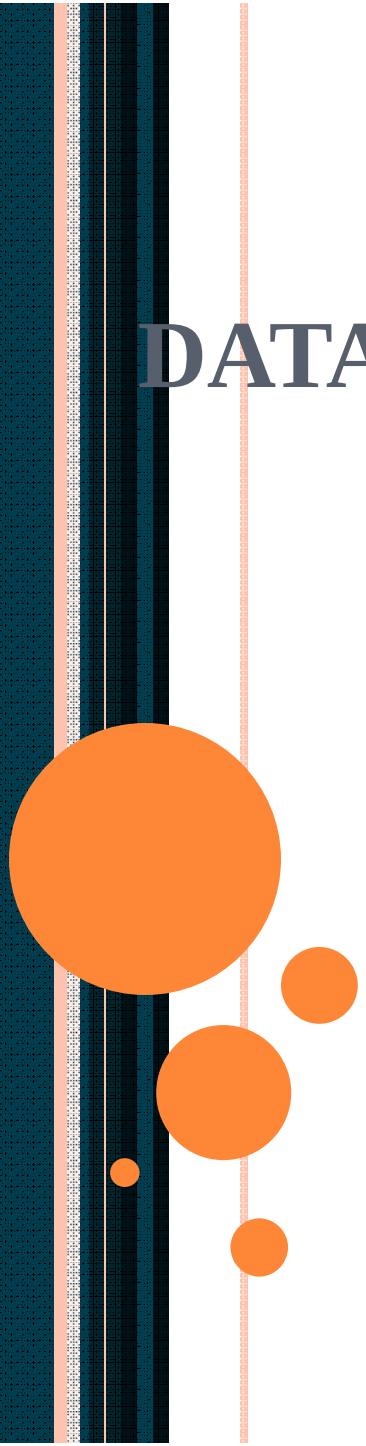
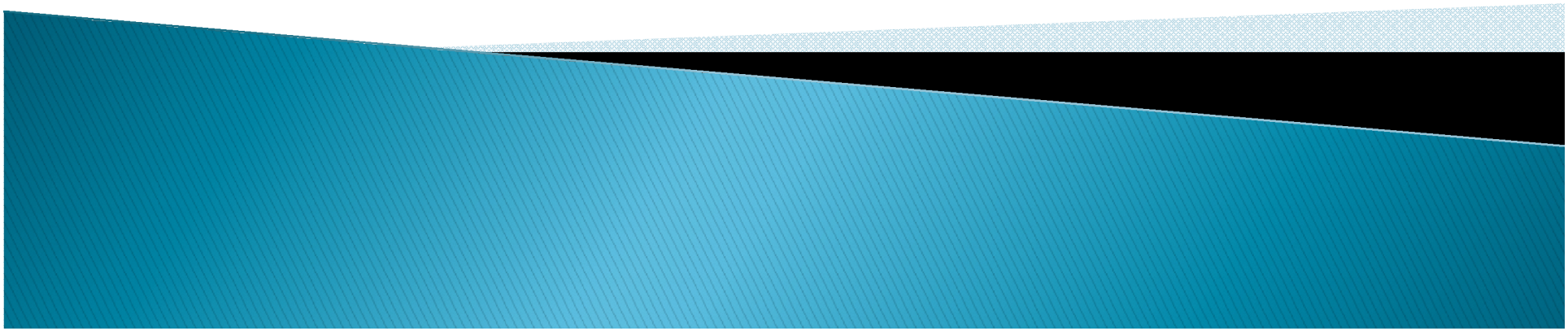
DATA STRUCTURES USING ‘C’

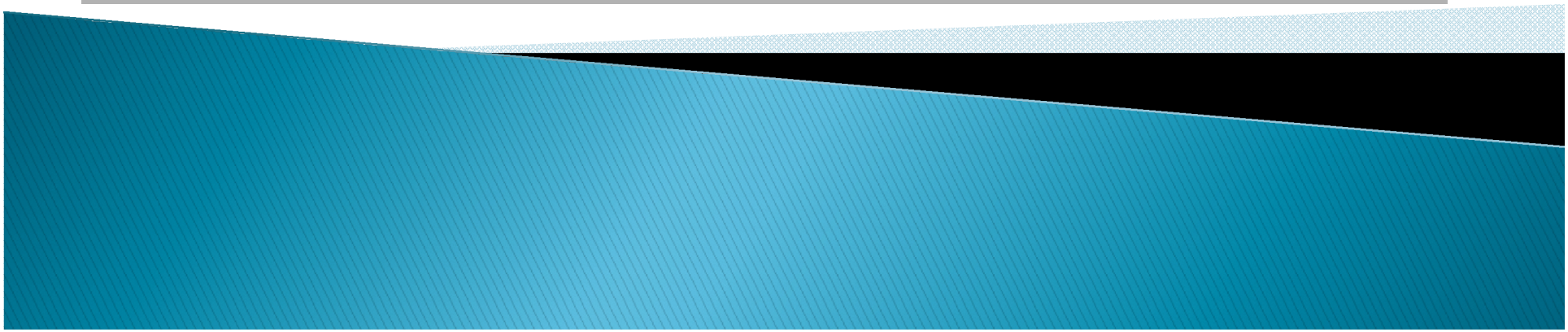


Lecture-11

Data Structures

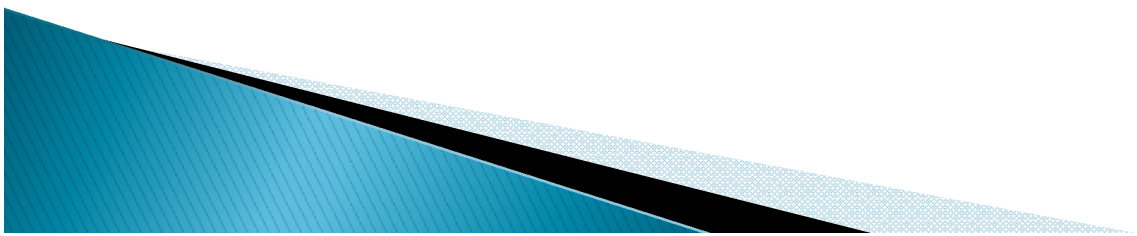


Different types of Sorting Techniques used in Data Structures



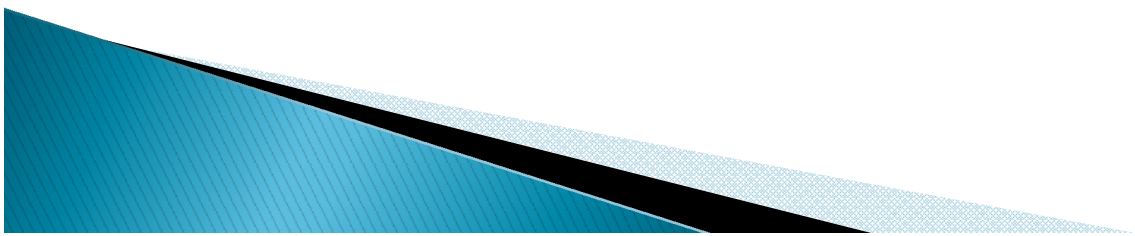
Selection Sort: Idea

1. We have two group of items:
   - sorted group, and
   - unsorted group
2. Initially, all items are in the unsorted group. The sorted group is empty.
   - We assume that items in the unsorted group unsorted.
   - We have to keep items in the sorted group sorted.



Selection Sort: Cont'd

1. Select the "best" (eg. smallest) item from the unsorted group, then put the "best" item at the end of the sorted group.
2. Repeat the process until the unsorted group becomes empty.



Selection Sort

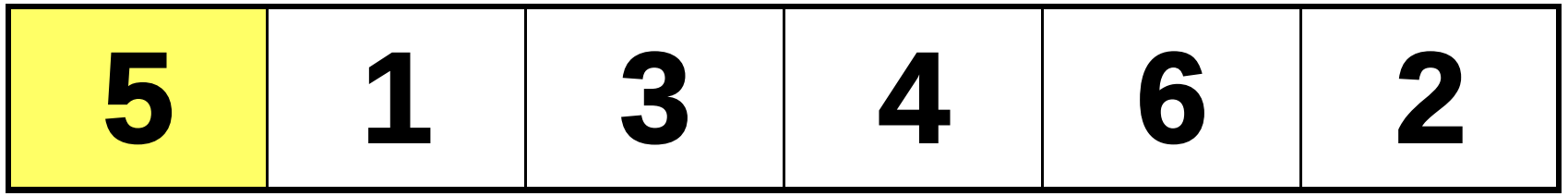
5	1	3	4	6	2

Comparison

Data Movement

Sorted

Selection Sort

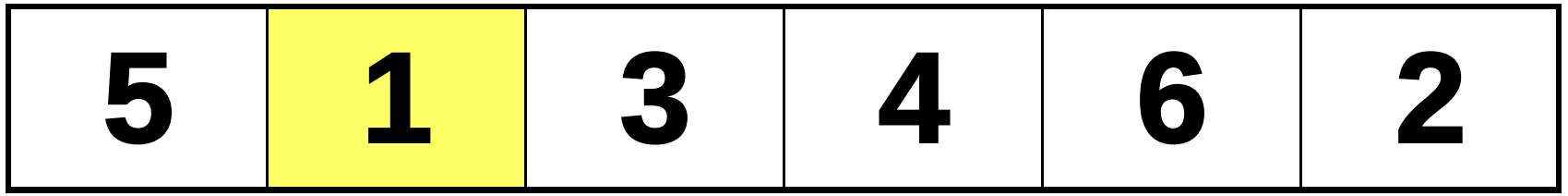


Comparison

Data Movement

Sorted

Selection Sort

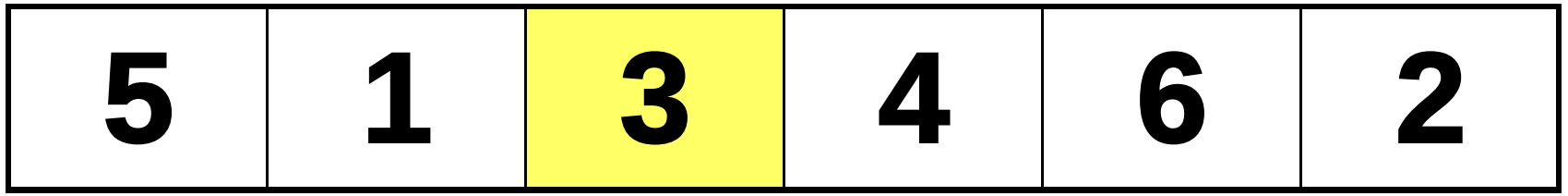


Comparison

Data Movement

Sorted

Selection Sort

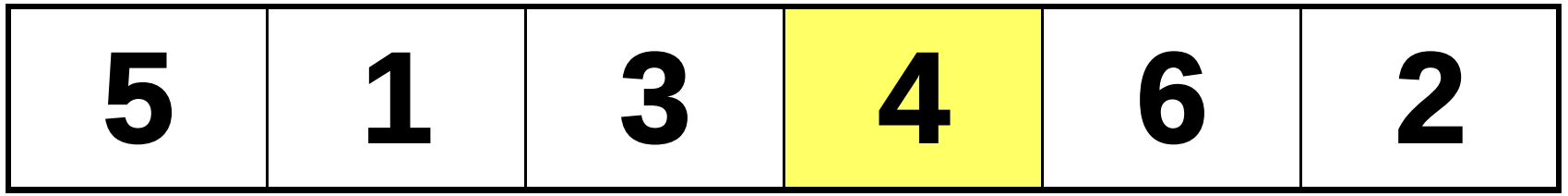


Comparison

Data Movement

Sorted

Selection Sort

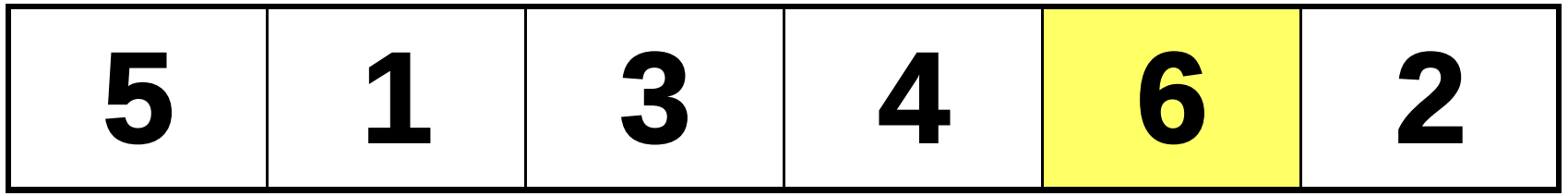


Comparison

Data Movement

Sorted

Selection Sort

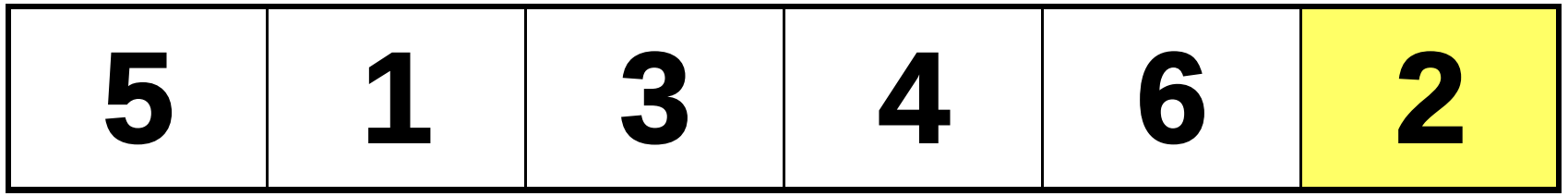


Comparison

Data Movement

Sorted

Selection Sort

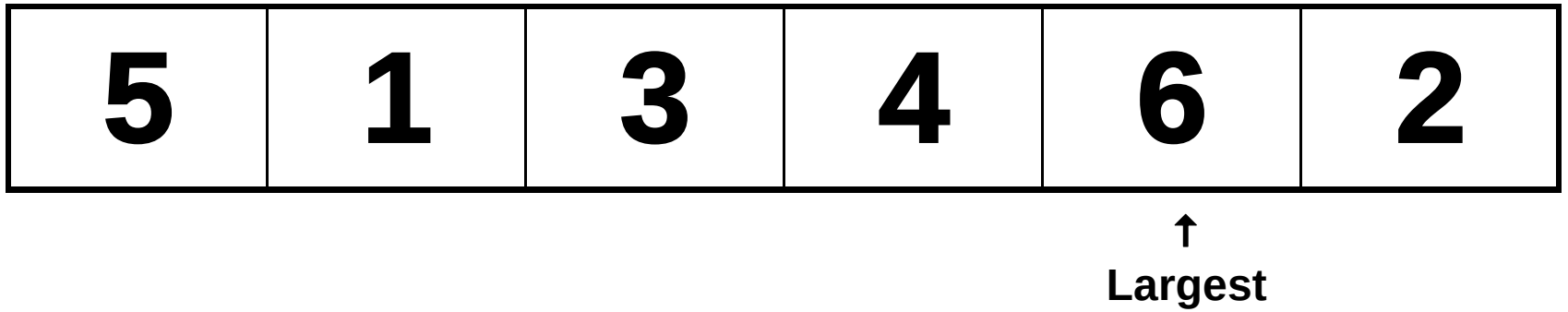


Comparison

Data Movement

Sorted

Selection Sort

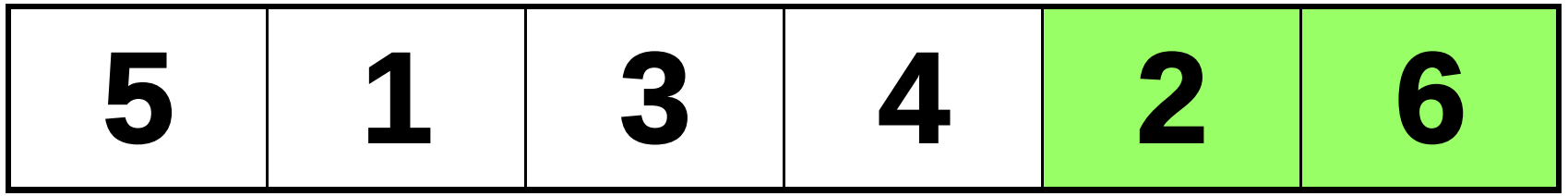


Comparison

Data Movement

Sorted

Selection Sort

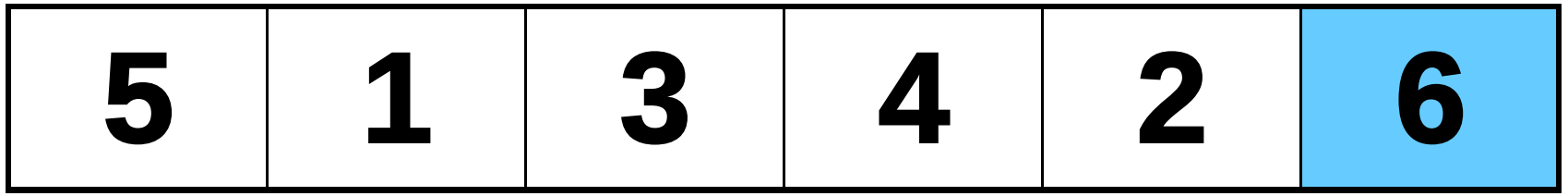


Comparison

Data Movement

Sorted

Selection Sort

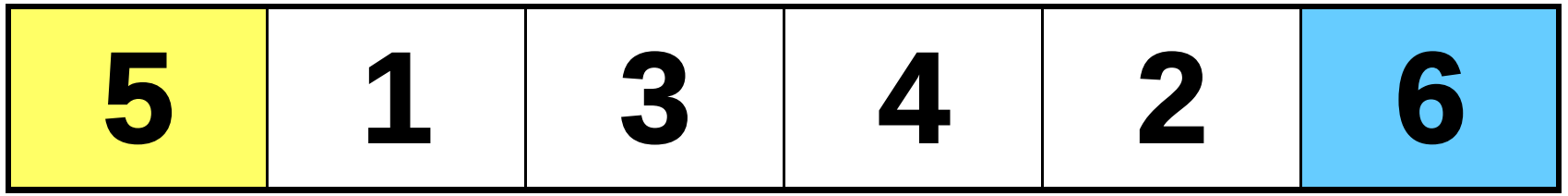


Comparison

Data Movement

Sorted

Selection Sort

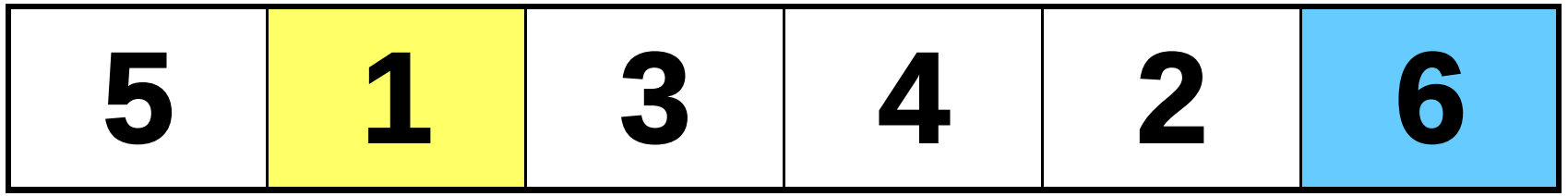


Comparison

Data Movement

Sorted

Selection Sort

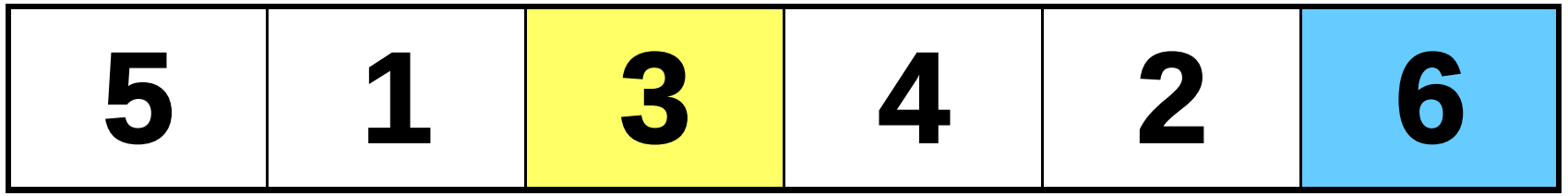


Comparison

Data Movement

Sorted

Selection Sort

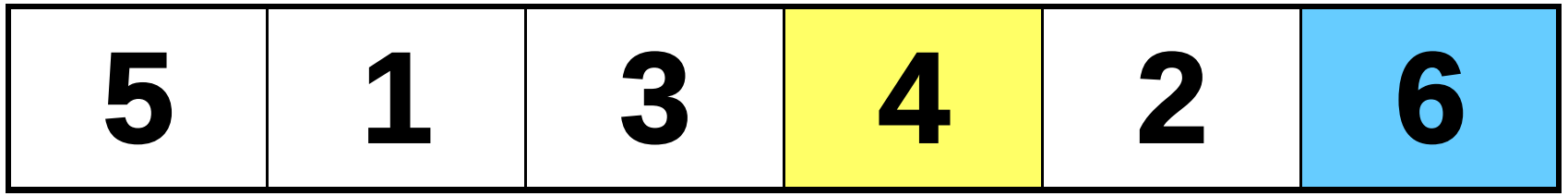


Comparison

Data Movement

Sorted

Selection Sort

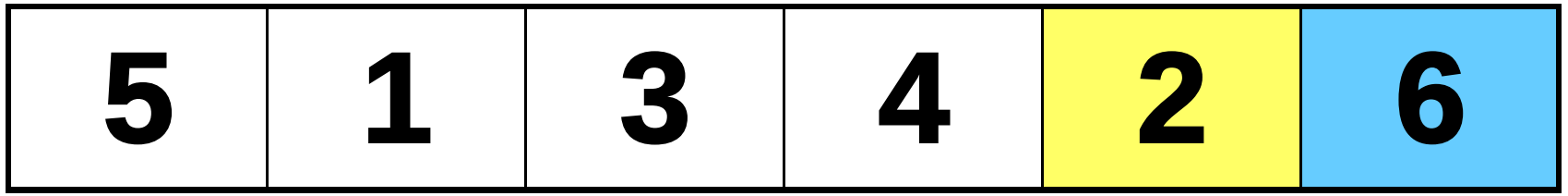


Comparison

Data Movement

Sorted

Selection Sort

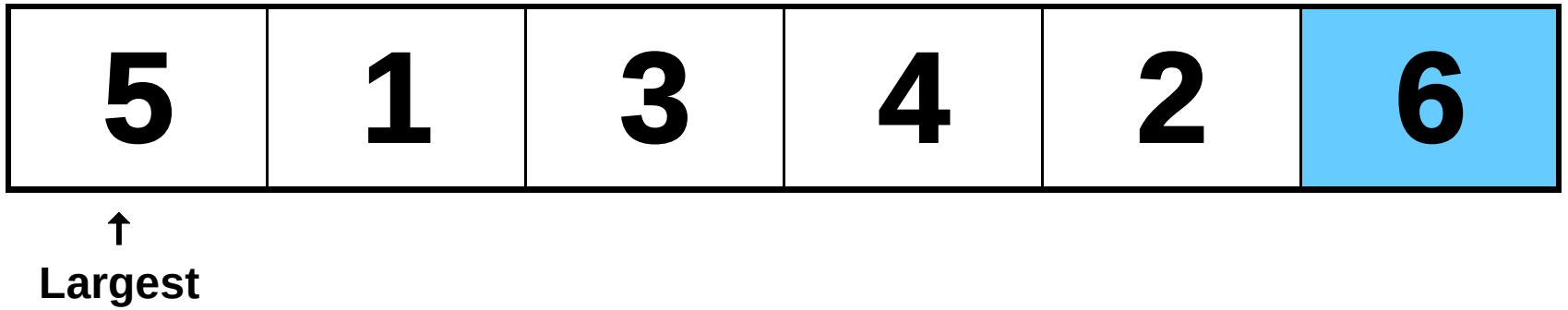


Comparison

Data Movement

Sorted

Selection Sort

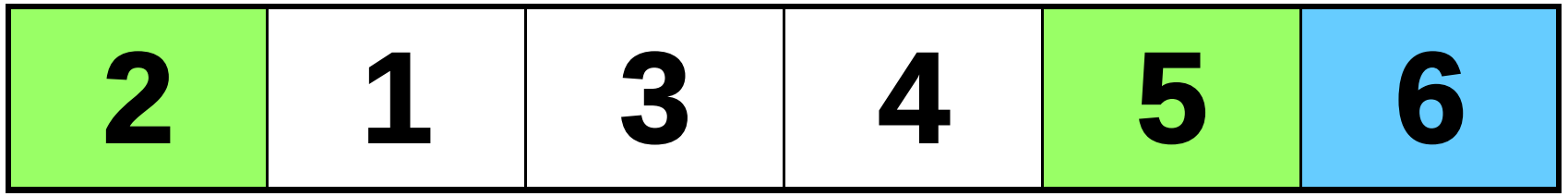


Comparison

Data Movement

Sorted

Selection Sort

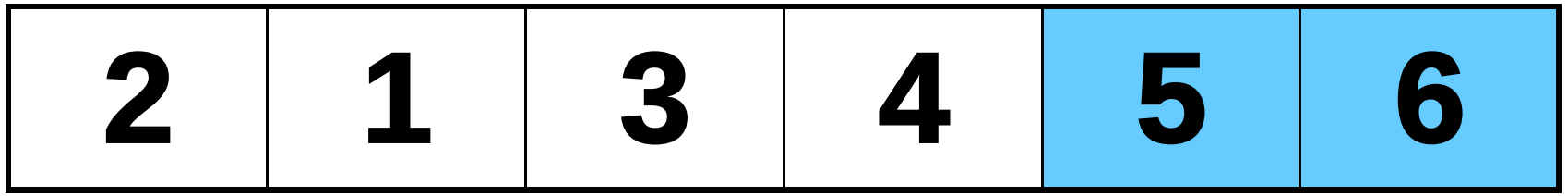


Comparison

Data Movement

Sorted

Selection Sort

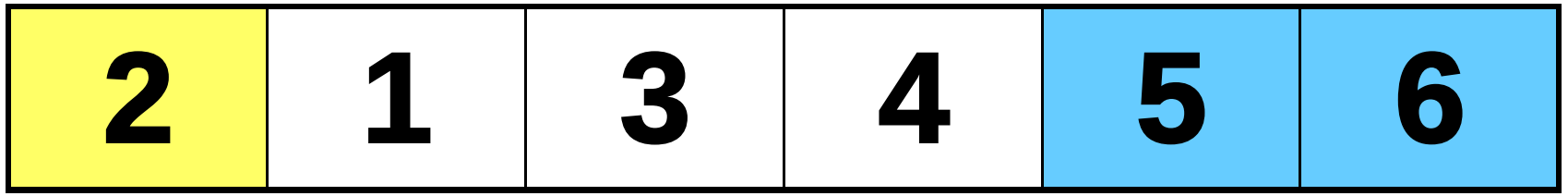


Comparison

Data Movement

Sorted

Selection Sort

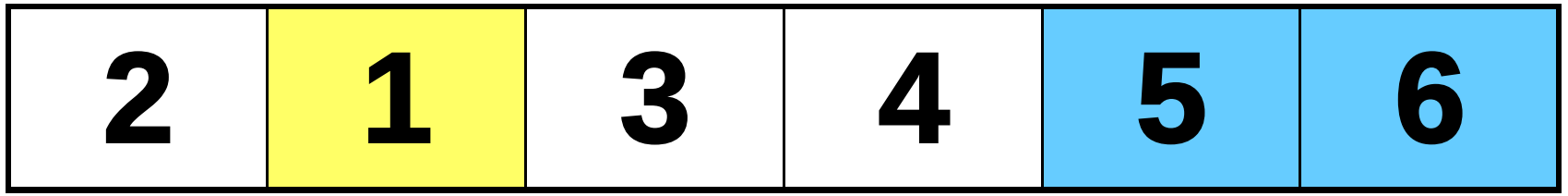


Comparison

Data Movement

Sorted

Selection Sort

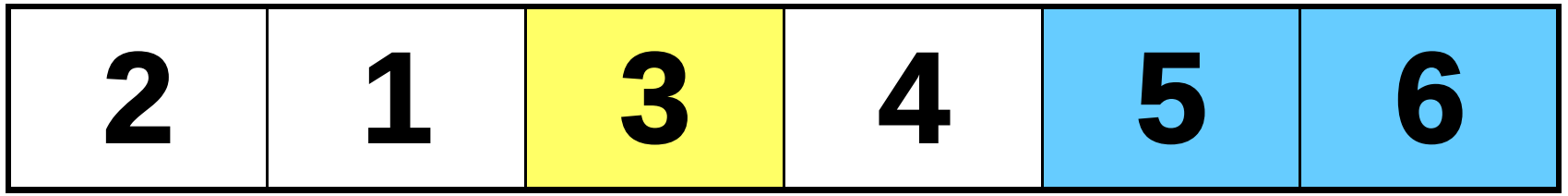


Comparison

Data Movement

Sorted

Selection Sort

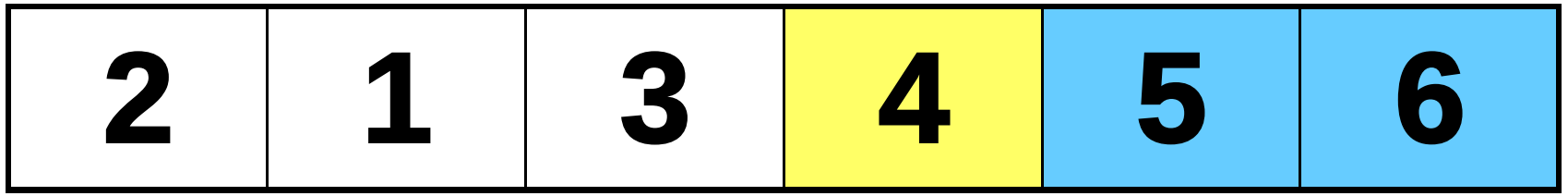


Comparison

Data Movement

Sorted

Selection Sort

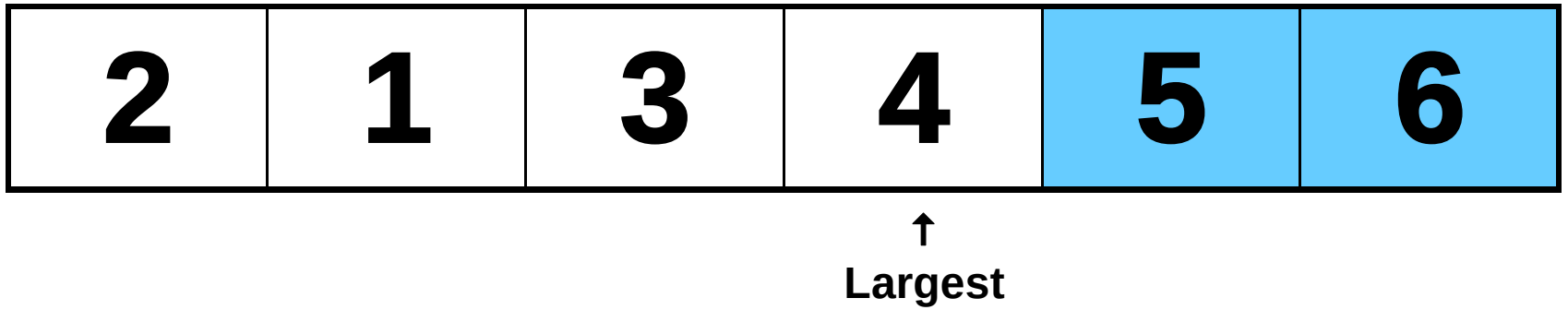


Comparison

Data Movement

Sorted

Selection Sort

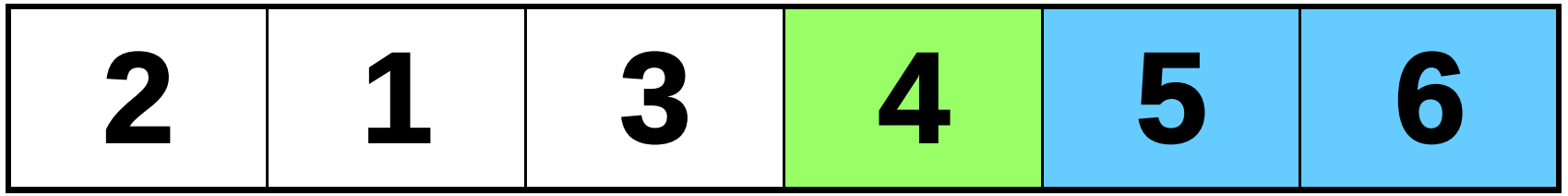


Comparison

Data Movement

Sorted

Selection Sort

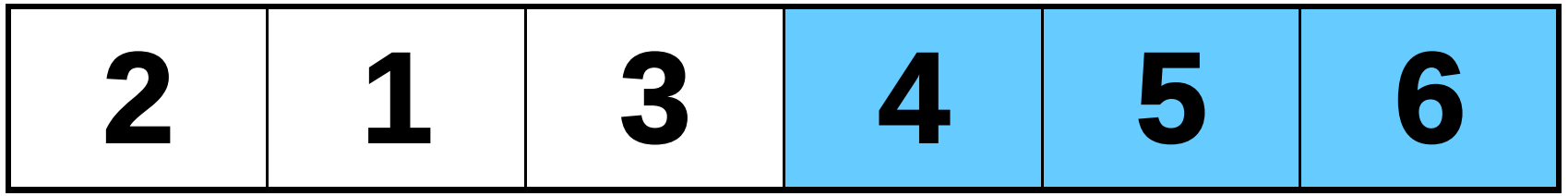


Comparison

Data Movement

Sorted

Selection Sort

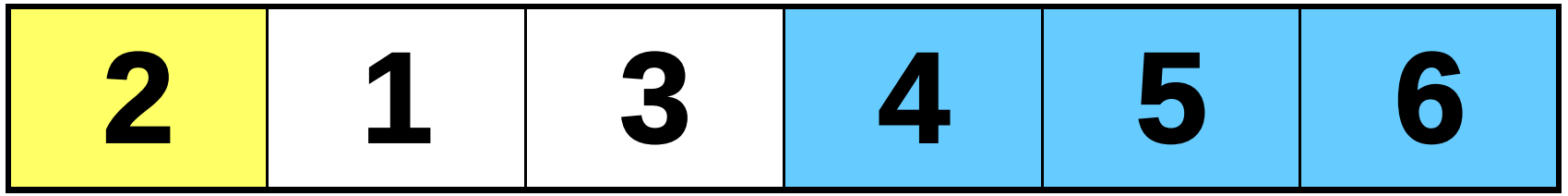


Comparison

Data Movement

Sorted

Selection Sort

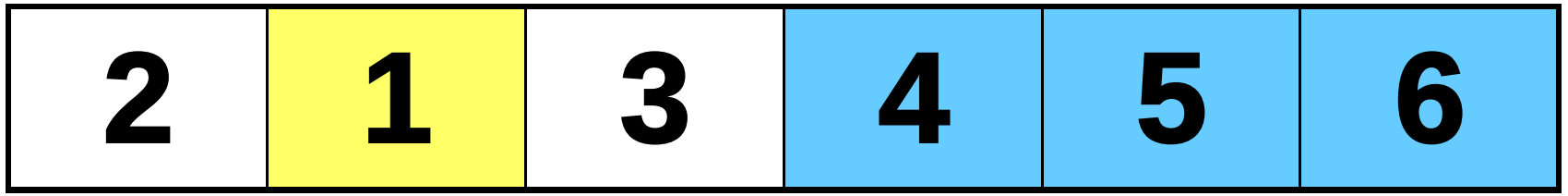


Comparison

Data Movement

Sorted

Selection Sort

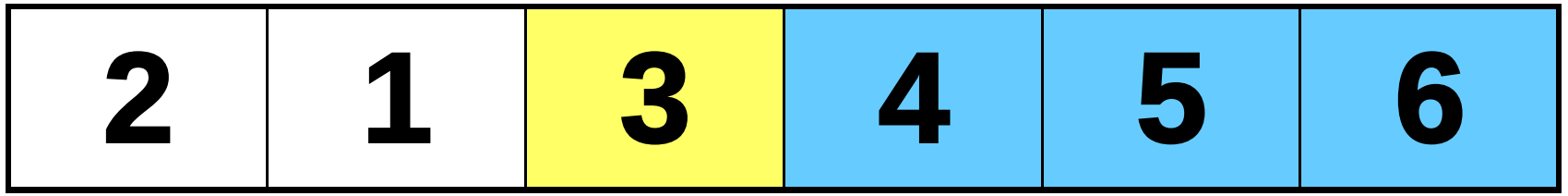


Comparison

Data Movement

Sorted

Selection Sort

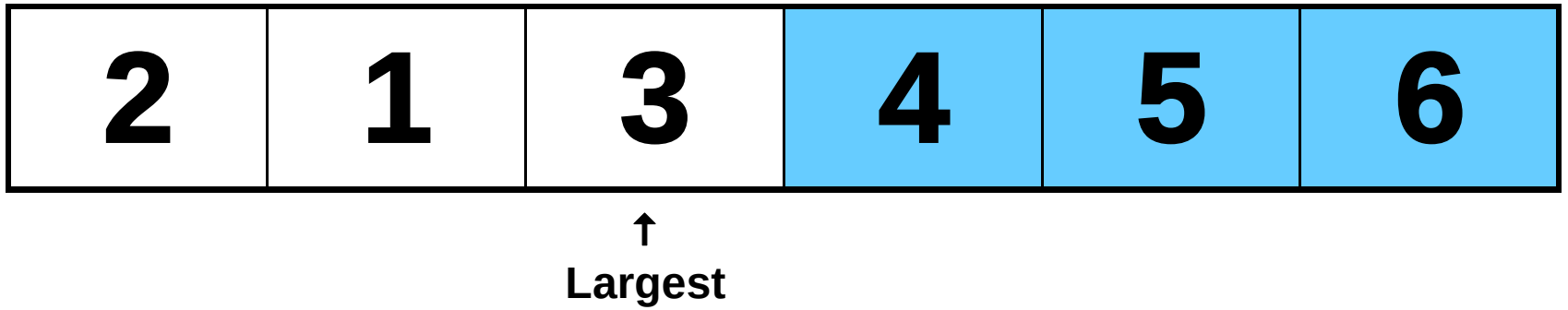


Comparison

Data Movement

Sorted

Selection Sort

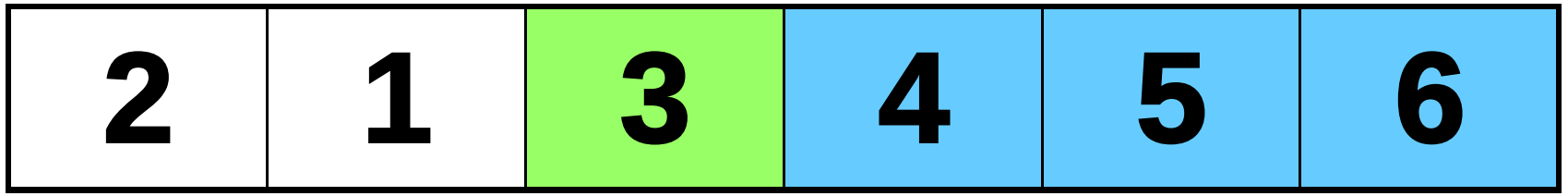


Comparison

Data Movement

Sorted

Selection Sort

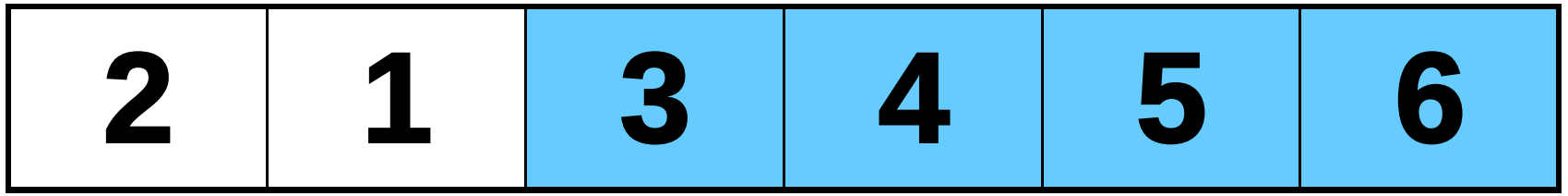


Comparison

Data Movement

Sorted

Selection Sort

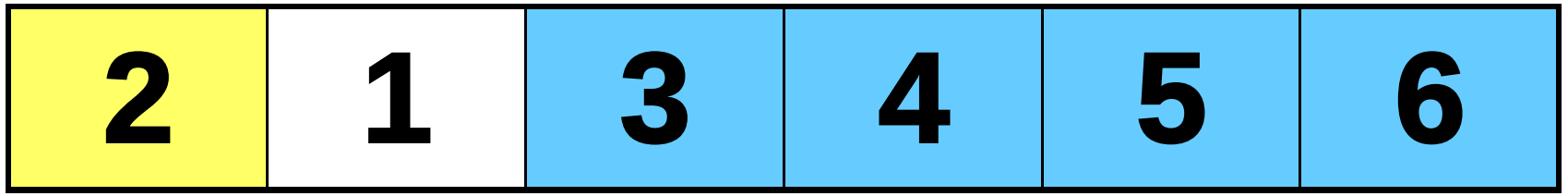


Comparison

Data Movement

Sorted

Selection Sort

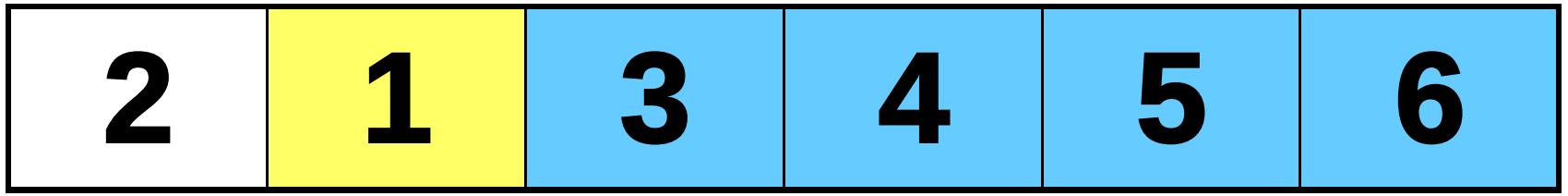


Comparison

Data Movement

Sorted

Selection Sort

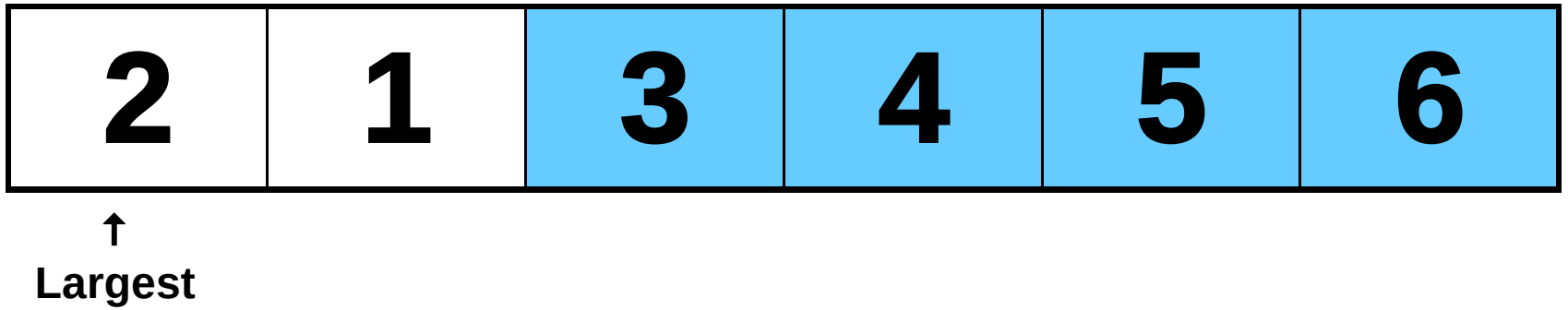


Comparison

Data Movement

Sorted

Selection Sort

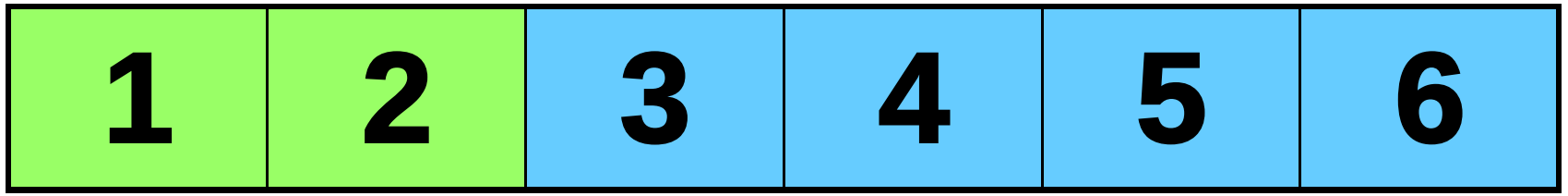


Comparison

Data Movement

Sorted

Selection Sort

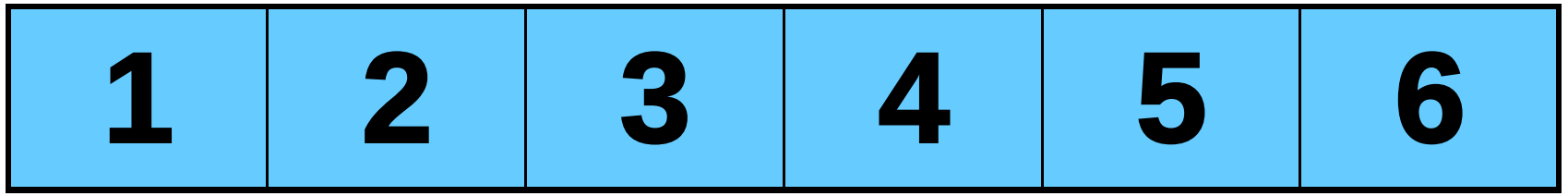


Comparison

Data Movement

Sorted

Selection Sort



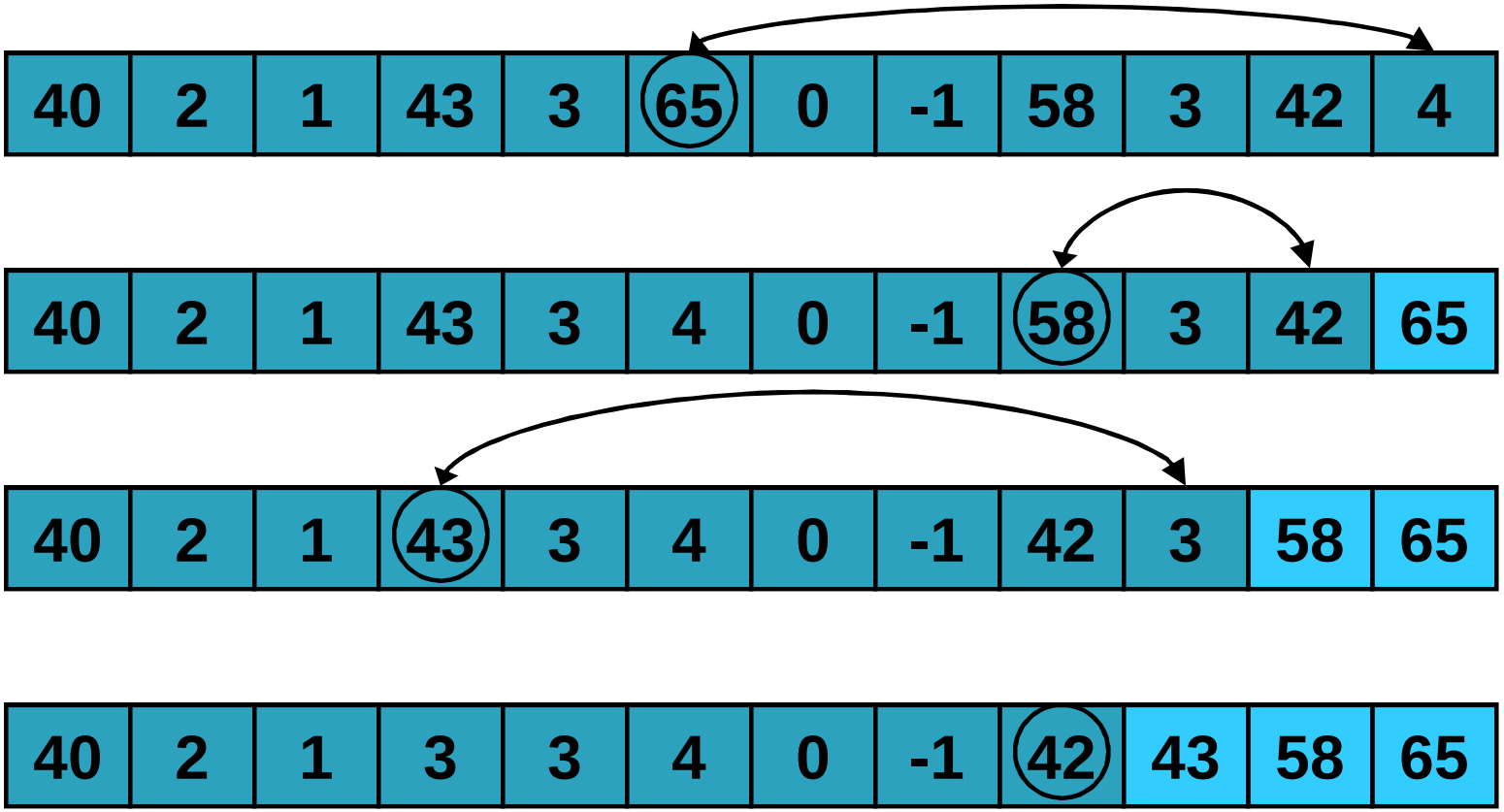
DONE!

Comparison

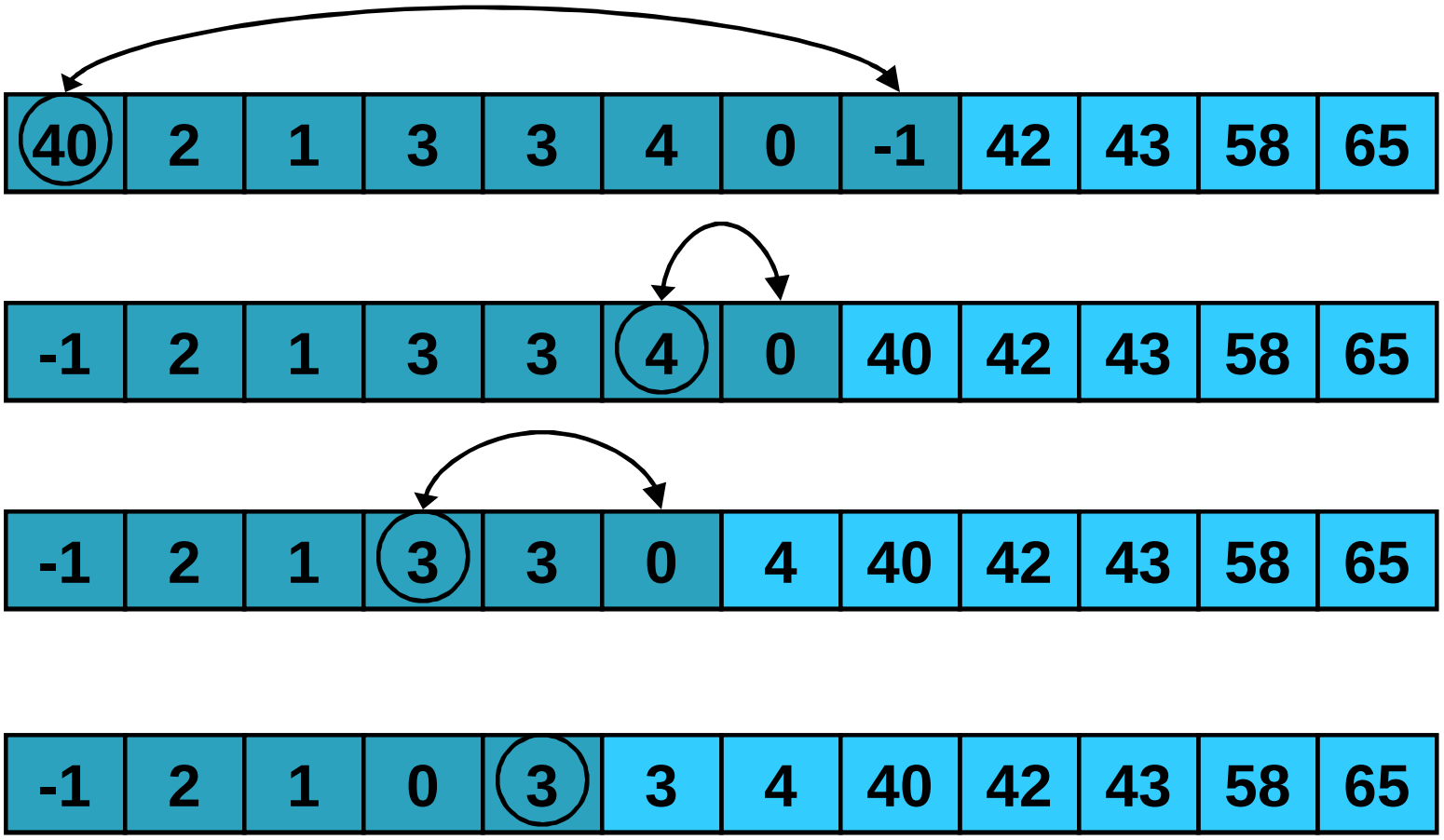
Data Movement

Sorted

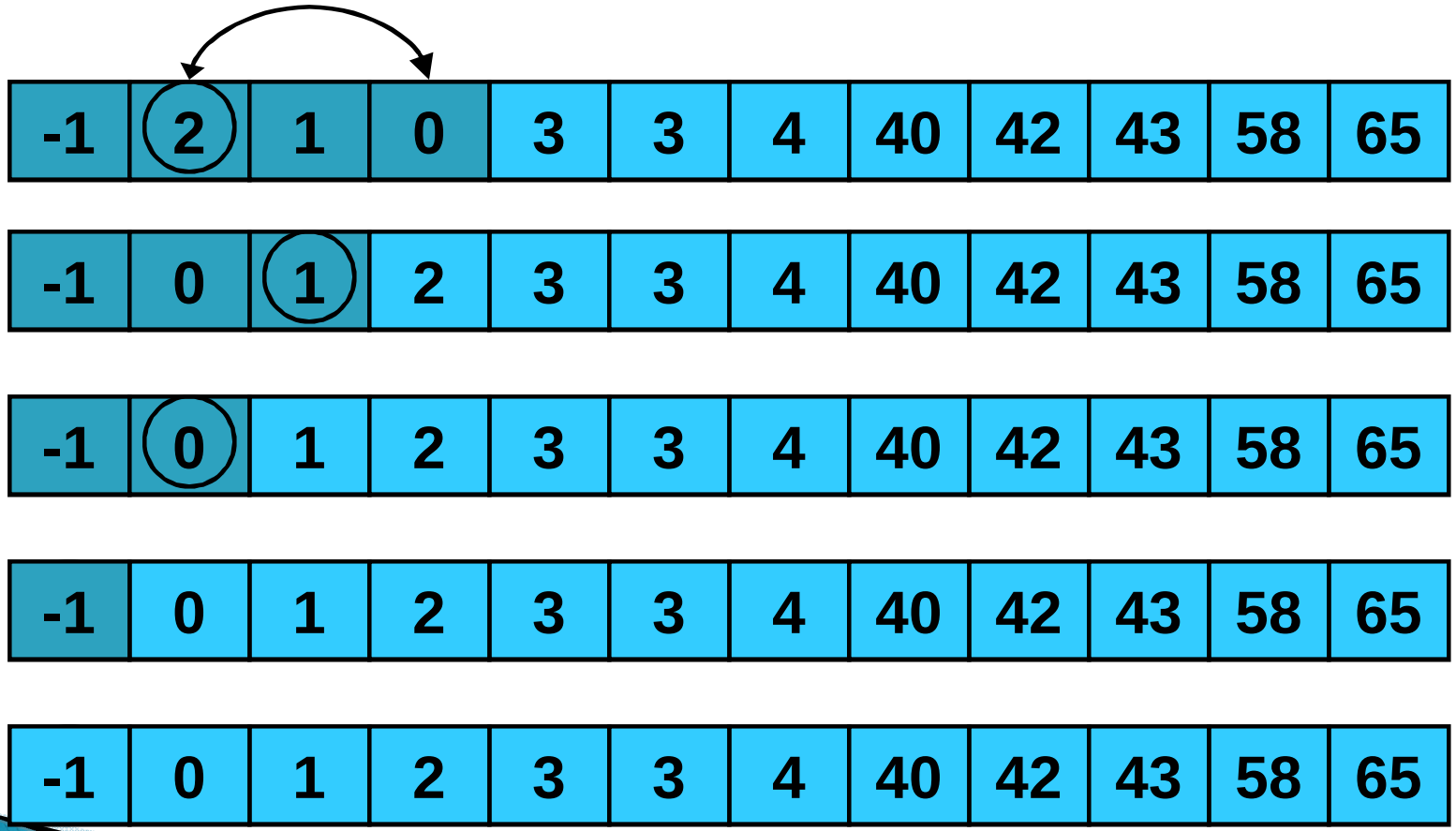
Selection Sort: Example



Selection Sort: Example

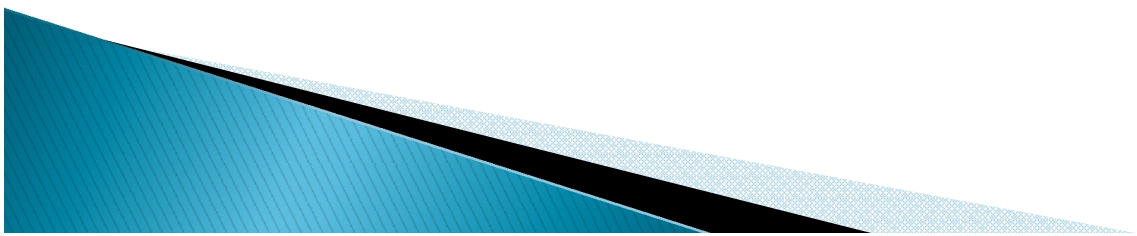


Selection Sort: Example



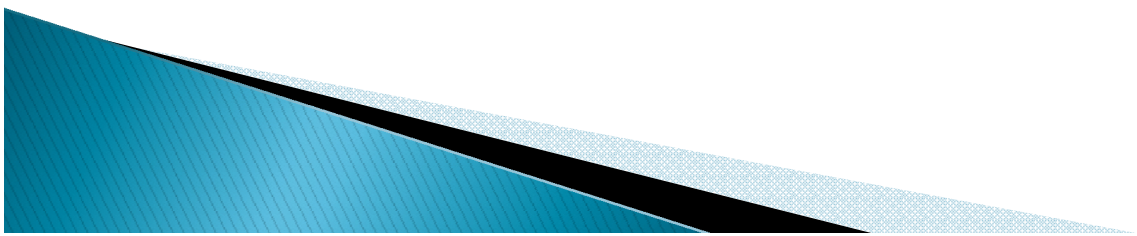
Selection Sort: Analysis

- Running time:
  - Worst case: $O(N^2)$
  - Best case: $O(N^2)$



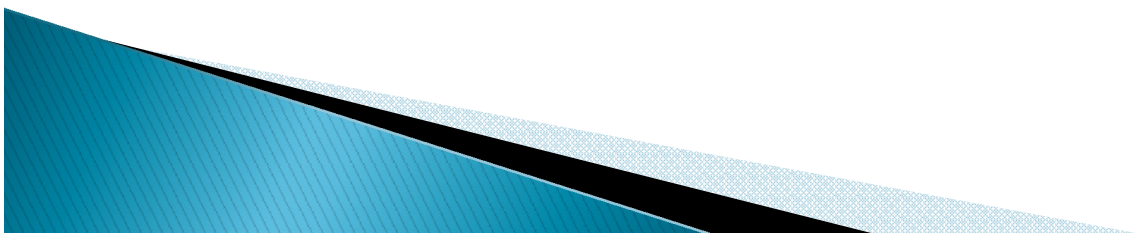
Insertion Sort: Idea

- Idea: sorting cards.
  - **8 | 5 9 2 6 3**
  - **5 8 | 9 2 6 3**
  - **5 8 9 | 2 6 3**
  - **2 5 8 9 | 6 3**
  - **2 5 6 8 9 | 3**
  - **2 3 5 6 8 9 |**

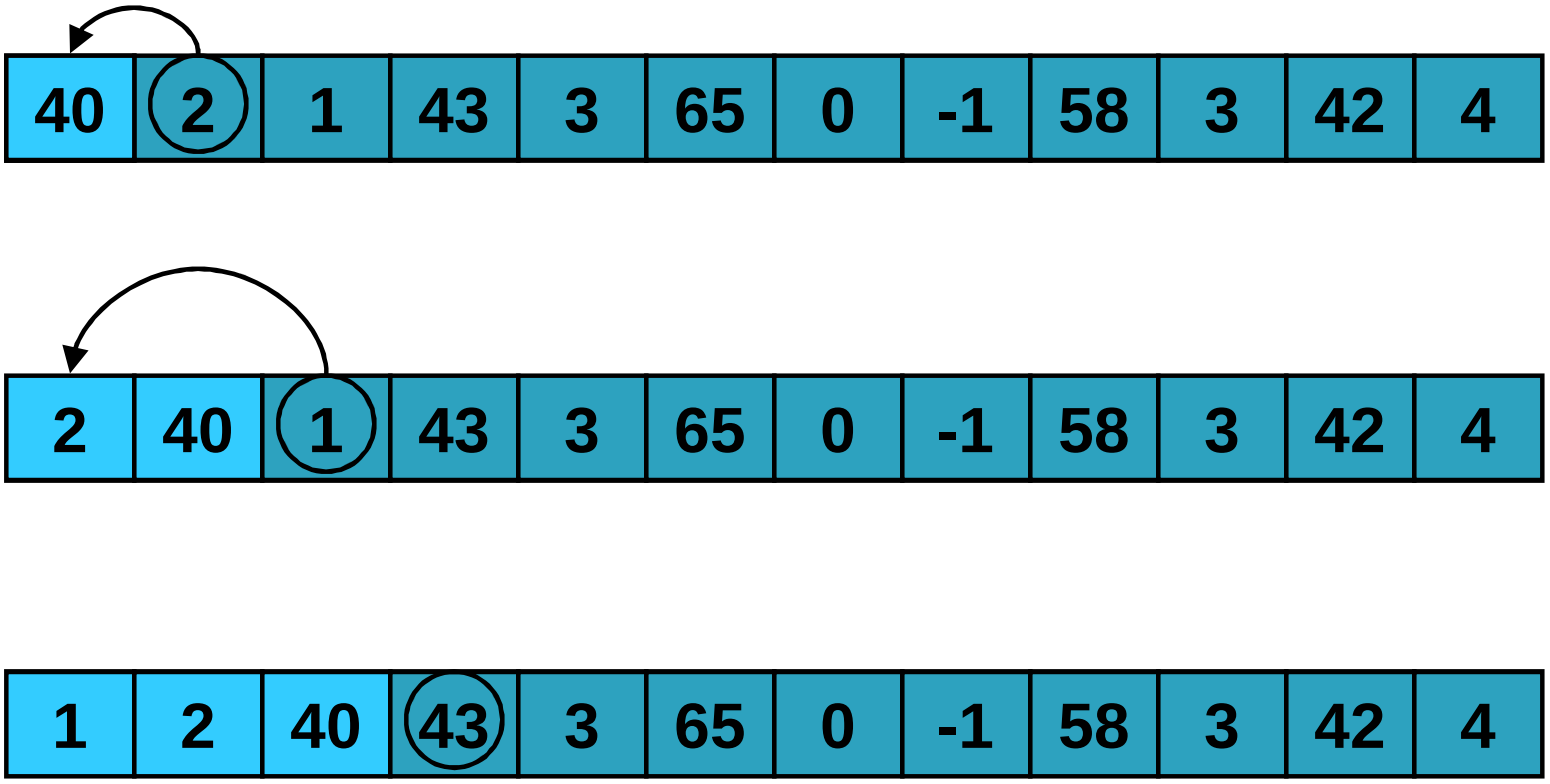


Insertion Sort: Idea

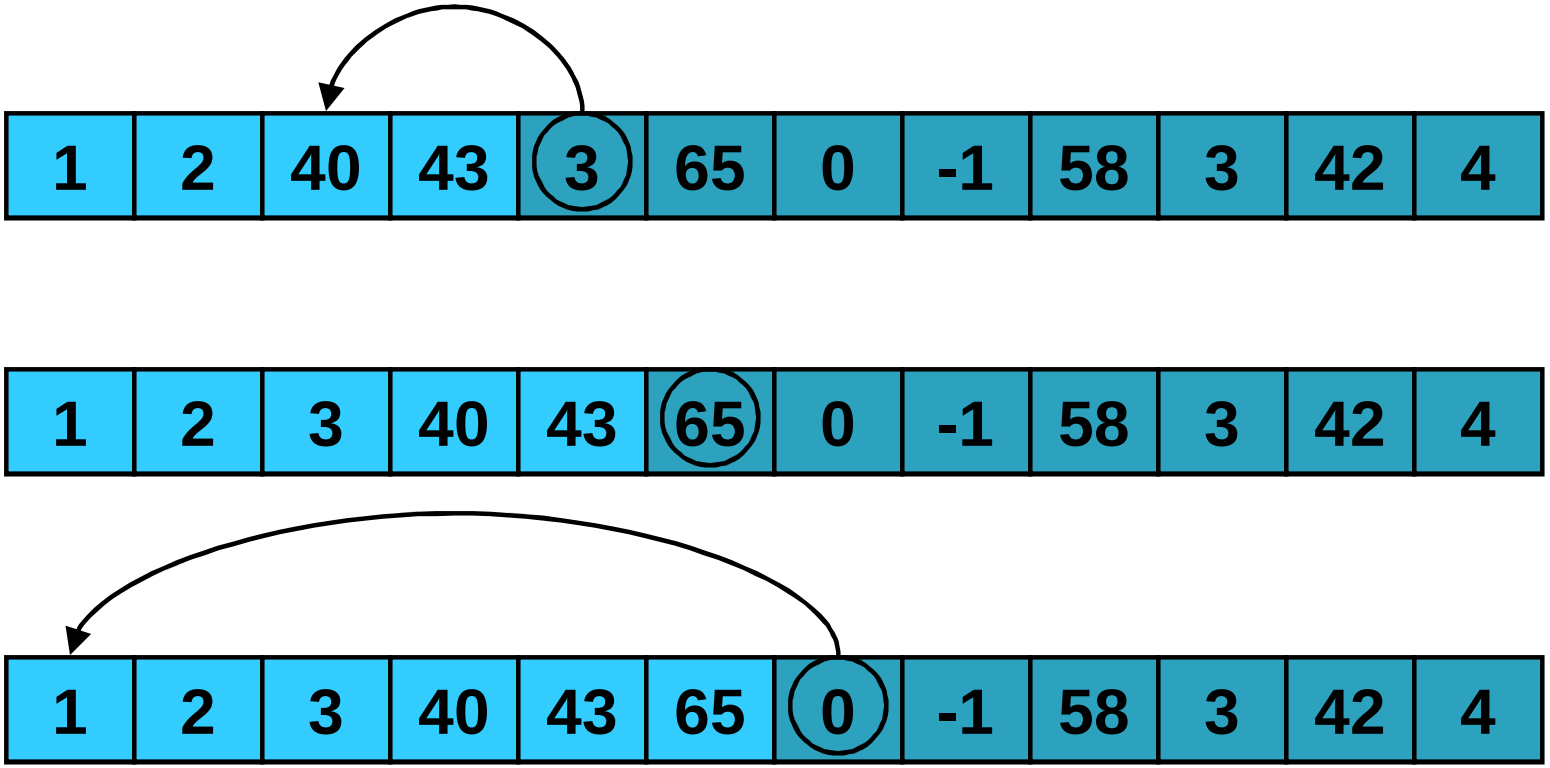
1. We have two group of items:
   - sorted group, and
   - unsorted group
2. Initially, all items in the unsorted group and the sorted group is empty.
   - We assume that items in the unsorted group unsorted.
   - We have to keep items in the sorted group sorted.
3. Pick any item from, then insert the item at the right position in the sorted group to maintain sorted property.
4. Repeat the process until the unsorted group becomes empty.



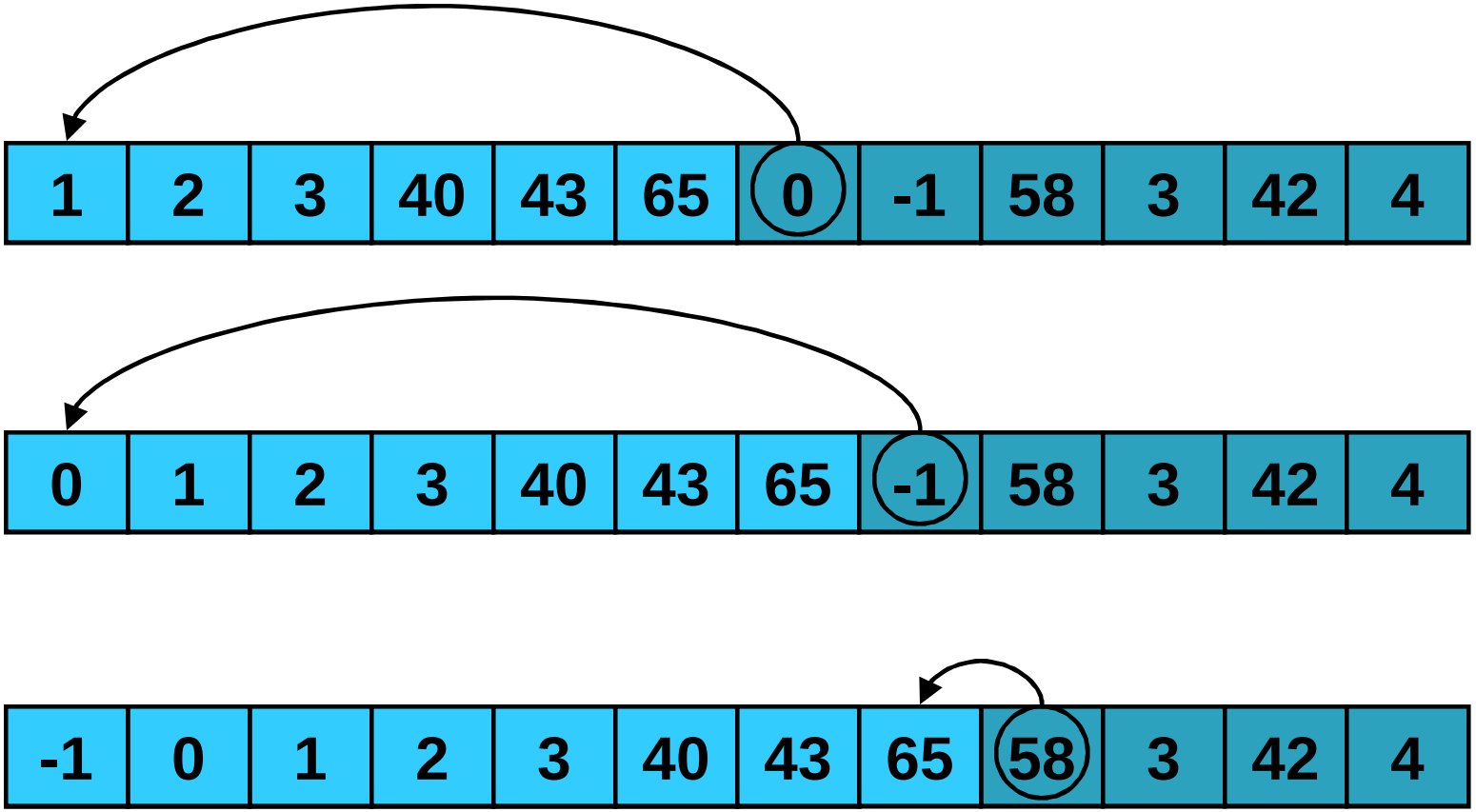
Insertion Sort: Example



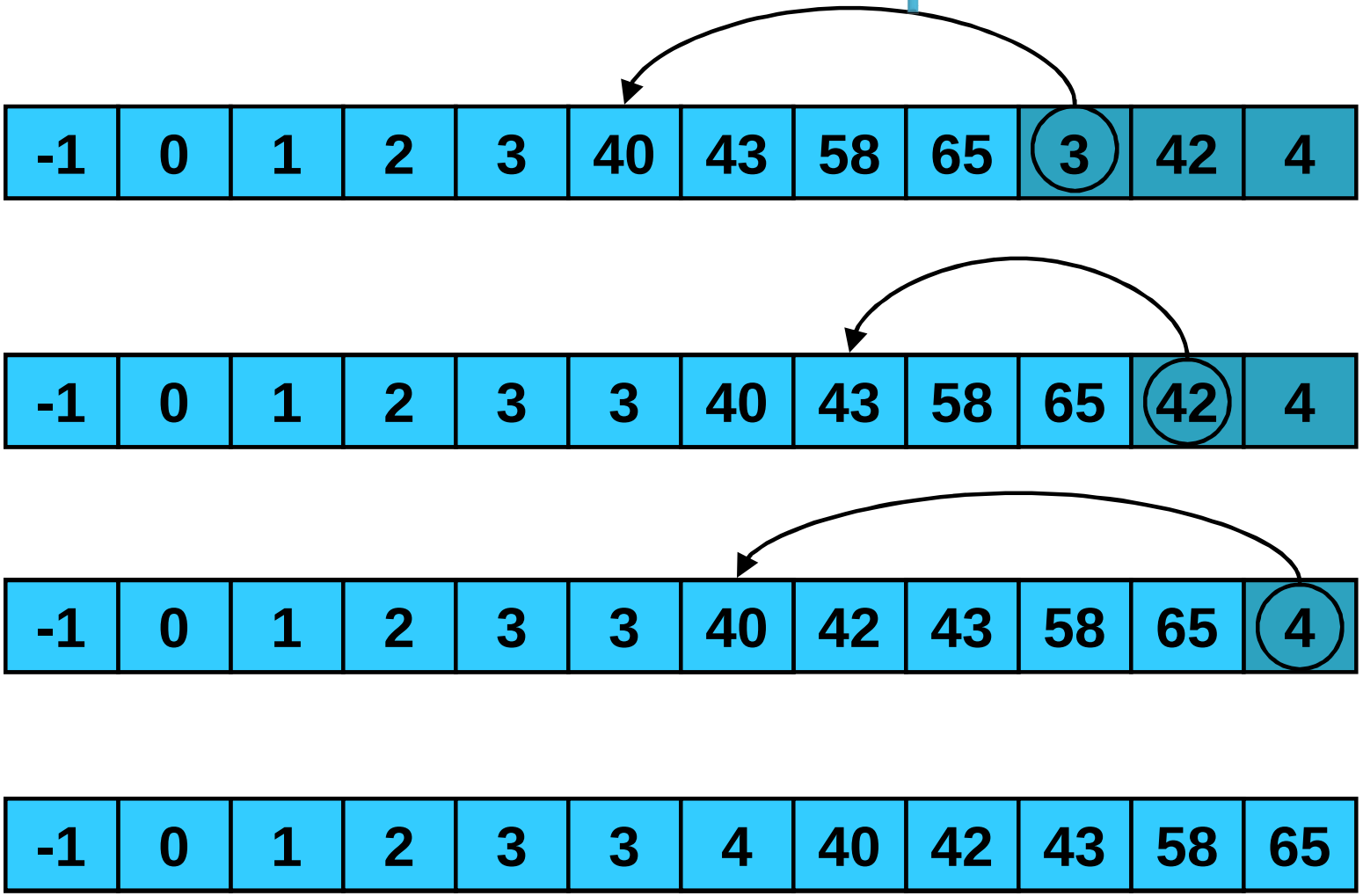
Insertion Sort: Example



Insertion Sort: Example

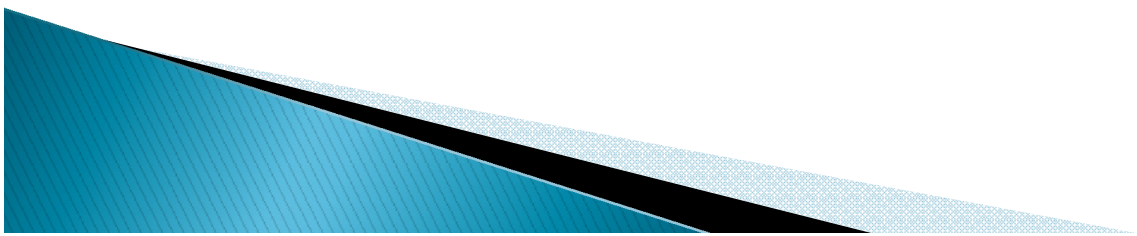


Insertion Sort: Example



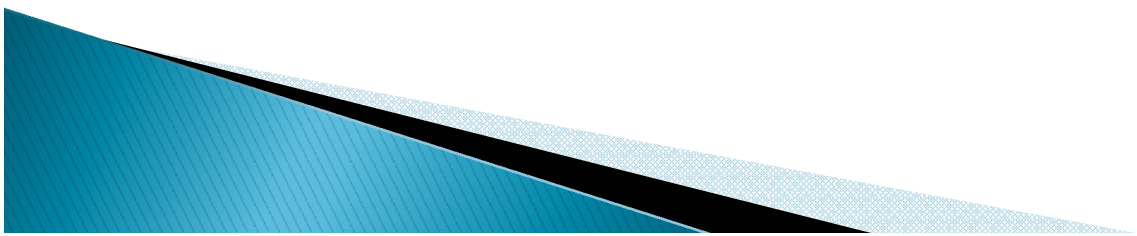
Insertion Sort: Analysis

- Running time analysis:
  - Worst case: $O(N^2)$
  - Best case: O(N)



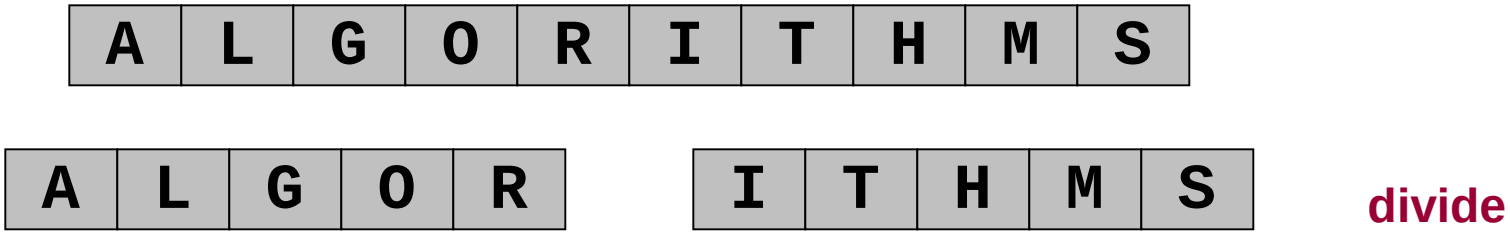
A Lower Bound

- Bubble Sort, Selection Sort, Insertion Sort all have worst case of $O(N^2)$.
- Turns out, for any algorithm that exchanges adjacent items, this is the best worst case: $\Omega(N^2)$
- In other words, this is a lower bound!



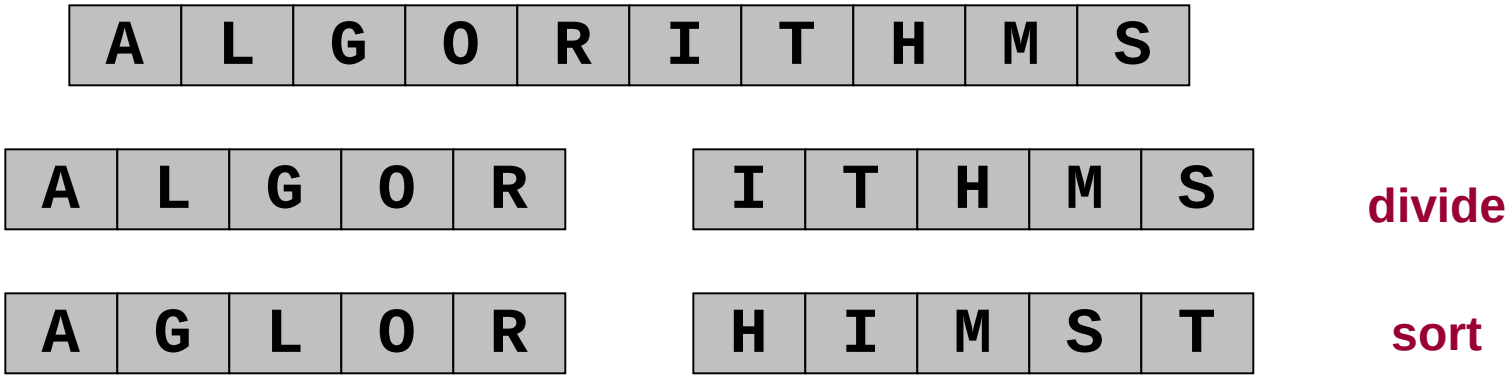
Mergesort

- Mergesort (divide-and-conquer)
  - Divide array into two halves.



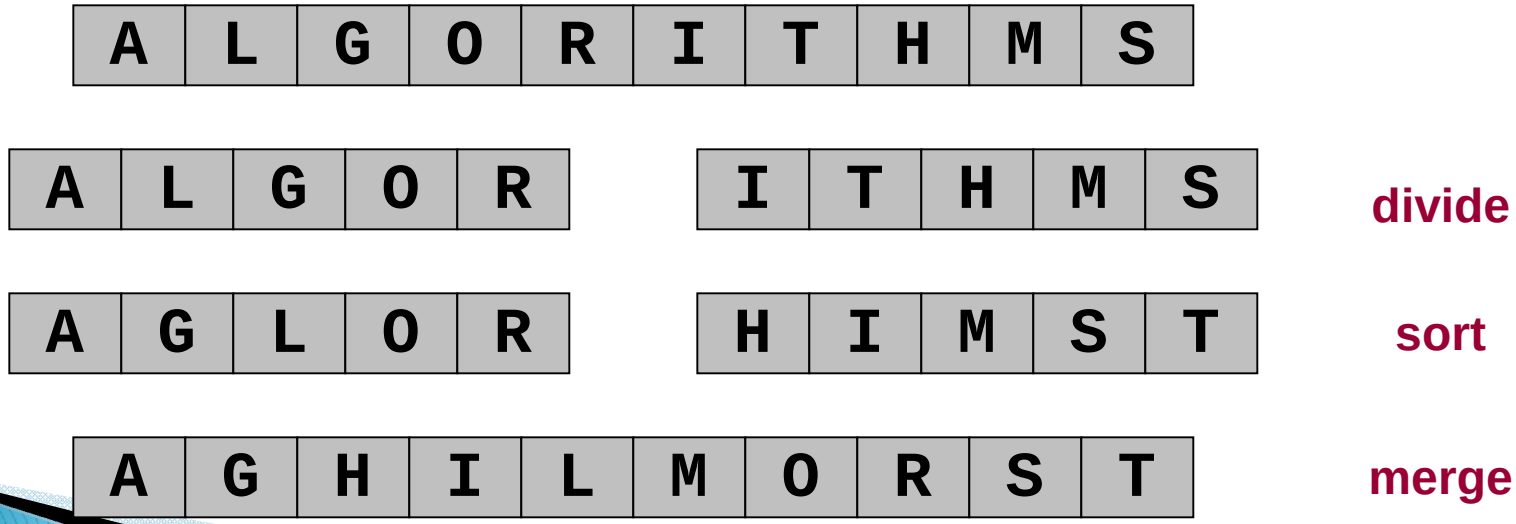
Mergesort

- Mergesort (divide-and-conquer)
  - Divide array into two halves.
  - Recursively sort each half.



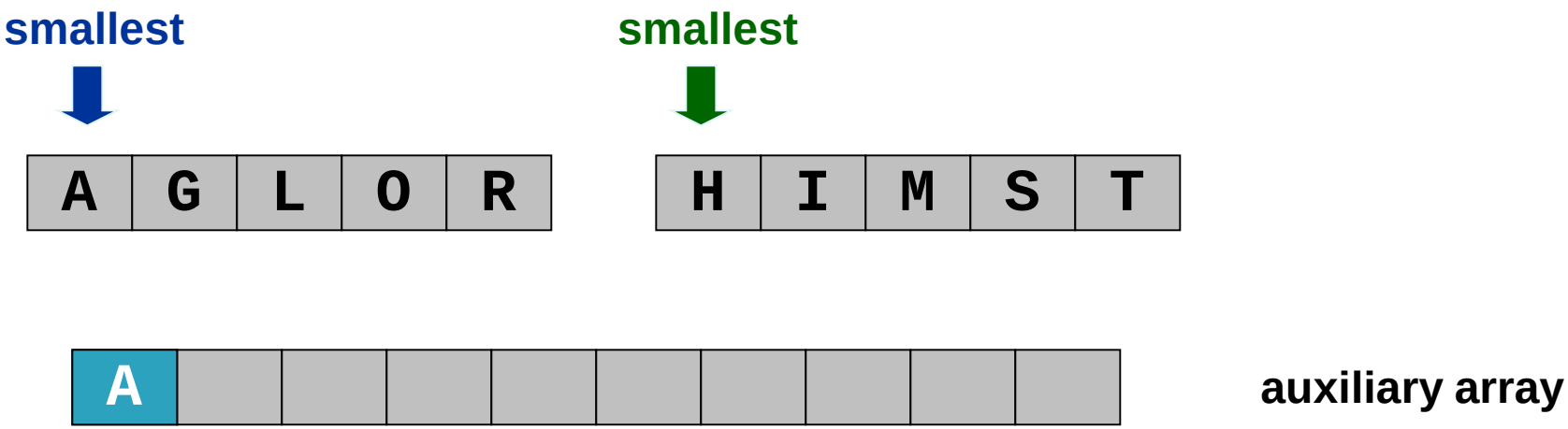
Mergesort

- Mergesort (divide-and-conquer)
  - Divide array into two halves.
  - Recursively sort each half.
  - Merge two halves to make sorted whole.



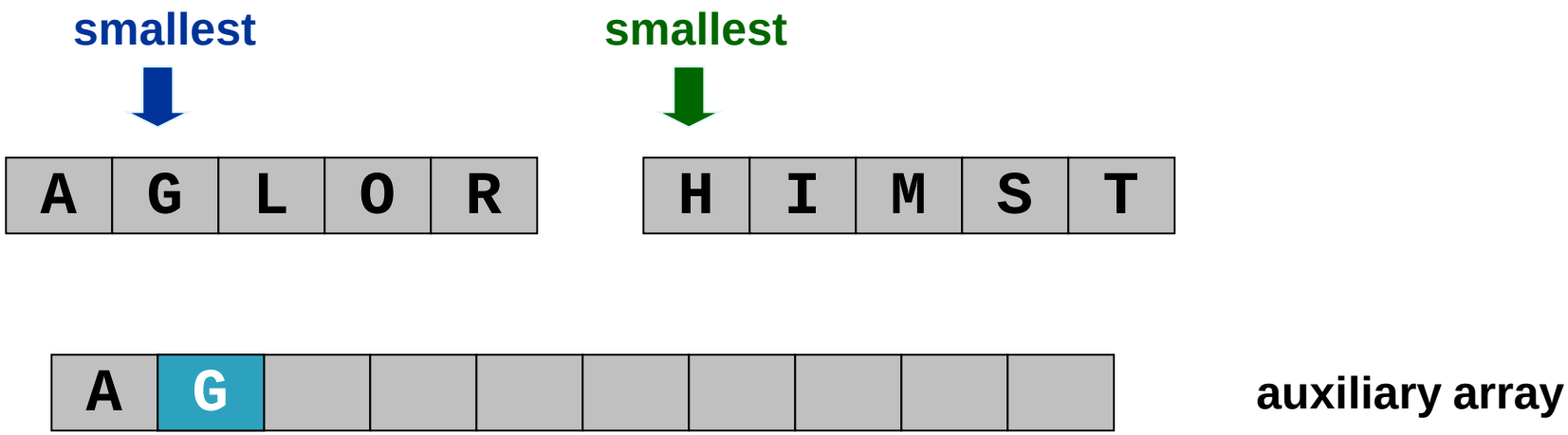
Merging

- Merge.
  - Keep track of smallest element in each sorted half.
  - Insert smallest of two elements into auxiliary array.
  - Repeat until done.



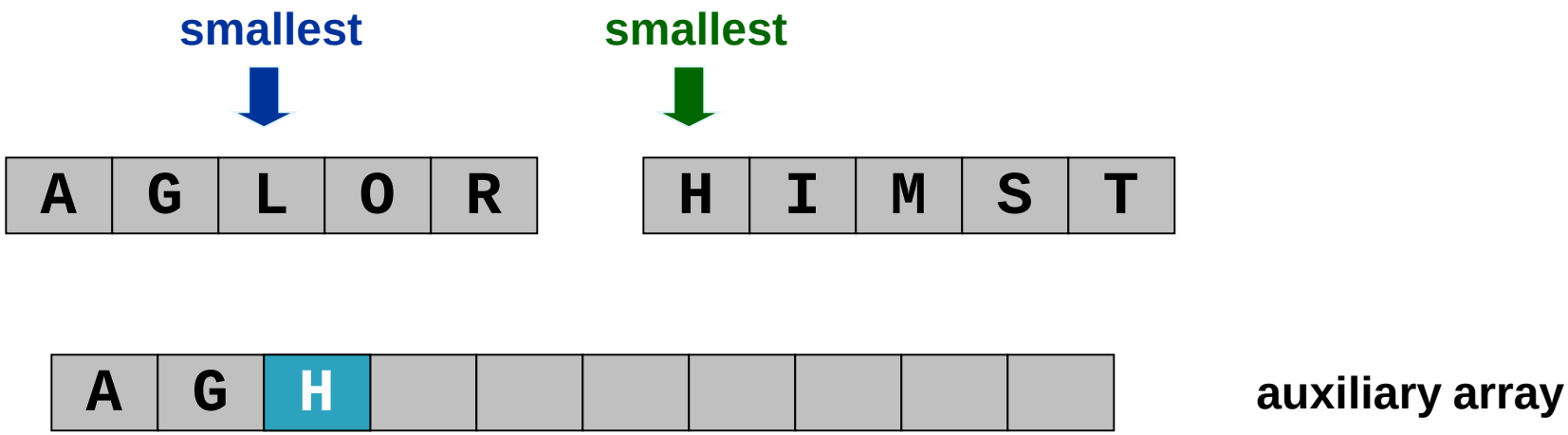
Merging

- Merge.
  - Keep track of smallest element in each sorted half.
  - Insert smallest of two elements into auxiliary array.
  - Repeat until done.



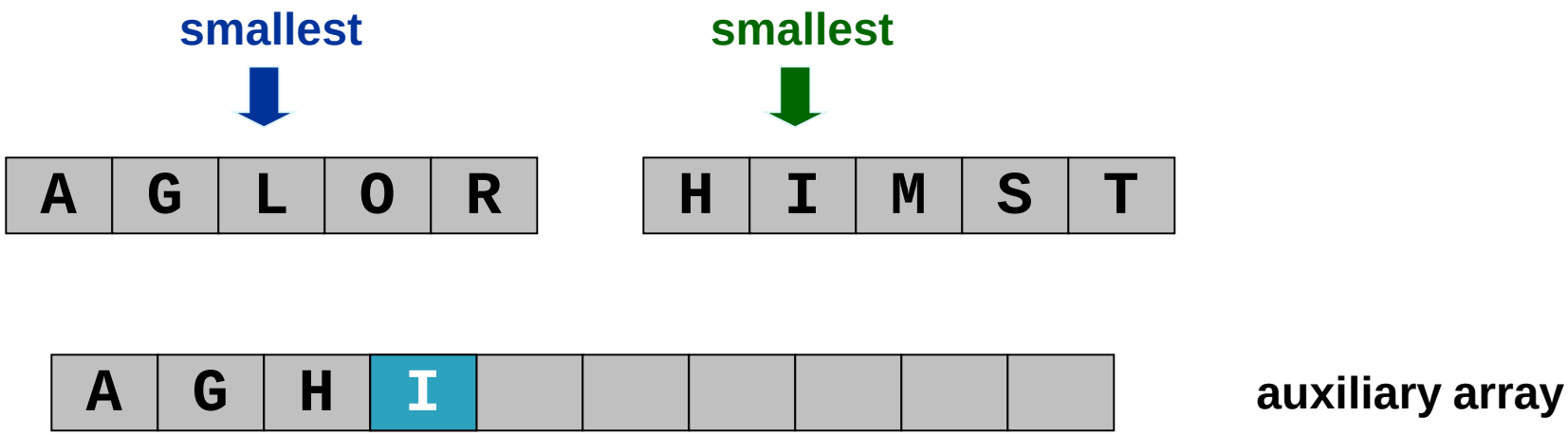
Merging

- Merge.
  - Keep track of smallest element in each sorted half.
  - Insert smallest of two elements into auxiliary array.
  - Repeat until done.



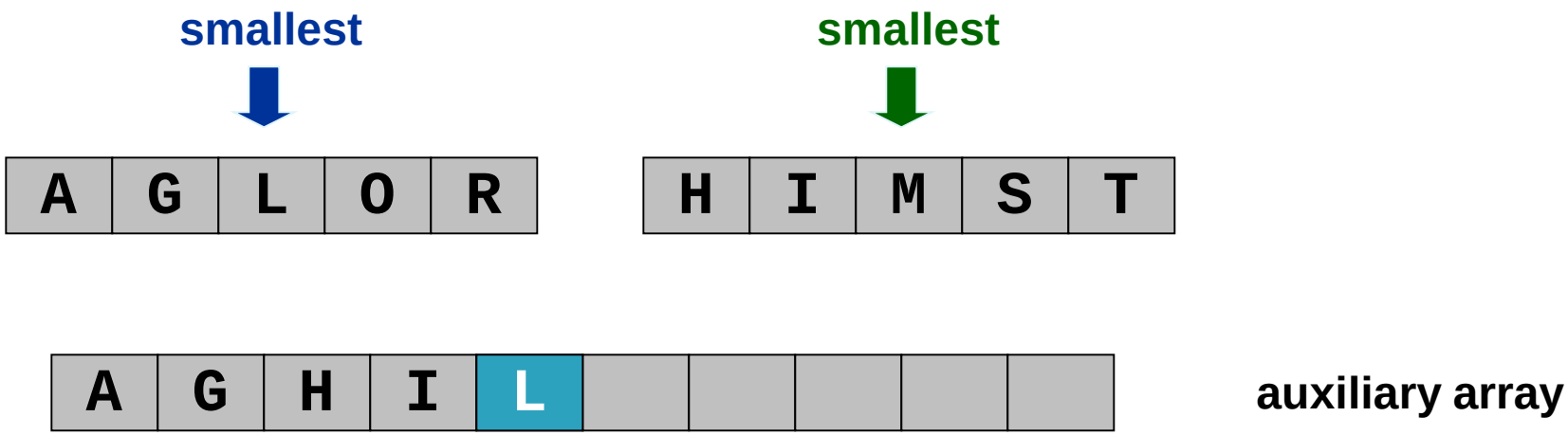
Merging

- Merge.
  - Keep track of smallest element in each sorted half.
  - Insert smallest of two elements into auxiliary array.
  - Repeat until done.



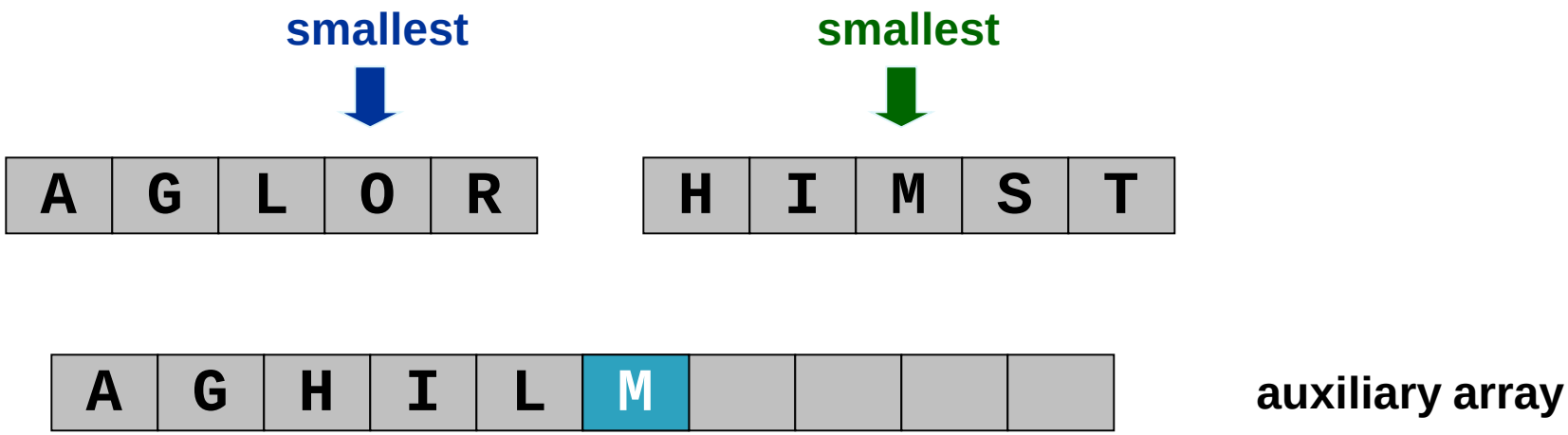
Merging

- Merge.
  - Keep track of smallest element in each sorted half.
  - Insert smallest of two elements into auxiliary array.
  - Repeat until done.



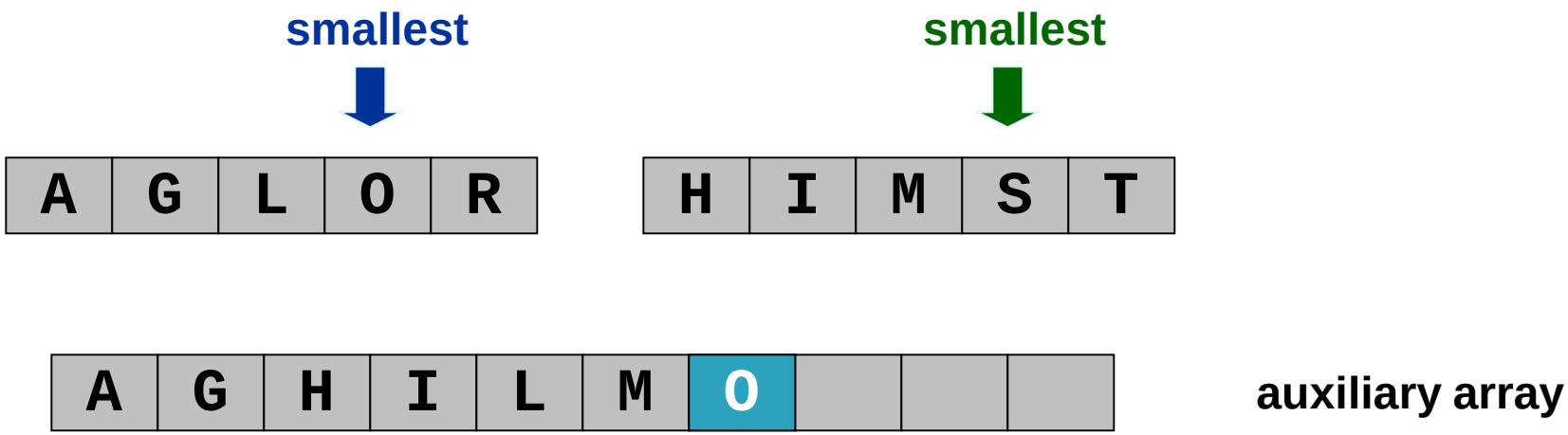
Merging

- Merge.
  - Keep track of smallest element in each sorted half.
  - Insert smallest of two elements into auxiliary array.
  - Repeat until done.



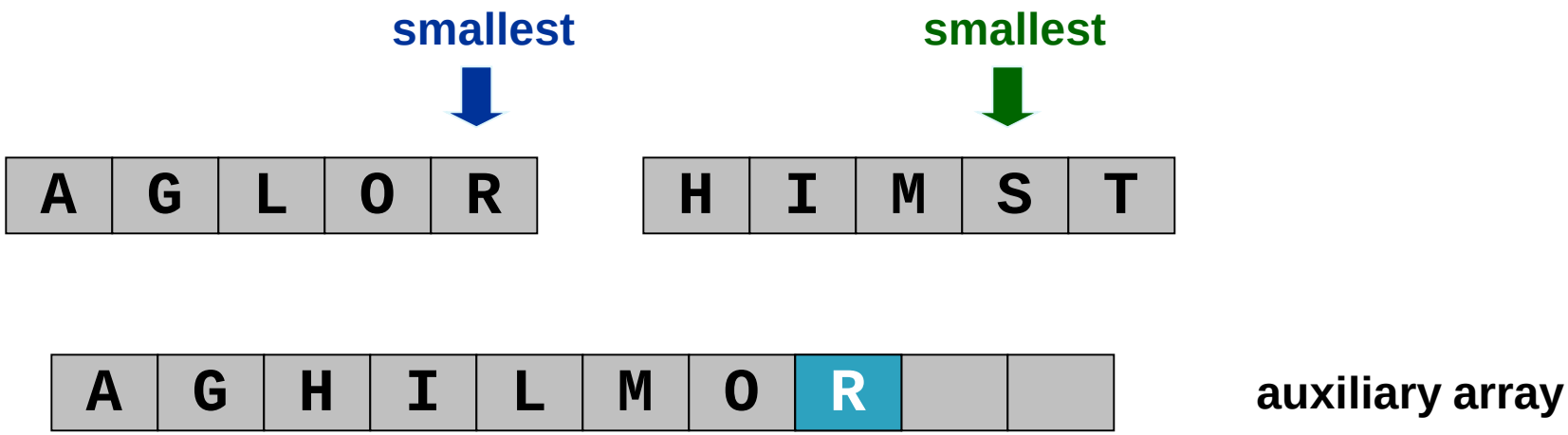
Merging

- Merge.
  - Keep track of smallest element in each sorted half.
  - Insert smallest of two elements into auxiliary array.
  - Repeat until done.



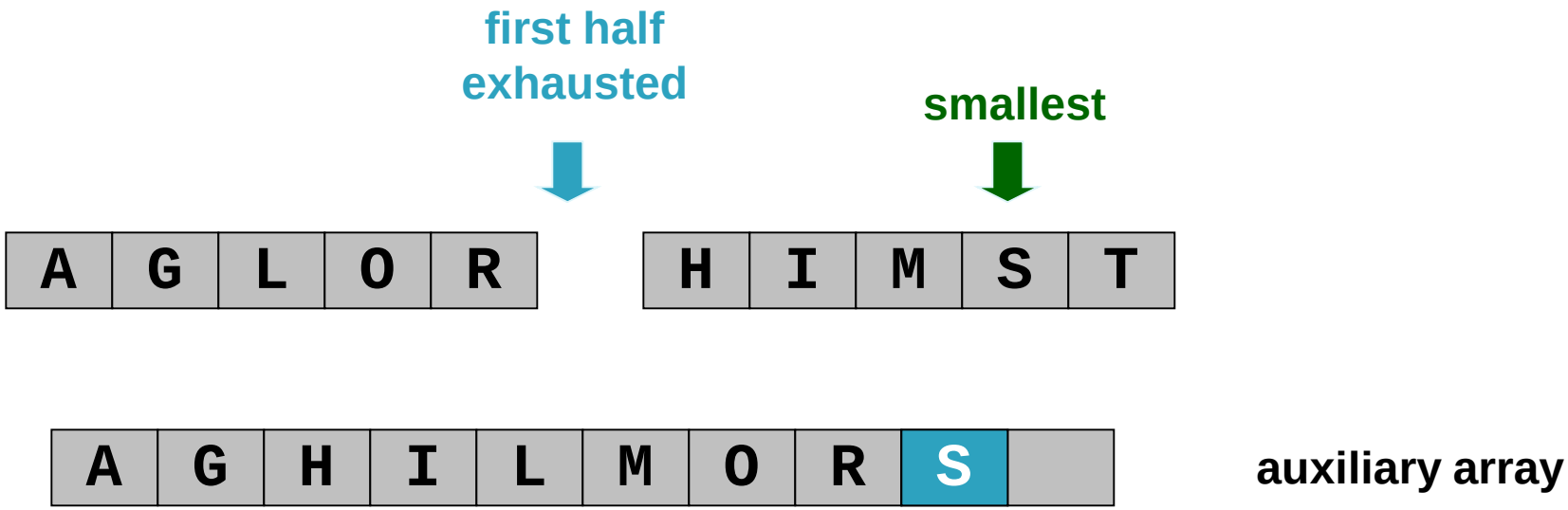
Merging

- Merge.
  - Keep track of smallest element in each sorted half.
  - Insert smallest of two elements into auxiliary array.
  - Repeat until done.



Merging

- Merge.
  - Keep track of smallest element in each sorted half.
  - Insert smallest of two elements into auxiliary array.
  - Repeat until done.



Merging

- Merge.
  - Keep track of smallest element in each sorted half.
  - Insert smallest of two elements into auxiliary array.
  - Repeat until done.