Sorting

Data and File Structures Laboratory

http://www.isical.ac.in/~dfslab/2017/index.html

Indian Statistical Institute Kolkata

September 14, 2017

Outline

The problem of sorting

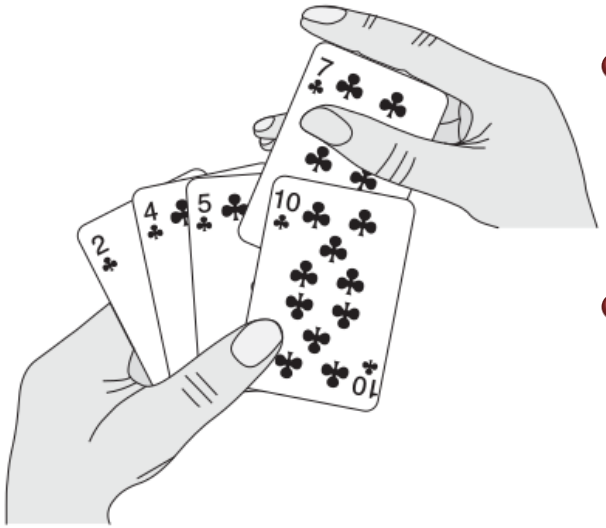
Input: sequence $< a1, a2, \ldots, an >$ of numbers.

Output: permutation $< a'1, a'2, \ldots, a'n >$ such that $a'1 \leq a'2 \leq \ldots \leq a'n$.

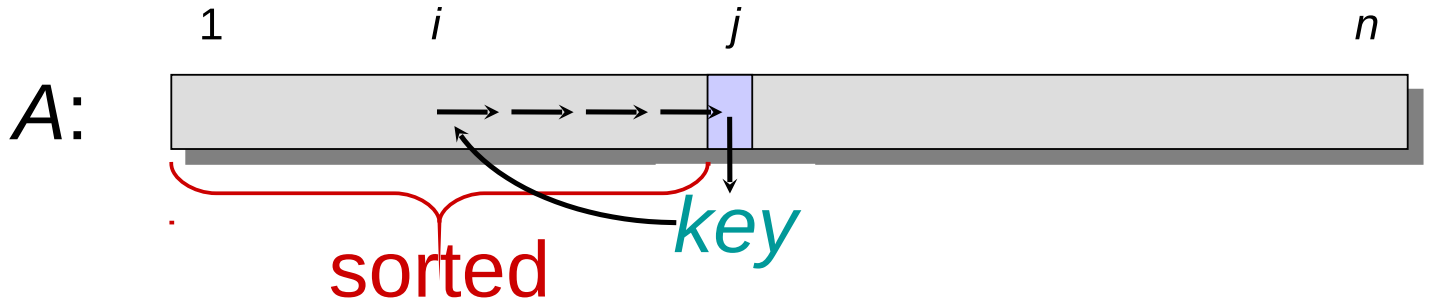
Example:
Input: 8 2 4 9 3 6
Output: 2 3 4 6 8 9

Outline

Insertion sort



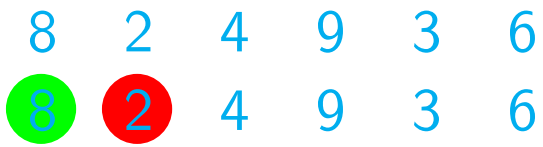
- Start with an empty left hand and the cards face down on the table.
- Remove one card at a time from the table and insert it into the correct position in the left hand.
- To find the correct position for a card, we compare it with each of the cards already in the hand, from right to left
- At all times, the cards held in the left hand are sorted, and these cards were originally the top cards of the pile on the table.



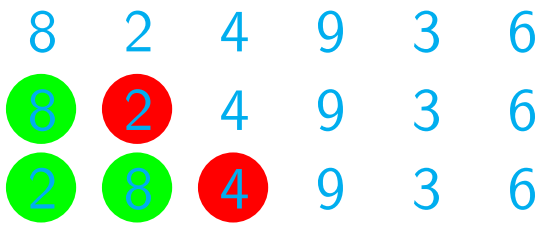
Example of insertion sort

8 2 4 9 3 6

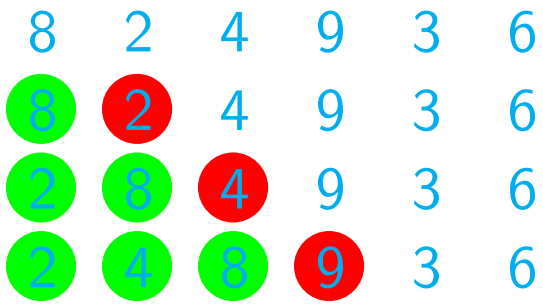
Example of insertion sort



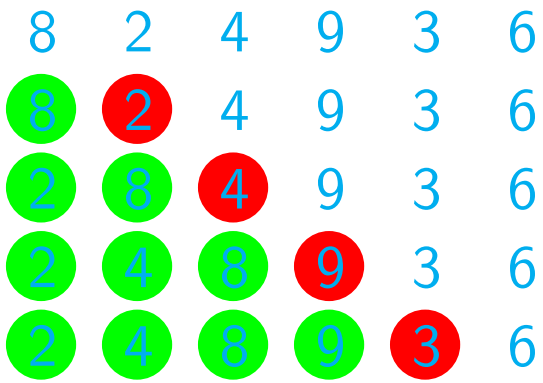
Example of insertion sort



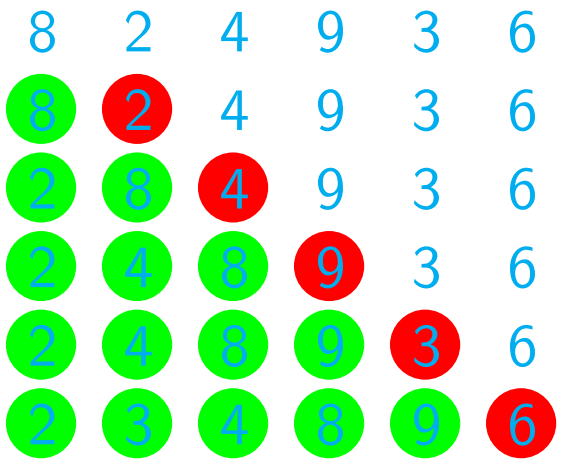
Example of insertion sort



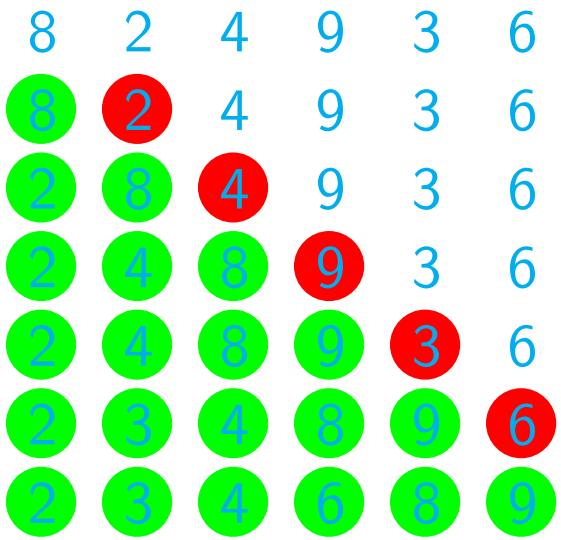
Example of insertion sort



Example of insertion sort



Example of insertion sort



Insertion Sort Pseudocode

INSERTION-SORT(A)

```
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1..j-1].
4      i = j - 1
5      while i > 0 and A[i] > key
6          A[i+1] = A[i]
7          i = i - 1
8      A[i+1] = key
```

Outline

Bubble Sort

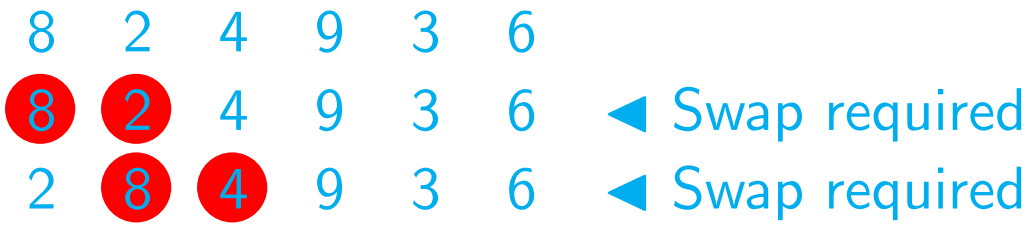
- Starts at the beginning of the data set.
- Compares the first two elements.
- If the first is greater than the second, then it swaps them.
- It continues doing this for each pair of adjacent elements to the end of the data set.
- It then starts again with the first two elements.
- Repeating until no swaps have occurred on the last pass.

Example of bubble sort

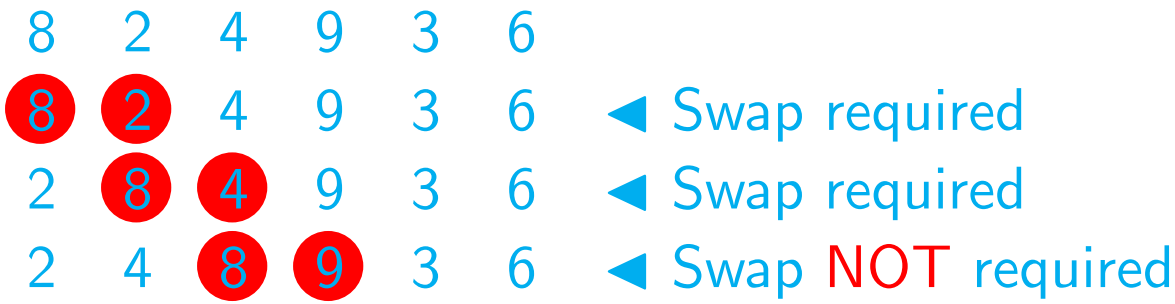
8 2 4 9 3 6

Example of bubble sort

Example of bubble sort



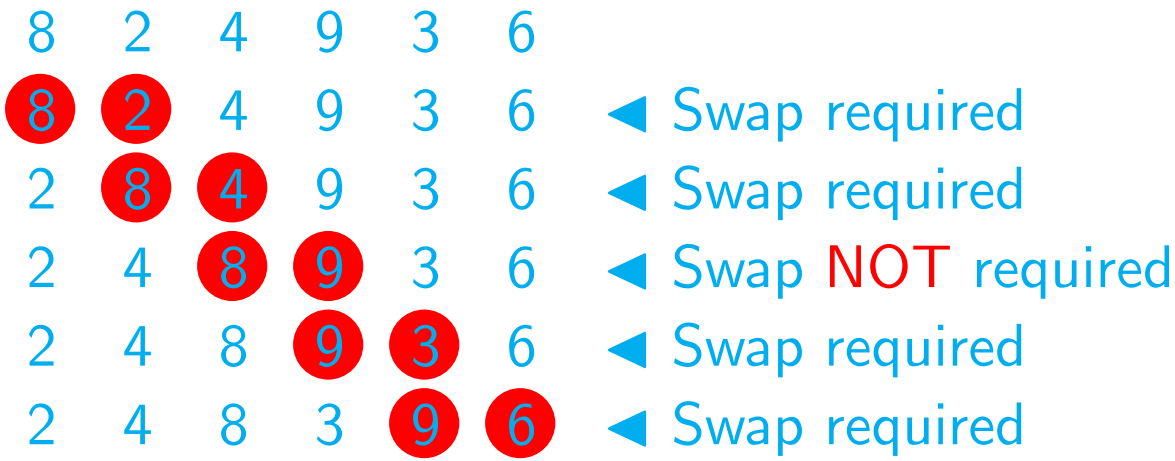
Example of bubble sort



Example of bubble sort

8	2	4	9	3	6	
8	2	4	9	3	6	◀ Swap required
2	8	4	9	3	6	◀ Swap required
2	4	8	9	3	6	◀ Swap NOT required
2	4	8	9	3	6	◀ Swap required

Example of bubble sort

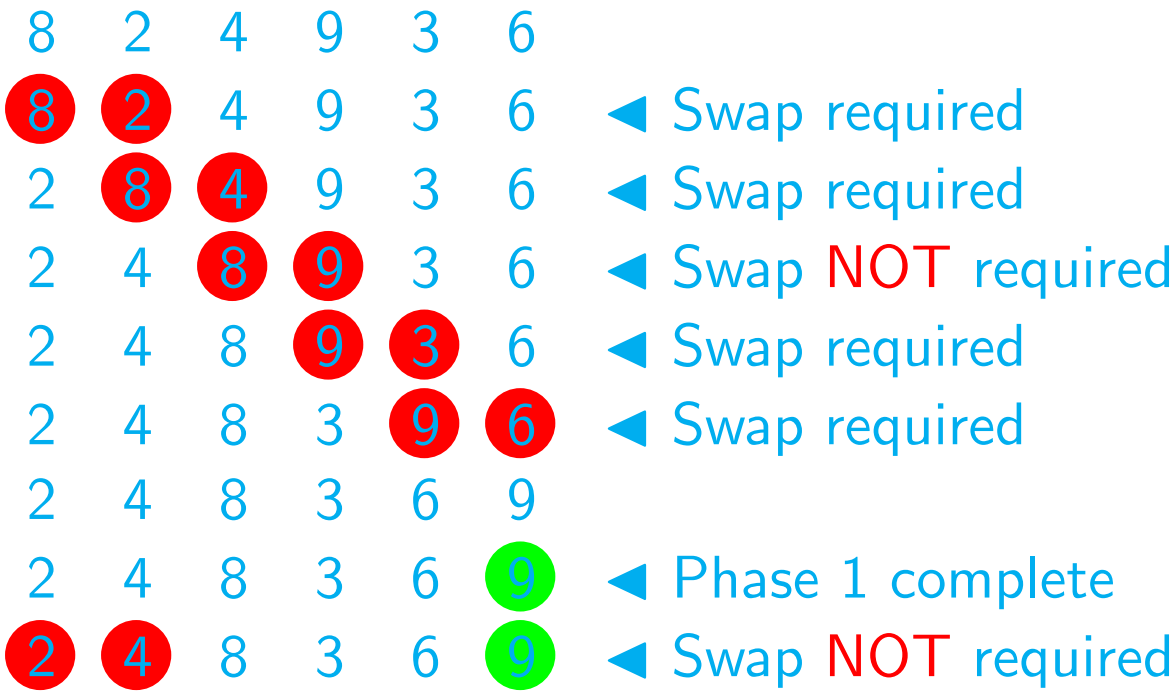


Example of bubble sort

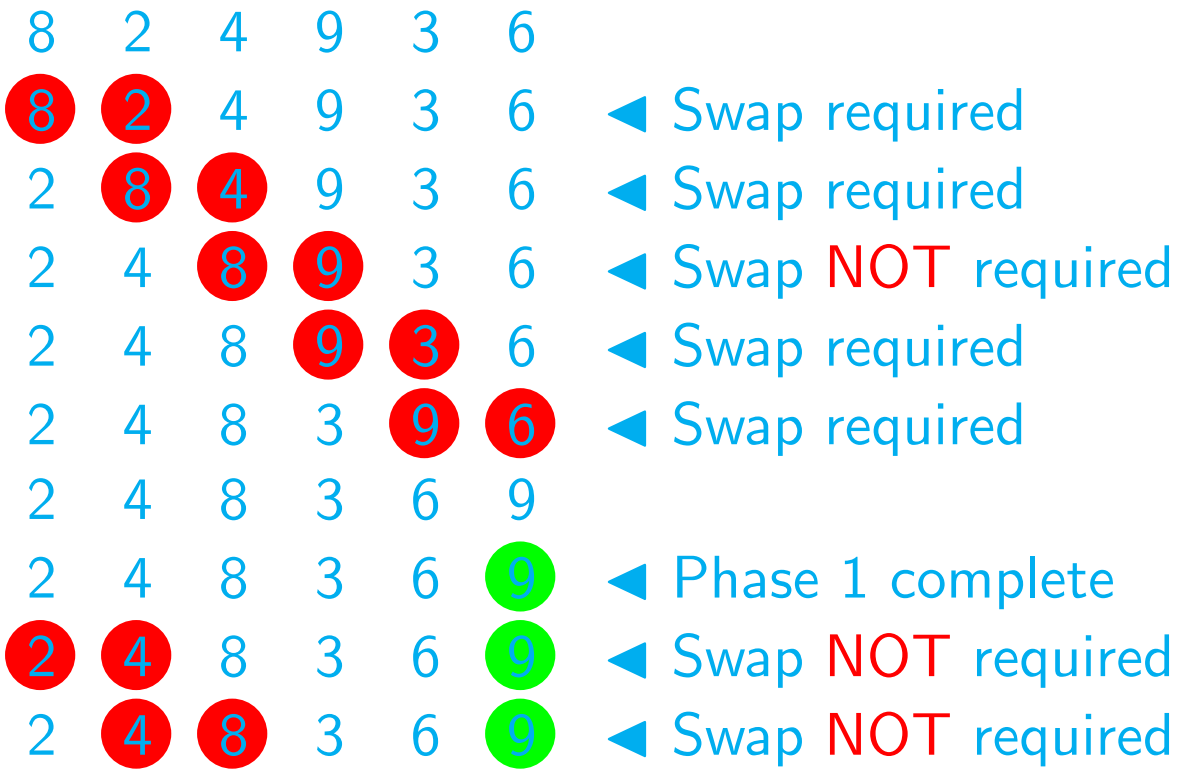
Example of bubble sort

8	2	4	9	3	6	
8	2	4	9	3	6	◀ Swap required
2	8	4	9	3	6	◀ Swap required
2	4	8	9	3	6	◀ Swap NOT required
2	4	8	9	3	6	◀ Swap required
2	4	8	3	9	6	◀ Swap required
2	4	8	3	6	9	
2	4	8	3	6	9	◀ Phase 1 complete

Example of bubble sort



Example of bubble sort



Example of bubble sort

8 2 4 9 3 6	
8 2 4 9 3 6	◀ Swap required
2 8 4 9 3 6	◀ Swap required
2 4 8 9 3 6	◀ Swap NOT required
2 4 8 9 3 6	◀ Swap required
2 4 8 3 9 6	◀ Swap required
2 4 8 3 6 9	
2 4 8 3 6 9	◀ Phase 1 complete
2 4 8 3 6 9	◀ Swap NOT required
2 4 8 3 6 9	◀ Swap NOT required
2 4 8 3 6 9	◀ Swap required

Example of bubble sort

8	2	4	9	3	6	
8	2	4	9	3	6	◀ Swap required
2	8	4	9	3	6	◀ Swap required
2	4	8	9	3	6	◀ Swap NOT required
2	4	8	9	3	6	◀ Swap required
2	4	8	3	9	6	◀ Swap required
2	4	8	3	6	9	
2	4	8	3	6	9	◀ Phase 1 complete
2	4	8	3	6	9	◀ Swap NOT required
2	4	8	3	6	9	◀ Swap NOT required
2	4	8	3	6	9	◀ Swap required
2	4	3	8	6	9	◀ Swap required

Example of bubble sort

8	2	4	9	3	6	
8	2	4	9	3	6	◀ Swap required
2	8	4	9	3	6	◀ Swap required
2	4	8	9	3	6	◀ Swap NOT required
2	4	8	9	3	6	◀ Swap required
2	4	8	3	9	6	◀ Swap required
2	4	8	3	6	9	
2	4	8	3	6	9	◀ Phase 1 complete
2	4	8	3	6	9	◀ Swap NOT required
2	4	8	3	6	9	◀ Swap NOT required
2	4	8	3	6	9	◀ Swap required
2	4	3	8	6	9	◀ Swap required
2	4	3	6	8	9	◀ Phase 2 complete

Bubble Sort Pseudocode

BUBBLESORT(A)

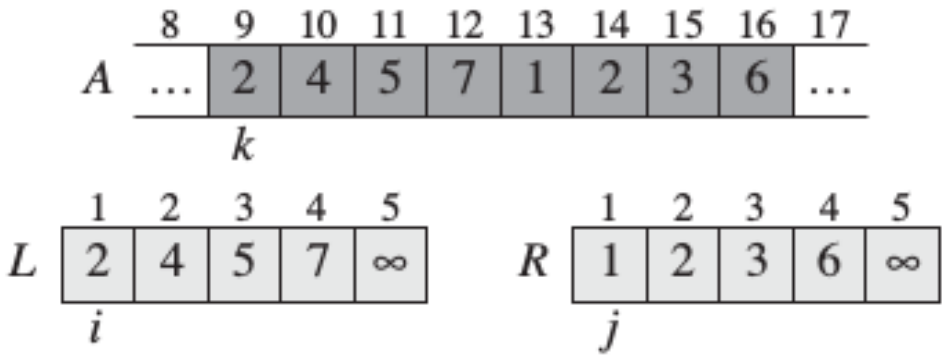
```
1  for i = 1 to A.length − 1
2      for j = A.length downto i + 1
3          if A[j] < A[j − 1]
4              Exchange A[j] with A[j − 1]
```

Outline

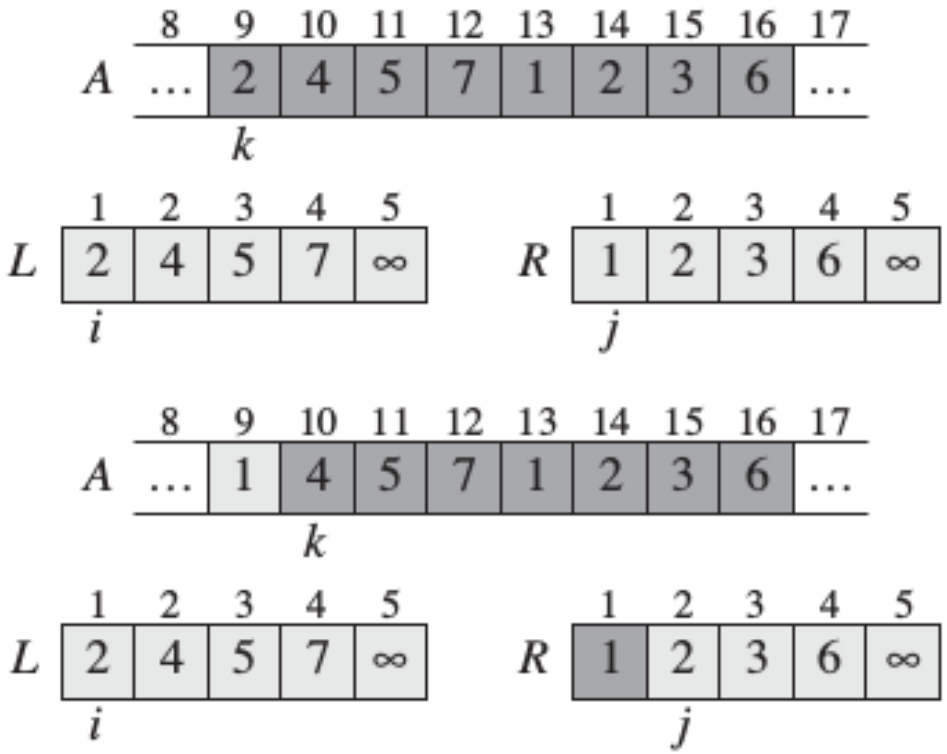
Merge Procedure

Question: Suppose we have two sorted array $L[p..q]$ and $R[q+1..r]$. How to merges them to form a single sorted array $A[p..r]$?

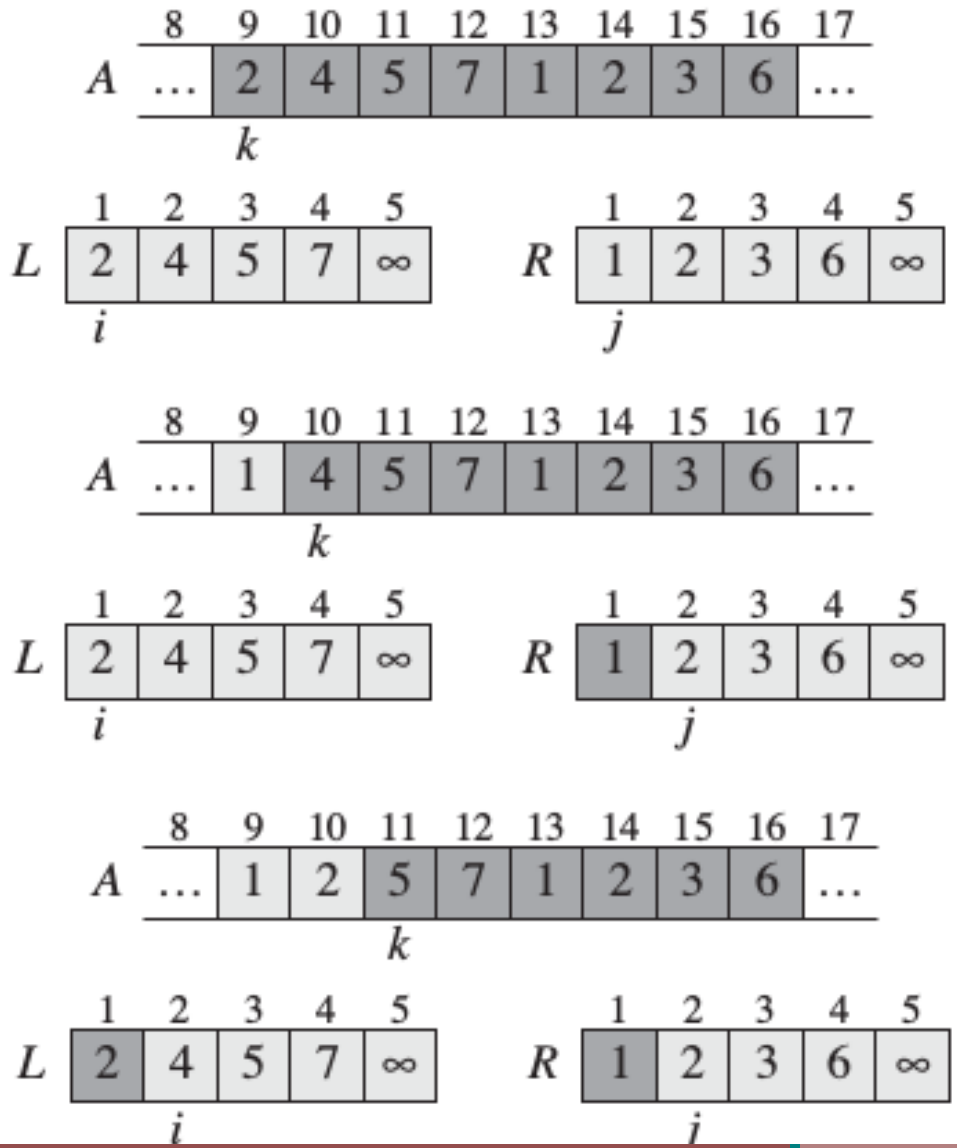
Merge Procedure



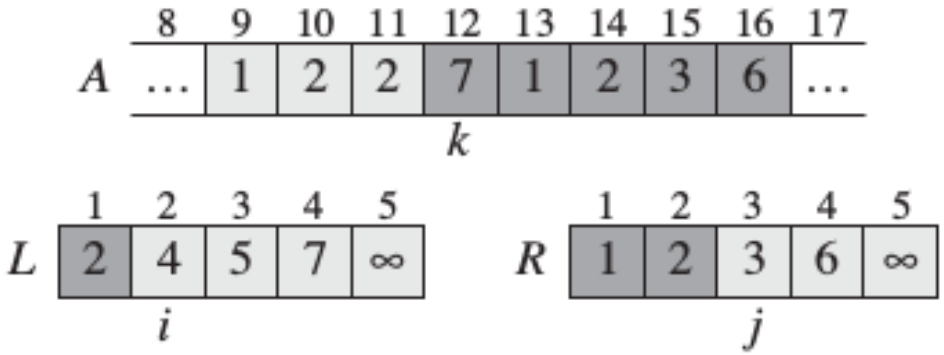
Merge Procedure



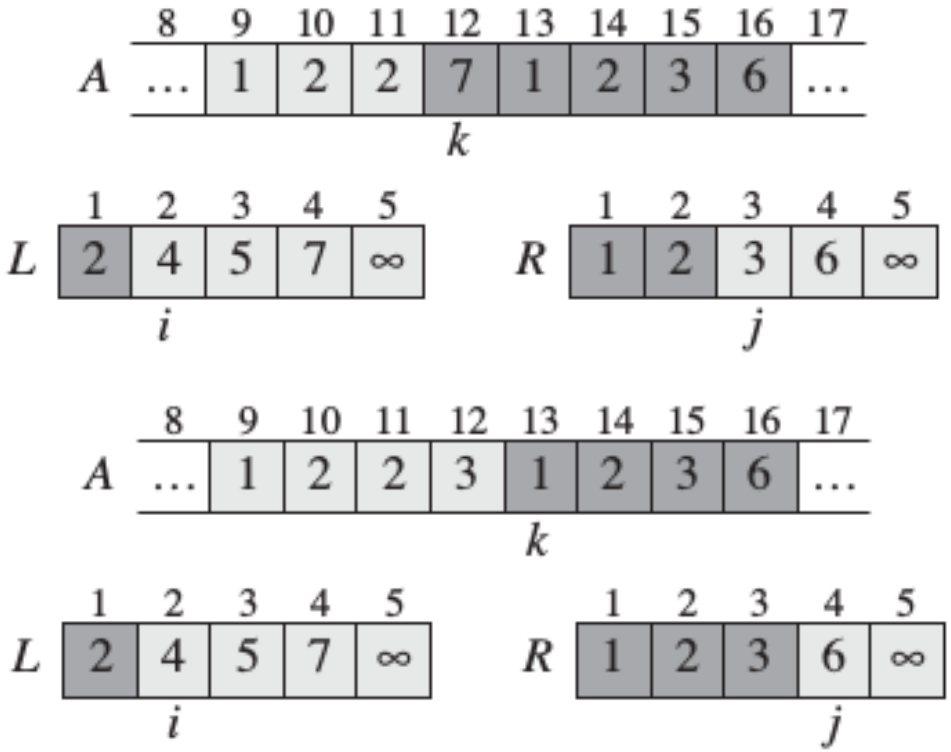
Merge Procedure



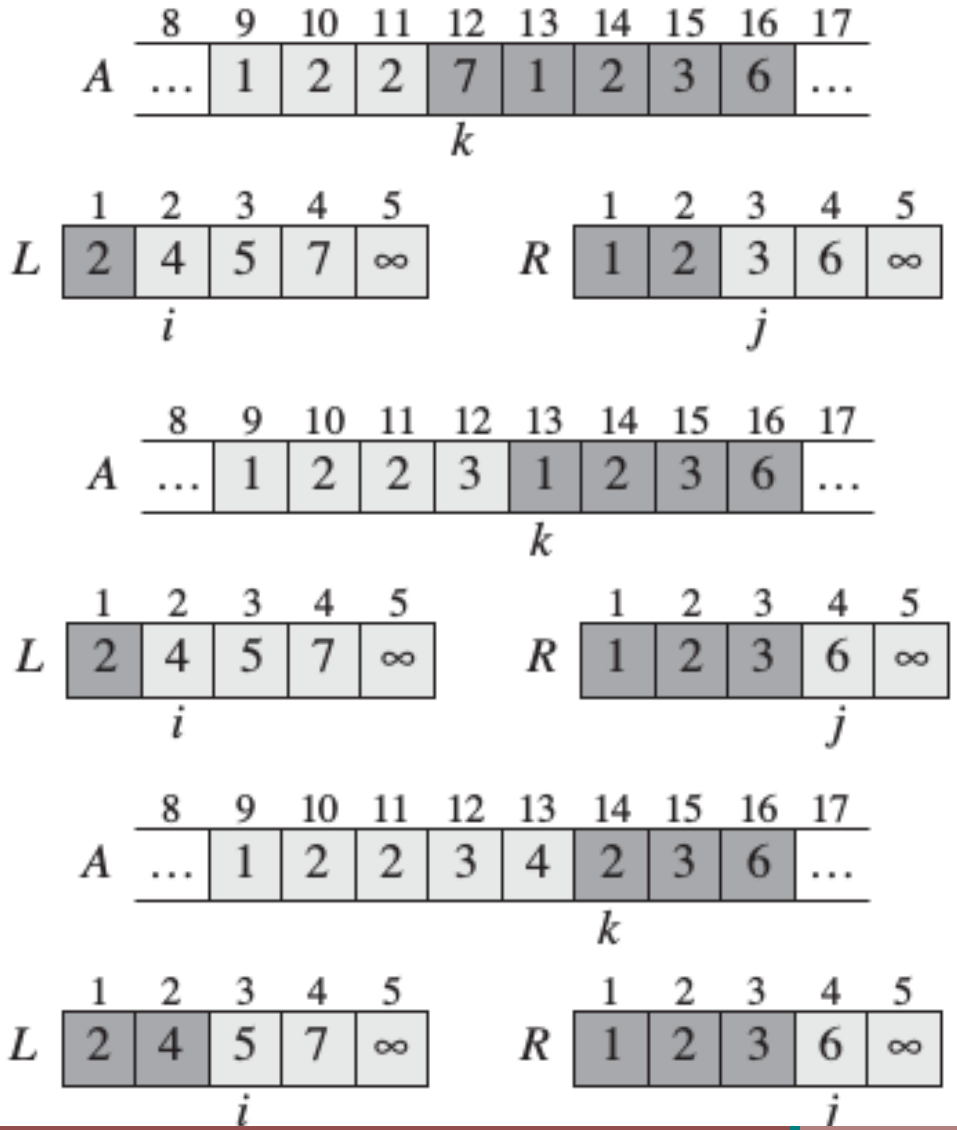
Merge Procedure



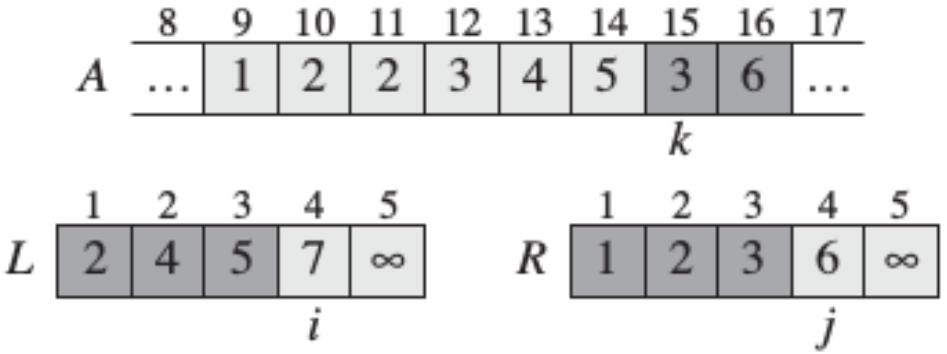
Merge Procedure



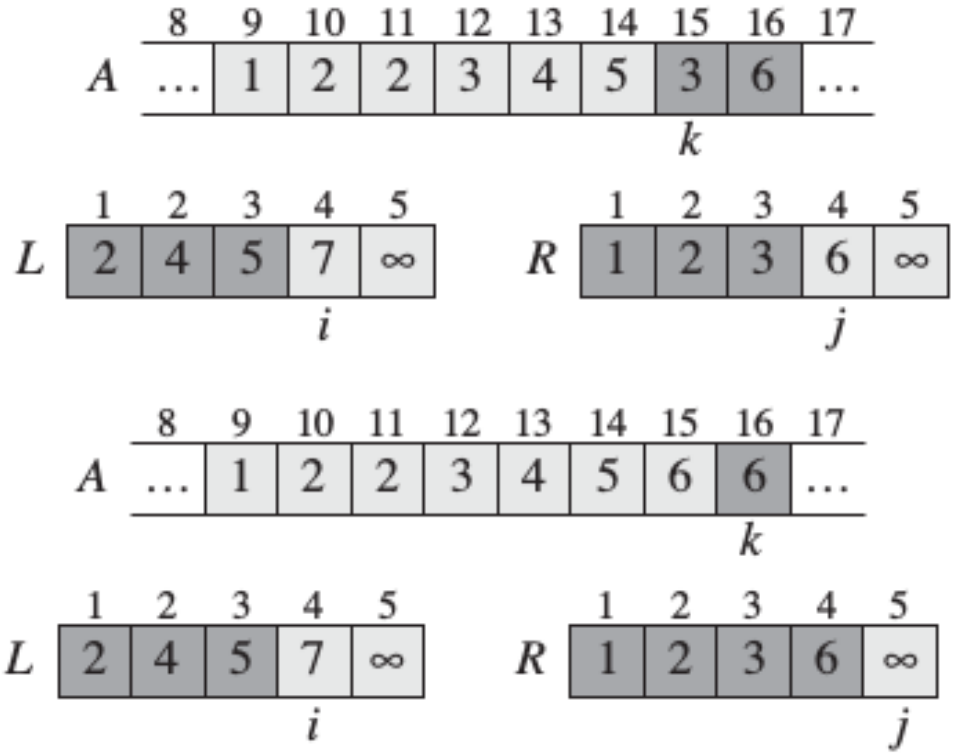
Merge Procedure



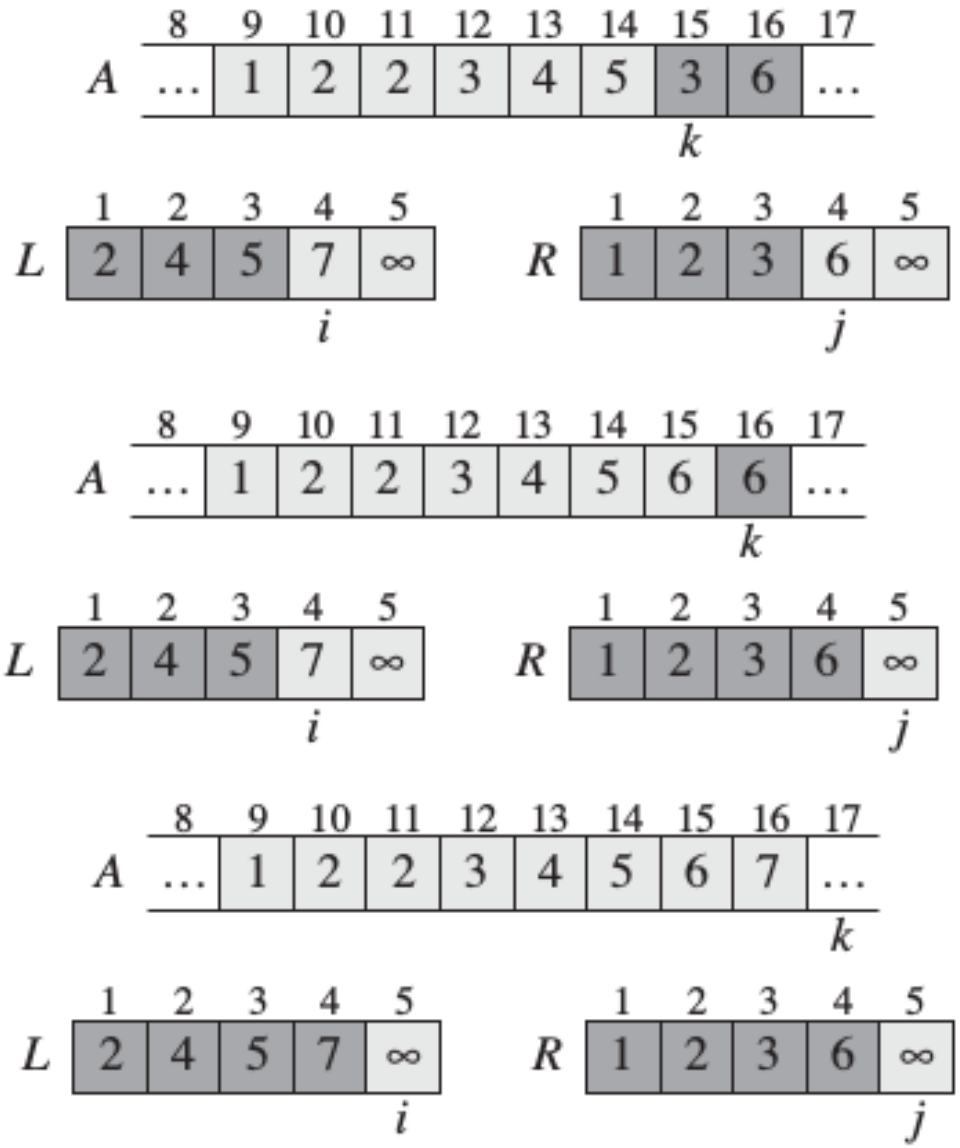
Merge Procedure



Merge Procedure



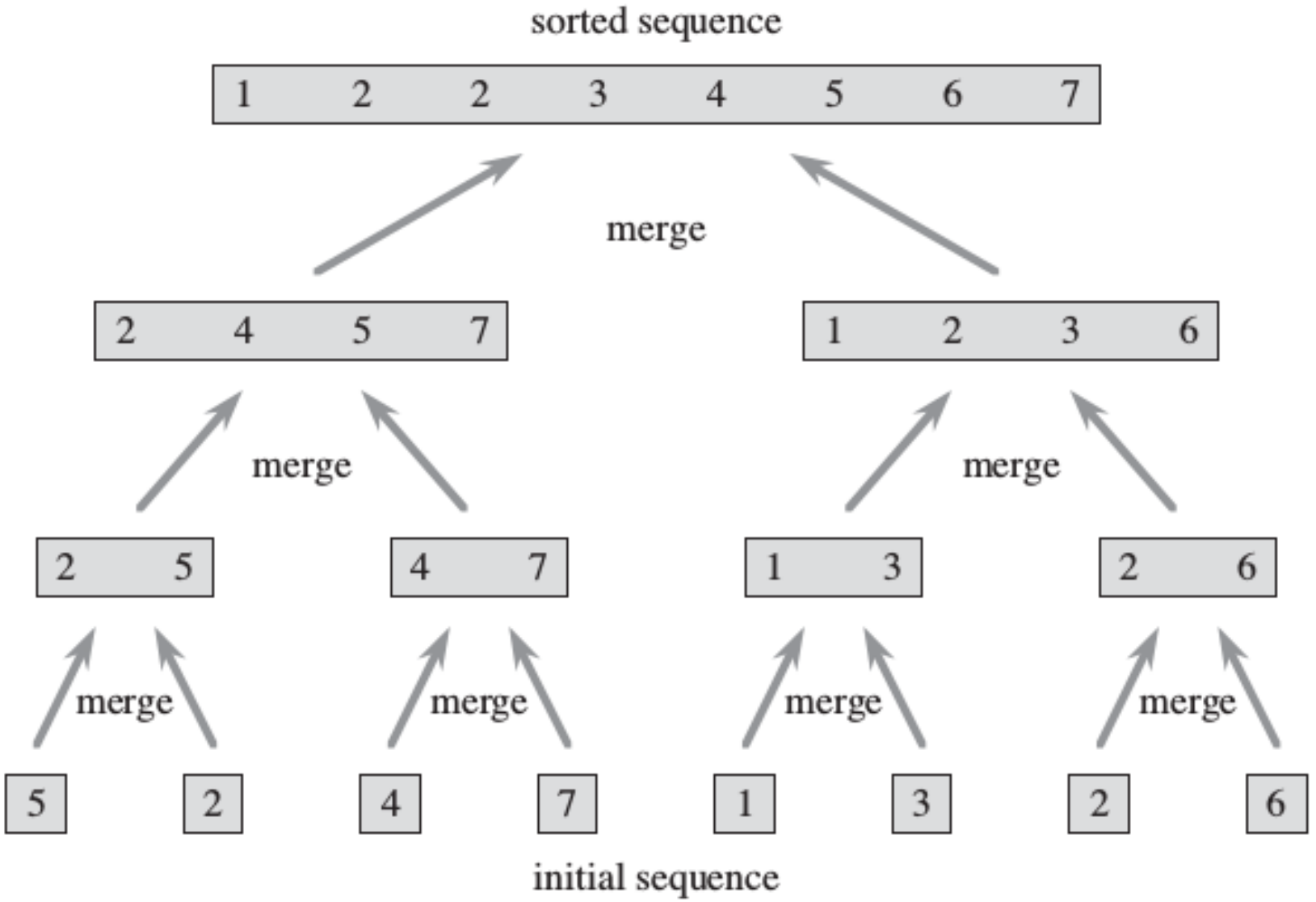
Merge Procedure



Merge Pseudocode

```
MERGE(A, p, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1..n1 + 1] and R[1..n2 + 1 be new arrays
 4  for i = 1 to n1
 5        L[i] = A[p + i − 1]
 6        for j = 1 to n2
 7              R[j] = A[q + j]
 8              L[n1 + 1] = ∞ and R[n2 + 1] = ∞
 9              i = 1 and j = 1
10              for k = p to r
11                    if L[i] ≤ R[j]
12                          A[k] = L[i]
13                          i = i + 1
14                    else A[k] = R[j]
15                          j = j + 1
```

Mergesort Algorithm



Merge Sort Pseudocode

```
MERGE-SORT(A, p, r)
1  if p < r
2        q = ⌊(p + r)/2⌋
3        MERGE-SORT(A, p, q)
4        MERGE-SORT(A, q + 1, r)
5        MERGE(A, p, q, r)
```

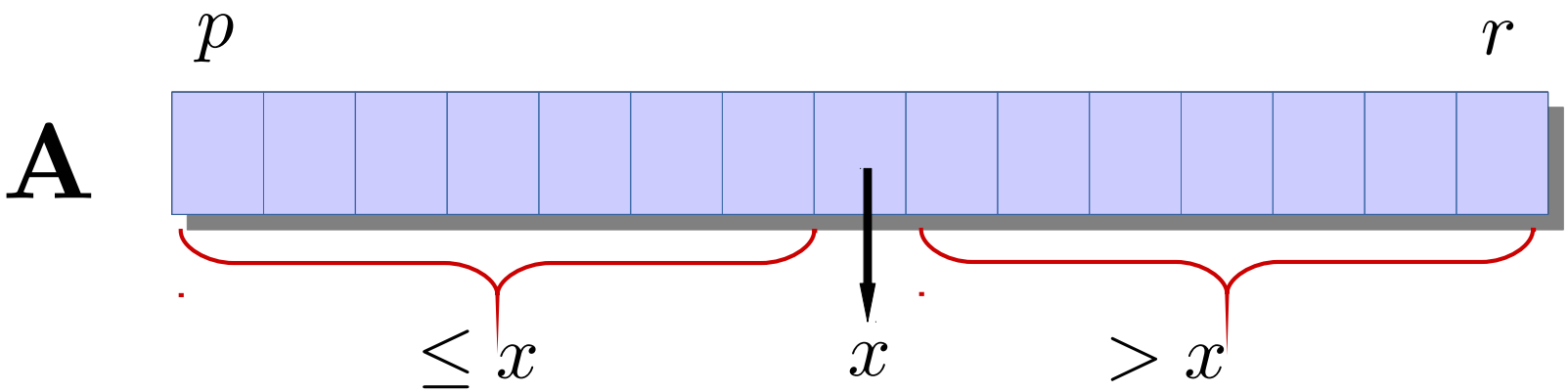
Outline

Quicksort Outline

- Choose one of the elements in the array to act as the pivot.
- Create a temporary array L to hold all elements smaller than the pivot and another array R to hold all those which are larger.
- Ideally the pivot should be chosen so that L and R are as nearly equal in size as possible – i.e., the pivot is the median – but in a simple implementation we can just take the pivot to be the last element in the array).
- Recursively call QuickSort on L and R sub-arrays.
- Append together the (sorted) left hand array L, the pivot, and the (sorted) right hand array R.

Partition Procedure

Question: Given an array $A[p..r]$, partition it into three parts as follows

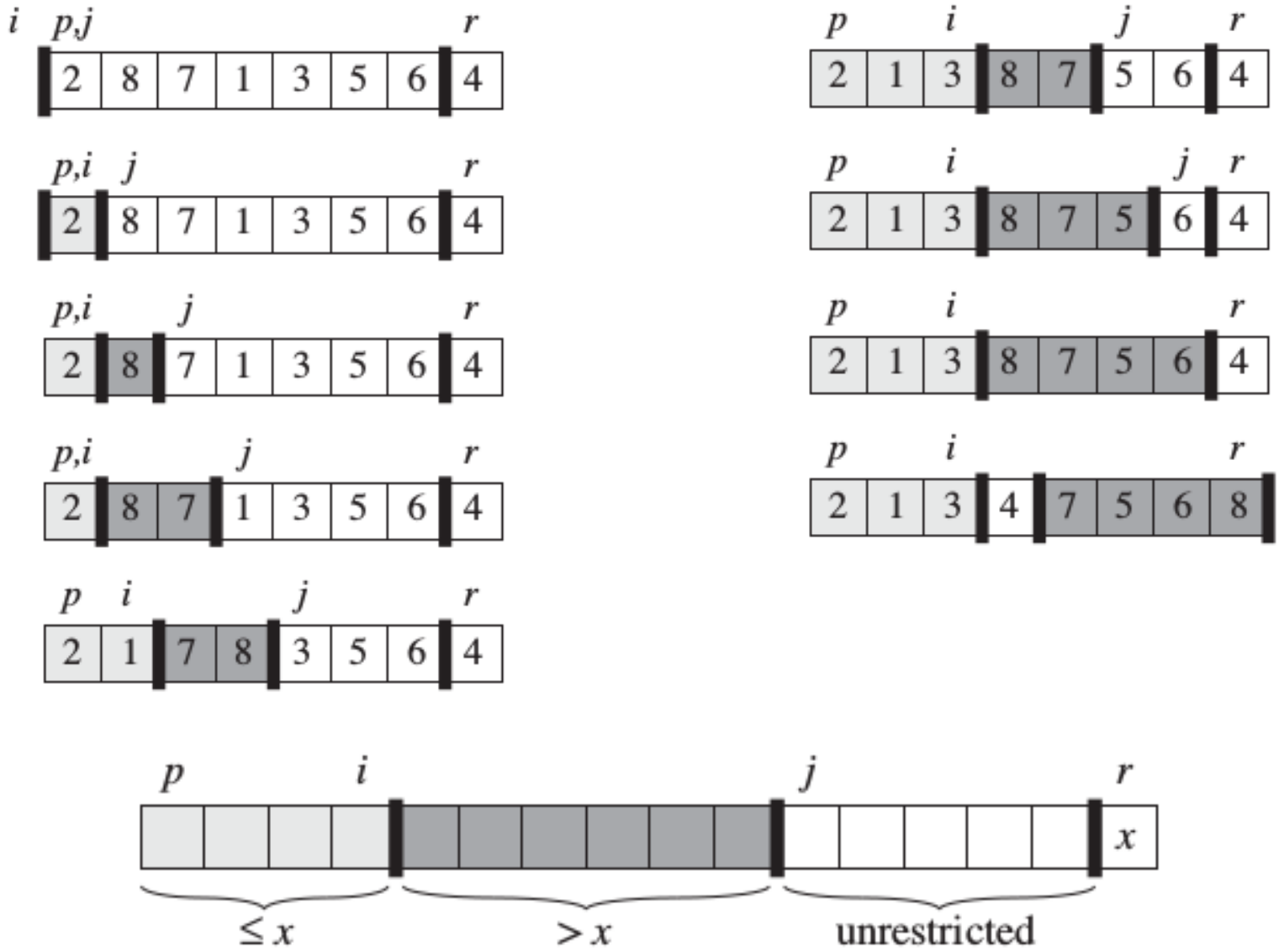


Partition Procedure

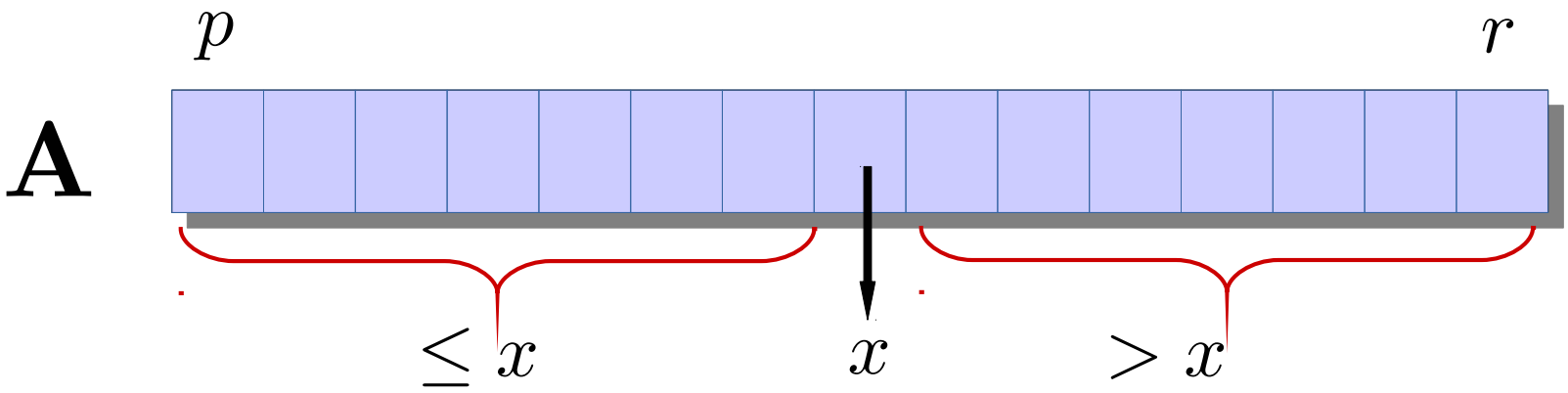
PARTITION(A, p, r)

```
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4        if A[j] ≤ x
5              i = i + 1
6              Exchange A[i] with A[j]
7  Exchange A[i + 1] with A[r]
8  return i + 1
```

Example of Partition Procedure



Quicksort Procedure



QUICKSORT(A, p, r)

```
1  if p < r
2        q = PARTITION(A, p, q)
3        QUICKSORT(A, p, q − 1)
4        QUICKSORT(A, q, r)
```

Outline

Counting Sort

- Each input elements is an integer in the range $[0, k]$, i.e., $n \Leftarrow$ no. of elements and $k \Leftarrow$ highest value element.
- Example: For input set : $4, 1, 3, 4, 3$, $n = 5$ and $k = 4$
- It determines for each input element x, the number of elements less than x. And it uses this information to place element x directly into its position in the output array.
- For example if there exits 17 elements less that x then x is placed into the 18-th position into the output array.
- The algorithm uses three array:
  Input Array: $A[1..n]$ store input data where $A[j] \in \{1, 2, 3, \ldots, k\}$
  Output Array: $B[1..n]$ finally store the sorted data
  Temporary Array: $C[1..k]$ store data temporarily

Counting Sort Pseudocode

Counting Sort

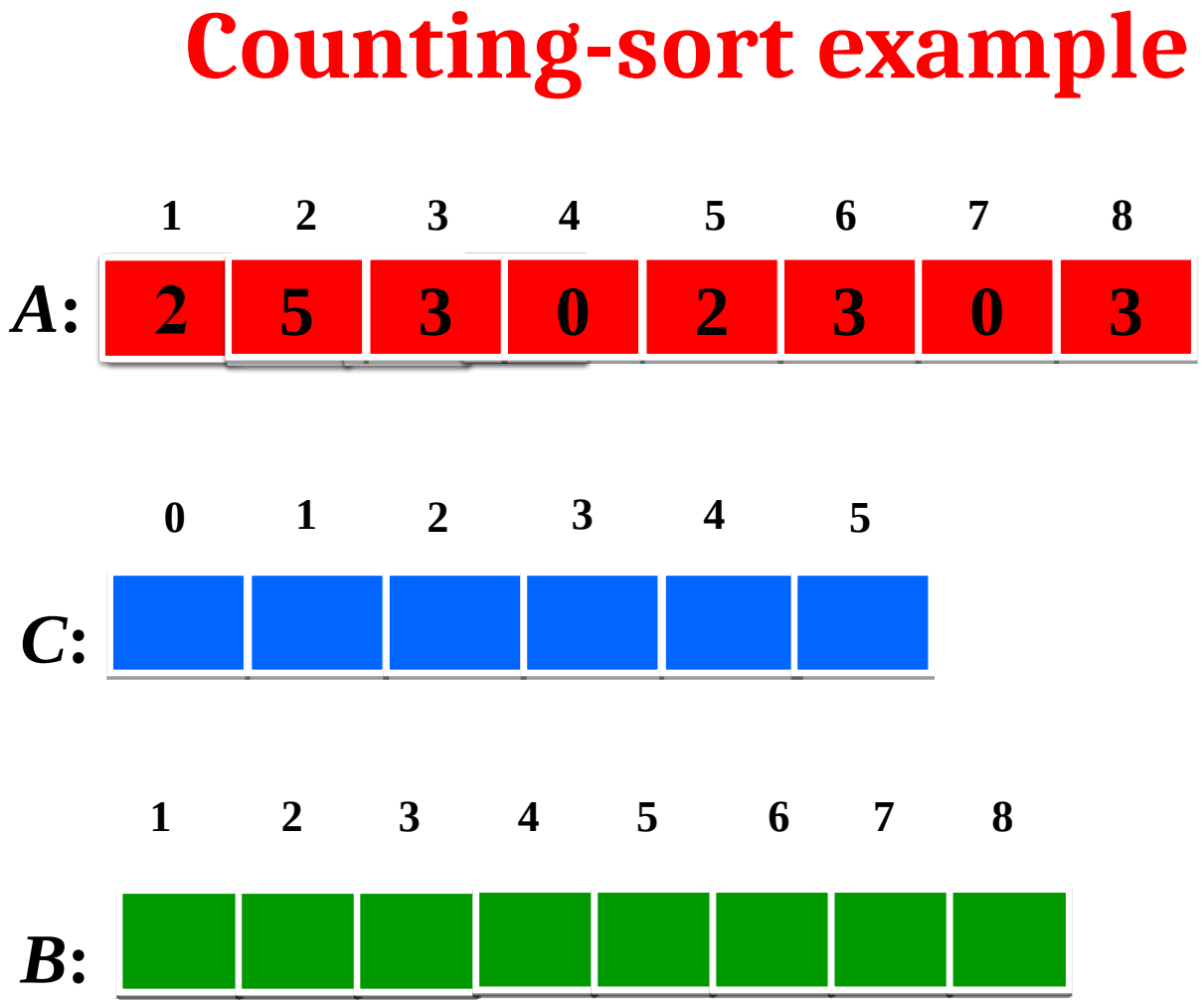
```
1.  Counting-Sort(A, B, k)
2.  Let C[0.....k] be a new array
3.  for i=0 to k
4.        C[i]= 0;
5.  for j=1 to A.length  or n
6.        C[ A[j] ] = C[ A[j] ]  + 1;
7.  for i=1 to k
8.        C[i] = C[i] + C[i-1];
9.  for j=n or A.length down to 1
10.       B[  C[ A[j] ]  ] = A[j];
11.       C[ A[j] ] = C[ A[j] ]  - 1;
```

Counting Sort Pseudocode

Counting Sort

```
1.  Counting-Sort(A, B, k)
2.  Let C[0.....k] be a new array
3.  for i=0 to k                                  [Loop 1]
4.        C[i]= 0;
5.  for j=1 to A.length( or n)                    [Loop 2]
6.        C[ A[j] ] = C[ A[j] ]  + 1;
7.  for i=1 to k                                  [Loop 3]
8.        C[i] = C[i] + C[i-1];
9.  for j=n or A.length down to 1                 [Loop 4]
10.       B[  C[ A[j] ]  ] = A[j];
11.       C[ A[j] ] = C[ A[j] ]  - 1;
```

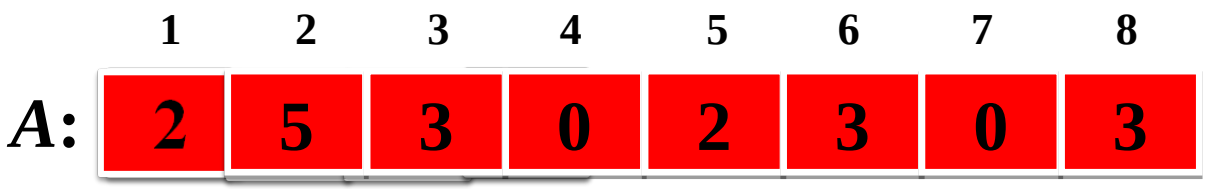
Counting Sort Example

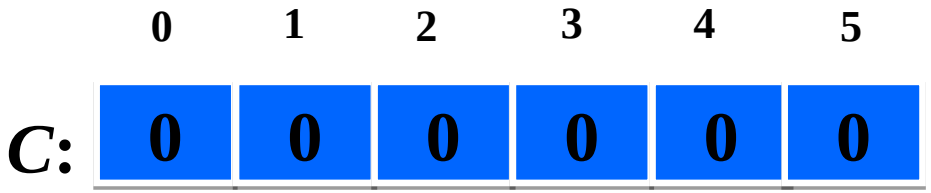


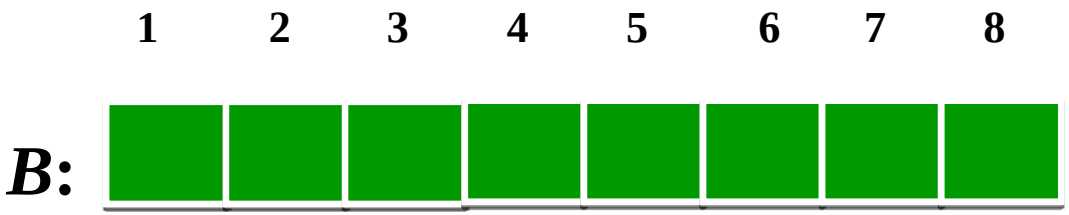
Counting Sort Example

Executing Loop 1

```
3.  for i=0 to k
4.      C[i]= 0;
```



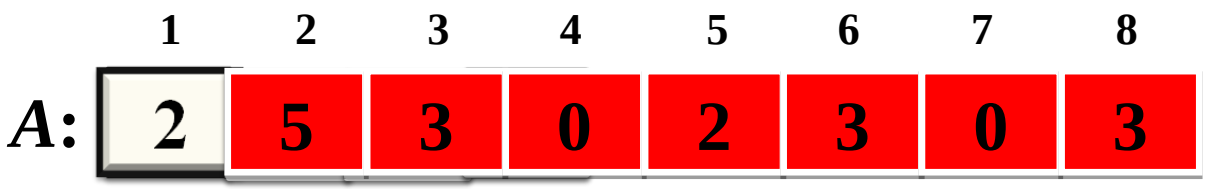


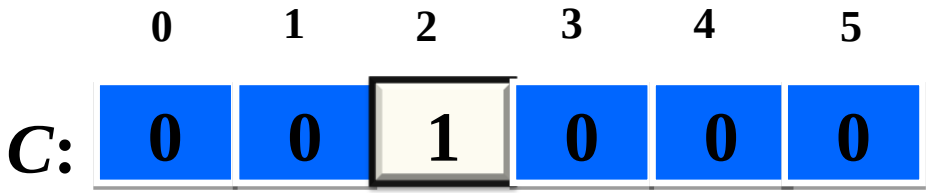


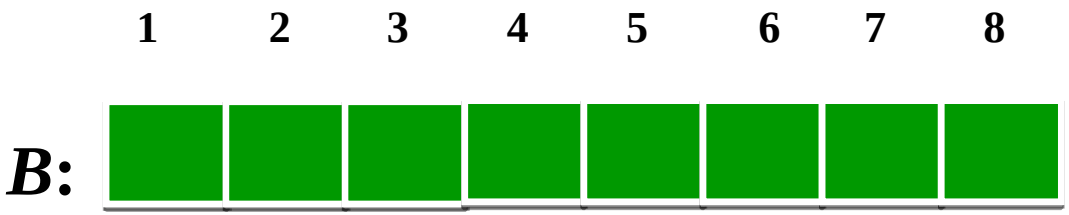
Counting Sort Example

Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```



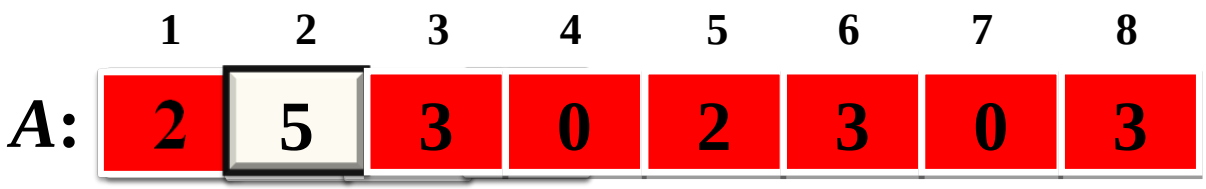


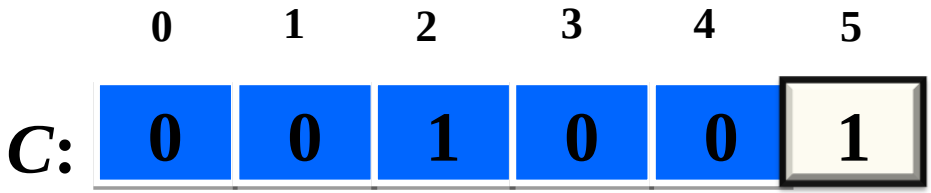


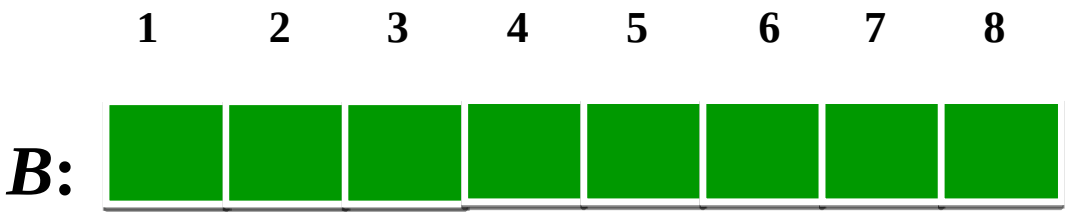
Counting Sort Example

Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```



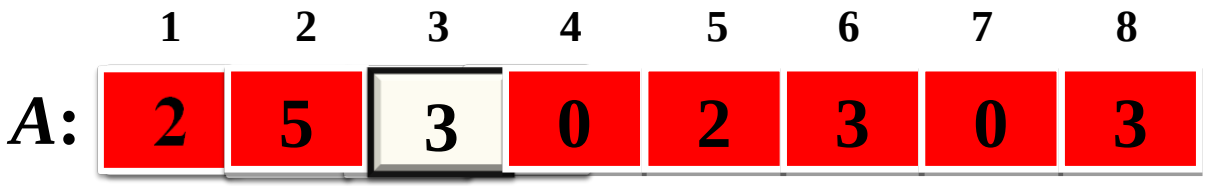


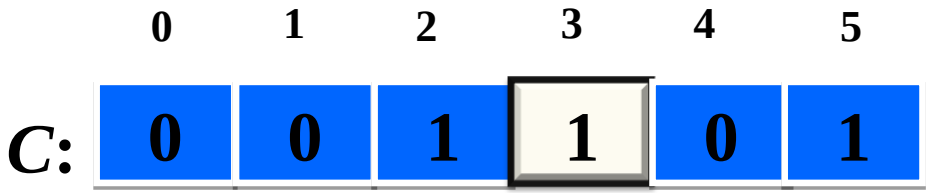


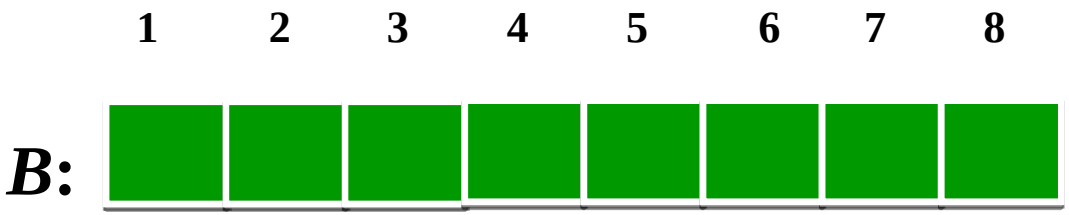
Counting Sort Example

Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```



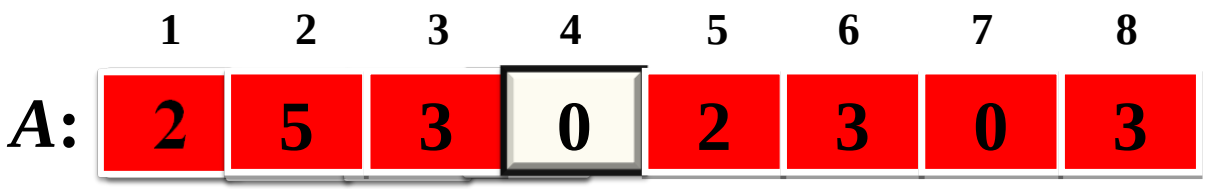


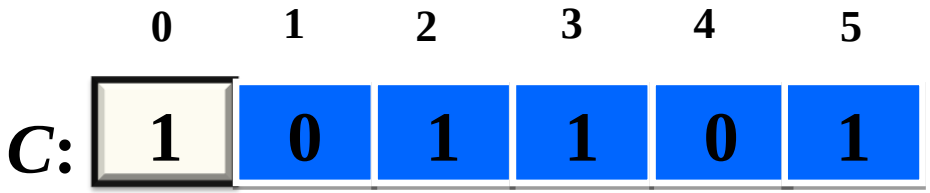


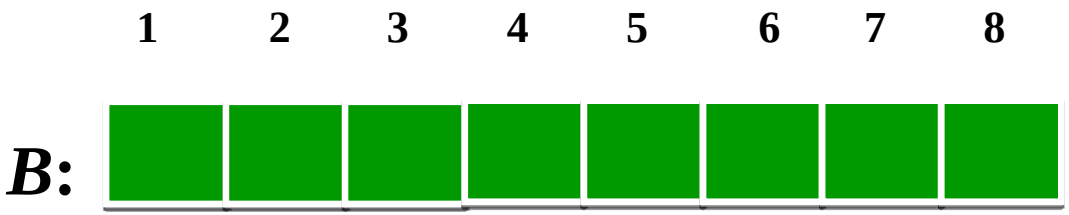
Counting Sort Example

Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```



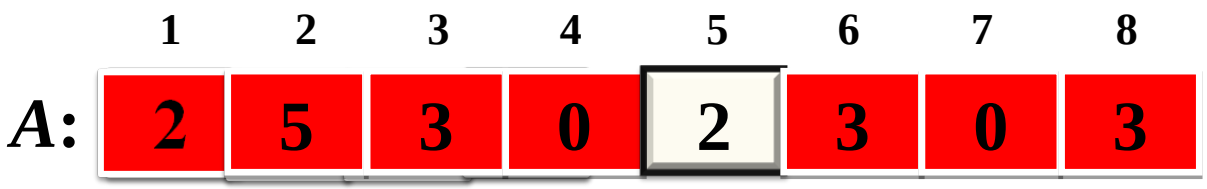


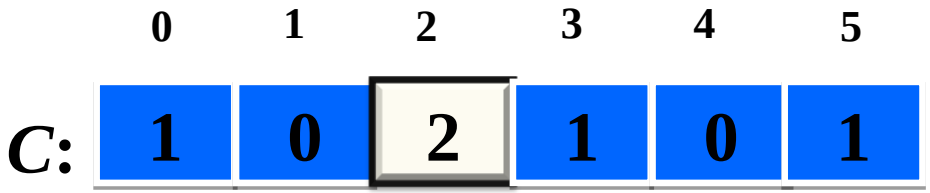


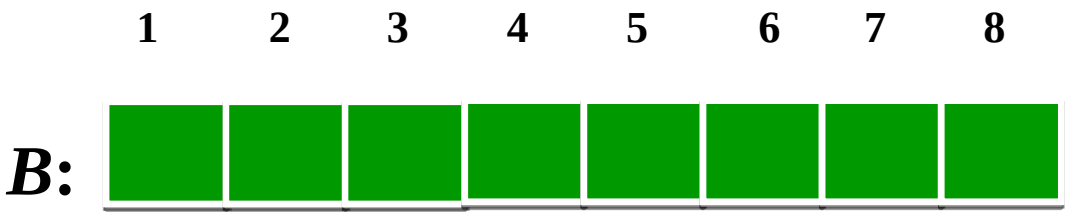
Counting Sort Example

Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```



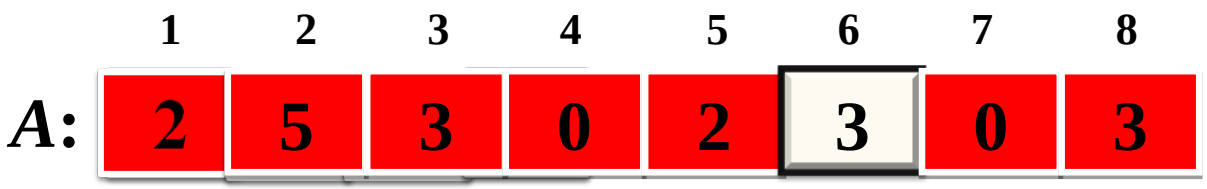


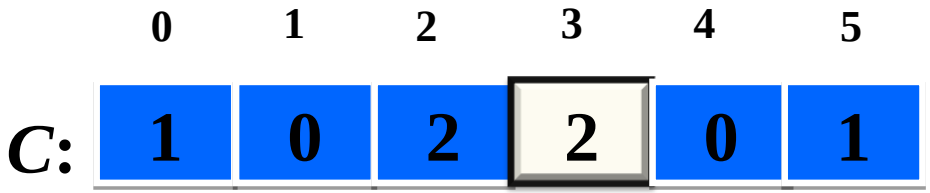


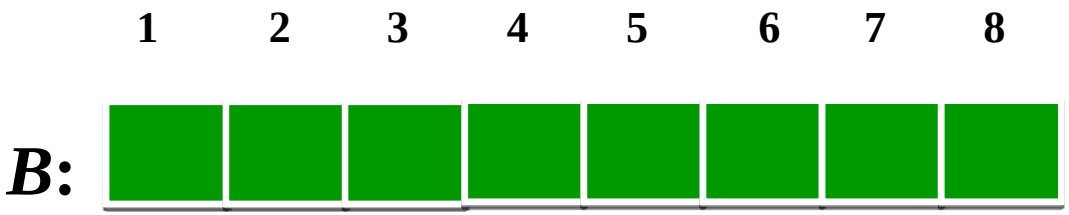
Counting Sort Example

Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```



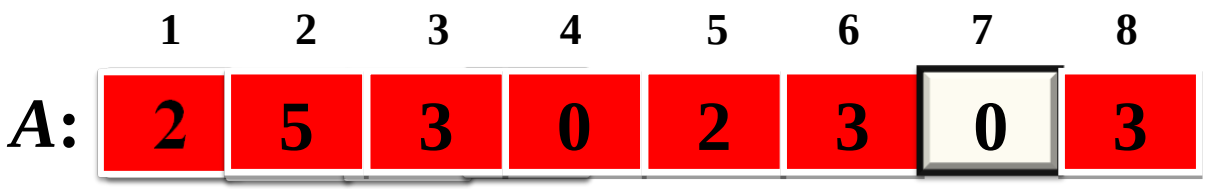


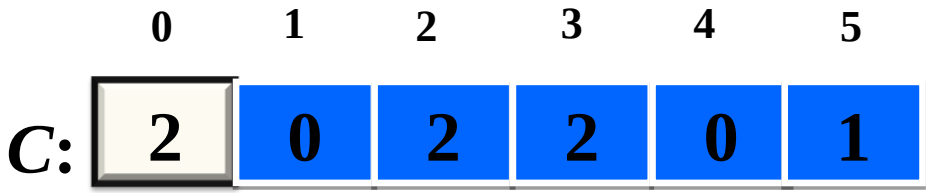


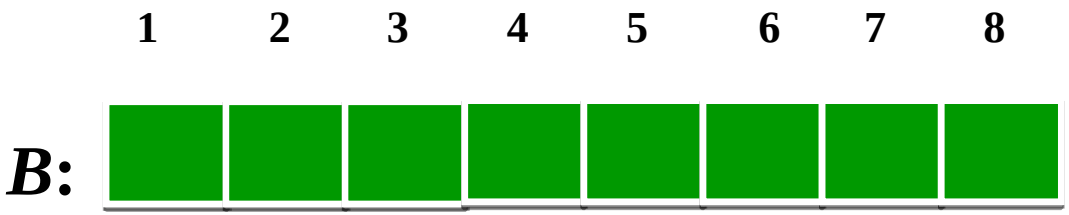
Counting Sort Example

Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```



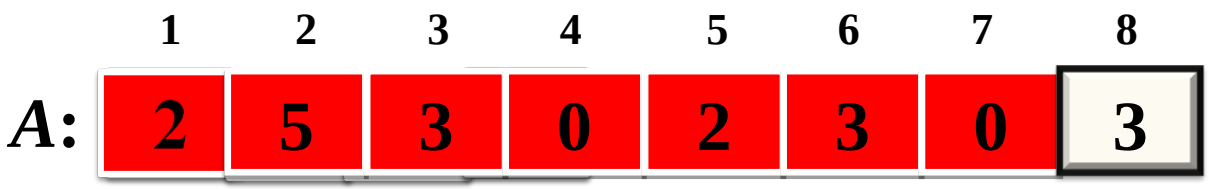


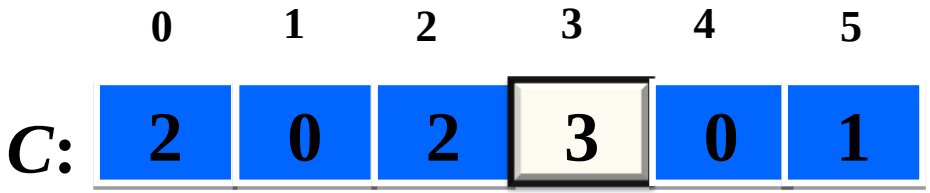


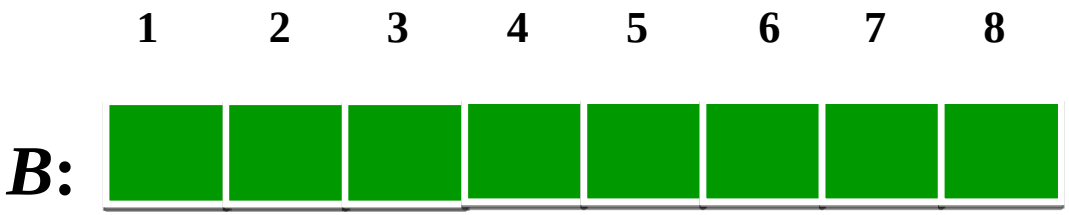
Counting Sort Example

Executing Loop 2

```
5.  for j=1 to A.length or n
6.      C[ A[j] ] = C[ A[j] ] + 1;
```

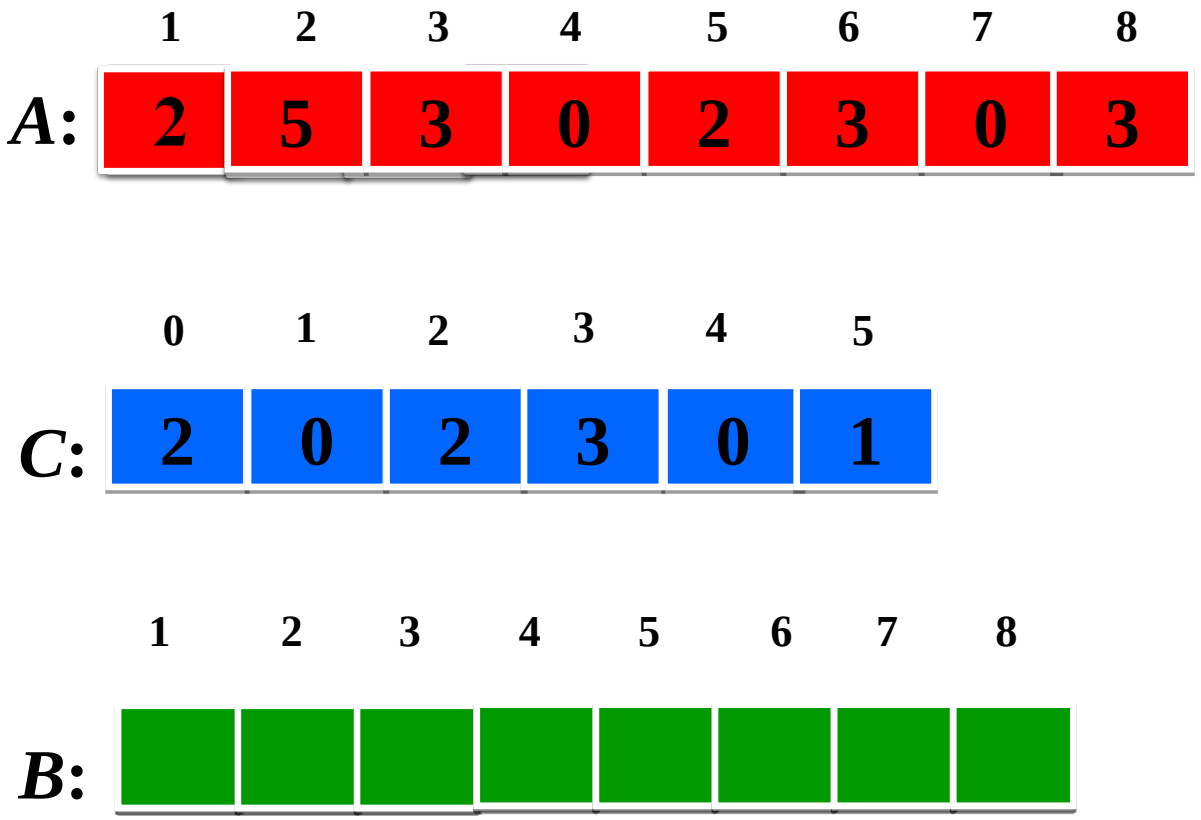






Counting Sort Example

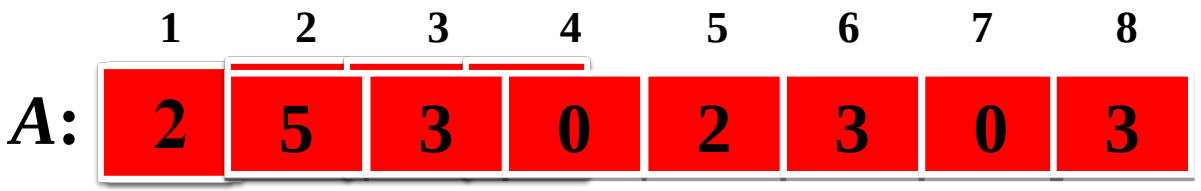
End of Loop 2

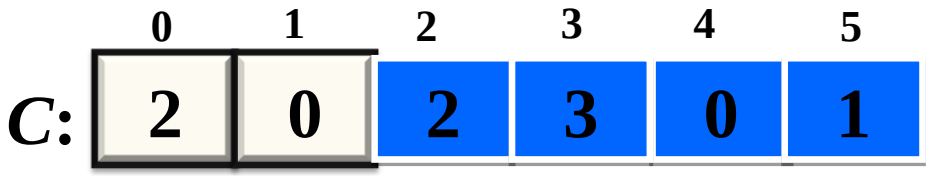


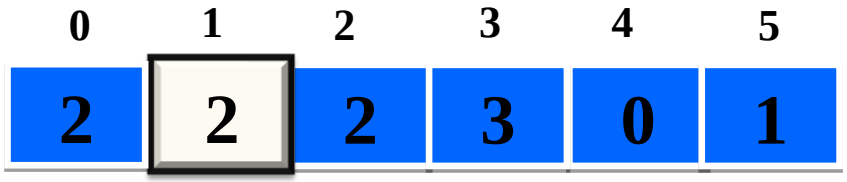
Counting Sort Example

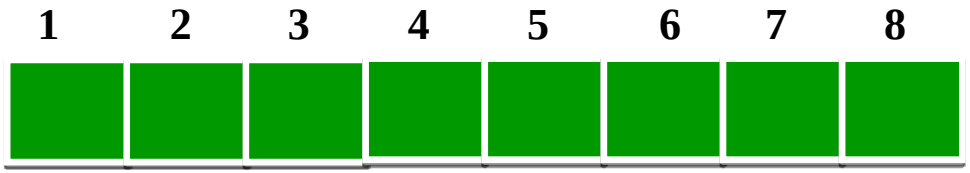
Executing Loop 3

```
7.  for i=1 to k
8.      C[i] = C[i] + C[i-1];
```





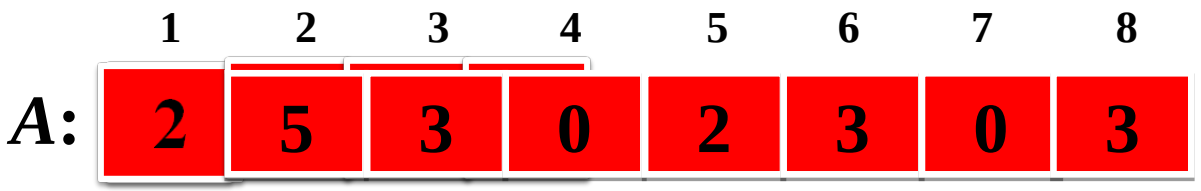


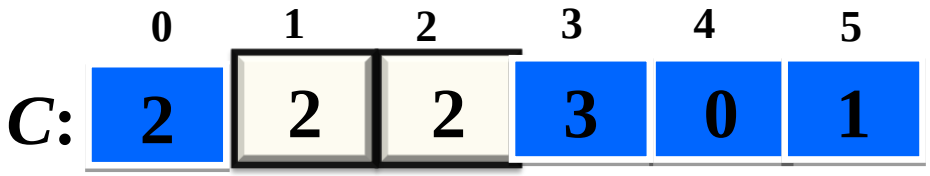


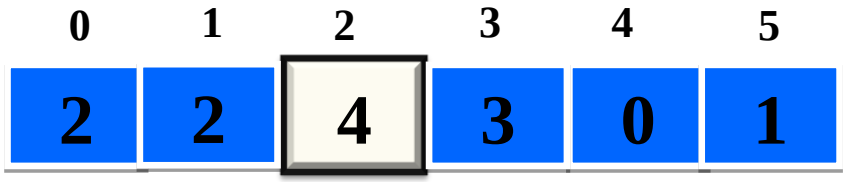
Counting Sort Example

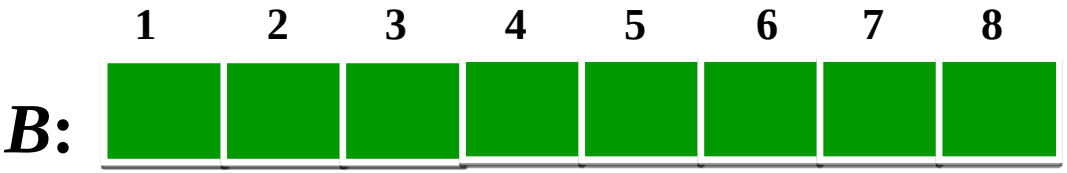
Executing Loop 3

```
7.  for i=1 to k
8.      C[i] = C[i] + C[i-1];
```





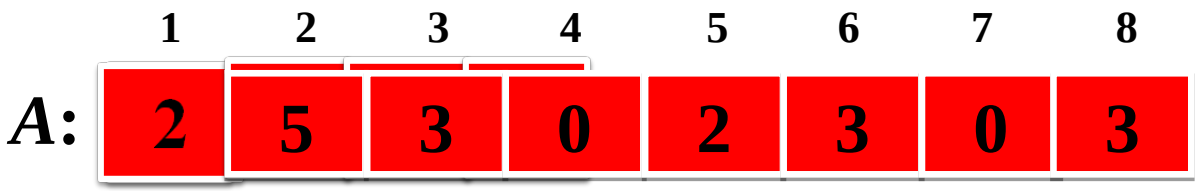


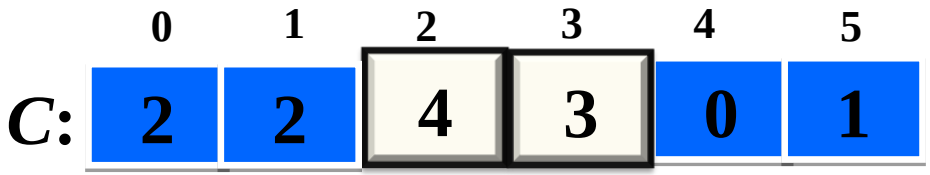


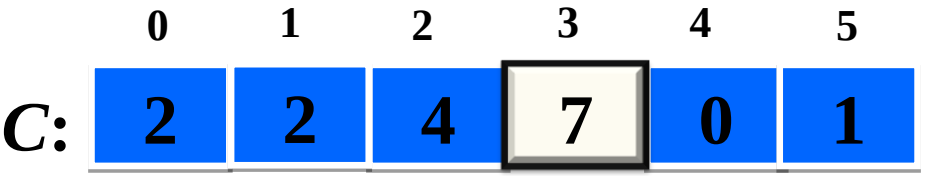
Counting Sort Example

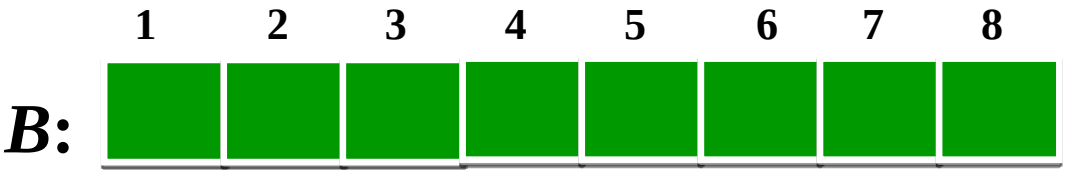
Executing Loop 3

```
7.  for i=1 to k
8.      C[i] = C[i] + C[i-1];
```





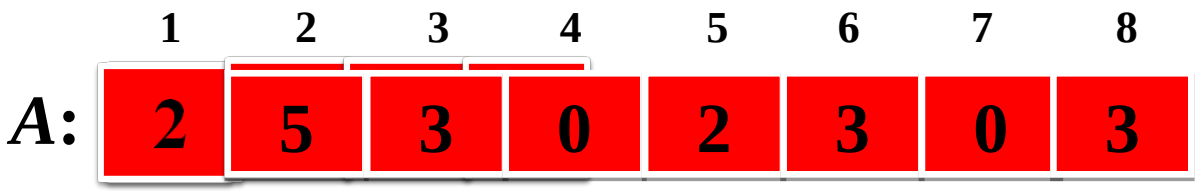


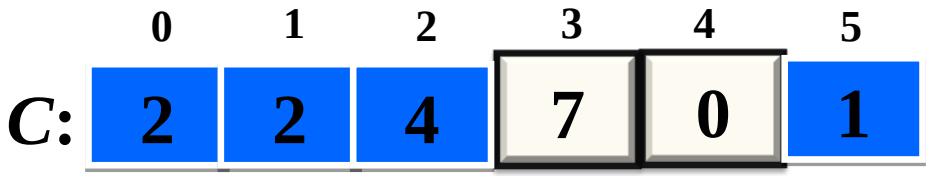


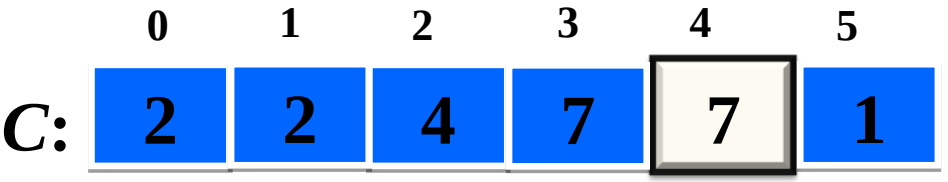
Counting Sort Example

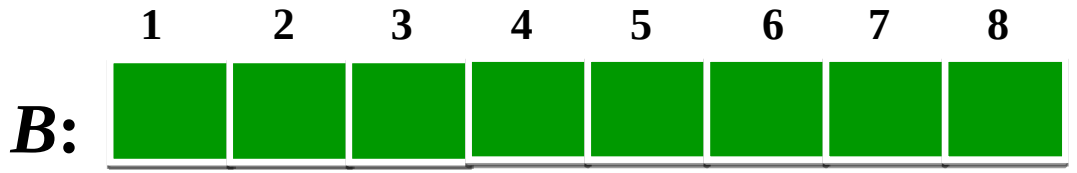
Executing Loop 3

```
7.  for i=1 to k
8.      C[i] = C[i] + C[i-1];
```





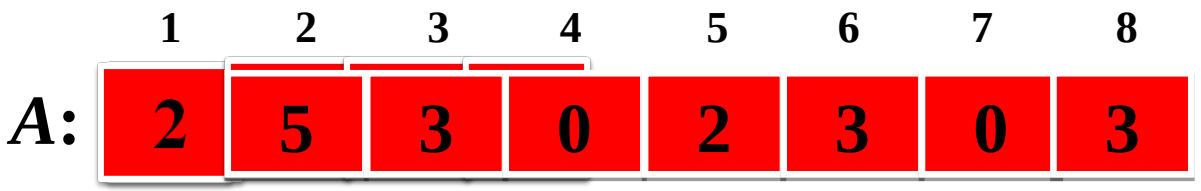


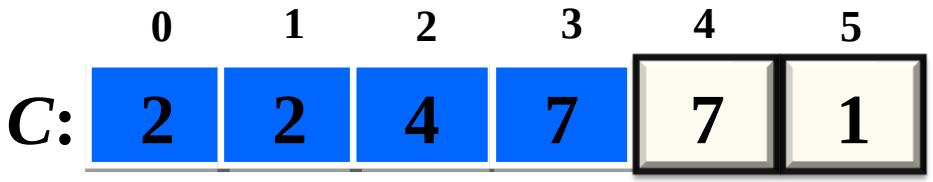


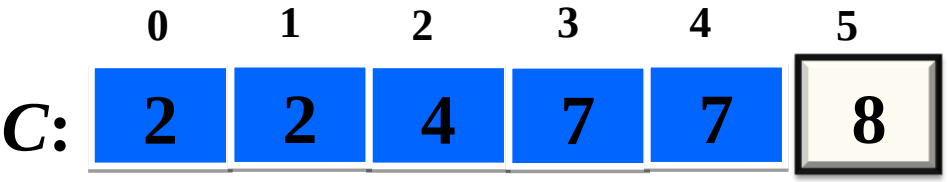
Counting Sort Example

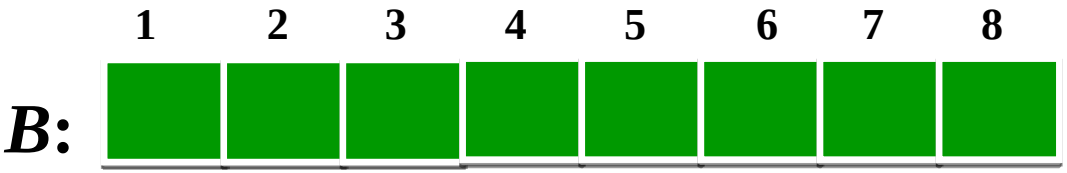
Executing Loop 3

```
7.  for i=1 to k
8.      C[i] = C[i] + C[i-1];
```



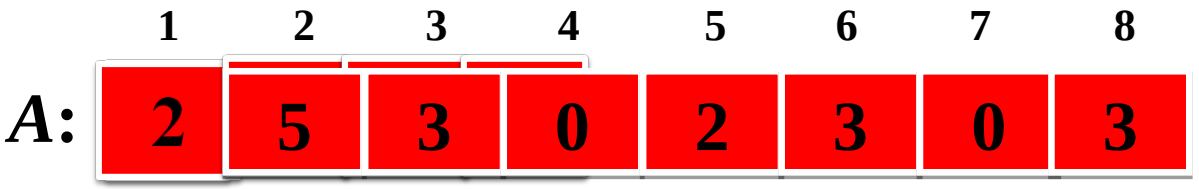


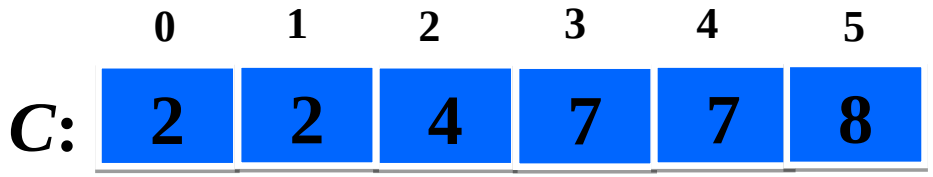


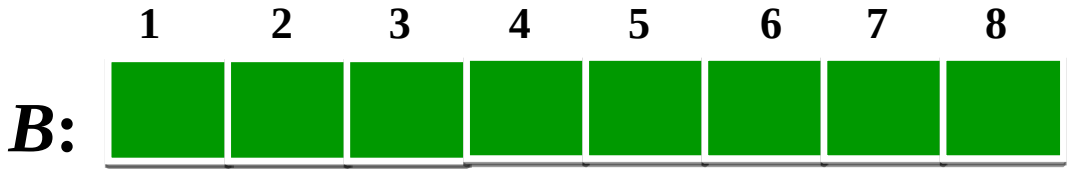


Counting Sort Example

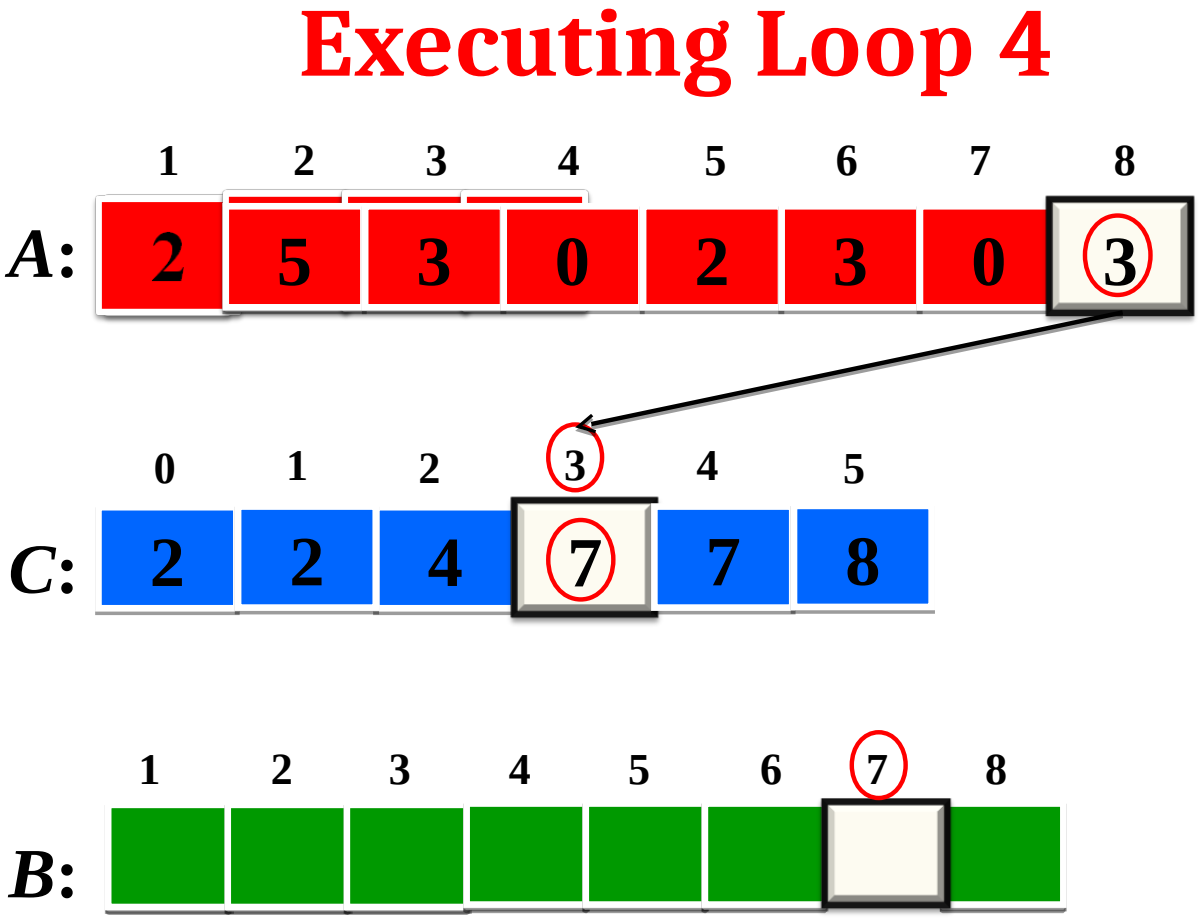
End of Loop 3







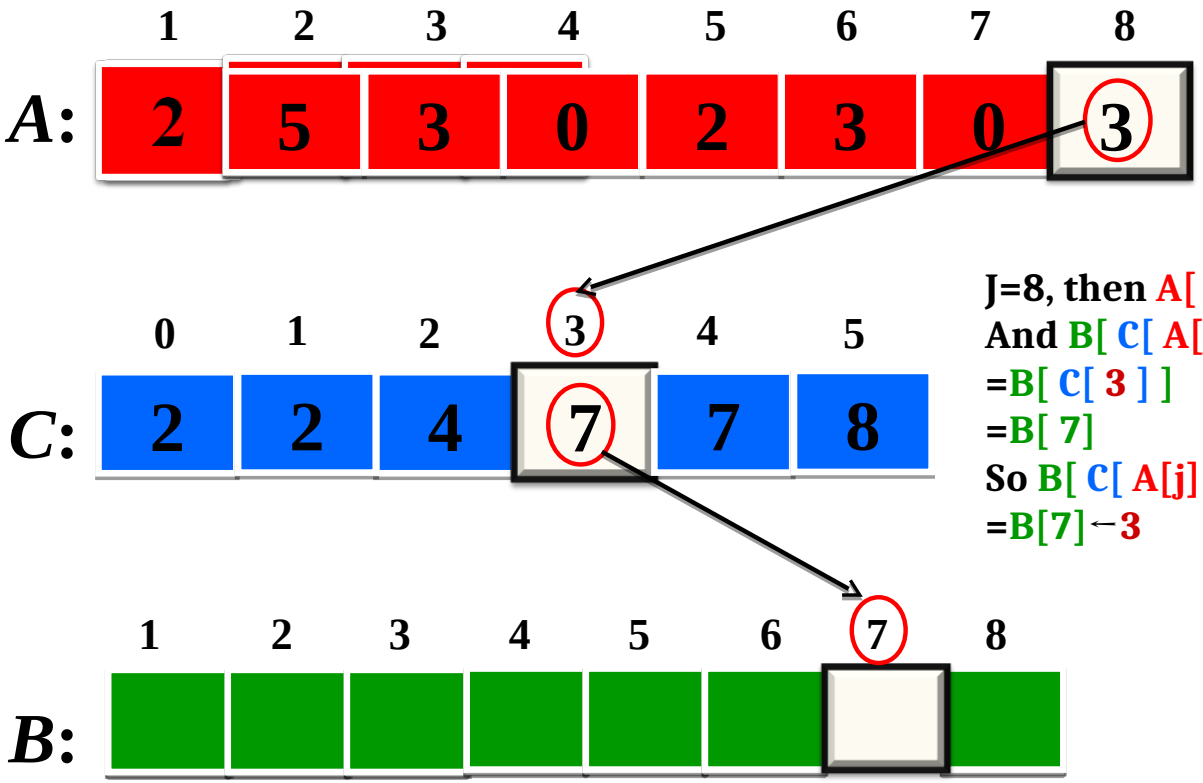
Counting Sort Example



Counting Sort Example

Executing Loop 4

```
9.  for j=n or A.length down to 1
10.     B[ C[ A[j] ] ] = A[j];
11.     C[ A[j] ] = C[ A[j] ] - 1;
```

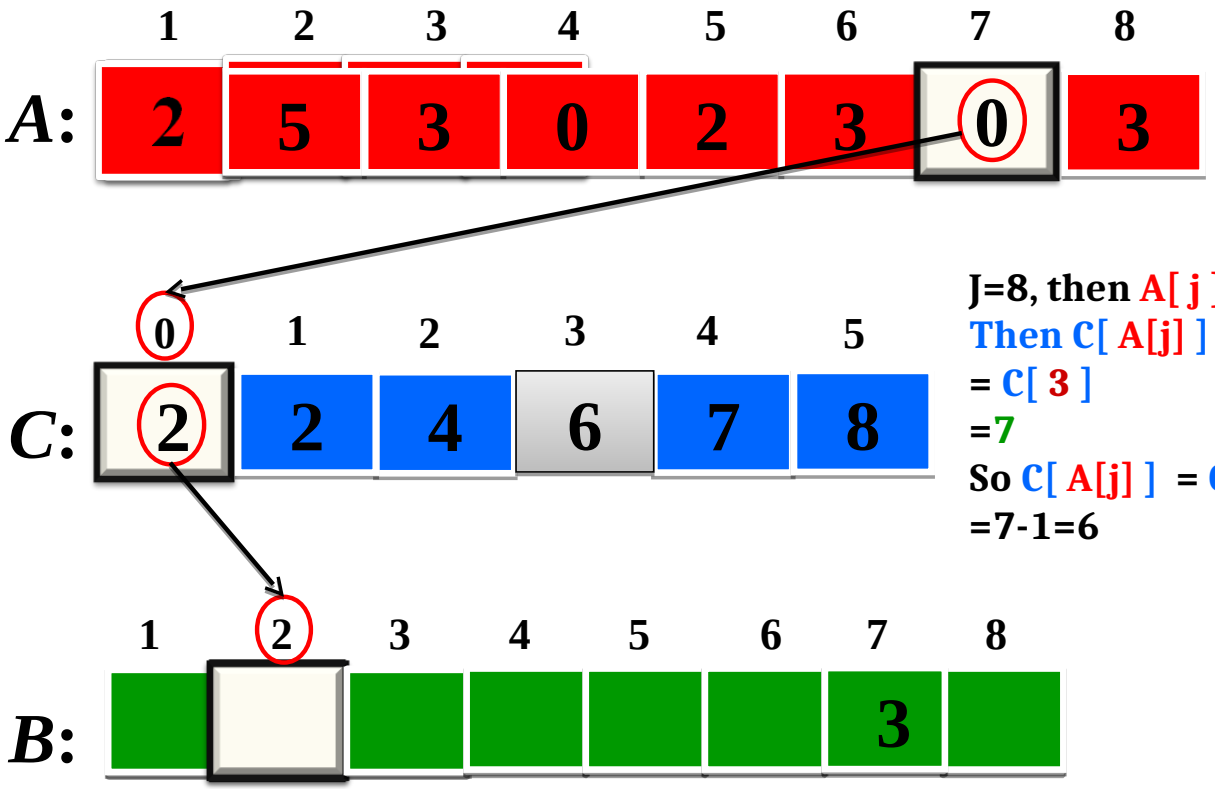


J=8, then A[j]=A[8]=3
And B[C[A[j]]]
=B[C[3]]
=B[7]
So B[C[A[j]]] ←A[j]
=B[7]←3

Counting Sort Example

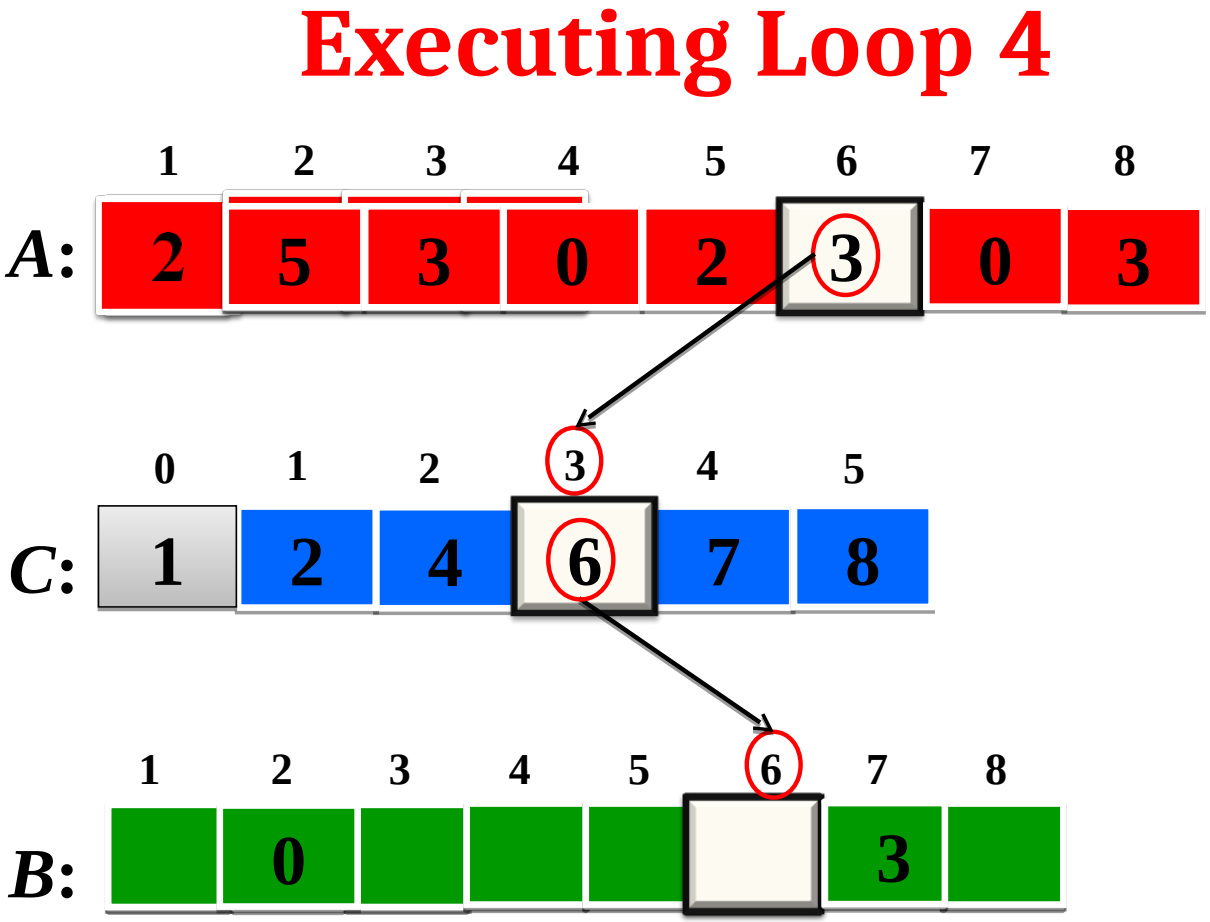
Executing Loop 4

```
9.  for j=n or A.length down to 1
10.     B[ C[ A[j] ] ] = A[j];
11.     C[ A[j] ] = C[ A[j] ] - 1;
```

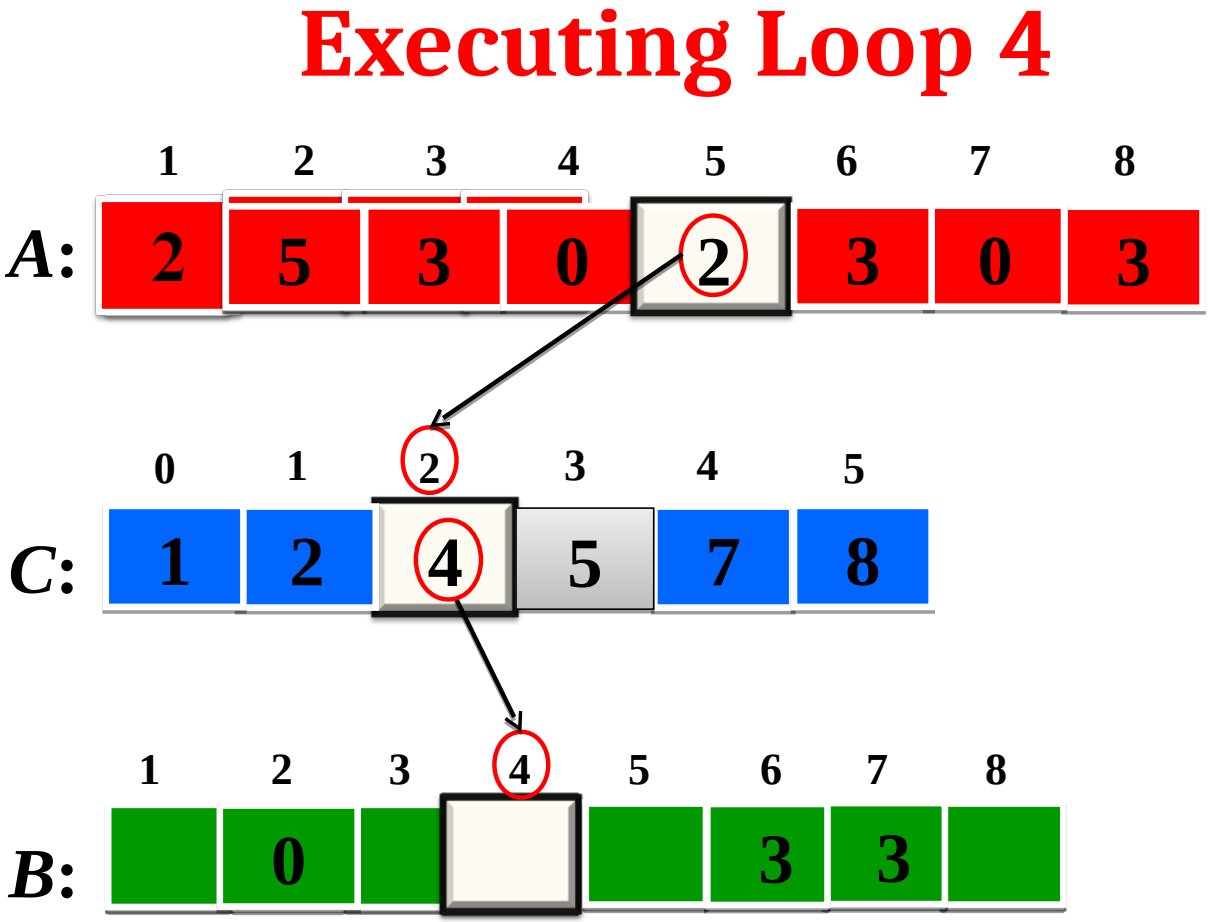


J=8, then A[j]=A[8]=3
Then C[A[j]]
= C[3]
=7
So C[A[j]] = C[A[j]] -1
=7-1=6

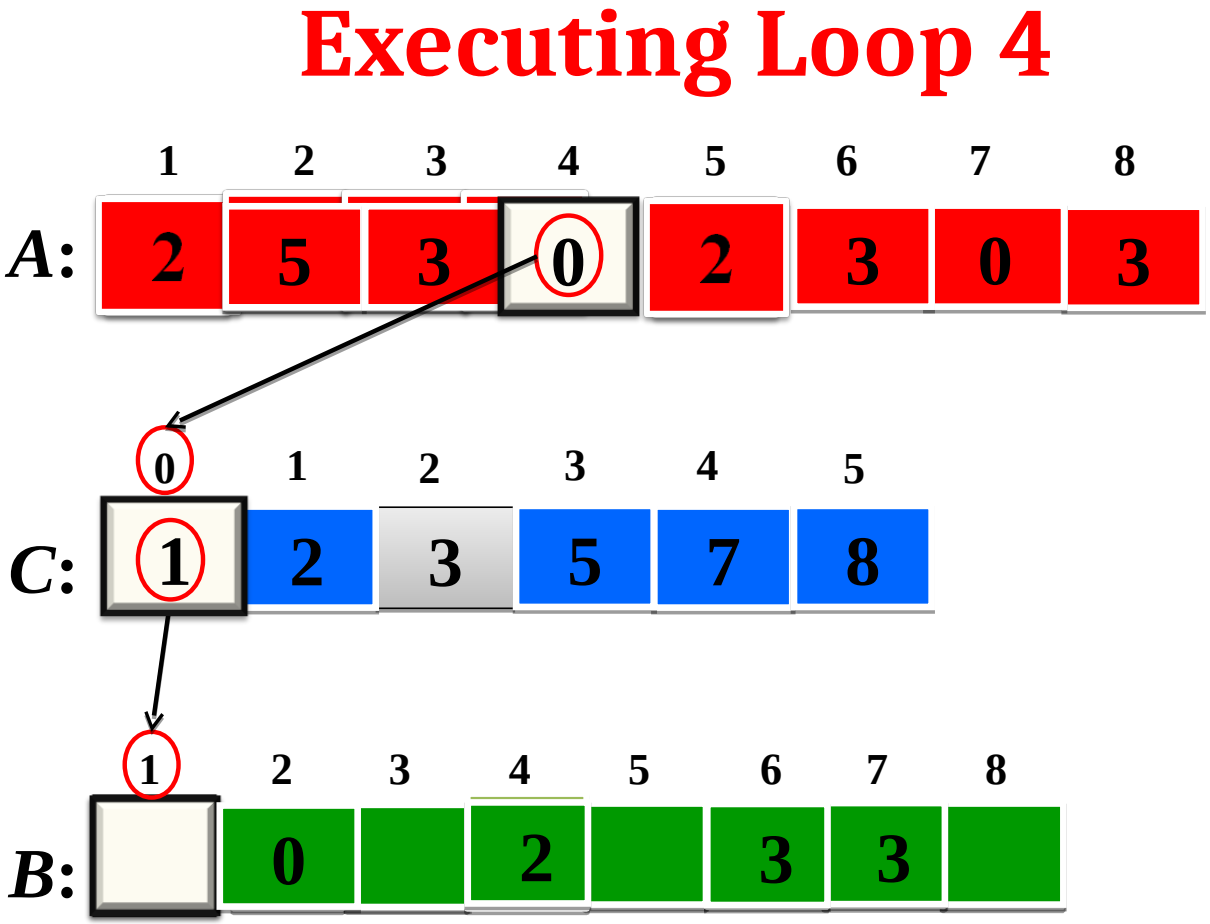
Counting Sort Example



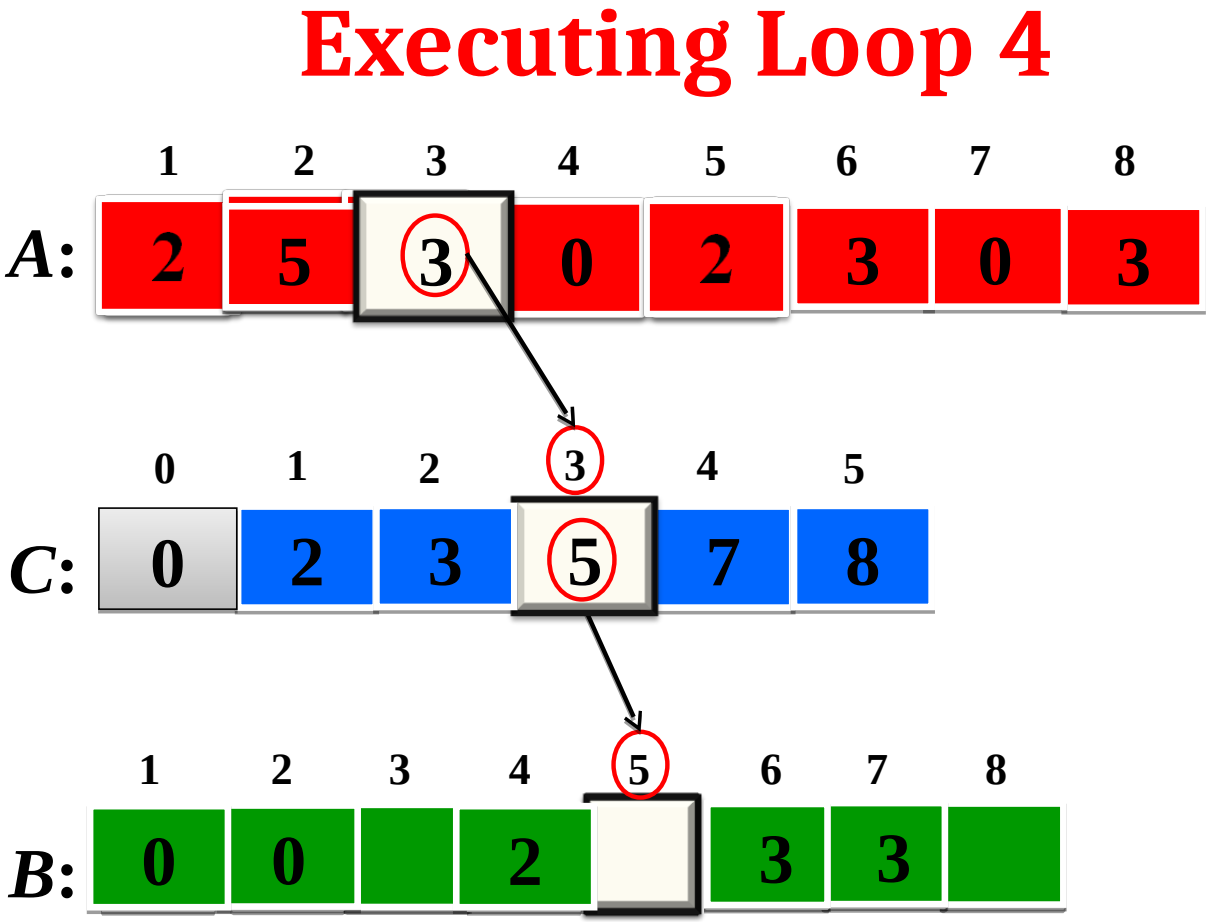
Counting Sort Example



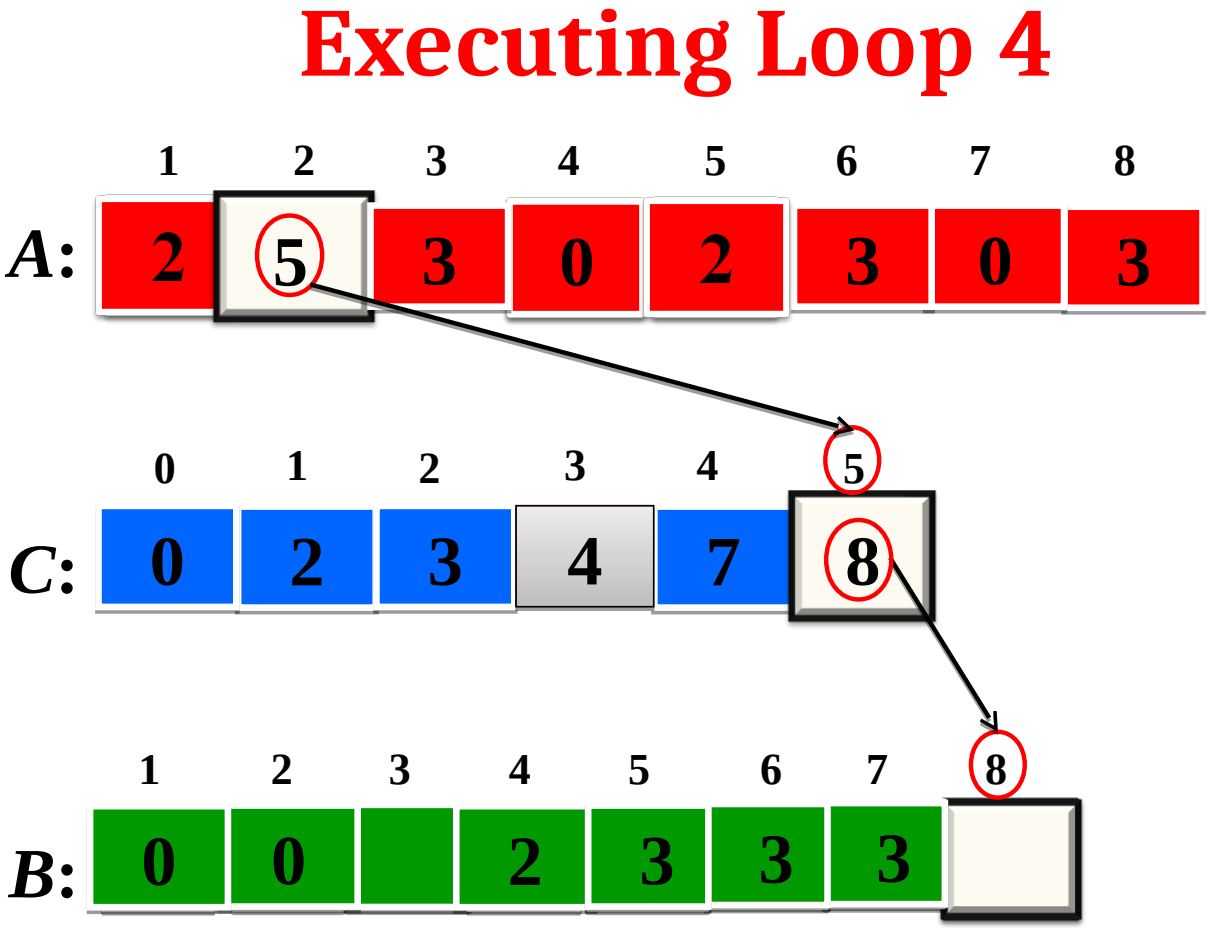
Counting Sort Example



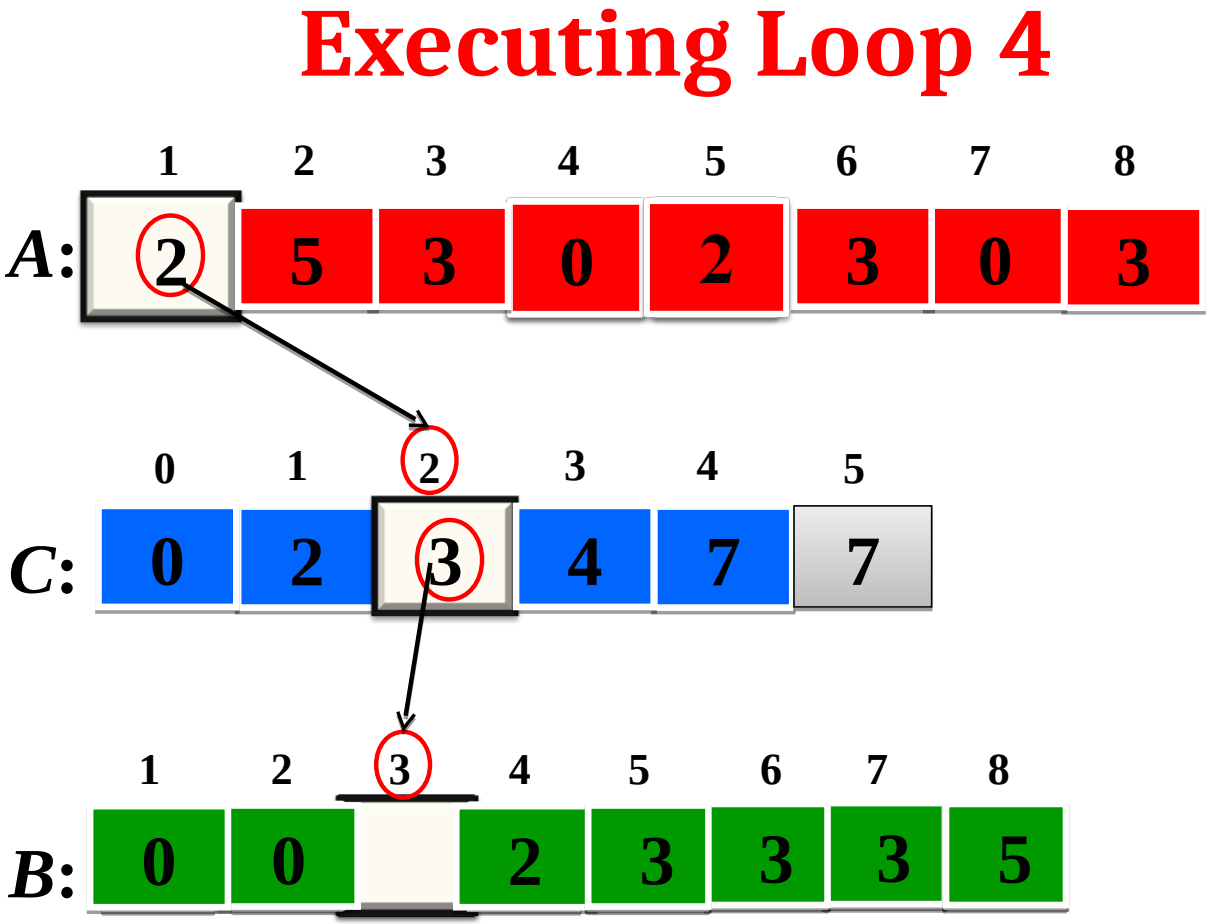
Counting Sort Example



Counting Sort Example

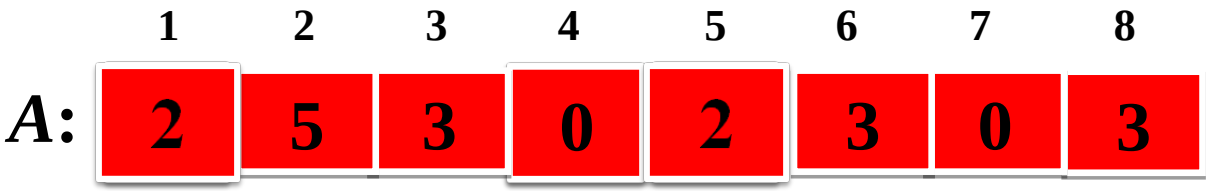


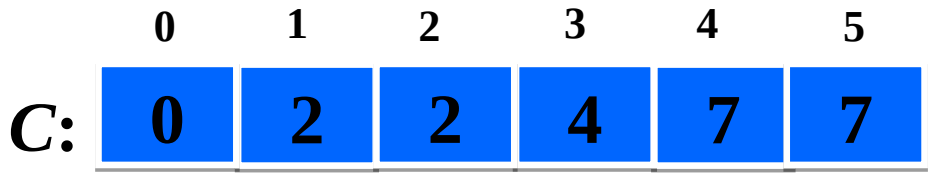
Counting Sort Example

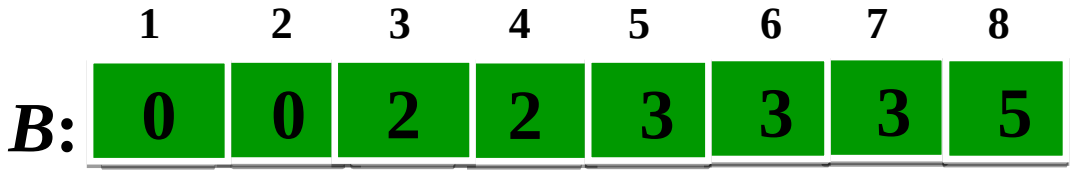


Counting Sort Example

End of Loop 4







Sorted data in Array B

Thank You !!!

Problem I

Calculate the number of comparisons and swaps involve in each of the sorting techniques for a given array of n numbers? (Selection Sort, Bubble Sort, Mergesort, Quicksort)

Lower bound of sorting

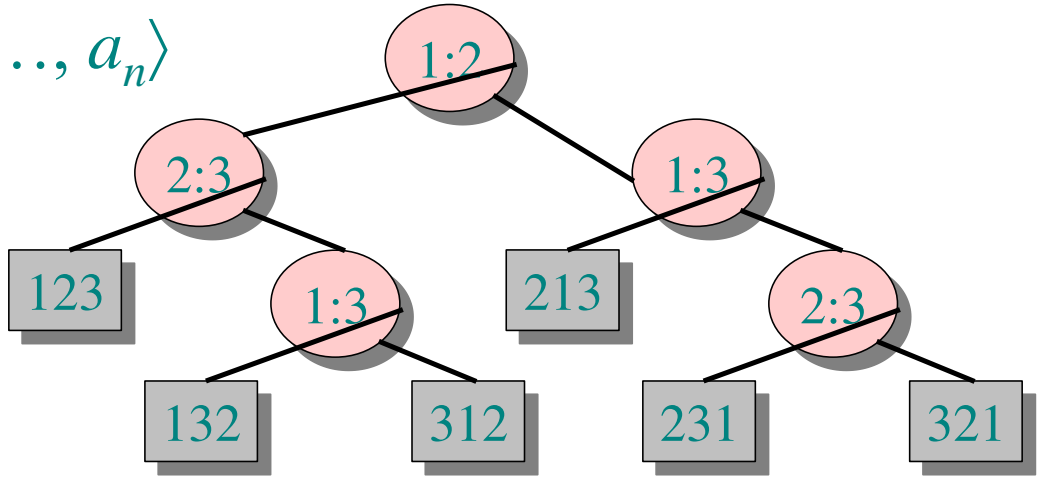
Prove a Lower Bound for
any comparison based algorithm
for the Sorting Problem

How?
Decision trees help us.

Lower bound of sorting

Decision-tree example

Sort $\langle a_1, a_2, \ldots, a_n \rangle$



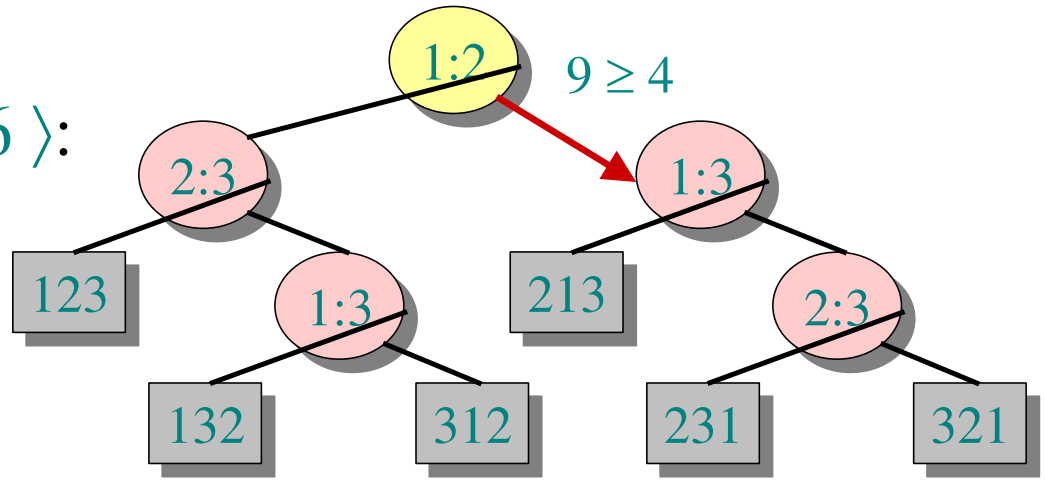
Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

Lower bound of sorting

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle = \langle 9, 4, 6 \rangle$:



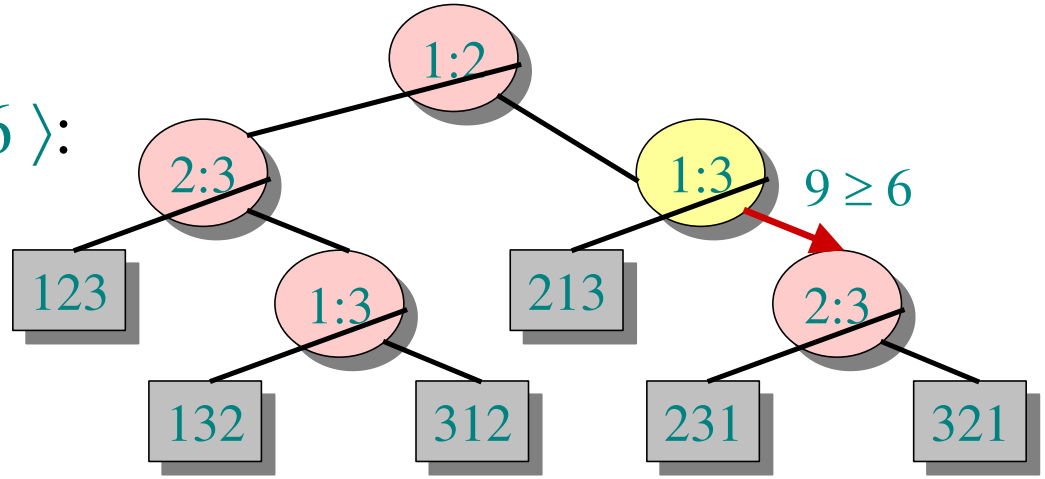
Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

Lower bound of sorting

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle = \langle 9, 4, 6 \rangle$:



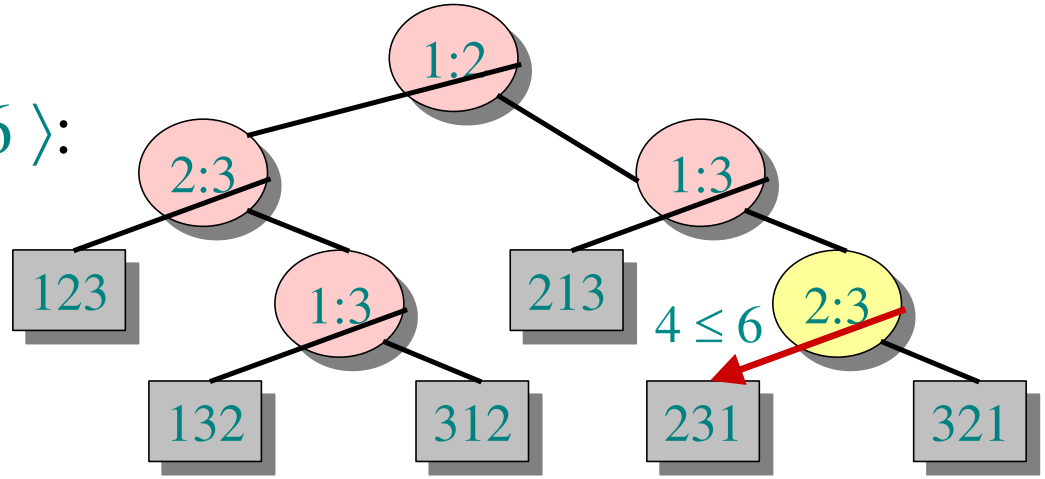
Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

Lower bound of sorting

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle = \langle 9, 4, 6 \rangle$:



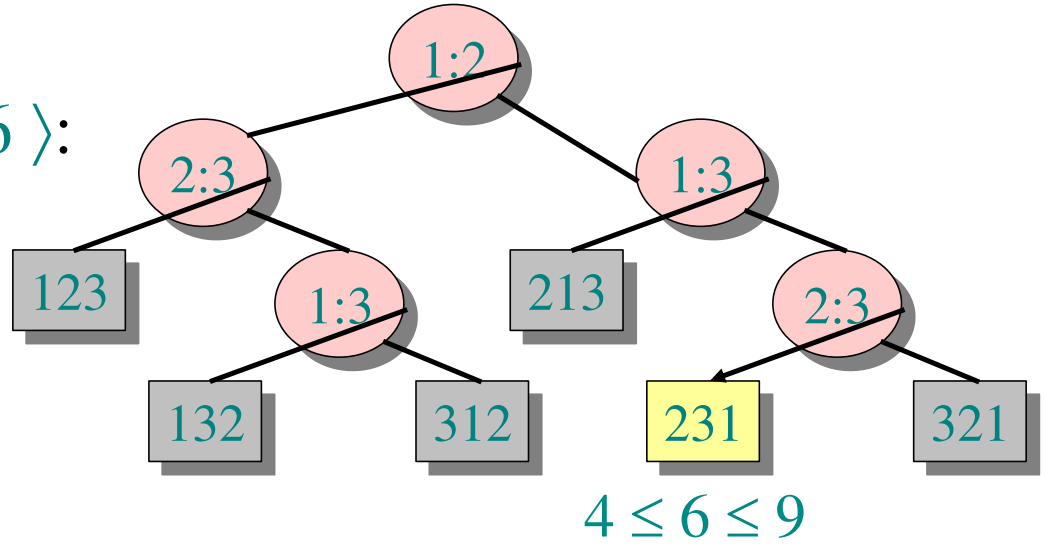
Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

Lower bound of sorting

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle = \langle 9, 4, 6 \rangle$:



Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$ has been established.

Lower bound of sorting

Decision-tree model

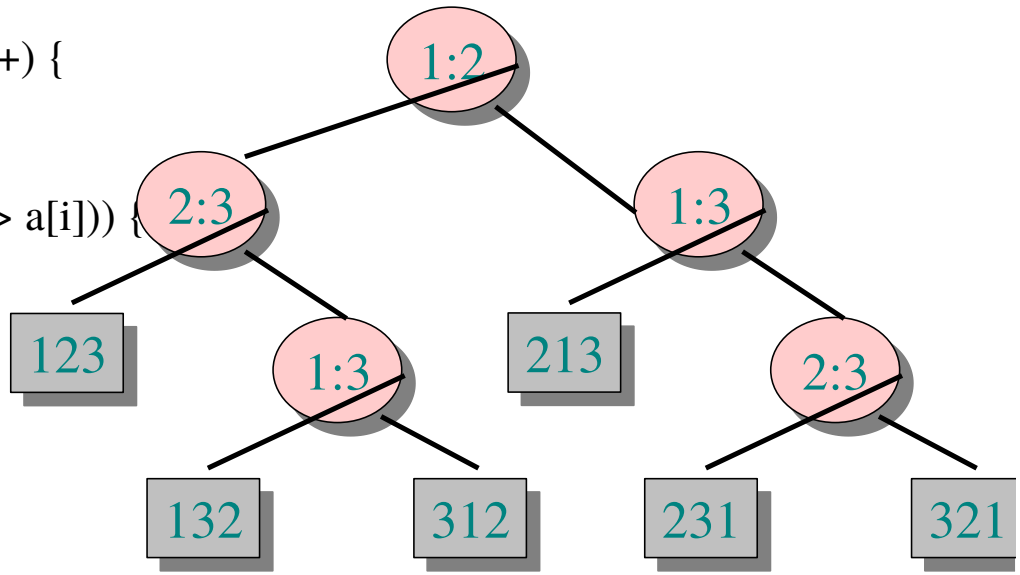
A decision tree can model the execution of any comparison sort:

- One tree for each input size n.
- View the algorithm as splitting whenever it compares two elements.
- The tree contains the comparisons along all possible instruction traces.
- The running time of the algorithm = the length of the path taken.
- Worst-case running time = height of tree.

Lower bound of sorting

Any comparison sort
Can be turned into a Decision tree

```
class InsertionSortAlgorithm  {
    for (int i = 1; i < a.length; i++) {
        int j = i;
        while ((j > 0) && (a[j-1] > a[i])) {
              a[j] = a[j-1];
              j--;  }
        a[j] = B;  }}
```



Lower bound of sorting

Lower bound for decision-tree sorting

Theorem. Any decision tree that can sort n elements must have height $\Omega(n \lg n)$.

Proof. The tree must contain $\geq n!$ leaves, since there are $n!$ possible permutations. A height-h binary tree has $\leq 2^h$ leaves. Thus, $n! \leq 2^h$.

$$
\begin{aligned}
\therefore\ h &\geq \lg(n!) && \text{($\lg$ is mono. increasing)} \\
&\geq \lg\left((n/e)^n\right) && \text{(Stirling's formula)} \\
&= n \lg n - n \lg e \\
&= \Omega(n \lg n)\,. && \square
\end{aligned}
$$