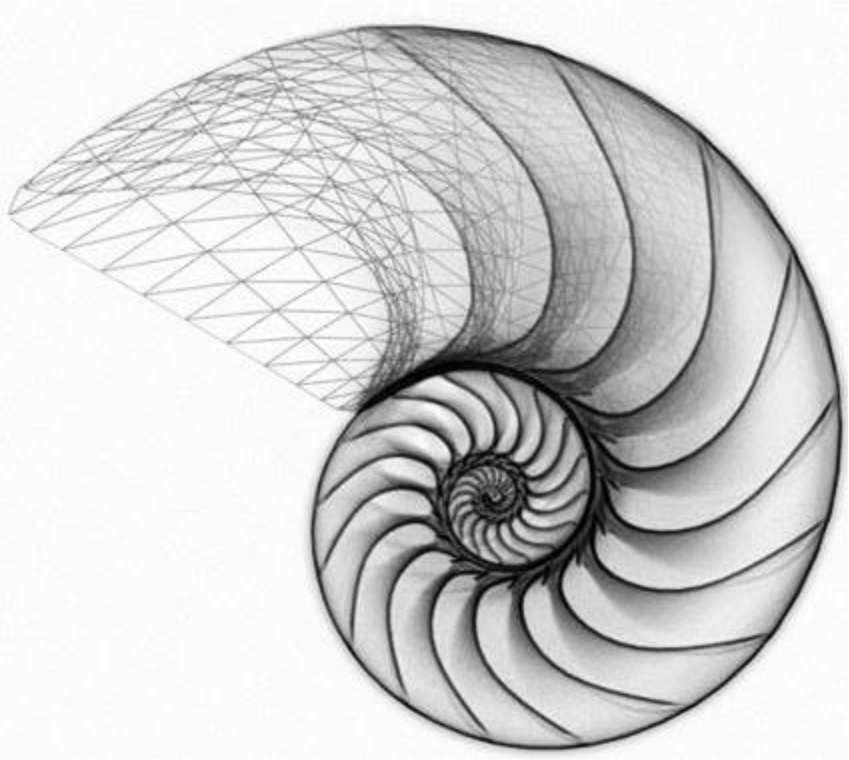


Lecture 18

Sorting

CSE225: Data Structures and Algorithms

Bubble Sort

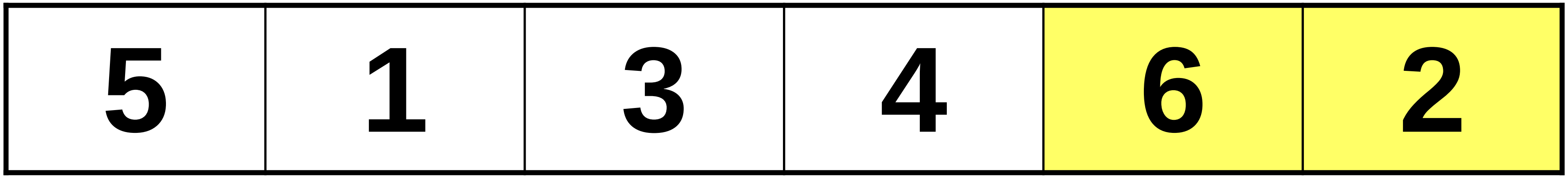
5	1	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



5	1	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort

5	1	3	4	2	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort

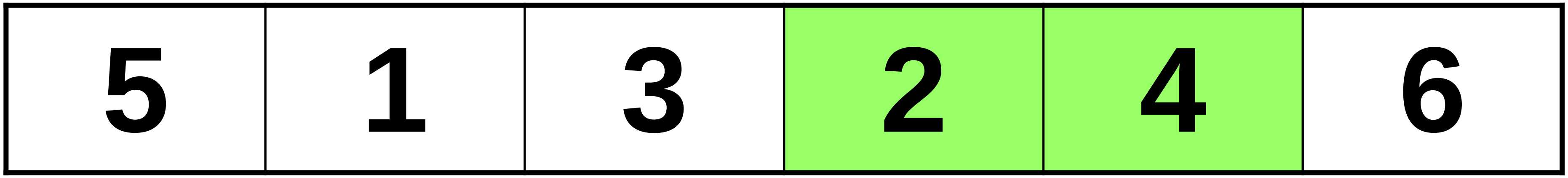
5	1	3	4	2	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



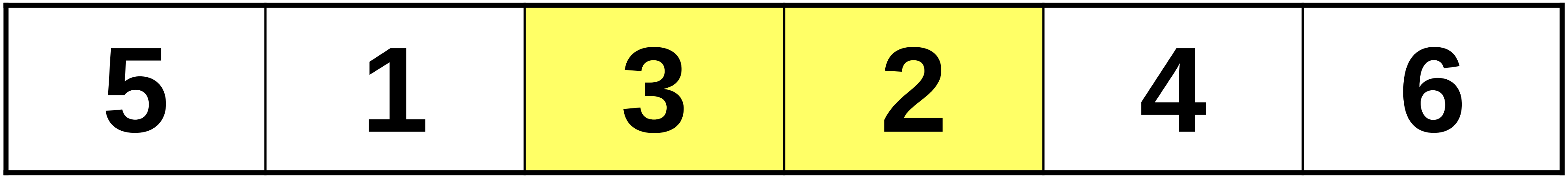
5	1	3	2	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



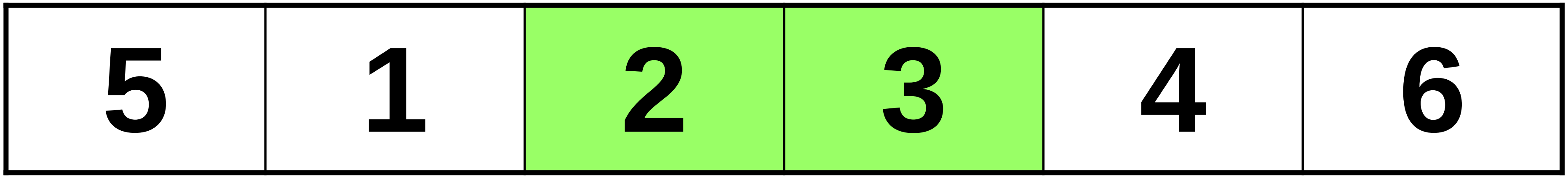
5	1	3	2	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



5	1	2	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort

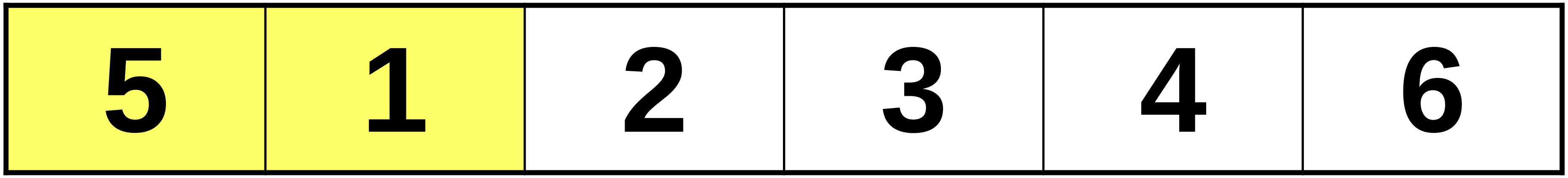
5	1	2	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



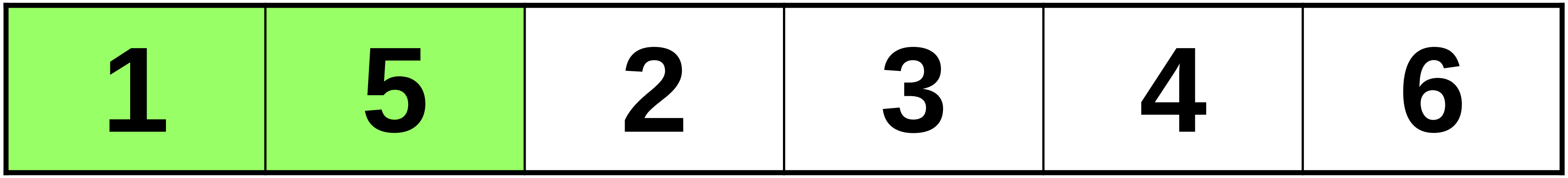
5	1	2	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



1	5	2	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort

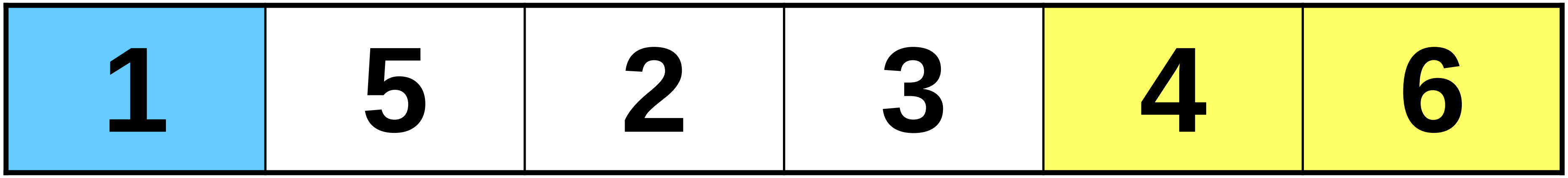
1	5	2	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



1	5	2	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort

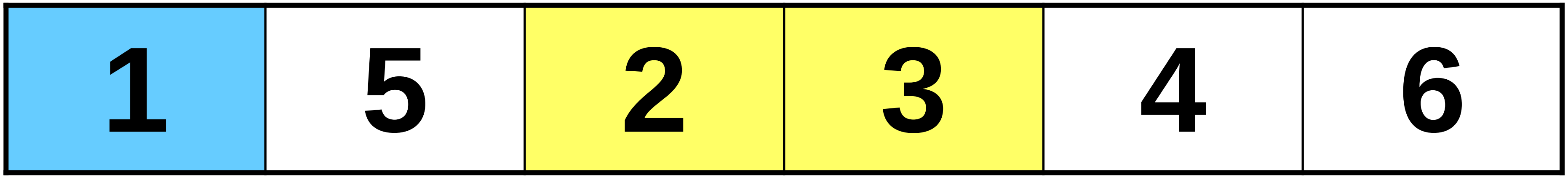
1	5	2	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



1	5	2	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort

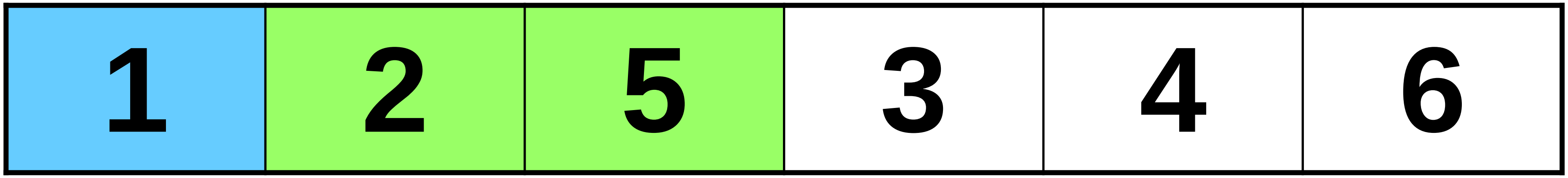
1	5	2	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



1	2	5	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort

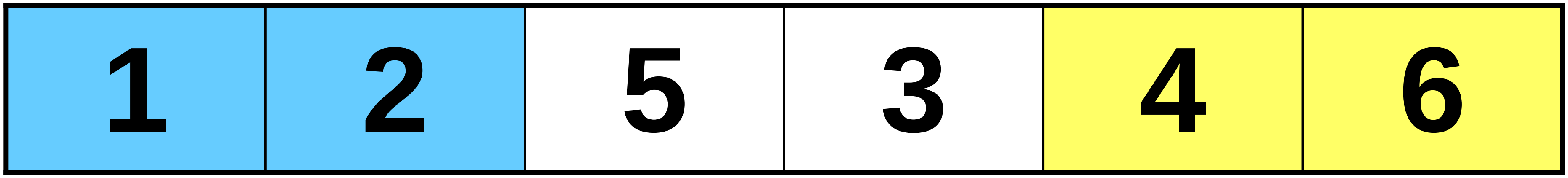
1	2	5	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



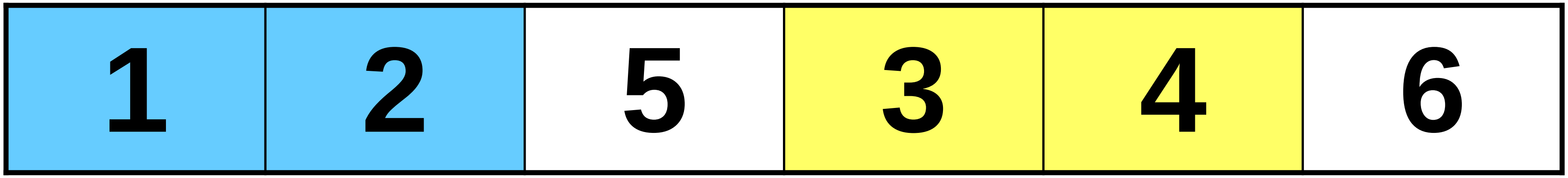
1	2	5	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



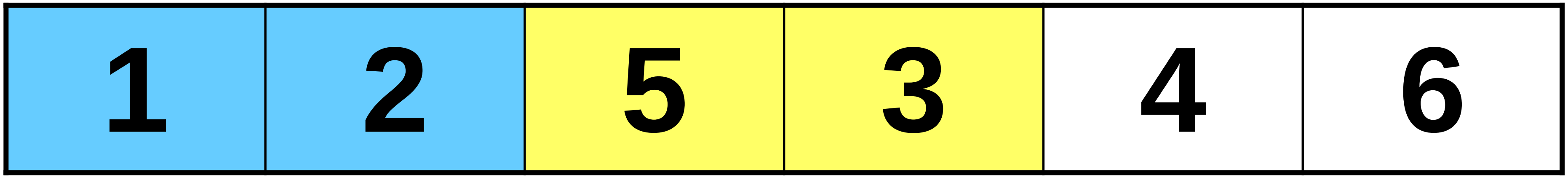
1	2	5	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



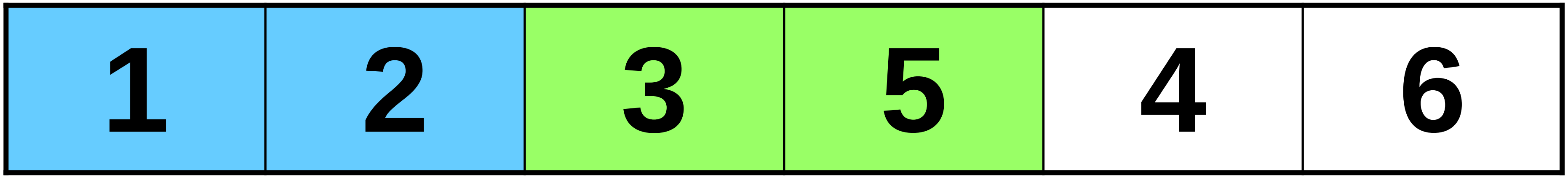
1	2	5	3	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



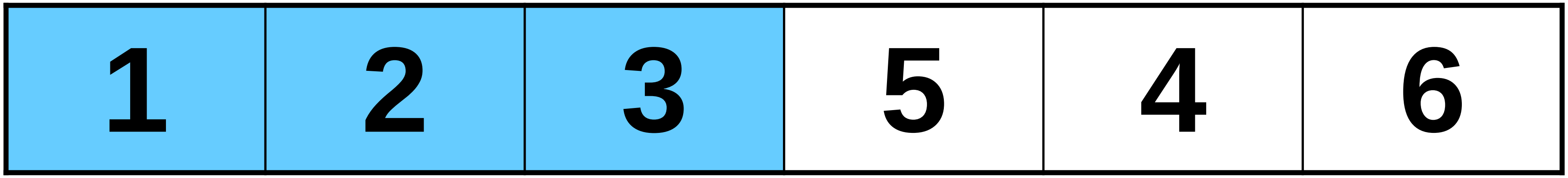
1	2	3	5	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



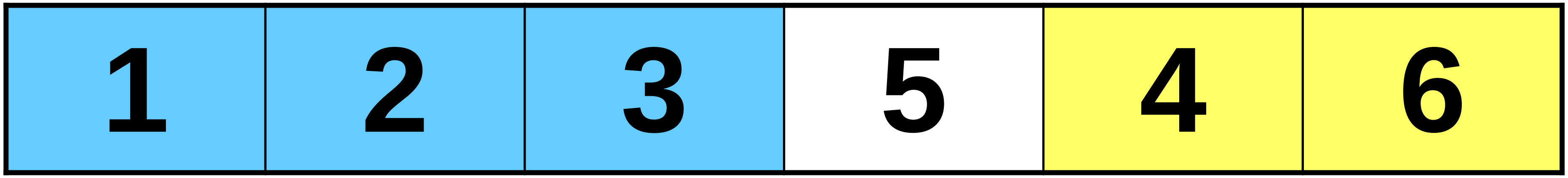
1	2	3	5	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



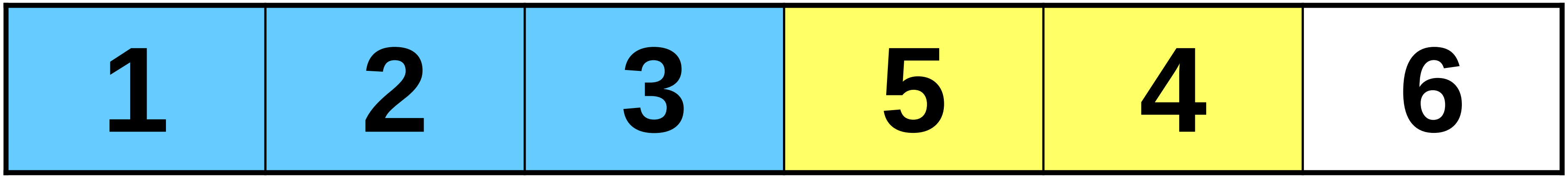
1	2	3	5	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



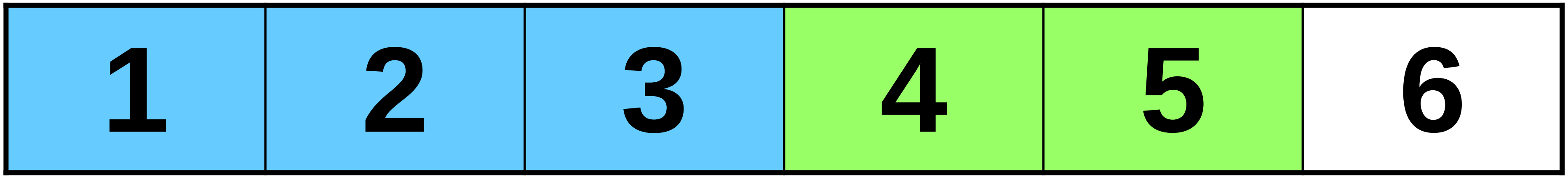
1	2	3	5	4	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort

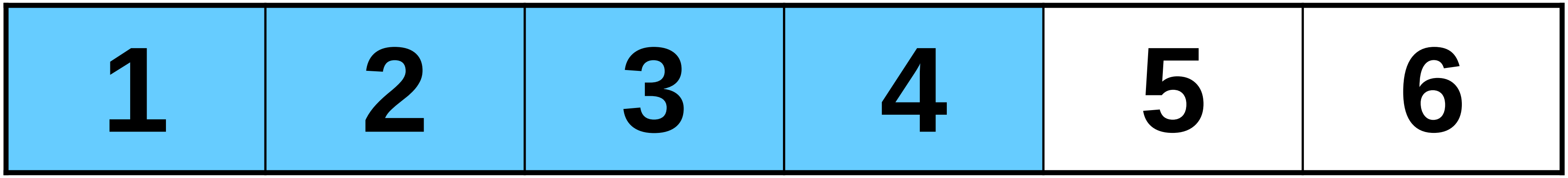


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



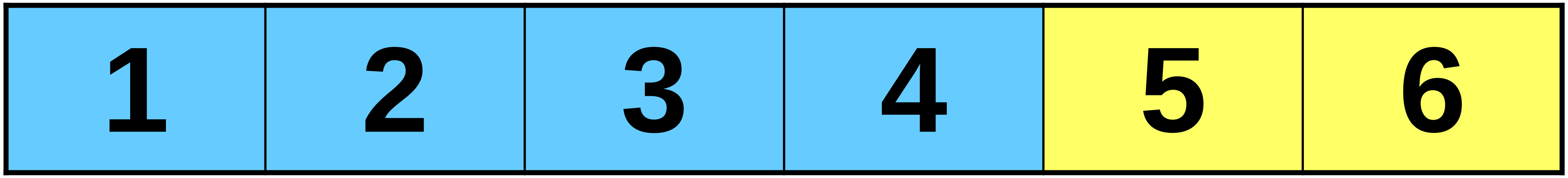
1	2	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort



🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort

1	2	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Bubble Sort

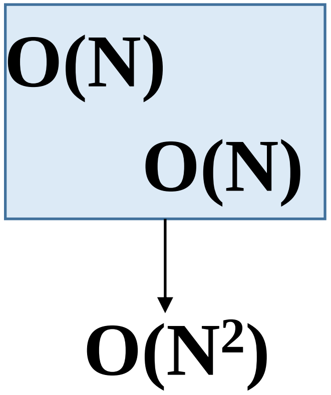
```
template<class ItemType>
void Swap(ItemType& item1, ItemType& item2)
{
        ItemType tempItem;

        tempItem = item1;
        item1 = item2;
        item2 = tempItem;
}
```

Bubble Sort

```
template<class ItemType>
void BubbleUp(ItemType values[] , int startIndex, int endIndex)
{
        for (int index = endIndex; index > startIndex; index--)
                if (values[index-1] > values[index])
                        Swap(values[index-1], values[index]);
}

template<class ItemType>
void BubbleSort(ItemType values[], int numValues)
{
        int current = 0;

        while (current < numValues - 1)
        {
                BubbleUp(values, current, numValues-1);
                current++;
        }
}
```

Bubble Sort

```
template<class ItemType>
void BubbleUp(ItemType values[] , int startIndex, int endIndex)
{
        for (int index = endIndex; index > startIndex; index--)
                if (values[index-1] > values[index])
                        Swap(values[index-1], values[index]);
}

template<class ItemType>
void BubbleSort(ItemType values[], int numValues)
{
        int current = 0;

        while (current < numValues - 1)
        {
                BubbleUp(values, current, numValues-1);
                current++;
        }
}
```



O(N)

O(N)

$O(N^2)$

Bubble Sort (little improved)

```
template<class ItemType>
void BubbleUp2(ItemType values[], int startIndex, int endIndex, bool&
sorted)
{
        sorted = true;
        for (int index = endIndex; index > startIndex; index--)
                if (values[index] < values[index-1])
                {
                        Swap(values[index], values[index-1]);
                        sorted = false;
                }
}
template<class ItemType>
void ShortBubble(ItemType values[], int numValues)
{
        int current = 0;
        bool sorted = false;
        while (current < numValues - 1 && !sorted)
        {
                BubbleUp2(values, current, numValues-1, sorted);
                current++;
        }
}
```

Selection Sort

5	1	3	4	6	2

🟨	Comparison
🟩	Data Movement
🟦	Sorted

Selection Sort

5	**1**	**3**	**4**	**6**	**2**

↑

Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

5	1	3	4	6	2

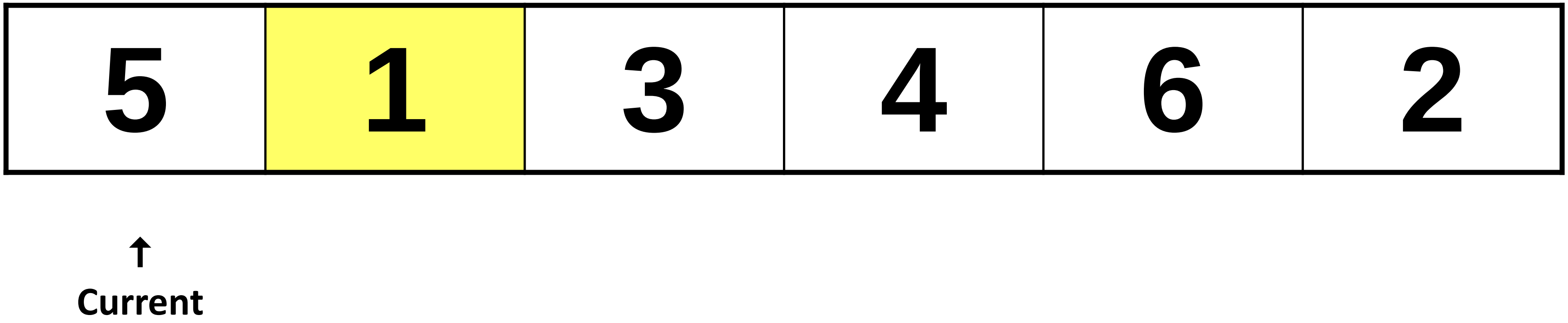
↑
Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



5	1	3	4	6	2

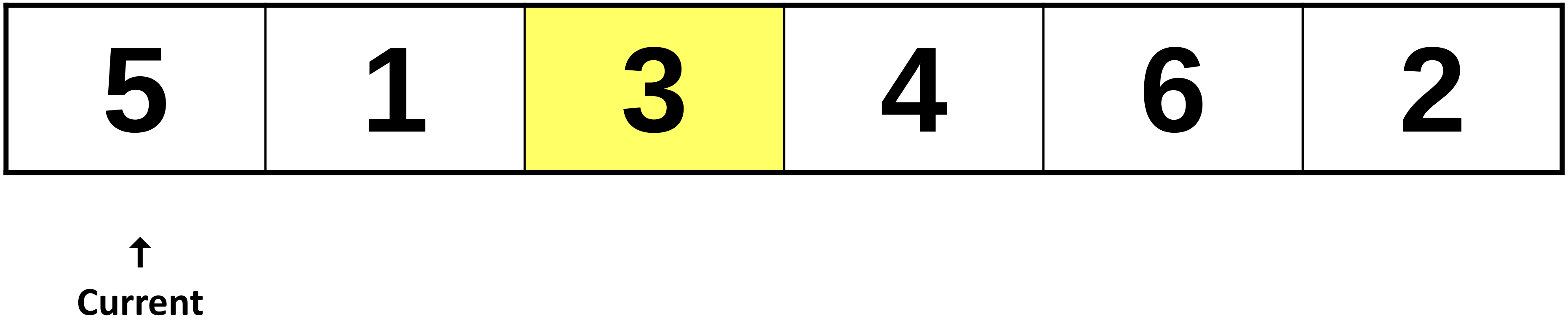
↑
Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



5	1	3	4	6	2

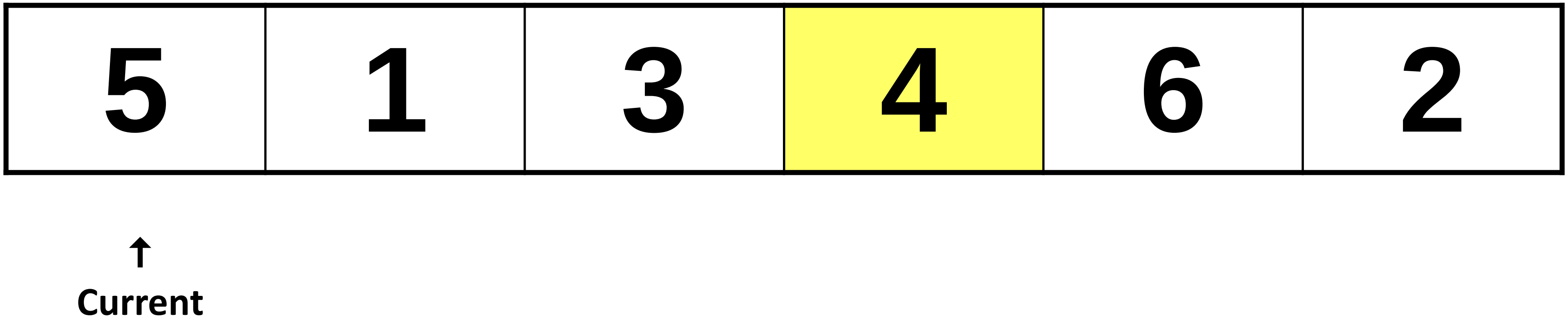
↑
Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



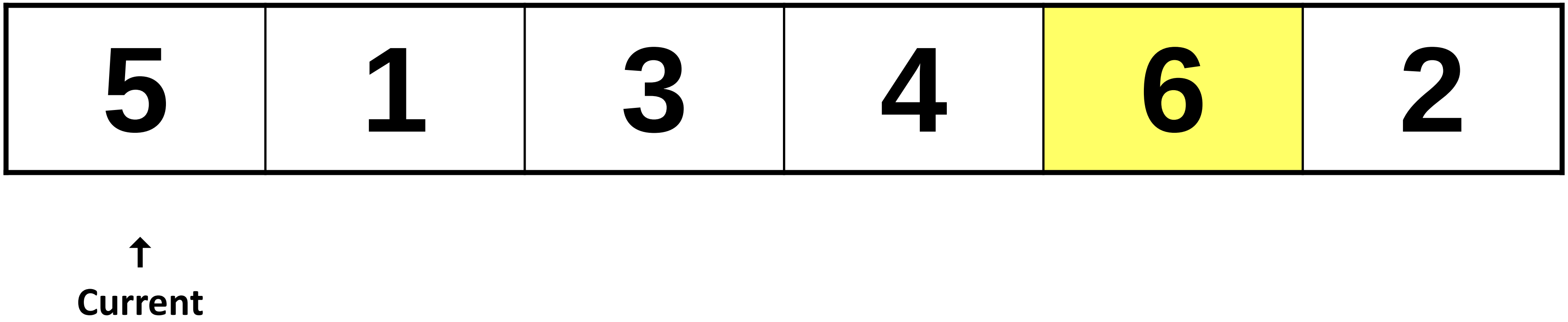
5	1	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



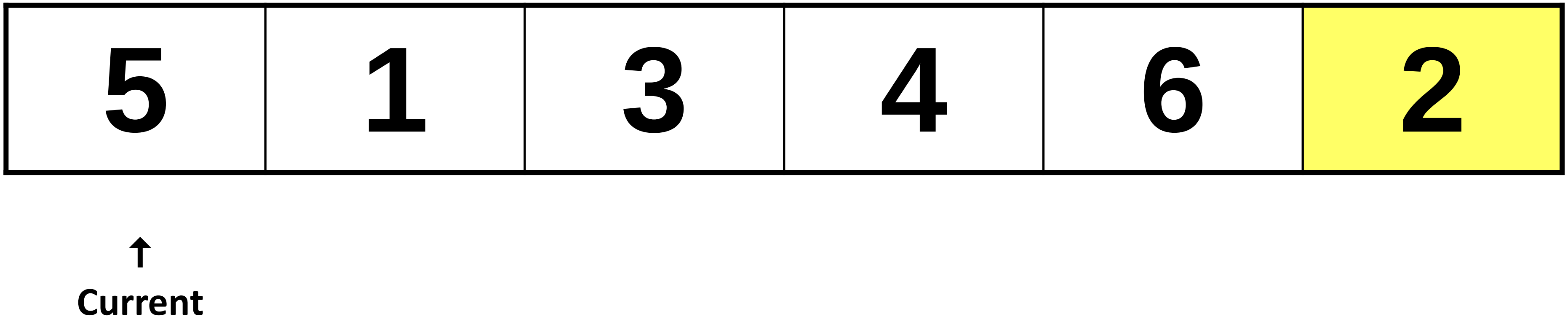
5	1	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



5	1	3	4	6	2

🟨 Comparison

🟩 Data Movement

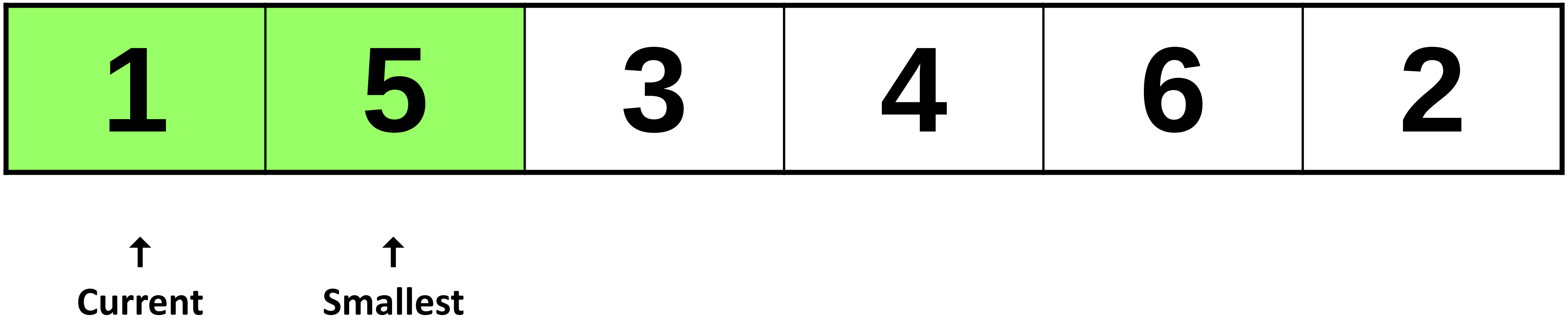
🟦 Sorted

Selection Sort

5	**1**	**3**	**4**	**6**	**2**
↑ **Current**	↑ **Smallest**				

🟨	Comparison
🟩	Data Movement
🟦	Sorted

Selection Sort



1	5	3	4	6	2
↑	↑				
Current	Smallest				

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

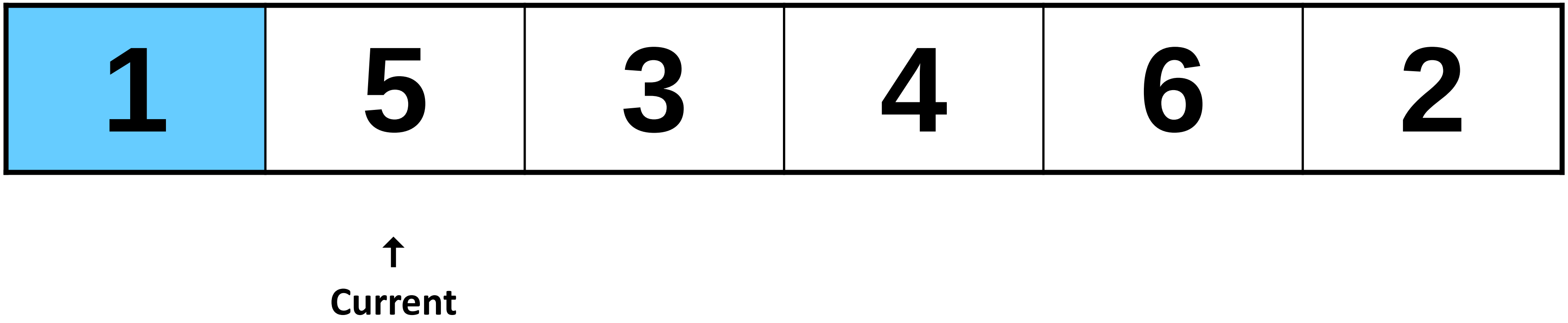
1	5	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

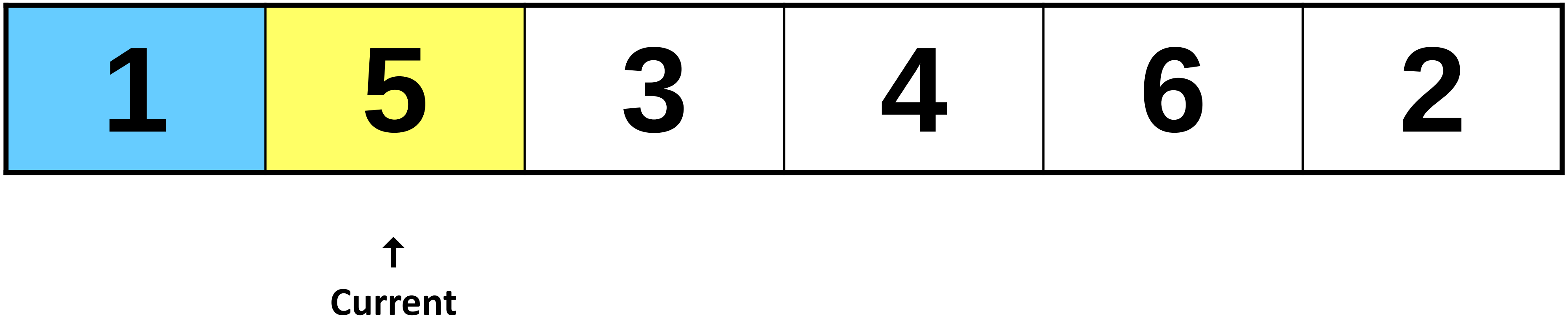


1	5	3	4	6	2

↑
Current

- 🟨 Comparison
- 🟩 Data Movement
- 🟦 Sorted

Selection Sort



1	5	3	4	6	2

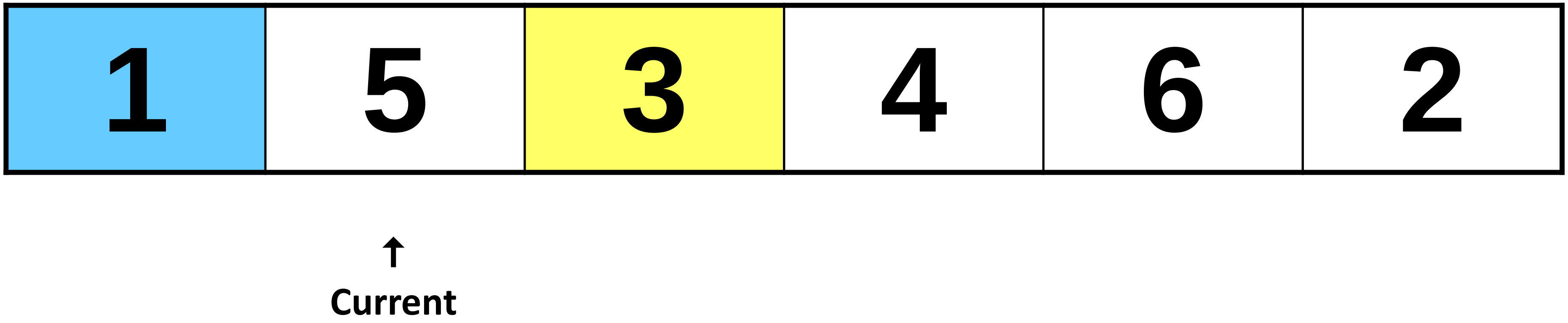
↑
Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



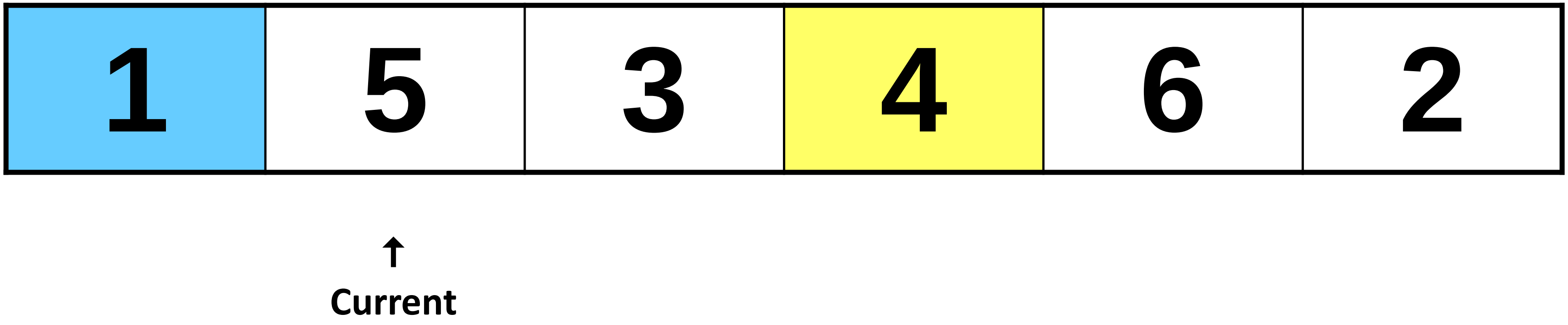
1	5	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	5	3	4	6	2

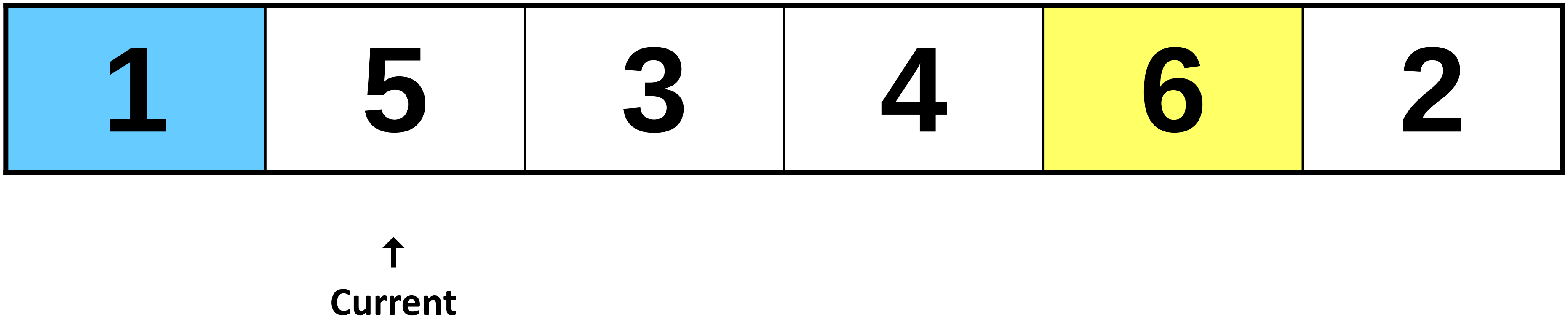
↑ Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



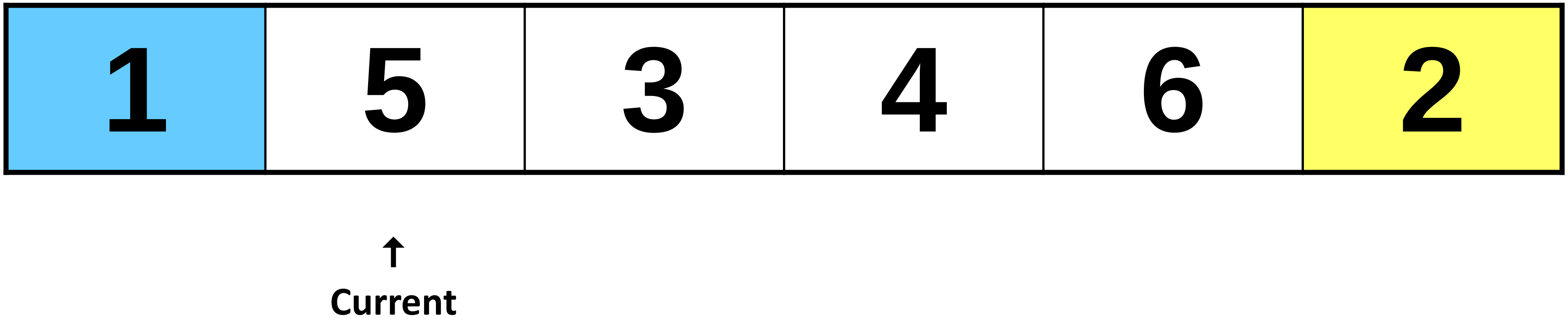
1	5	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	5	3	4	6	2

↑

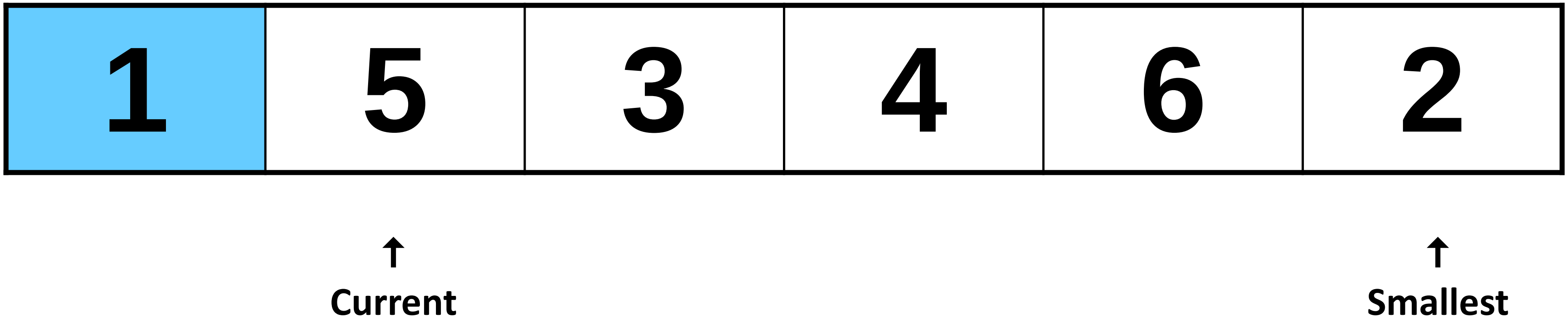
Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



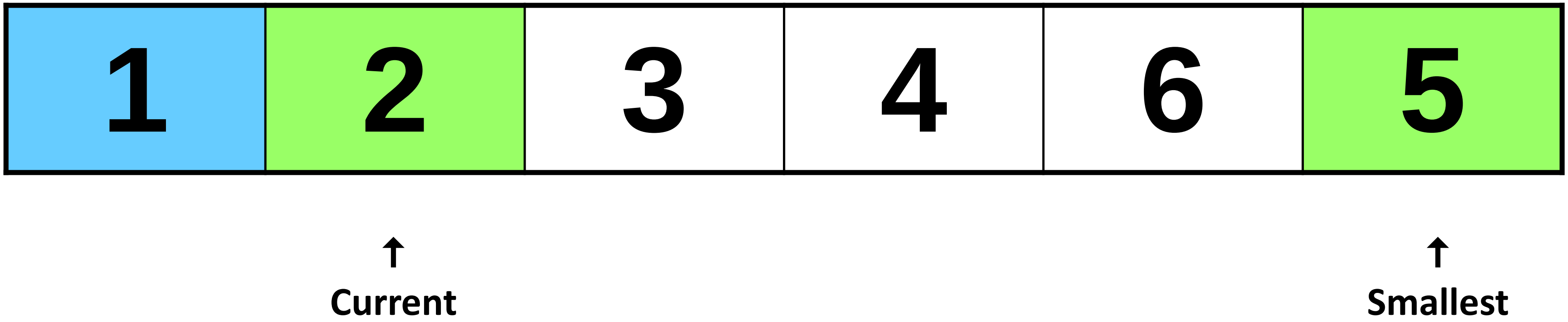
1	5	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

↑ Current

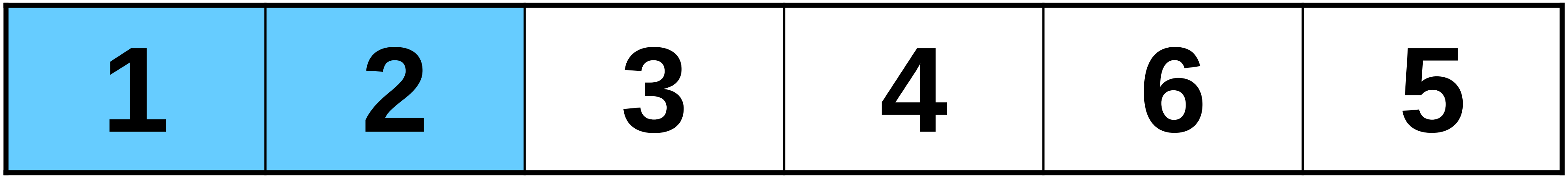
↑ Smallest

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

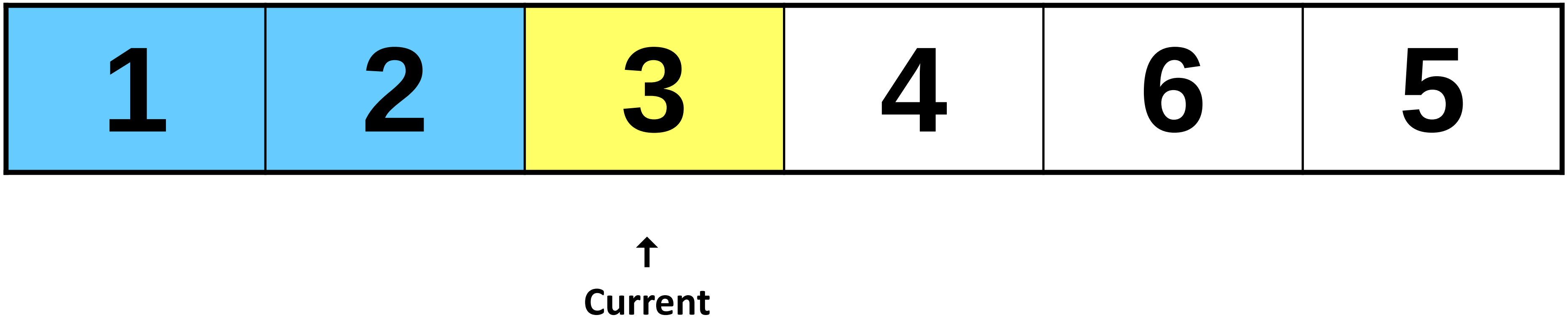
1	2	3	4	6	5
		↑ **Current**			

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

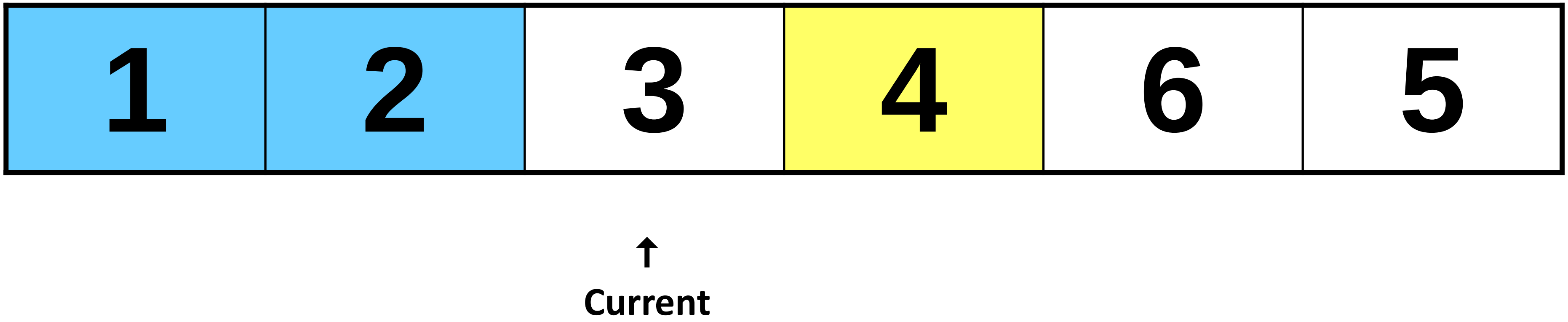
Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

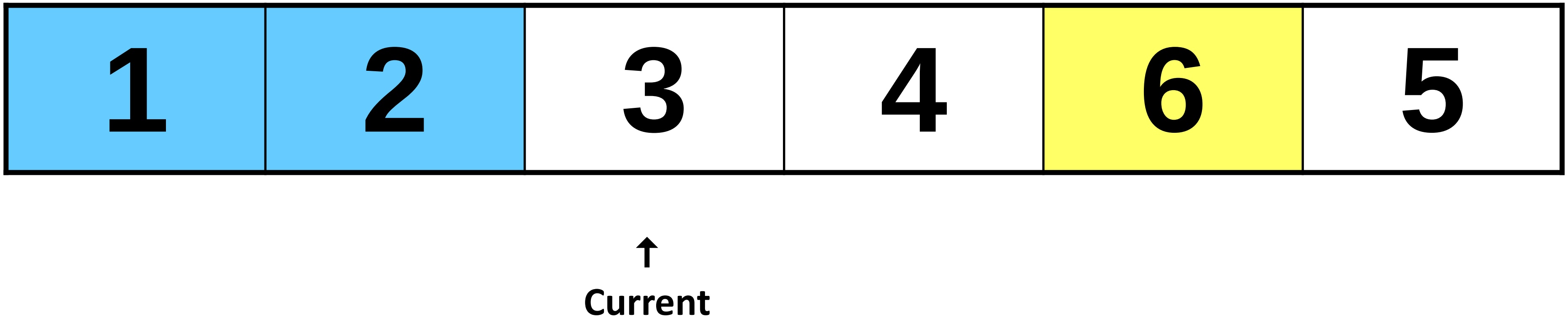
Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

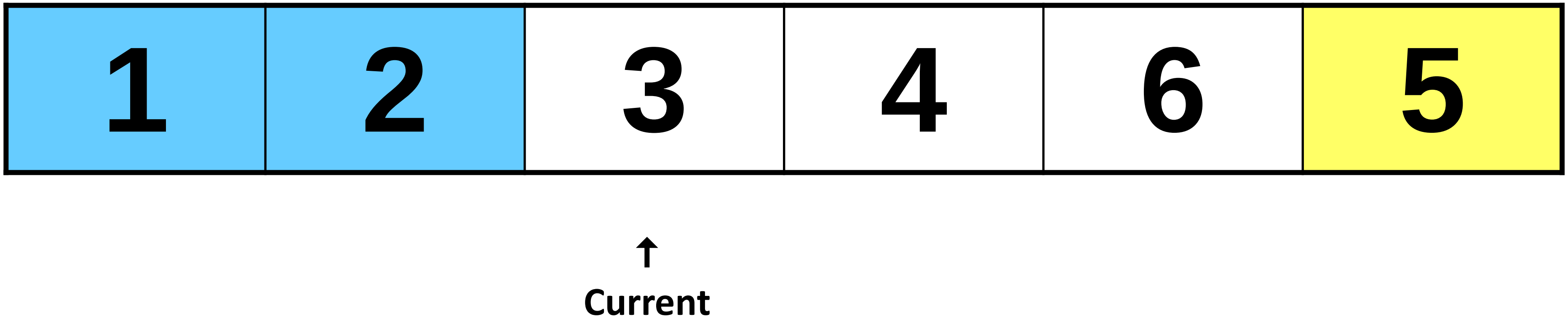
↑
Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

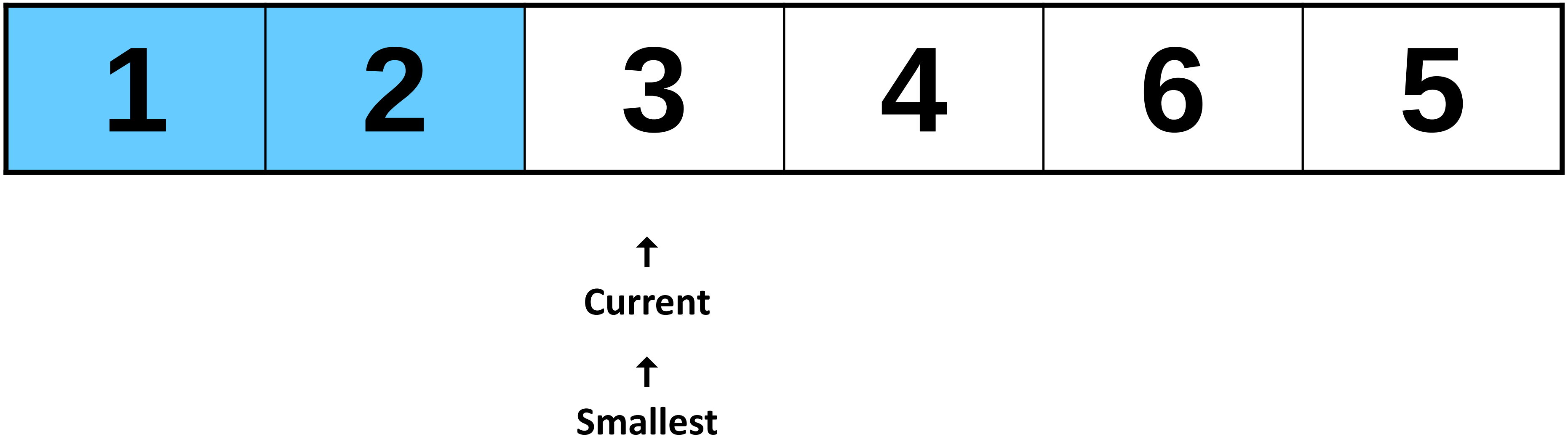


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

↑
Current

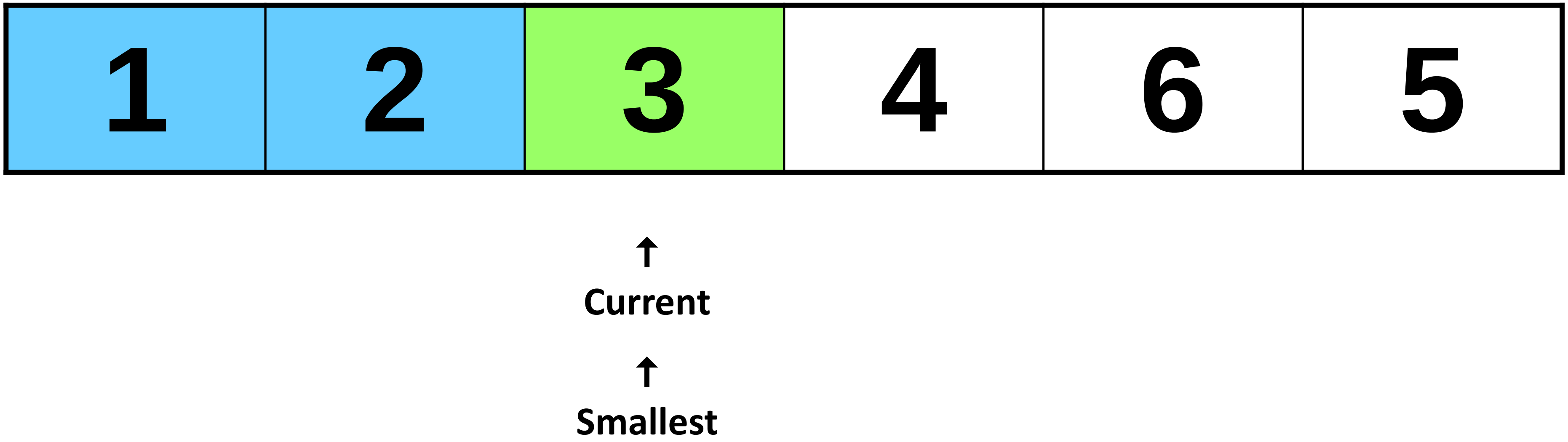
↑
Smallest

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

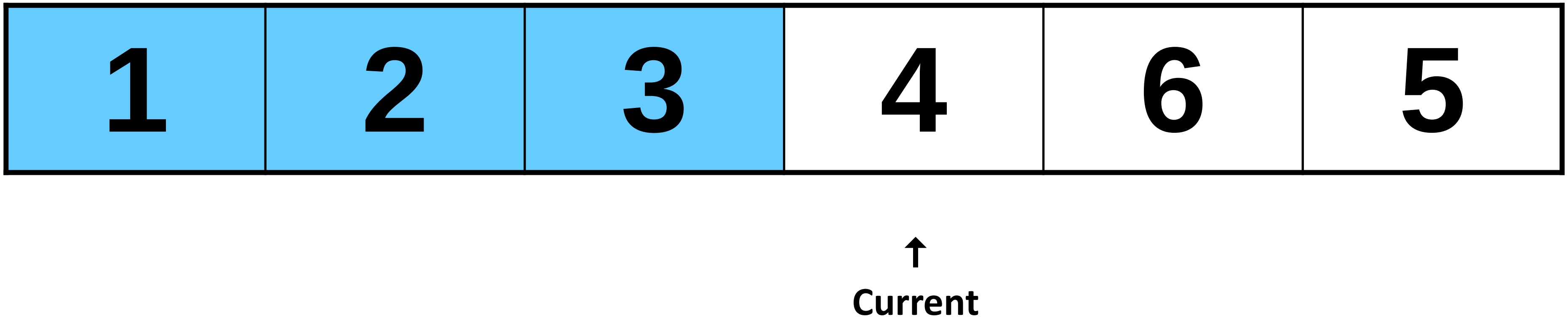
1	2	3	4	6	5

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

↑

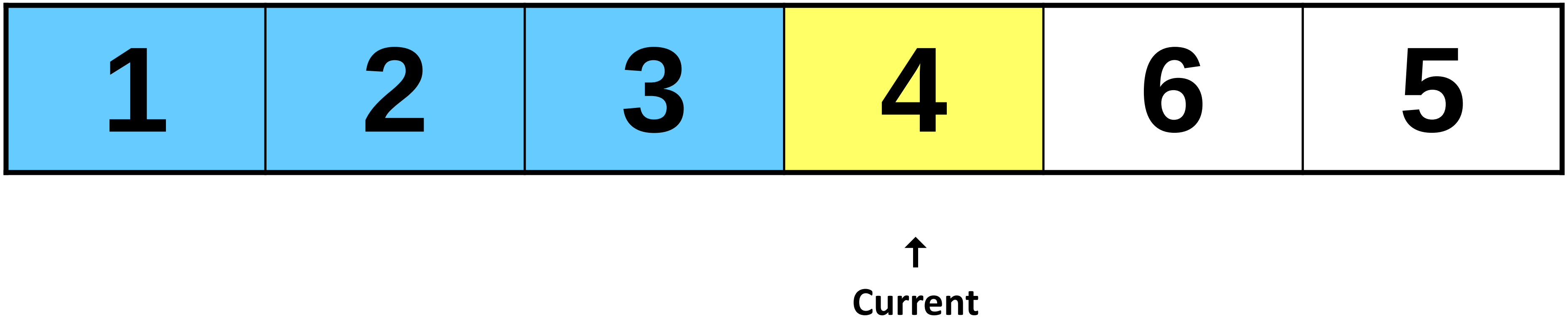
Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

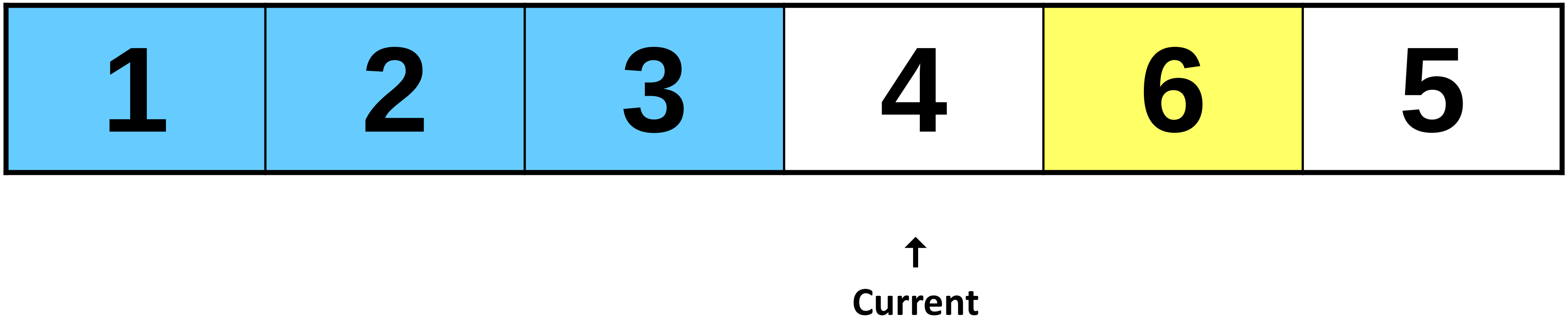
↑ Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

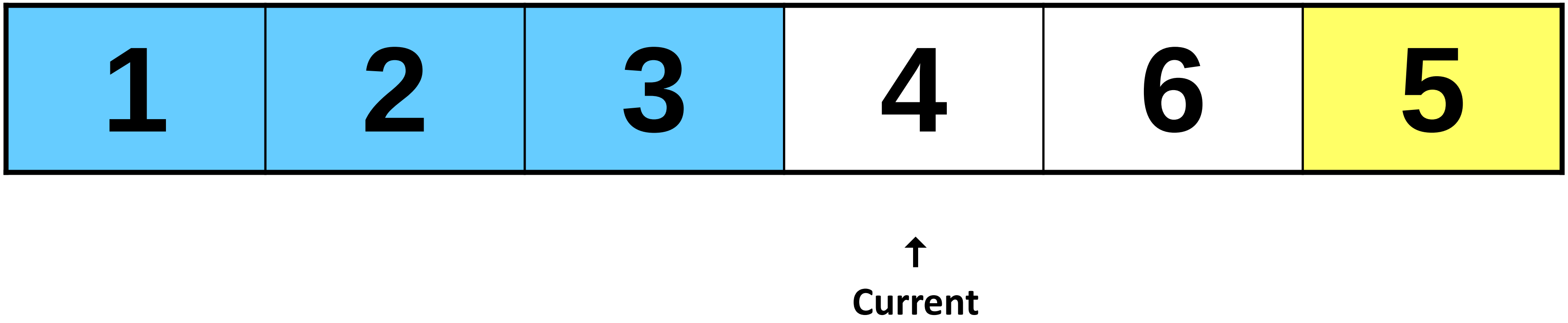
↑
Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



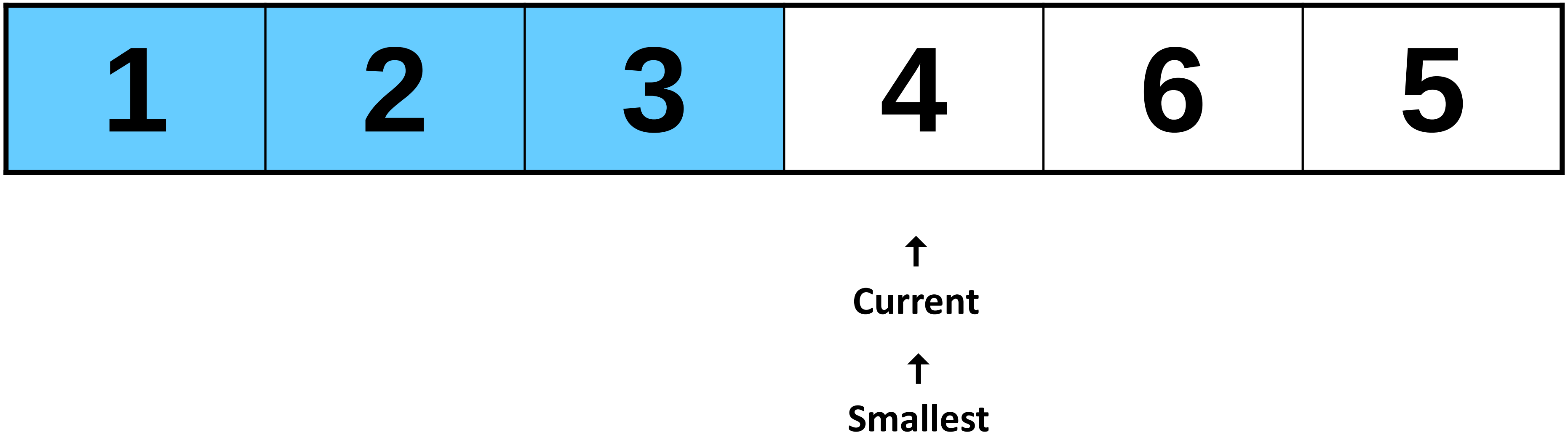
1	2	3	4	6	5

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

↑

Current

↑

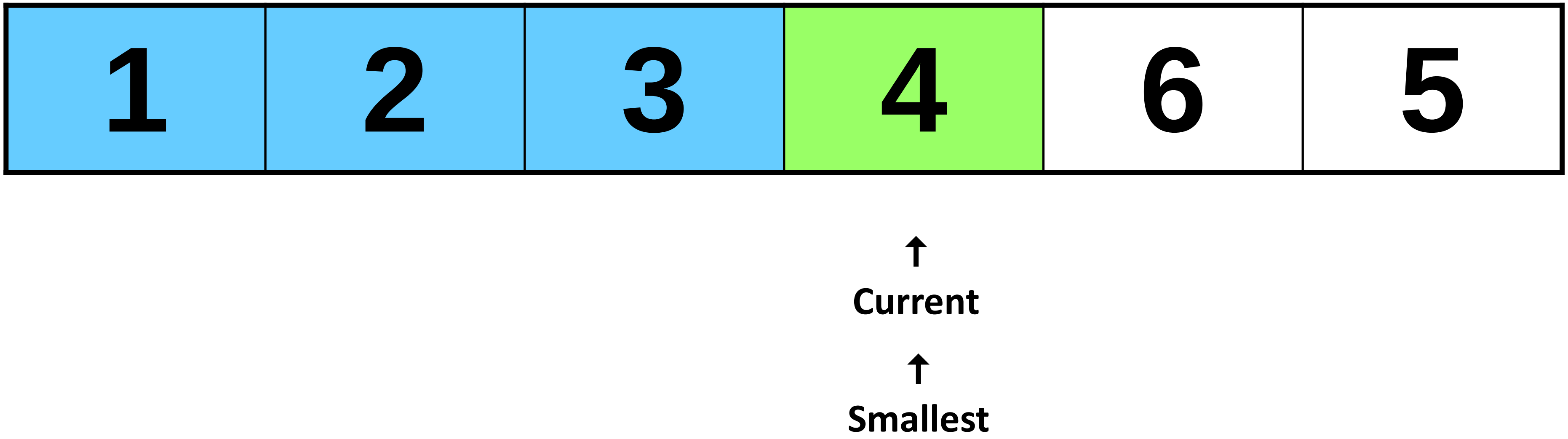
Smallest

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

↑ Current

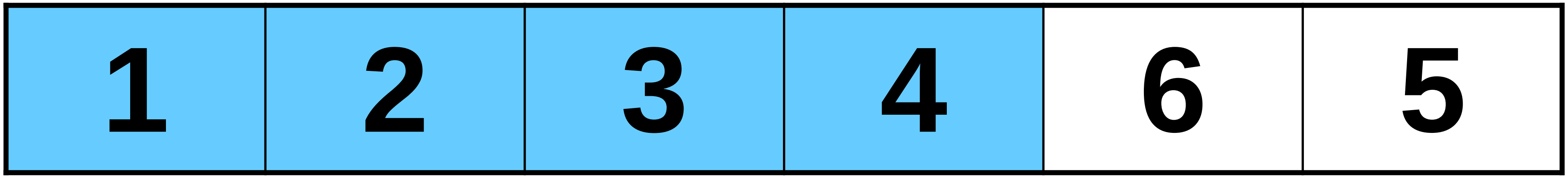
↑ Smallest

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



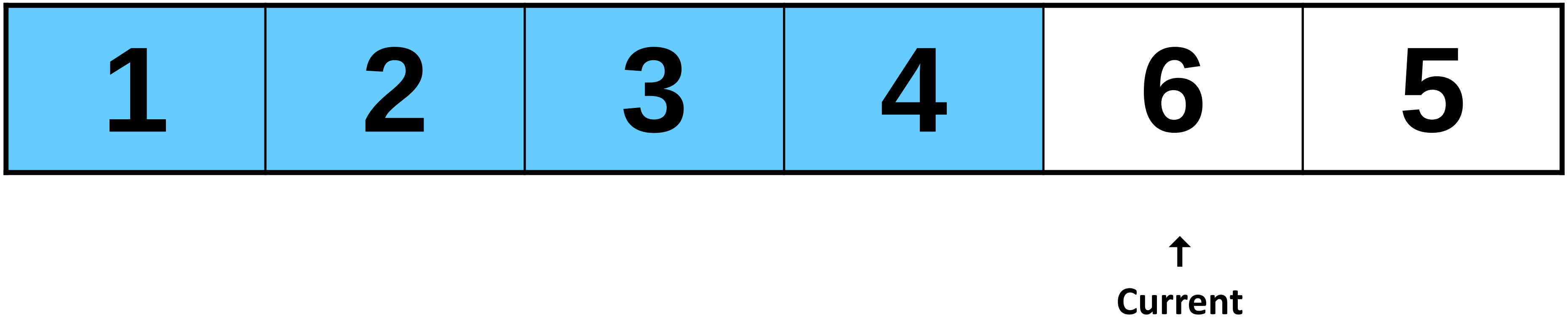
1	2	3	4	6	5

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

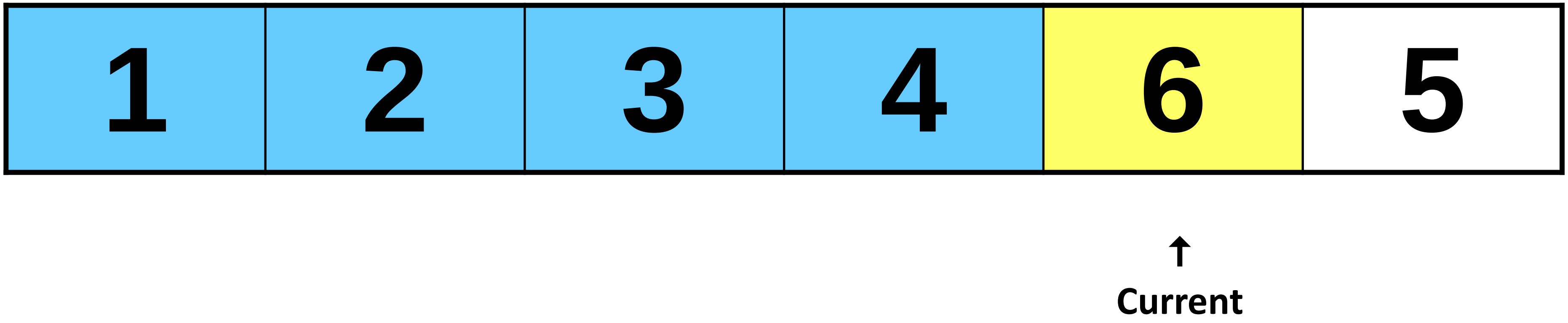
↑ Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



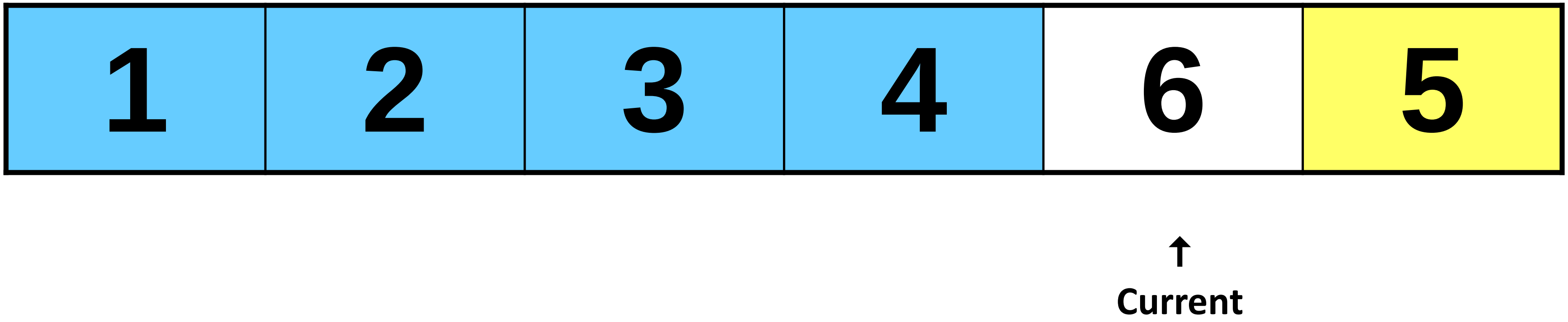
1	2	3	4	6	5

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	6	5

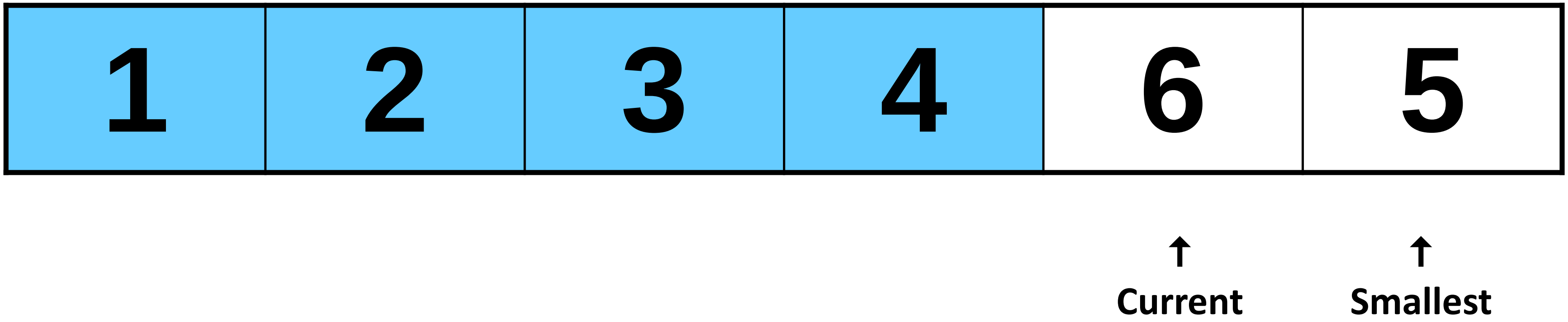
↑
Current

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



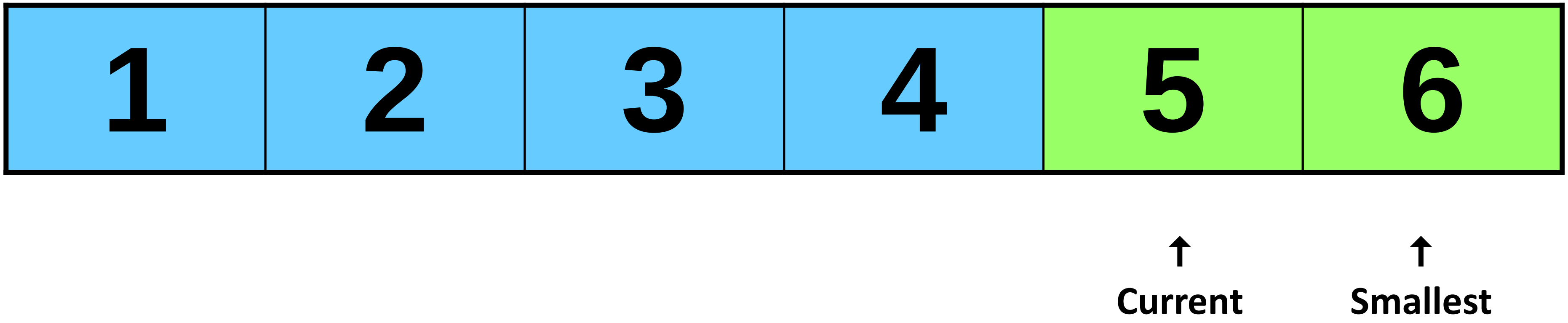
1	2	3	4	6	5

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

1	2	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

```
template<class ItemType>
int getMinIndex(ItemType values[], int startIndex, int endIndex)
{
        int indexOfMin = startIndex;
        for (int index = startIndex + 1; index <= endIndex; index++)
                if (values[index] < values[indexOfMin])
                        indexOfMin = index;
        return indexOfMin;
}
template<class ItemType>
void SelectionSort(ItemType values[], int numValues)
{
        int endIndex = numValues-1;
        int minIndex;
        for (int current = 0; current < endIndex; current++)
        {
                minIndex = getMinIndex(values, current, endIndex);
                Swap(values[current], values[minIndex]);
        }
}
```

Selection Sort

```
template<class ItemType>
int getMinIndex(ItemType values[], int startIndex, int endIndex)
{
        int indexOfMin = startIndex;
        for (int index = startIndex + 1; index <= endIndex; index++)
                if (values[index] < values[indexOfMin])
                        indexOfMin = index;
        return indexOfMin;
}
template<class ItemType>
void SelectionSort(ItemType values[], int numValues)
{
        int endIndex = numValues-1;
        int minIndex;
        for (int current = 0; current < endIndex; current++)
        {
                minIndex = getMinIndex(values, current, endIndex);
                Swap(values[current], values[minIndex]);
        }
}
```

O(N)

O(N)

$O(N^2)$

Insertion Sort

5	**1**	**3**	**4**	**6**	**2**

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort

5	1	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort

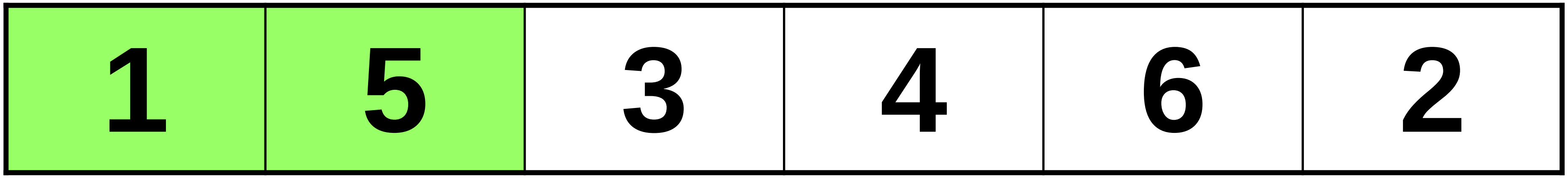
5	1	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



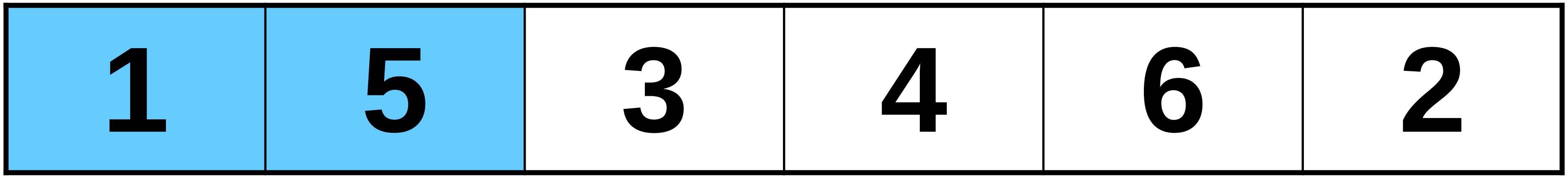
1	5	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



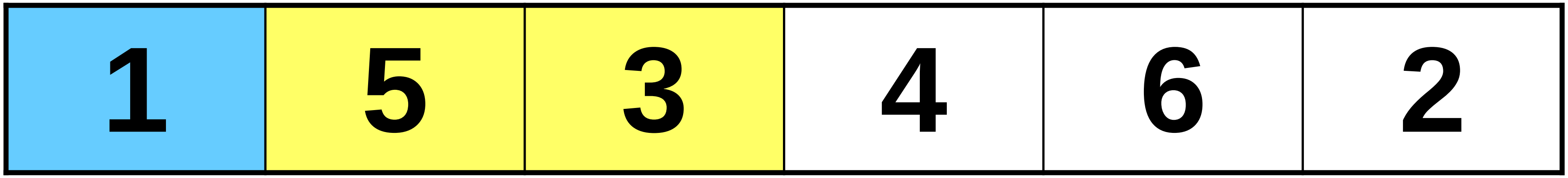
1	5	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



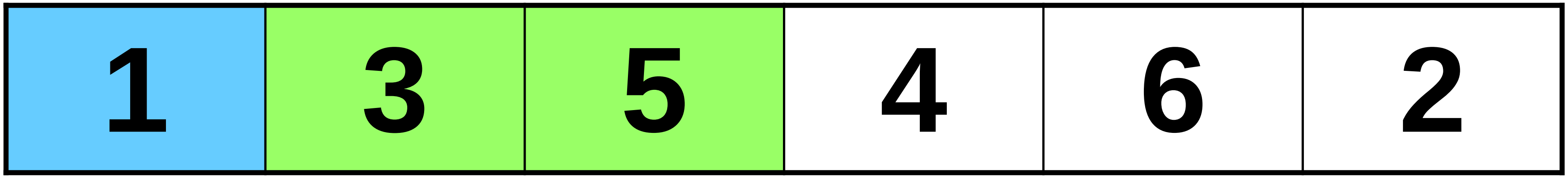
1	5	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



1	3	5	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort

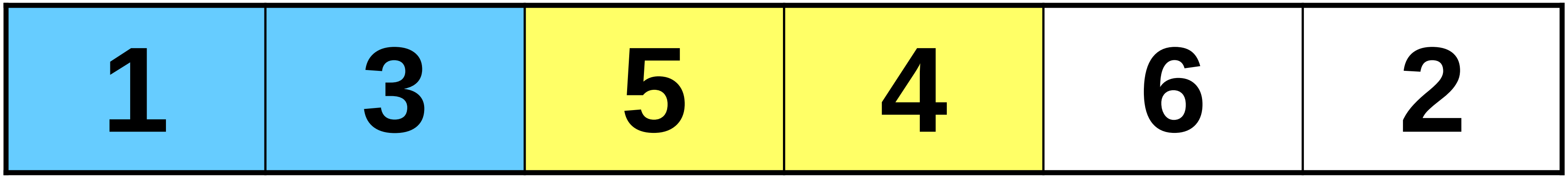
1	3	5	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



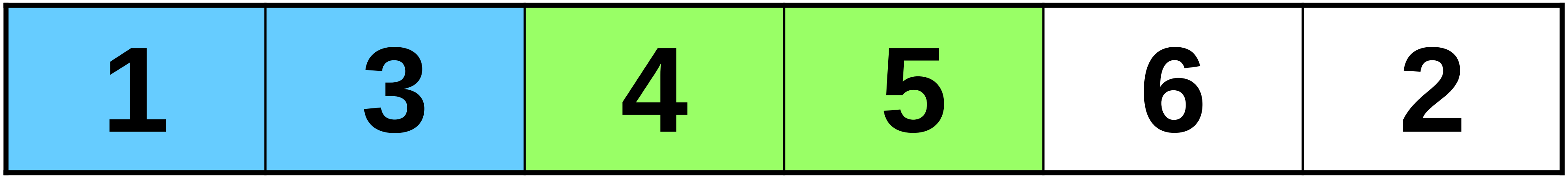
1	3	5	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



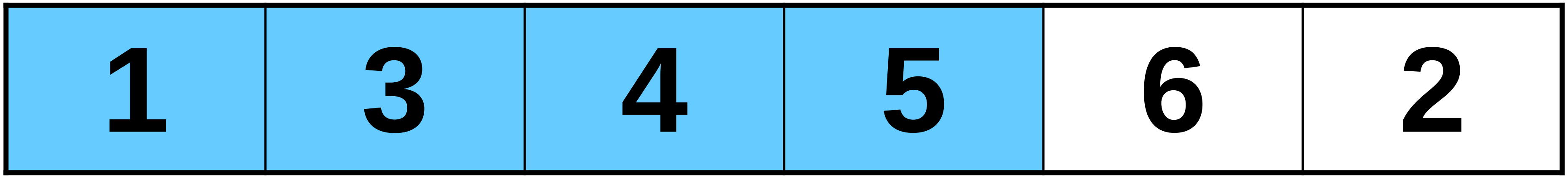
1	3	4	5	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



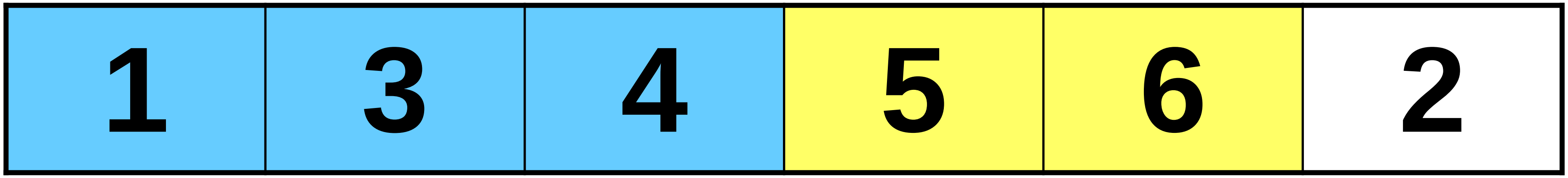
1	3	4	5	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



1	3	4	5	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort

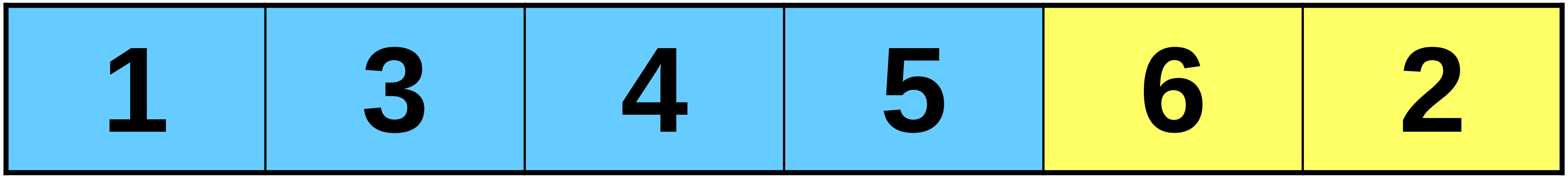
1	3	4	5	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



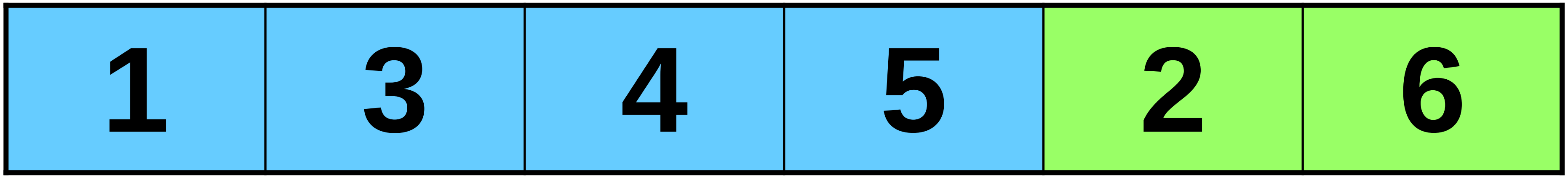
1	3	4	5	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



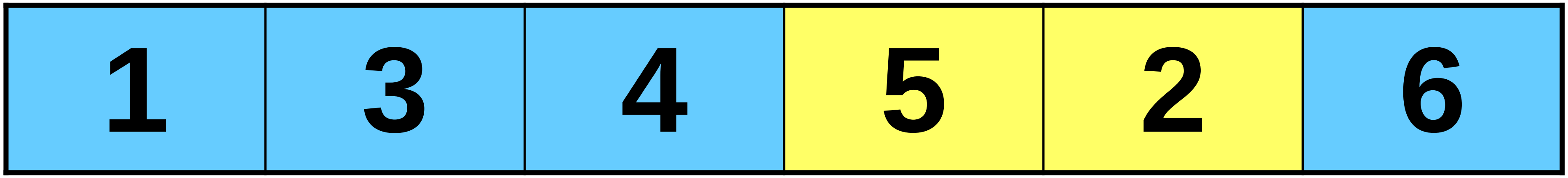
1	3	4	5	2	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



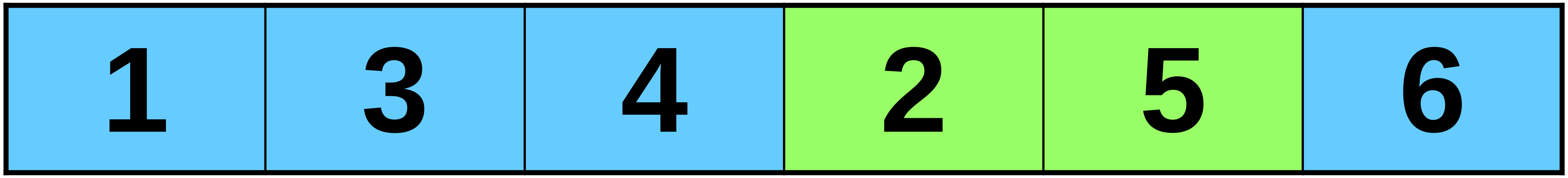
1	3	4	5	2	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort

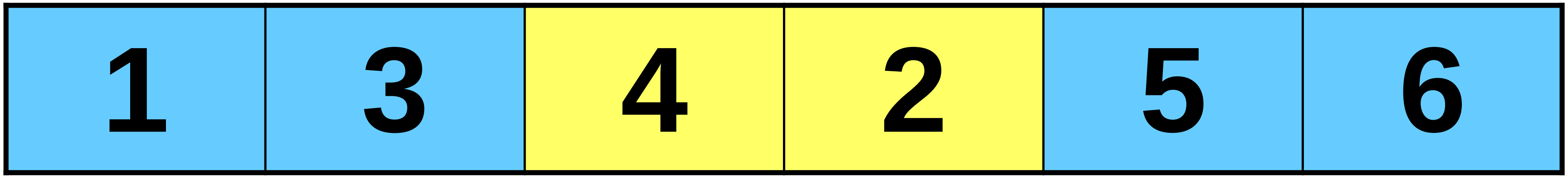


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



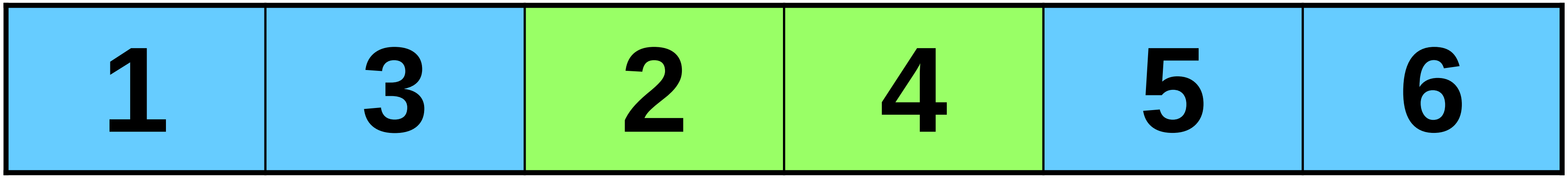
1	3	4	2	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



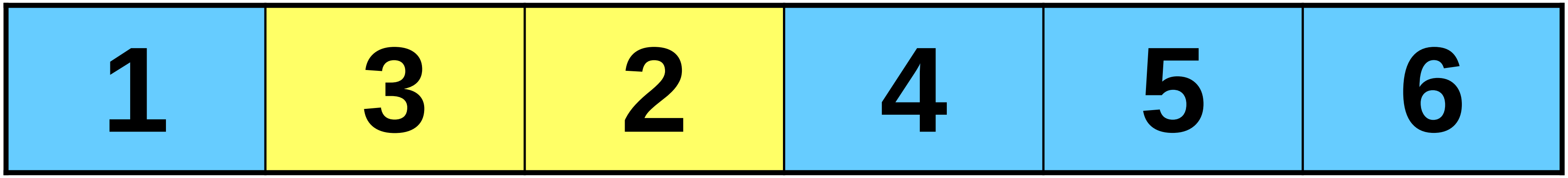
1	3	2	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



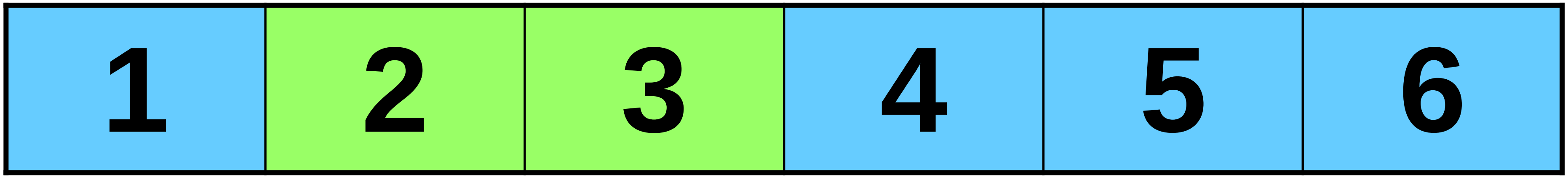
1	3	2	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort



1	2	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort

1	2	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Insertion Sort

```
template<class ItemType>
void InsertItem(ItemType values [], int startIndex, int endIndex)
{
        bool finished = false;
        int current = endIndex;
        bool moreToSearch = (current != startIndex);
        while (moreToSearch && !finished)
        {
                if (values[current] < values[current -1] )
                {
                        Swap(values[current], values[current-1]);
                        current--;
                        moreToSearch = (current != startIndex);
                }
                else finished = true;
        }
)
template<class ItemType>
void InsertionSort(ItemType values[], int numValues)
{
        for (int count = 0; count < numValues; count++)
                InsertItem(values, 0, count);
}
```

Insertion Sort

```
template<class ItemType>
void InsertItem(ItemType values [], int startIndex, int endIndex)
{
        bool finished = false;
        int current = endIndex;
        bool moreToSearch = (current != startIndex);
        while (moreToSearch && !finished)
        {
                if (values[current] < values[current -1] )
                {
                        Swap(values[current], values[current-1]);
                        current--;
                        moreToSearch = (current != startIndex);
                }
                else finished = true;
        }
)
template<class ItemType>
void InsertionSort(ItemType values[], int numValues)
{
        for (int count = 0; count < numValues; count++)
                InsertItem(values, 0, count);
}
```

$O(N^2)$