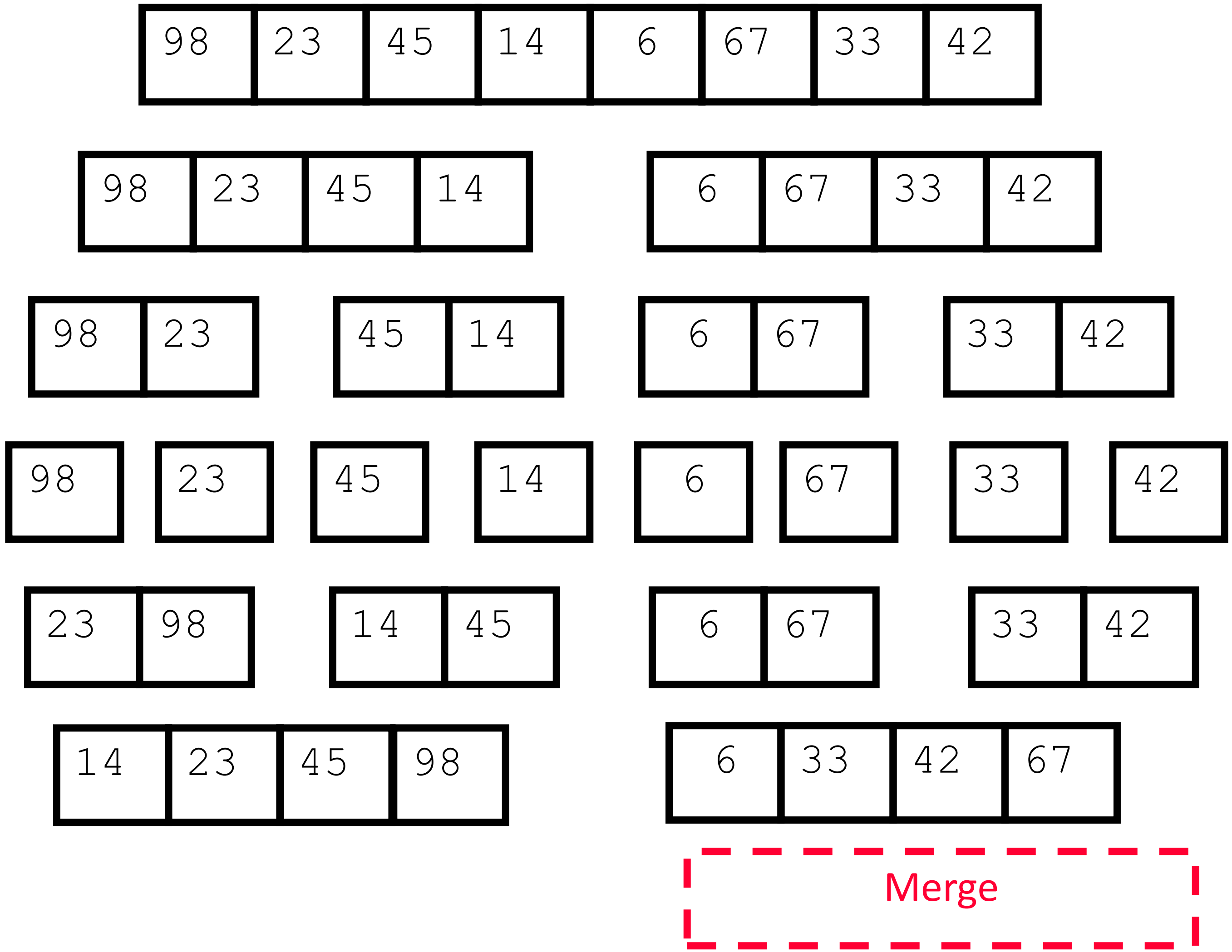

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
Merge


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

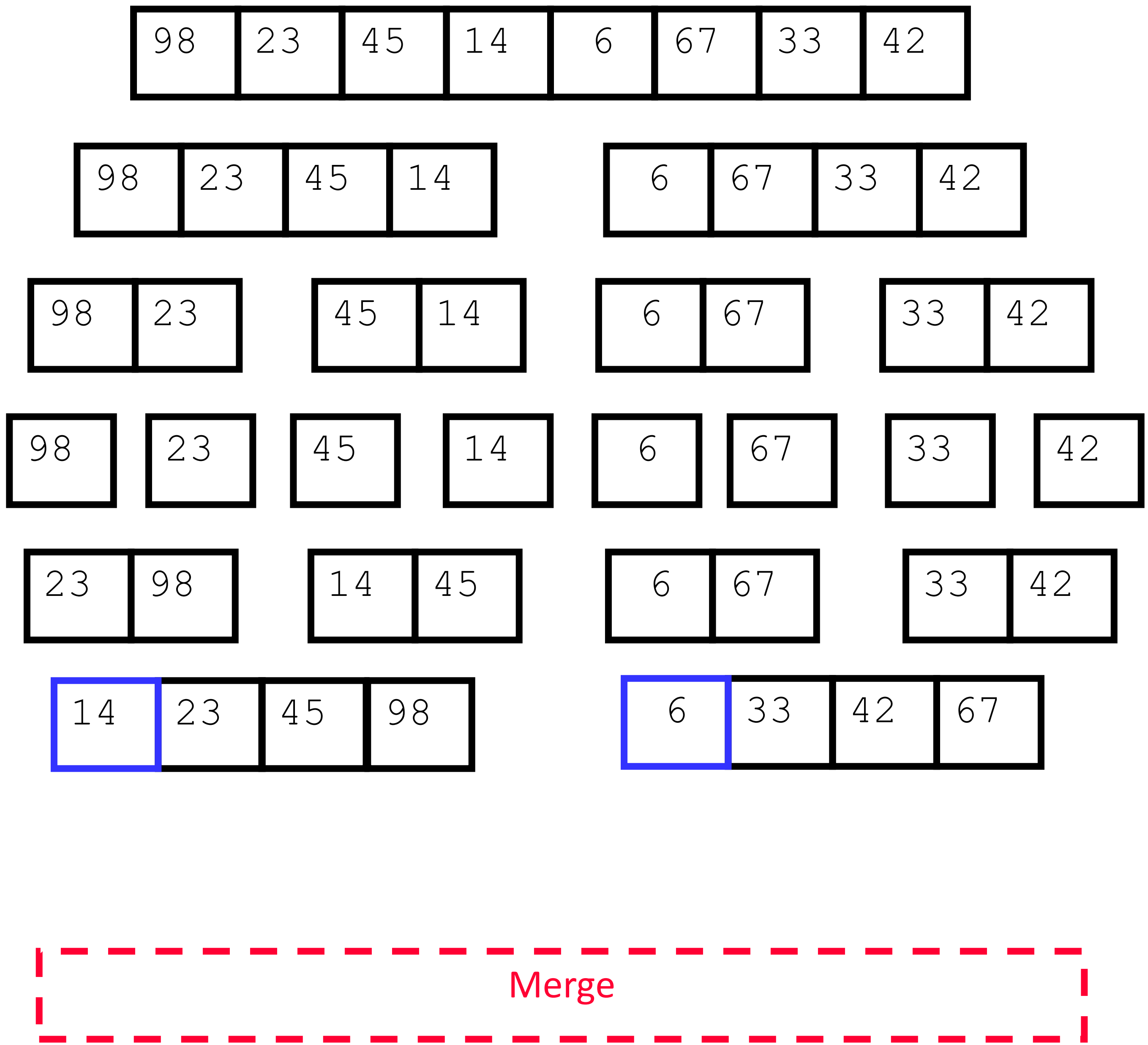

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6
Merge

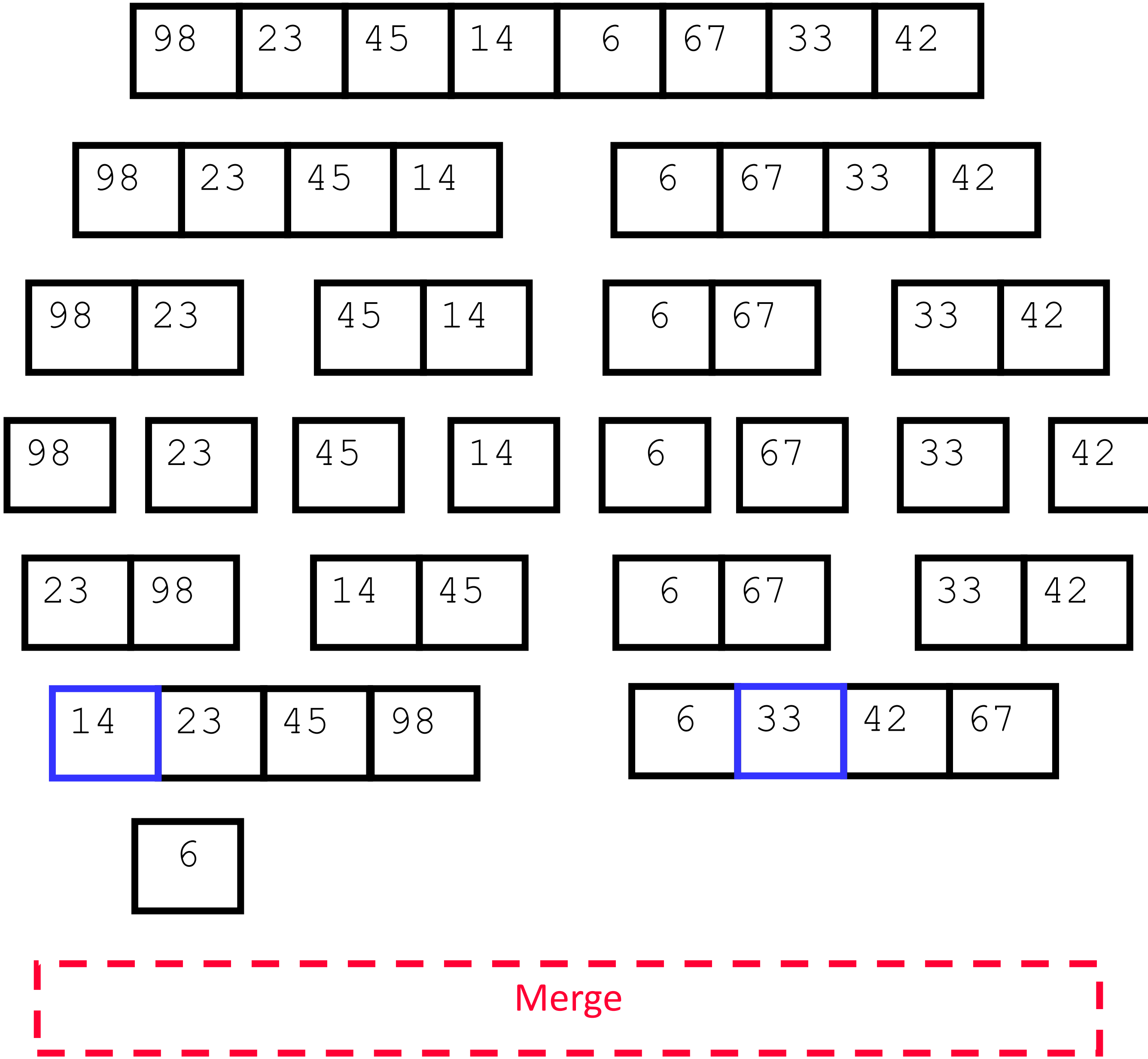

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33
Merge

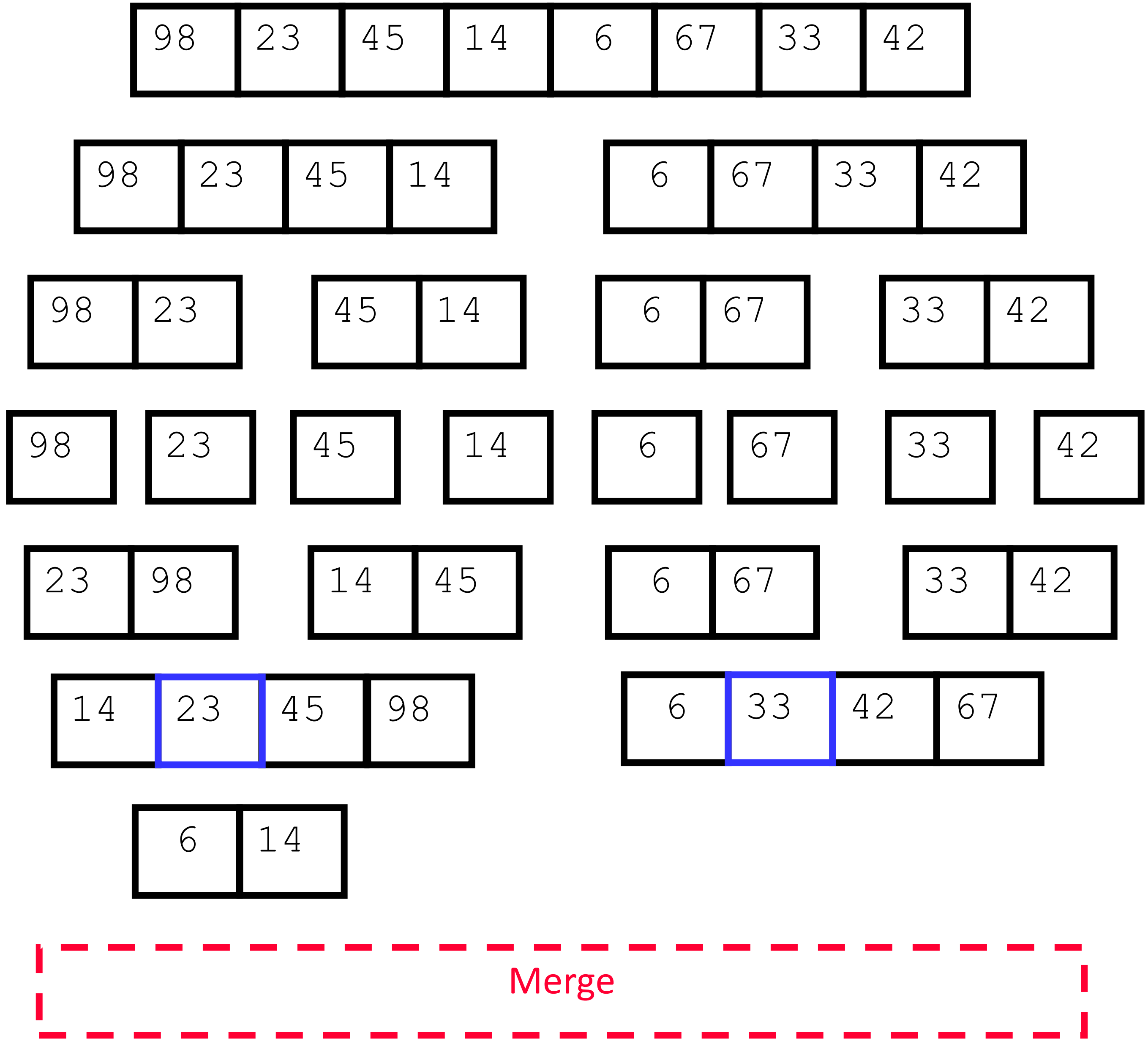

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42
Merge

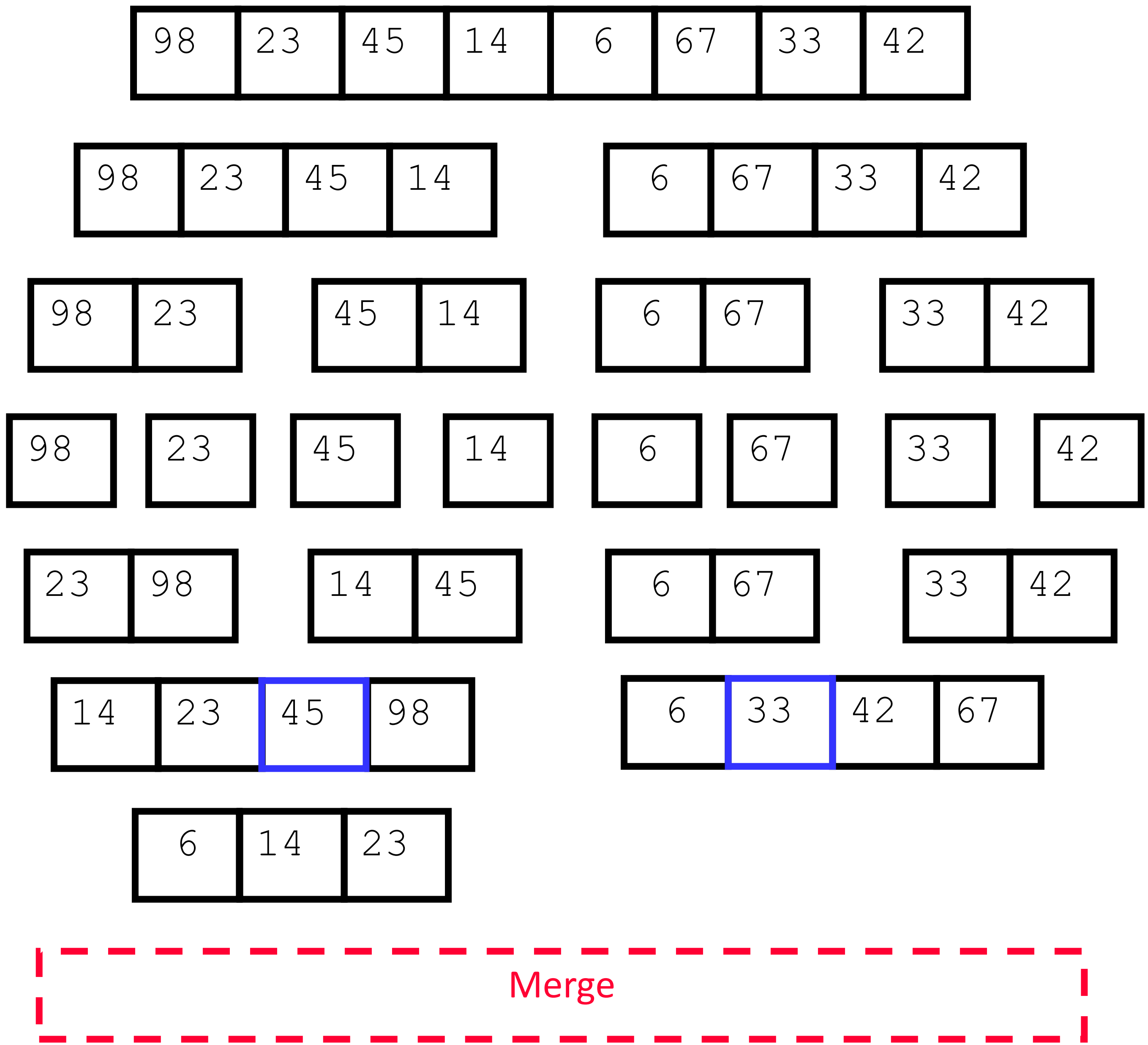


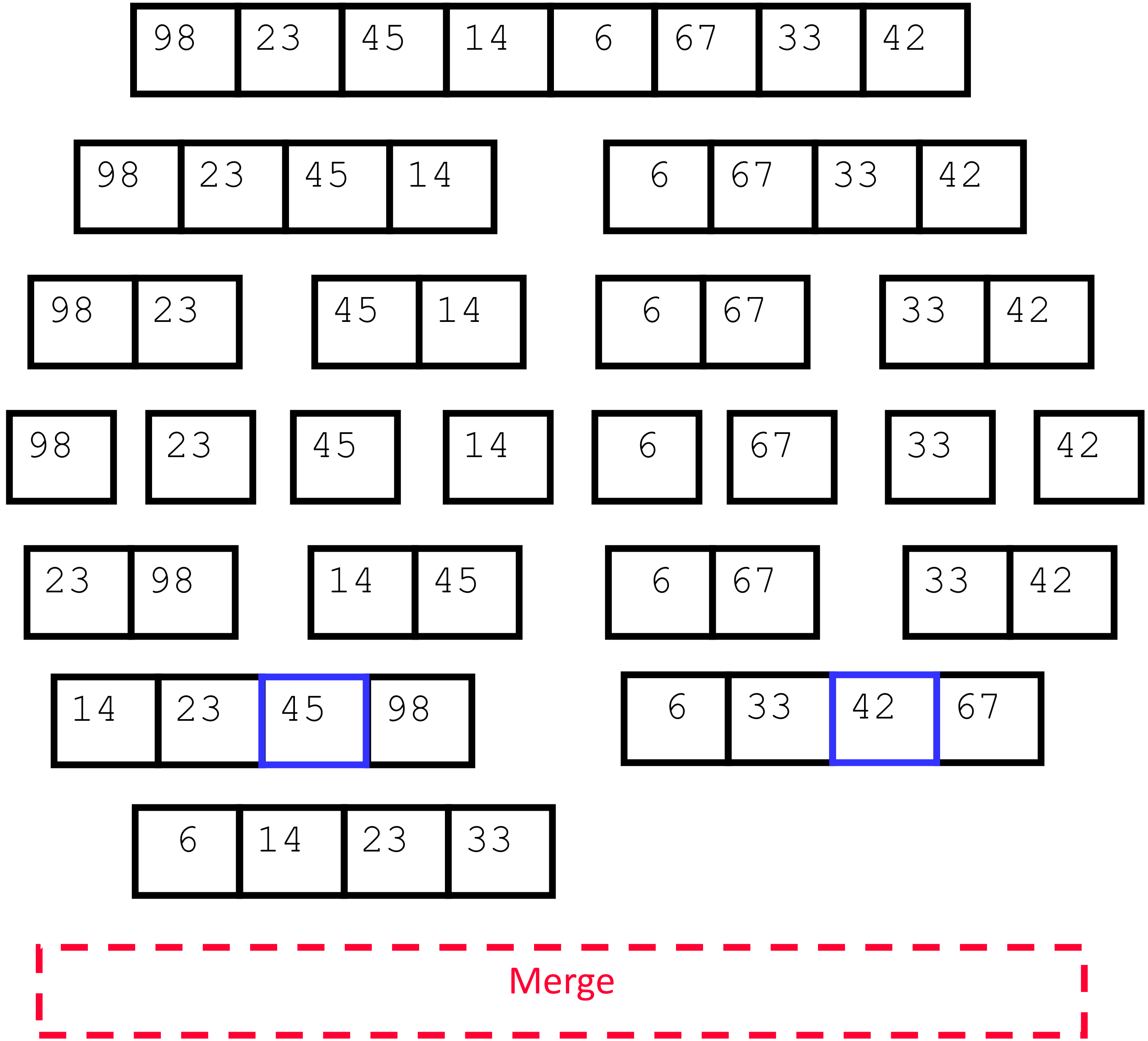
98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
Merge

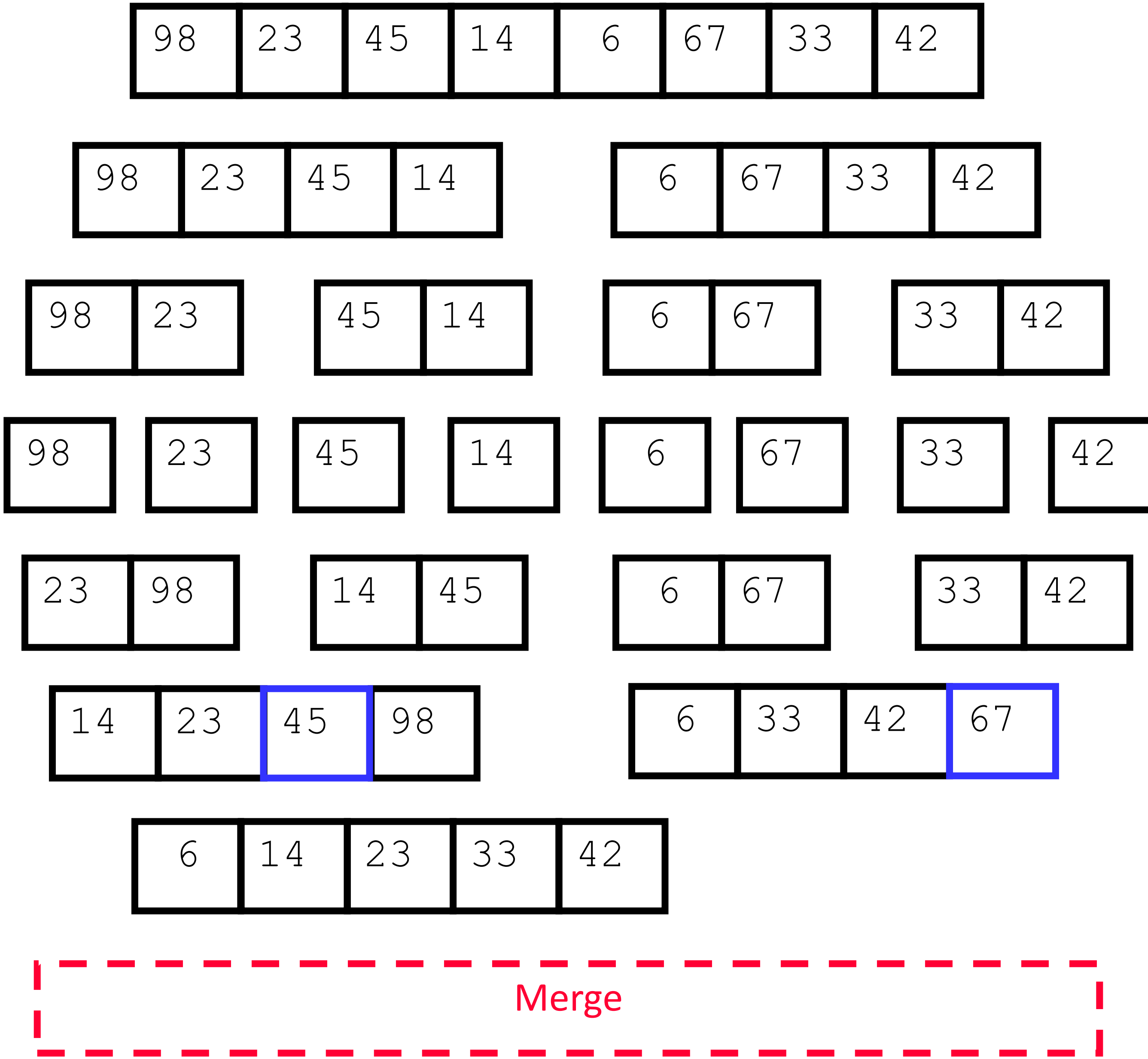

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
Merge

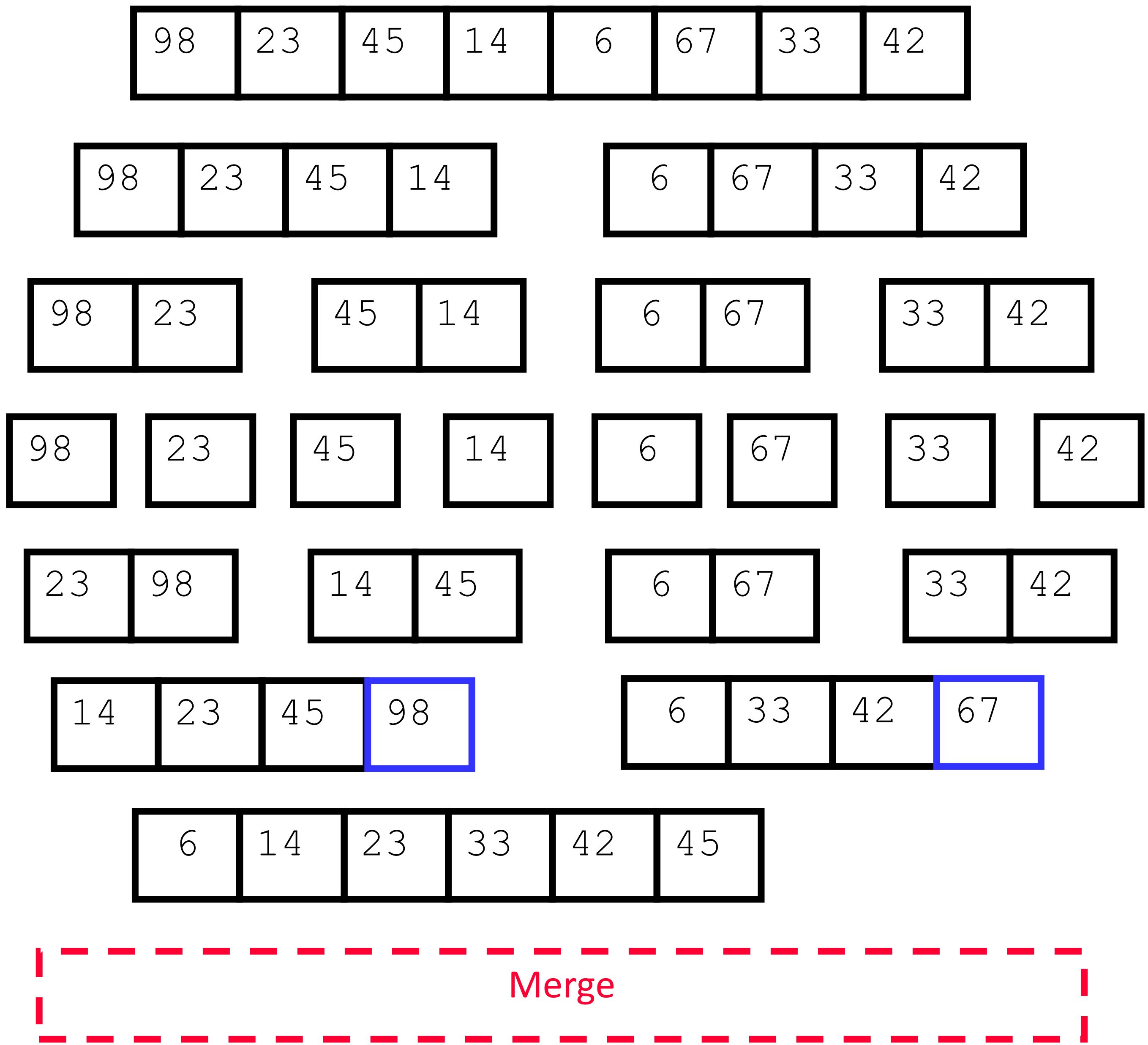

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6
Merge

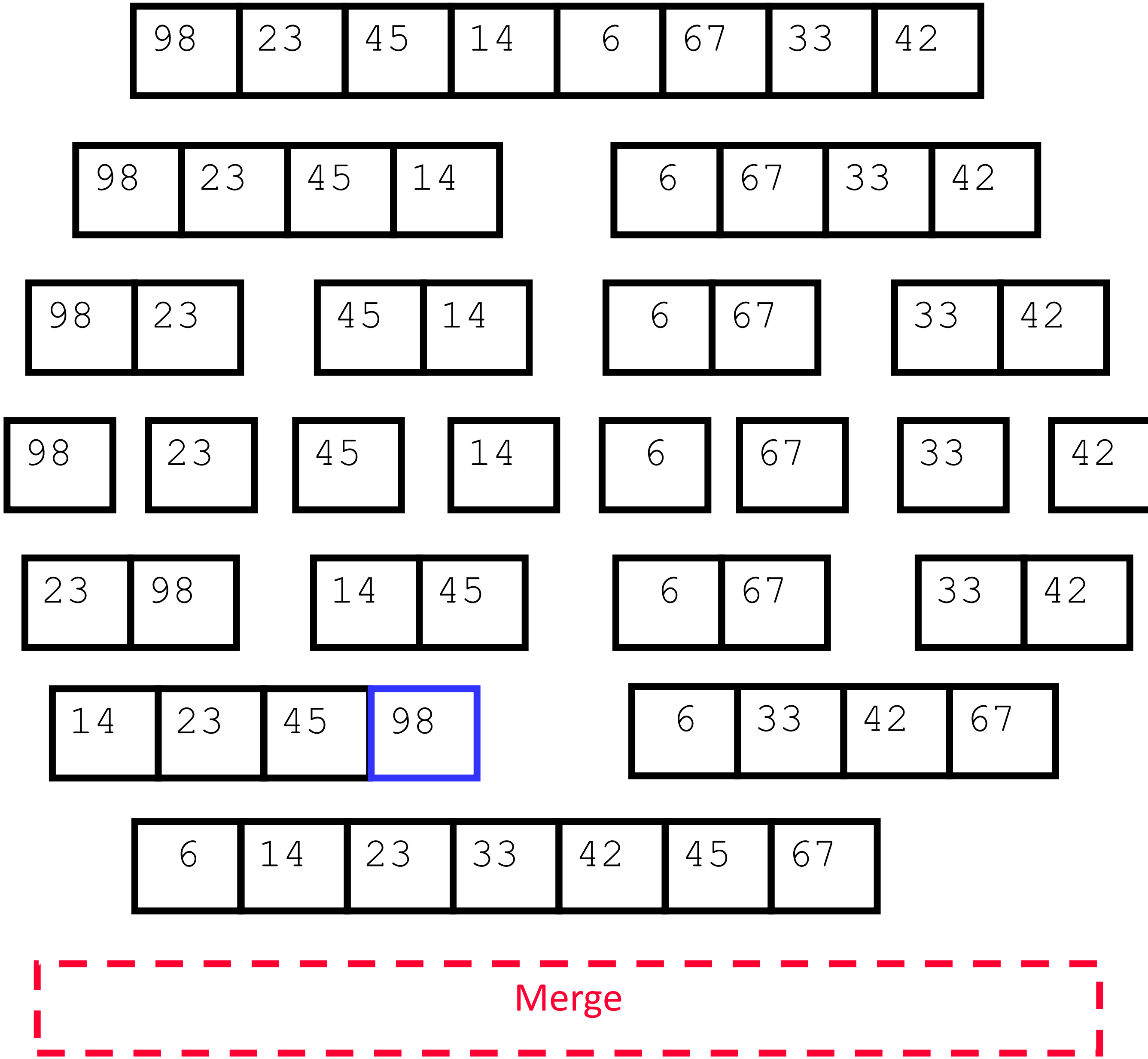

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14
Merge

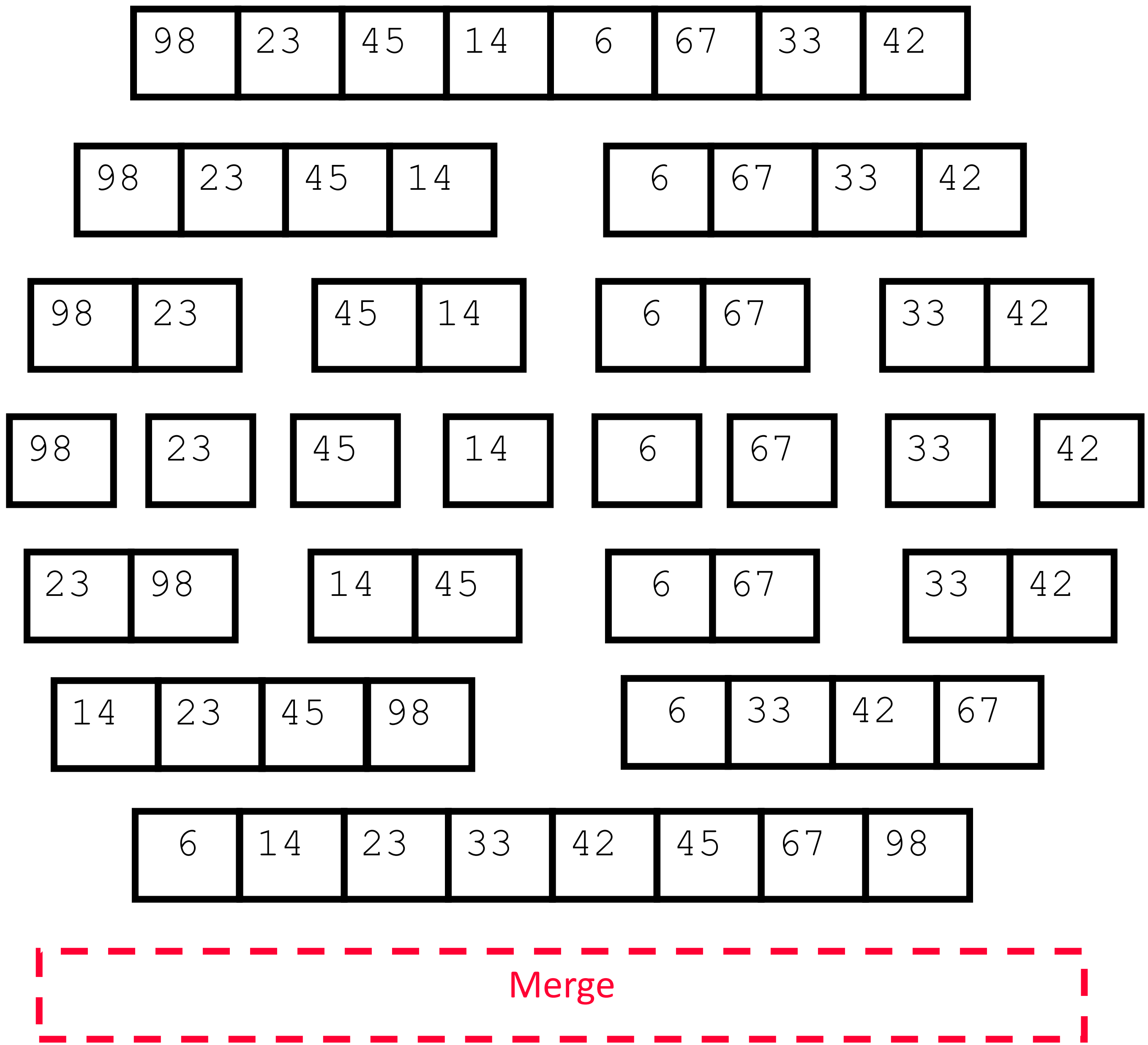

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23
Merge


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

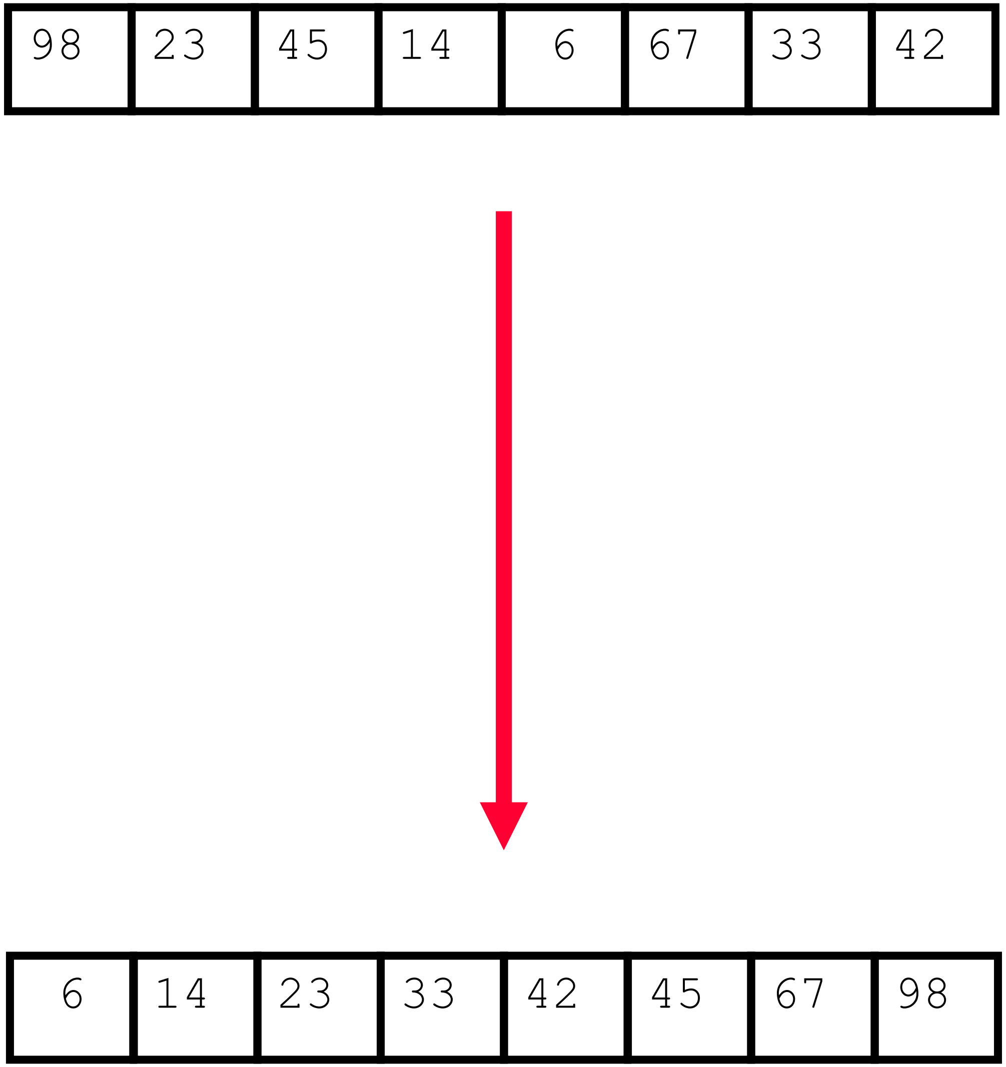

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42
Merge


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
6 14 23 33 42 45 67 98

Analysis of Merge Sort

Statement	Cost
MergeSort (A, p, r) //initial call: MergeSort(A,1,n)	**T(n) [let]**
if *p < r*	
then $q \leftarrow \lfloor (p+r)/2 \rfloor$	
MergeSort (A, p, q)	
MergeSort (A, q+1, r)	
Merge (A, p, q, r)	

Analysis of Merge Sort

Statement	Cost (time)
MergeSort (A, p, r) *//initial call: MergeSort(A,1,n)*	**T(n)**, to sort n elements
1 ***if*** $p < r$	$\Theta(1)$
2 ***then*** $q \leftarrow \lfloor (p+r)/2 \rfloor$ //q ≈ n/2	$\Theta(1)$
3 *MergeSort (A, p, q)*	**T(n/2)**, to sort n/2 elements
4 *MergeSort (A, q+1, r)*	**T(n/2)**, to sort n/2 elements
5 *Merge (A, p, q, r)*	$\Theta(n)$

So T(n) = $\Theta(1)$; when n = 1, and

$2T(n/2) + \Theta(n) + 2\Theta(1)$; when n > 1

It's a recurrence relation. Equivalent recurrence relation:

T(n) = $\Theta(1)$; when n = 1, and

$2T(n/2) + \Theta(n)$; when n > 1

Equivalent recurrence relation:

$$T(n) = c \quad \text{if } n = 1$$
$$= 2T(n/2) + cn \quad \text{if } n > 1$$

Recurrence Relations (RR)

Equation or an inequality that characterizes a function by its values on smaller inputs.

Recurrence relations arise when we analyze the running time of iterative or recursive algorithms.

Ex: Divide and Conquer algorithms typically have r.r. of the form:

$T(n) = \Theta(1)$	if $n \leq c$
$T(n) = a\,T(n/b) + D(n)$	otherwise

Mthods to solve recurrence relations

- Substitution Method.
- Recursion-tree Method.

Substitution Method

Illustration of guessing solution of a r.r. (representing time complexity of MergeSort) via substitution method:

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

$= 2(2T(n/4)+cn/2) + cn = 2^2T(n/2^2) + 2cn$

$= 2^2(2T(n/8)+cn/4) + 2cn = 2^3T(n/2^3) + 3cn$

…

$= 2^kT(n/2^k) + kcn$ *[guess the pattern from previous equations]*

Let $2^k = n$ (so that we get $T(n/2^k) = T(1)$ which is known to us)

$\therefore T(n) = n\,T(n/n) + (\lg n)\,cn$

$= n\,T(1) + (\lg n)\,cn$

$= n\,T(1) + cn \lg n$

$= cn + (\lg n)\,cn$ *which is* $\Theta(n \lg n)$

Recursion-tree Method

- **Recursion trees** can also be used to solve r.r.

Recursion Trees

- Show successive expansions of recurrences using trees.
- Keep track of the time spent on the subproblems of a divide and conquer algorithm.
- Help organize the algebraic bookkeeping necessary to solve a recurrence.

Recursion Tree – Example

Running time of Merge Sort:

$$T(n) = \Theta(1) \qquad \text{if } n = 1$$

$$T(n) = 2T(n/2) + \Theta(n) \qquad \text{if } n > 1$$

Rewrite the recurrence as

$$T(n) = c \qquad \text{if } n = 1$$

$$T(n) = 2T(n/2) + cn \qquad \text{if } n > 1$$

$c > 0$: Running time for the base case and time per array element for the divide and combine steps.

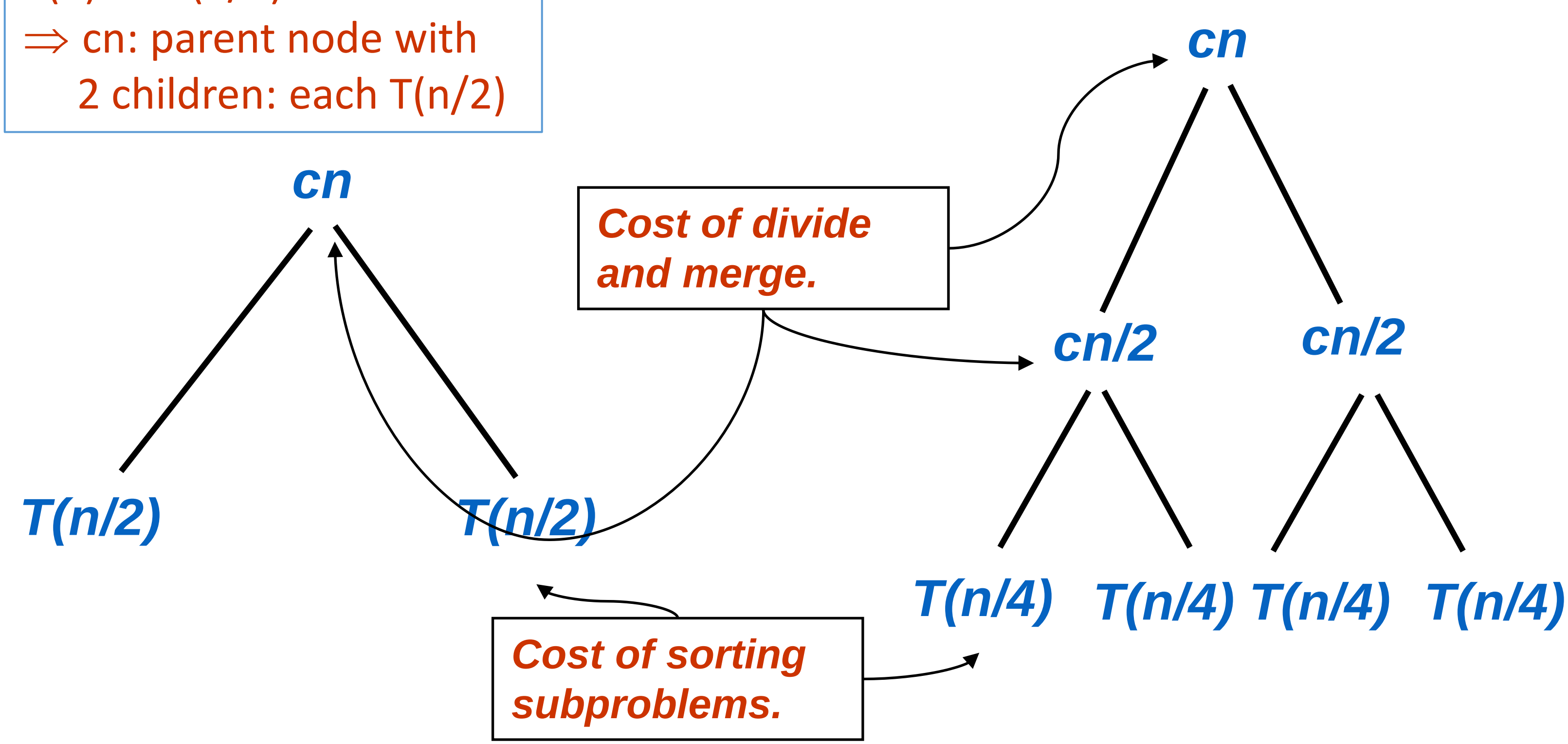
Recursion Tree for Merge Sort

For the original problem, we have a cost of cn, plus two subproblems each of size (n/2) and running time T(n/2).

$T(n) = 2T(n/2) + cn$
$\Rightarrow$ cn: parent node with
2 children: each T(n/2)

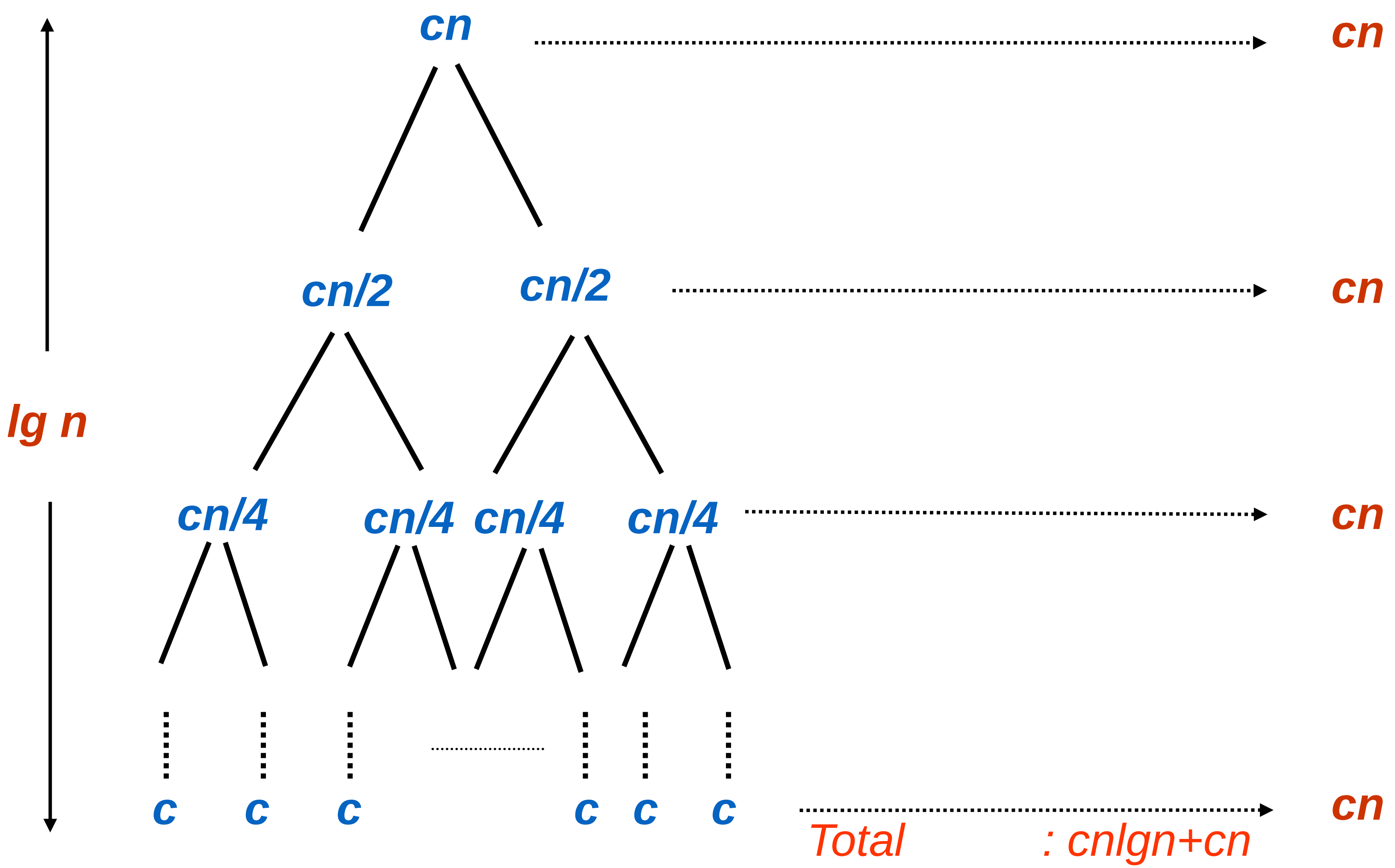
Each of the size n/2 problems has a cost of cn/2 plus two subproblems, each costing T(n/4).

$T(n/2) = 2T(n/4) + cn/2$
$\Rightarrow$ cn/2: parent node with
2 children: each T(n/4)



Recursion Tree for Merge Sort

Continue expanding until the problem size reduces to 1.



Counting Inversions Problem

- Given two ranked list of items, how can you compare these two lists?
- **Application**: Recommendation systems try to match your preferences (for books, movies, restaurants, etc.) with those of other people in the internet
- Idea: represent one ranked list by <1,2, …, n> and another by a permutation of the first list. Then count the number of inversions (i.e. out-of-order pairs in the second list.

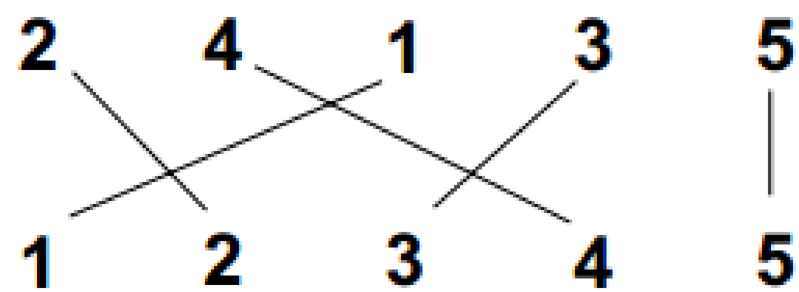


Figure 5.4: Counting the number of inversions in the sequence 2, 4, 1, 3, 5. Each crossing pair of line segments corresponds to one pair that is in the opposite order in the input list and the ascending list — in other words, an inversion.

Merging & Counting Inversions

```
MergeAndCount(A, p, q, r)
n1 ← q – p + 1
n2 ← r – q
     for i ← 1 to n1
        do L[i] ← A[p + i – 1]
     for j ← 1 to n2
        do R[j] ← A[q + j]
     L[n1+1] ← ∞
     R[n2+1] ← ∞
     i ← 1
     j ← 1
     cnt ← 0
     for k ←p to r
        do if L[i] ≤ R[j]
           then A[k] ← L[i]
                i ← i + 1
           else A[k] ← R[j]
                j ← j + 1
                cnt ← cnt + n1-i+1
     return cnt
```

Input: Array containing sorted subarrays A[p..q] and A[q+1..r].

Output: Merged sorted subarray in A[p..r].

Counting Inversions

Statement Cost

```
CountInversions(A, p, r)
    if p < r
        then q ← ⌊(p+r)/2⌋
            x ← CountInversions (A, p, q)
            y ← CountInversions(A, q+1, r)
            z ← MergeAndCount(A, p, q, r)
    return x+y+z
```

So T(n) = $\Theta(1)$ when n = 1, and

$2T(n/2) + \Theta(n)$ when n > 1