SORTING ALGORITHM

Prepared By : JAYSHREE PARMAR

ASSISTANT PROFESSOR,PIET,PARUL UNIVERSITY

Sorting

- Sorting is a process that organizes a collection of data into either ascending or descending order.

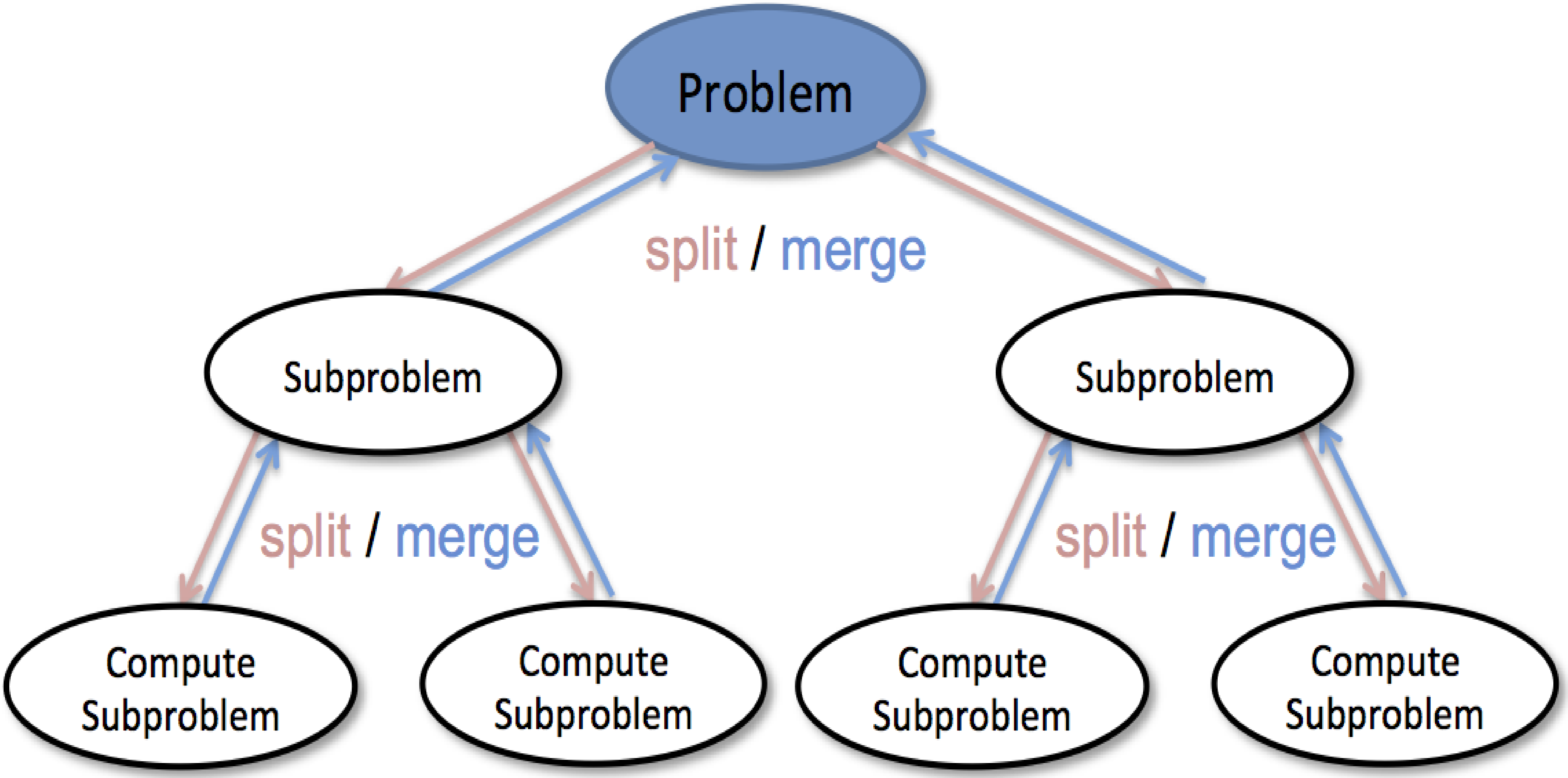
- In-place sorting vs not In-place sorting.
- Quick sort and Merge sort use divide and conquer approach.

Why Sorting is important ?

- Contact numbers in the mobile phone
- List of exam scores
- Words of dictionary in alphabetical order
- Students names listed alphabetically
- Student records sorted by ID#
- **Searching(**enhance the performance of an algorithm)

.

DIVIDE AND CONQUER



Divide and Conquer

- **Basic Idea**
- Divide instance of problem into two or more smaller instances
- Solve smaller instances recursively
- Obtain solution to original (larger) instance by combining these solutions
- Divide and Conquer algorithms consist of two parts:
  - ***Divide:*** Smaller problems are solved recursively (except, of course, the base cases).
  - ***Conquer:*** The solution to the original problem is then formed from the solutions to the subproblems.

QUICK SORT

- **Quicksort** (sometimes called **partition-exchange sort**) is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order.
- Quicksort can operate in-place on an array, requiring small additional amounts of memory to perform the sorting.
- It is very similar to selection sort, except that it does not always choose worst-case partition.
- On average, the algorithm takes $O(n \log n)$ comparisons to sort n items.
- In the worst case, it makes $O(n^2)$ comparisons, though this behaviour is rare.

Quicksort Procedure

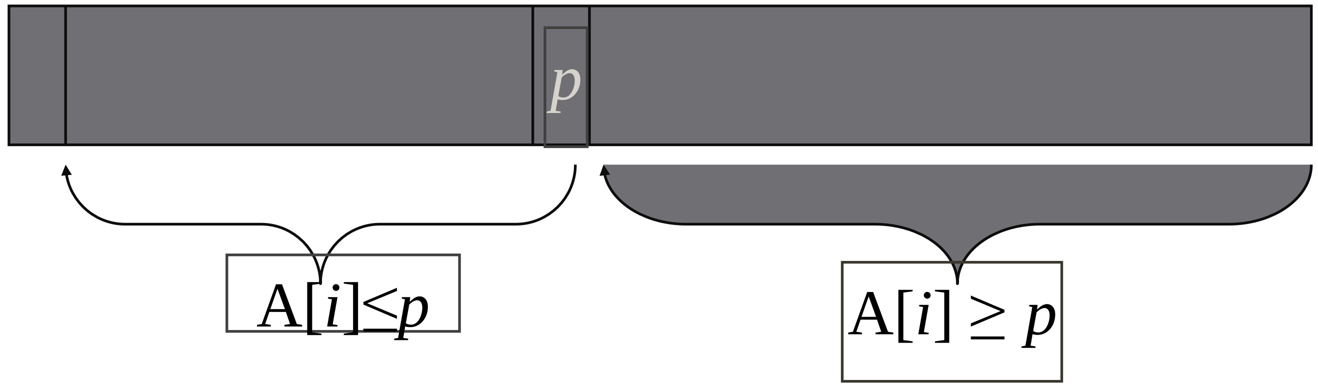
- Select a pivot which divide or partition the list in two part.
  - e.g., pivot = A[n-1] or A [0] or any random element

Step 1 – Make the left-most index value pivot

Step 2 – partition the array using pivot value

Step 3 – quicksort left partition recursively

Step 4 – quicksort right partition recursively



Example of partition

LB UB

5	3	2	6	4	1	3	7

START

PIVOT

3 <= 5

END

WHILE A[START] < = PIVOT

START++

Initially START=LB+1 And
END=UB

LB UB

5	3	2	6	4	1	3	7

START

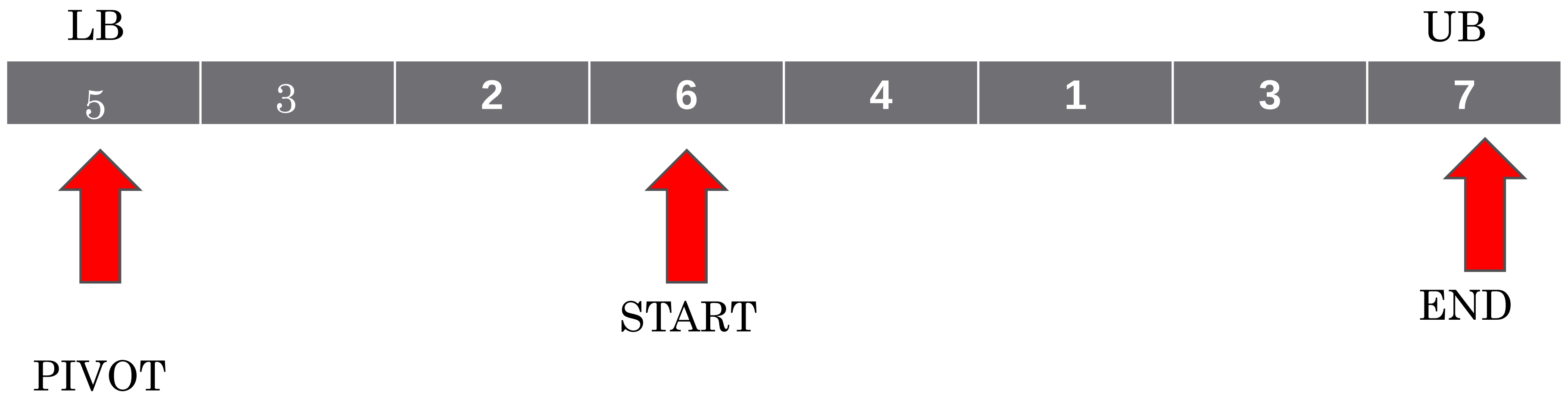
2 <= 5

END

PIVOT

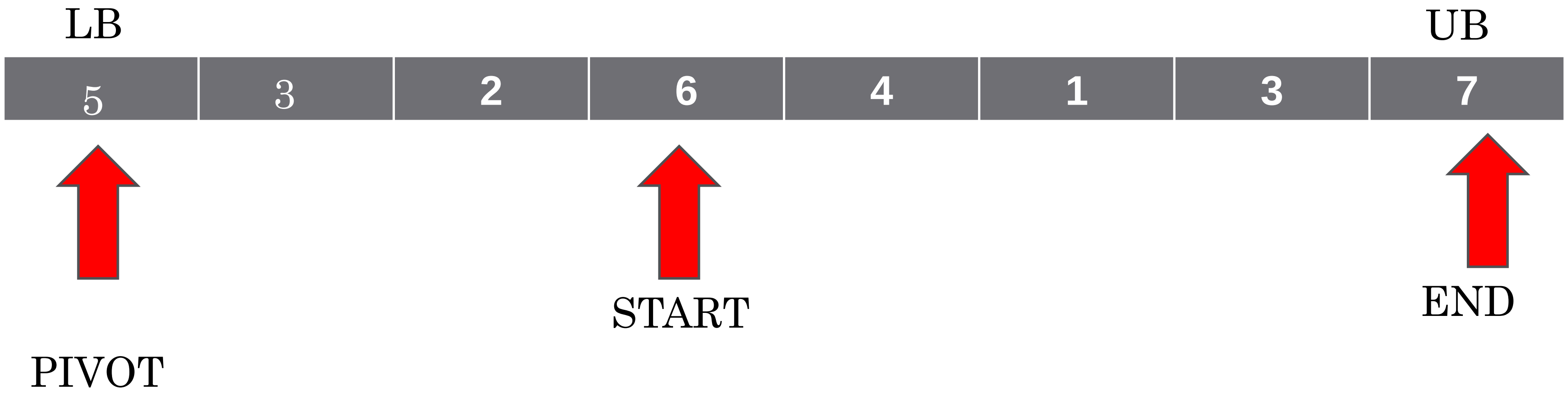
WHILE A[START] < = PIVOT

START++



6 <= 5 FALSE HENCE STOP

WHILE A[START] < = PIVOT



7 > 5

WHILE A[END] > PIVOT

END--

LB							UB
5	3	2	6	4	1	3	7

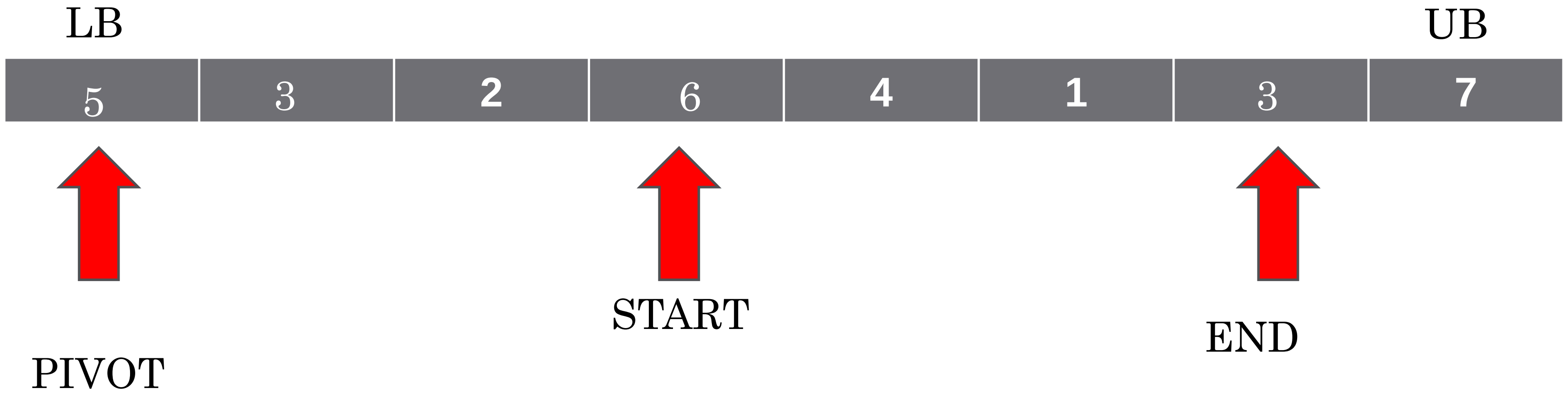
START

END

PIVOT

3 > 5 FALSE HENCE STOP

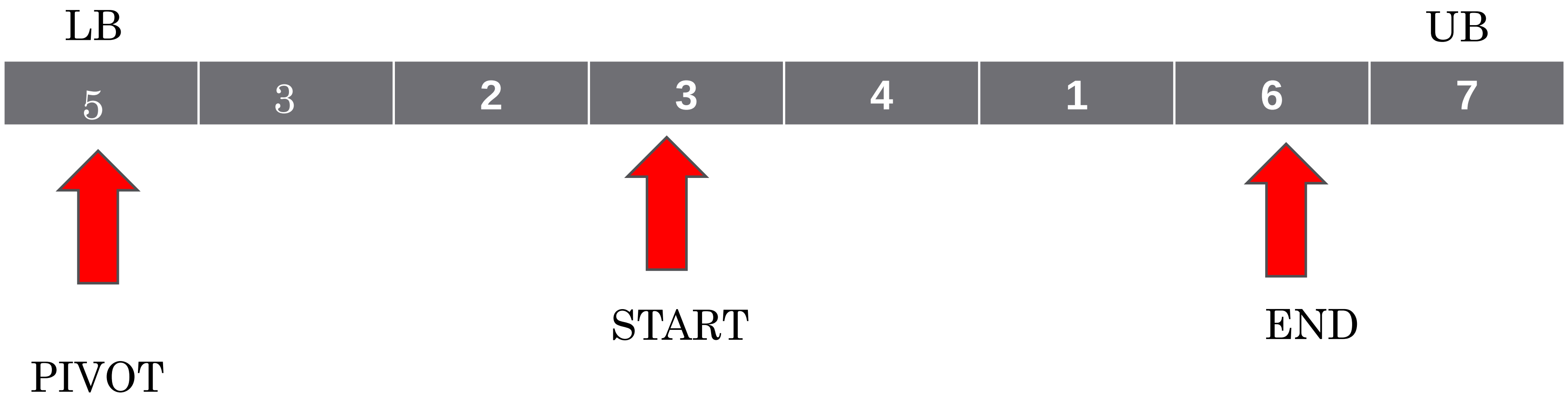
WHILE A[END] > PIVOT



3 < 6

IF START < END

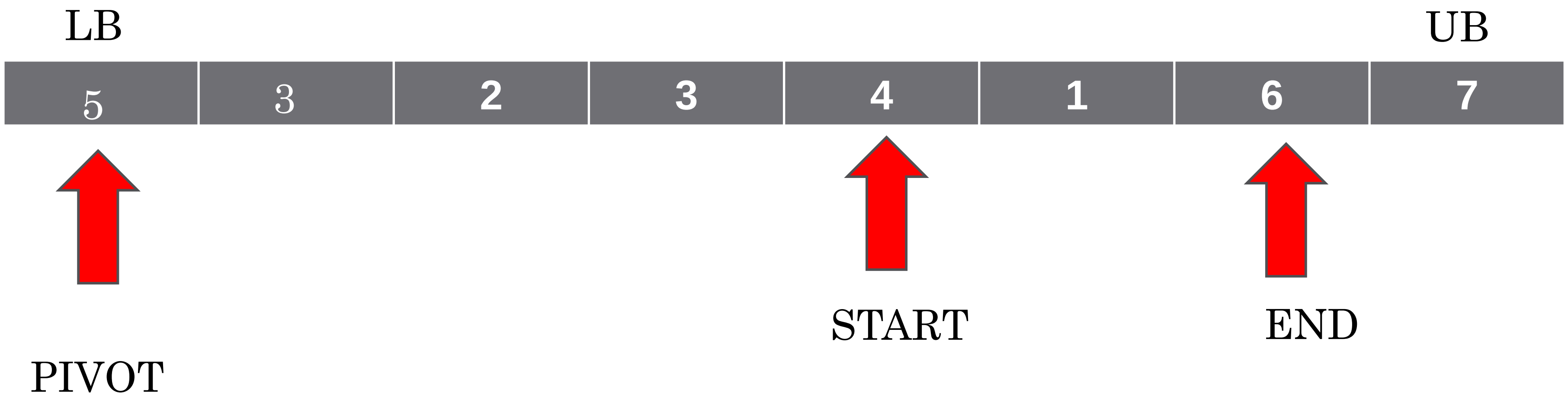
SWAP (A[START],A[END])



3 <= 5

WHILE A[START] < = PIVOT

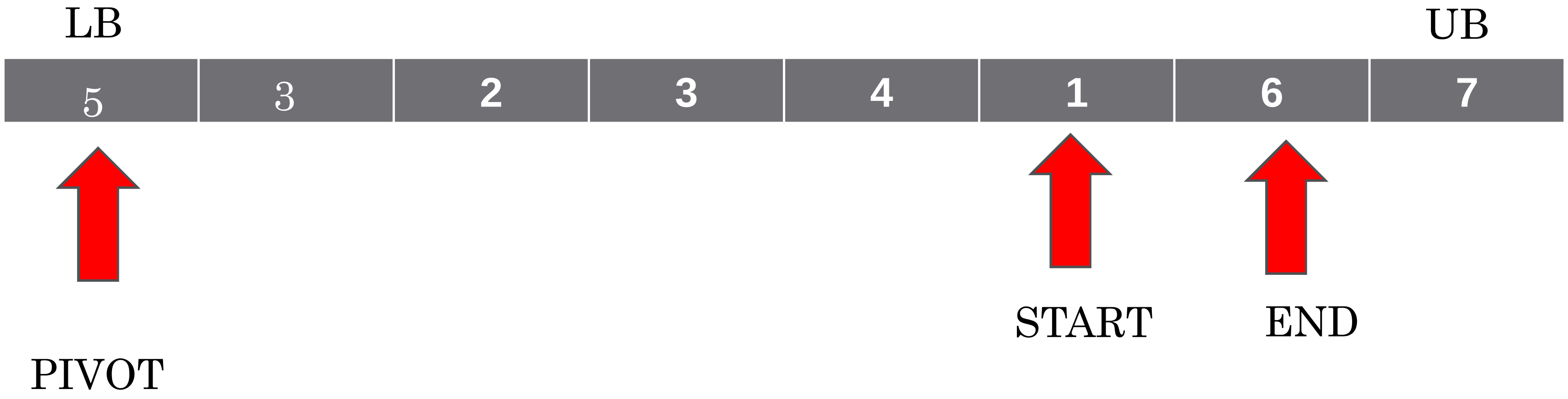
START++



4 <= 5

WHILE A[START] < = PIVOT

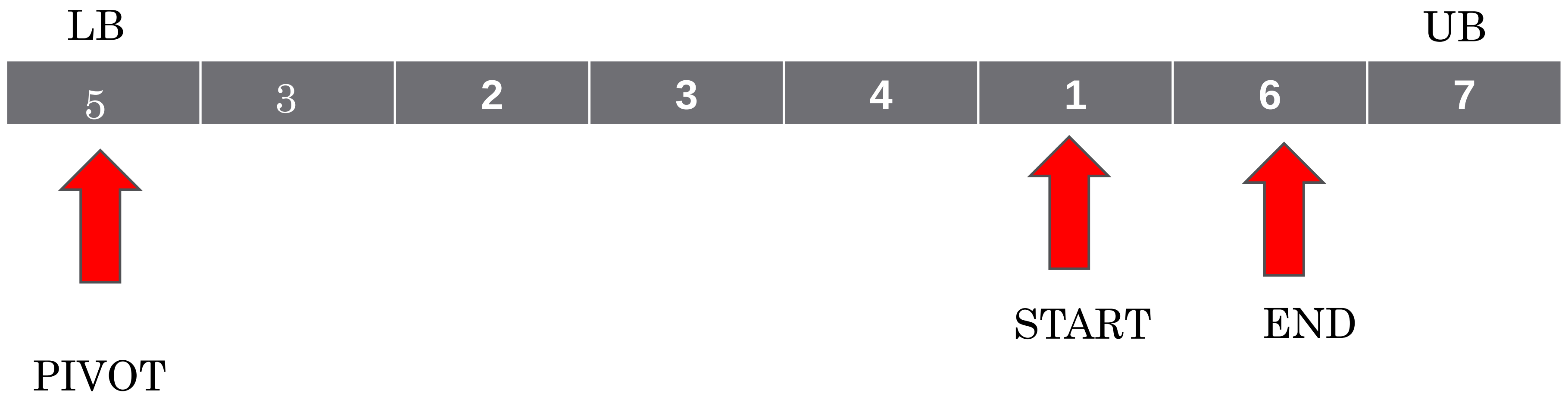
START++



1 <= 5

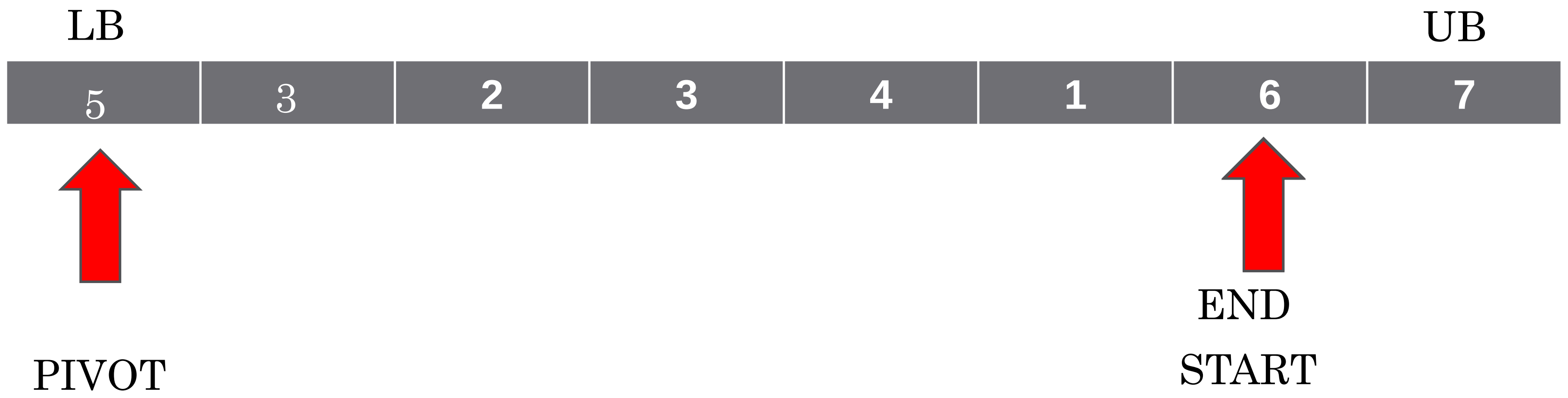
WHILE A[START] < = PIVOT

START++



6 <= 5 FALSE HENCE STOP

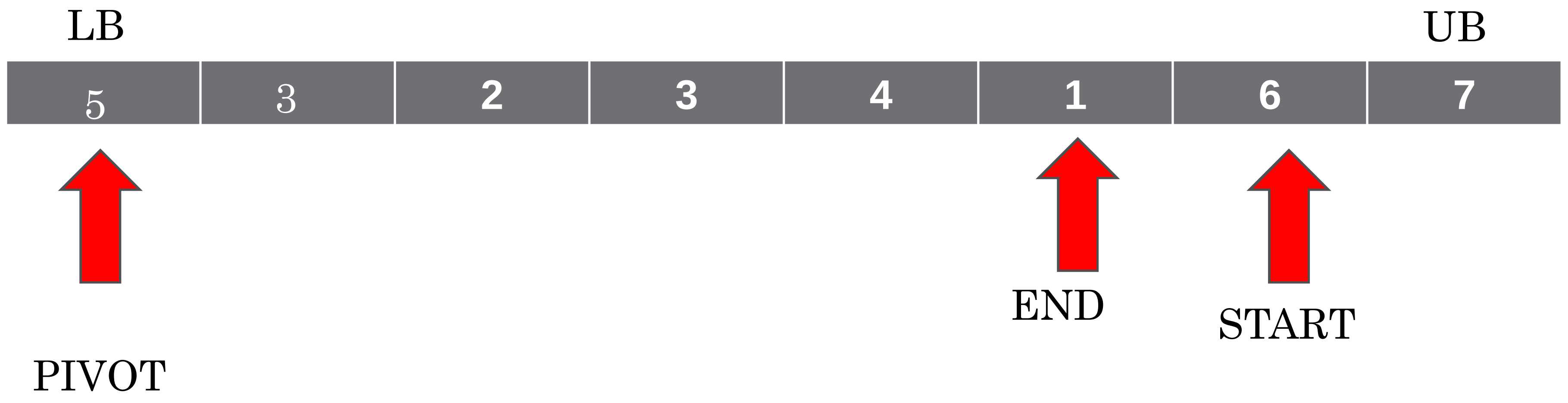
WHILE A[START] < = PIVOT



$6 > 5$

WHILE A[END] > PIVOT

END--



1 > 5

FALSE HENCE STOP

WHILE A[END] > PIVOT

LB UB

5	3	2	3	4	1	6	7

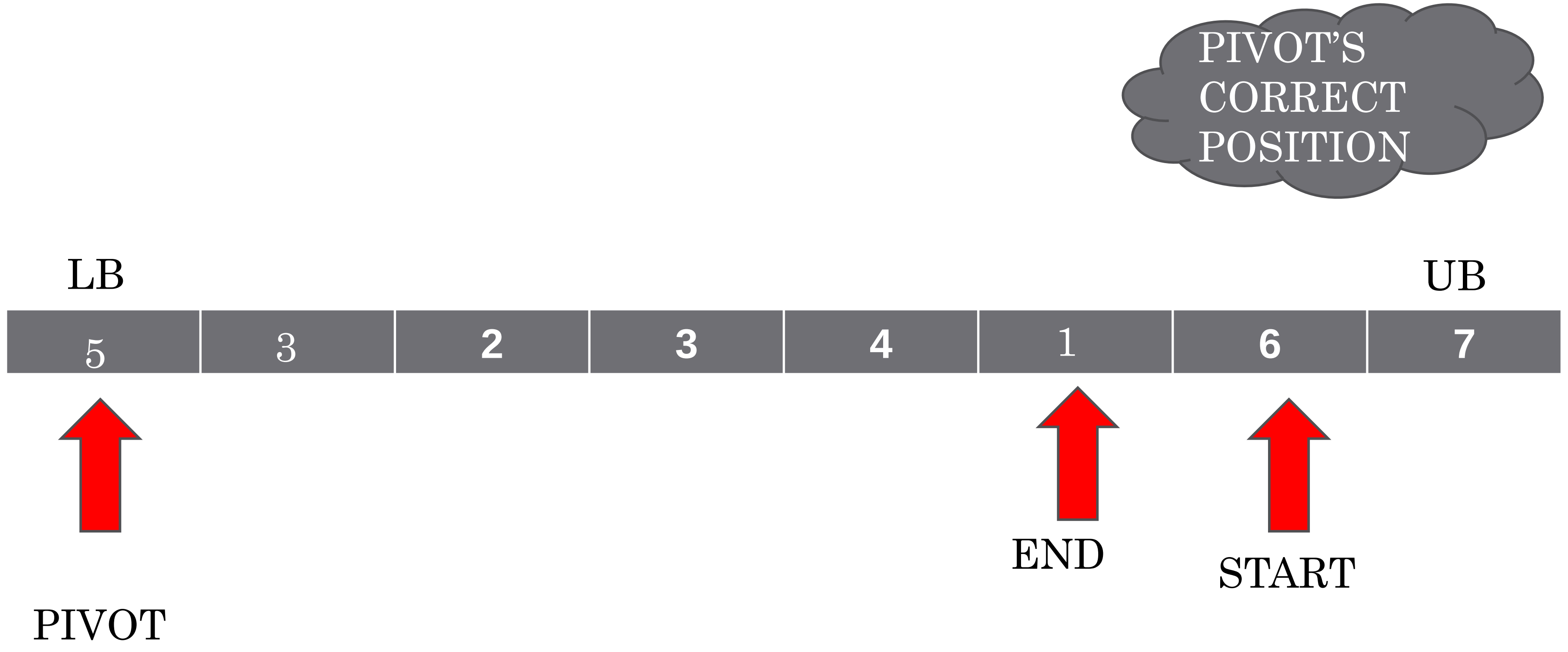
END START

PIVOT

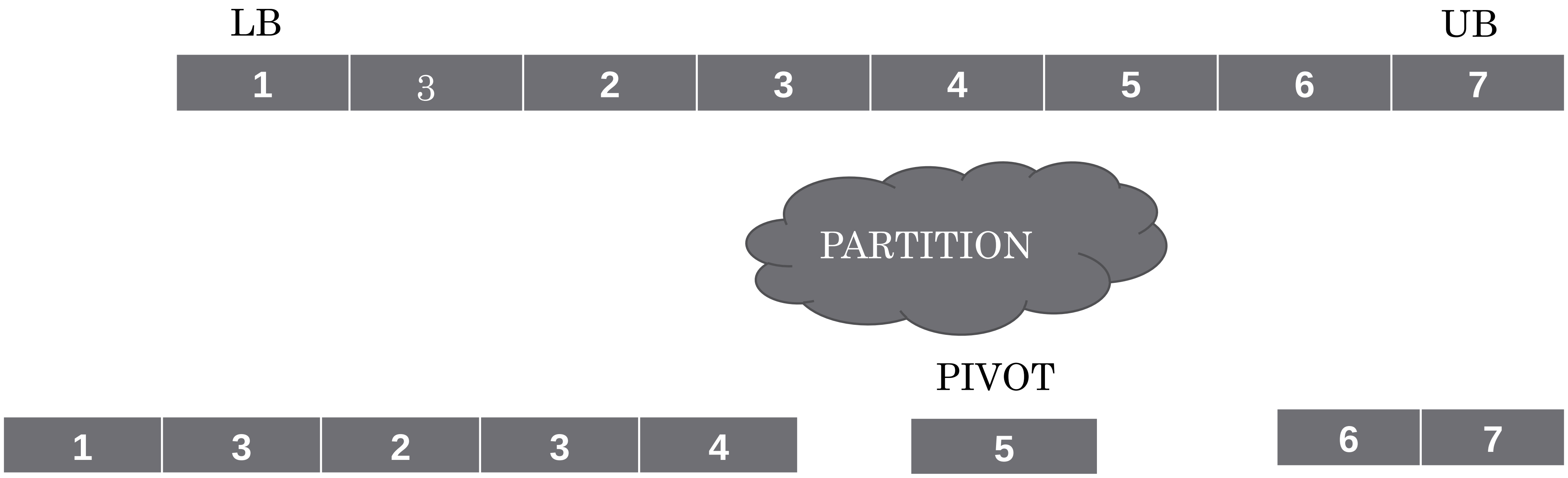
6 < 5

FALSE HENCE NO SWAP

IF START < END

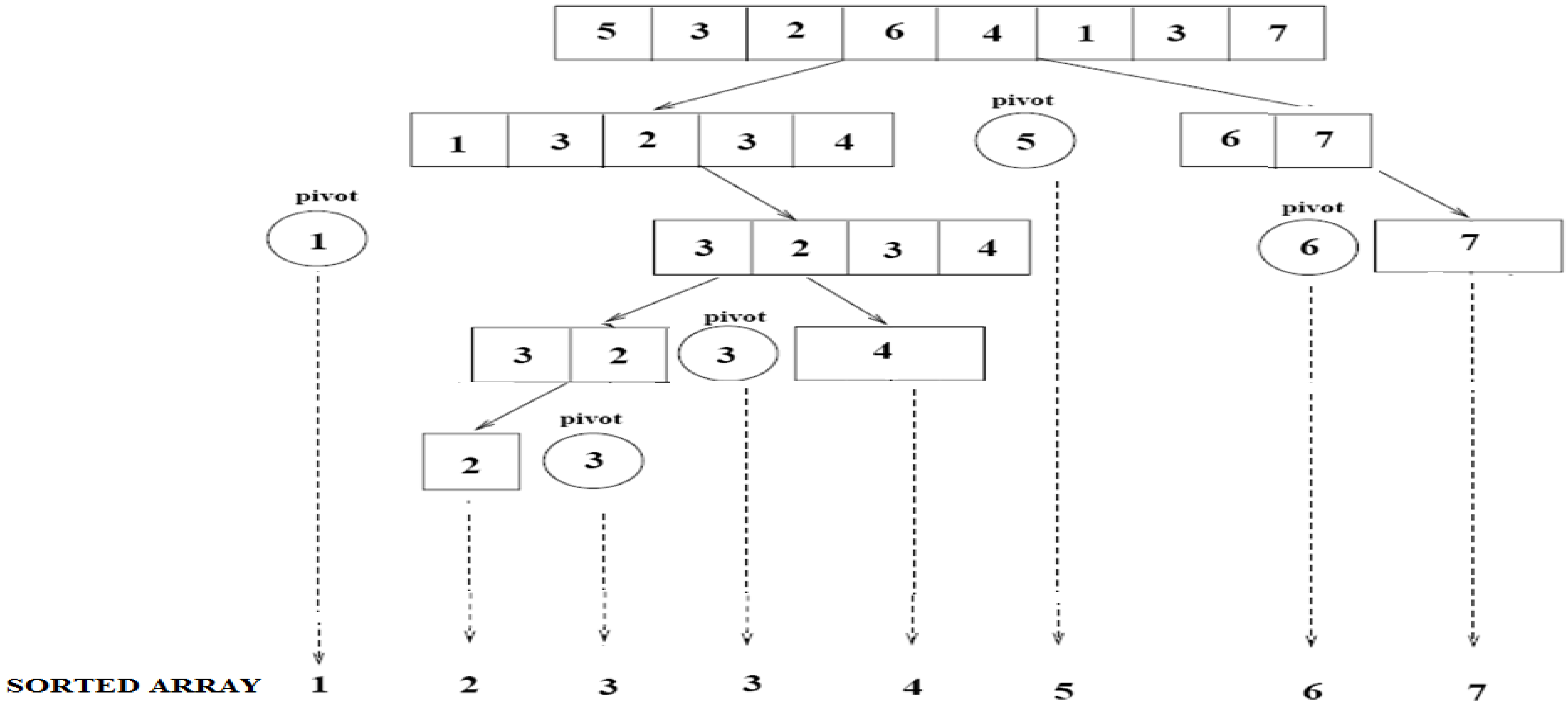


NOW (SWAP A[LB] ,A[END])



LB
UB
1
3
2
3
4
5
6
7
PARTITION
PIVOT
1
3
2
3
4
5
6
7

FINAL EXAMPLE



Quicksort Algorithm

```
quicksort ( A , lb , ub )
{
   If ( lb <  ub )

      {
          q = Partition ( A , lb , ub );
          quicksort ( A , lb , q-1 );
          quicksort ( A , q+1 , ub );
      }
}
 Partition ( A , lb , ub )
{
  pivot = A[lb];
  start = lb+1;
  end = ub;
  while ( start < end )
      {
            while ( A[start] <= pivot )
              {
                 start ++;
              }
             while ( A[end] > pivot )
             {
                 end - -;
             }
             if ( start < end )
               {
                    swap( A[srart] , A[end] );
               }
         }
     swap( A[lb] , A[end] );
     return end;
}
```

Merge sort: Motivation

If I have two helpers, I'd…

- Give each helper half the array to sort
- Then I get back the sorted subarrays and merge them.

What if those two helpers each had two sub-helpers?

And the sub-helpers each had two sub-sub-helpers? And…

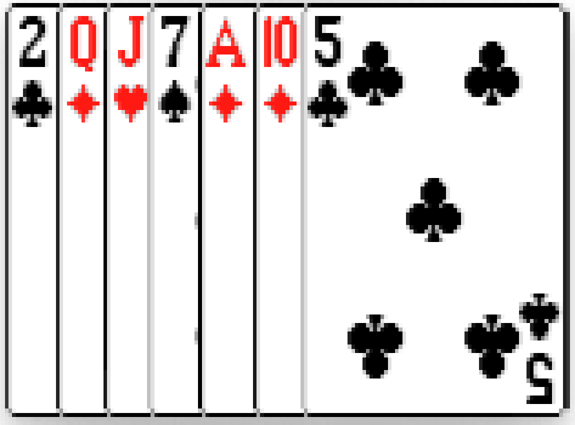
Mergesort

- Mergesort algorithm is one of two important divide-and-conquer sorting algorithms (the other one is quicksort).
- The merge sort algorithm uses “Bottom up” approach
  - start by solving the smallest pieces of original problem
  - keep combining their results into larger solutions
  - eventually the original problem will be solved
- It is a recursive algorithm.
  - Divides the list into halves,
  - Sort each halve separately, and
  - Then merge the sorted halves into one sorted array.

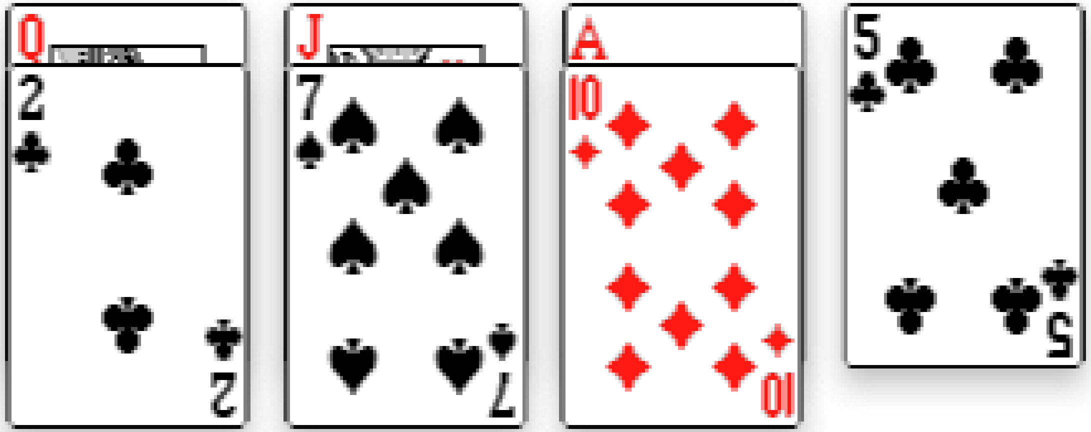
Example: sorting playing cards

- divide the cards into groups of two
- sort each group -- put the smaller of the two on the top
- merge groups of two into groups of four
- merge groups of four into groups of eight
- …

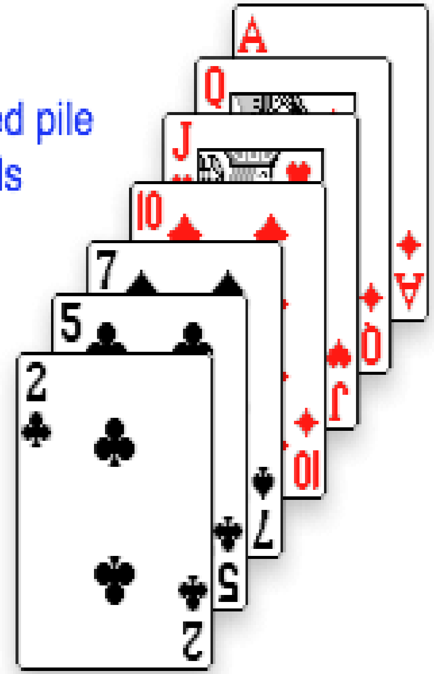
initial hand



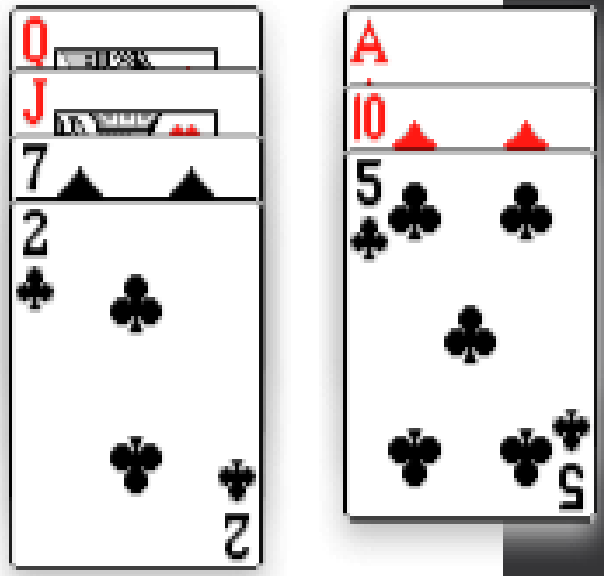
sorted piles of size two



final sorted pile of all cards



sorted piles of size four



In this example:

compare 2 with 5, pick up the 2
compare 5 with 7, pick up the 5
compare 7 with 10, pick up the 7
….

Merge Sort Procedure

Step 1 – if it is only one element in the list it is already sorted, return.

Step 2 – divide the list recursively into two halves until it can no more be divided.

Step 3 – merge the smaller lists into new list in sorted order.

Another Example

- Subdivide the sorting task

H	E	M	G	B	K	A	Q	F	L	P	D	R	C	J	N

H	E	M	G	B	K	A	Q

F	L	P	D	R	C	J	N

Subdivide again

H	E	M	G	B	K	A	Q

F	L	P	D	R	C	J	N

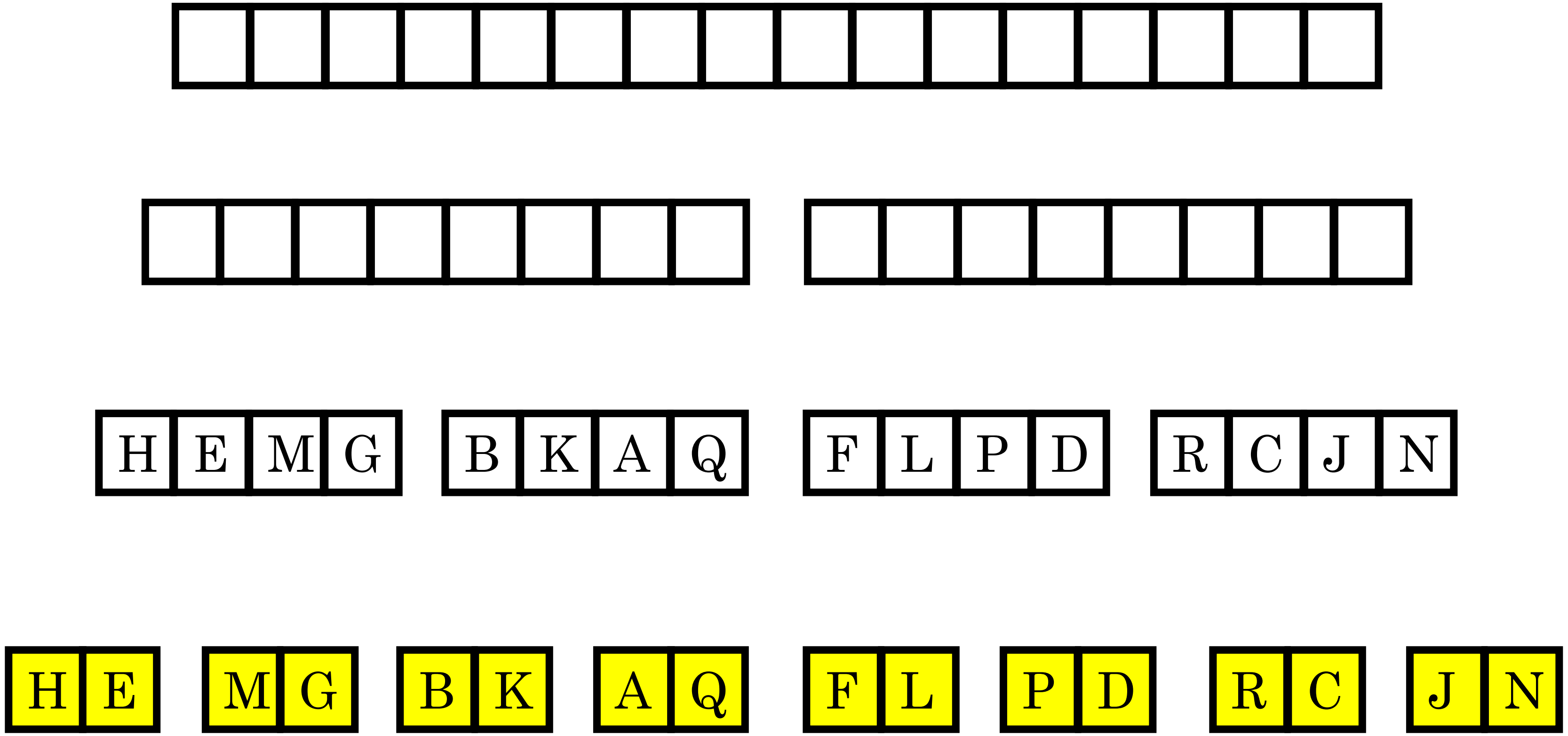
H	E	M	G

B	K	A	Q

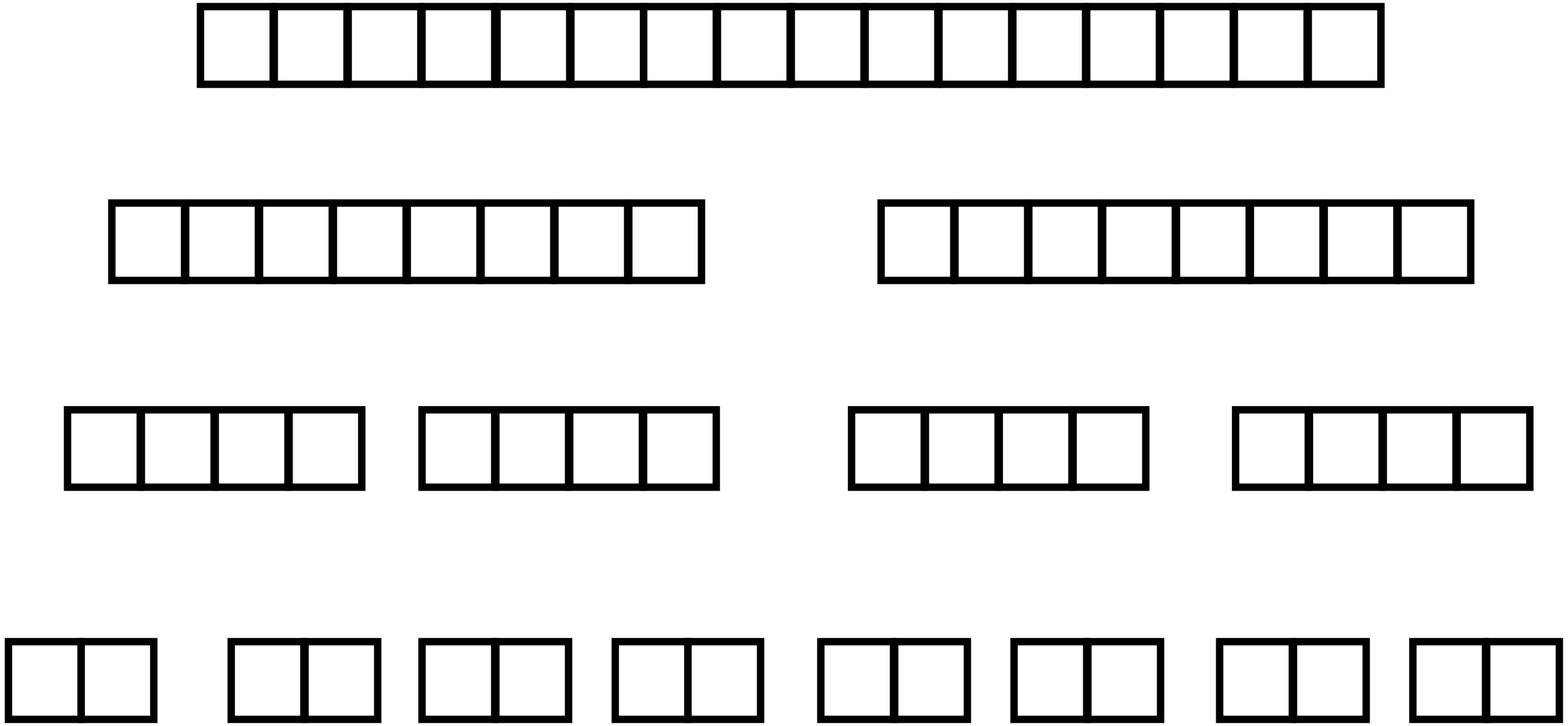
F	L	P	D

R	C	J	N

And again

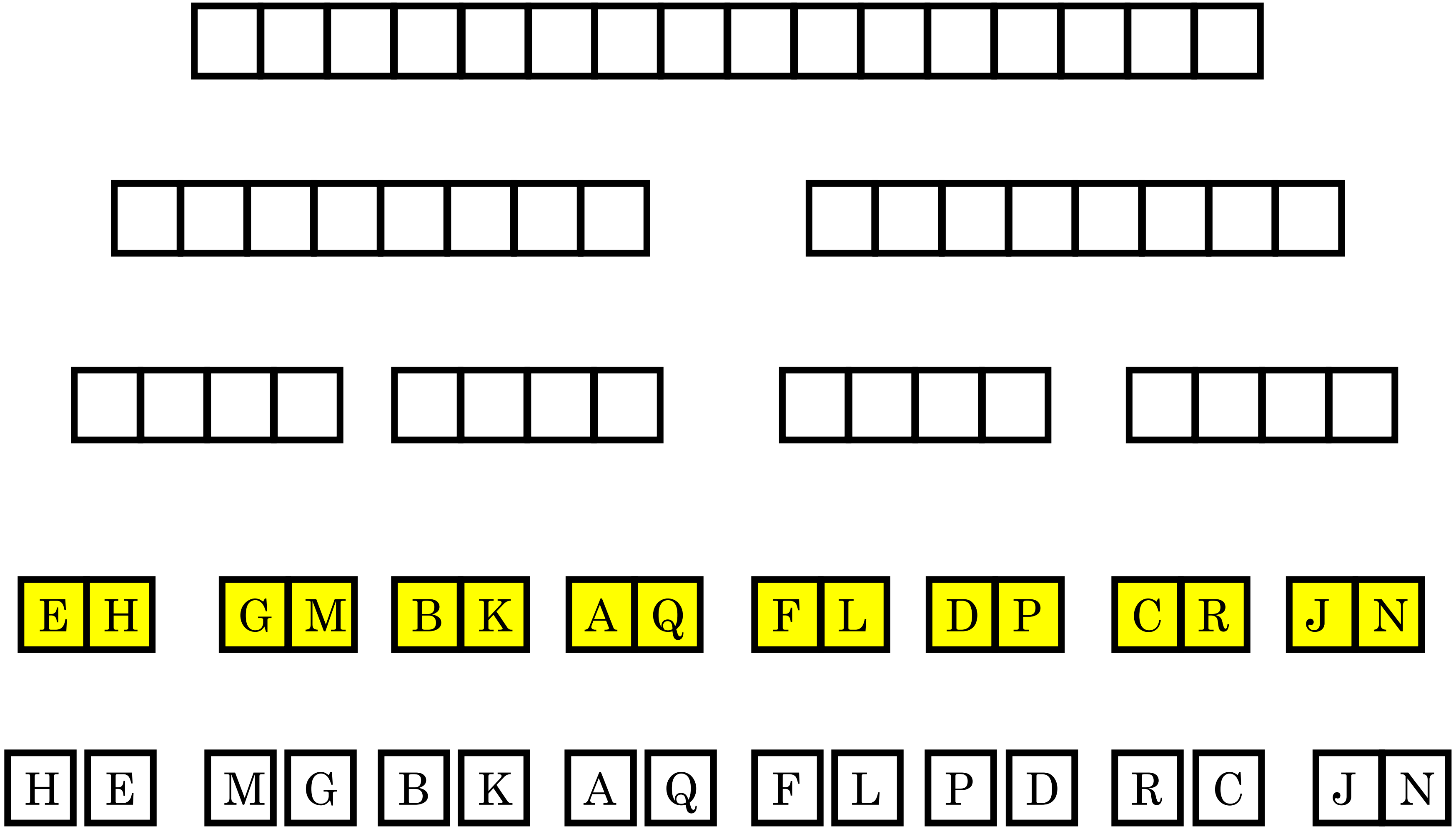


And one last time

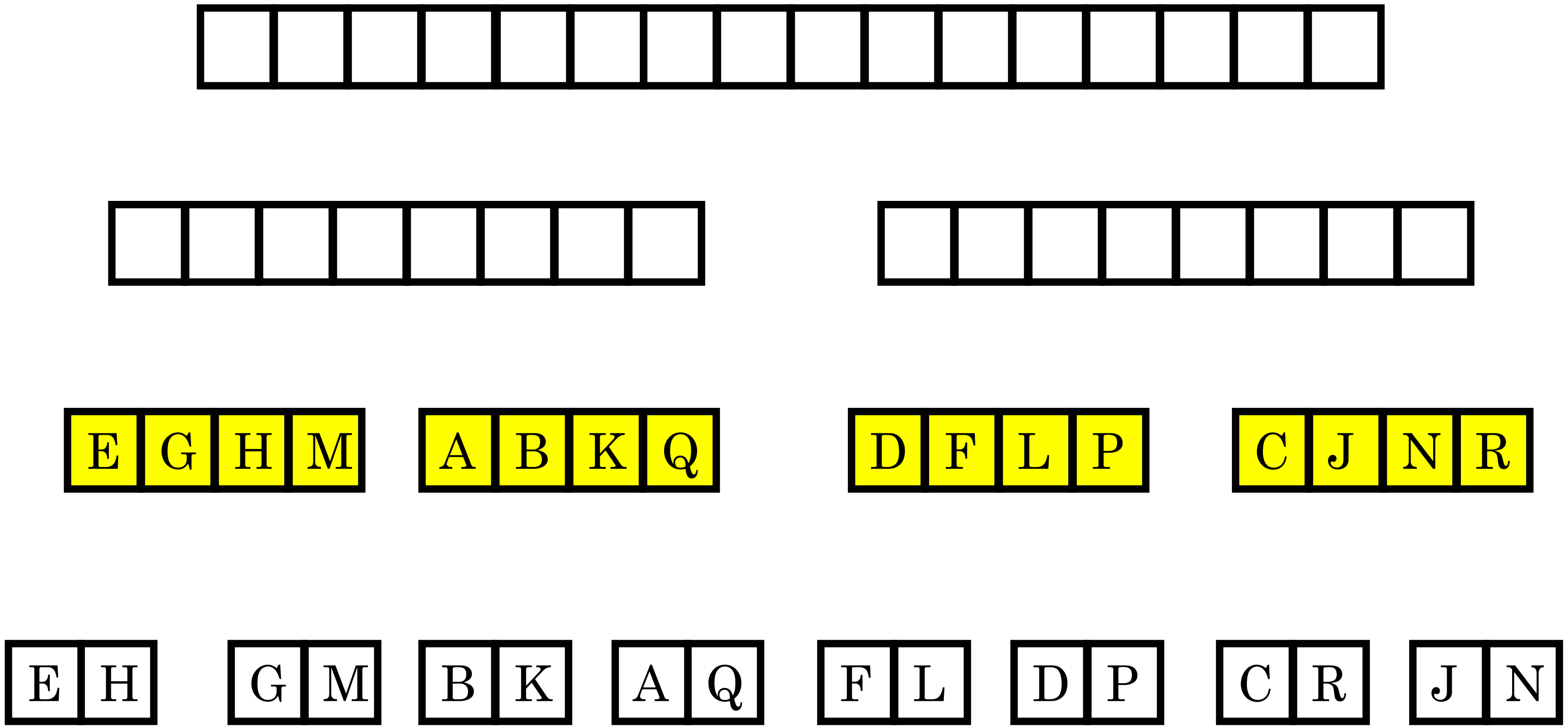


H E M G B K A Q F L P D R C J N

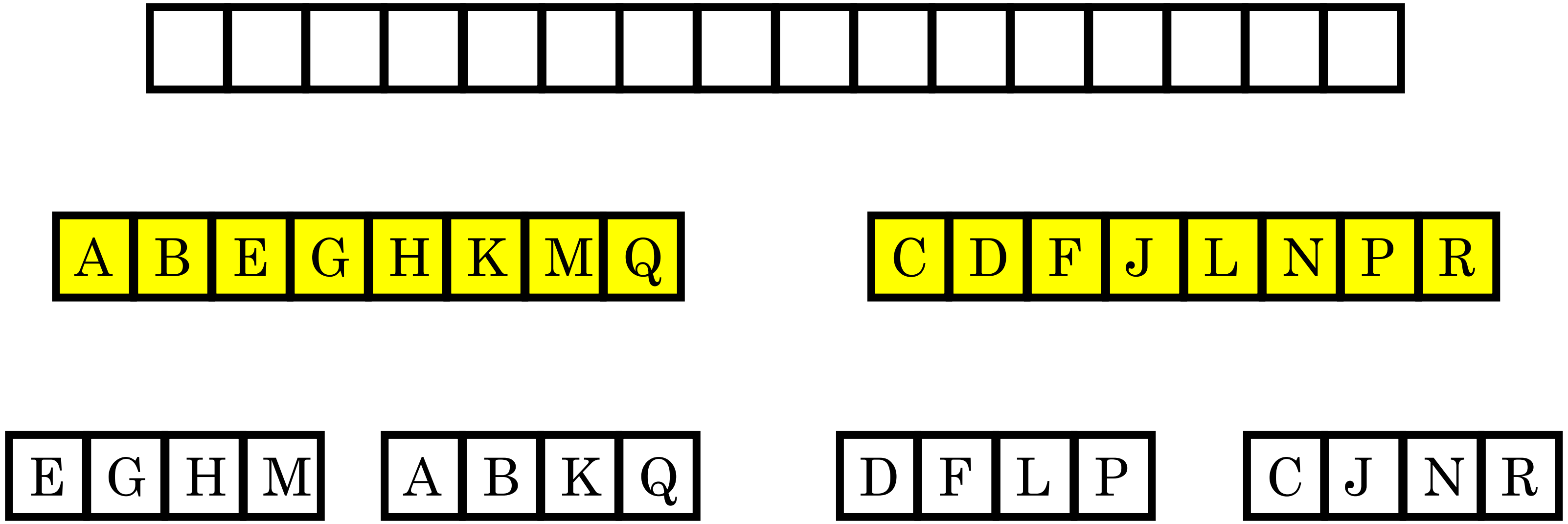
Now merge



And merge again



And again



And one last time

A	B	C	D	E	F	G	H	J	K	L	M	N	P	Q	R

A	B	E	G	H	K	M	Q

C	D	F	J	L	N	P	R

Done!

A	B	C	D	E	F	G	H	J	K	L	M	N	P	Q	R

Merging two sorted Array : Example

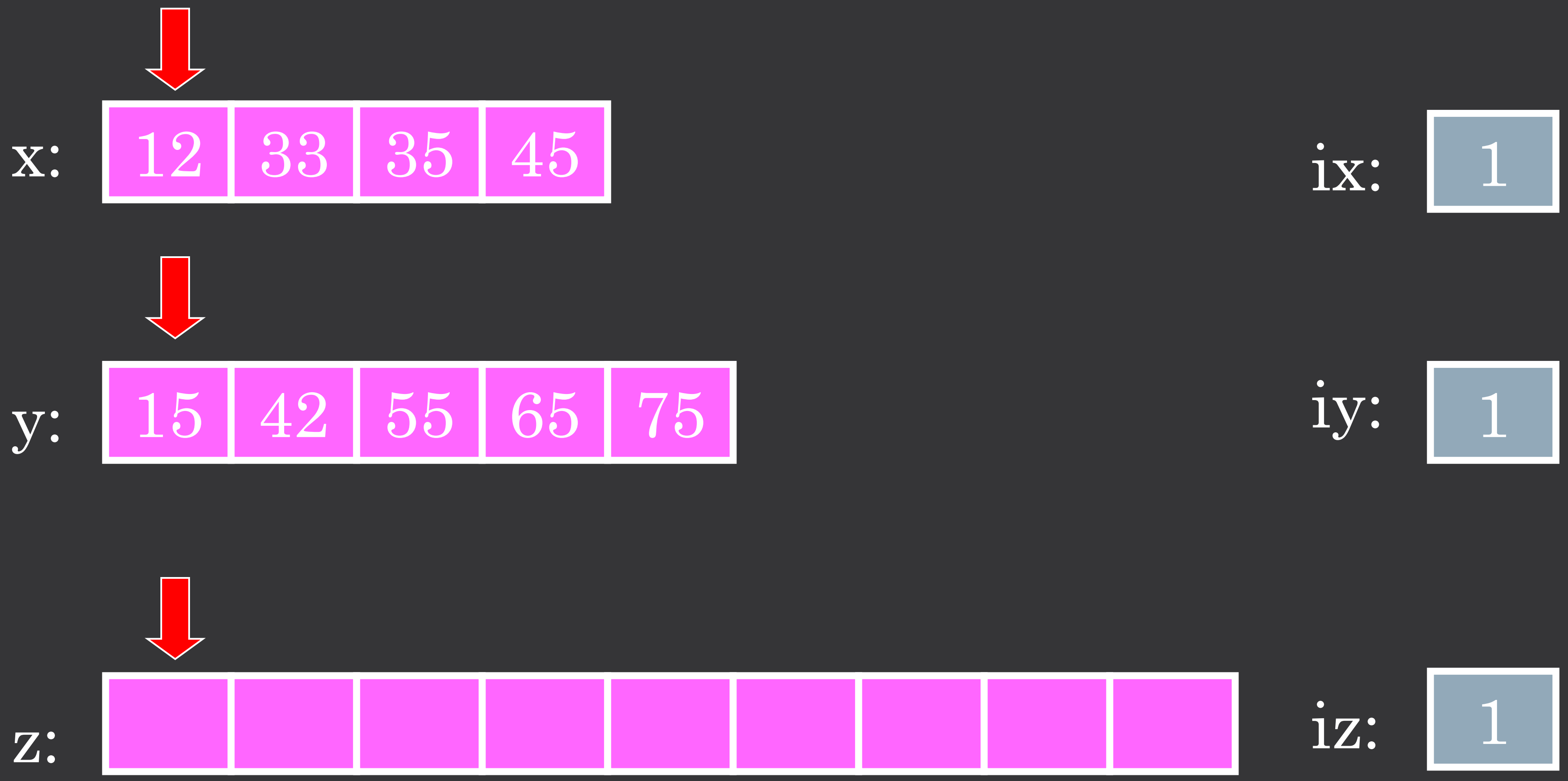
- The central sub-problem is the merging of two sorted arrays into one single sorted array

12	33	35	45

15	42	55	65	75

12	15	33	35	42	45	55	65	75

Merge



ix<=4 and iy<=5: x(ix) <= y(iy) ???

Merge

x: | 12 | 33 | 35 | 45 |

y: | 15 | 42 | 55 | 65 | 75 |

z: | 12 | | | | | | | | |

ix: 1

iy: 1

iz: 1

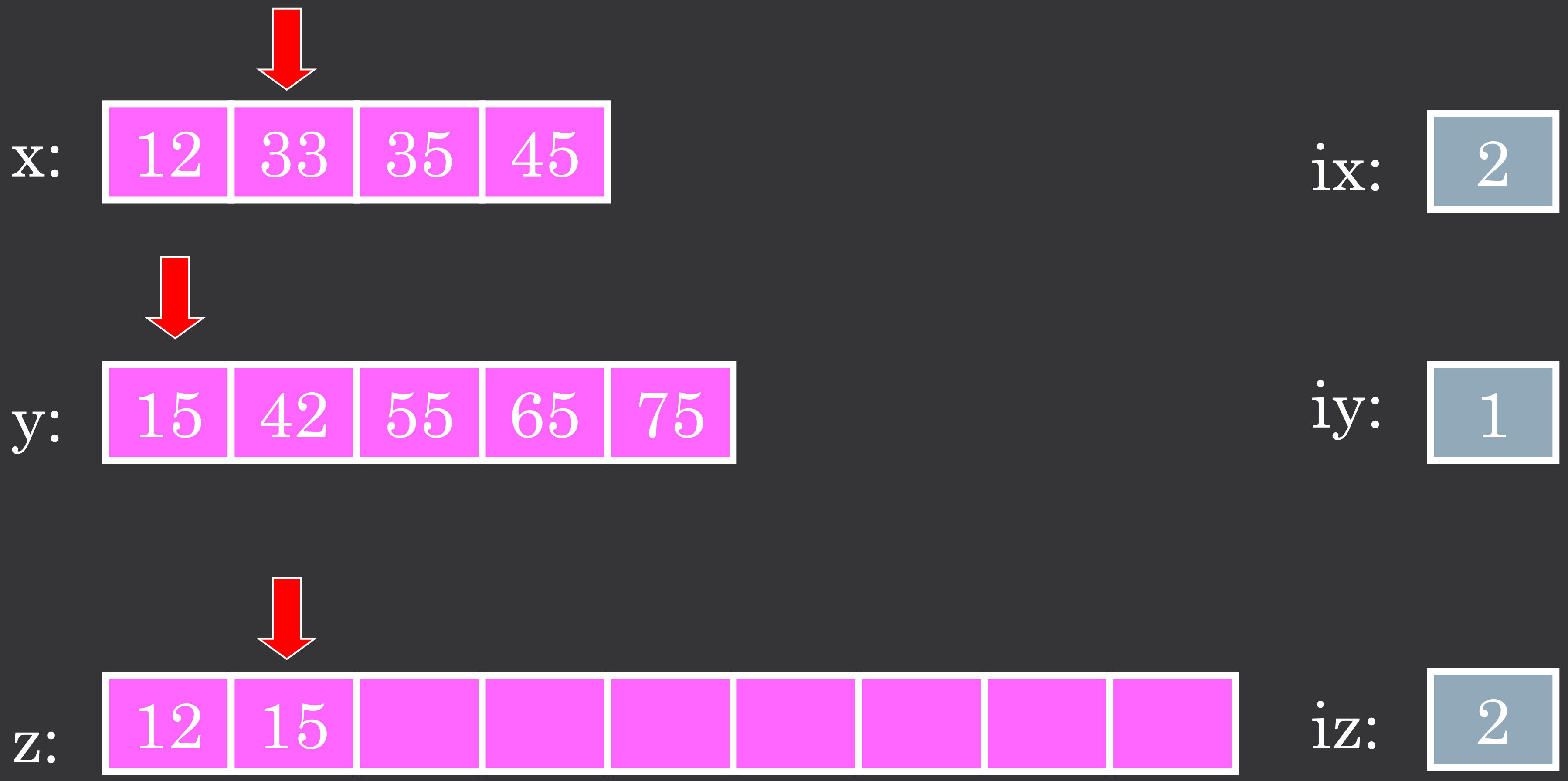
ix<=4 and iy<=5: x(ix) <= y(iy) YES

Merge

x:	12	33	35	45						ix:	2
y:	15	42	55	65	75					iy:	1
z:	12									iz:	2

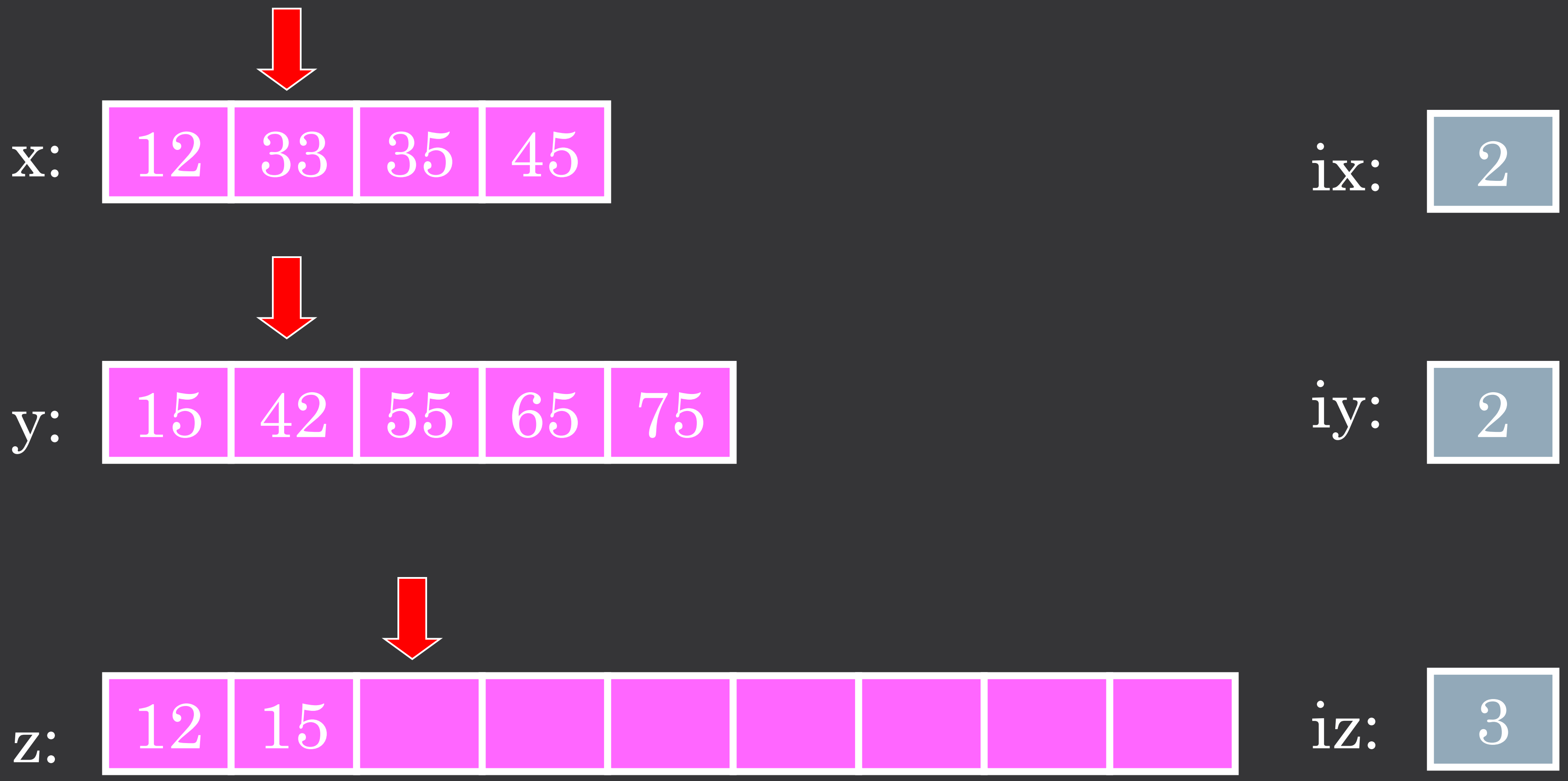
ix<=4 and iy<=5: x(ix) <= y(iy) ???

Merge



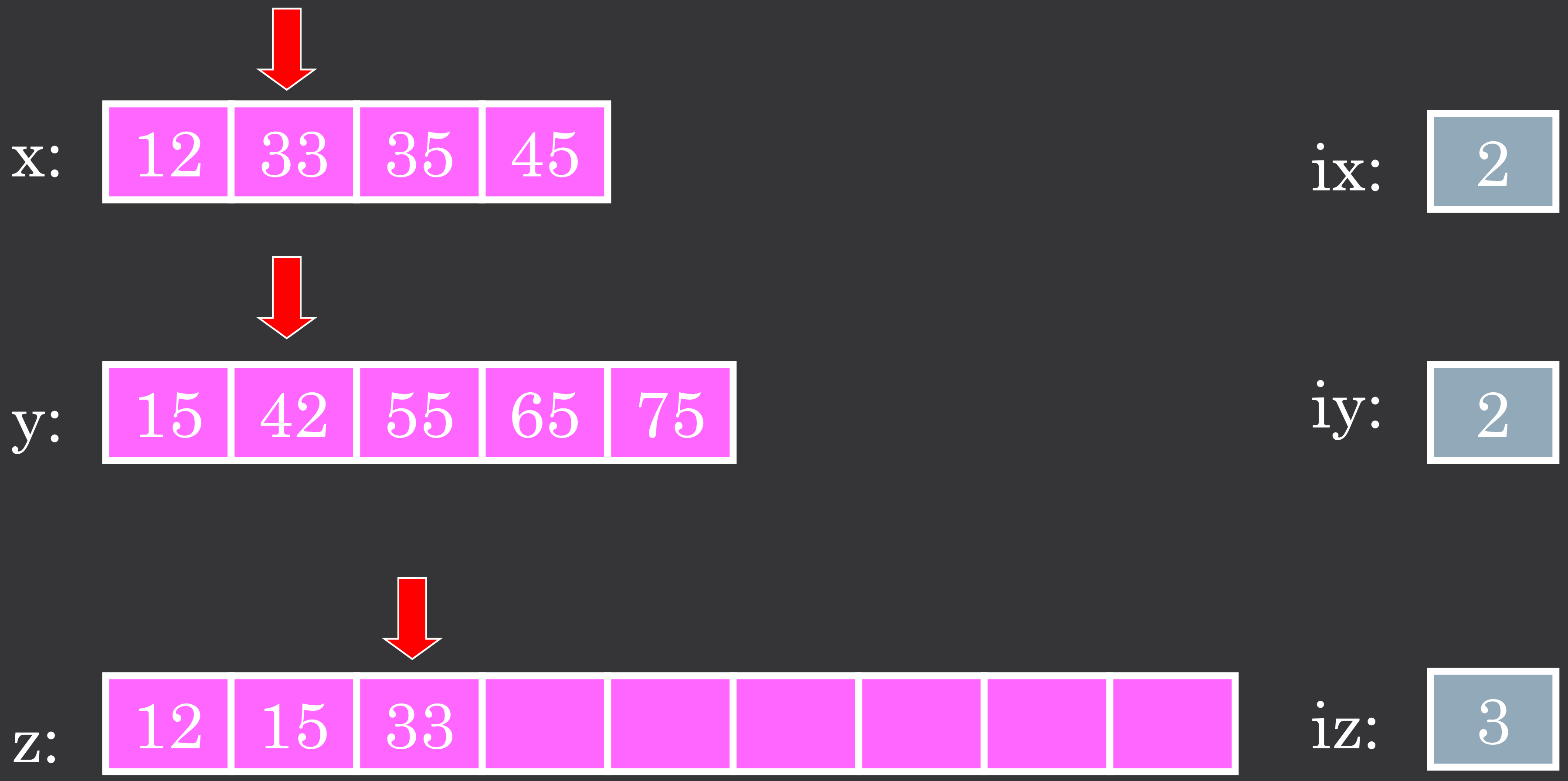
ix<=4 and iy<=5: x(ix) <= y(iy) NO

Merge



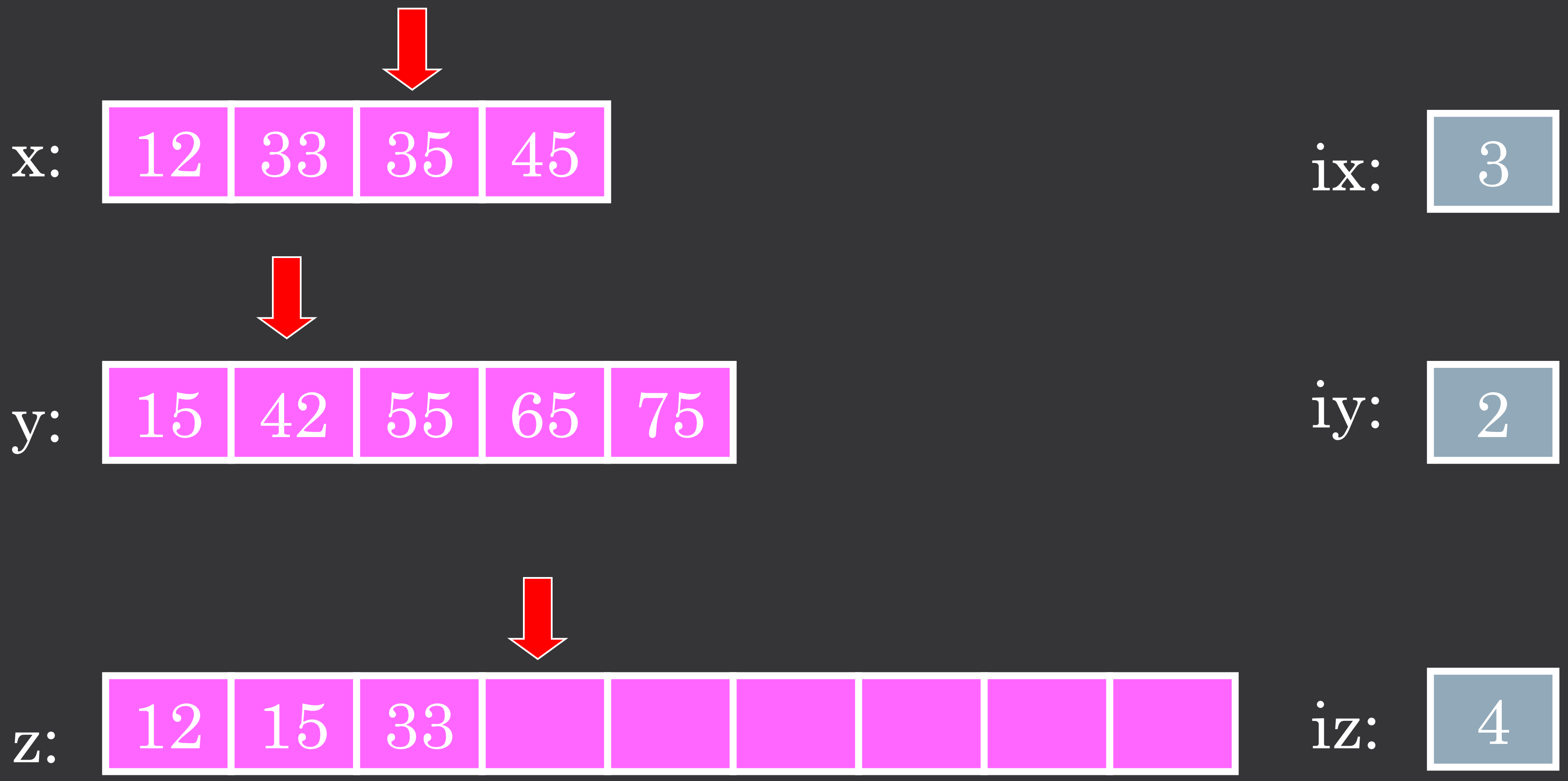
ix<=4 and iy<=5: x(ix) <= y(iy) ???

Merge



ix<=4 and iy<=5: x(ix) <= y(iy) YES

Merge



ix<=4 and iy<=5: x(ix) <= y(iy) ???

Merge

x: | 12 | 33 | 35 | 45 |

ix: 3

y: | 15 | 42 | 55 | 65 | 75 |

iy: 2

z: | 12 | 15 | 33 | 35 | | | | | |

iz: 4

ix<=4 and iy<=5: x(ix) <= y(iy) YES

Merge

x: 12 33 35 45

ix: 4

y: 15 42 55 65 75

iy: 2

z: 12 15 33 35

iz: 5

ix<=4 and iy<=5: x(ix) <= y(iy) ???

Merge

x: | 12 | 33 | 35 | 45 |

ix: 4

y: | 15 | 42 | 55 | 65 | 75 |

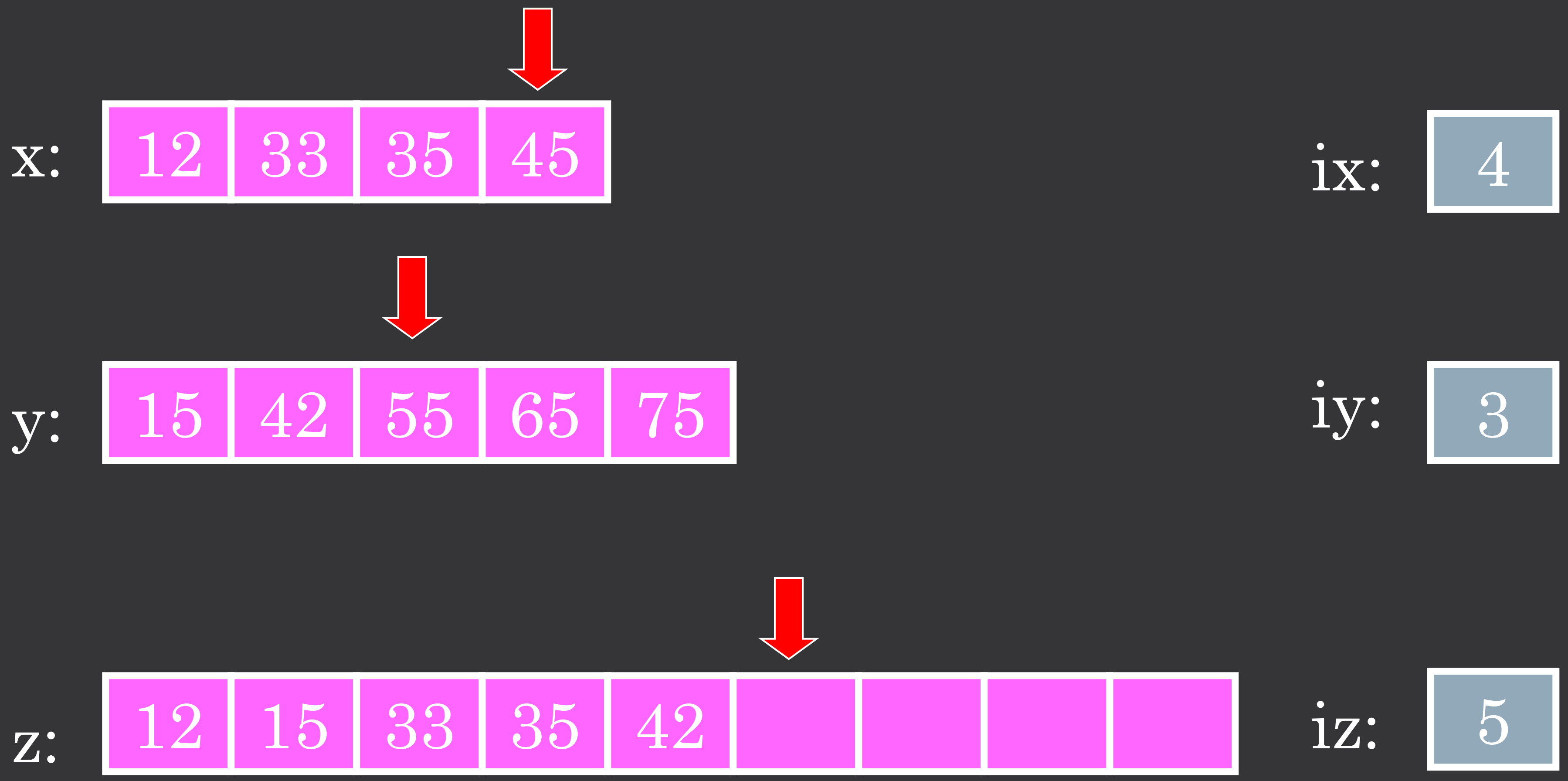
iy: 2

z: | 12 | 15 | 33 | 35 | 42 | | | | |

iz: 5

ix<=4 and iy<=5: x(ix) <= y(iy) NO

Merge



ix<=4 and iy<=5: x(ix) <= y(iy) ???

Merge

x:

12	33	35	45

ix: 4

y:

15	42	55	65	75

iy: 3

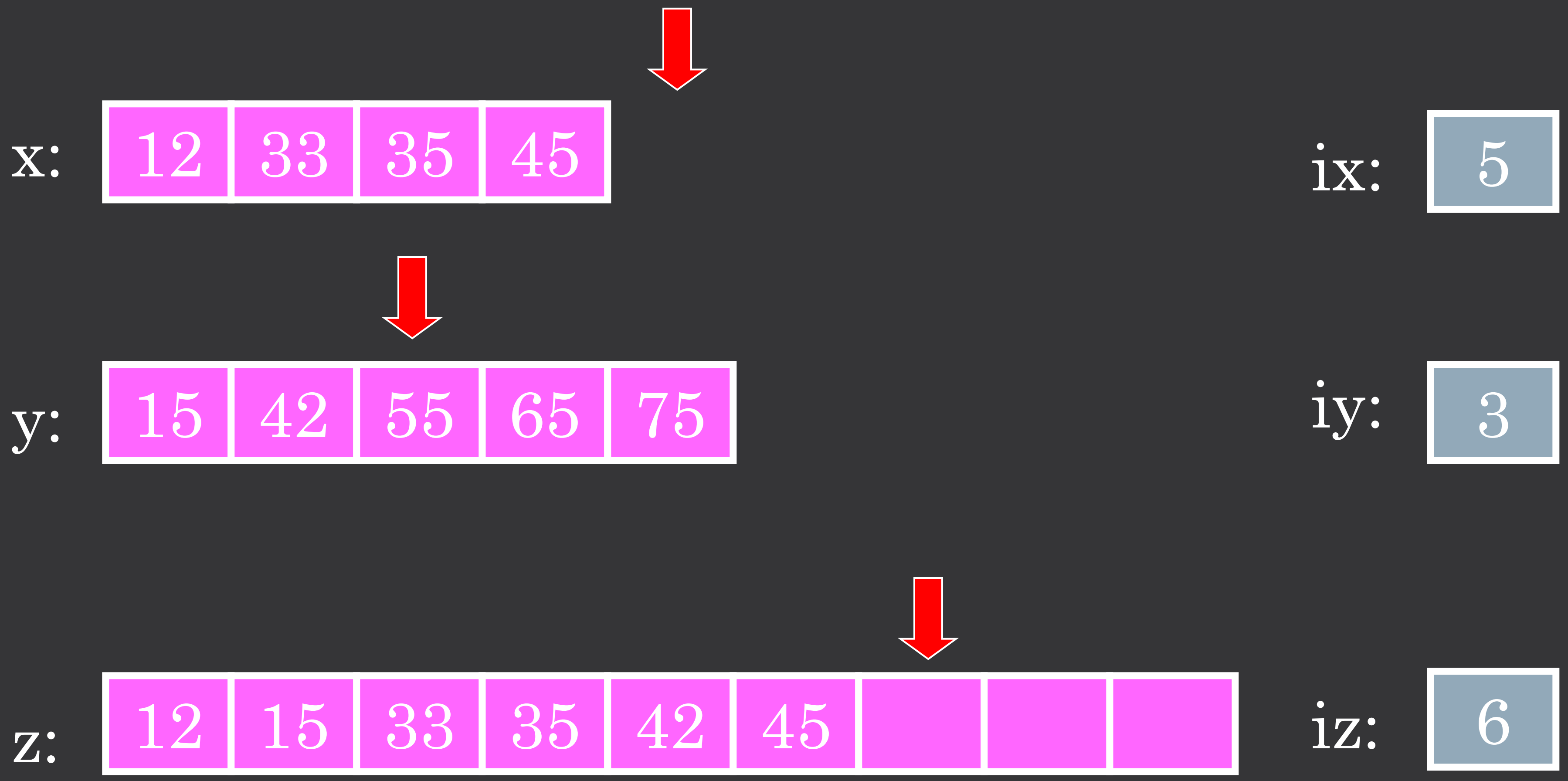
z:

12	15	33	35	42	45			

iz: 5

ix<=4 and iy<=5: x(ix) <= y(iy) YES

Merge



$ix > 4$

Merge

x:	12	33	35	45

ix: 5

y:	15	42	55	65	75

iy: 3

z:	12	15	33	35	42	45	55		

iz: 6

ix > 4: take y(iy)

Merge

x:	12	33	35	45

y:	15	42	55	65	75

z:	12	15	33	35	42	45	55		

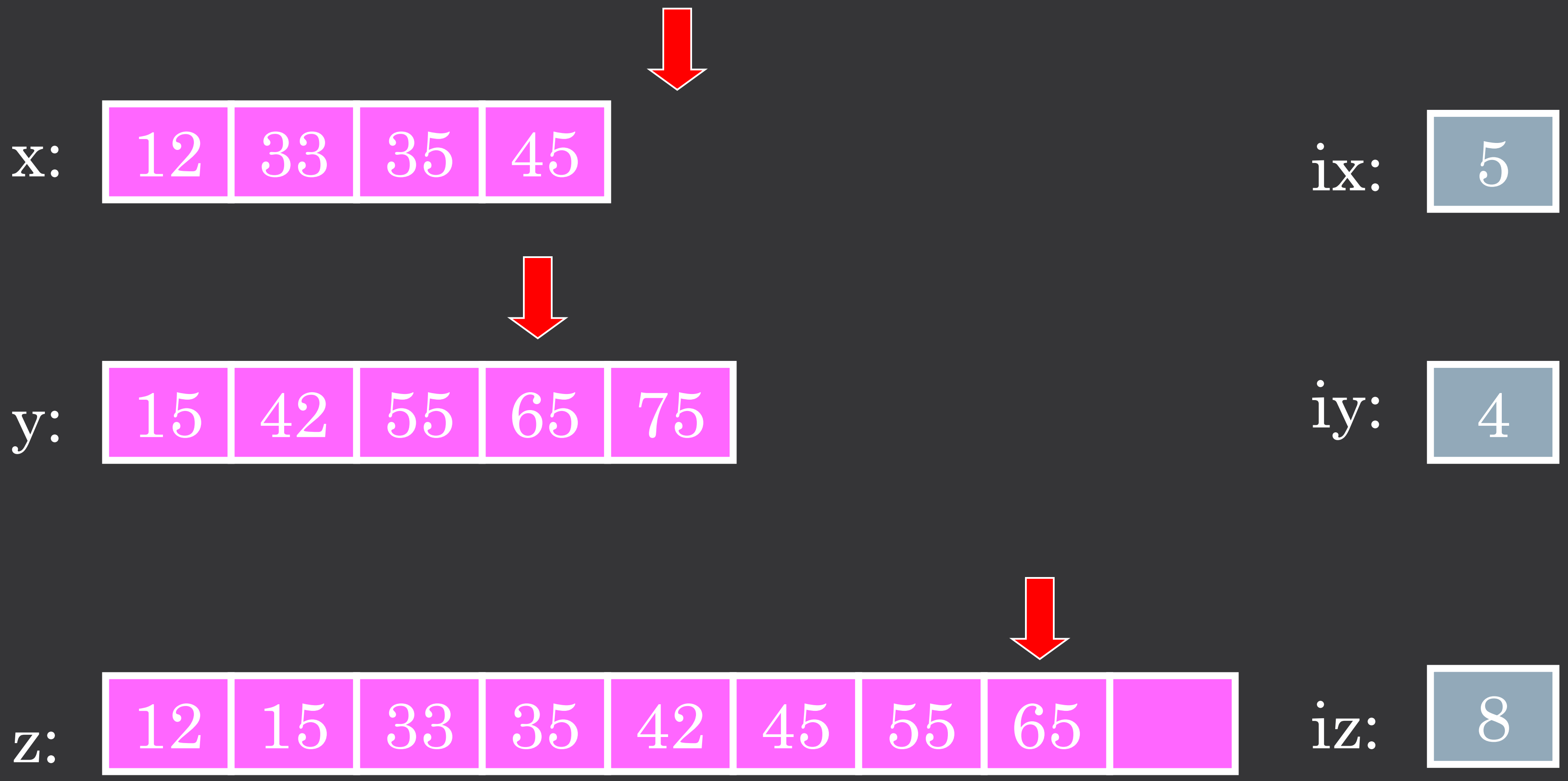
ix: 5

iy: 4

iz: 8

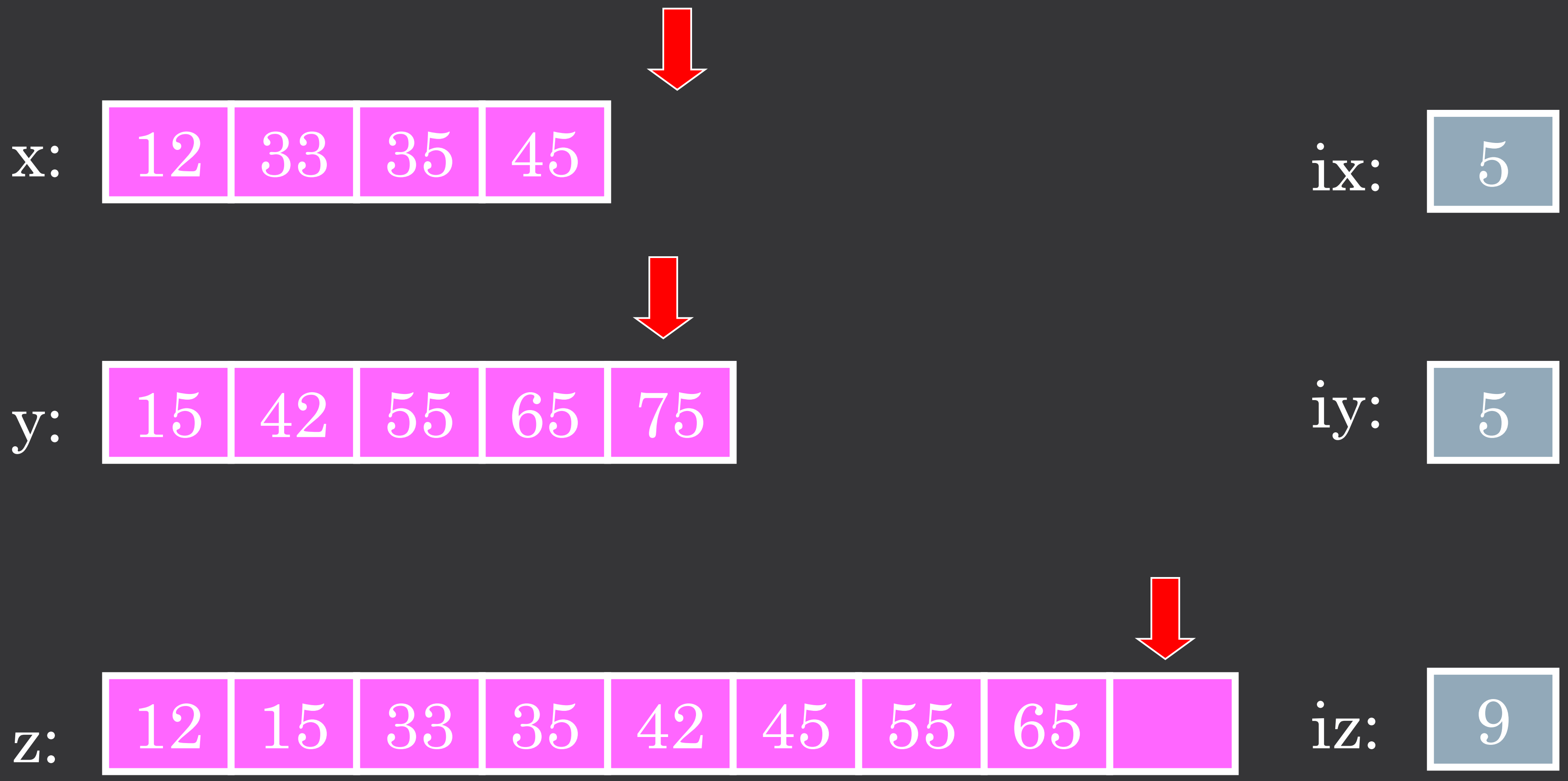
iy <= 5

Merge



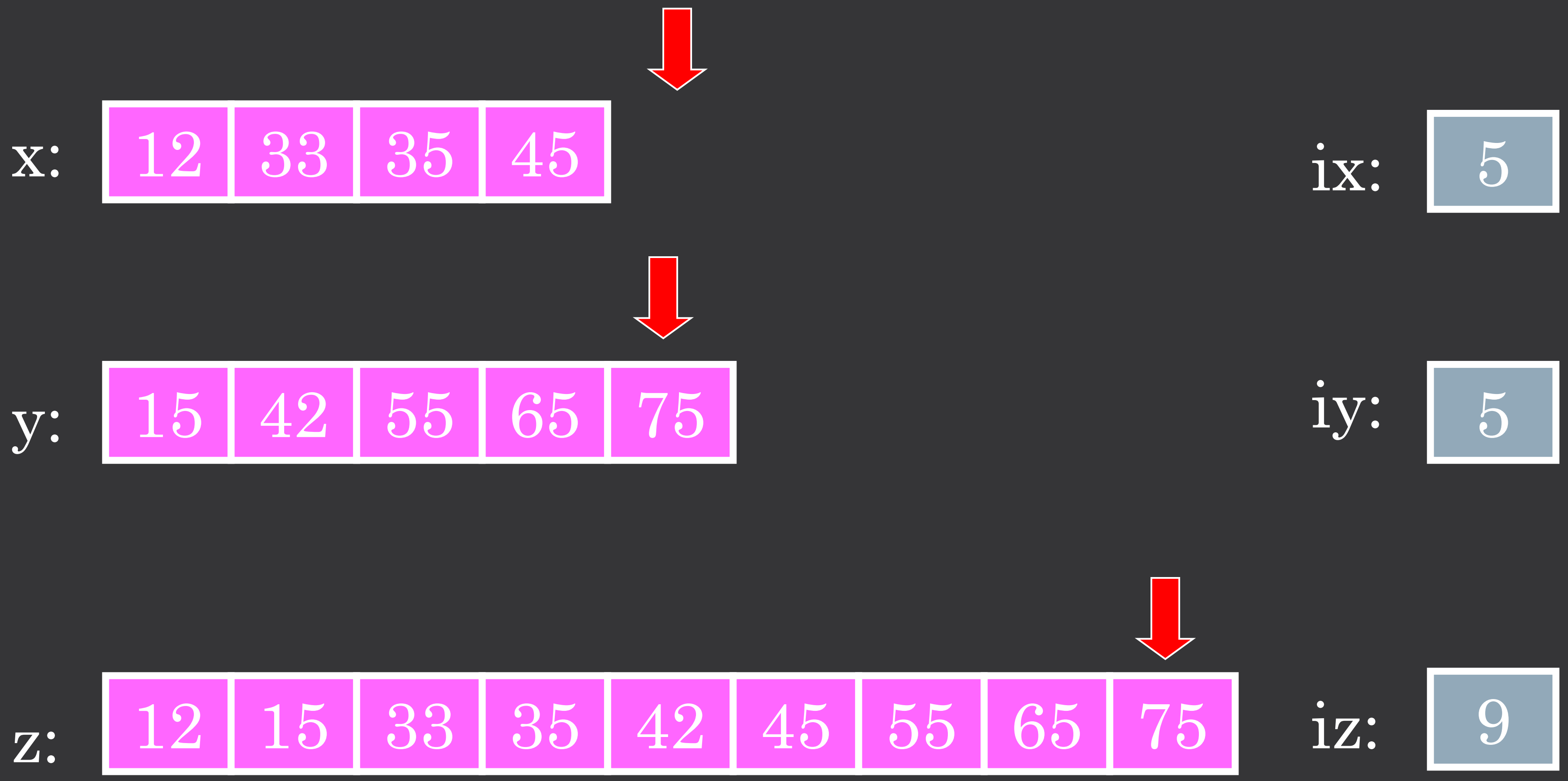
iy <= 5

Merge



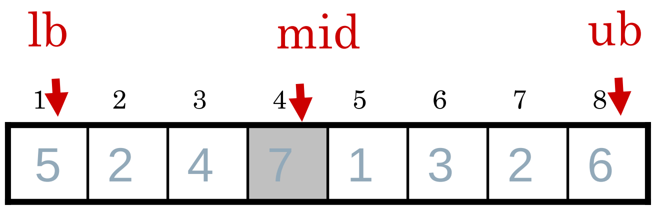
iy <= 5

Merge



iy <= 5

Merge Sort Algorithm



Alg.: MERGE-SORT(A, lb, up)

if lb < ub

{

mid $\leftarrow \lfloor (lb + ub)/2 \rfloor$

MERGE-SORT(A, lb, mid)

MERGE-SORT(A, mid + 1, ub)

MERGE(A, lb, mid, ub)

}

Merge Algorithm

```
Merge(A, lb, mid, ub)
{
        i=lb;j=mid+1;k=lb;
        while(i<= mid && j <=
        ub)
              {
                    if(A[i] <= A[j])
                    {
                          B[k]=A[i];
                          i++; k++;
                    }
                    else
                       {
                          B[k]=A[j
                          ]; j++;
                          k++;
              }           }

        while( i <= mid)
          {
              B[k]=A[i];
              i++; k++;
          }
          while( j <= ub)
          {
              B[k]=A[j]; j++;
              k++;
          }
        for(i=lb; i<=ub;i++)
         {A[k]=B[i]; }
}
```

Merge Sort - Discussion

- **Running time insensitive of the input**

- **Advantages:**
  - **Mergesort is extremely efficient algorithm with respect to time.**
  - **Guaranteed to run in $\Theta(nlogn)$**

- **Disadvantage**
  - **Mergesort requires an extra array whose size equals to the size of the original array.(Not Inplace sorting)**
  - **So, it requires extra space ≈N = $O(n)$**