Analysis of Sorting Algorithms

Lec 2, CMSC 142

Sorting

- **Input:** array of items, **Output**: sorted array
- Given an array of items, arrange them according to a specified order
- **Out-of-place sorting** - uses extra data structure / memory in sorting
- **In-place sorting** - sorting on the input array itself, using swaps; no extra memory needed

Note:

- The pseudocodes are 1-indexed (array index starts at 1, not 0).
- Let N = length of array.
- Memory in analysis → extra memory (e.g. array, data structure) needed by algorithm;

Insertion Sort

Insertion Sort

- **In-place** sorting algorithm that inserts items into their rightful positions
- **Analogy:** Similar to sorting a hand of playing cards - remove one card at a time from the table and insert it into correct position by comparing it with each of the cards already in hand
- **Idea:** assume that the items to the left of current item are already sorted; add the current item to the sorted items by looking for the right position to insert it in
- Efficient for sorting small number of elements and when array is already nearly sorted

Insertion Sort

```
insertion_sort(array A):
    for i = 1 to N:
        sorted_items = items before A[i]
        shift items higher than A[i] in sorted_items one place to the right
        insert A[i] into its correct position in sorted_items
```

Insertion Sort

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]
        left = i – 1
        while left > 0 and A[left ] > item :
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

1	2	3	4	5	6	7	8
24	13	9	64	7	23	34	47

Insertion Sort

i=0

```
insertion_sort(array A):
    for $i = 1$ to $N$:
        $item = A[i]$    24
        $left = i – 1$    0
        while $left > 0$ and $A[left ] > item$ :    F
            $A[left + 1] = A[left ]$
            $left = left – 1$
    $A[left + 1] = item$    A[1]=24
```

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
	24	13	9	64	7	23	34	47
	item							

Insertion Sort

$$24 > 13$$

i=2

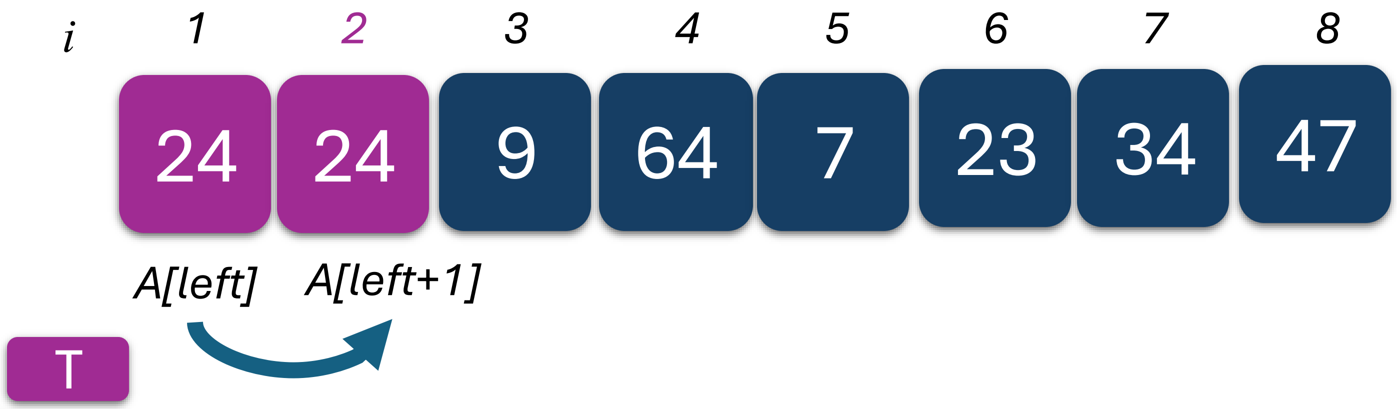
```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                              13
        left = i – 1                             1
        while left > 0 and A[left ] > item :     T
            A[left + 1] = A[left]
            left = left – 1
    A[left + 1] = item
```

i	1	2	3	4	5	6	7	8
	24	13	9	64	7	23	34	47
	A[left]	item						

Insertion Sort

$24 > 13$

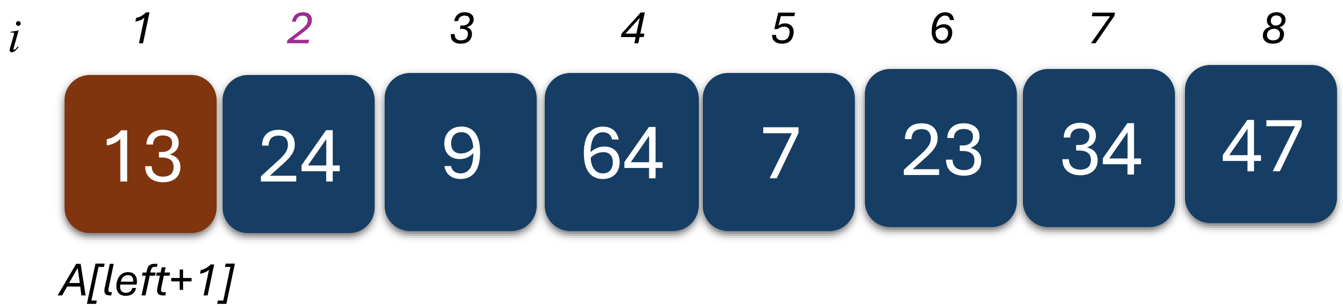
```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]          13
        left = i – 1         1
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left]      A[2] = A[1]
            left = left – 1
    A[left + 1] = item
```



i	1	2	3	4	5	6	7	8
	24	24	9	64	7	23	34	47
	A[left]	A[left+1]						

Put 13 back!

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                                  13
        left = i – 1
        while left > 0 and A[left ] > item :         F
            A[left + 1] = A[left]
            left = left – 1                          0
    A[left + 1] = item                               A[1] = 13
```



i	1	2	3	4	5	6	7	8
	13	24	9	64	7	23	34	47
	A[left+1]							

Insertion Sort

$24 > 9$

i=3

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]        9
        left = i – 1       2
        while left > 0 and A[left ] > item :   T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

i	1	2	3	4	5	6	7	8
	13	24	9	64	7	23	34	47
		A[left]	item					

Insertion Sort

$24 > 9$

i=3

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                              9
        left = i – 1                             2
        while left > 0 and A[left ] > item :     T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

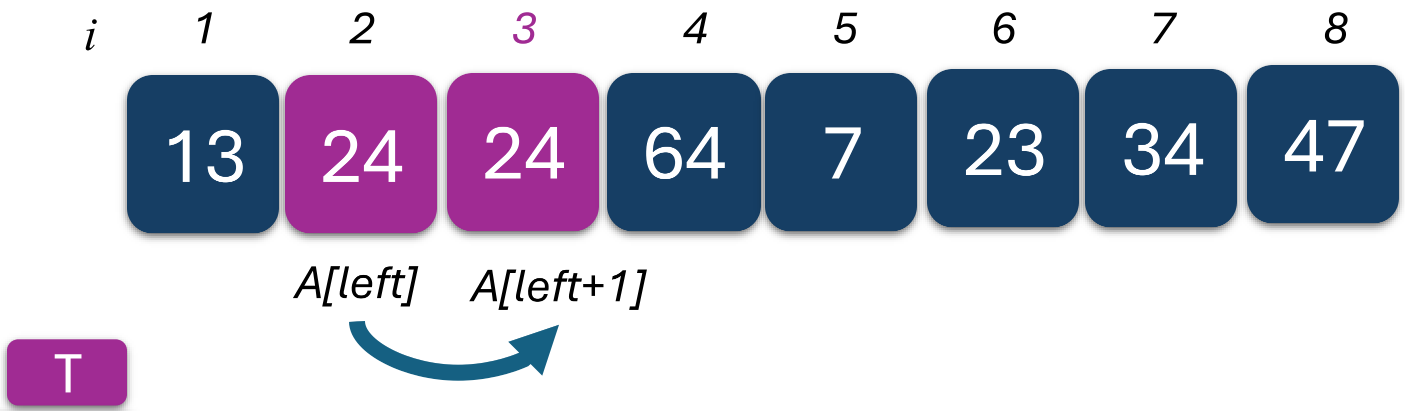
i	1	2	3	4	5	6	7	8
	13	24	9	64	7	23	34	47
		A[left]	item					

Insertion Sort

$24 > 9$

i=3

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                                9
        left = i – 1                               2
        while left > 0 and A[left ] > item :       T
            A[left + 1] = A[left ]                 A[3] = A[2]
            left = left – 1
    A[left + 1] = item
```



i	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
	13	24	24	64	7	23	34	47

Insertion Sort

13 > 9

i=3

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]   9
        left = i – 1
        while left > 0 and A[left ] > item :   T
            A[left + 1] = A[left ]
            left = left – 1   1
    A[left + 1] = item
```

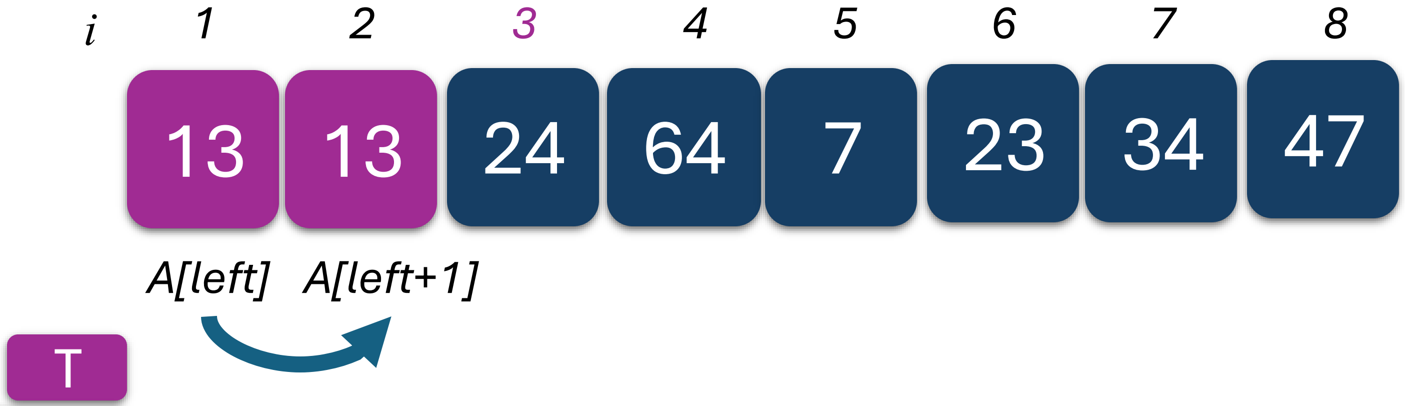
i	1	2	3	4	5	6	7	8
	13	24	24	64	7	23	34	47
	A[left]							

Insertion Sort

$13 > 9$

i=3

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                              9
        left = i – 1
        while left > 0 and A[left ] > item :     T
            A[left + 1] = A[left ]               A[2] = A[1]
            left = left – 1
    A[left + 1] = item
```



i	1	2	3	4	5	6	7	8
	13	13	24	64	7	23	34	47

Insert 9

i=3

```
insertion_sort(array A):
    for $i = 1$ to $N$:
        $item = A[i]$   9
        $left = i – 1$
        while $left > 0$ and $A[left ] > item$ :   F
            $A[left + 1] = A[left ]$
            $left = left – 1$   0
    $A[left + 1] = item$   A[1] = 9
```

i	1	2	3	4	5	6	7	8
	9	13	24	64	7	23	34	47

Insertion Sort

$24 < 64$

i=4

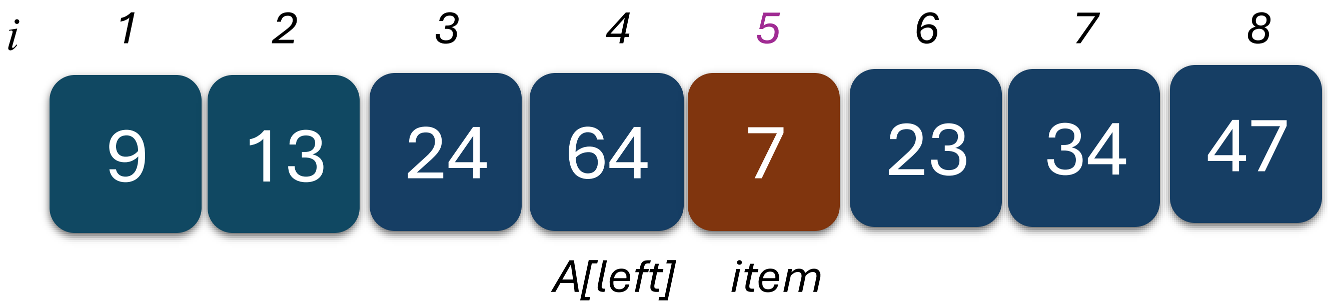
```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]          64
        left = i – 1         3
        while left > 0 and A[left ] > item :     F
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item       A[4] = 64
```

i	1	2	3	4	5	6	7	8
	9	13	24	64	7	23	34	47
			A[left]	item				

Insertion Sort

i=5

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]    7
        left = i – 1    4
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```



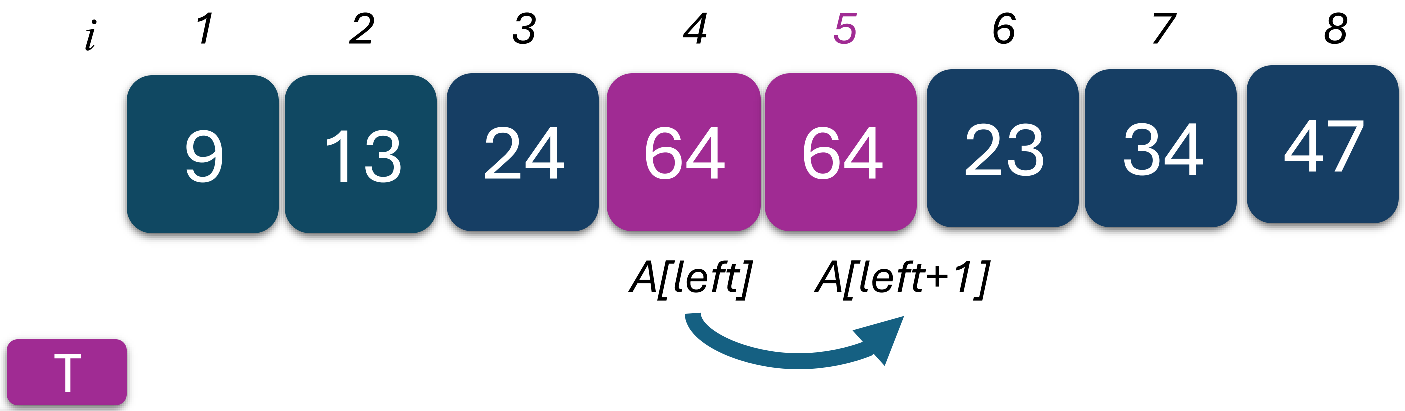
i	1	2	3	4	5	6	7	8
	9	13	24	64	7	23	34	47
				A[left]	item			

Insertion Sort

64>7

i=5

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                          7
        left = i – 1                         4
        while left > 0 and A[left ] > item :   T
            A[left + 1] = A[left ]           A[5] = A[4]
            left = left – 1
    A[left + 1] = item
```



i	1	2	3	4	5	6	7	8
	9	13	24	64	64	23	34	47

Insertion Sort

$24>7$

i=5

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]   7
        left = i – 1
        while left > 0 and A[left ] > item :   T
            A[left + 1] = A[left ]
            left = left – 1   3
    A[left + 1] = item
```

i	1	2	3	4	5	6	7	8
	9	13	24	64	64	23	34	47
			A[left]					

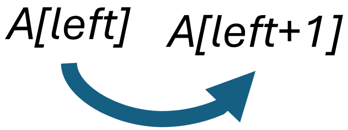
Insertion Sort

$24>7$

i=5

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                          7
        left = i – 1
        while left > 0 and A[left ] > item :   T
            A[left + 1] = A[left ]           A[4] = A[3]
            left = left – 1
    A[left + 1] = item
```

i	1	2	3	4	5	6	7	8
	9	13	24	24	64	23	34	47



Insertion Sort

$13>7$

i=5

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]    7
        left = i – 1
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]
            left = left – 1    2
    A[left + 1] = item
```

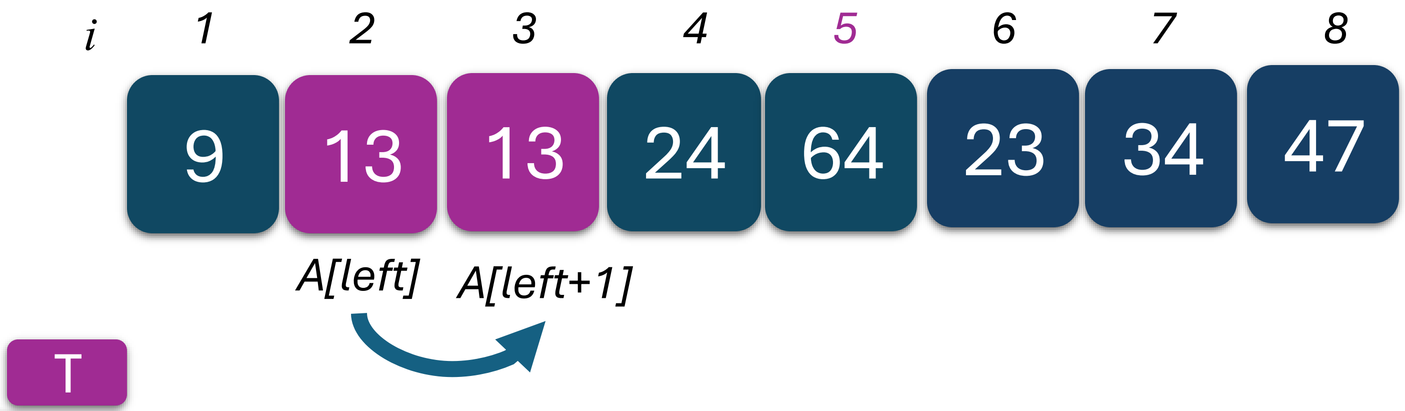
i	1	2	3	4	5	6	7	8
	9	13	24	24	64	23	34	47
		A[left]						

Insertion Sort

$13>7$

i=5

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                             7
        left = i – 1
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]              A[3] = A[2]
            left = left – 1
    A[left + 1] = item
```



i	1	2	3	4	5	6	7	8
	9	13	13	24	64	23	34	47

Insertion Sort

9>7

i=5

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]   7
        left = i – 1
        while left > 0 and A[left ] > item :   T
            A[left + 1] = A[left ]
            left = left – 1   1
    A[left + 1] = item
```

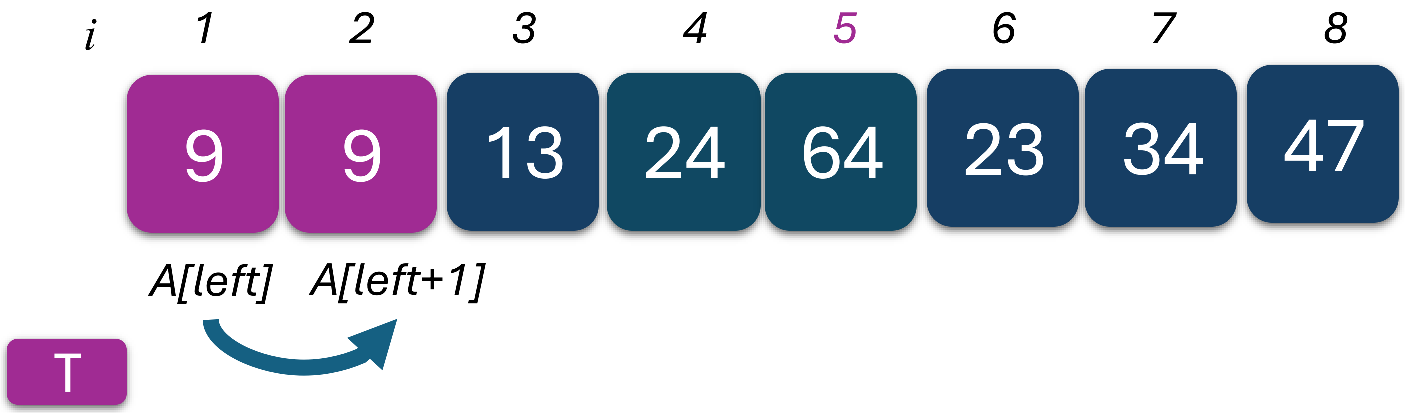
i	1	2	3	4	5	6	7	8
	9	13	13	24	64	23	34	47
	A[left]							

Insertion Sort

$9>7$

i=5

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]   7
        left = i – 1
        while left > 0 and A[left ] > item :   T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```



i	1	2	3	4	5	6	7	8
	9	9	13	24	64	23	34	47
	A[left]	A[left+1]						

Insertion Sort

Insert 7

i	1	2	3	4	5	6	7	8
	7	9	13	24	64	23	34	47

i=5

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                           7
        left = i – 1
        while left > 0 and A[left ] > item :    F
            A[left + 1] = A[left ]
            left = left – 1                   0
    A[left + 1] = item                        A[1] =7
```

Insertion Sort

Item 23

i=6

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]    23
        left = i – 1    5
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

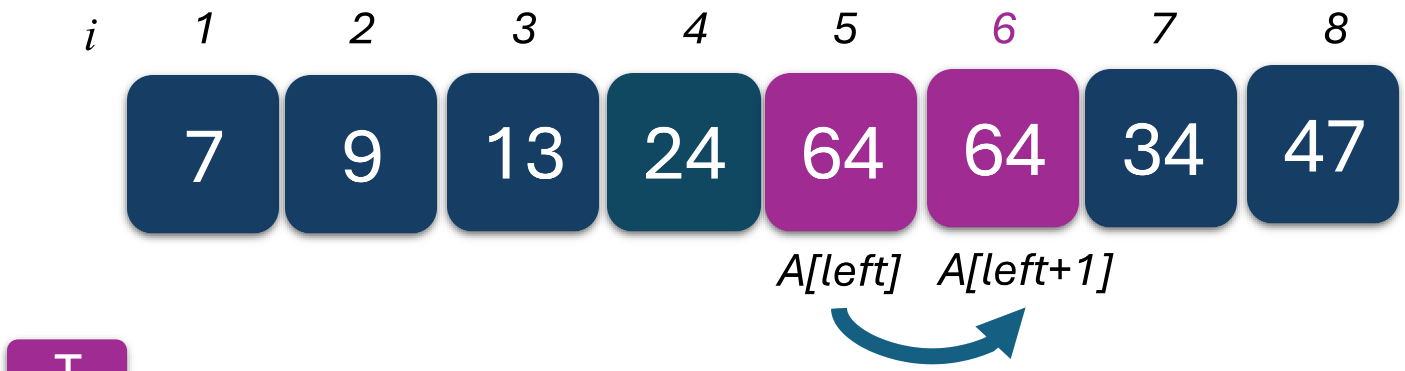
i	1	2	3	4	5	6	7	8
	7	9	13	24	64	23	34	47
					A[left]	item		

Insertion Sort

$64>23$

i=6

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                              23
        left = i – 1
        while left > 0 and A[left ] > item :     T
            A[left + 1] = A[left ]               A[6] =A[5]
            left = left – 1
    A[left + 1] = item
```



i	1	2	3	4	5	6	7	8
	7	9	13	24	64	64	34	47

Insertion Sort

24>23

i=6

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]    23
        left = i – 1
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]
            left = left – 1    4
    A[left + 1] = item
```

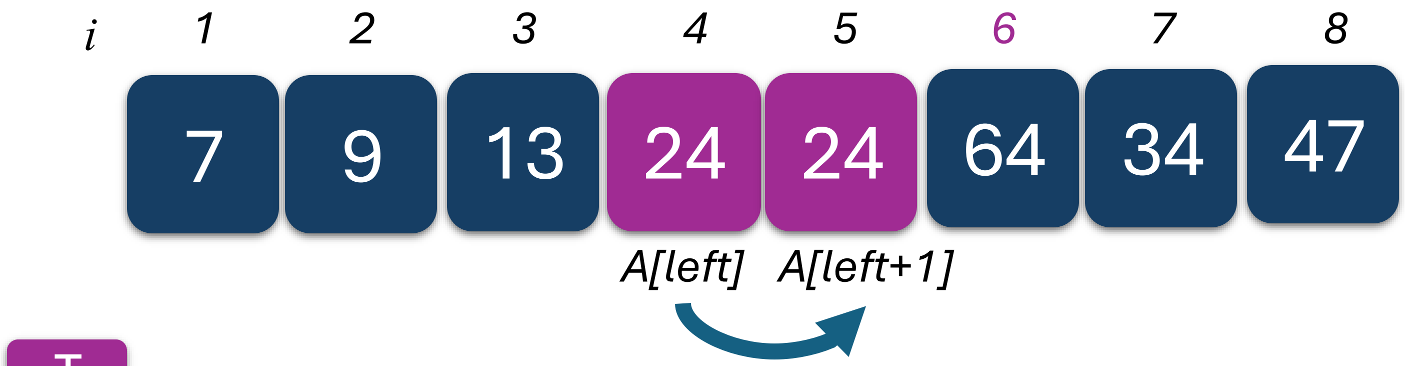
i	1	2	3	4	5	6	7	8
	7	9	13	24	64	64	34	47
				A[left]				

Insertion Sort

$24>23$

i=6

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                          23
        left = i – 1
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]           A[5] =A[4]
            left = left – 1
    A[left + 1] = item
```



i	1	2	3	4	5	6	7	8
	7	9	13	24	24	64	34	47

Insertion Sort

$13<23$

i=6

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]   23
        left = i – 1
        while left > 0 and A[left ] > item :   F
            A[left + 1] = A[left ]
            left = left – 1   3
    A[left + 1] = item
```

i	1	2	3	4	5	6	7	8
	7	9	13	24	24	64	34	47
			A[left]					

Insertion Sort

13<23

i=6

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]    23
        left = i – 1
        while left > 0 and A[left ] > item :    F
            A[left + 1] = A[left ]
            left = left – 1    3
    A[left + 1] = item
```

i	1	2	3	4	5	6	7	8
	7	9	13	24	24	64	34	47
			A[left]					

Insertion Sort

Insert 23

i=6

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]              23
        left = i – 1
        while left > 0 and A[left ] > item :     F
            A[left + 1] = A[left ]
            left = left – 1      3
    A[left + 1] = item           A[4] =23
```

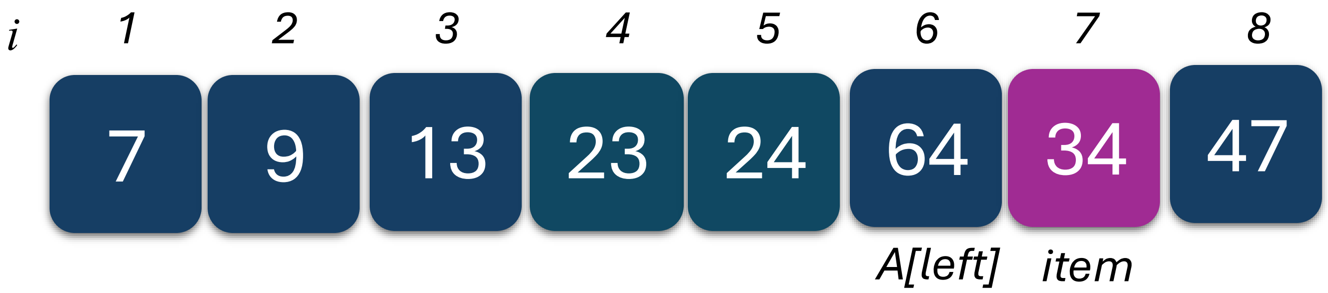
i	1	2	3	4	5	6	7	8
	7	9	13	23	24	64	34	47
			A[left]	A[left+1]				

Insertion Sort

Item 34

i=7

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]   34
        left = i – 1   6
        while left > 0 and A[left ] > item :   T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```



i	1	2	3	4	5	6	7	8
	7	9	13	23	24	64	34	47
						A[left]	item	

Insertion Sort

64>34

i=7

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                             34
        left = i – 1                            6
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]              A[6] =A[5]
            left = left – 1
    A[left + 1] = item
```

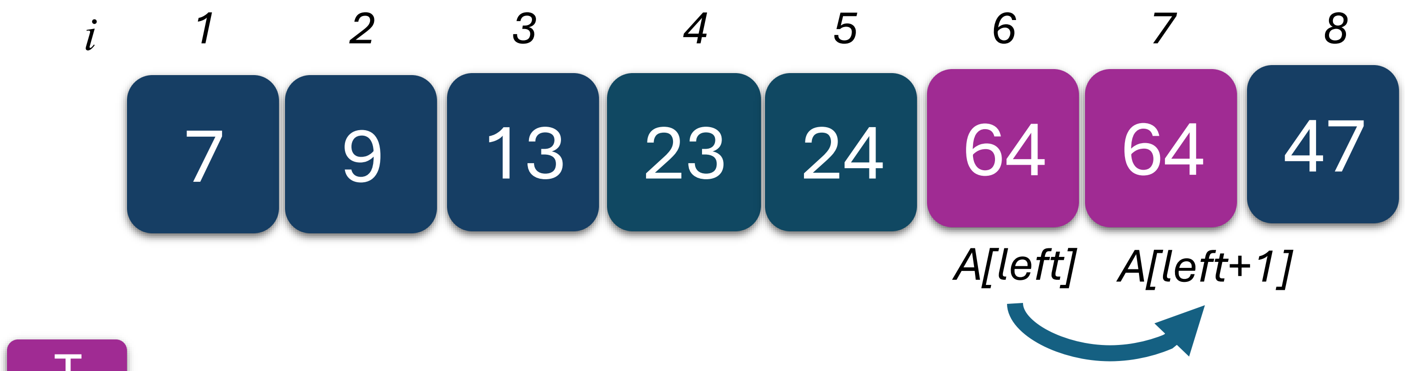
i	1	2	3	4	5	6	7	8
	7	9	13	23	24	64	34	47
						A[left]		

Insertion Sort

64>34

i=7

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]              34
        left = i – 1             6
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]              A[6] =A[5]
            left = left – 1
    A[left + 1] = item
```



i	1	2	3	4	5	6	7	8
	7	9	13	23	24	64	64	47

Insertion Sort

24<34

i=7

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]   34
        left = i – 1
        while left > 0 and A[left ] > item :   F
            A[left + 1] = A[left ]
            left = left – 1   5
    A[left + 1] = item
```

i	1	2	3	4	5	6	7	8
	7	9	13	23	24	64	64	47
					A[left]			

Insertion Sort

$24<34$

i=7

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]   34
        left = i – 1
        while left > 0 and A[left ] > item :   F
            A[left + 1] = A[left ]
            left = left – 1   5
    A[left + 1] = item
```

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
	7	9	13	23	24	64	64	47
					A[left]			

Insertion Sort

Insert 34

i=7

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                              34
        left = i – 1
        while left > 0 and A[left ] > item :
            A[left + 1] = A[left ]
            left = left – 1                      5
    A[left + 1] = item                           A[6] =34
```

i	1	2	3	4	5	6	7	8
	7	9	13	23	24	34	64	47
					A[left]	A[left+1]		

Insertion Sort

Item 47

i=8

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]        47
        left = i – 1       7
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

i	1	2	3	4	5	6	7	8
	7	9	13	23	24	34	64	47
							A[left]	item

Insertion Sort

64>47

i=8

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]        47
        left = i – 1       7
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

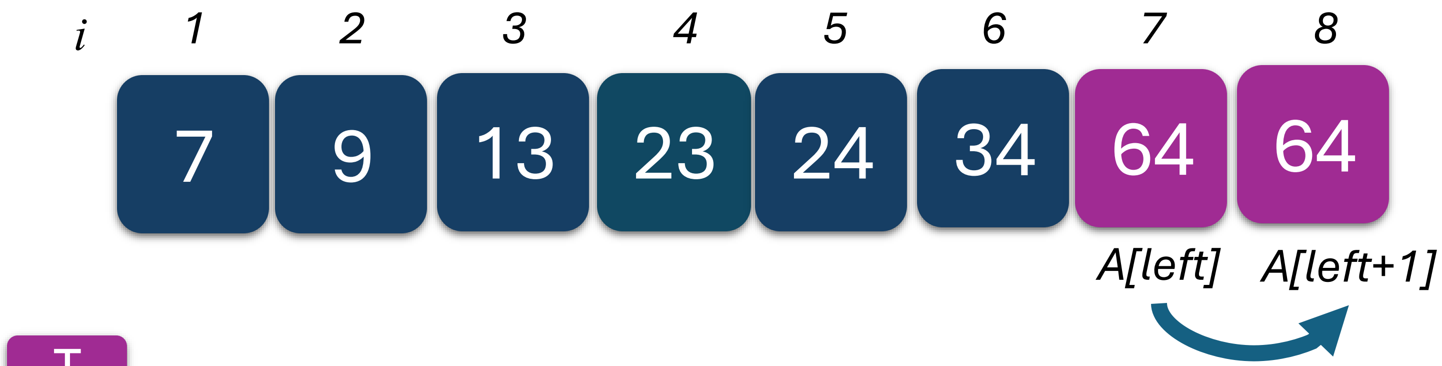
i	1	2	3	4	5	6	7	8
	7	9	13	23	24	34	64	47
							A[left]	

Insertion Sort

64>47

i=8

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]            47
        left = i – 1           7
        while left > 0 and A[left ] > item :     T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```



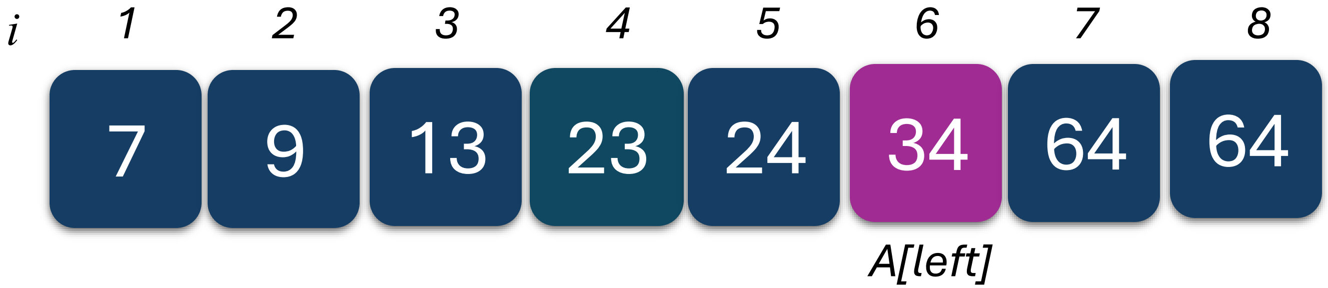
i	1	2	3	4	5	6	7	8
	7	9	13	23	24	34	64	64

Insertion Sort

$34<47$

i=8

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]        47
        left = i – 1
        while left > 0 and A[left ] > item :        F
            A[left + 1] = A[left ]
            left = left – 1        6
    A[left + 1] = item
```



i	1	2	3	4	5	6	7	8
	7	9	13	23	24	34	64	64
						A[left]		

Insertion Sort

$34<47$

i=8

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]   47
        left = i – 1
        while left > 0 and A[left ] > item :   F
            A[left + 1] = A[left ]
            left = left – 1   6
    A[left + 1] = item
```

i	1	2	3	4	5	6	7	8
	7	9	13	23	24	34	64	64
						A[left]		

Insert 47

i=8

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]   47
        left = i – 1
        while left > 0 and A[left ] > item :   F
            A[left + 1] = A[left ]
            left = left – 1   6
   A[left + 1] = item   A[6]=47
```

i	1	2	3	4	5	6	7	8
	7	9	13	23	24	34	47	64
						A[left]	A[left+1]	

Insertion Sort

i=9

```
insertion_sort(array A):
    for i = 1 to N:   F
        item = A[i]
        left = i – 1
        while left > 0 and A[left ] > item :
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

i	1	2	3	4	5	6	7	8
	7	9	13	23	24	34	47	64

Insertion Sort

i=9

```
insertion_sort(array A):
    for i = 1 to N:    F
        item = A[i]
        left = i – 1
        while left > 0 and A[left ] > item :
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

i	1	2	3	4	5	6	7	8
	7	9	13	23	24	34	47	64

Insertion Sort Complexity

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]
        left = i – 1
        while left > 0 and A[left ] > item :
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

C_1 n

C_2 n

C_3 n

Best Case?

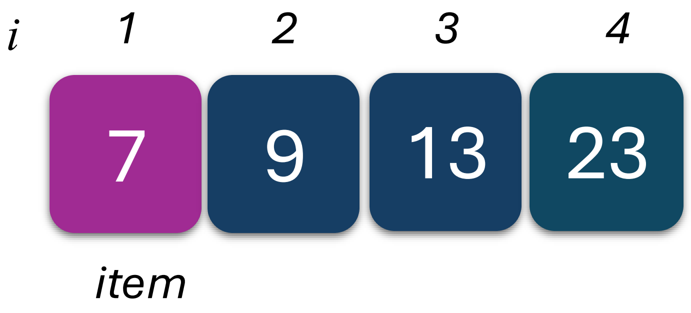
If the array is already sorted while-loop runs for 1 iteration, no inserts / shifts will happen

$$T(n) = (c_1 + +c_2+c_3)(n)$$
$$= an$$
$$= O(n)$$

Insertion Sort

i=1

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]        7
        left = i – 1       0
        while left > 0 and A[left ] > item :    F
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```



Insertion Sort

i=2

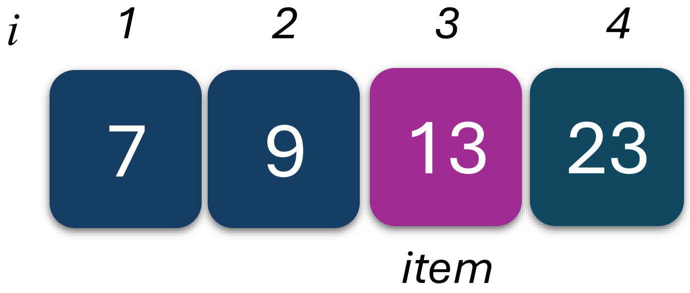
```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]   9
        left = i – 1   0
        while left > 0 and A[left ] > item :   F
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

i	1	2	3	4
	7	9	13	23
		item		

Insertion Sort

i=3

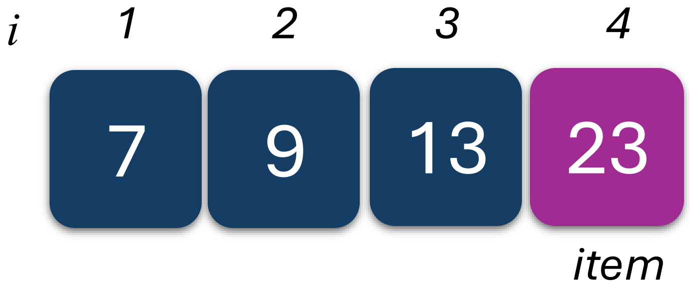
```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]          13
        left = i – 1         2
        while left > 0 and A[left ] > item :     F
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```



Insertion Sort

i=4

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                              13
        left = i – 1                             3
        while left > 0 and A[left ] > item :     F
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```



i	1	2	3	4
	7	9	13	23
				item

And so on and so forth....

Insertion Sort Complexity

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                                  }
        left = i – 1                                 }  C1      n
        while left > 0 and A[left ] > item :         }
            A[left + 1] = A[left ]                   }  1+2+3…+n
            left = left – 1                          }
        A[left + 1] = item                           }  C3      n
```

Worst Case?

If the array is reverse-sorted while-loop runs for i-1 iterations (dependent on iteration no.), because item has to be inserted all the way in front of the i-1 previous items

$$
\begin{aligned}
T(n) &= (c_1 + c_3)(n-1) + \\
&\qquad \{1+2+3+\ldots+(n-1)\}\, c_2 \\
&= (c_1 + c_3)(n-1) + n(n-1)/2 * c_2 \\
&= an^2 + bn + c \\
&= O(n^2)
\end{aligned}
$$

Insertion Sort

i=1

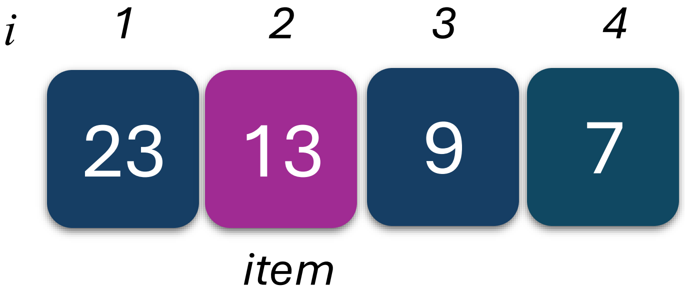
```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]        23
        left = i – 1        0
        while left > 0 and A[left ] > item :    F
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

i	1	2	3	4
	23	13	9	7
	item			

Insertion Sort

i=2

```
insertion_sort(array A):
    for $i = 1$ to $N$:
        $item = A[i]$    13
        $left = i - 1$    1
        while $left > 0$ and $A[left] > item$ :    T
            $A[left + 1] = A[left]$
            $left = left - 1$
    $A[left + 1] = item$
```

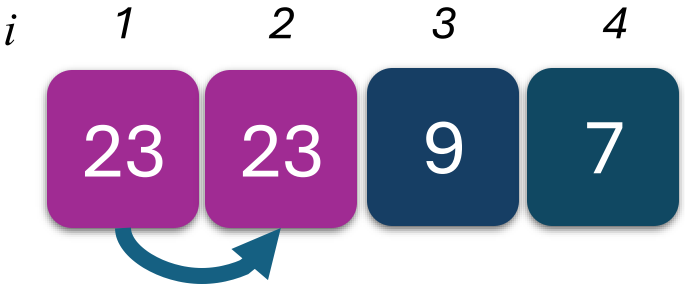


i	1	2	3	4
	23	13	9	7
		item		

Insertion Sort

i=2

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]            13
        left = i – 1           1
        while left > 0 and A[left ] > item :   T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

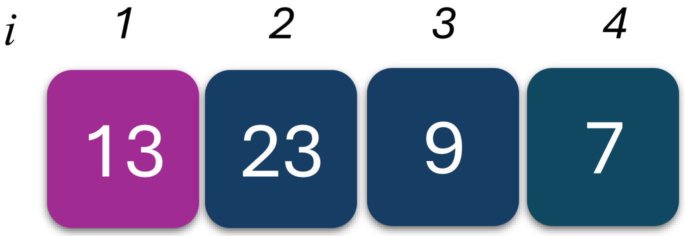


i	*1*	*2*	*3*	*4*
	23	23	9	7

Insertion Sort

i=2

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]            13
        left = i – 1           1
        while left > 0 and A[left ] > item :     T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```



i	1	2	3	4
	13	23	9	7

Number of iterations:
1

Insertion Sort

i=3

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]          9
        left = i – 1         2
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

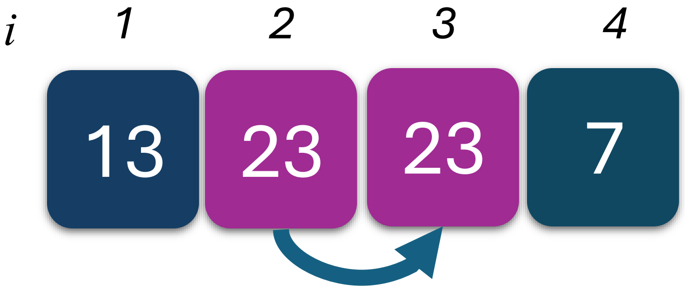
i	*1*	*2*	*3*	*4*
	13	23	9	7

Number of iterations:
1 +()

Insertion Sort

i=3

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                          9
        left = i – 1                         2
        while left > 0 and A[left ] > item :   T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

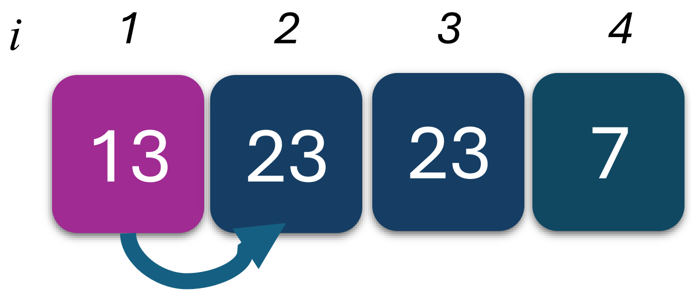


Number of iterations:
1 +(1)

Insertion Sort

i=3

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]          9
        left = i – 1         2
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

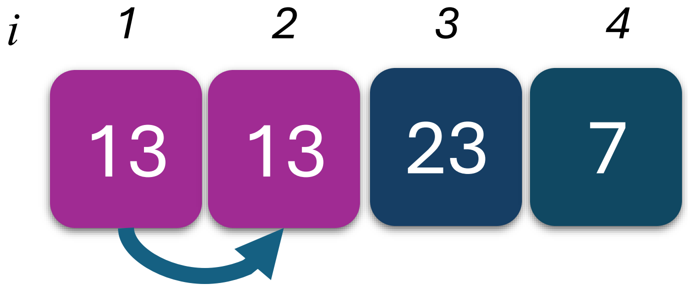


Number of iterations:
1 +(1)

Insertion Sort

i=3

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                              9
        left = i – 1                             2
        while left > 0 and A[left ] > item :     T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```



i	*1*	*2*	*3*	*4*
	13	13	23	7

Number of iterations:
1 +(1+1)

Insertion Sort

i=3

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                              9
        left = i – 1                             2
        while left > 0 and A[left ] > item :     T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

i	*1*	*2*	*3*	*4*
	9	13	23	7

Number of iterations:
1 +(1+1)

Insertion Sort

i=4

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                              7
        left = i – 1                             3
        while left > 0 and A[left ] > item :     T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

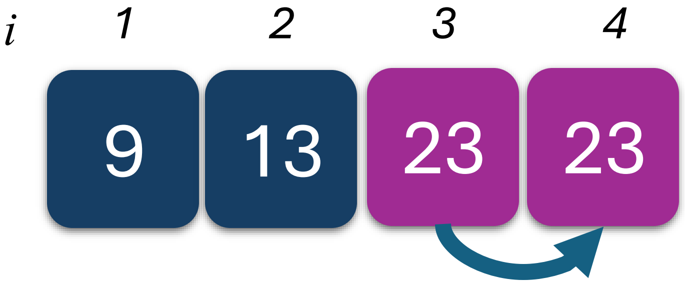
i	*1*	*2*	*3*	*4*
	9	13	23	7

Number of iterations:
1 +(1+1)+

Insertion Sort

i=4

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]                                  7
        left = i – 1                                 3
        while left > 0 and A[left ] > item :         T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```



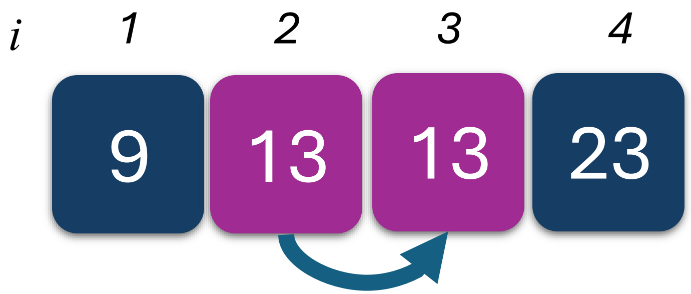
i	*1*	*2*	*3*	*4*
	9	13	23	23

Number of iterations:
1 +(1+1)+(1)

Insertion Sort

i=4

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]      7
        left = i – 1
        while left > 0 and A[left ] > item :      T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

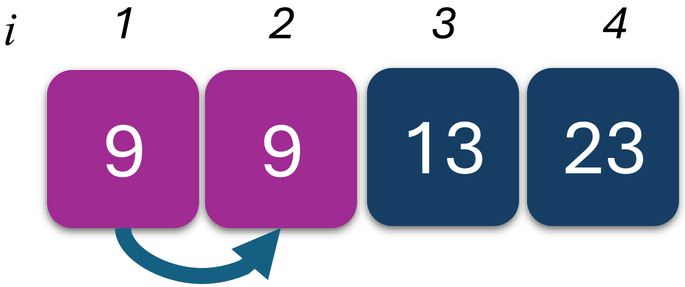


Number of iterations:
1 +(1+1)+(1+1)

Insertion Sort

i=4

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]    7
        left = i – 1
        while left > 0 and A[left ] > item :    T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```



Number of iterations:
1 +(1+1)+(1+1+1)

Insertion Sort

i=4

```
insertion_sort(array A):
    for i = 1 to N:
        item = A[i]        7
        left = i – 1
        while left > 0 and A[left ] > item :        T
            A[left + 1] = A[left ]
            left = left – 1
    A[left + 1] = item
```

i	*1*	*2*	*3*	*4*
	7	9	13	23

Number of iterations:
1 +(1+1)+(1+1+1) = 1+2+3

Having n items….
Number of iterations:
1 +(1+1)+(1+1+1)+…+n = 1+2+3+…+n

Insertion Sort Complexity

- Memory: O(1) → no extra memory needed

Selection Sort

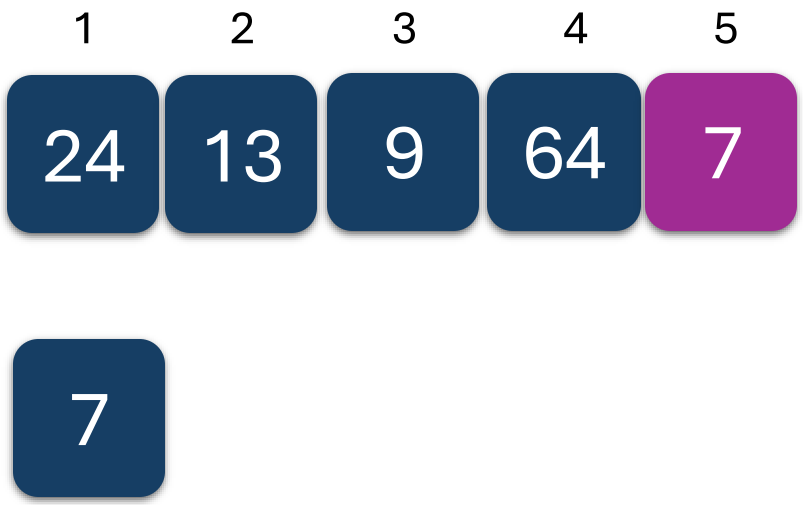
Selection Sort

- A naive sorting algorithm; one of the slowest sorting algorithms since it repeatedly performs same task without learning from previous iterations
- **Analogy**: Given a pile of cards, a common way to sort it is to select and remove the smallest card, and repeat the process until all cards are gone
- **Idea**: Find 1st smallest element and exchange it with element in 1st position; find 2nd smallest element and exchange it with element in 2nd position, and so on.
- Minimizes number of swaps; useful for applications where cost of swapping in memory is high

Selection Sort

1	2	3	4	5
24	13	9	64	7

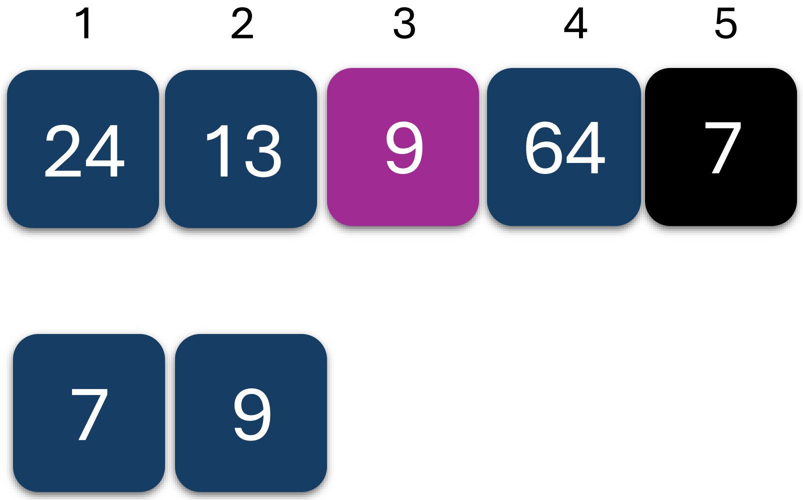
Selection Sort



1	2	3	4	5
24	13	9	64	7

7

Selection Sort



1	2	3	4	5
24	13	9	64	7
7	9			

Selection Sort

1	2	3	4	5
24	13	9	64	7

7	9	13

Selection Sort

1	2	3	4	5
24	13	9	64	7

7	9	13	24

Selection Sort

1	2	3	4	5
24	13	9	64	7
7	9	13	24	64

Selection Sort

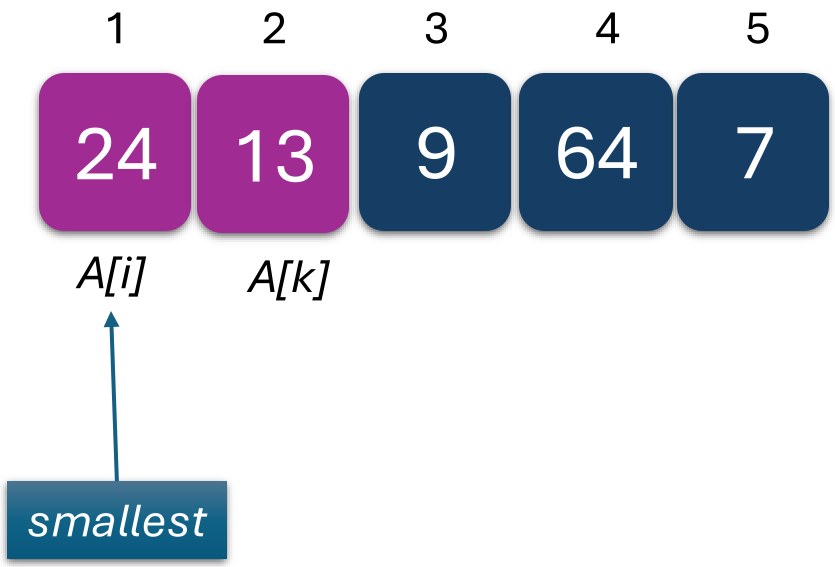
```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]
    for k = i + 1 to N:
        if A[k] < ith_smallest :
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```

1	2	3	4	5
24	13	9	64	7

Selection Sort

i=1

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]          24
    for k = i + 1 to N:          k=2
        if A[k] < ith_smallest :     T
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```

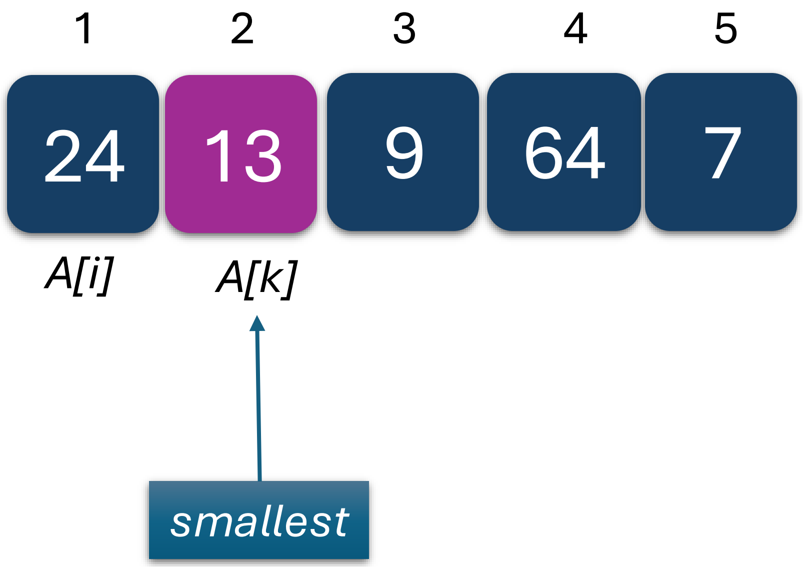


Total Number of iteration (inner loop):
For i = 1
1

Selection Sort

i=1

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]   24
    for k = i + 1 to N:   k=2
        if A[k] < ith_smallest :   T
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```

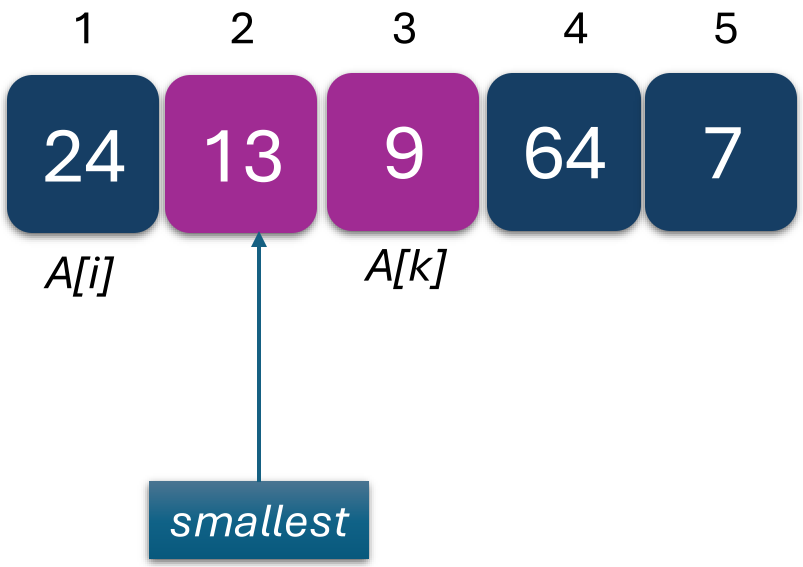


Total Number of iteration (inner loop):
For i = 1
1

Selection Sort

i=1

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]          24
    for k = i + 1 to N:          k=3
        if A[k] < ith_smallest :
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```



1	2	3	4	5
24	13	9	64	7

Total Number of iteration (inner loop):

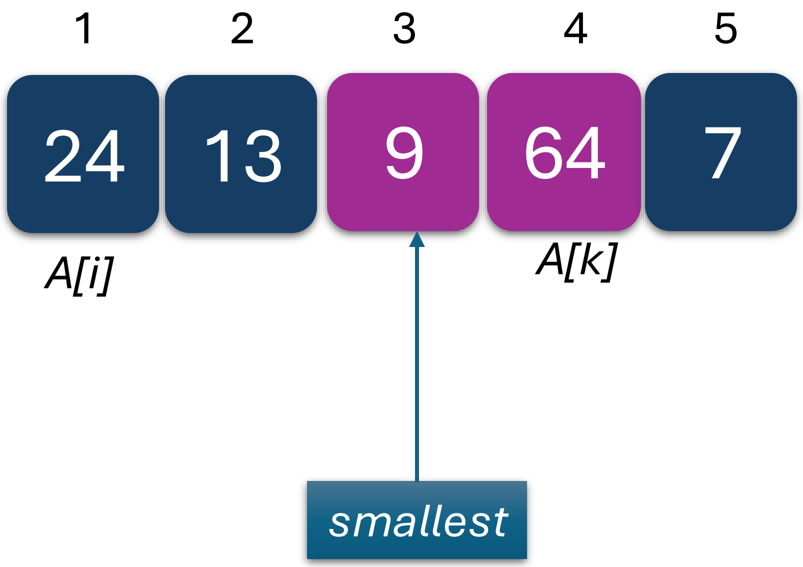
For i = 1

1+1

Selection Sort

i=1

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]          24
    for k = i + 1 to N:          k=4
        if A[k] < ith_smallest :     F
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```



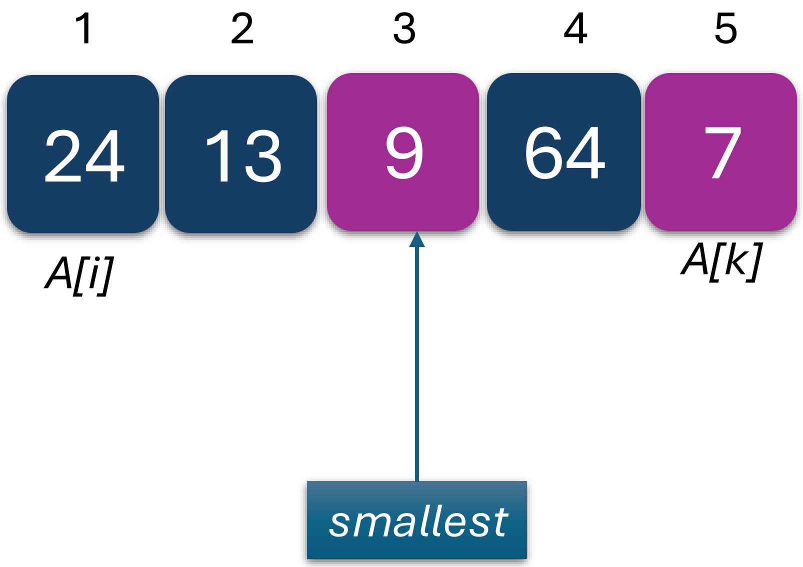
1	2	3	4	5
24	13	9	64	7
A[i]			*A[k]*	

smallest

Total Number of iteration (inner loop):
For i = 1
1+1+1

i=1

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]              24
    for k = i + 1 to N:              k=5
        if A[k] < ith_smallest :     T
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```

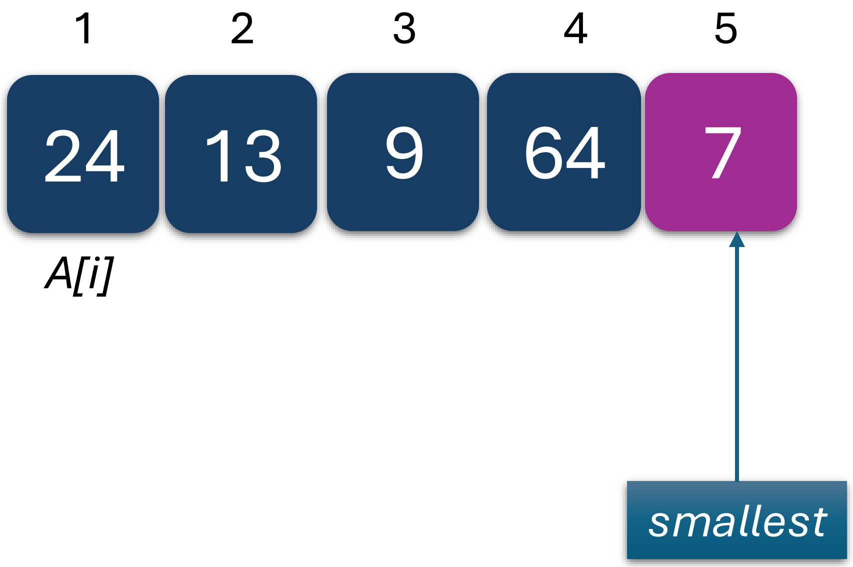


Total Number of iteration (inner loop):
For i = 1
1+1+1+1 = 4

Selection Sort

i=1

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]                24
    for k = i + 1 to N:                k=5
        if A[k] < ith_smallest :       T
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```



Selection Sort

i=1

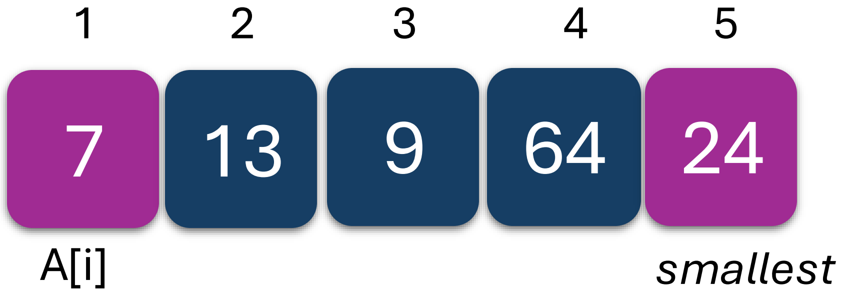
```
selection_sort(array A):
for $i = 1$ to $N$:
    ith_smallest = A[i]            24
    for $k = i + 1$ to $N$:        k=5
        if $A[k] <$ ith_smallest :
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest       A[1] ↔ A[5]
```

1	2	3	4	5
24	13	9	64	7
A[i]				*smallest*

Selection Sort

i=1

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]          24
    for k = i + 1 to N:          k=5
        if A[k] < ith_smallest :
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest     A[1] ↔ A[5]
```

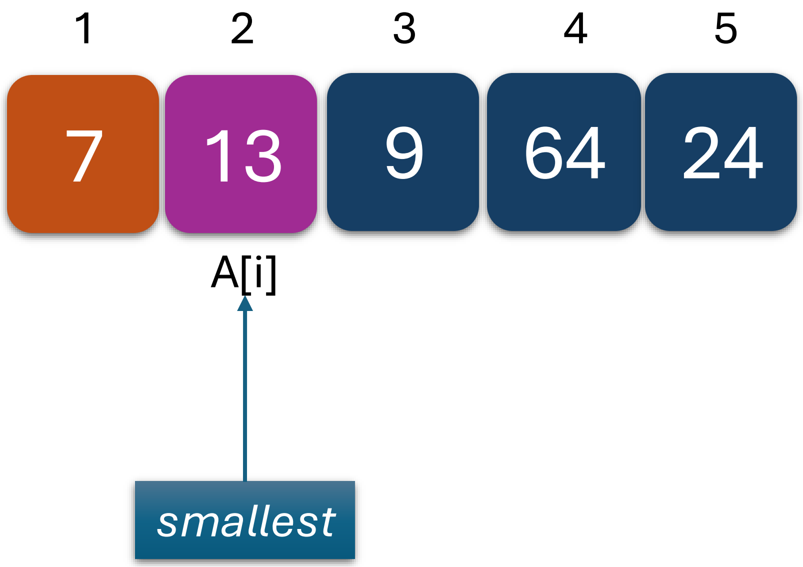


1	2	3	4	5
7	13	9	64	24
A[i]				*smallest*

Selection Sort

i=2

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]    13
    for k = i + 1 to N:
        if A[k] < ith_smallest :
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```

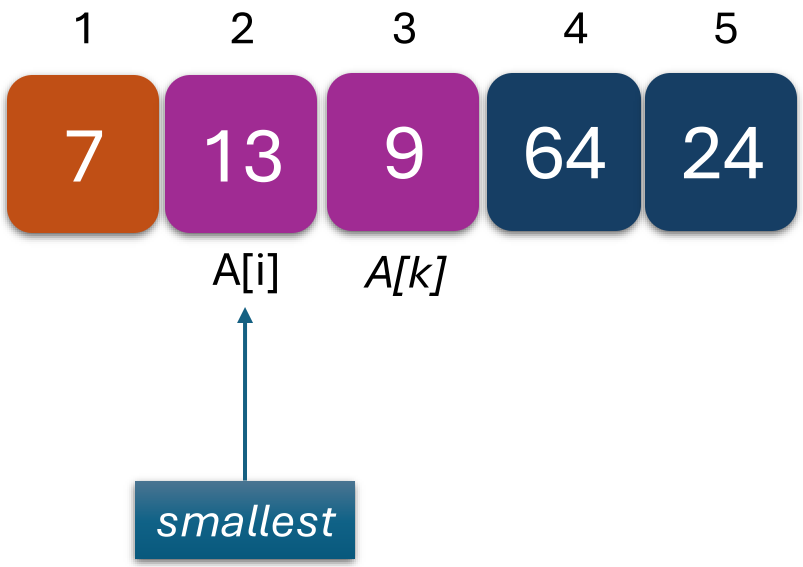


Total Number of iteration (inner loop):
For i = 2
4+ ()

Selection Sort

i=2

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]          13
    for k = i + 1 to N:          k=3
        if A[k] < ith_smallest :     T
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```

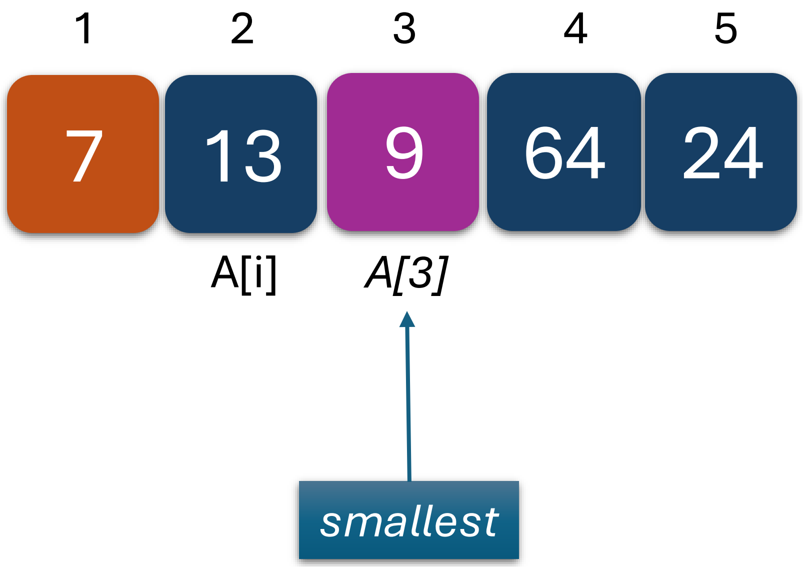


Total Number of iteration (inner loop):
For i = 2
4+ (1)

Selection Sort

i=2

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]
    for k = i + 1 to N:
        if A[k] < ith_smallest :      T
            ith_smallest = A[k]       9
    swap A[i] ↔ ith_smallest
```



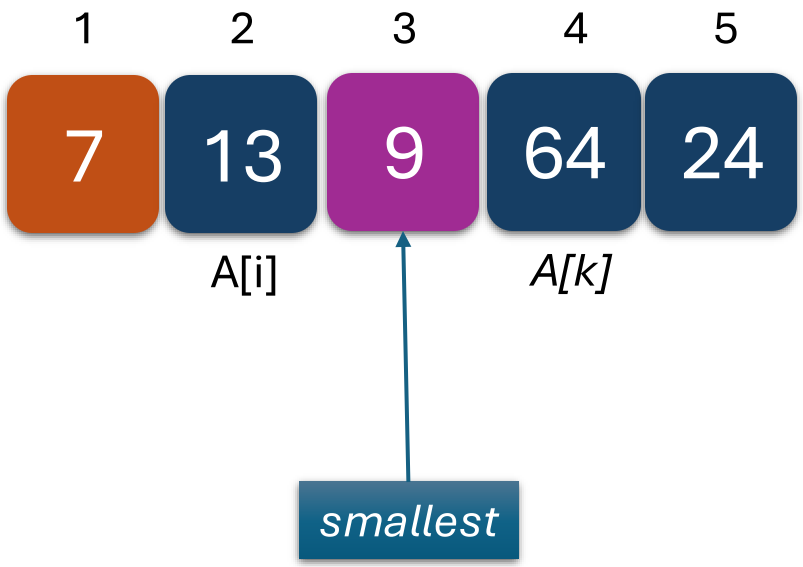
1	2	3	4	5
7	13	9	64	24

Total Number of iteration (inner loop):
For i = 2
4+ (1)

Selection Sort

i=2

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]
    for k = i + 1 to N:          k=4
        if A[k] < ith_smallest :     F
            ith_smallest = A[k]      9
    swap A[i] ↔ ith_smallest
```

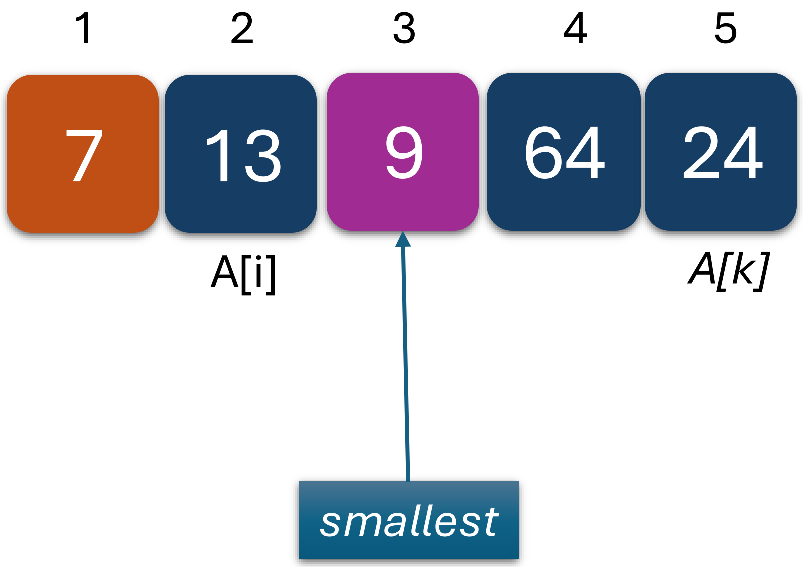


Total Number of iteration (inner loop):
For i = 2
4+ (1+1)

Selection Sort

i=2

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]
    for k = i + 1 to N:            k=5
        if A[k] < ith_smallest :   F
            ith_smallest = A[k]    9
    swap A[i] ↔ ith_smallest
```



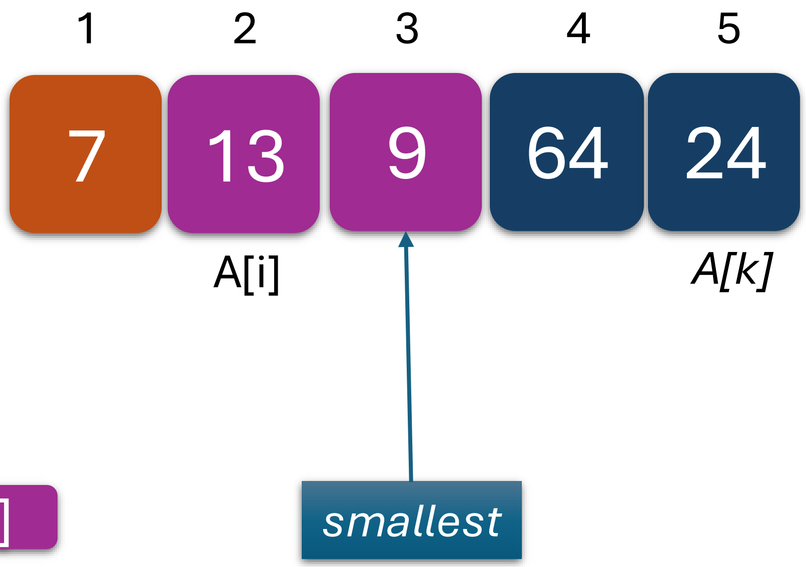
1	2	3	4	5
7	13	9	64	24

Total Number of iteration (inner loop):
For i = 2
4+ (1+1+1)

Selection Sort

i=2

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]
    for k = i + 1 to N:
        if A[k] < ith_smallest :
            ith_smallest = A[k]    9
    swap A[i] ↔ ith_smallest    A[2] ↔ A[3]
```



1	2	3	4	5
7	13	9	64	24

Total Number of iteration (inner loop):
For i = 2
4+ 3

Selection Sort

i=2

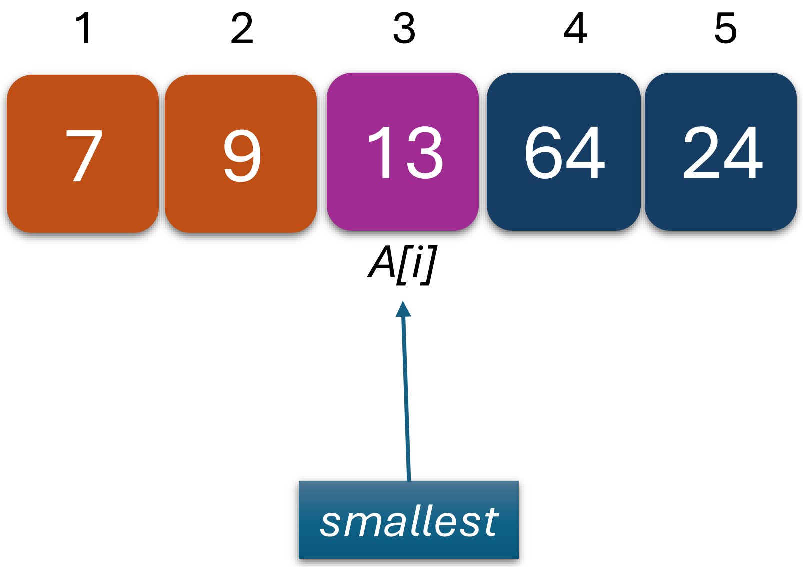
```
selection_sort(array A):
for $i = 1$ to N:
    ith_smallest = A[i]
    for $k = i + 1$ to N:
        if A[k] < ith_smallest :
            ith_smallest = A[k]   9
    swap A[i] ↔ ith_smallest   A[2] ↔ A[3]
```

1	2	3	4	5
7	9	13	64	24

Selection Sort

i=3

```
selection_sort(array A):
for i = 1 to N:
      ith_smallest = A[i]   13
      for k = i + 1 to N:
            if A[k] < ith_smallest :
                  ith_smallest = A[k]
      swap A[i] ↔ ith_smallest
```

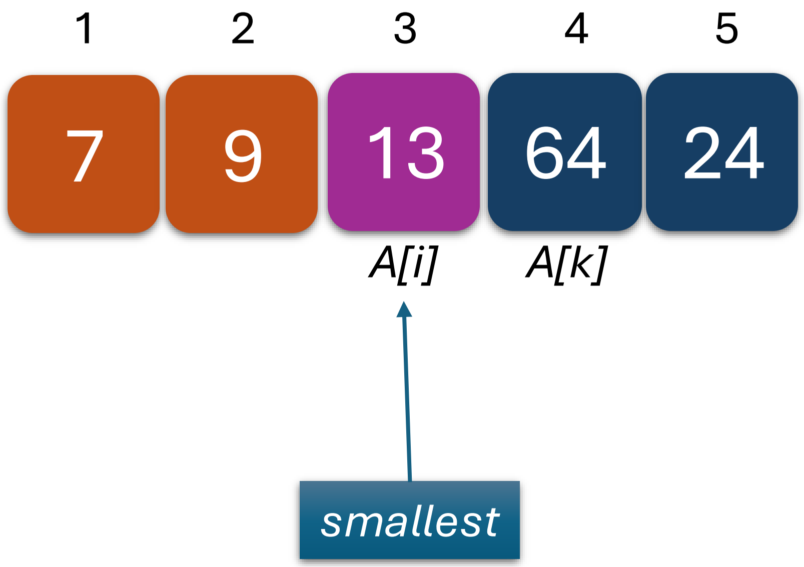


Total Number of iteration (inner loop):
For i = 3
4+ 3+()

Selection Sort

i=3

```
selection_sort(array A):
for $i = 1$ to N:
    $ith\_smallest = A[i]$          13
    for $k = i + 1$ to N:          k=4
        if $A[k] < ith\_smallest$ :          F
            $ith\_smallest = A[k]$
    swap $A[i] \leftrightarrow ith\_smallest$
```

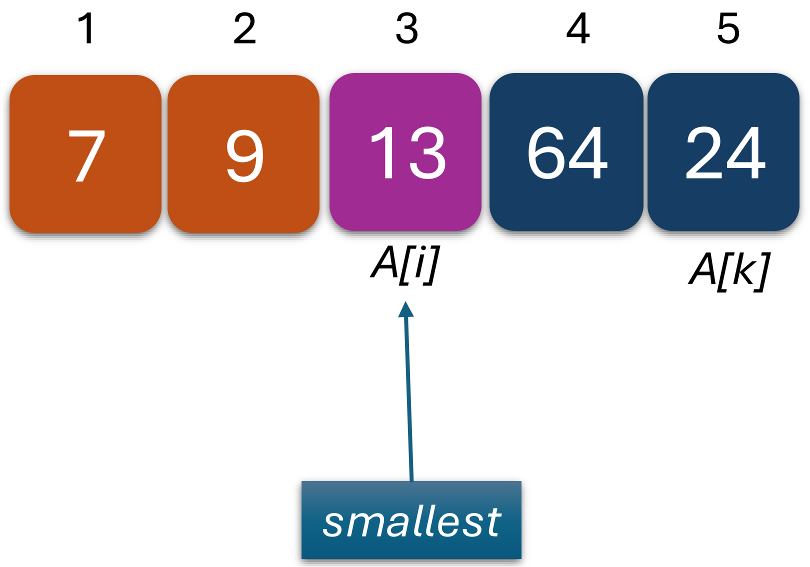


Total Number of iteration (inner loop):
For i = 3
4+ 3+(1)

Selection Sort

i=3

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]        13
    for k = i + 1 to N:        k=5
        if A[k] < ith_smallest :       F
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```



Total Number of iteration (inner loop):
For i = 3
4+ 3+(1+1)

Selection Sort

i=3

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]          13
    for k = i + 1 to N:          k=5
        if A[k] < ith_smallest :
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest     A[3] ↔ A[3]
```

1	2	3	4	5
7	9	13	64	24

Total Number of iteration (inner loop):

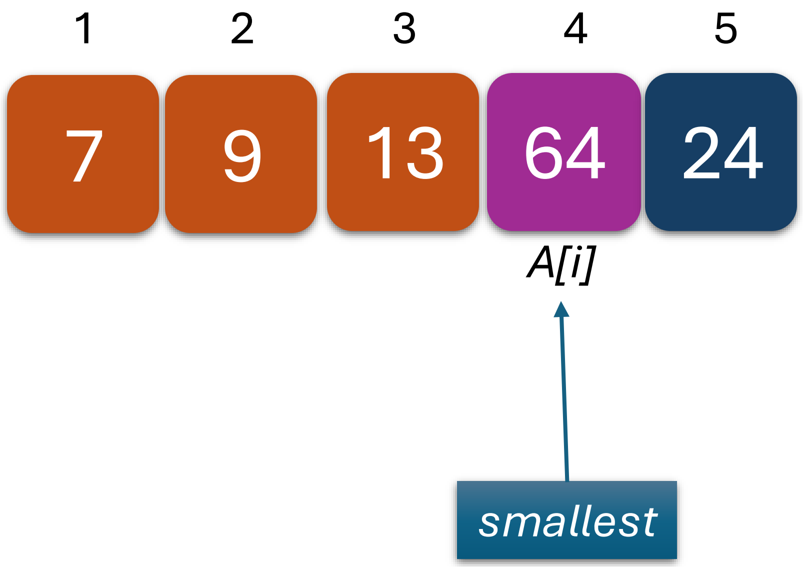
For i = 3

4+ 3+2

Selection Sort

i=4

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]   64
    for k = i + 1 to N:
        if A[k] < ith_smallest :
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```



1	2	3	4	5
7	9	13	64	24

A[i]

smallest

Total Number of iteration (inner loop):

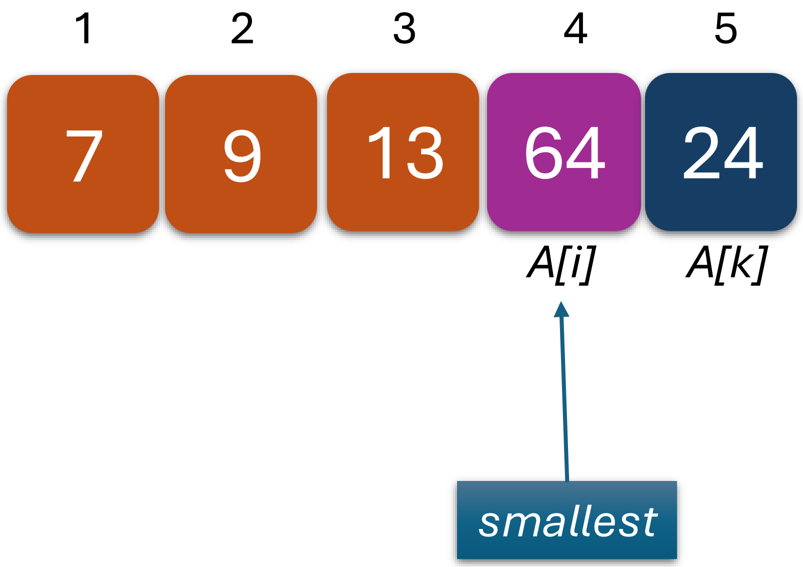
For i = 4

4+ 3+2+()

Selection Sort

i=4

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]          64
    for k = i + 1 to N:          k=5
        if A[k] < ith_smallest :     T
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```

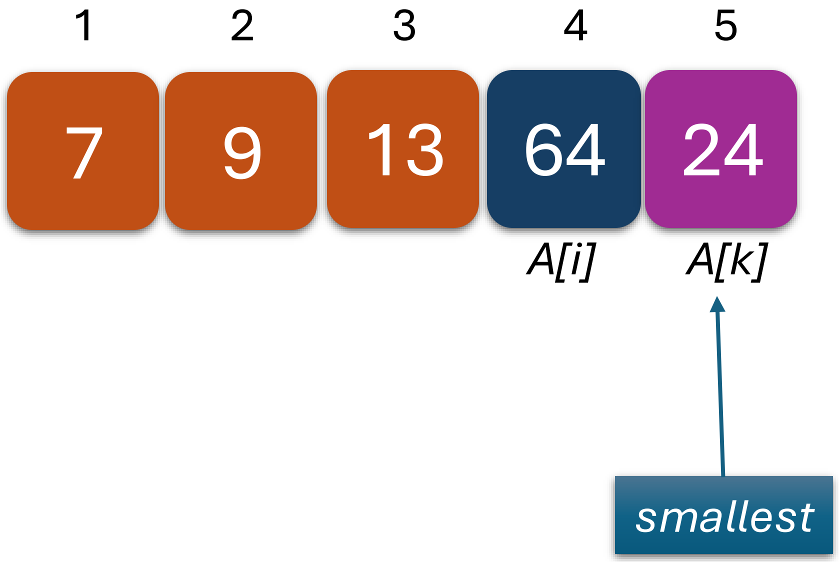


Total Number of iteration (inner loop):
For i = 4
4+ 3+2+(1)

Selection Sort

i=4

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]
    for k = i + 1 to N:          k=5
        if A[k] < ith_smallest :     T
            ith_smallest = A[k]      24
    swap A[i] ↔ ith_smallest
```

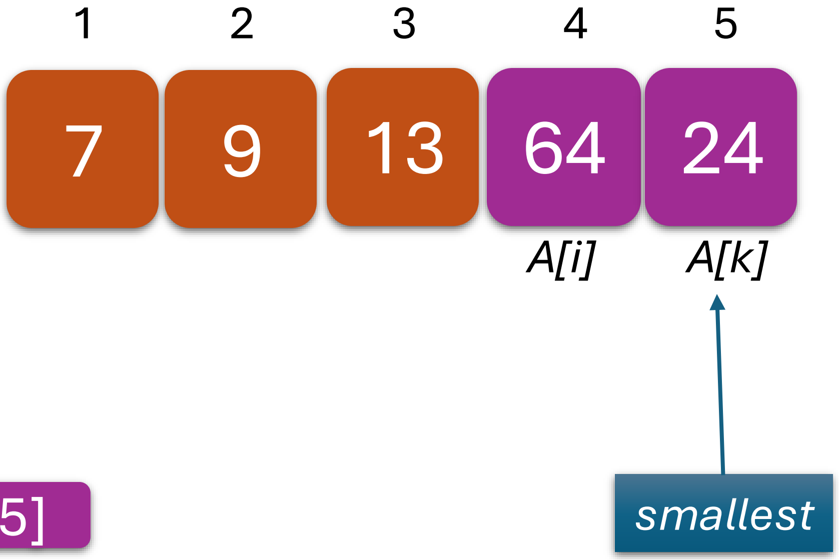


1	2	3	4	5
7	9	13	64	24
			A[i]	A[k]

Selection Sort

i=4

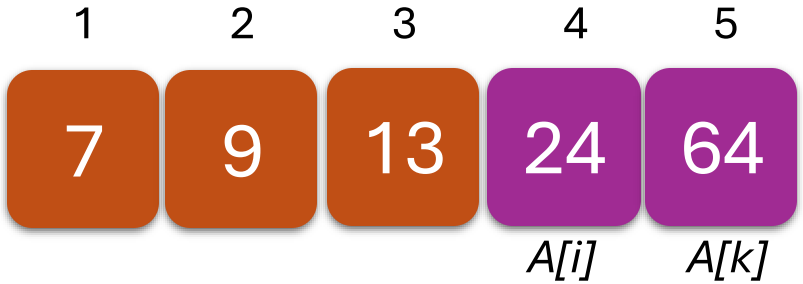
```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]
    for k = i + 1 to N:            k=5
        if A[k] < ith_smallest :
            ith_smallest = A[k]    24
    swap A[i] ↔ ith_smallest       A[4] ↔ A[5]
```



Selection Sort

i=4

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]
    for k = i + 1 to N:          k=5
        if A[k] < ith_smallest :
            ith_smallest = A[k]  24
    swap A[i] ↔ ith_smallest     A[4] ↔ A[5]
```



Selection Sort

i=4

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]
    for k = i + 1 to N:
        if A[k] < ith_smallest :
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```

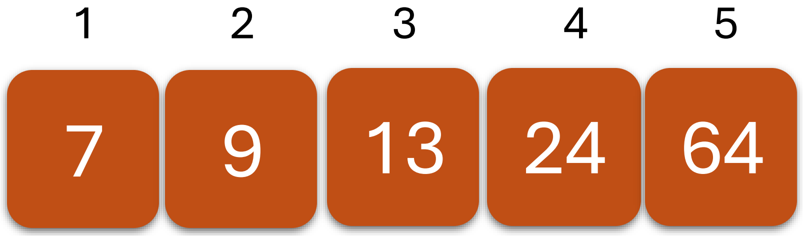
1	2	3	4	5
7	9	13	24	64

Total Number of iteration (inner loop):
For i = 4
4+ 3+2+1+0

Selection Sort

i=5

```
selection_sort(array A):
for $i = 1$ to N:
    ith_smallest = A[i]
    for $k = i + 1$ to N:
        if $A[k] <$ ith_smallest :
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```



1	2	3	4	5
7	9	13	24	64

Selection Sort Complexity

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]
    for k = i + 1 to N:
        if A[k] < ith_smallest :
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```

Best Case?

Worst Case?

Both **O(n^2)**

Selection Sort Complexity

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]                  $C_1$                          n
    for k = i + 1 to N:
        if A[k] < ith_smallest :         $C_2[(n-1)+(n-2)+...+2+1]$     n
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest             $C_3$                          n
```

- **Outer for-loop**
  - runs from 1 to N
- **Inner for-loop**
  - finding the ith smallest element
  - runs from i+1 to N (dependent on iteration number)

$$T(n) = c_1(n)+(n(n-1)/2)c_2+ c_3(n)$$
$$= O(n^2)$$

Insertion Sort Complexity

```
insertion_sort(array A):
    for $i = 1$ to $N$:
        $item = A[i]$
        $left = i – 1$
        while $left > 0$ and $A[left] > item$ :
            $A[left + 1] = A[left]$
            $left = left – 1$
    $A[left + 1] = item$
```

Best Case?

Worst Case?

RT is $O(N^2)$ regardless of whether the array is already sorted, in reverse order, or in random order; the algorithm will still naively look for the i-th smallest element per iteration, without learning anything.

Selection Sort Complexity

```
selection_sort(array A):
for i = 1 to N:
    ith_smallest = A[i]
    for k = i + 1 to N:
        if A[k] < ith_smallest :
            ith_smallest = A[k]
    swap A[i] ↔ ith_smallest
```

C_1	n
$C_2[(n-1)+(n-2)+...+2+1]$	n
C_3	n

Example:

N = 5

Iteration1	i = 1 k = 2 to 5	inner loop = 4 iterations
Iteration 2	i = 2 k = 3 to 5	inner loop = 3 iterations
Iteration 3	i = 3 k = 4 to 5	inner loop = 2 iterations
Iteration 4	i = 4 k = 5	inner loop = 1 iteration
Iteration 5	i = 5 k = None	inner loop = 0 iterations

Selection Sort Complexity

- Memory: O(1) → no extra memory needed

Bubble Sort

Bubble Sort

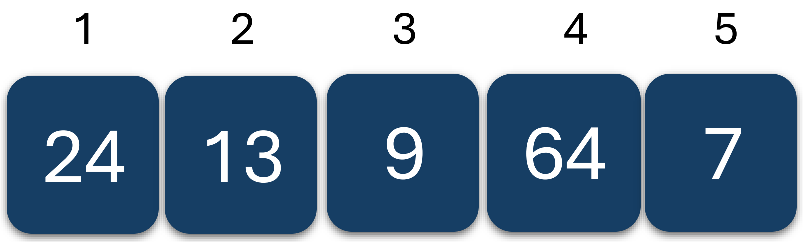
- Works by repeatedly swapping adjacent elements that are out of order
- Small numbers bubble up the array during execution
- At the end of each loop iteration, first i elements are sorted
- Stop if no swaps happened anymore → array is sorted

Bubble Sort

```
bubble_sort(array A):
    do:
        for each adjacent pair of items:
            swap if not in order
    while a swap occured
```

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

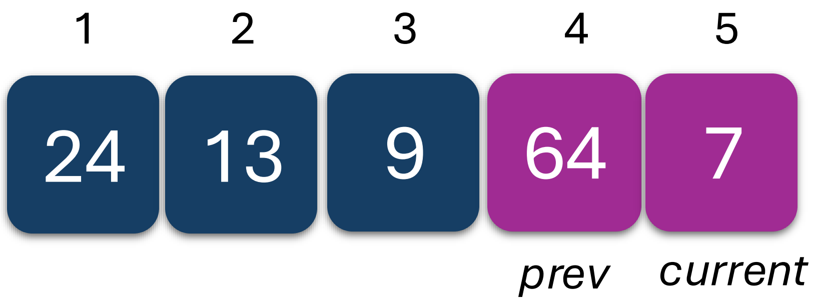


1	2	3	4	5
24	13	9	64	7

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:          5
            prev = current – 1         4
            if A[prev] > A[current]:   T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass



1	2	3	4	5
24	13	9	64	7
			prev	current

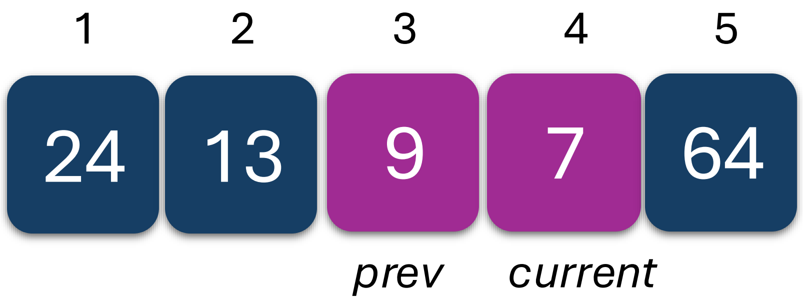
First pass
Swapped=True

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:          5
            prev = current – 1         4
            if A[prev] > A[current]:        T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

1	2	3	4	5
24	13	9	7	64
			prev	*current*

First pass
Swapped=True

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:        4
            prev = current – 1       3
            if A[prev] > A[current]:     T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

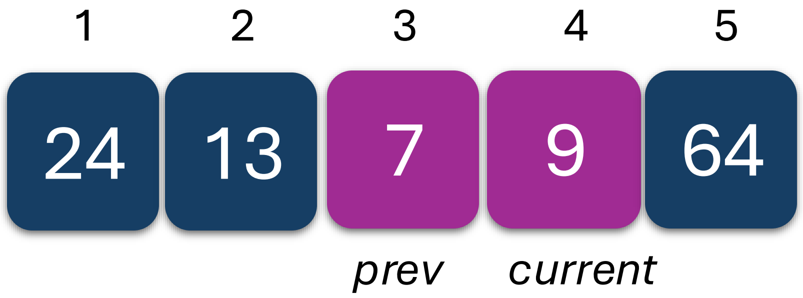


1	2	3	4	5
24	13	9	7	64
		prev	current	

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:        4
            prev = current – 1       3
            if A[prev] > A[current]:     T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass
Swapped=True



Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:          3
            prev = current – 1         2
            if A[prev] > A[current]:       T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

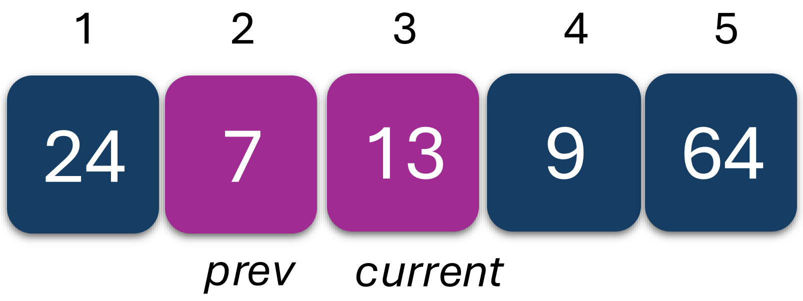
First pass
Swapped=True

1	2	3	4	5
24	13	7	9	64
	prev	*current*		

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:          3
            prev = current – 1         2
            if A[prev] > A[current]:      T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass
Swapped=True

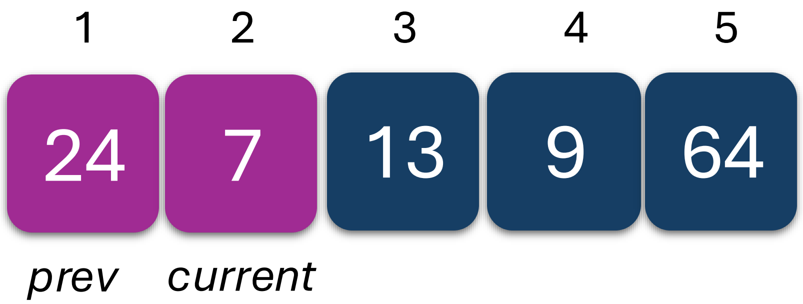


1	2	3	4	5
24	7	13	9	64
	prev	*current*		

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:      2
            prev = current – 1     1
            if A[prev] > A[current]:    T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

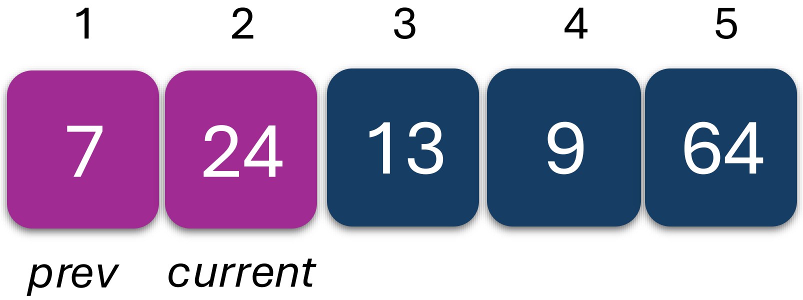
First pass
Swapped=True



Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:      2
            prev = current – 1     1
            if A[prev] > A[current]:      T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass
Swapped=True

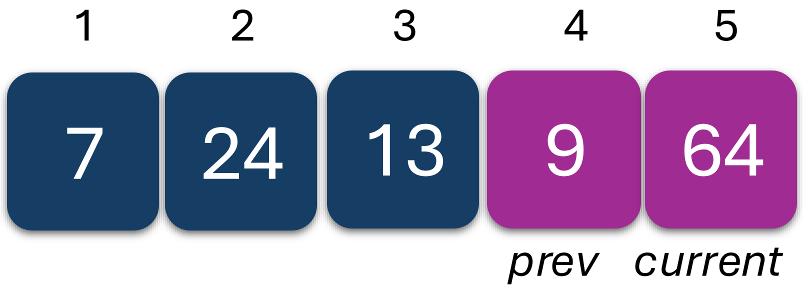


1	2	3	4	5
7	24	13	9	64
prev	current			

Bubble Sort

Second pass
Swapped=False

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:          5
            prev = current – 1         4
            if A[prev] > A[current]:   F
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

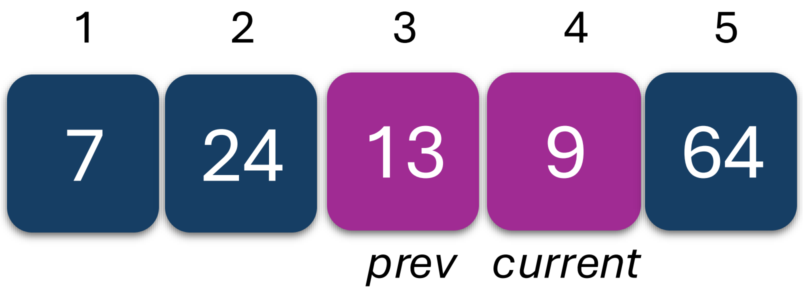


1	2	3	4	5
7	24	13	9	64
			prev	current

Bubble Sort

Second pass
Swapped=False

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:     4
            prev = current – 1    3
            if A[prev] > A[current]:   T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

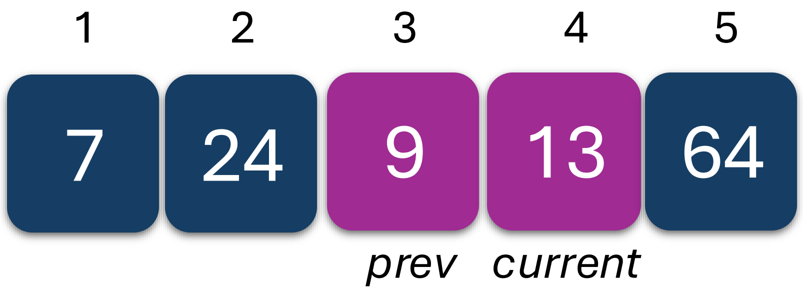


1	2	3	4	5
7	24	13	9	64
		prev	*current*	

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:            4
            prev = current – 1           3
            if A[prev] > A[current]:     T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Second pass
Swapped=True



1	2	3	4	5
7	24	9	13	64
		prev	*current*	

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:        3
            prev = current – 1       2
            if A[prev] > A[current]:      T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

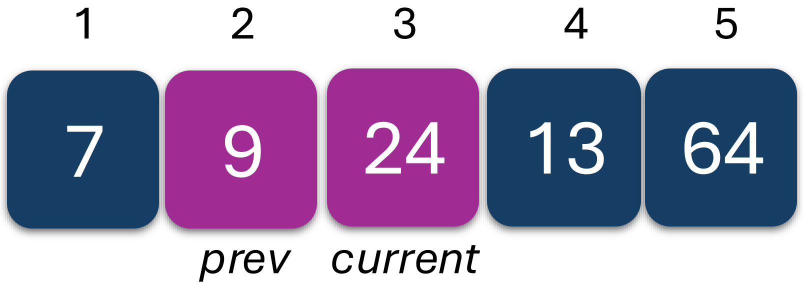
Second pass
Swapped=True

1	2	3	4	5
7	24	9	13	64
	prev	current		

Bubble Sort

Second pass
Swapped=True

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:        3
            prev = current – 1       2
            if A[prev] > A[current]:     T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```



1	2	3	4	5
7	9	24	13	64
	prev	*current*		

Bubble Sort

Second pass
Swapped=True

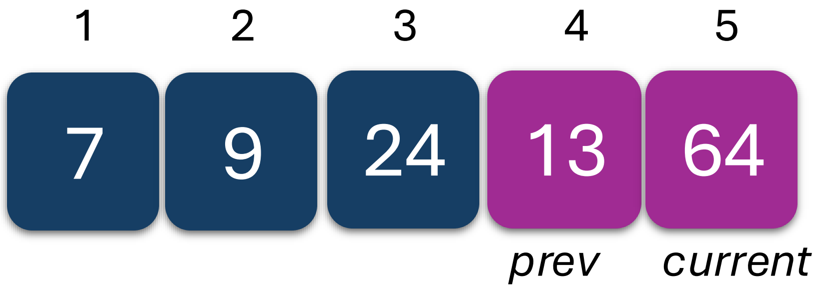
```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:        2
            prev = current – 1       1
            if A[prev] > A[current]:     F
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

1	2	3	4	5
7	9	24	13	64
prev	*current*			

Bubble Sort

Third pass
Swapped=False

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:        5
            prev = current – 1        4
            if A[prev] > A[current]:        F
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```



1	2	3	4	5
7	9	24	13	64
			prev	current

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:          4
            prev = current – 1         3
            if A[prev] > A[current]:   T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Third pass

Swapped=False

1	2	3	4	5
7	9	24	13	64
		prev	*current*	

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:        4
            prev = current – 1       3
            if A[prev] > A[current]:    T
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Third pass
Swapped=True

1	2	3	4	5
7	9	13	24	64
		prev	*current*	

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:          3
            prev = current – 1         2
            if A[prev] > A[current]:   F
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Third pass
Swapped=True

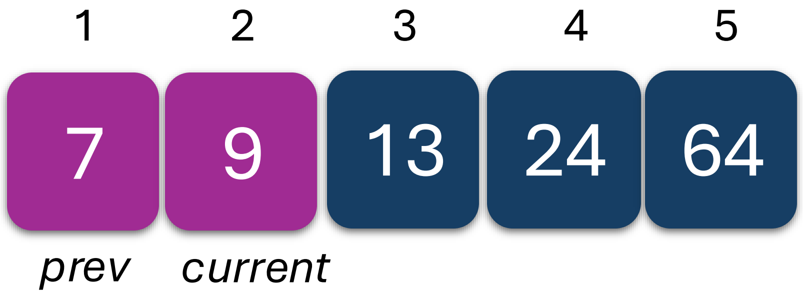
1	2	3	4	5
7	9	13	24	64
	prev	current		

Bubble Sort

Third pass

Swapped=True

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:          2
            prev = current – 1         1
            if A[prev] > A[current]:   F
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```



1	2	3	4	5
7	9	13	24	64
prev	*current*			

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:          5
            prev = current – 1         4
            if A[prev] > A[current]:   F
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Fourth pass

Swapped=False

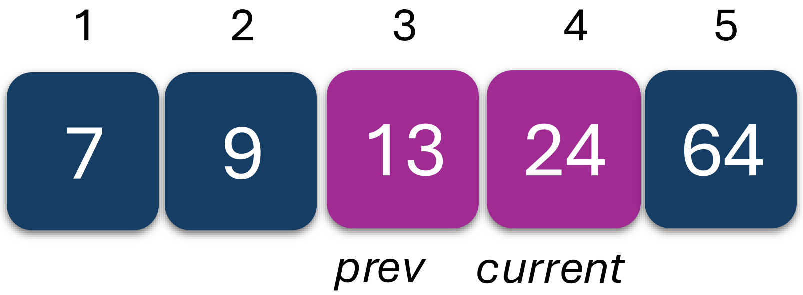
1	2	3	4	5
7	9	13	24	64
			prev	current

Bubble Sort

Fourth pass

Swapped=False

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:          4
            prev = current – 1         3
            if A[prev] > A[current]:   F
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```



1	2	3	4	5
7	9	13	24	64
		prev	current	

Bubble Sort

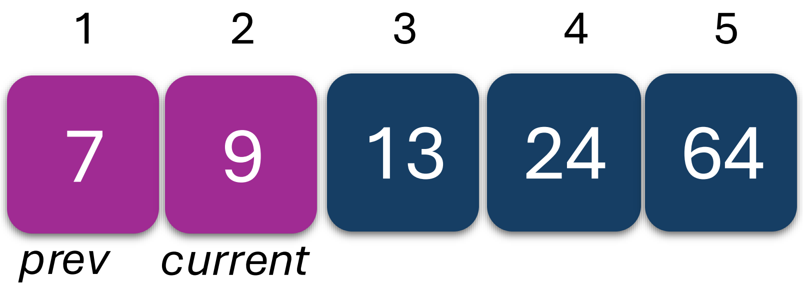
```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:     3
            prev = current – 1    2
            if A[prev] > A[current]:     F
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Fourth pass
Swapped=False

1	2	3	4	5
7	9	13	24	64
	prev	current		

Fourth pass
Swapped=False

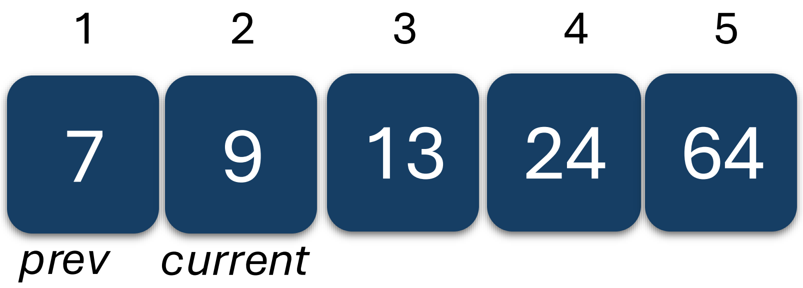
```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:        2
            prev = current – 1       1
            if A[prev] > A[current]:     F
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```



Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:          2
            prev = current – 1         1
            if A[prev] > A[current]:   F
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Fourth pass
Swapped=False



Bubble Sort Complexity

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Best Case?

Worst Case?

Bubble Sort Complexity

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Outer Loop:
no. of iterations depend on the input structure (sorted, reverse sorted, random)

Inner Loop:
runs from N to 1 → O(N)

Bubble Sort Complexity

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Best Case?

input is already sorted → do-while loop will only run once, since no more swaps will happen

O(N) → outer do-while loop is 1 iteration x inner loop is N iterations

Bubble Sort Complexity

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Outer Loop:

C_1 1

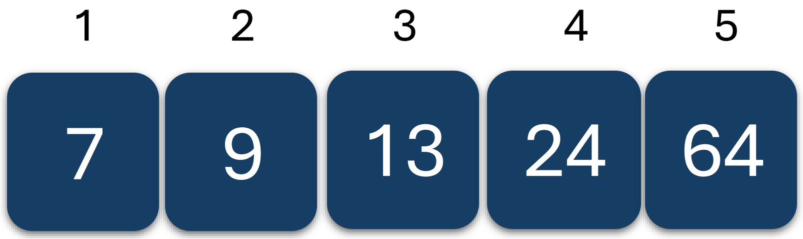
Inner Loop:

C_2 n

$$T(n) = c_1(1)+ c_1(n)$$
$$= O(n)$$

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

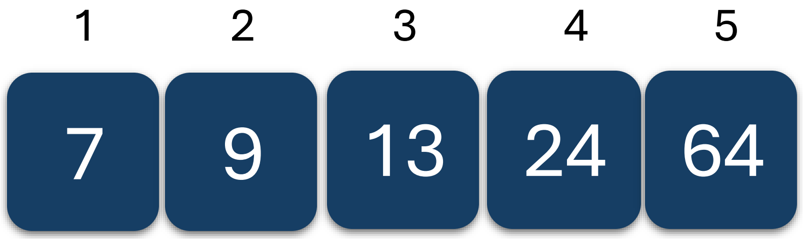


1	2	3	4	5
7	9	13	24	64

Bubble Sort

First pass
Swapped=False

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

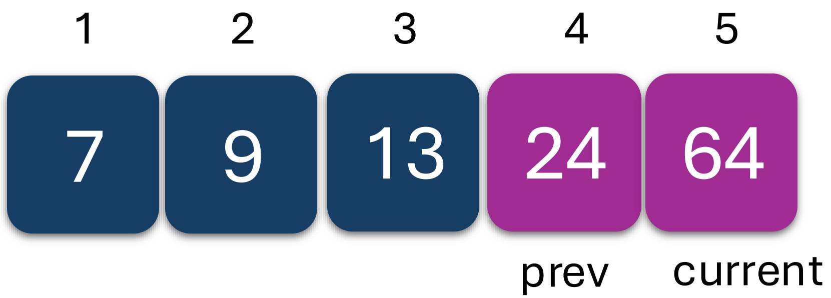


1	2	3	4	5
7	9	13	24	64

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

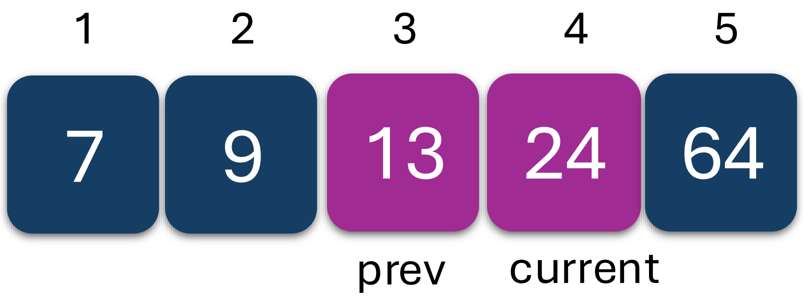
First pass
Swapped=False



1	2	3	4	5
7	9	13	24	64
			prev	current

First pass
Swapped=False

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

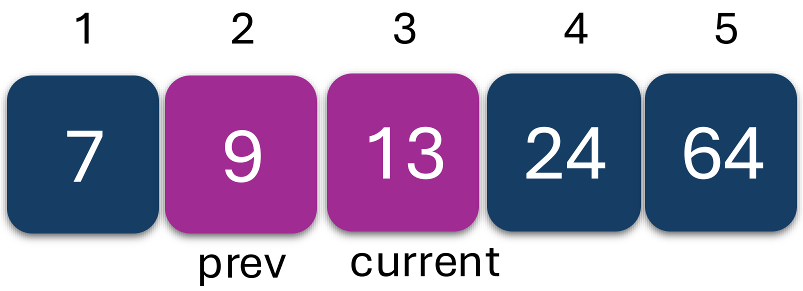


1	2	3	4	5
7	9	13	24	64
		prev	current	

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

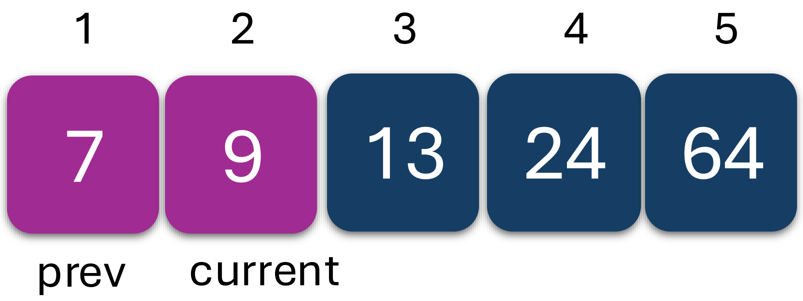
First pass
Swapped=False



Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

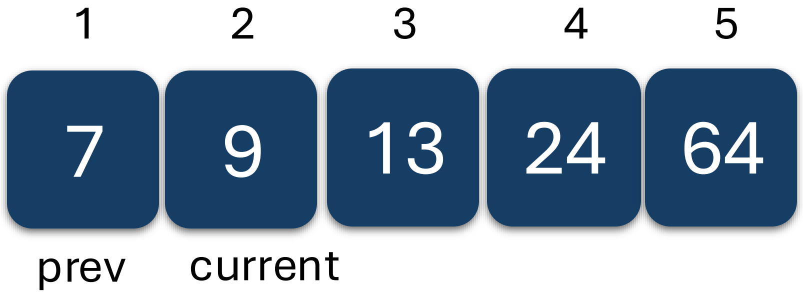
First pass
Swapped=False



Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass
Swapped=False



1	2	3	4	5
7	9	13	24	64
prev	current			

Do while exits since swapped=False

Only 1 pass!

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Worst Case?

- input is reverse sorted
- do-while loop needs N iterations; in each iteration, the ith smallest element will bubble up to the ith-position (correct position)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Outer Loop:
n times

Inner Loop:
runs from N to 1 → O(N)

= $O(n^2)$

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Outer Loop:

C_1 n

Inner Loop:

C_2 n

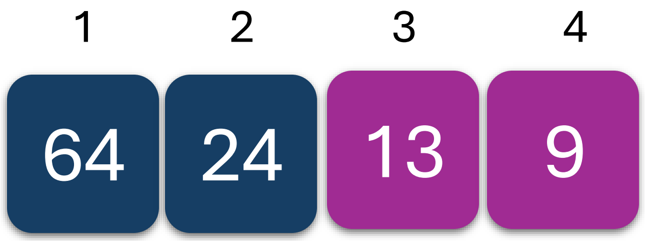
$T(n) = c_1(n) x c_2(n)$

$= O(n^2)$

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass
Swapped=False



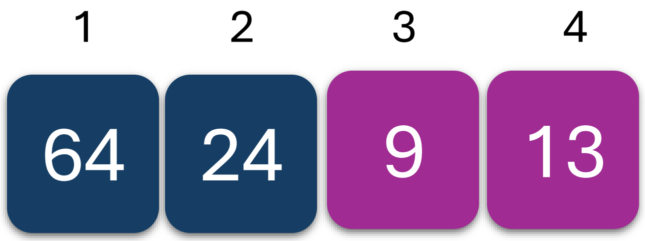
1	2	3	4
64	24	13	9

Number of iterations:
First pass:

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass
Swapped=True



1	2	3	4
64	24	9	13

Number of iterations:
First pass:
(1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass
Swapped=True

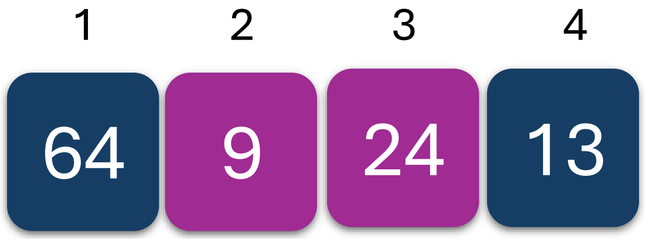
1	2	3	4
64	24	9	13

Number of iterations:
First pass:
(1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass
Swapped=True



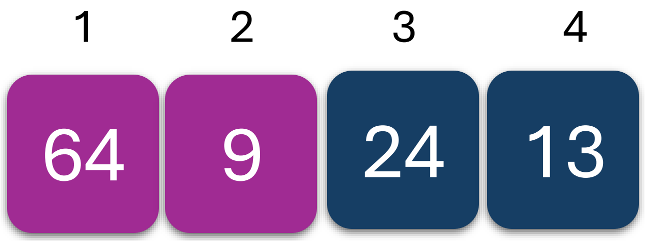
1	2	3	4
64	9	24	13

Number of iterations:
First pass:
(1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass
Swapped=True



1	2	3	4
64	9	24	13

Number of iterations:
First pass:
(1+1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass
Swapped=True

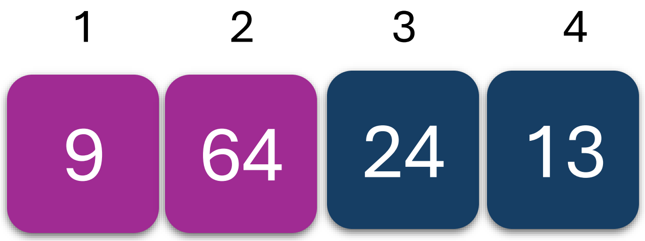
1	2	3	4
64	9	24	13

Number of iterations:
First pass:
(1+1+1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

First pass
Swapped=True



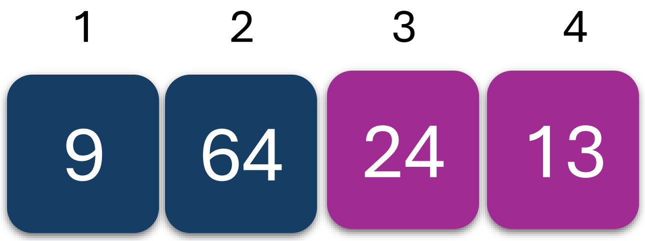
1	2	3	4
9	64	24	13

Number of iterations:
First pass:
(1+1+1+1)=4

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Second pass
Swapped=False



1	2	3	4
9	64	24	13

Number of iterations:
First pass:
4+(1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Second pass
Swapped=True

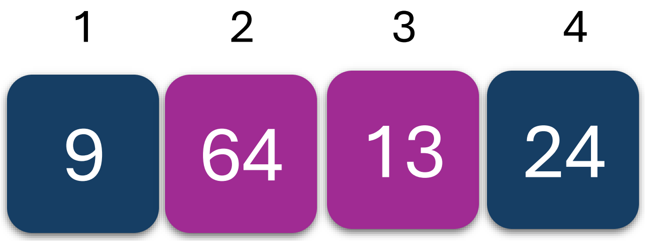
1	2	3	4
9	64	13	24

Number of iterations:
First pass:
4+(1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Second pass
Swapped=True



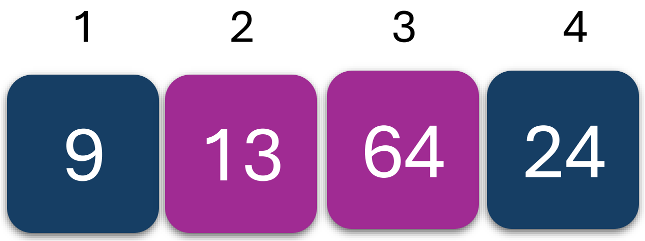
1	2	3	4
9	64	13	24

Number of iterations:
First pass:
4+(1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Second pass
Swapped=True



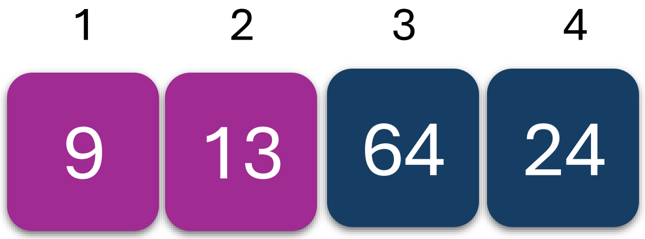
1	2	3	4
9	13	64	24

Number of iterations:
First pass:
4+(1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Second pass
Swapped=True



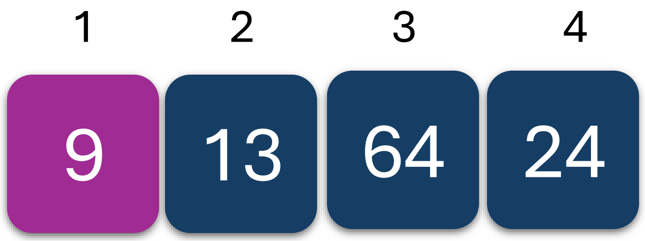
1	2	3	4
9	13	64	24

Number of iterations:
First pass:
4+(1+1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Second pass
Swapped=True



1	2	3	4
9	13	64	24

Number of iterations:
First pass:
4+4

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Third pass
Swapped=False

1	2	3	4
9	13	64	24

Number of iterations:
First pass:
4+4+(1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Third pass
Swapped=True

1	2	3	4
9	13	24	64

Number of iterations:
First pass:
4+4+(1)

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Third pass
Swapped=True

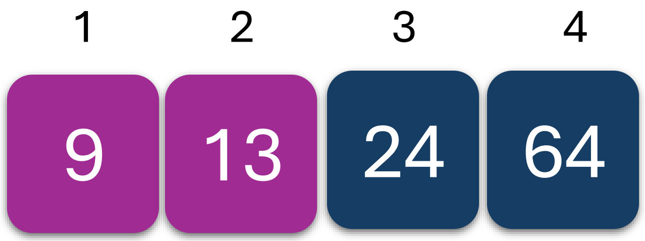
1	2	3	4
9	13	24	64

Number of iterations:
First pass:
4+4+(1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Third pass
Swapped=True

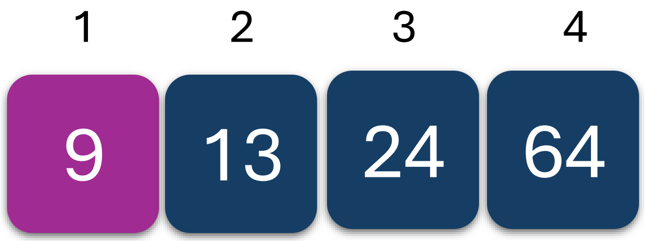


Number of iterations:
First pass:
4+4+(1+1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Third pass
Swapped=True



1	2	3	4
9	13	24	64

Number of iterations:
First pass:
4+4+(1+1+1+1)

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

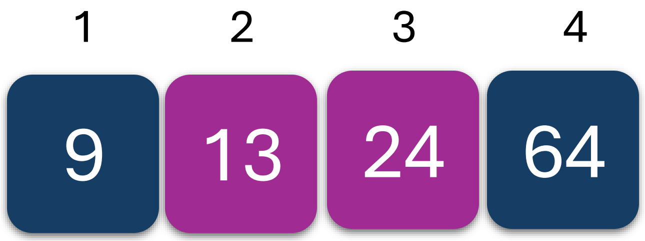
Fourth pass
Swapped=False

1	2	3	4
9	13	24	64

Number of iterations:
First pass:
4+4+4+(1)

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Fourth pass
Swapped=False



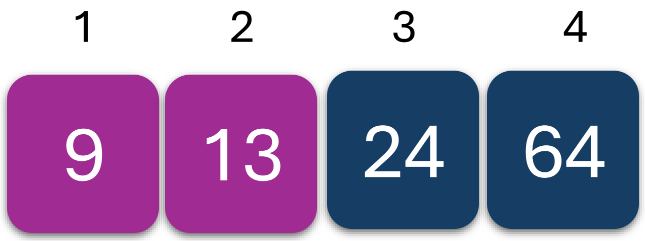
1	2	3	4
9	13	24	64

Number of iterations:
First pass:
4+4+4+(1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Fourth pass
Swapped=False



1	2	3	4
9	13	24	64

Number of iterations:
First pass:
4+4+4+(1+1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Fourth pass
Swapped=False

1	2	3	4
9	13	24	64

Number of iterations:
First pass:
4+4+4+(1+1+1+1)

Bubble Sort

```
bubble_sort(array A):
    do:
        swapped = False
        for current = N to 1:
            prev = current – 1
            if A[prev] > A[current]:
                swap A[prev] ↔ A[current]
                swapped = True
    while swapped = True
```

Fourth pass
Swapped=False

1	2	3	4
9	13	24	64

Total Number of iterations:
4 passes with 4 iterations each

Bubble Sort Complexity

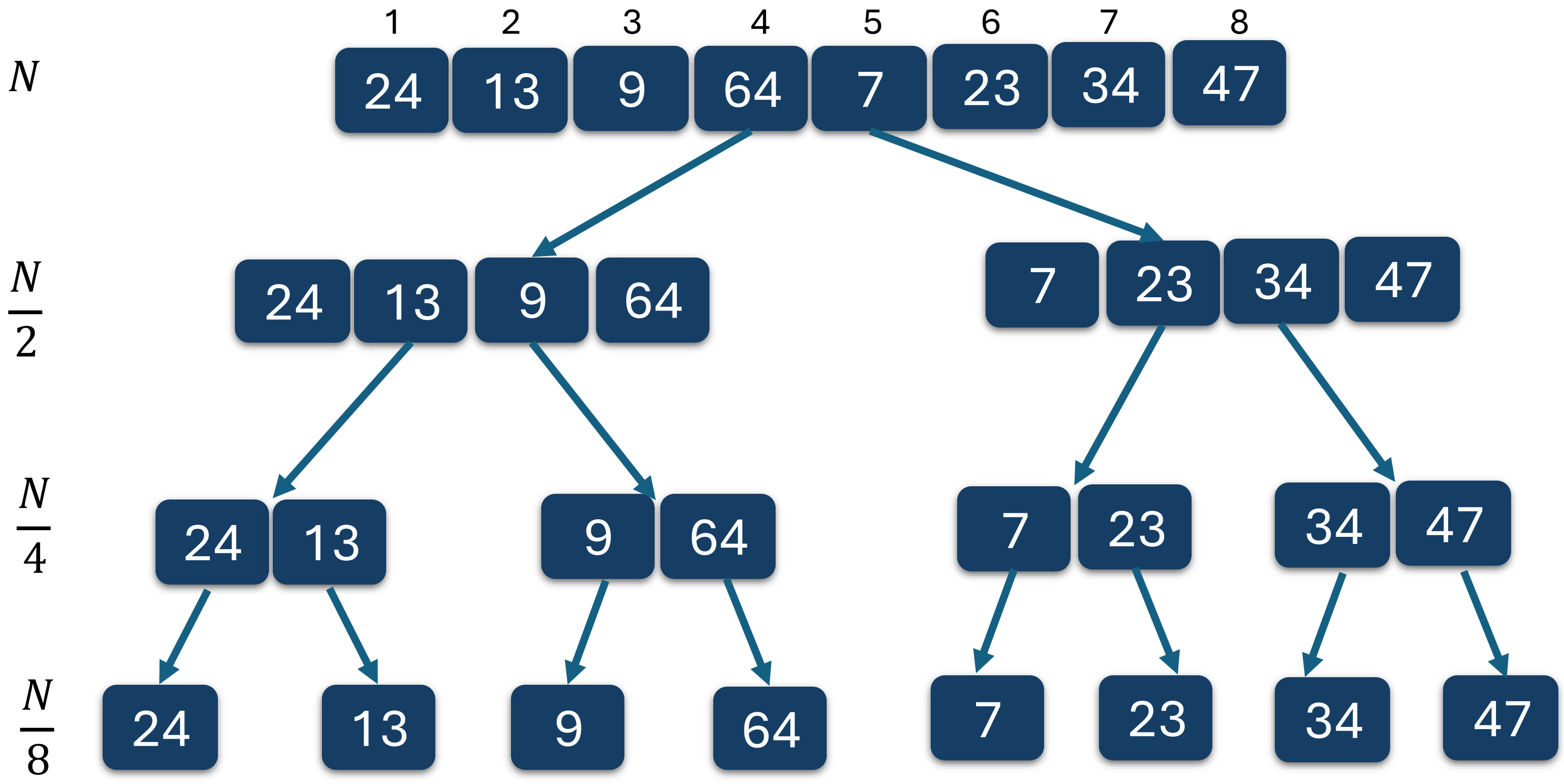
- Memory: O(1) → no extra memory needed

Merge Sort

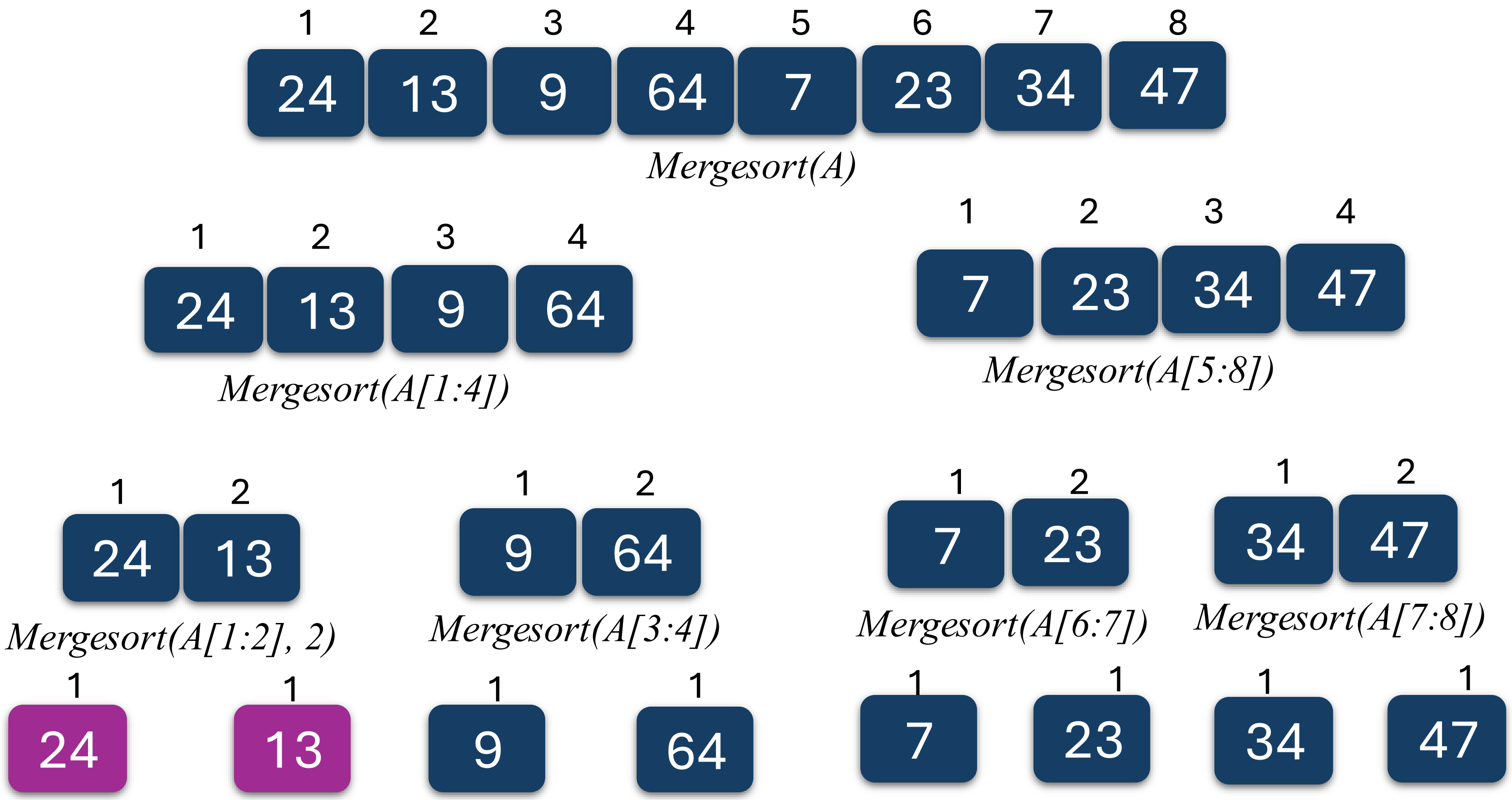
Merge Sort

- Uses **divide-and-conquer** approach; has a recursive structure
  - **Divide:** Break problem into subproblems (similar to the original problem, but smaller in size)
  - **Conquer:** Solve the subproblems recursively
  - **Merge**: Combine subproblems' solutions to create solution to original problem
- Divide step is trivial through recursion; Merge step is where sorting happens
- During Merge, we compare the first elements of the left subproblem's solution and the right subproblem's solution → whichever is smaller gets taken first

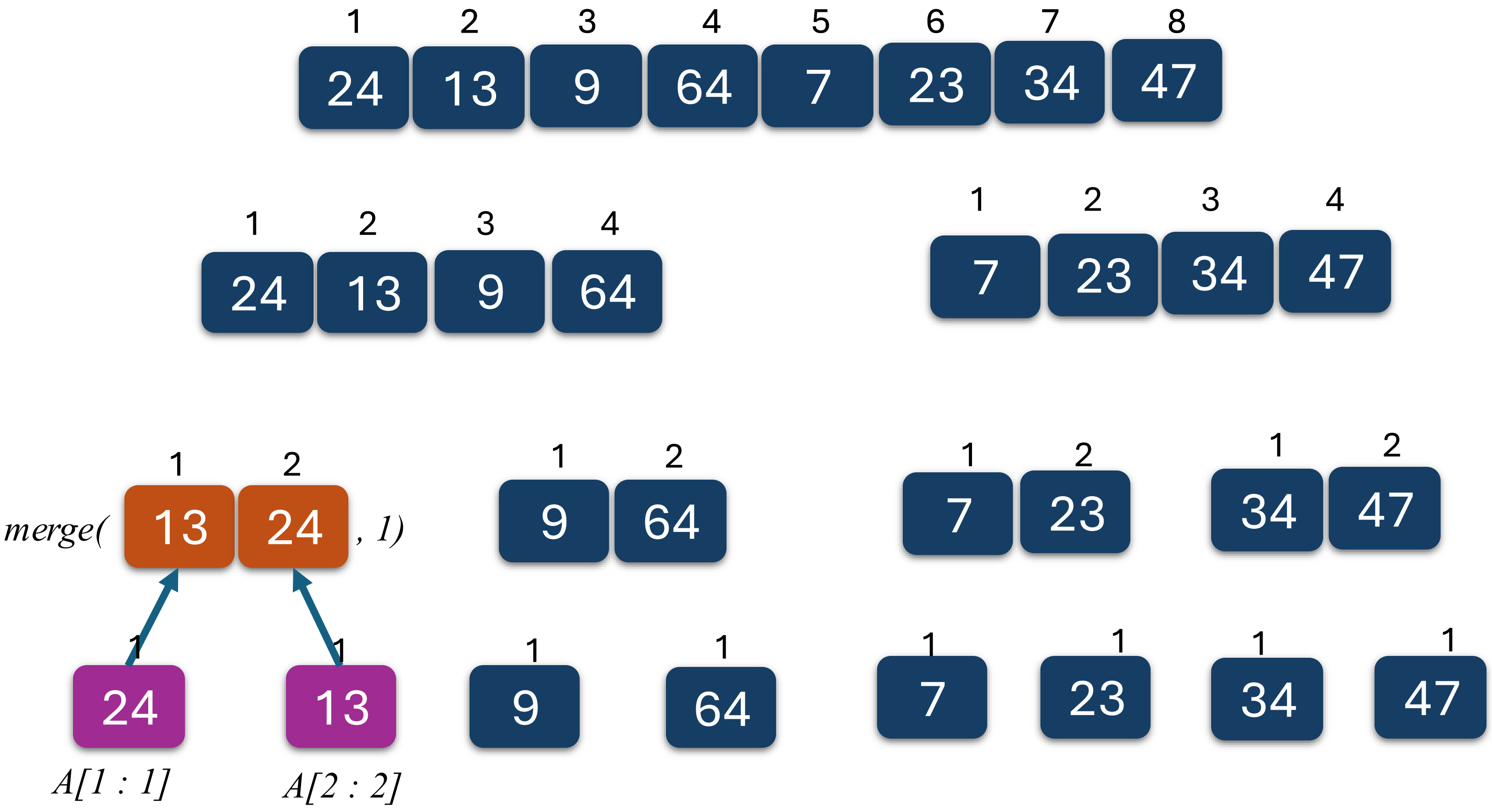
```
merge_sort(array A):
    if $N > 1$:
        $M = N / 2$
        merge_sort(A[1 : M])
        merge_sort(A[M+1 : N])
        merge(A[1:N], M)
```



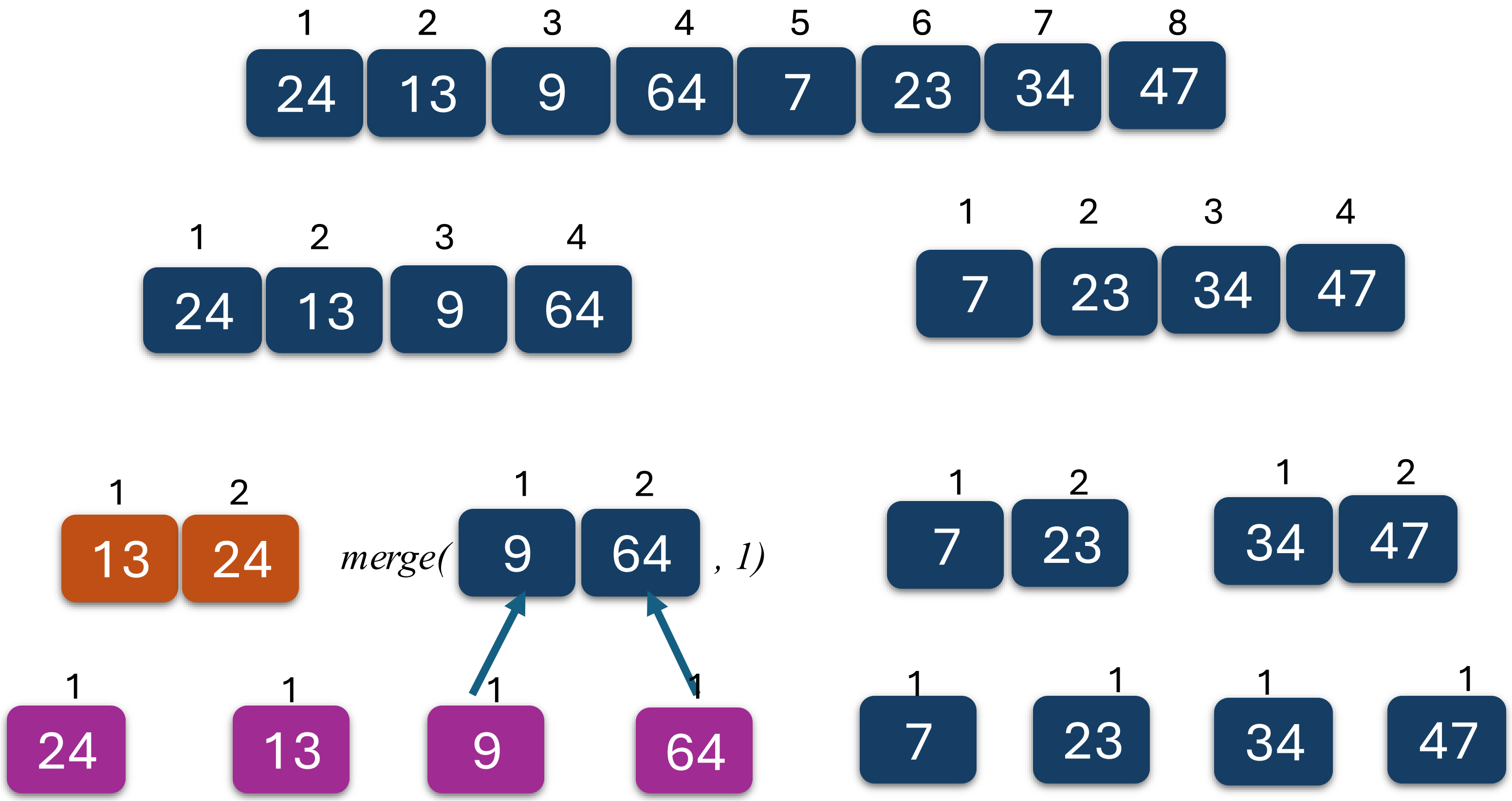
```
merge_sort(array A):
    if N > 1:
        M = N / 2
        merge_sort(A[1 : M])
        merge_sort(A[M+1 : N])
        merge(A[1:N], M)
```



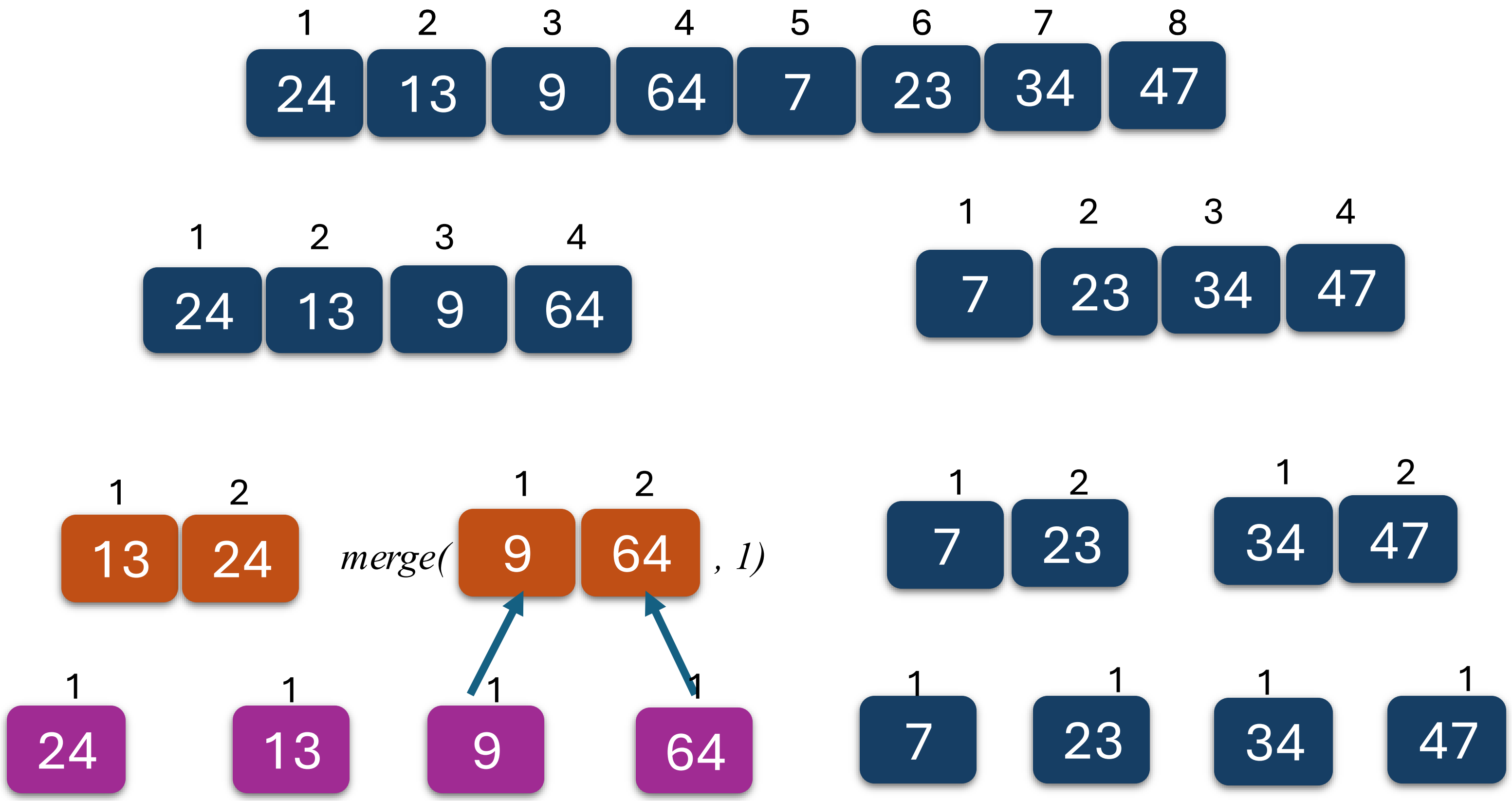
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



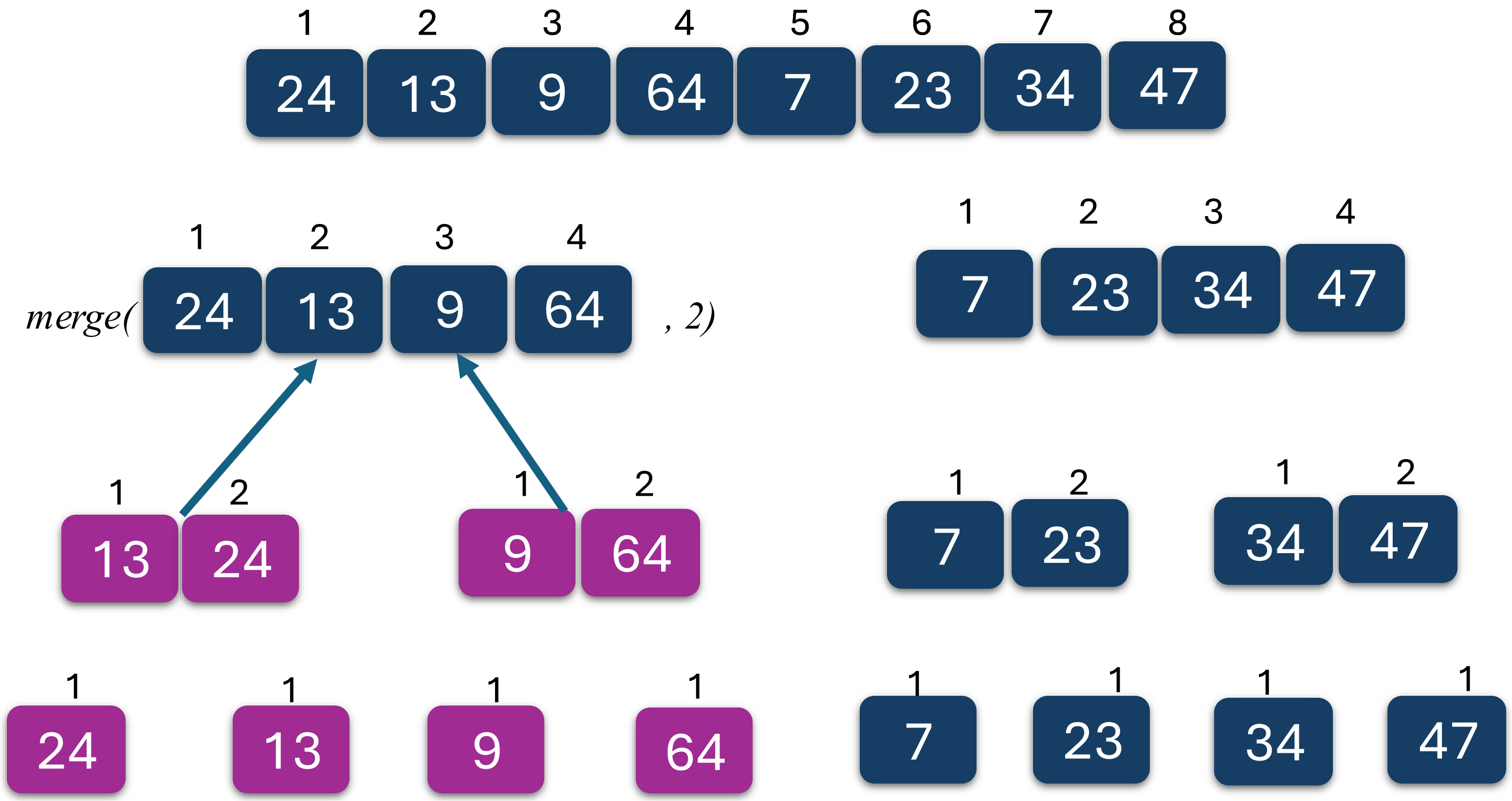
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



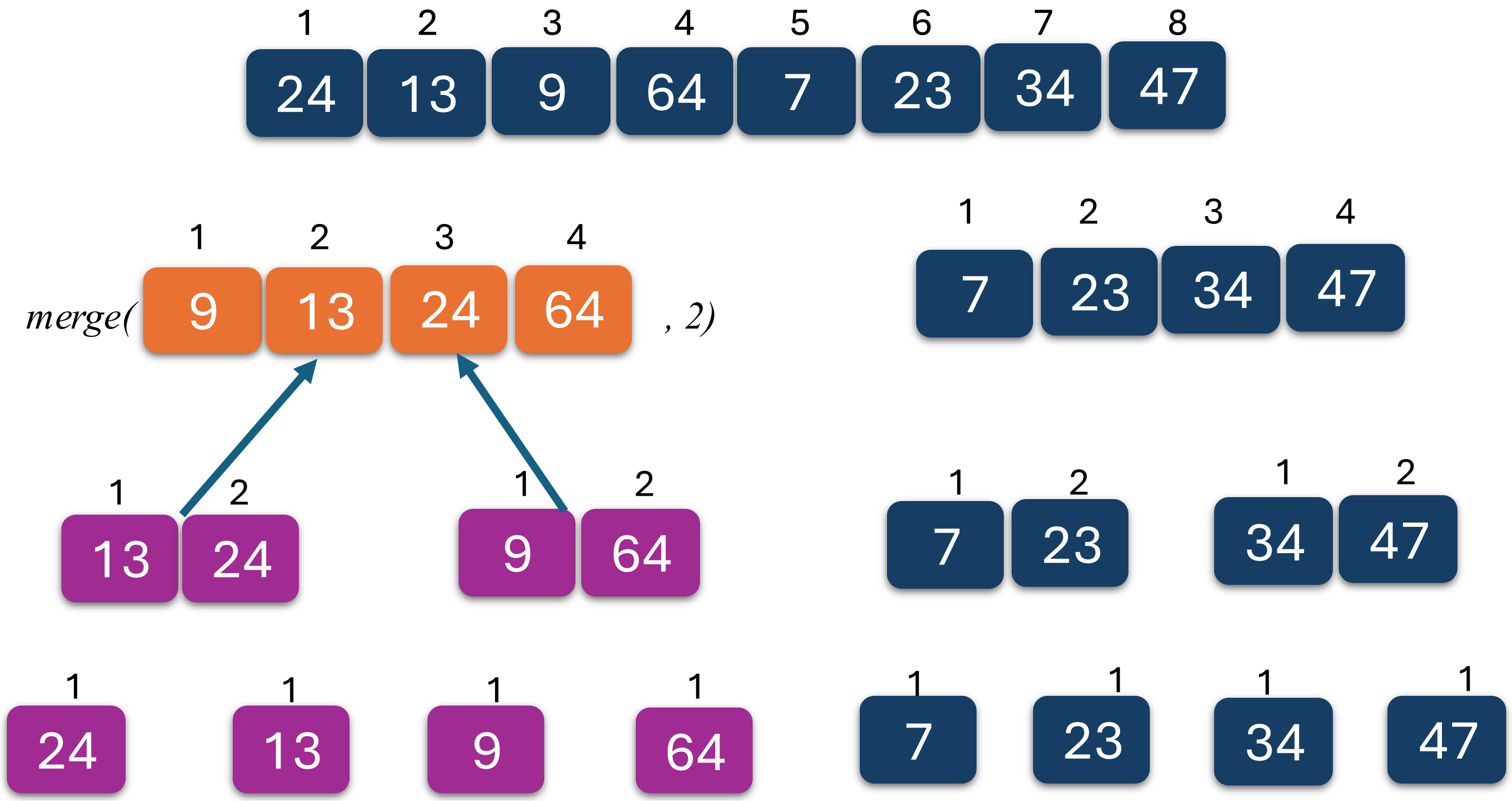
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



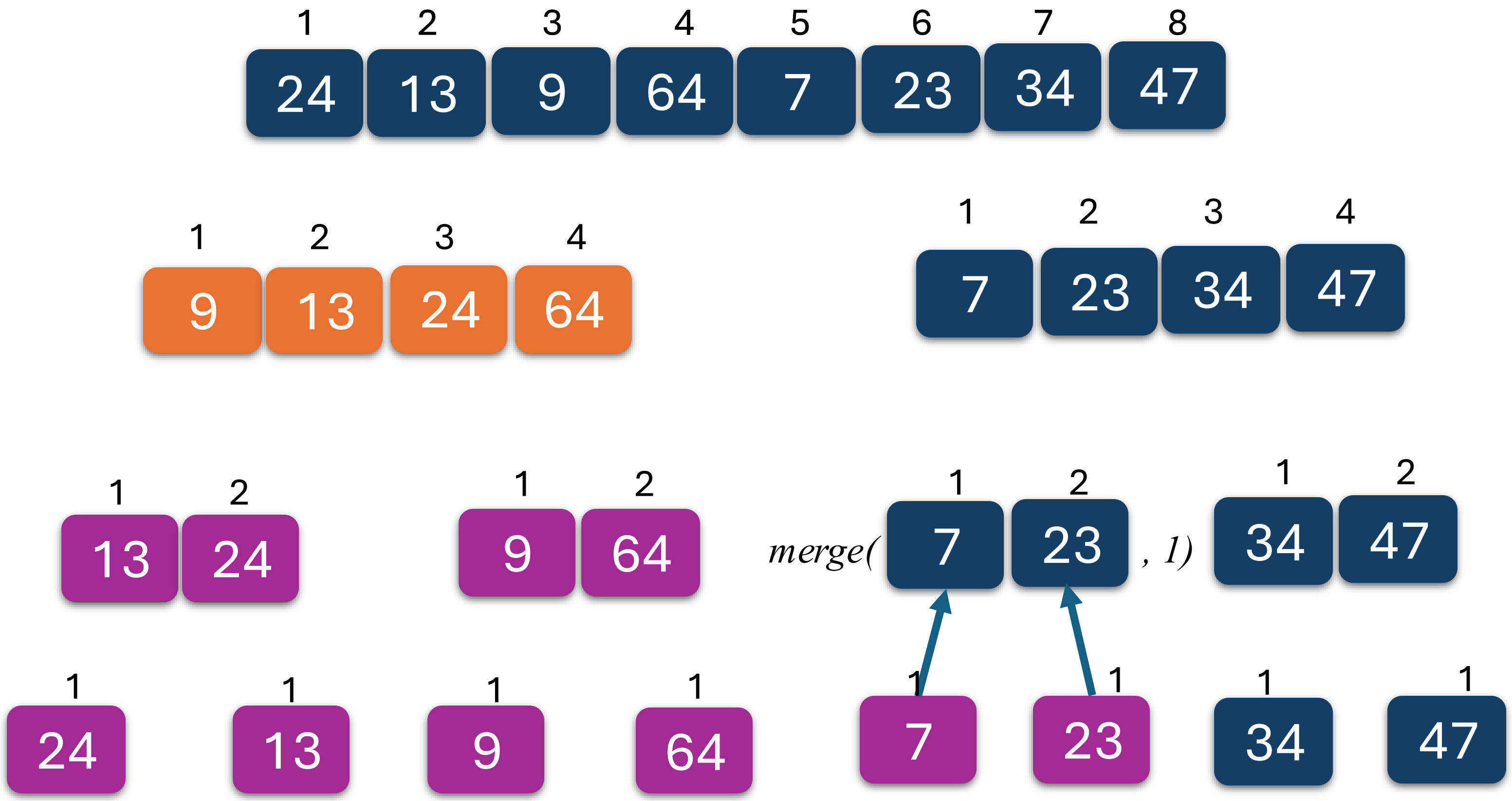
```
merge(array A, integer M):
        left_half = A[1 : M]
        right_half = A[M+1 : N]
        for i = 1 to N:
                L = first item of left_half
                R = first item of right_half
                B[i] = min(L, R)
        copy B to A
```



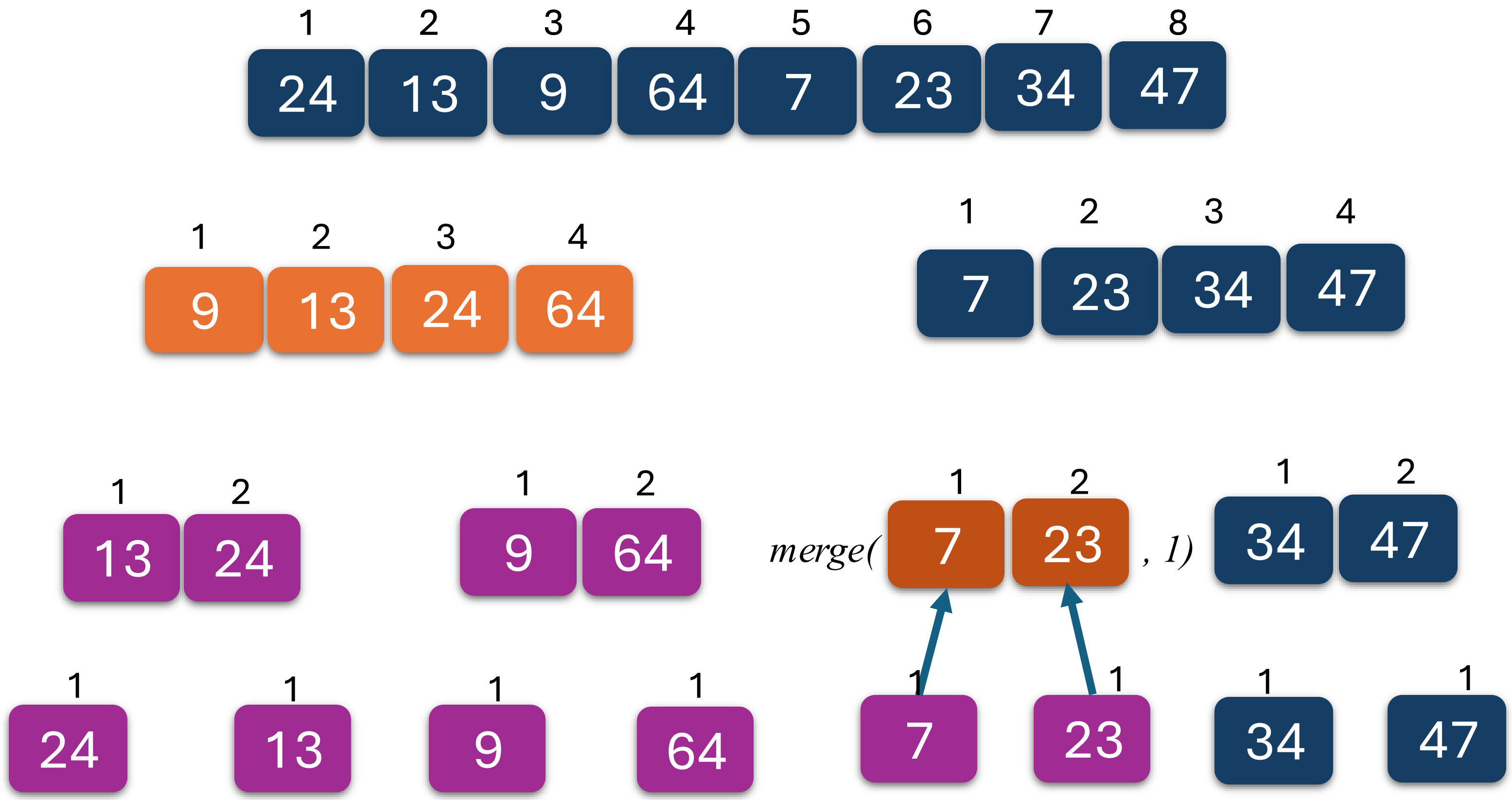
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



1 2 3 4 5 6 7 8

24 13 9 64 7 23 34 47

1 2 3 4

9 13 24 64

1 2 3 4

7 23 34 47

1 2

13 24

1 2

9 64

1 2

merge(7 23 *, 1)*

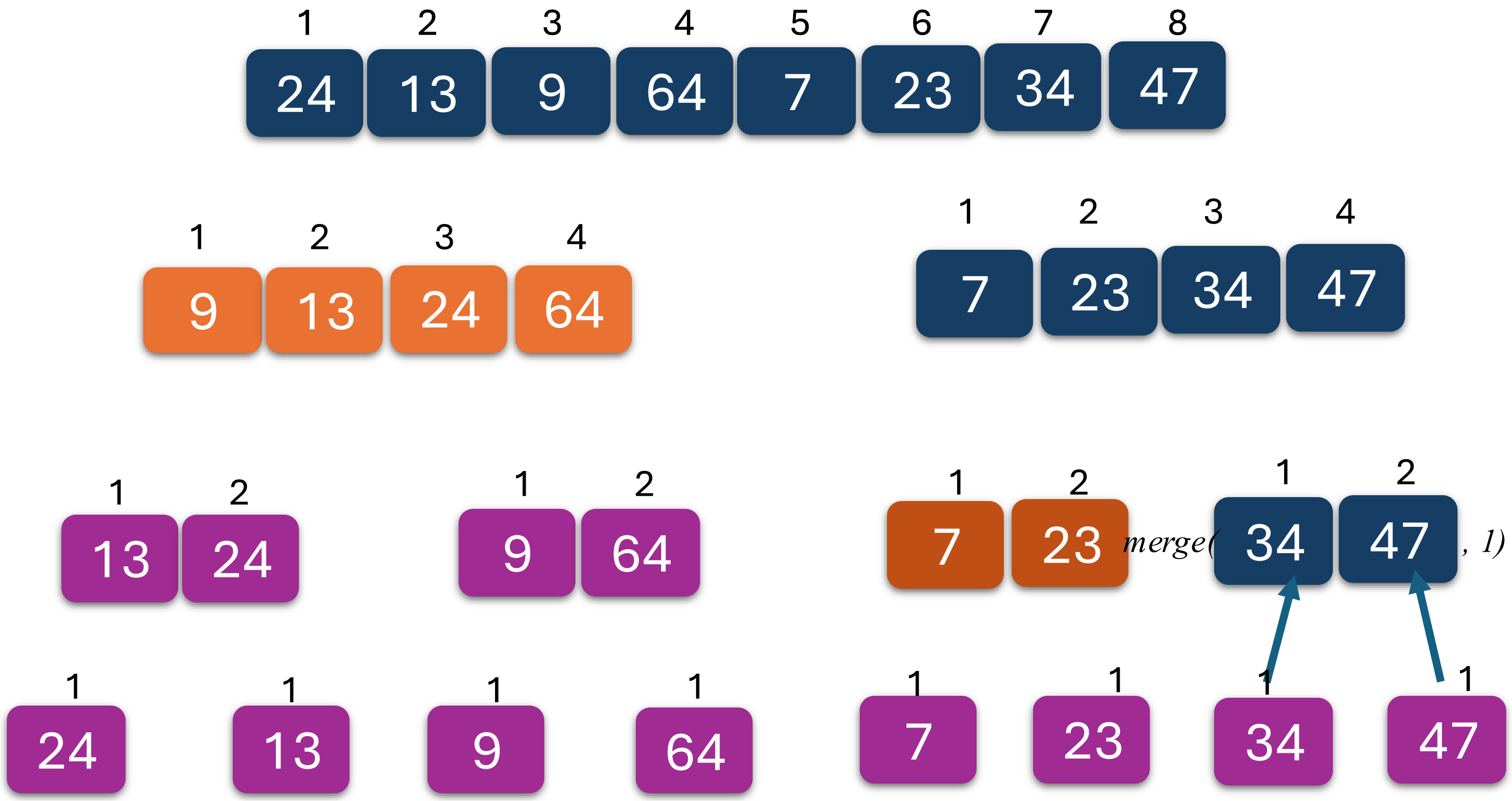
1 2

34 47

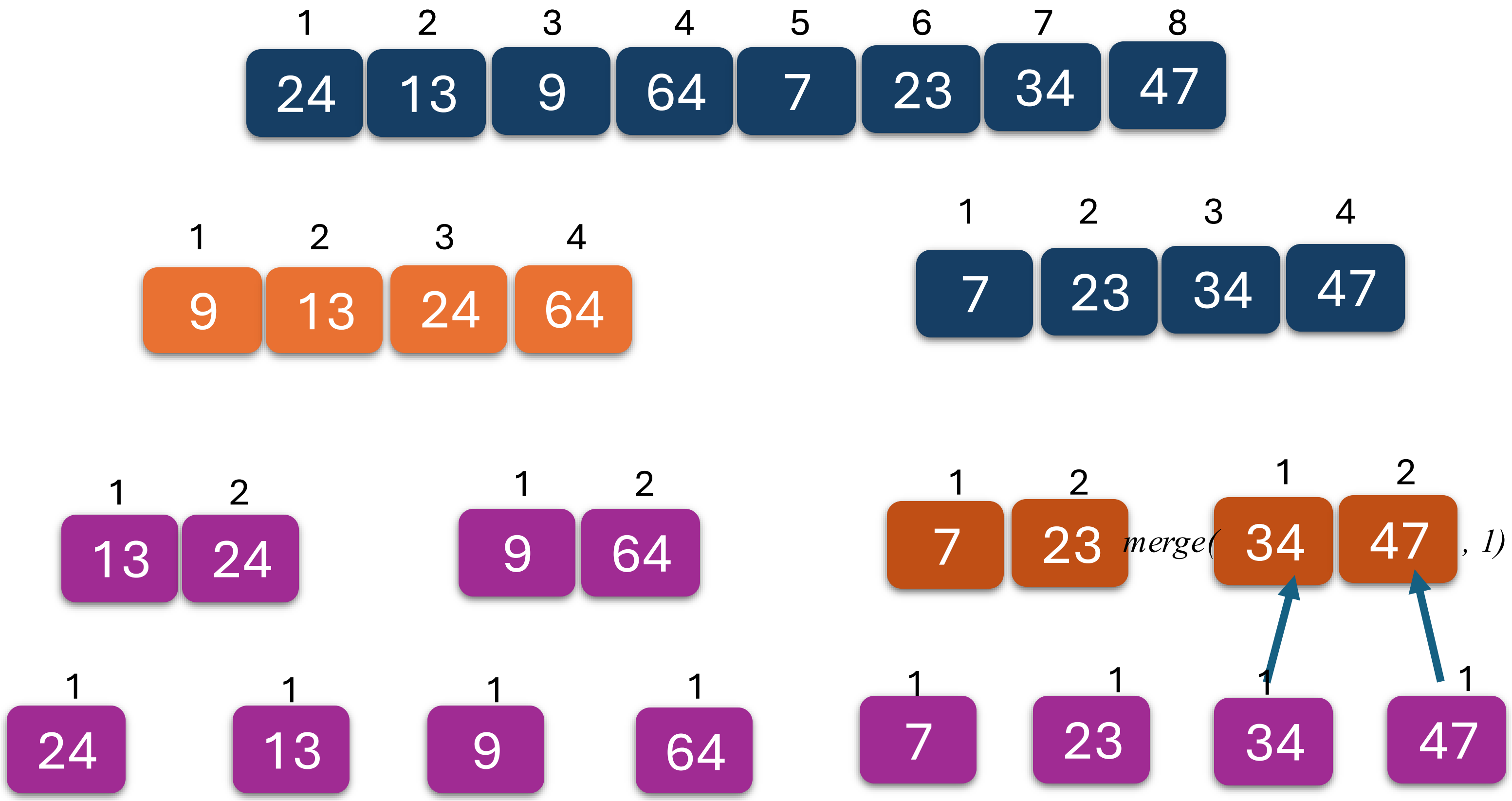
1 1 1 1 1 1 1 1

24 13 9 64 7 23 34 47

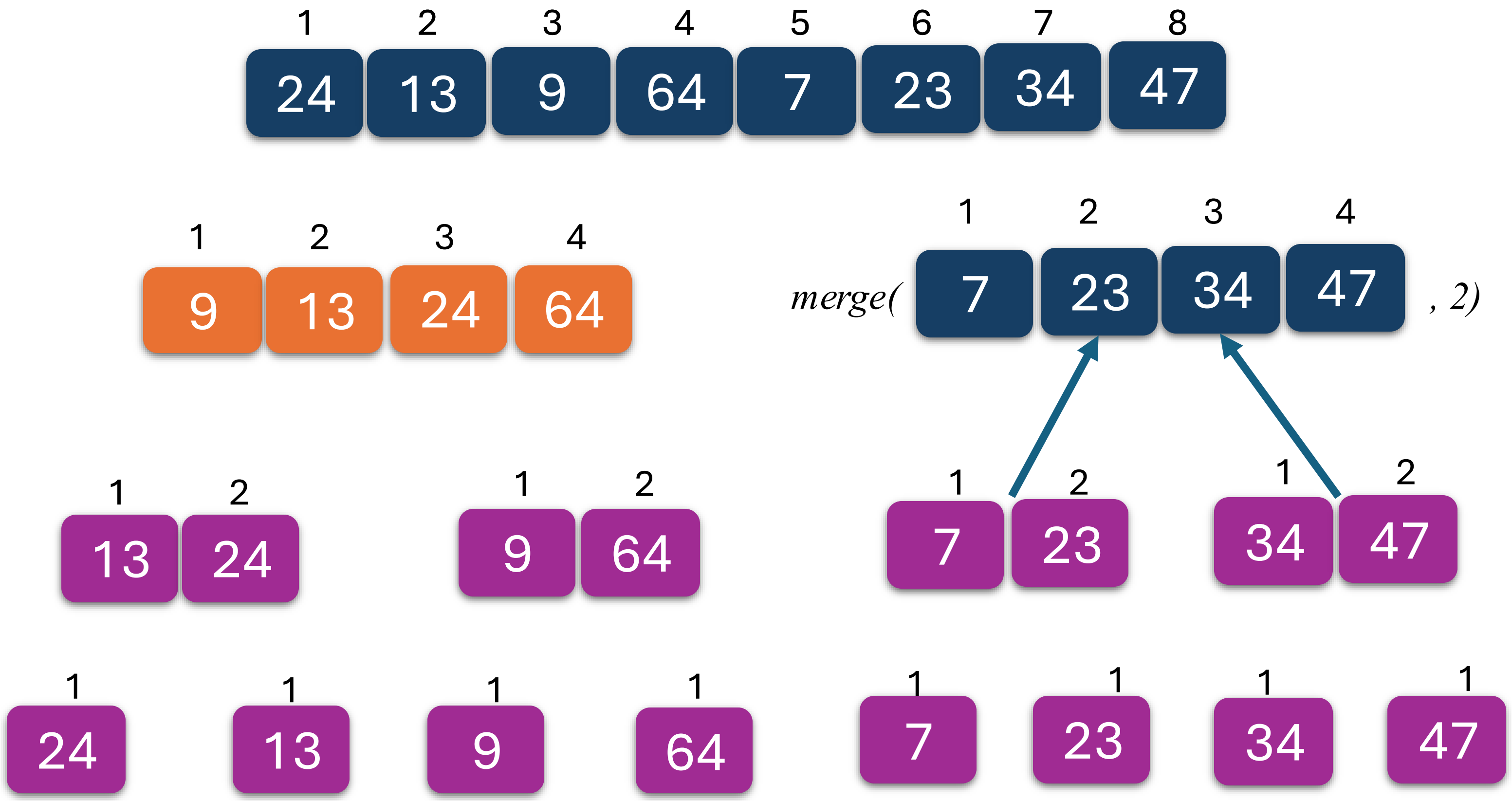
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



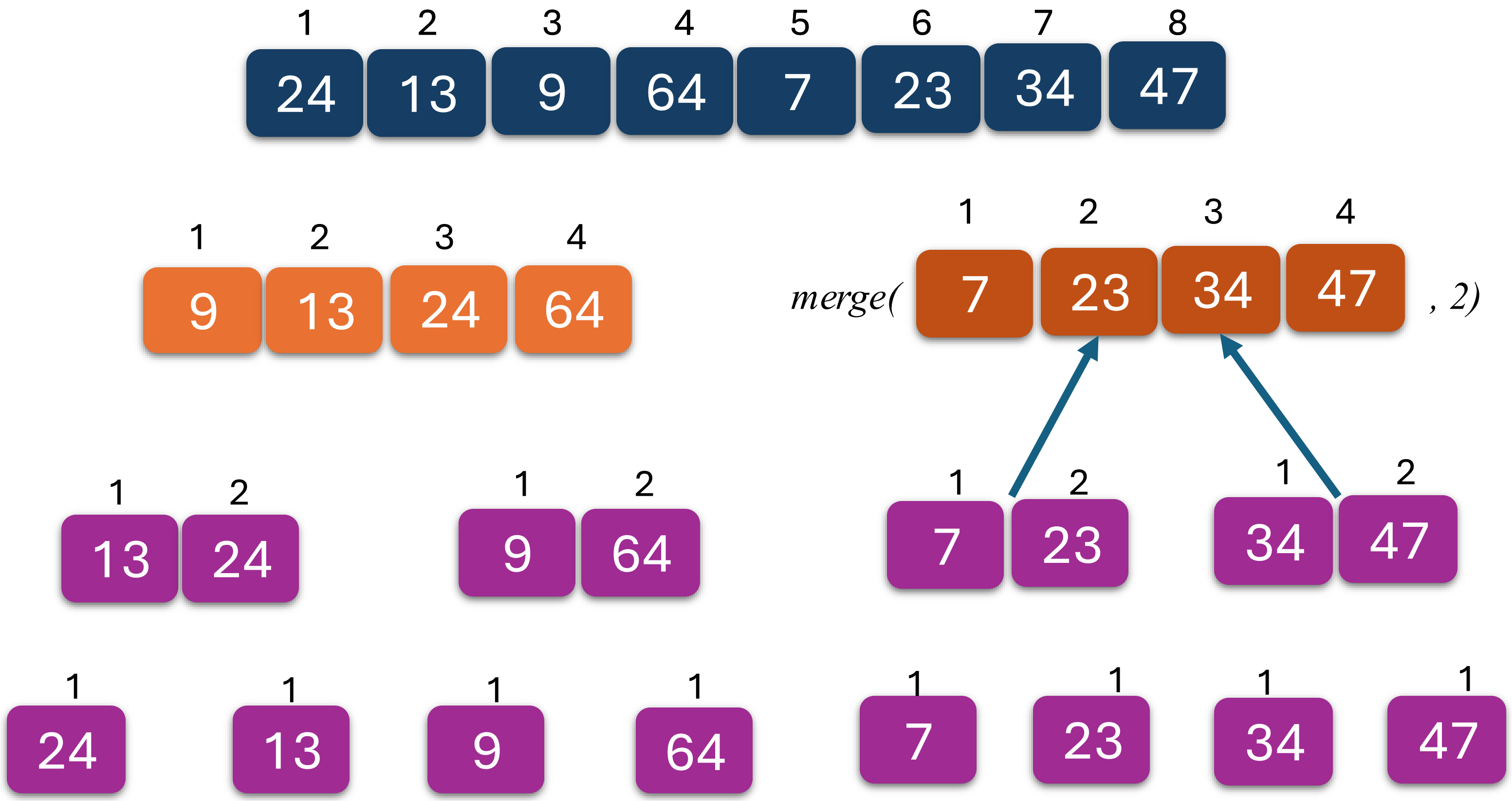
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



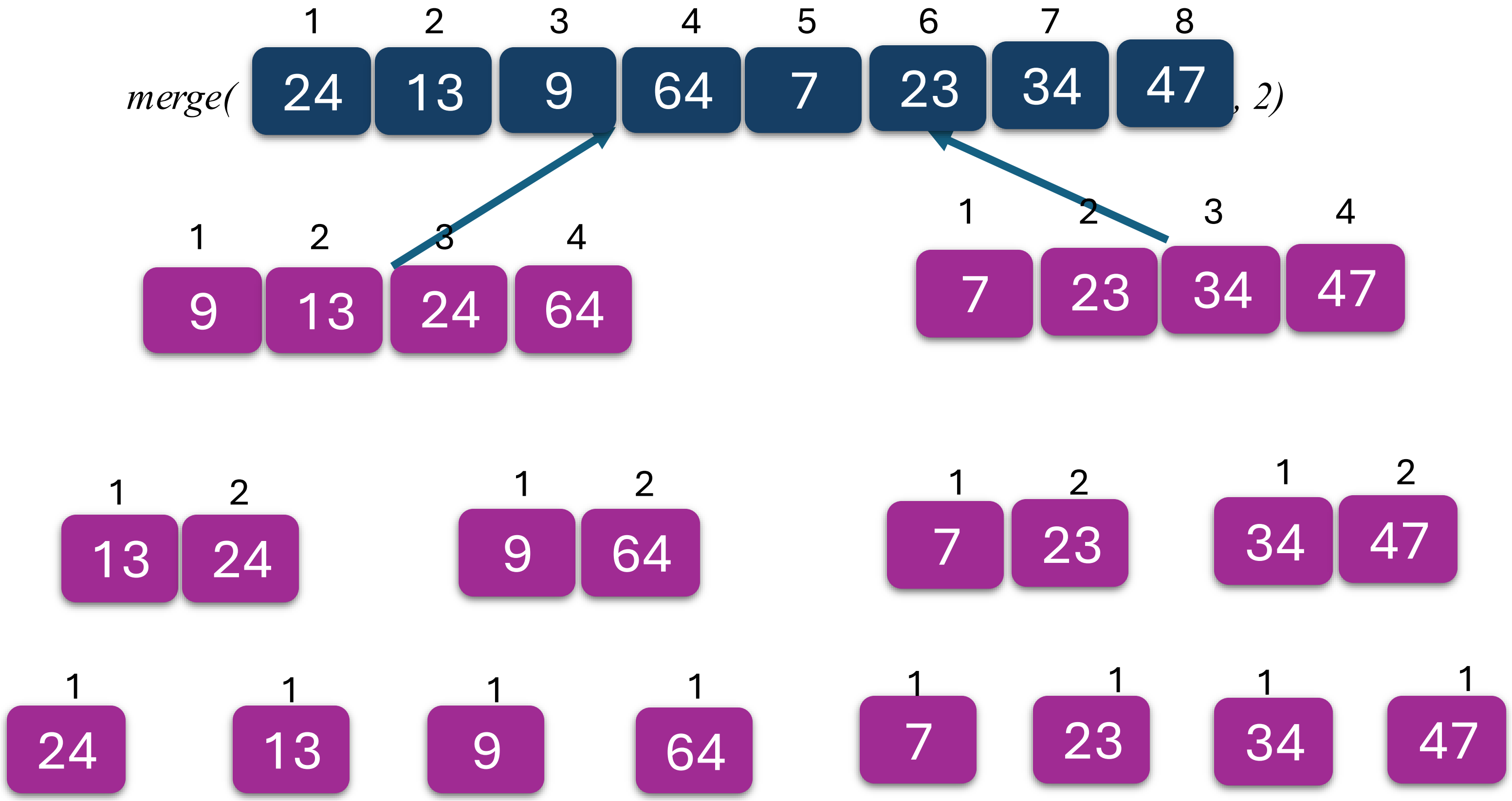
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



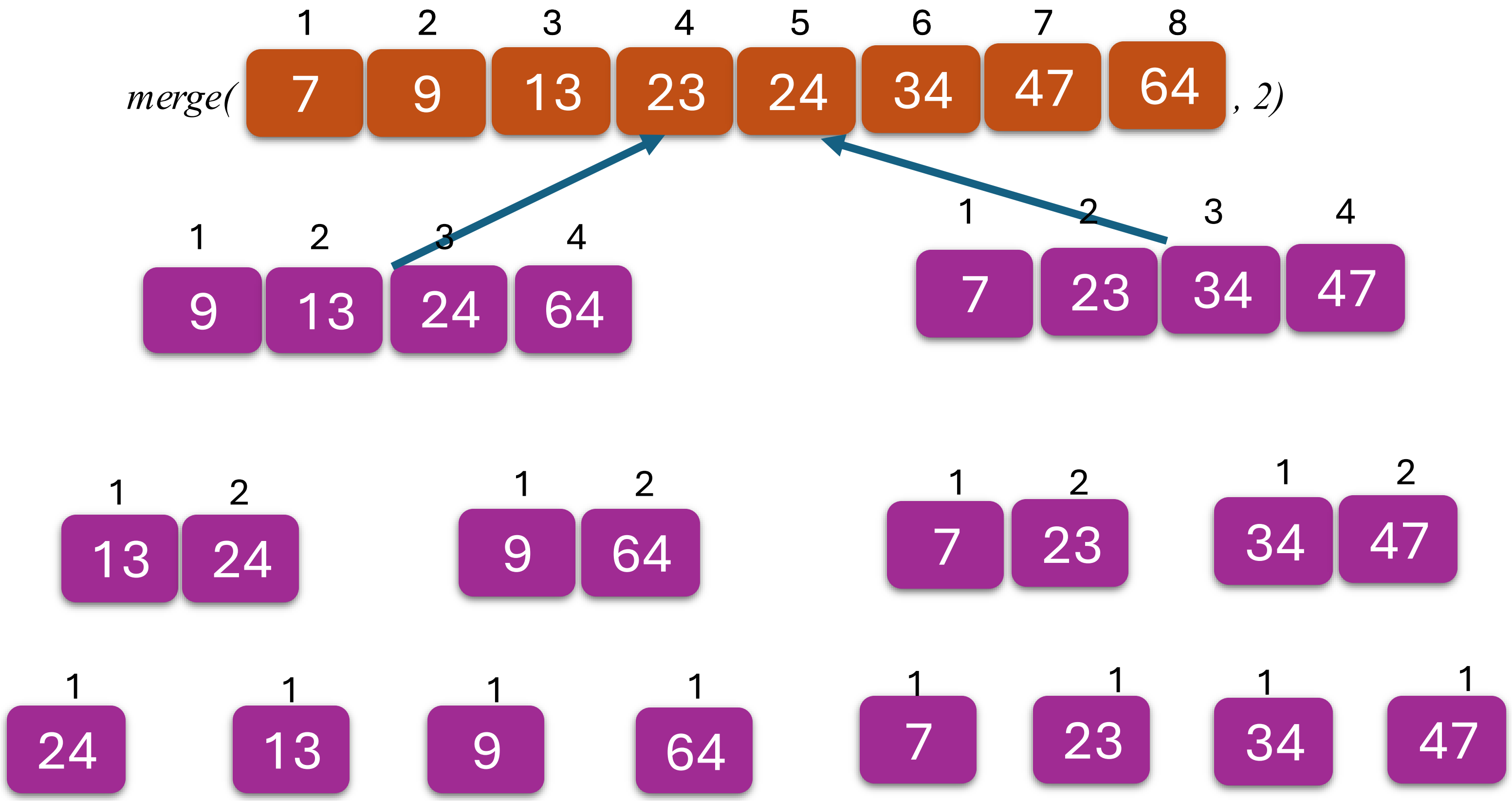
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



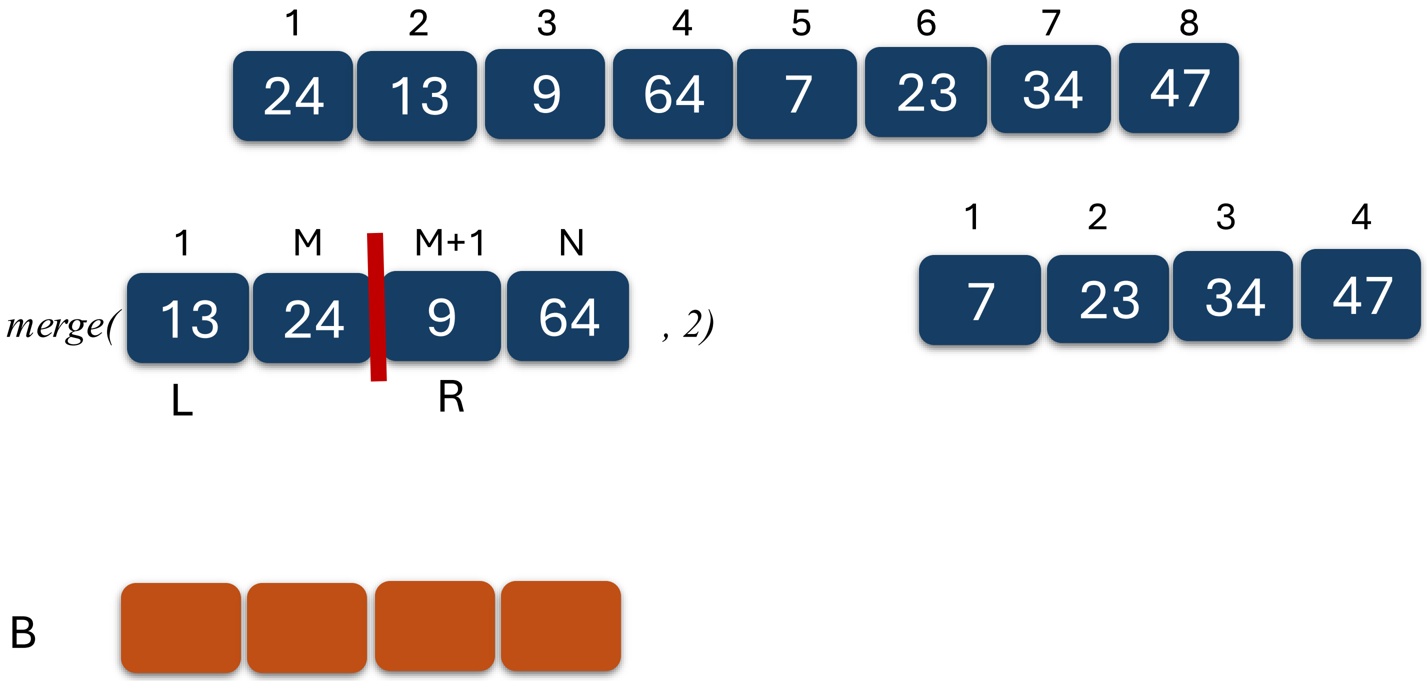
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```



Let’s explore more the “merge” part

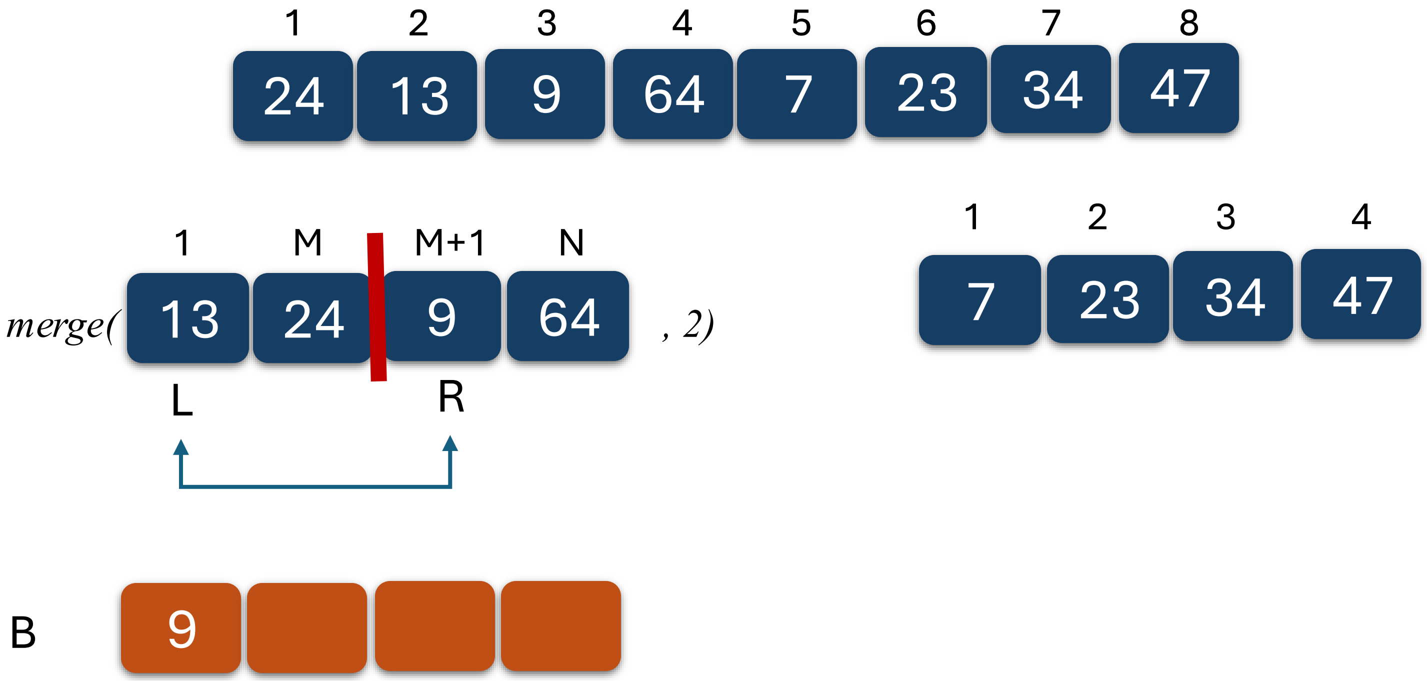
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] < A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



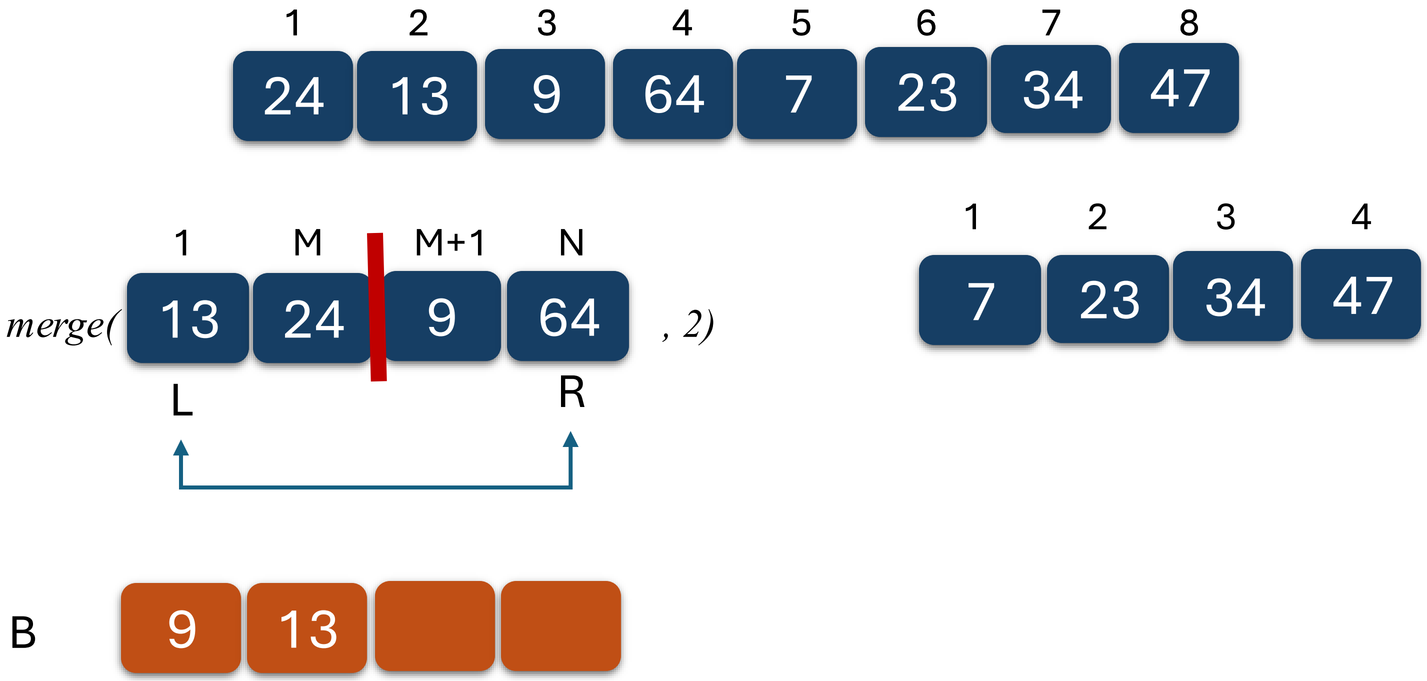
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



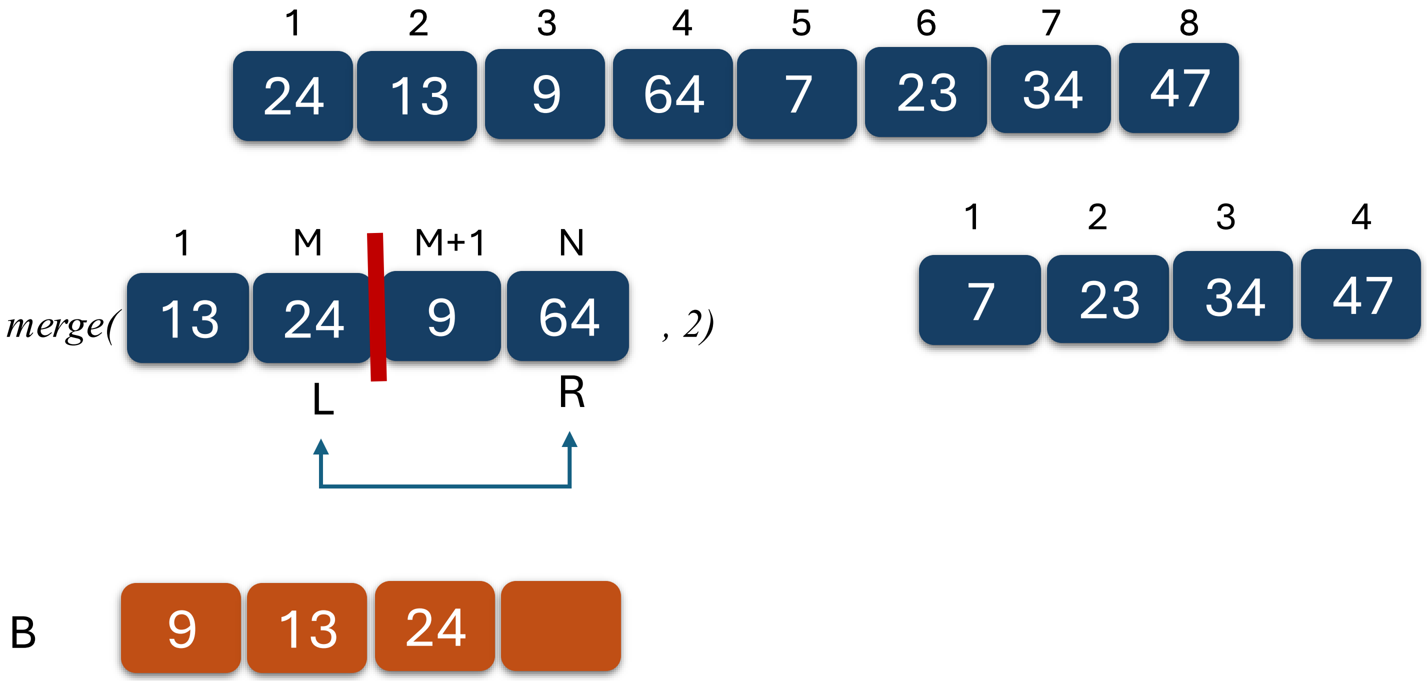
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] < A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



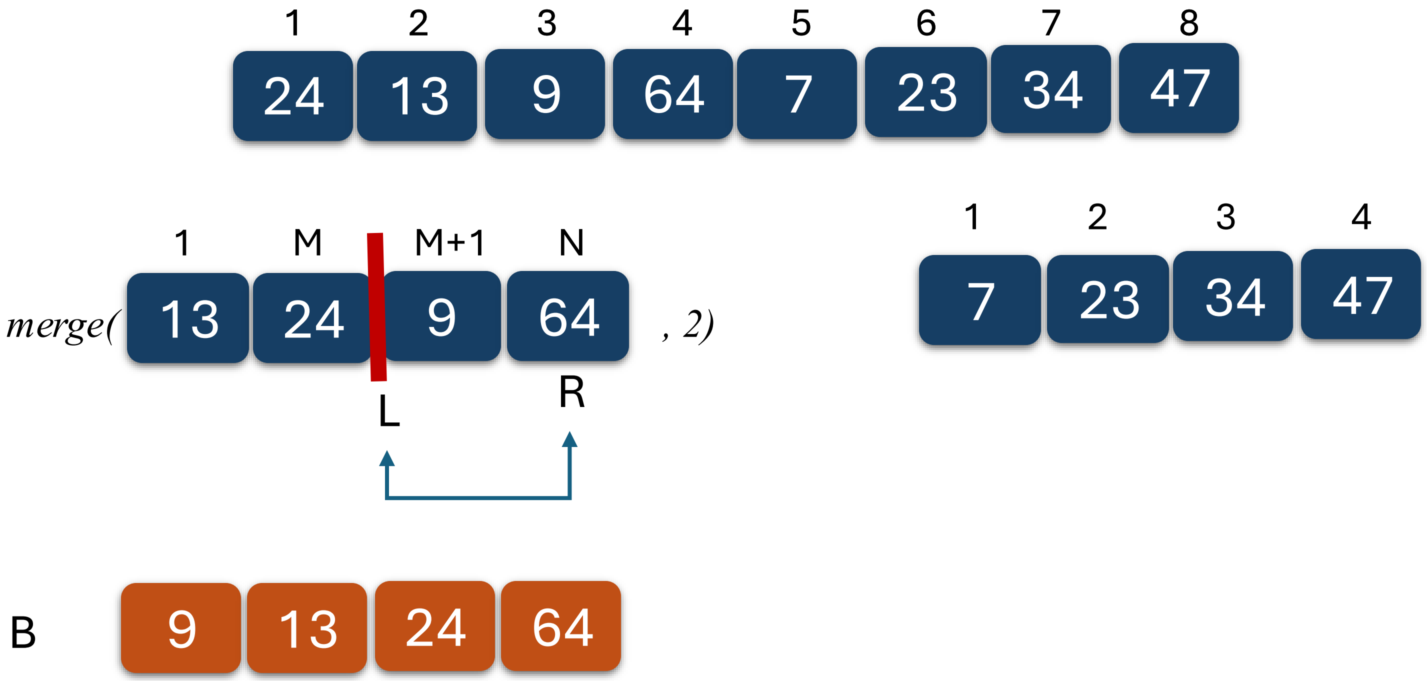
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



1	2	3	4	5	6	7	8
24	13	9	64	7	23	34	47

merge(

1	M	M+1	N
13	24	9	64

, 2)

L R

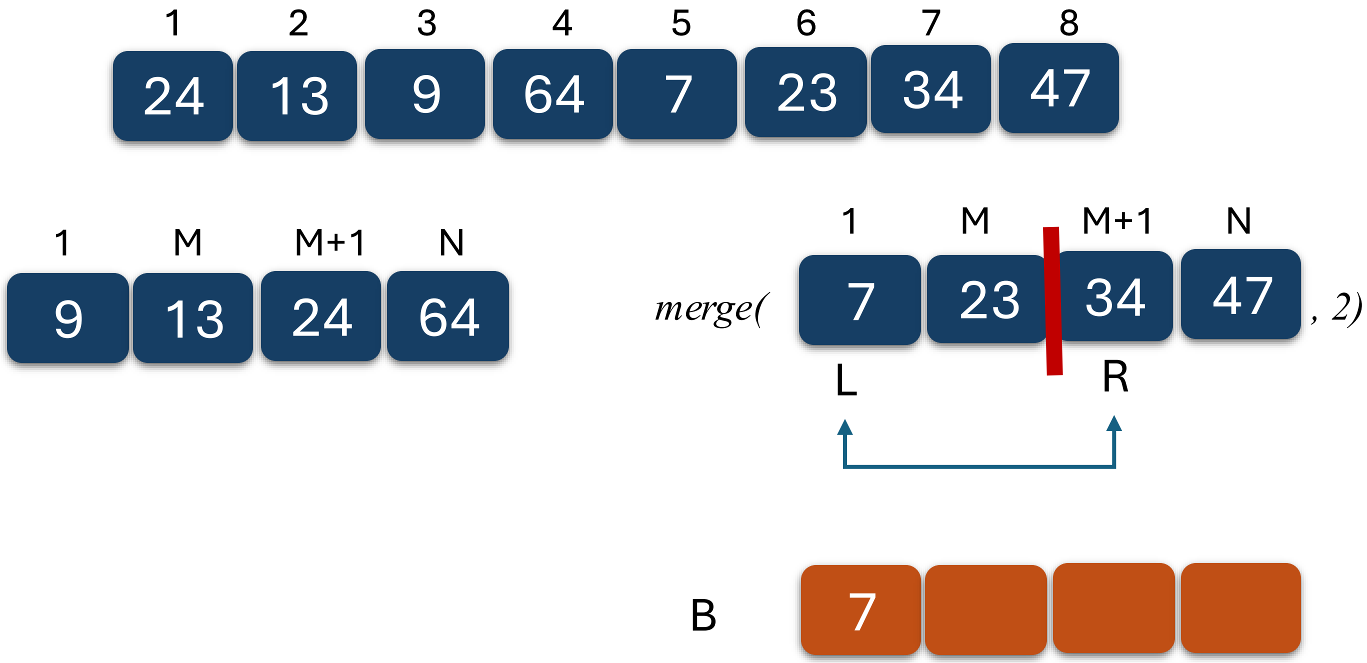
1	2	3	4
7	23	34	47

B

9	13	24	64

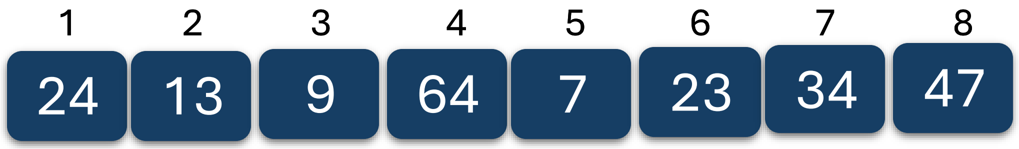
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

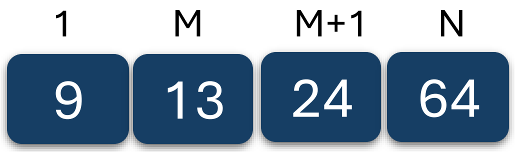
```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
        copy B to A
```

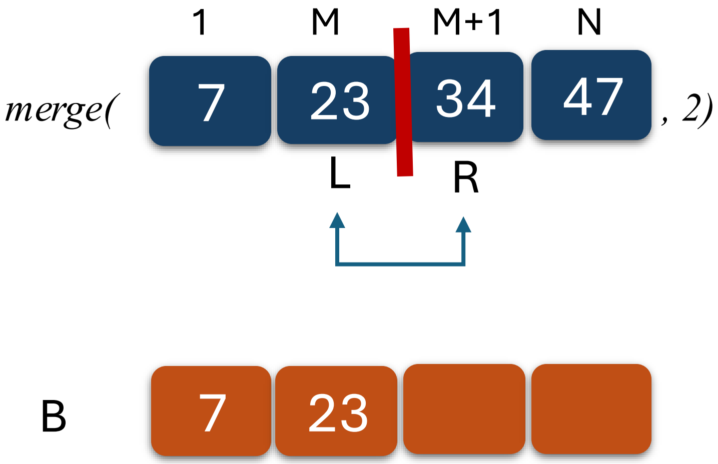


```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```

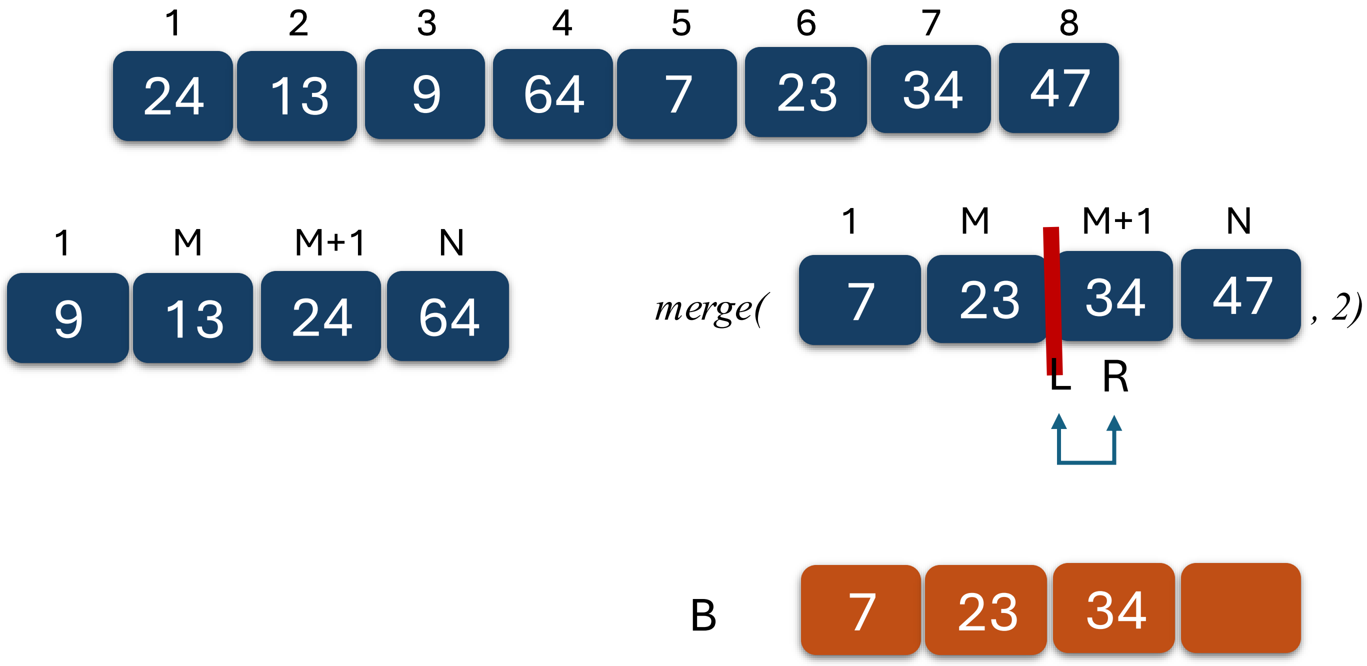






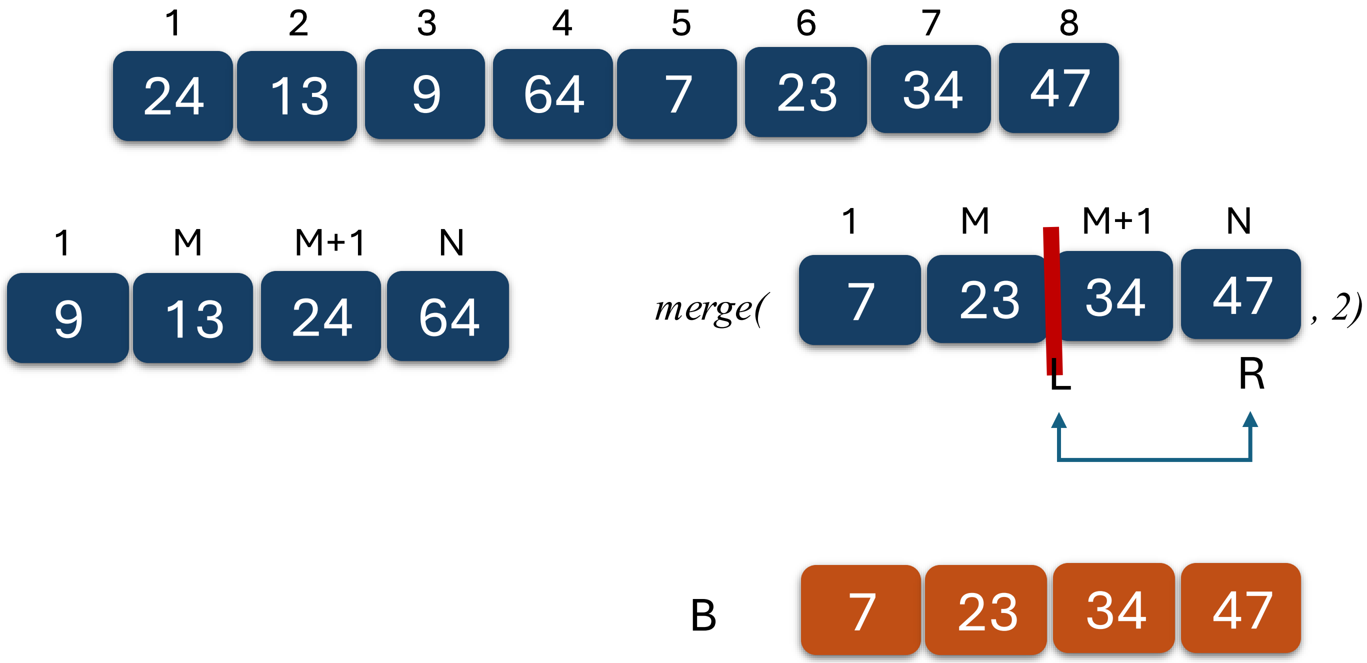
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <   A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



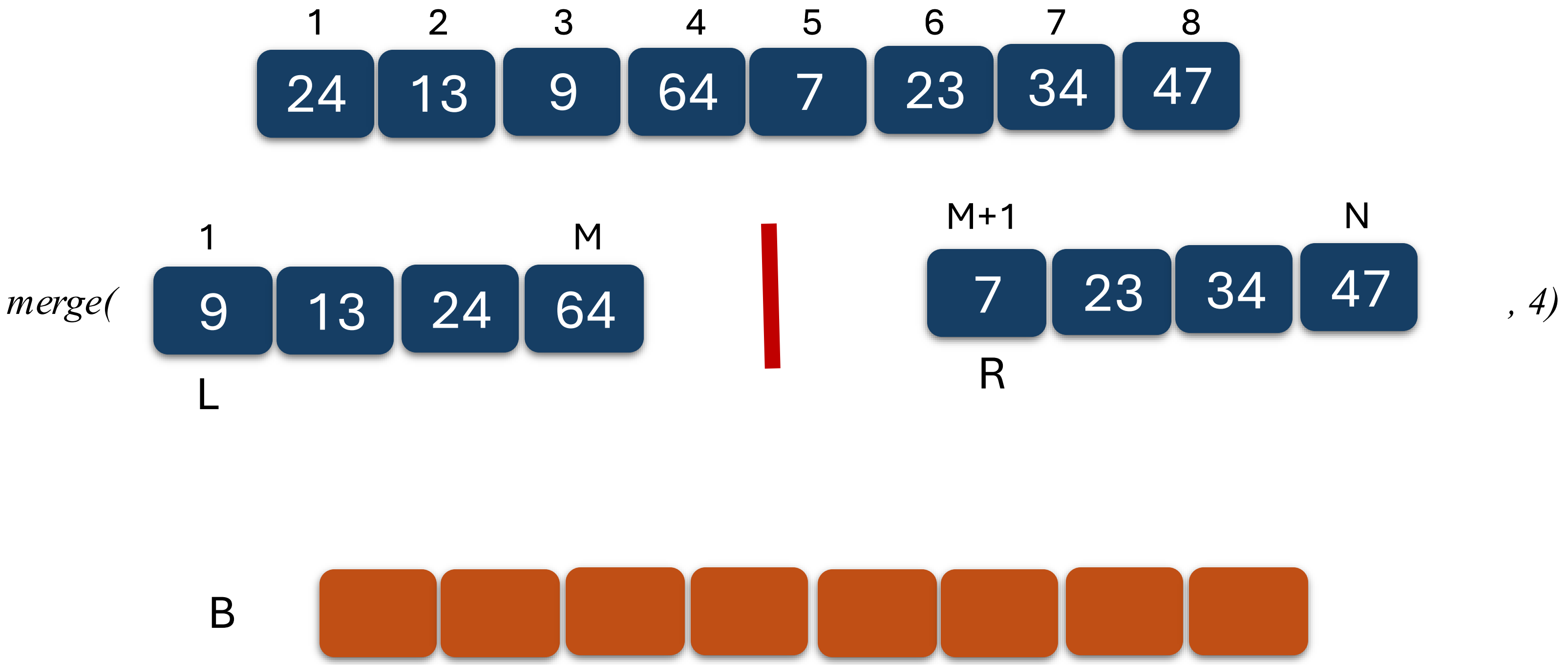
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] < A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



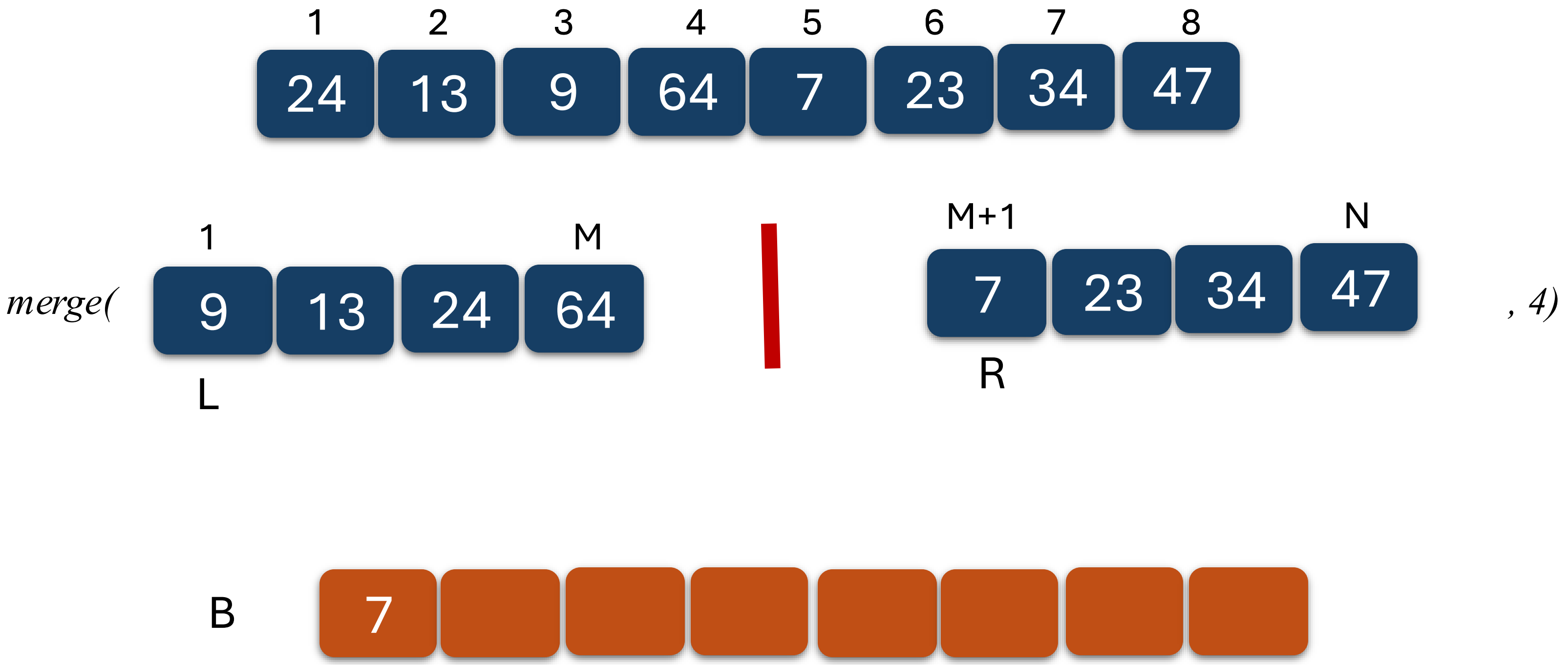
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <  A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



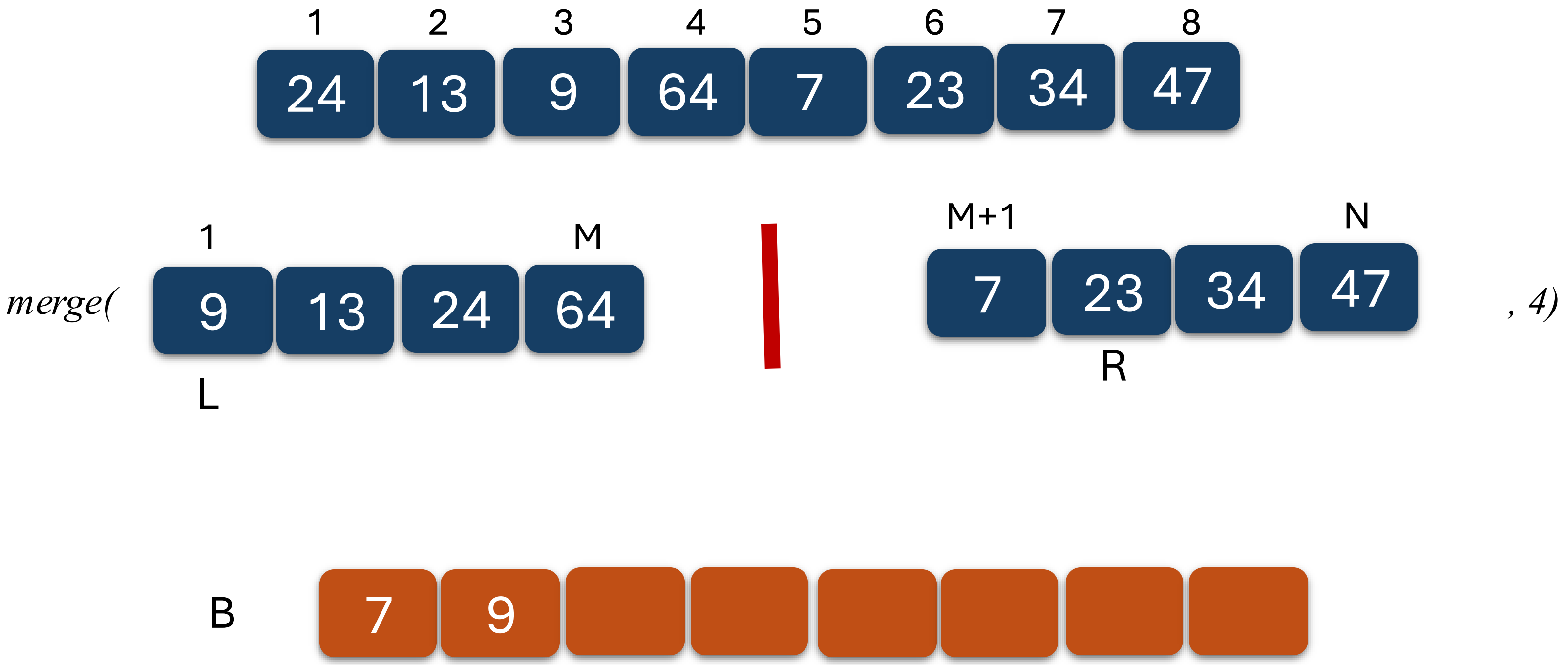
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <   A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



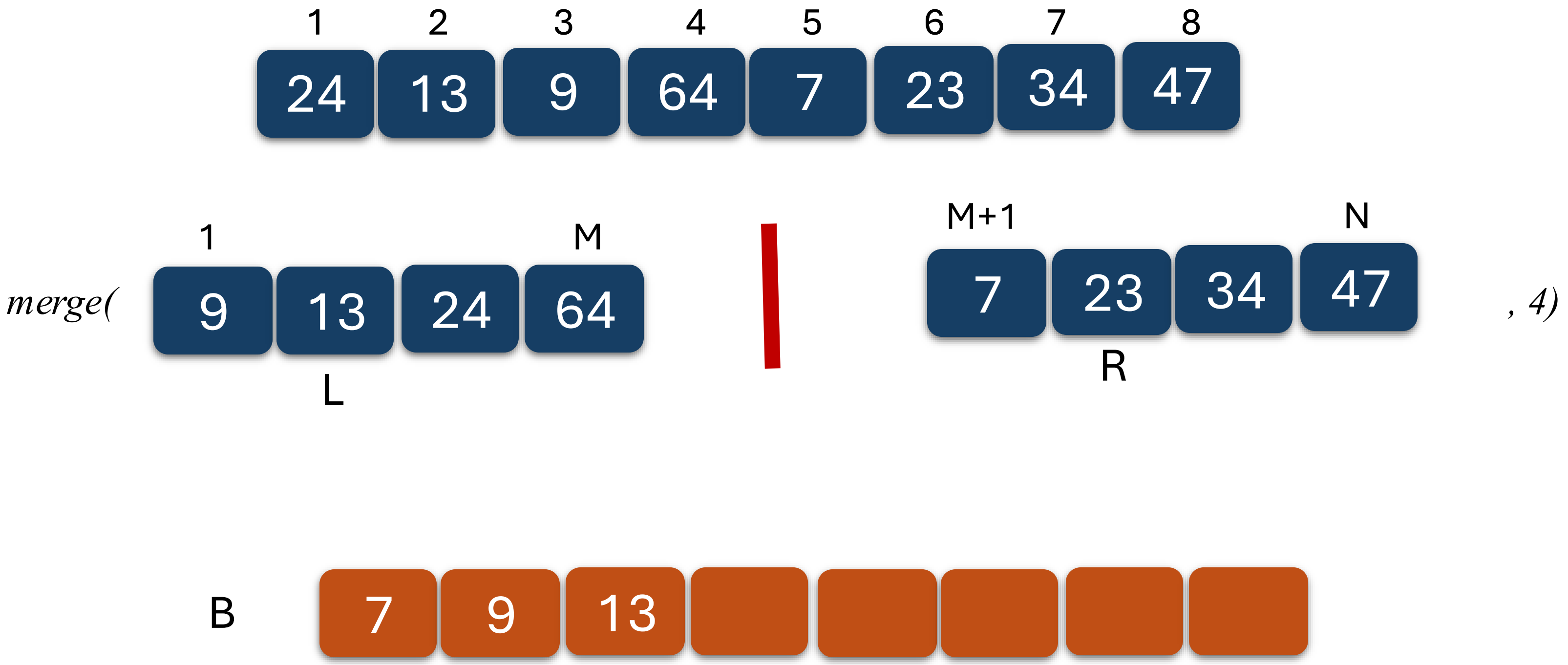
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
        copy B to A
```



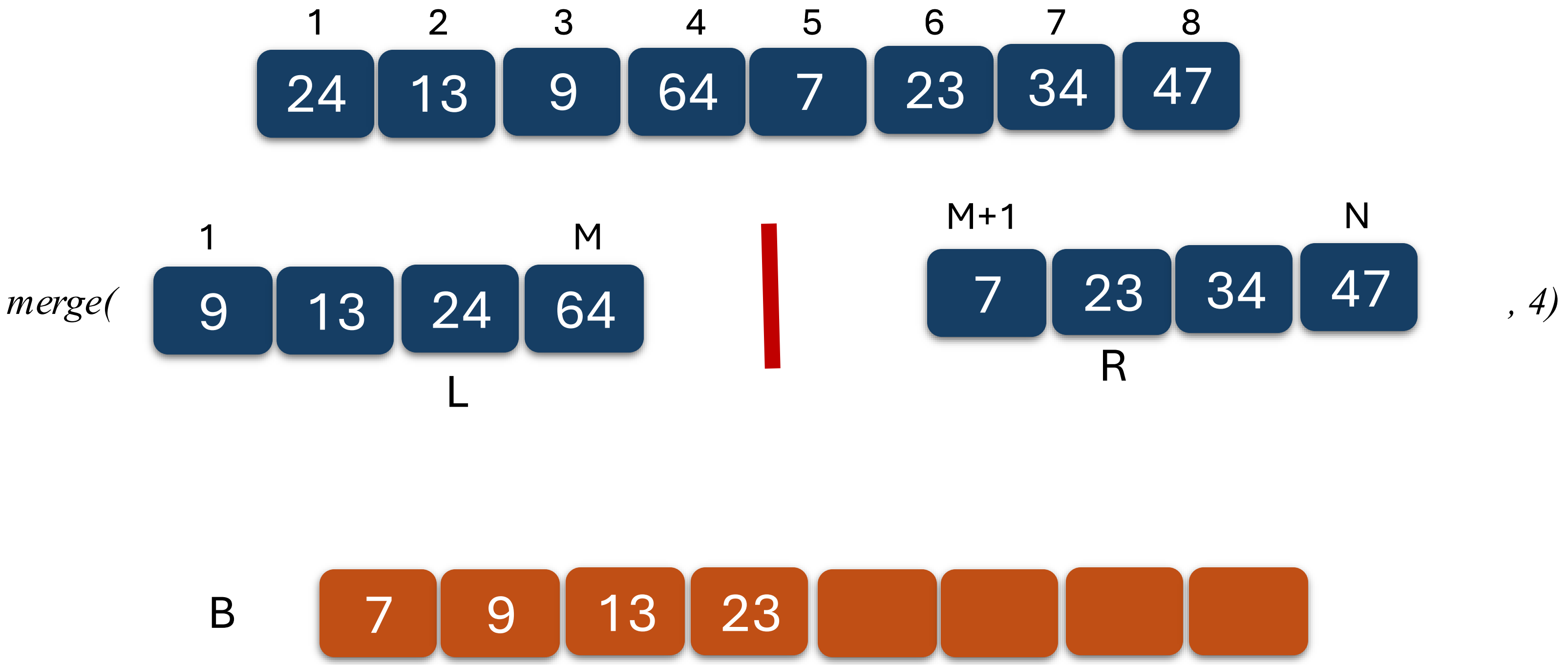
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
        copy B to A
```



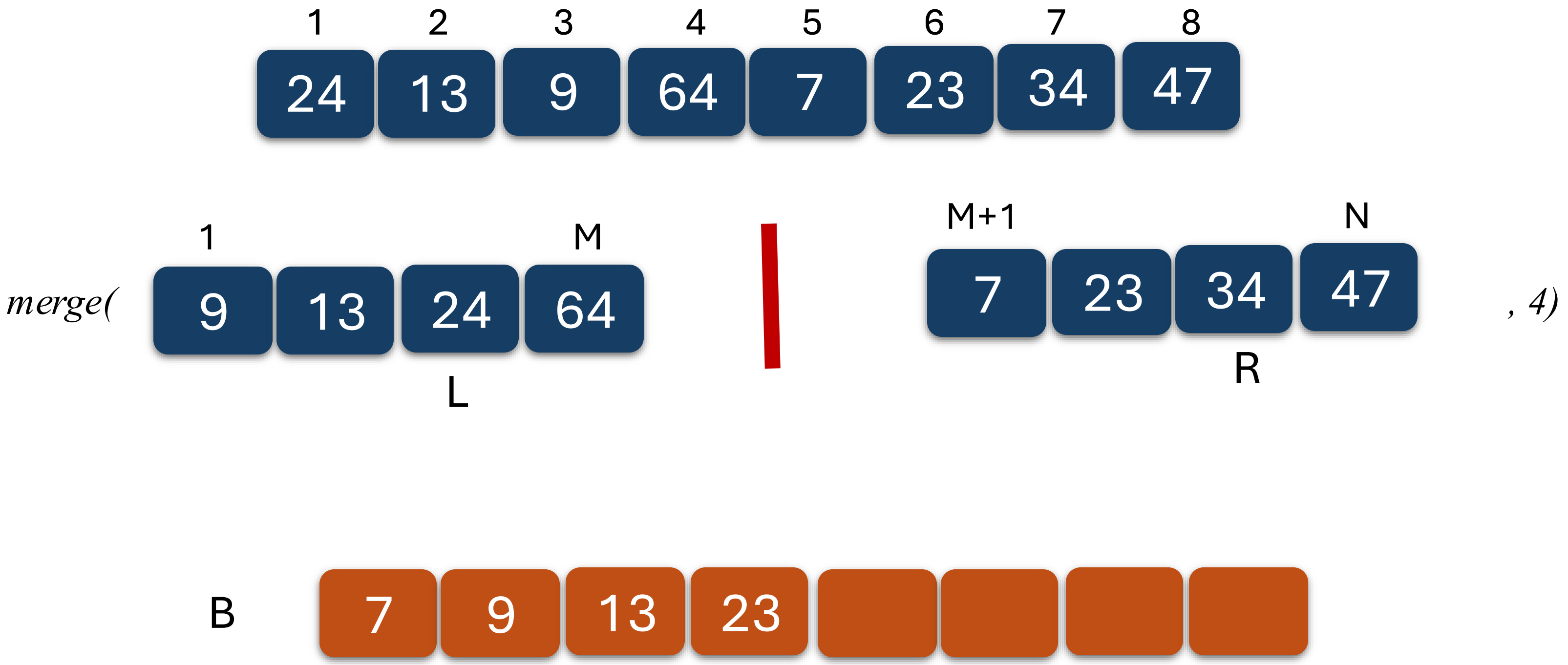
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
        copy B to A
```



```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
        copy B to A
```



1	2	3	4	5	6	7	8
24	13	9	64	7	23	34	47

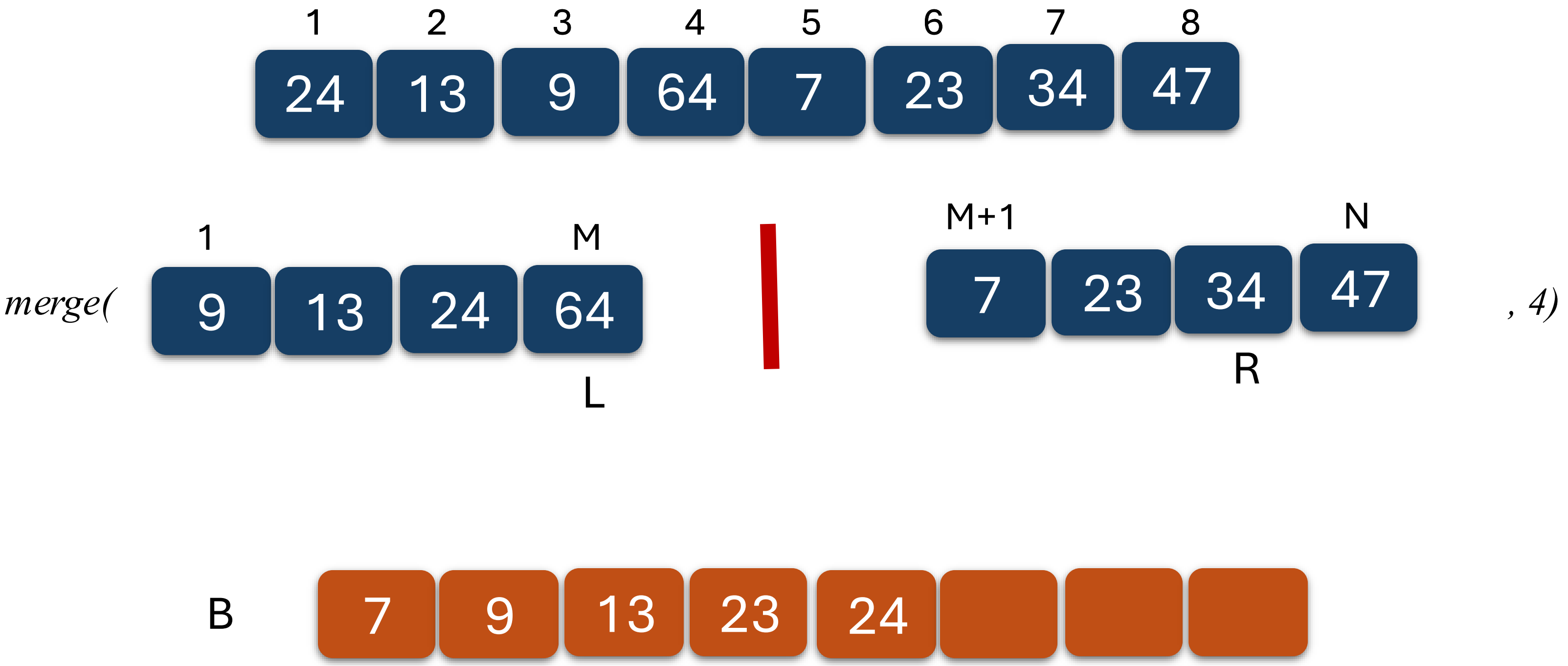
1			M
9	13	24	64

M+1			N
7	23	34	47

B							
7	9	13	23				

```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <   A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



1	2	3	4	5	6	7	8
24	13	9	64	7	23	34	47

merge(

1			M
9	13	24	64
			L

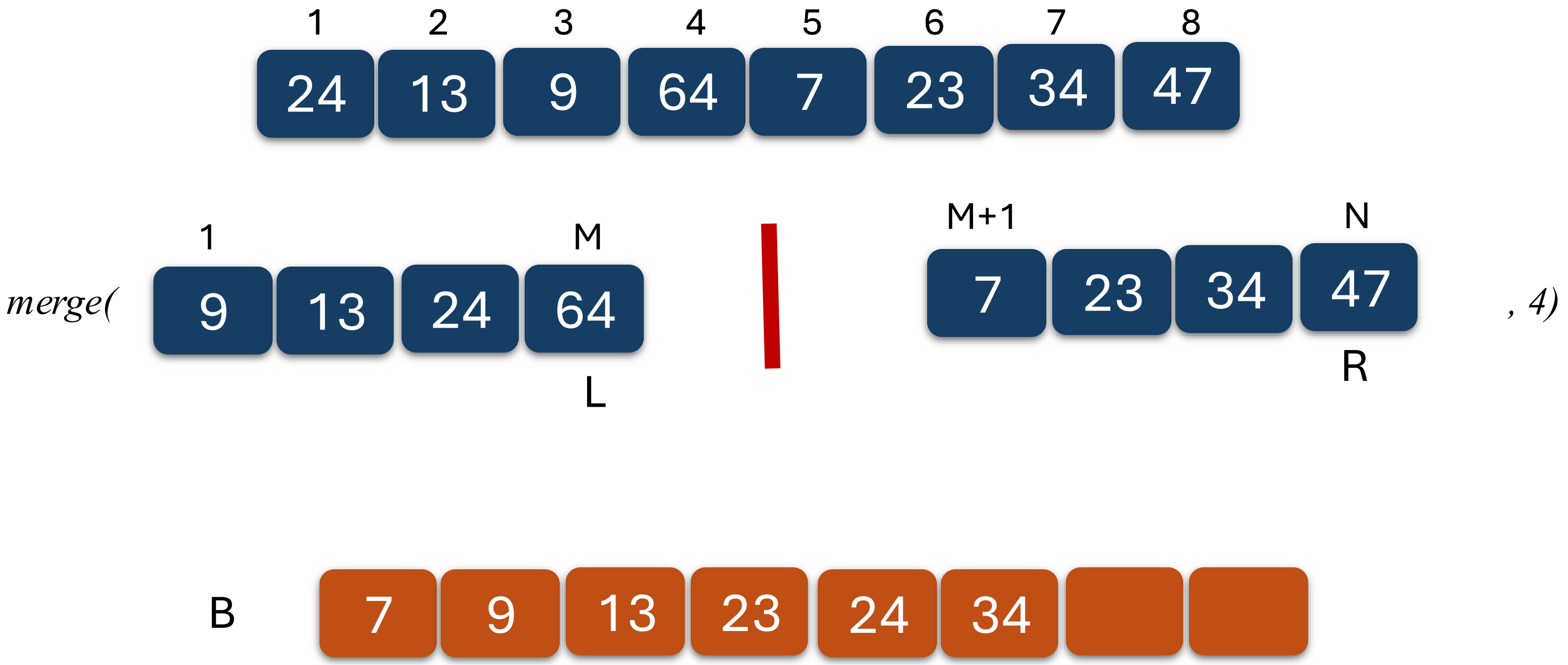
M+1			N
7	23	34	47
		R	

, 4)

B								
	7	9	13	23	24			

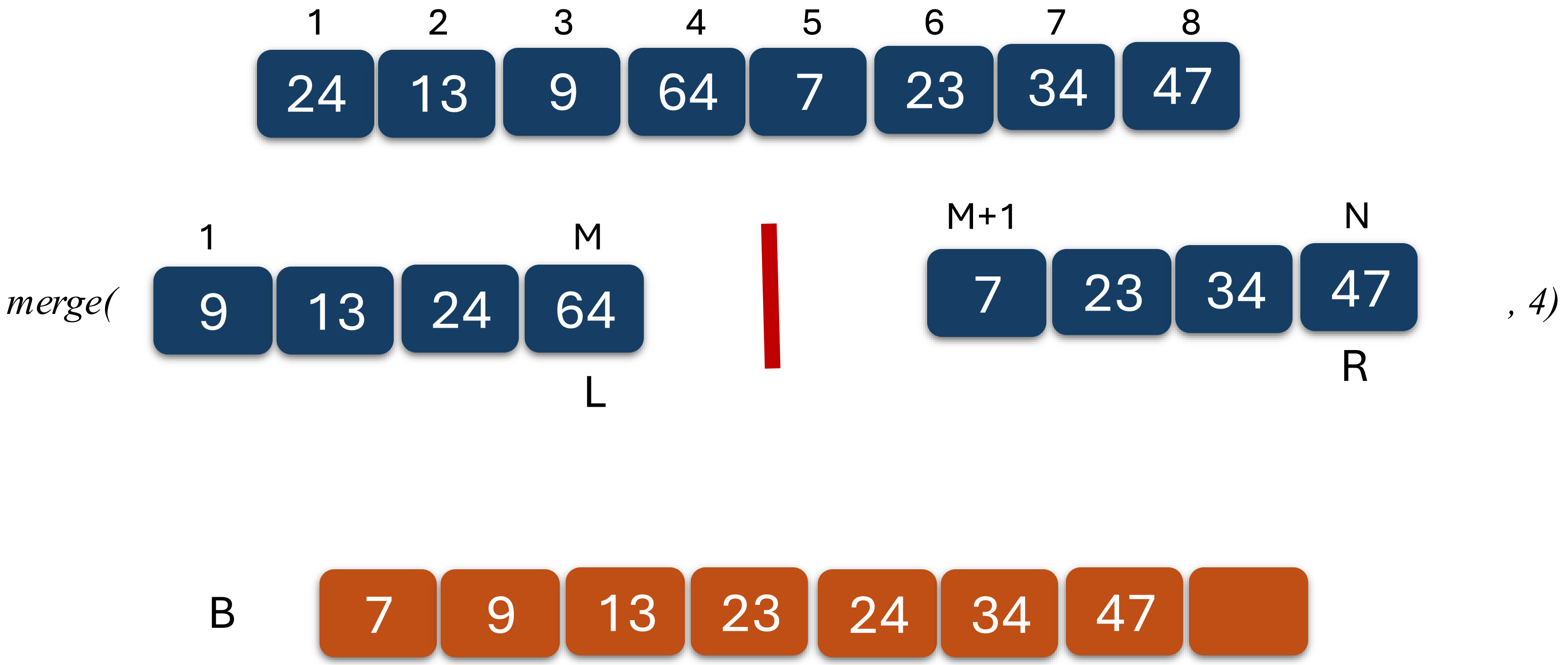
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



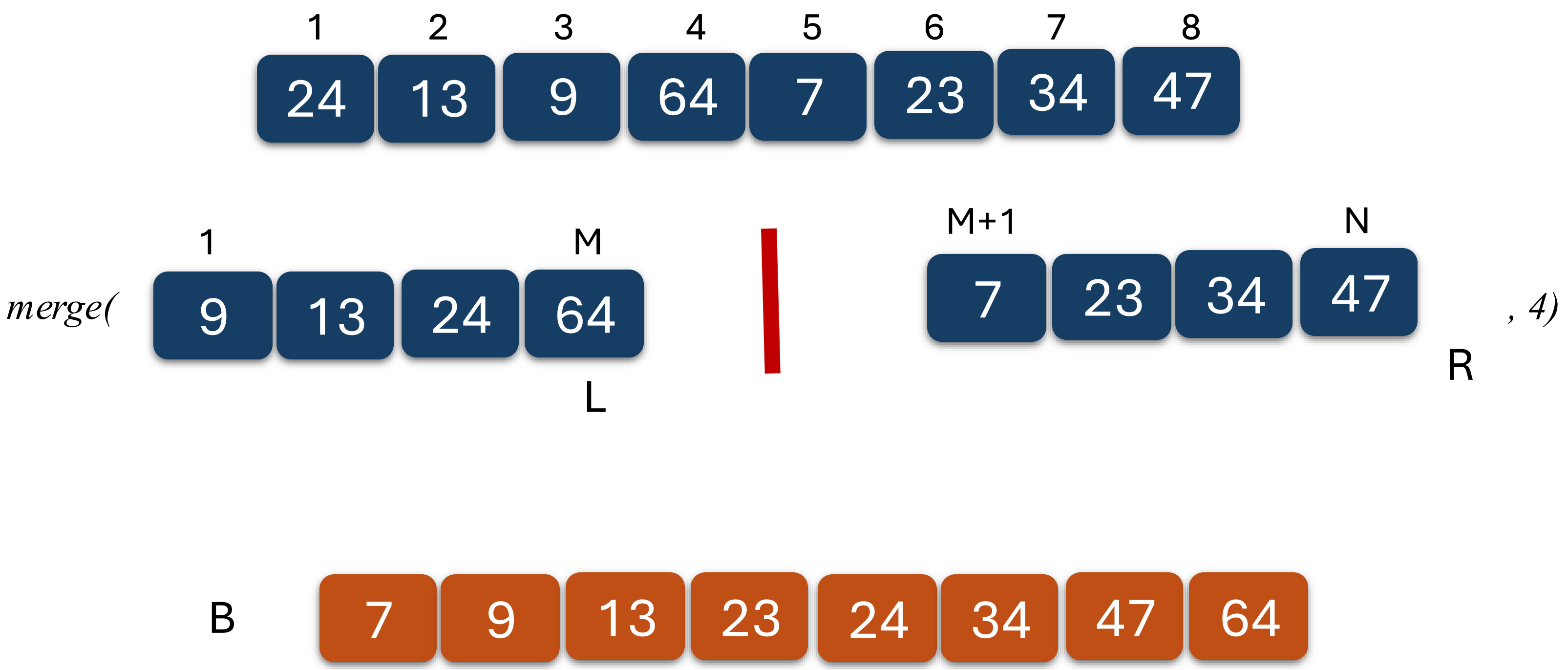
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥A[right]:
            B[ i ] = A[right]
            right += 1
        copy B to A
```



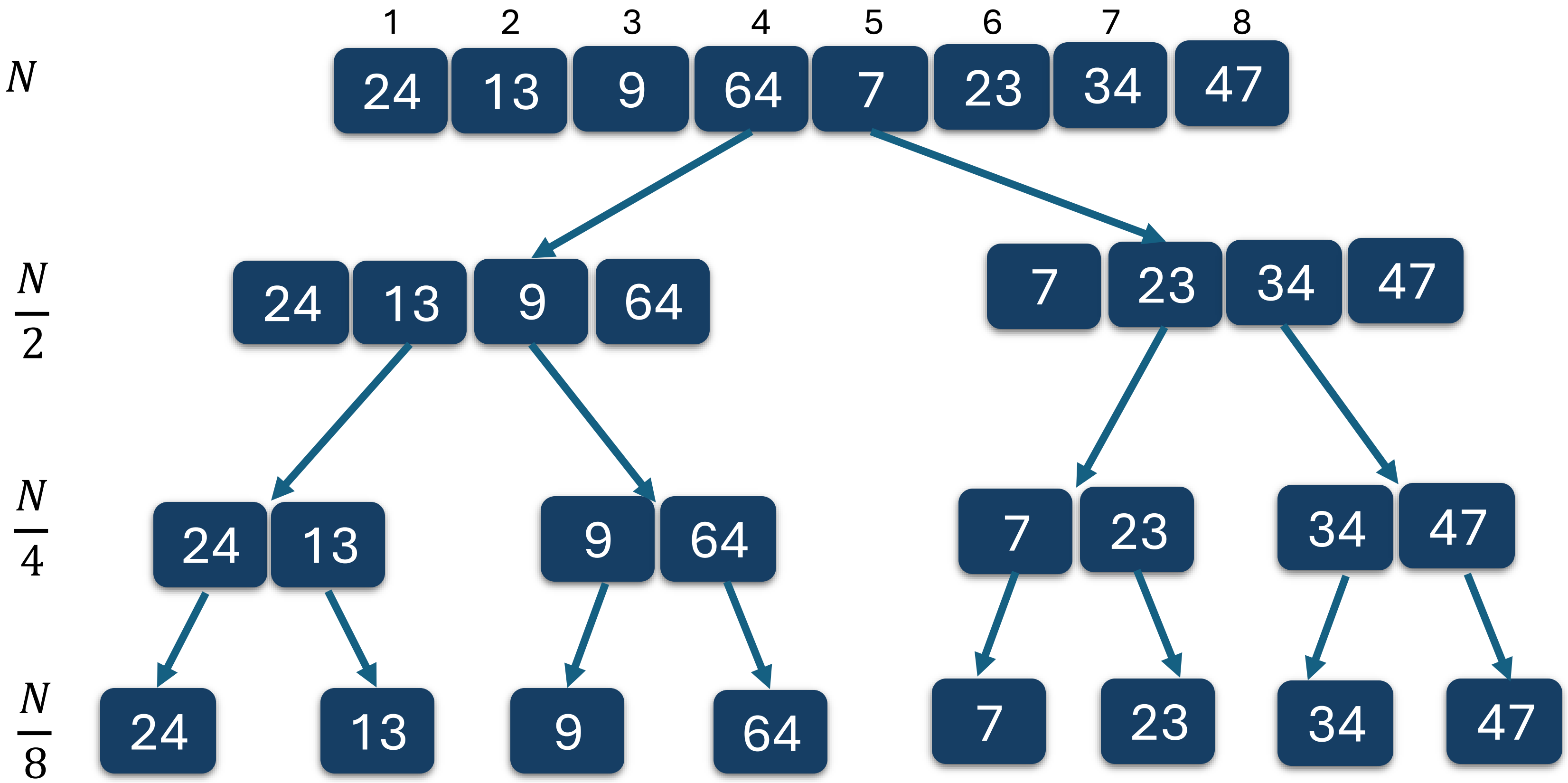
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
    copy B to A
```



Merge Sort Complexity

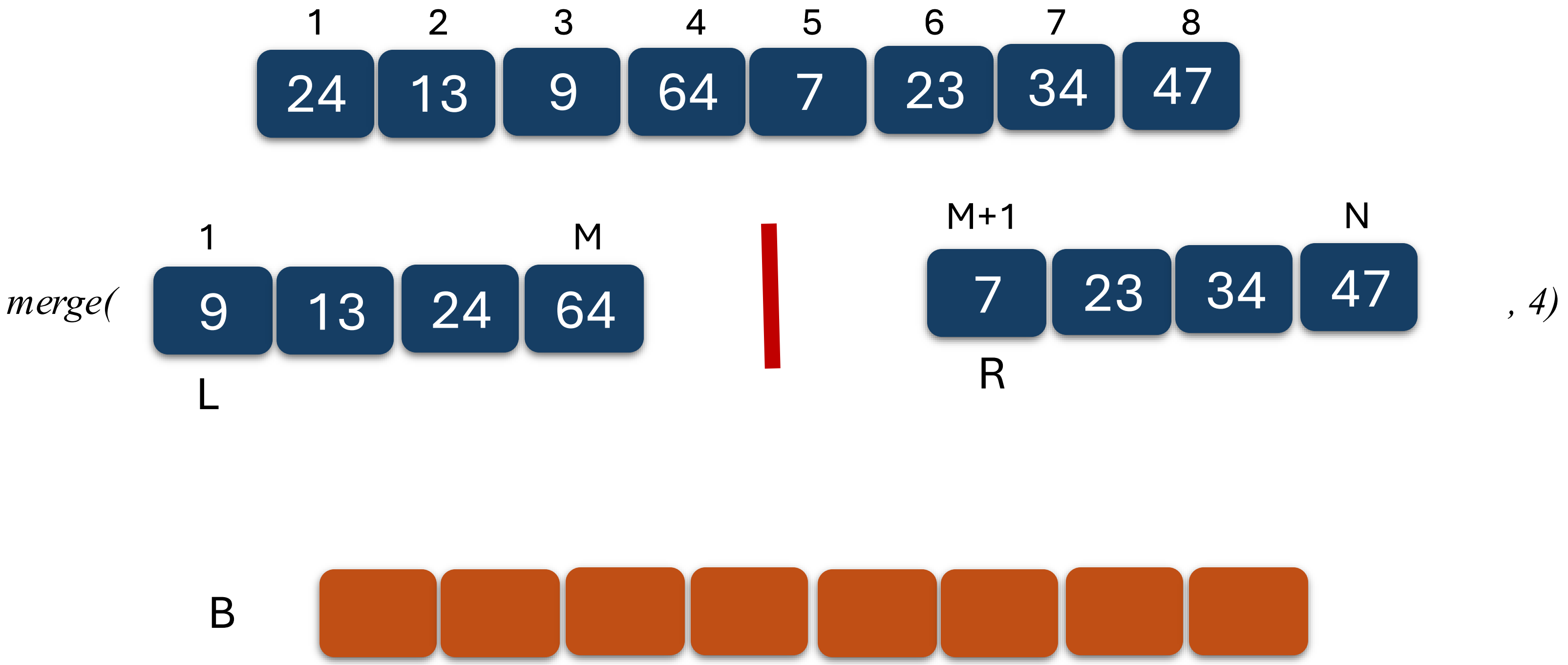
```
merge_sort(array A):
    if $N > 1$:
        $M = N / 2$
        merge_sort(A[1 : M])
        merge_sort(A[M+1 : N])
        merge(A[1:N], M)
```



- Height of recursion tree = $O(\log_2 N)$, since we divide the problem in half during recursion
- There are O(log N) iterations / recursion levels
- In each level, we perform O(N) work during Merge since we are comparing the left and right solution’s first items, at most N items.

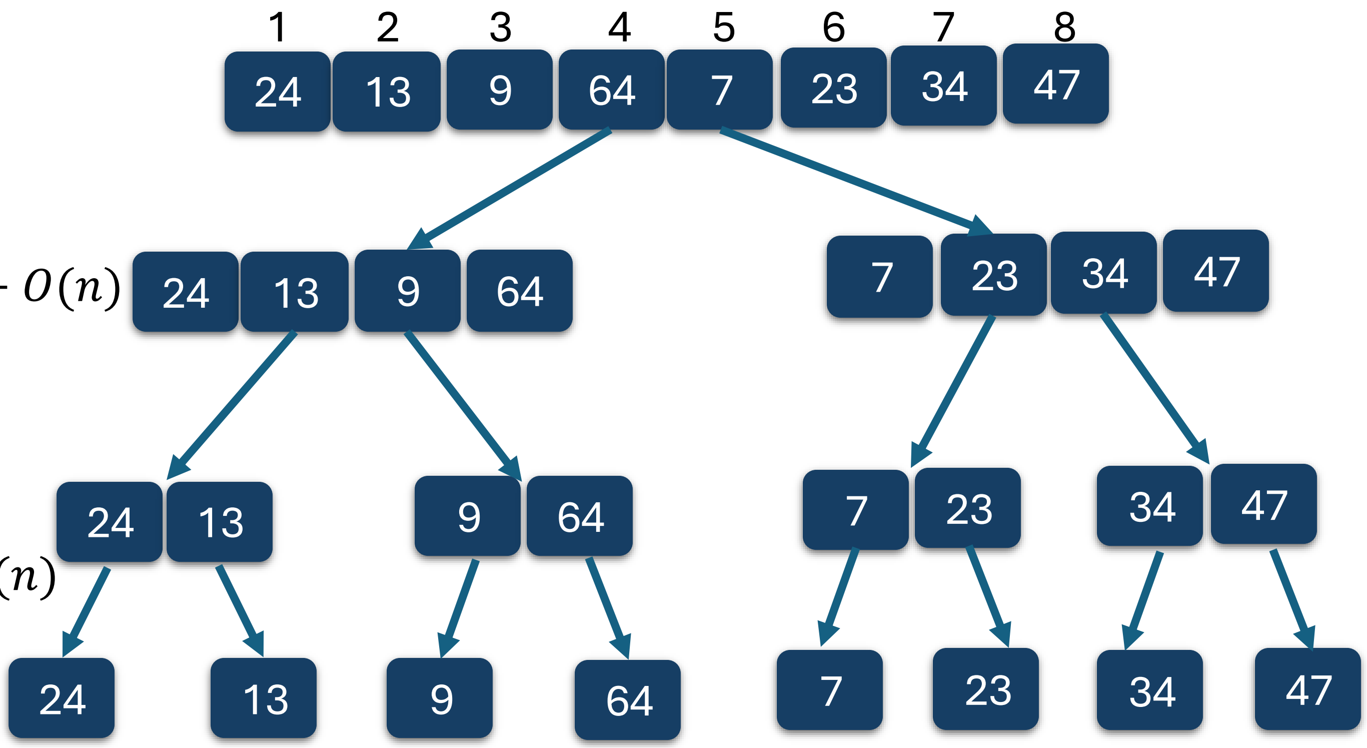
```
merge(array A, integer M):
    left_half = A[1 : M]
    right_half = A[M+1 : N]
    for i = 1 to N:
        L = first item of left_half
        R = first item of right_half
        B[i] = min(L, R)
    copy B to A
```

```
merge(array A, integer M):
    B = empty array of size N
    left = 1
    right= M + 1
    for i = 1 to N:
        if right > N:
            B[ i ] = A[left]
            left += 1
        else if left > M:
            B[ i ] = A[right]
            right += 1
        else if A[left] <    A[right]:
            B[ i ] = A[left]
            left += 1
        else if A[left] ≥ A[right]:
            B[ i ] = A[right]
            right += 1
        copy B to A
```



- Height of recursion tree = O($\log_2$ N), since we divide the problem in half during recursion
- There are O(log N) iterations / recursion levels
- In each level, we perform O(N) work during Merge since we are comparing the left and right solution’s first items, at most N items.

```
merge_sort(array A):
    if N > 1:                      $O(1)$
        M = N / 2
        merge_sort(A[1 : M])       $T\left(\frac{N}{2}\right)$
        merge_sort(A[M+1 : N])     $T\left(\frac{N}{2}\right)$
        merge(A[1:N], M)           $O(n)$
```



$$2T\left(\frac{n}{2}\right) + O(n)$$

$$4\left[T\left(\frac{n}{4}\right)\right] + O(n)$$

$$8\left[T\left(\frac{n}{8}\right)\right] + O(n)$$

$$n[T(1)] + O(n)$$

$$= O(nlogn)$$

```
merge_sort(array A):
    if N > 1:                          $O(1)$
        M = N / 2
        merge_sort(A[1 : M])           $T\left(\frac{N}{2}\right)$
        merge_sort(A[M+1 : N])         $T\left(\frac{N}{2}\right)$
        merge(A[1:N], M)               $O(n)$
```

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2}$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4}$$

$$\cdots$$

$$T(1) = 2T\left(\frac{n}{n}\right) + n$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$$

$$T(n) = 4T\left(\frac{n}{4}\right) + n + n$$

$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n$$

$$T(n) = \left(8T\left(\frac{n}{8}\right) + n\right) + 2n$$

$$T(n) = 2^kT\left(\frac{n}{2^k}\right) + kn$$

$$\cdots$$

$$T(n) = 2^kT\left(\frac{2^k}{2^k}\right) + kn$$

$$T(n) = 2^kT(1) + \boxed{k}n$$

Stops until T(1) or if the array size=1 or if $\frac{N}{2^k} = 1$

Now, if $n = 2^k$

$n = 2^k$

$k = \log_2 n$

```
merge_sort(array A):
    if N > 1:                              $O(1)$
        M = N / 2
        merge_sort(A[1 : M])               $T\left(\frac{N}{2}\right)$
        merge_sort(A[M+1 : N])             $T\left(\frac{N}{2}\right)$
        merge(A[1:N], M)                   $O(n)$
```

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$$

$$T(n) = 4T\left(\frac{n}{4}\right) + n + n$$

$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n$$

$$T(n) = \left(8T\left(\frac{n}{8}\right) + n\right) + 2n$$

$$T(n) = 2^kT\left(\frac{n}{2^k}\right) + kn$$

$$\cdots$$

$$T(n) = 2^kT\left(\frac{2^k}{2^k}\right) + kn$$

$$T(n) = 2^kT(1) + \boxed{k}n$$

Stops until T(1) or if the array size=1 or if $\frac{N}{2^k} = 1$

Now, if $n = 2^k$

$n = 2^k$

$k = \log_2 n$

```
merge_sort(array A):
    if N > 1:                          $O(1)$
        M = N / 2
        merge_sort(A[1 : M])           $T\left(\frac{N}{2}\right)$
        merge_sort(A[M+1 : N])         $T\left(\frac{N}{2}\right)$
        merge(A[1:N], M)               $O(n)$
```

$$Merge\ Sort = 2^k T\left(\frac{N}{2^k}\right) + k * O(n)$$

Stops until T(1) or if the array size=1 or if $\frac{N}{2^k} = 1$

$$\frac{N}{2^k} = 1$$

$$N = 2^k$$

$$\log_2 N = \log_2 2^k$$

$$\log_2 N = k$$

...

$$Merge\ Sort = 2^k T\left(\frac{N}{2^k}\right) + k * O(n)$$

$$Merge\ Sort = n * T(1) + logn * O(n)$$

$$Merge\ Sort = n + logn * O(n) = O(nlogn)$$

Next meeting….

We’ll discuss more sorting algorithms.