INTERNATIONAL COLLEGE OF BUSINESS & TECHNOLOGY
ICBT

Cardiff Metropolitan University
Prifysgol Fetropolitan Caerdydd
UWIC

HND in Computing / Software Engineering

Data Structures and Algorithms SED52013 Credit -20

Learning outcomes covered

- Understand Data Structures and storage in computer systems.

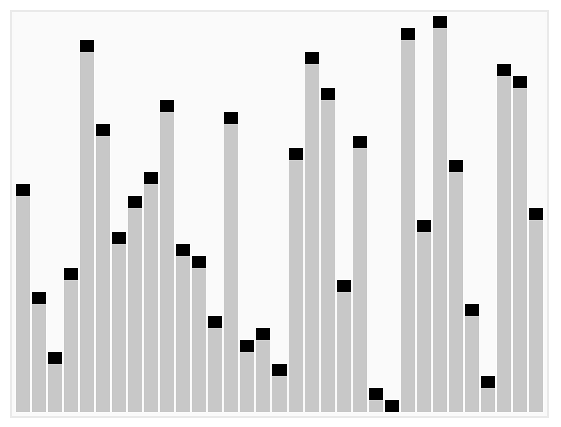
- Develop and implement Suitable Data Structures given requirement.

- Develop, implement and use searching and sorting techniques.

Sorting

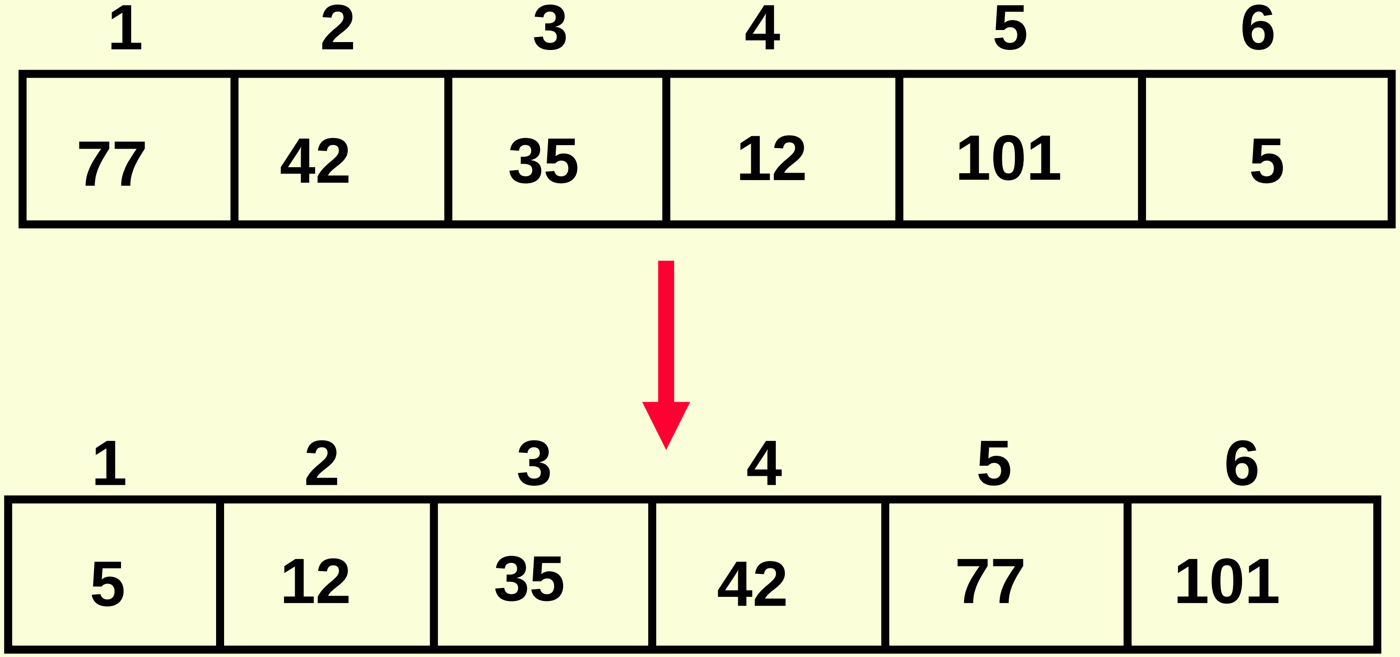
Priyanga Siriwardana

Msc in IT,Bsc(sp) Hons in IT



Sorting

- Sorting takes an unordered collection and makes it an ordered one.



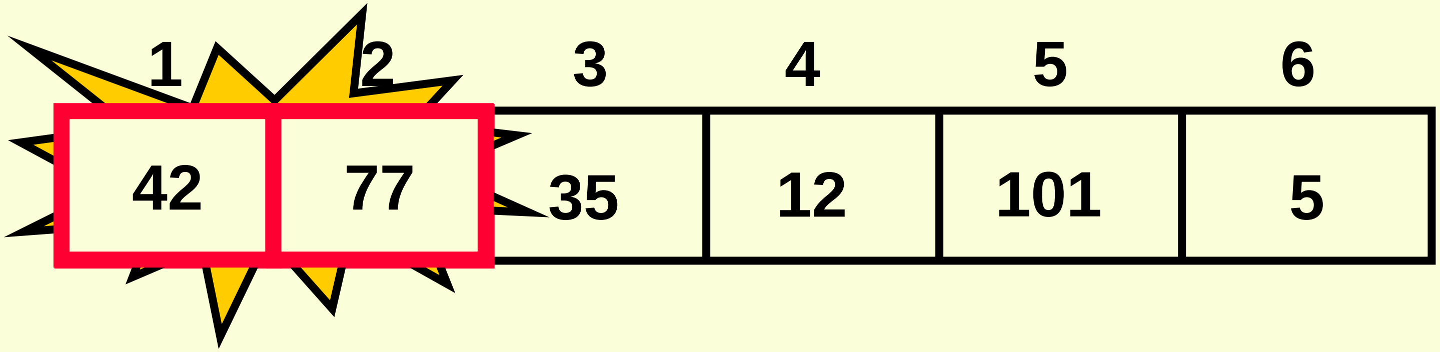
1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
5	12	35	42	77	101

Bubble Sort

"Bubbling Up" the Largest Element

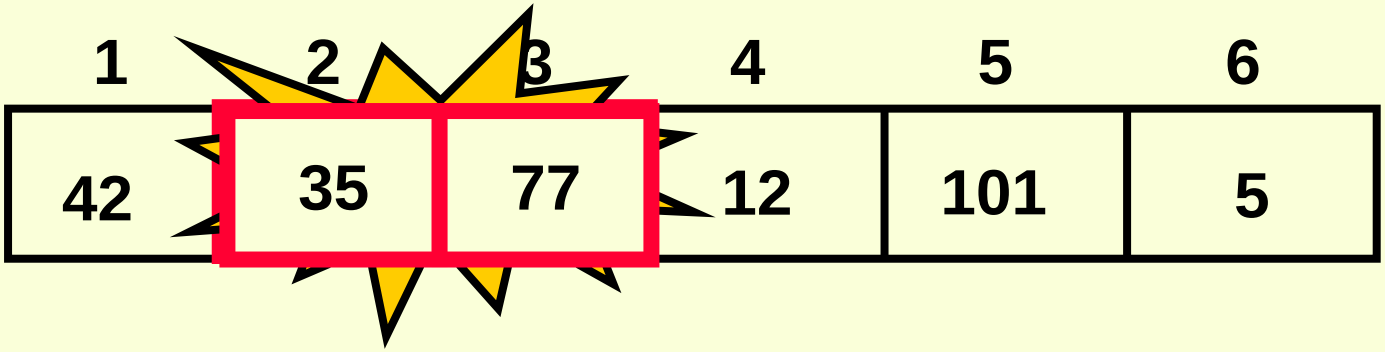
- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



1	2	3	4	5	6
42	77	35	12	101	5

"Bubbling Up" the Largest Element

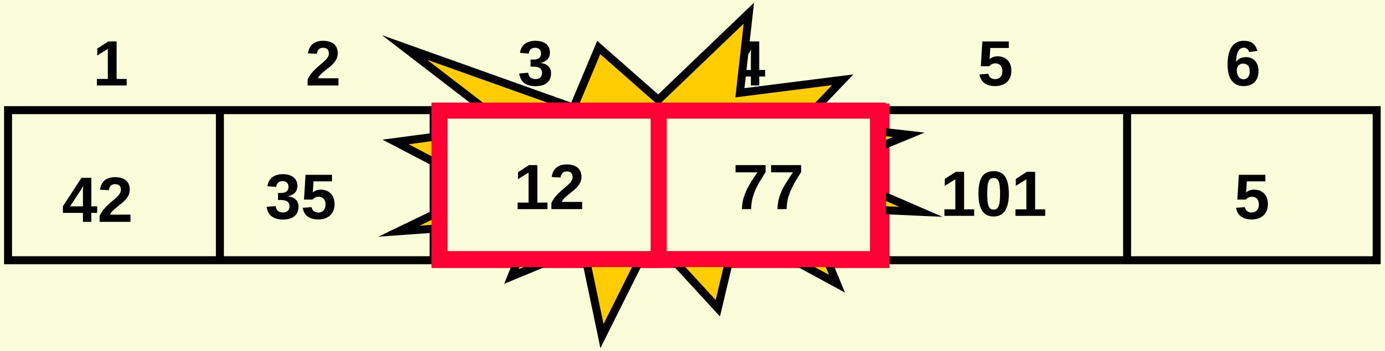
- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



1	2	3	4	5	6
42	35	77	12	101	5

"Bubbling Up" the Largest Element

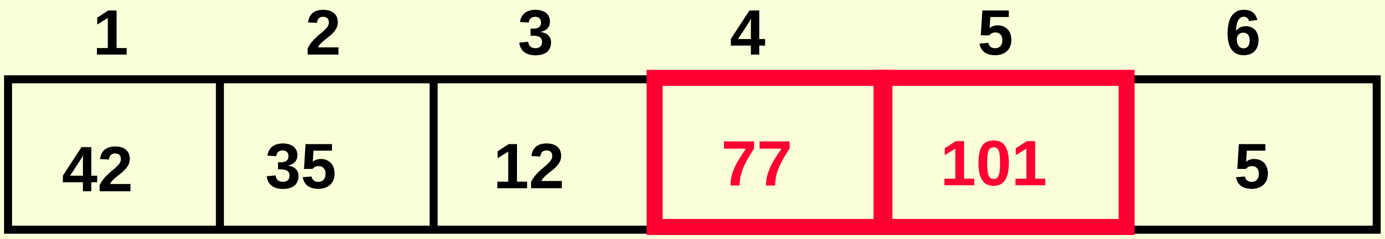
- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



1	2	3	4	5	6
42	35	12	77	101	5

"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping

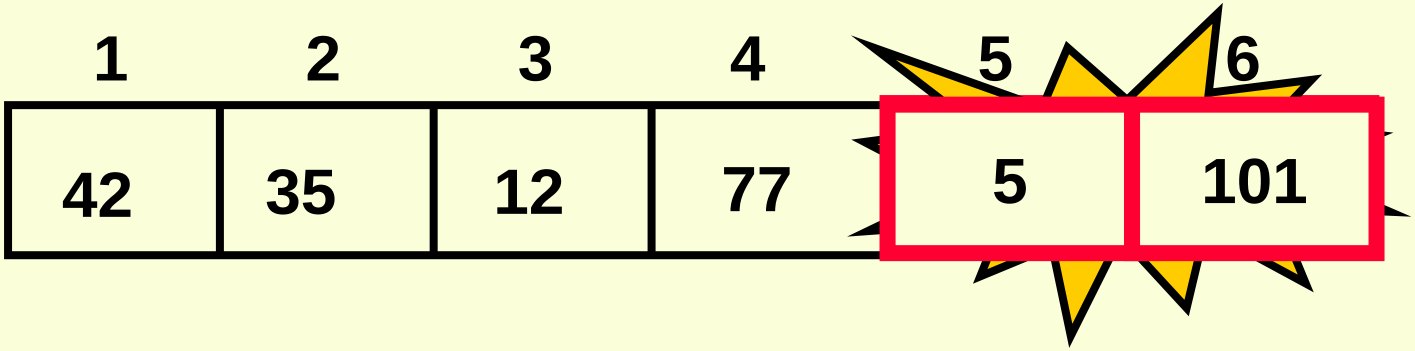


1	2	3	4	5	6
42	35	12	77	101	5

No need to swap

"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



1	2	3	4	5	6
42	35	12	77	5	101

"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping

1	2	3	4	5	6
42	35	12	77	5	101

Largest value correctly placed

```
•   public static void bubbleSort1(int[] x)
•   {
•       int n = x.length;
•       for (int pass=1; pass < n; pass++)
•      {
•           for (int i=0; i < n-pass; i++)
•           {
•               if (x[i] > x[i+1])
•               {
•                   int temp = x[i];
•                        x[i] = x[i+1];
•                        x[i+1] = temp;
•               }
•           }
•       }
•   }
```

Items of Interest

- Notice that only the largest value is correctly placed
- All other values are still out of order
- So we need to repeat this process

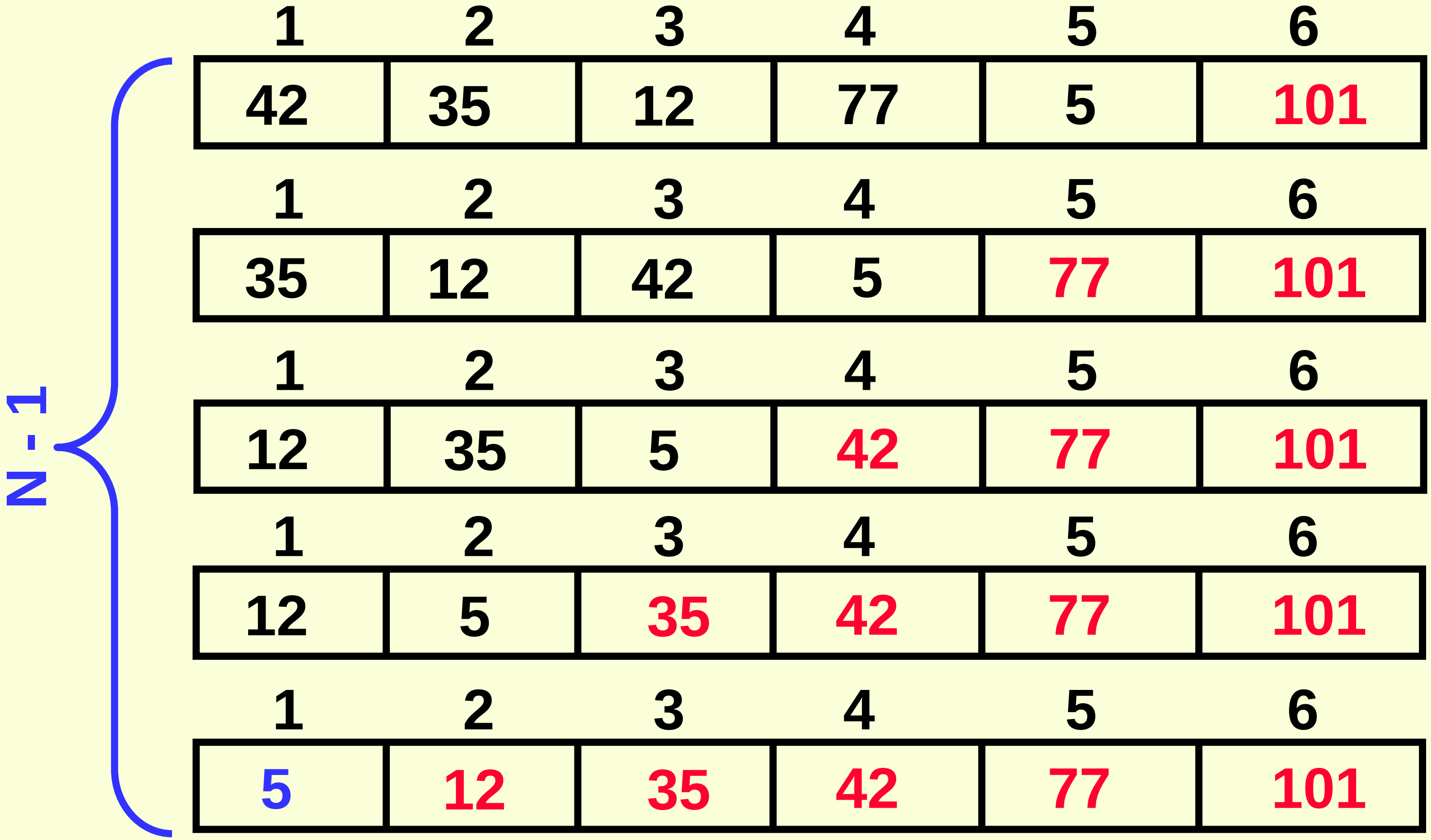
1	2	3	4	5	6
42	35	12	77	5	101

Largest value correctly placed

Repeat “Bubble Up” How Many Times?

- If we have N elements…
- And if each time we bubble an element, we place it in its correct location…
- Then we repeat the “bubble up” process N – 1 times.
- This guarantees we’ll correctly place all N elements.

“Bubbling” All the Elements



N - 1

1	2	3	4	5	6
42	35	12	77	5	101

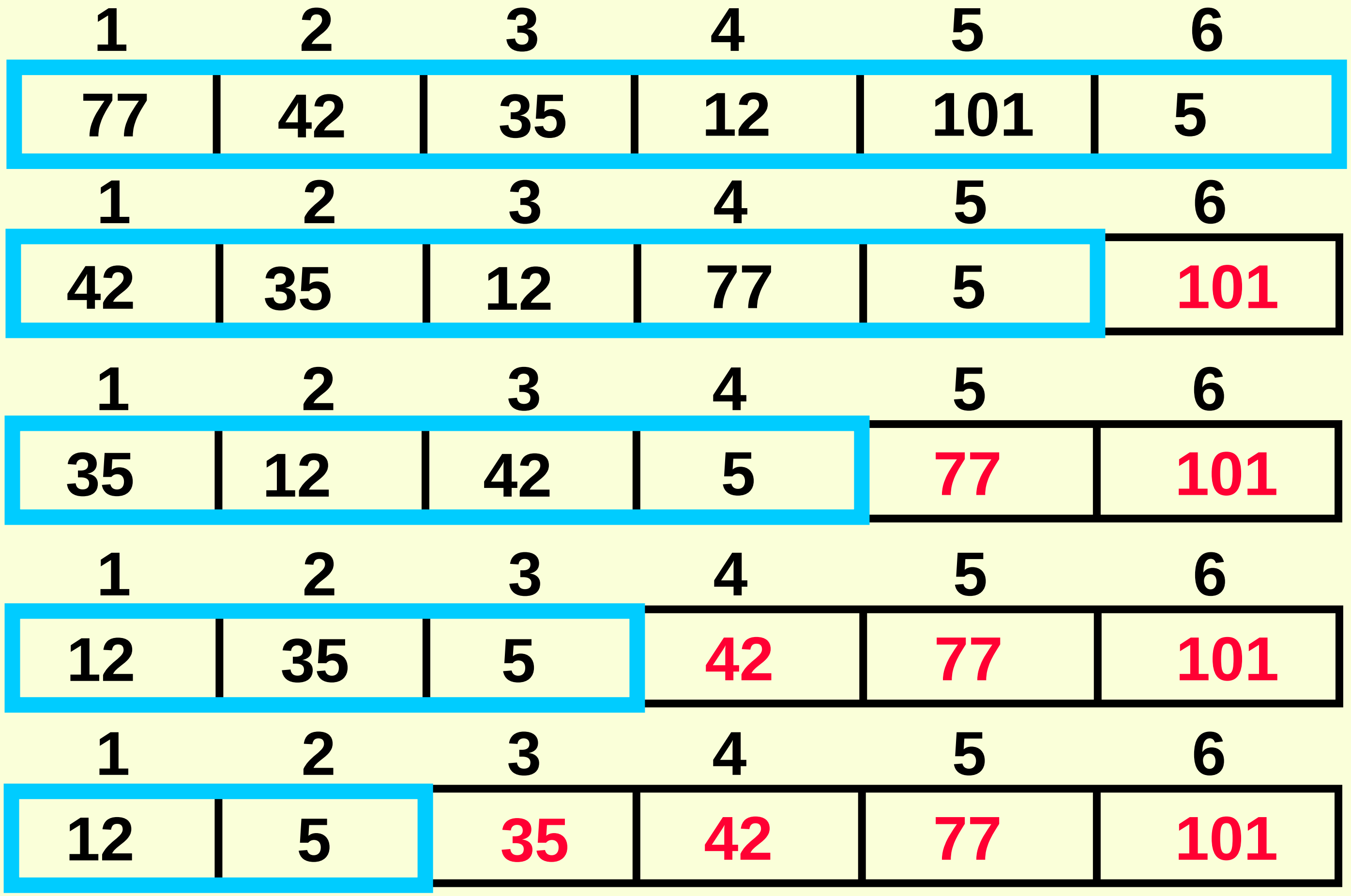
1	2	3	4	5	6
35	12	42	5	77	101

1	2	3	4	5	6
12	35	5	42	77	101

1	2	3	4	5	6
12	5	35	42	77	101

1	2	3	4	5	6
5	12	35	42	77	101

Reducing the Number of Comparisons



1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
42	35	12	77	5	101

1	2	3	4	5	6
35	12	42	5	77	101

1	2	3	4	5	6
12	35	5	42	77	101

1	2	3	4	5	6
12	5	35	42	77	101

Efficiency of the Bubble Sort

- If N number of Items in the Array ,
- there are N-1 comparisons in First
- there are N-2 comparisons in Second
- Then N-3
- (N-1)+(N-2)+(N-3)+………….+1=N*(N-1)/2
- In Big O notation O(N2) times run in the Algorithm

Selection Sort

Selection Sort

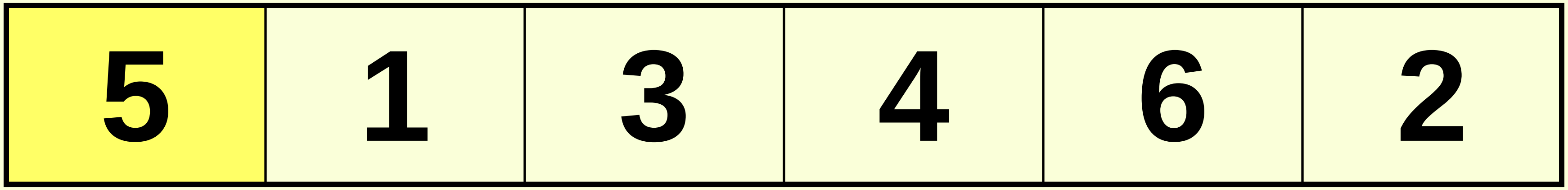
5	1	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



5	1	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

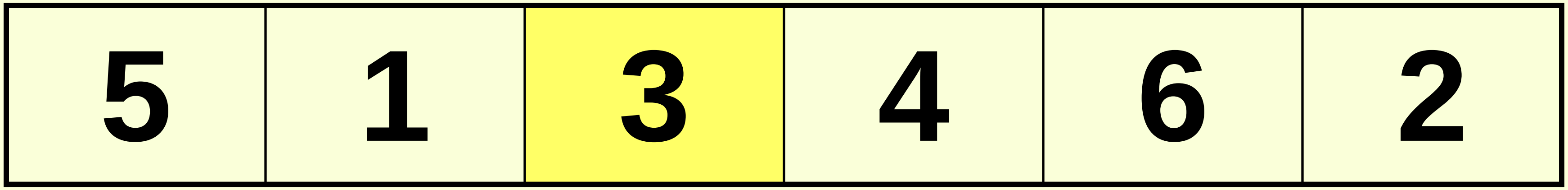
5	1	3	4	6	2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



5	1	3	4	6	2

🟨 **Comparison**

🟩 **Data Movement**

🟦 **Sorted**

Selection Sort

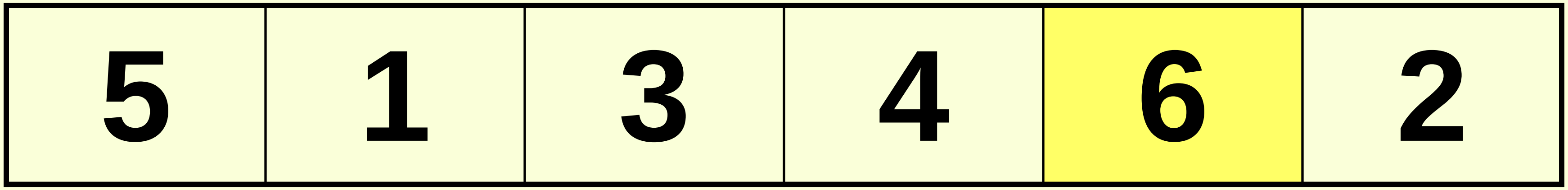
5	1	3	4	6	2

🟨 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

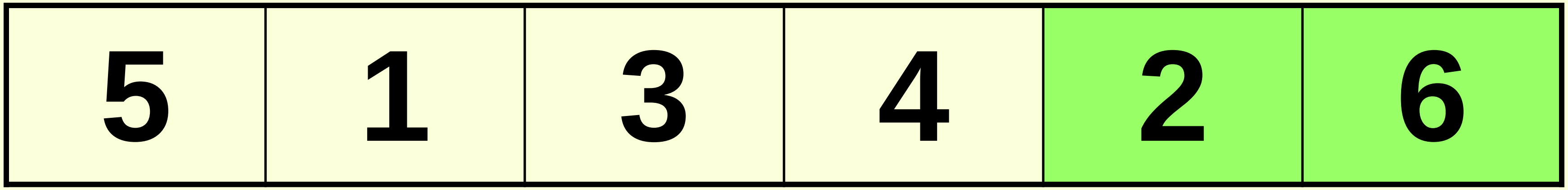
↑ Largest

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



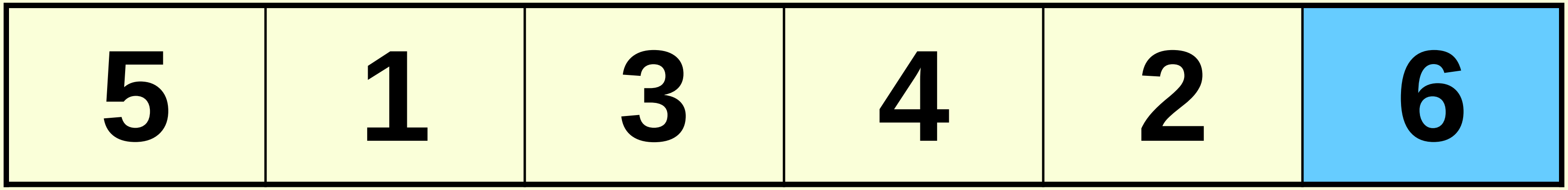
5	1	3	4	2	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



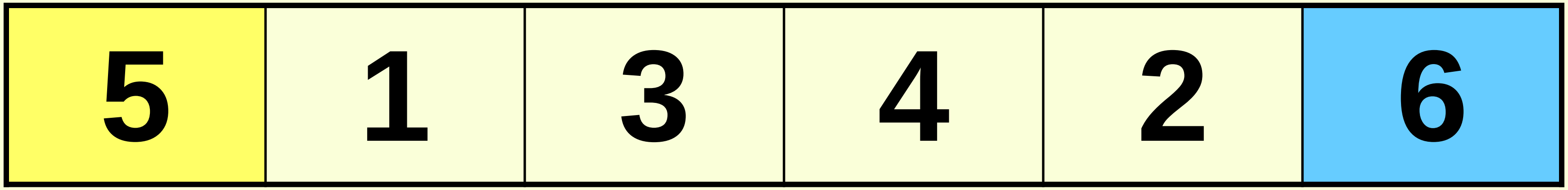
5	1	3	4	2	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



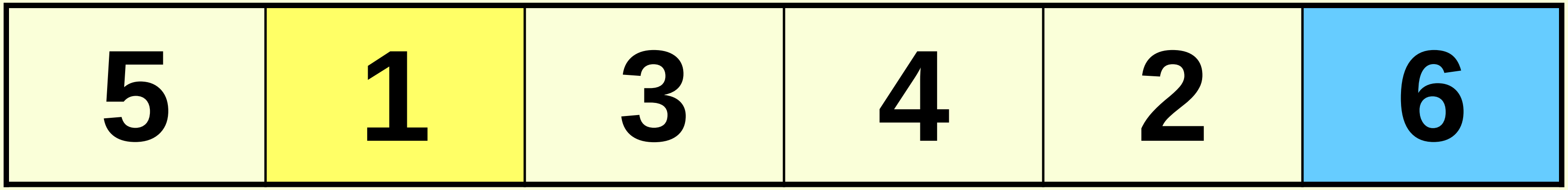
5	1	3	4	2	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



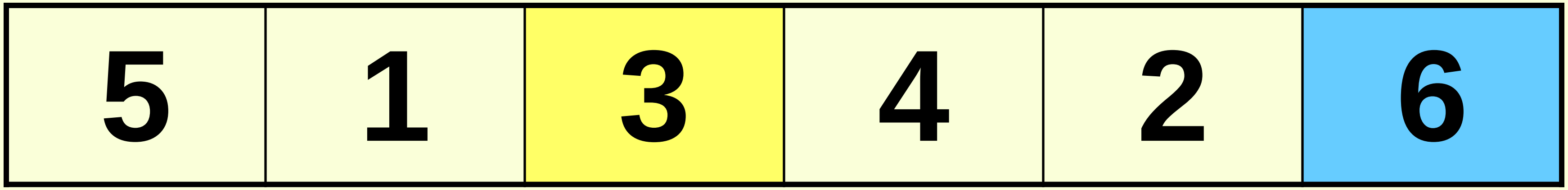
5	1	3	4	2	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



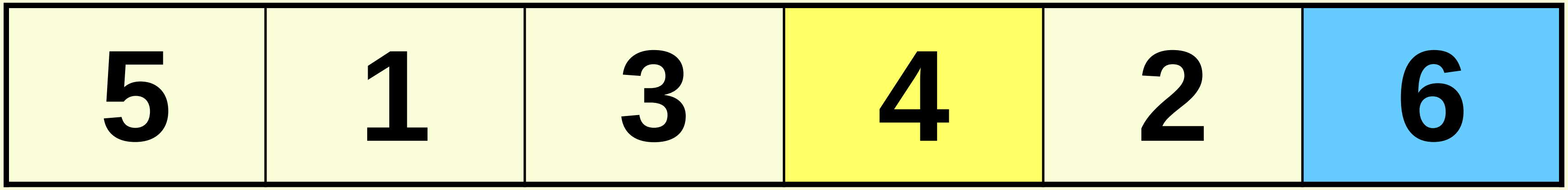
5	1	3	4	2	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



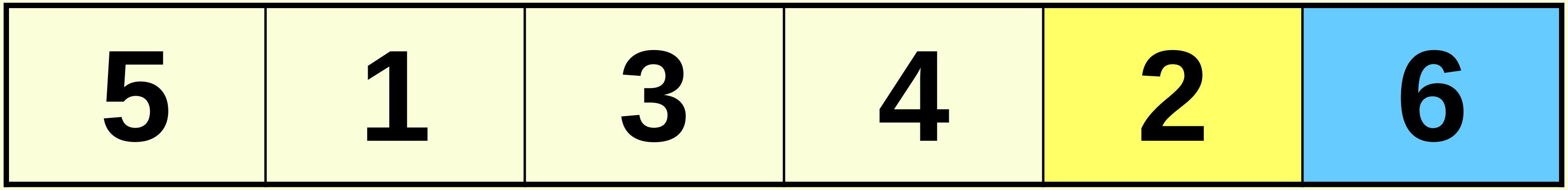
5	1	3	4	2	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



5	1	3	4	2	6

🟨 **Comparison**

🟩 **Data Movement**

🟦 **Sorted**

Selection Sort

5	1	3	4	2	6

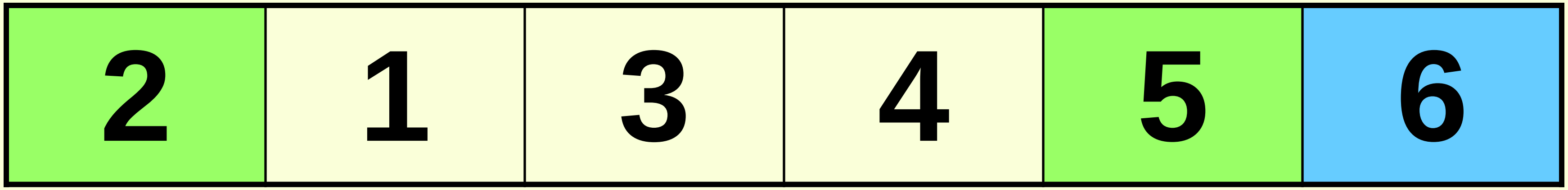
↑
Larges
t

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



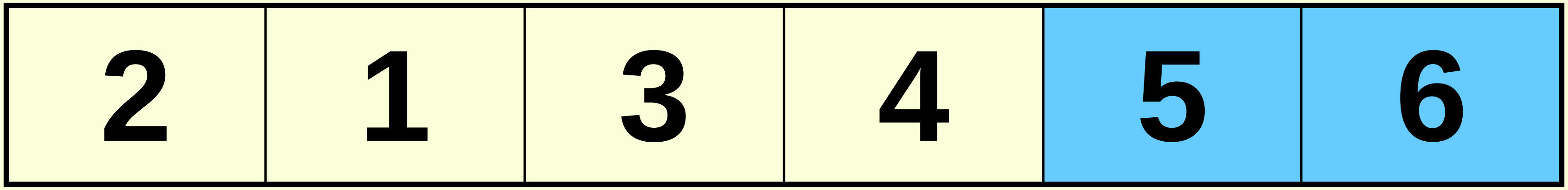
2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



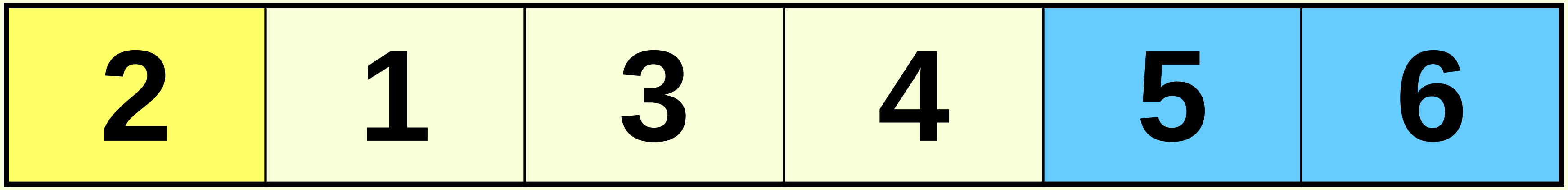
2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



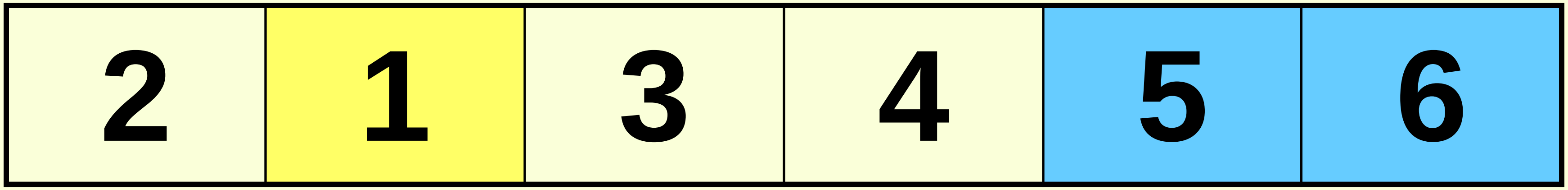
2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



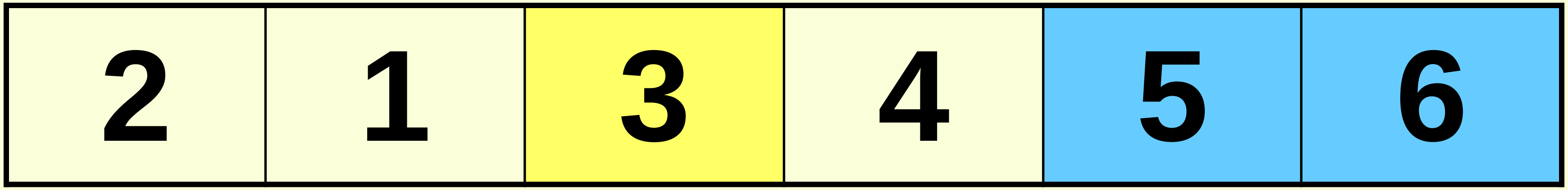
2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



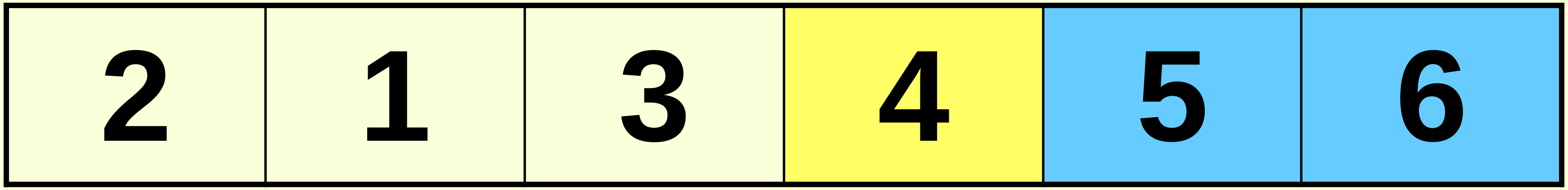
2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



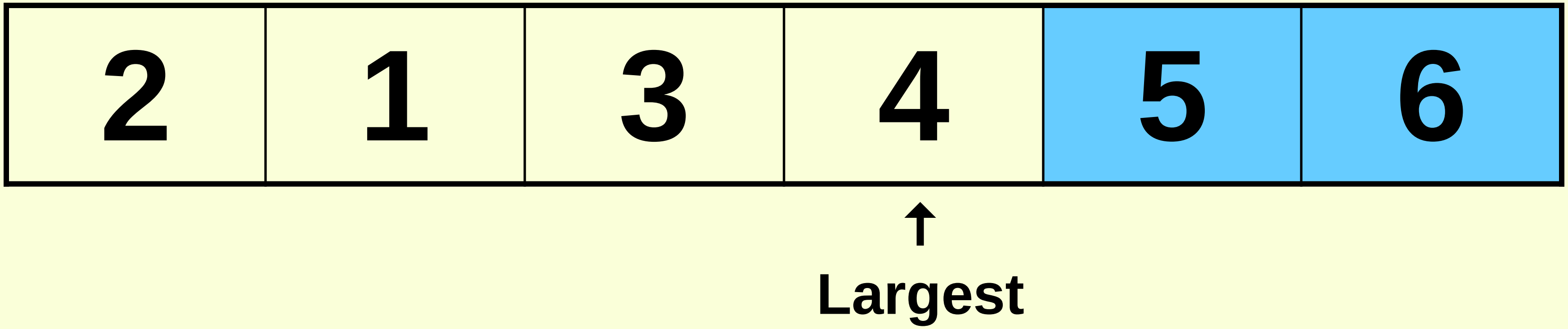
2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



2	1	3	4	🟦 5	🟦 6

↑ Largest

- 🟨 Comparison
- 🟩 Data Movement
- 🟦 Sorted

Selection Sort

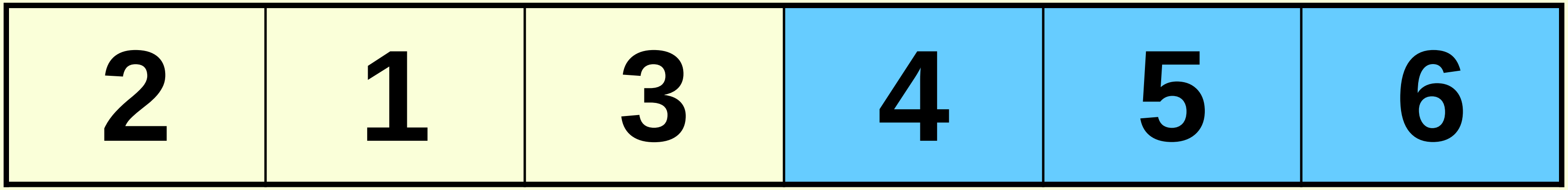
2	1	3	🟩 4	🟦 5	🟦 6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



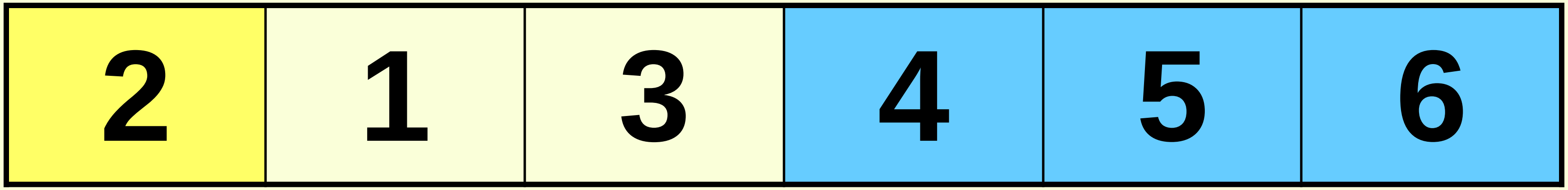
2	1	3	🟦 4	🟦 5	🟦 6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



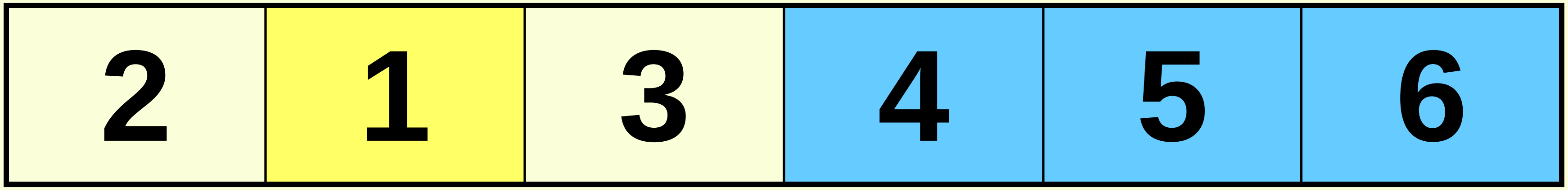
2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



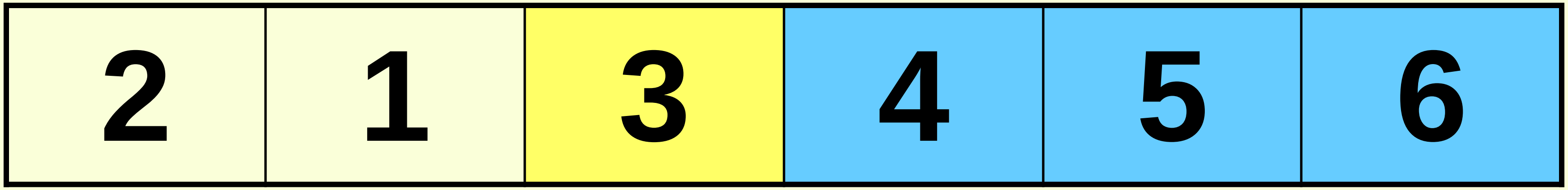
2 1 3 4 5 6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



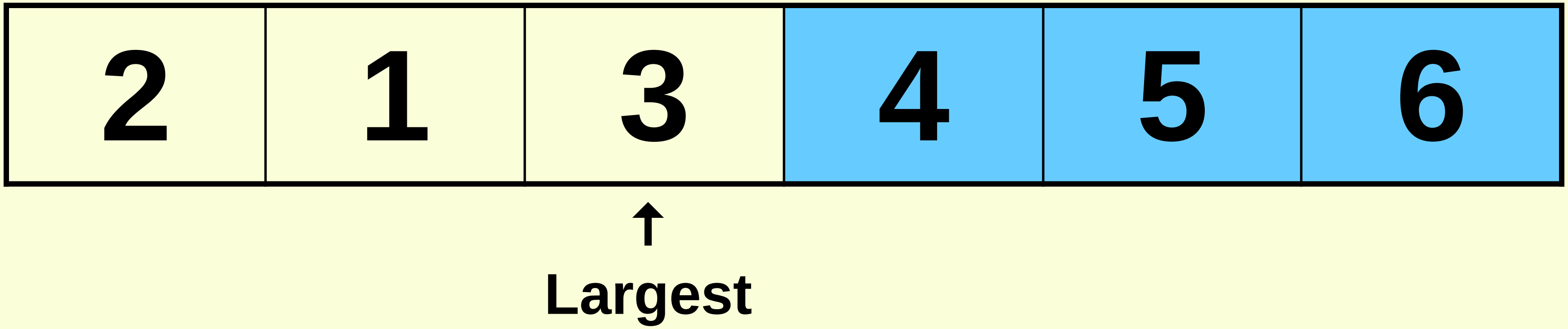
2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



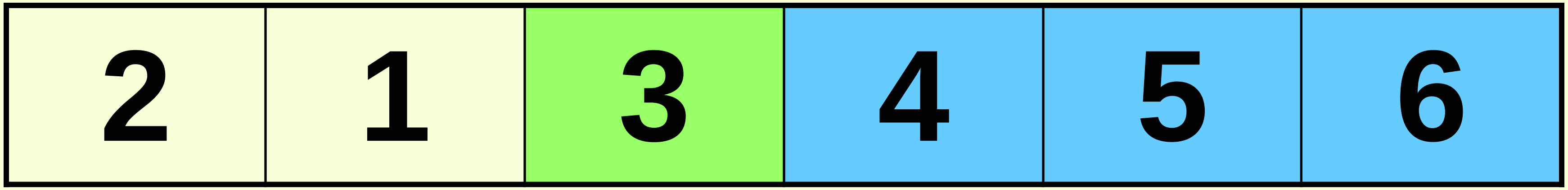
2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

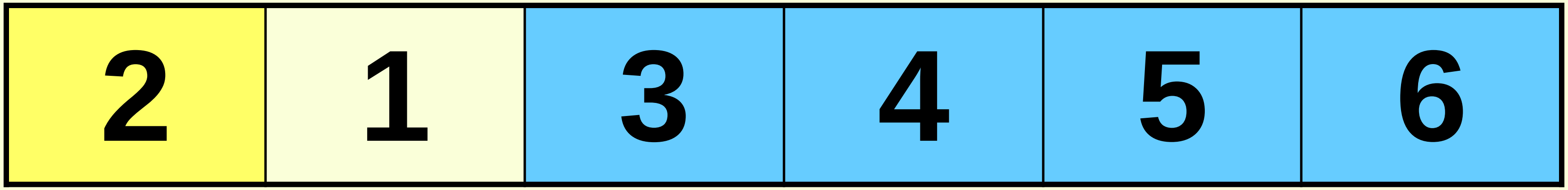
2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



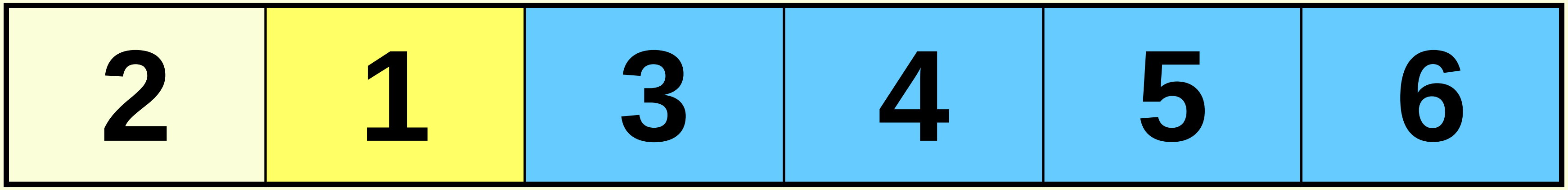
2	1	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



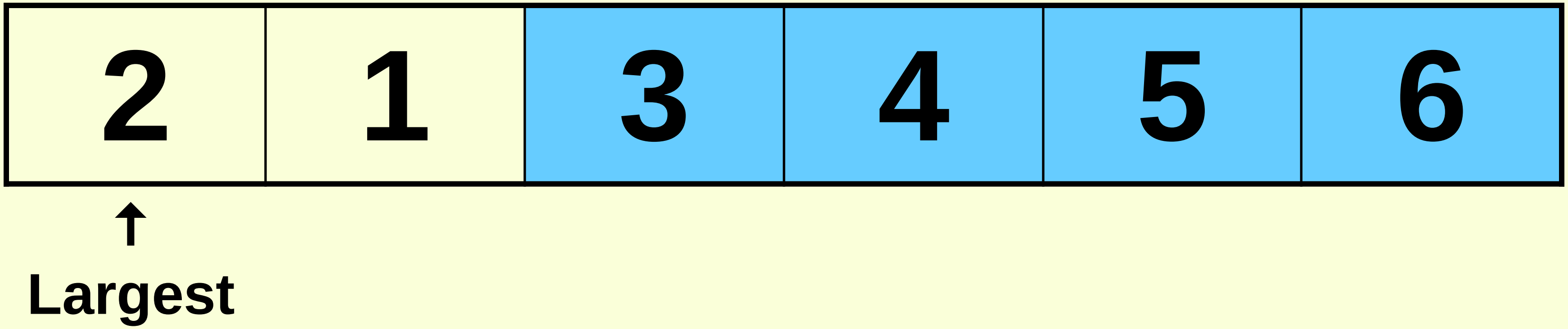
2 1 3 4 5 6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



2	1	3	4	5	6

↑

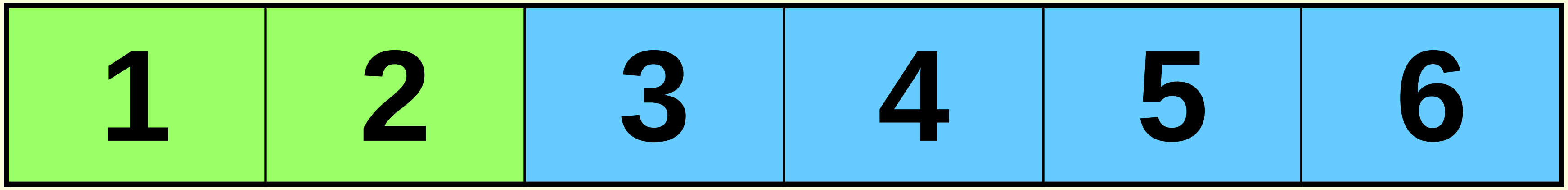
Largest

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



1	2	3	4	5	6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

1	2	3	4	5	6

DONE!

🟨 **Comparison**

🟩 **Data Movement**

🟦 **Sorted**

```
public static void selectionSort2(int[] x)
 {
    for (int i=0; i<x.length-1; i++)
    {
        int minIndex = i;
        for (int j=i+1; j<x.length; j++)
        {
            if (x[minIndex] > x[j])
             {
                minIndex = j;
            }
        }
        if (minIndex != i)
         {
             int temp = x[i];
            x[i] = x[minIndex];
            x[minIndex] = temp;
        }
    }
}
```

Efficiency of the Selection Sort

- If it is N number of elements the number of Comparisons will be N*(N-1)/2
- Normally Selection Sort will run the O(N2)