Data Structures and Algorithms CSE220

Prof. Ramesh Ragala

August 13, 2015

Introduction to Brute Force Approach

- It is a one of the simplest of Algorithm design strategies.
- **A straightforward approach to solving problem,**
  - **Usually based on problem statement**
  - **Definitions of the concepts involved**
- It uses a Strategy → Just Do It → Another name
- The results in the algorithm can be improved later and easiest to apply to solve problems.
- The "force" implied by the strategys definitions is that of a computer and not that of ones intellect.
- Examples: Selection Sort, Bubble Sort, Pattern Matching
- consider computing a^n for a given number a and a is non-negative integer n.
- By the definition of exponentiation, a^n = a X a X a X a ... n-times

Selection Sort

- Problem:
  - Given a list of n-orderable items (e.g., numbers, characters from some alphabet, character strings), rearrange them in non-decreasing order.
- Solving using Brute force approach
  - Scan the entire given list to find its smallest element and exchange it with the first element position. $\rightarrow$ Putting it into its final position.
  - Start Scanning the list form the second element onwards, find the smallest element and put into its final position. (scanning is done on (n-1) elements)
  - This process repeats until all elements are sorted. sorts after (n-1)passes

$$\underbrace{A_0 \leq A_1 \leq \cdots \leq A_{i-1}}_{\text{in their final positions}} \mid \underbrace{A_i, \ldots, A_{min}, \ldots, A_{n-1}}_{\text{the last } n-i \text{ elements}}$$

Selection Sort

- Selection sort is among the simplest of sorting techniques.
- This Selection Sort works well for small data.
- Section sort is a good choice for sorting files with very large objects (records) and small keys.
- We can also first find the largest in the list and swap with the last position of the list.
- Then Second largest element and exchange it with the element in the second largest position. $\rightarrow$ Repeat this Process.

Selection Sort

- Algorithm for Selection Sort

Algorithm Selection(A,n)

```
{
for i ← 0 to n − 2 do
    min ← i;
    for j ← i + 1 to n − 1 do
        if (A[j] < A[min]) then
            min ← j;
        end if
    end for
end for
swap A[i] and A[min];
}
```

Selection Sort

- Analysis of Selection Sort
  - Input Size is given bye number of elements
  - Basic Operation: Key Comparisons

$$
\begin{aligned}
C(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \\
&= \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] \\
&= \sum_{i=0}^{n-2} (n-1-i) \\
&= (n-1)n/2 \\
&= \Theta(n^2)
\end{aligned}
$$

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

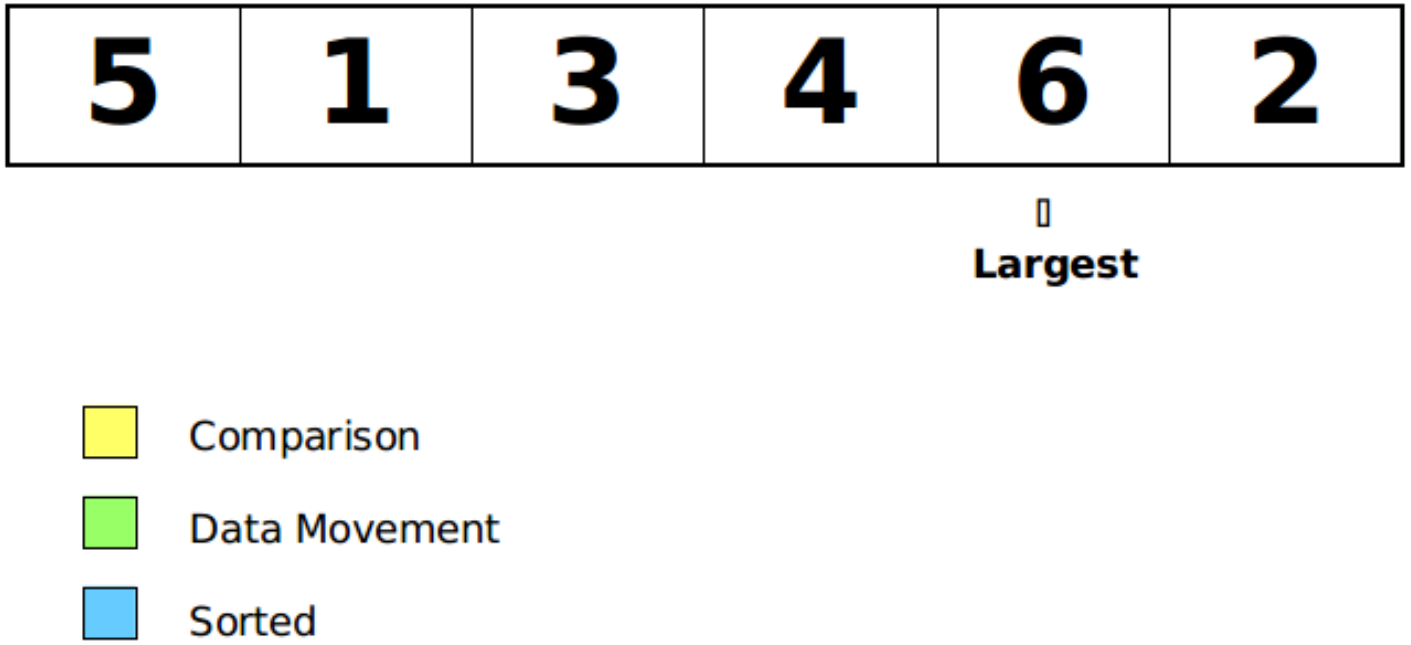
- Comparison
- Data Movement
- Sorted

Selection Sort

5	1	3	4	6	2

- Comparison
- Data Movement
- Sorted

Selection Sort



Selection Sort

5	1	3	4	2	6

- Comparison
- Data Movement
- Sorted

Selection Sort

5	1	3	4	2	6

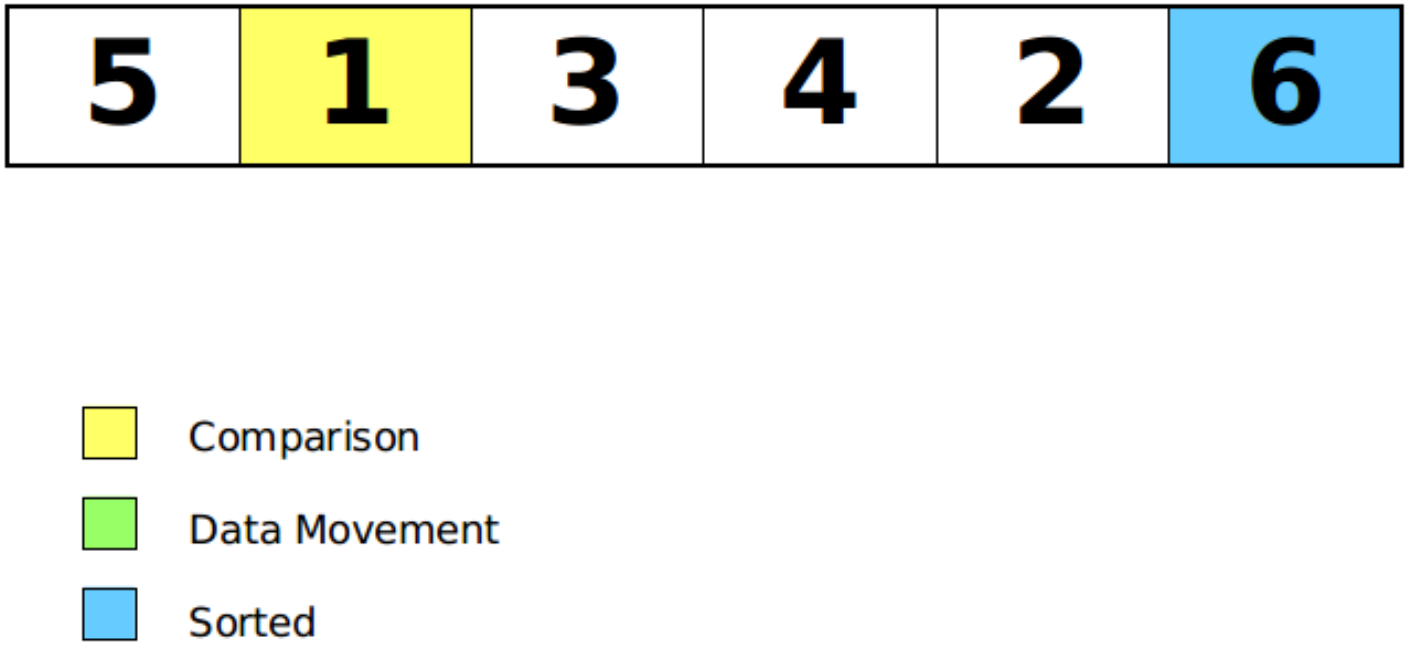
- Comparison
- Data Movement
- Sorted

Selection Sort

5	1	3	4	2	6

- Comparison
- Data Movement
- Sorted

Selection Sort



Selection Sort

5	1	3	4	2	6

- Comparison
- Data Movement
- Sorted

Selection Sort

5	1	3	4	2	6

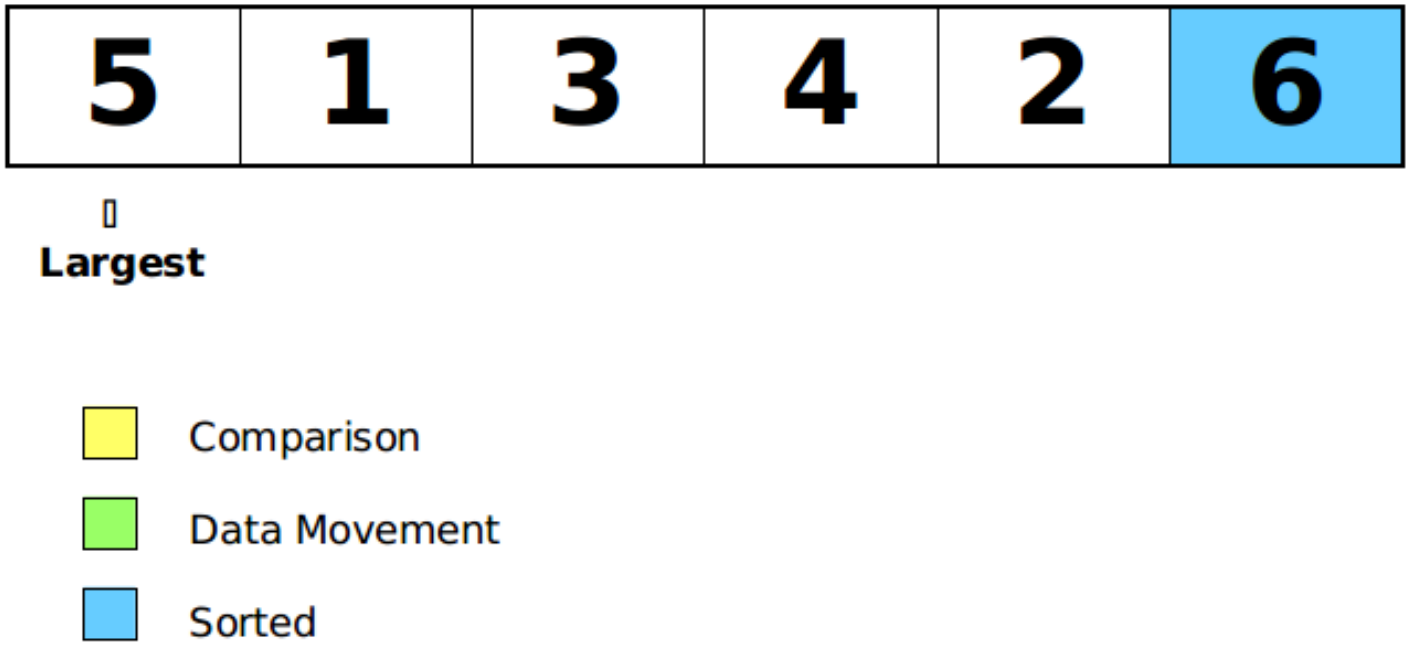
- Comparison
- Data Movement
- Sorted

Selection Sort

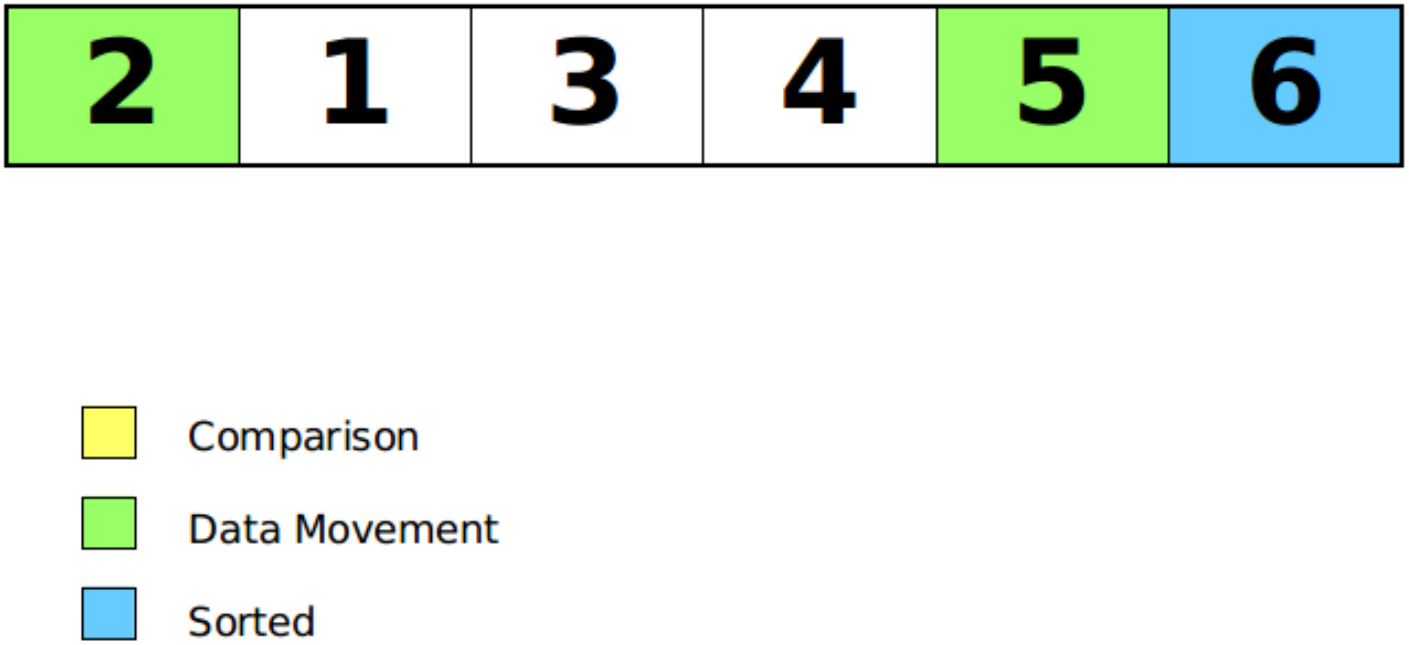
5	1	3	4	2	6

- Comparison
- Data Movement
- Sorted

Selection Sort



Selection Sort

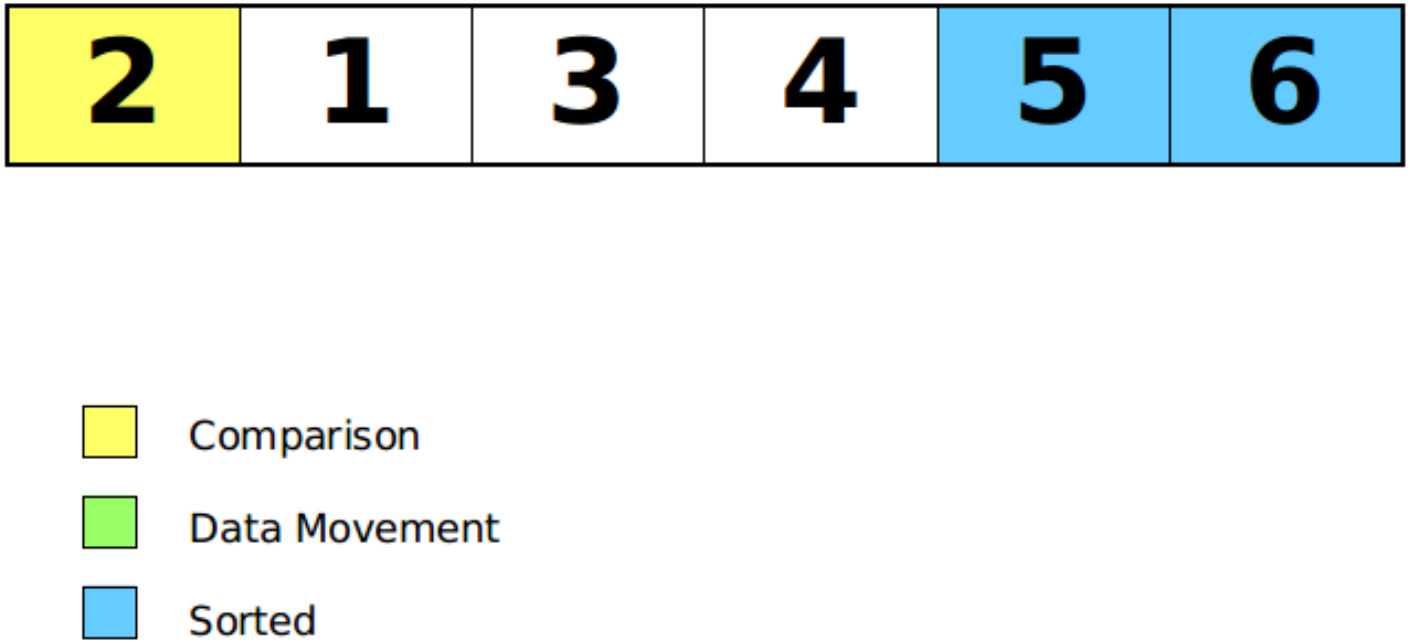


Selection Sort

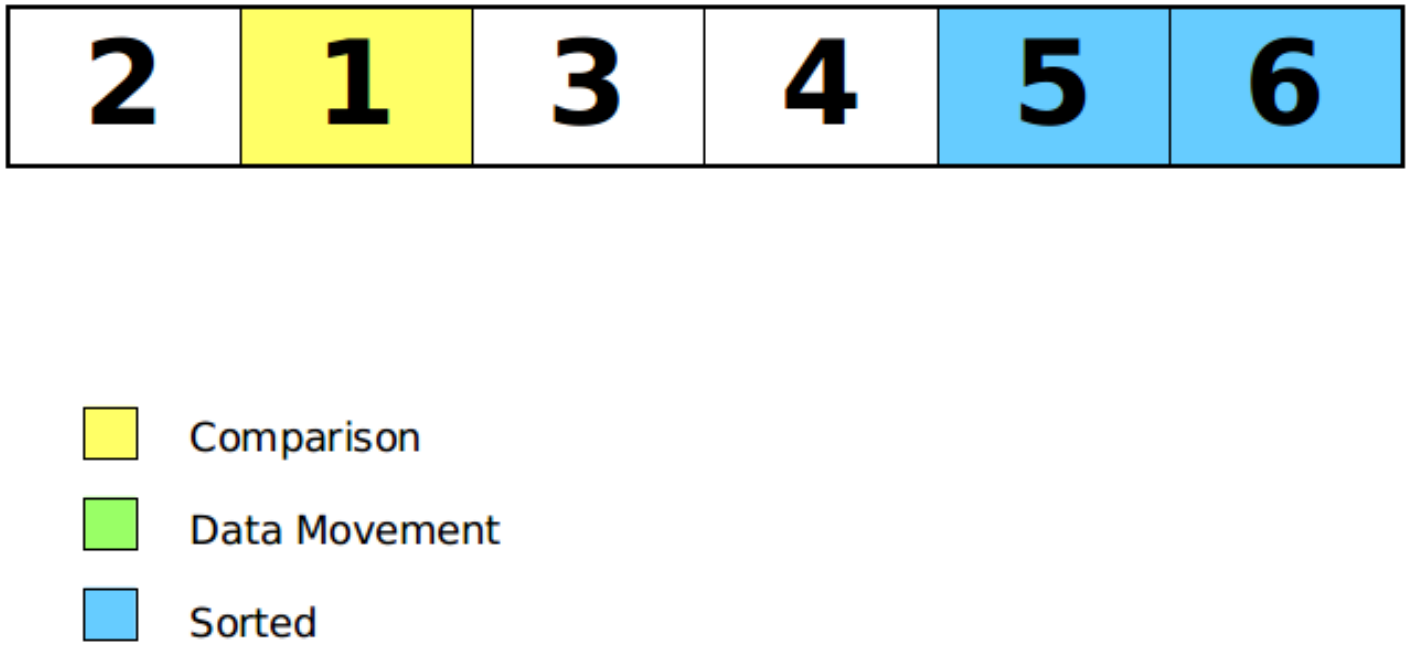
2	1	3	4	5	6

- Comparison
- Data Movement
- Sorted

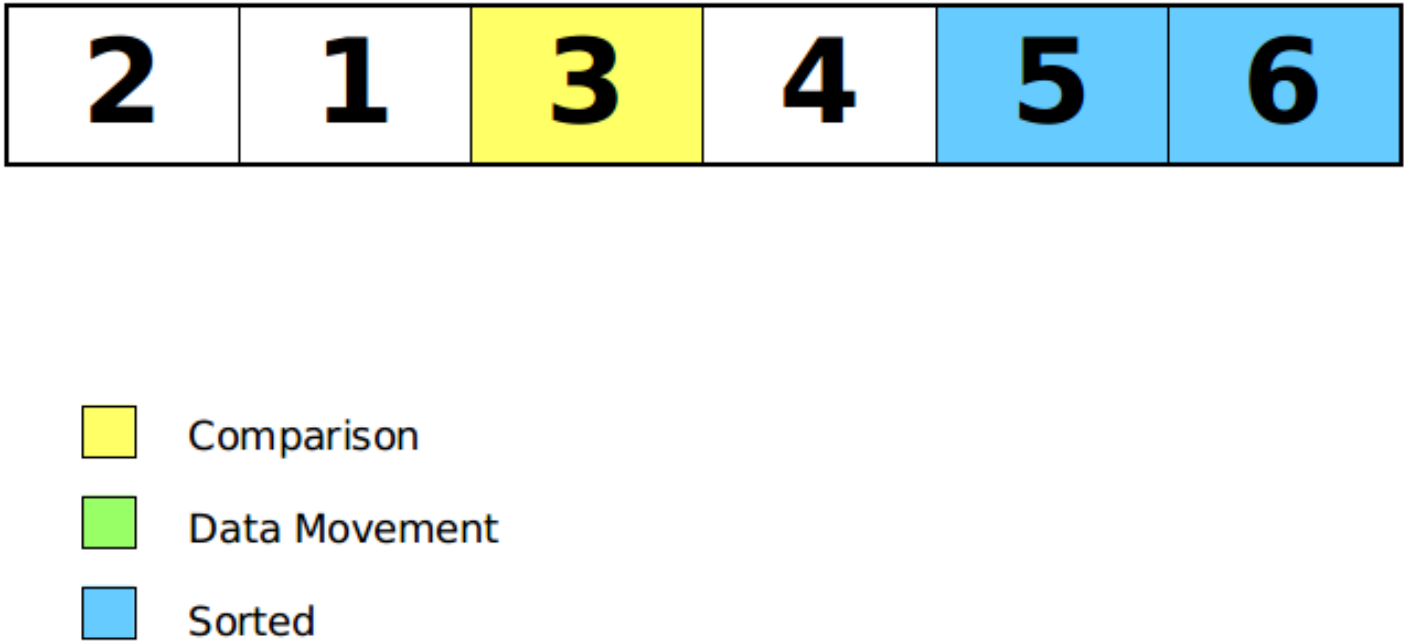
Selection Sort



Selection Sort



Selection Sort



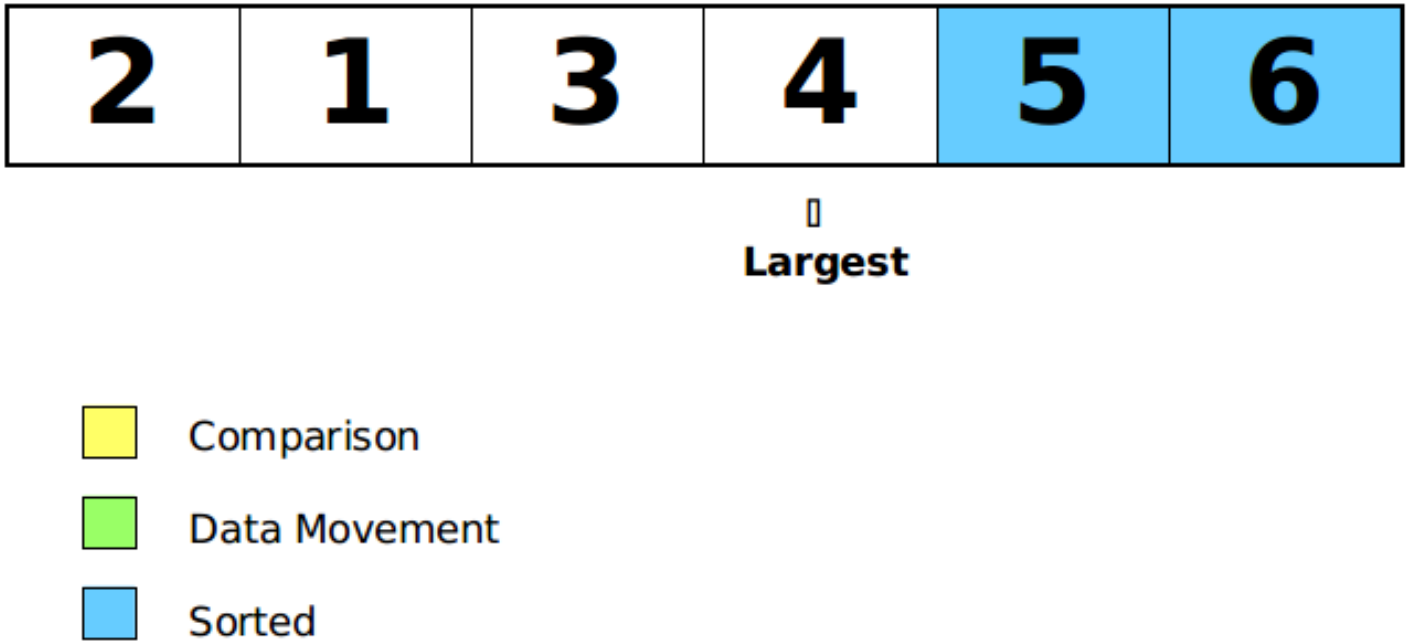
Selection Sort

Comparison

Data Movement

Sorted

Selection Sort



Selection Sort

Comparison

Data Movement

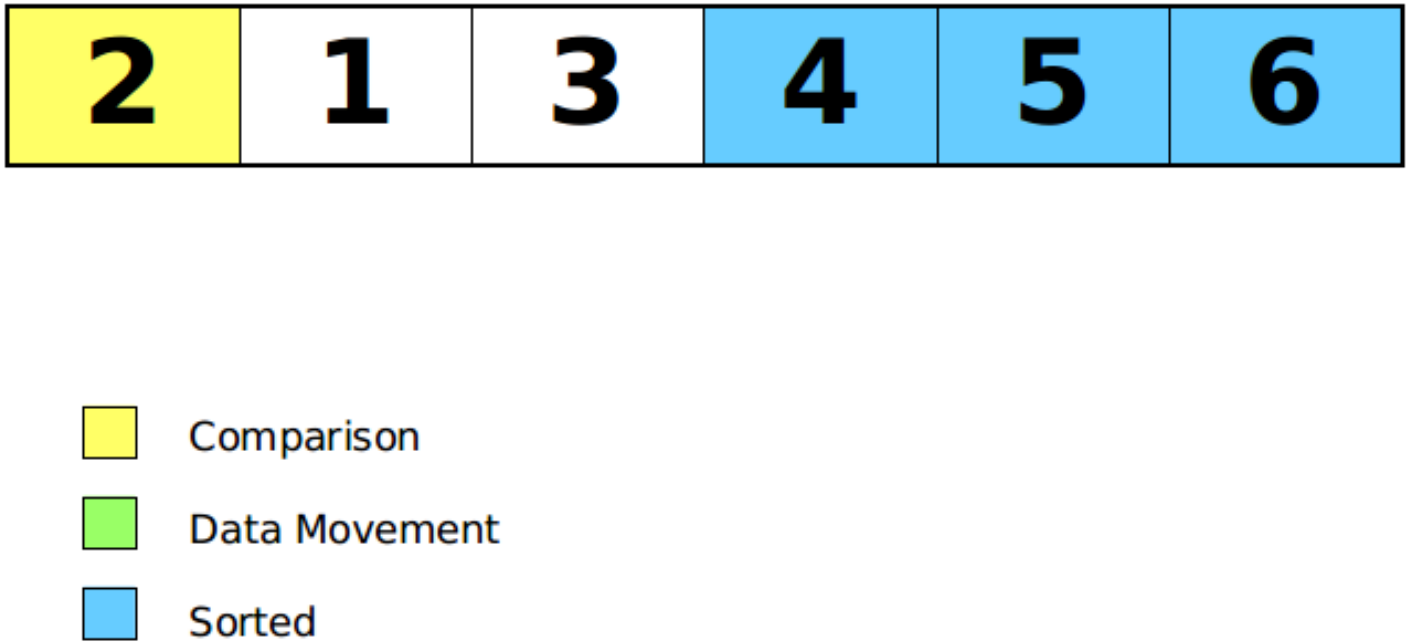
Sorted

Selection Sort

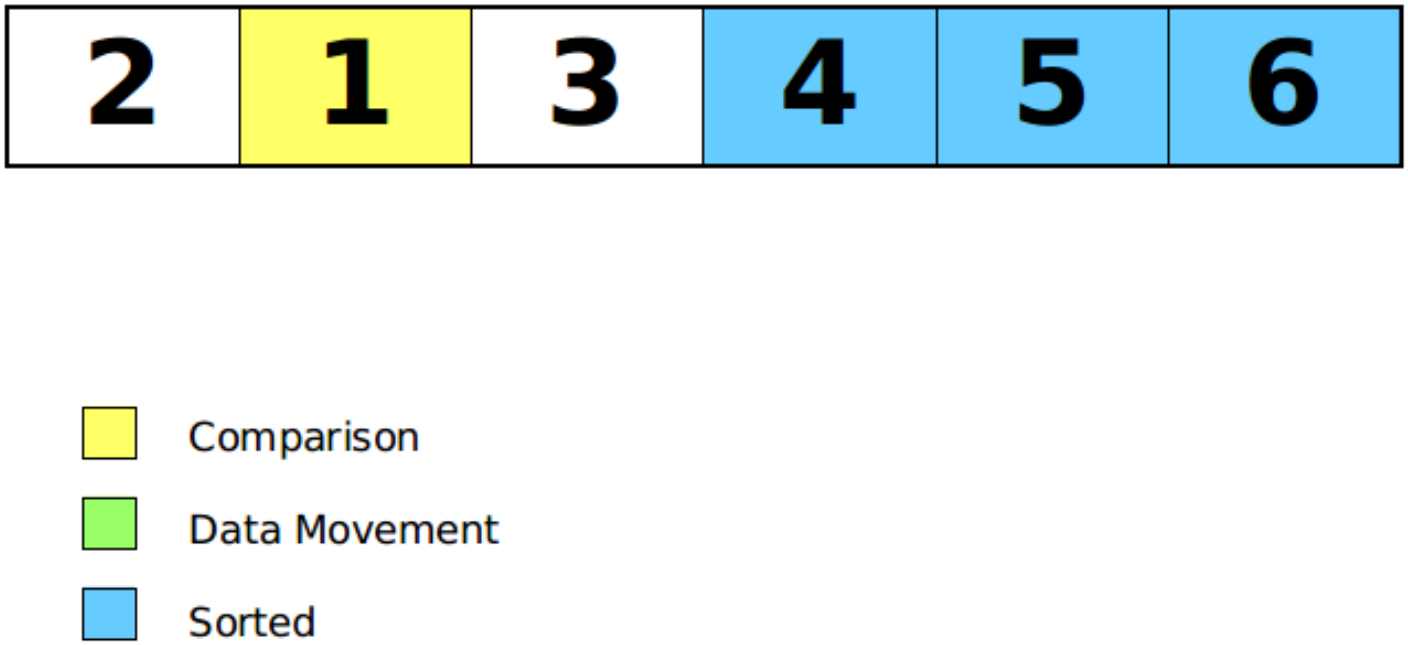
2	1	3	4	5	6

- Comparison
- Data Movement
- Sorted

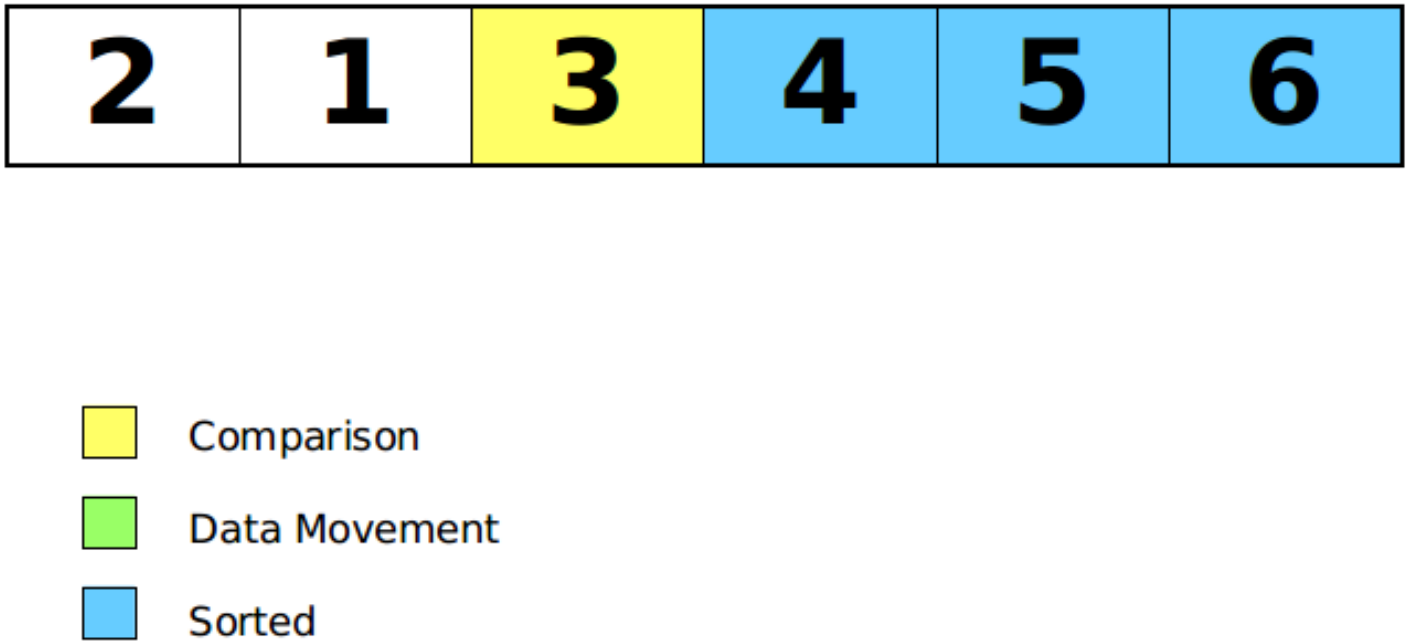
Selection Sort



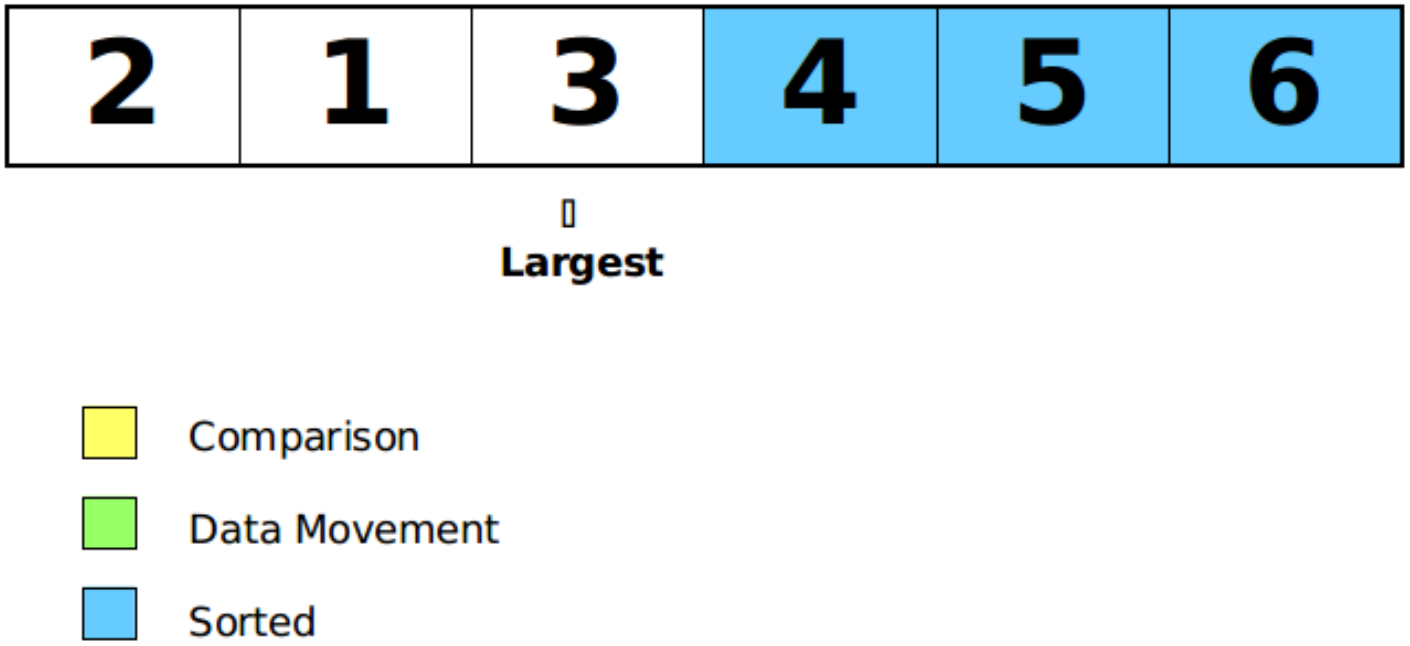
Selection Sort



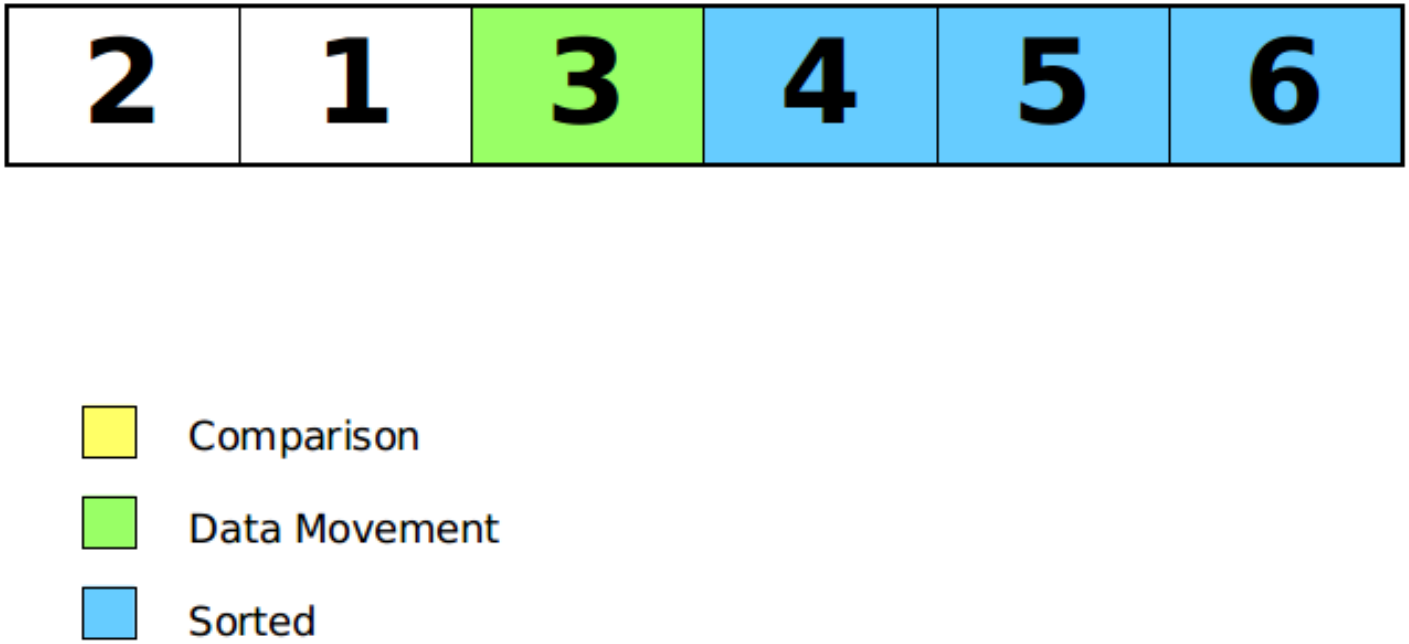
Selection Sort



Selection Sort



Selection Sort

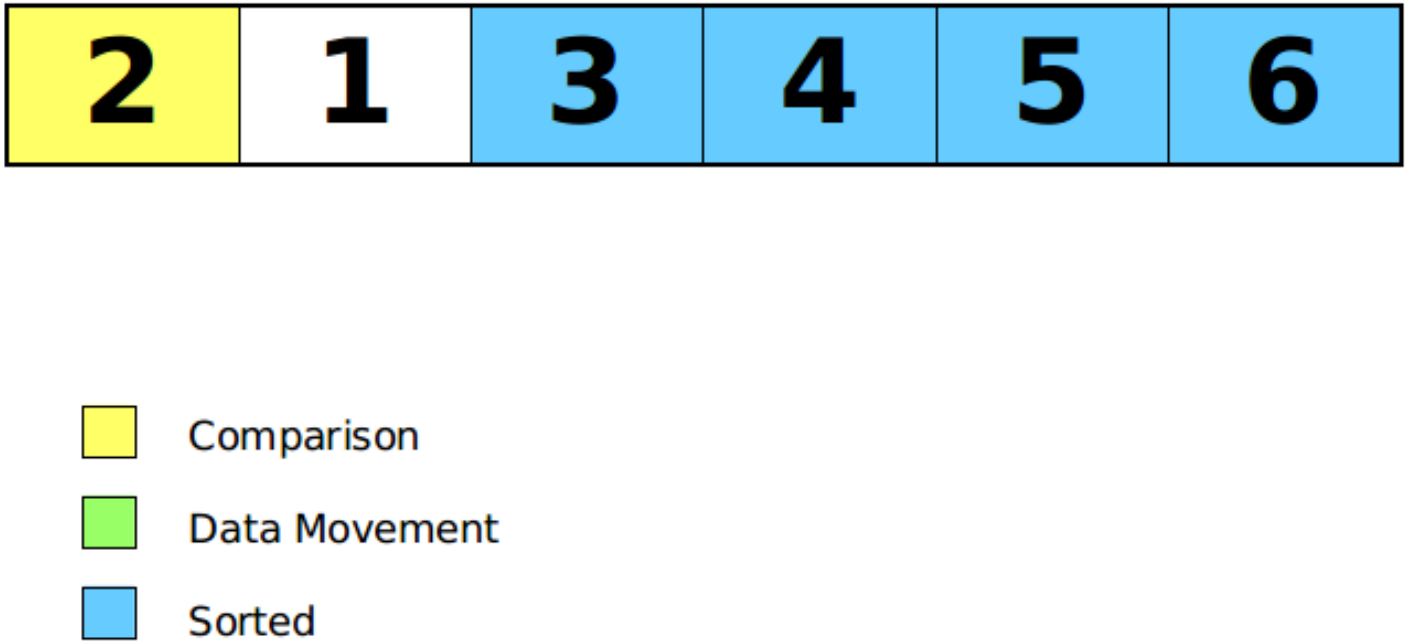


Selection Sort

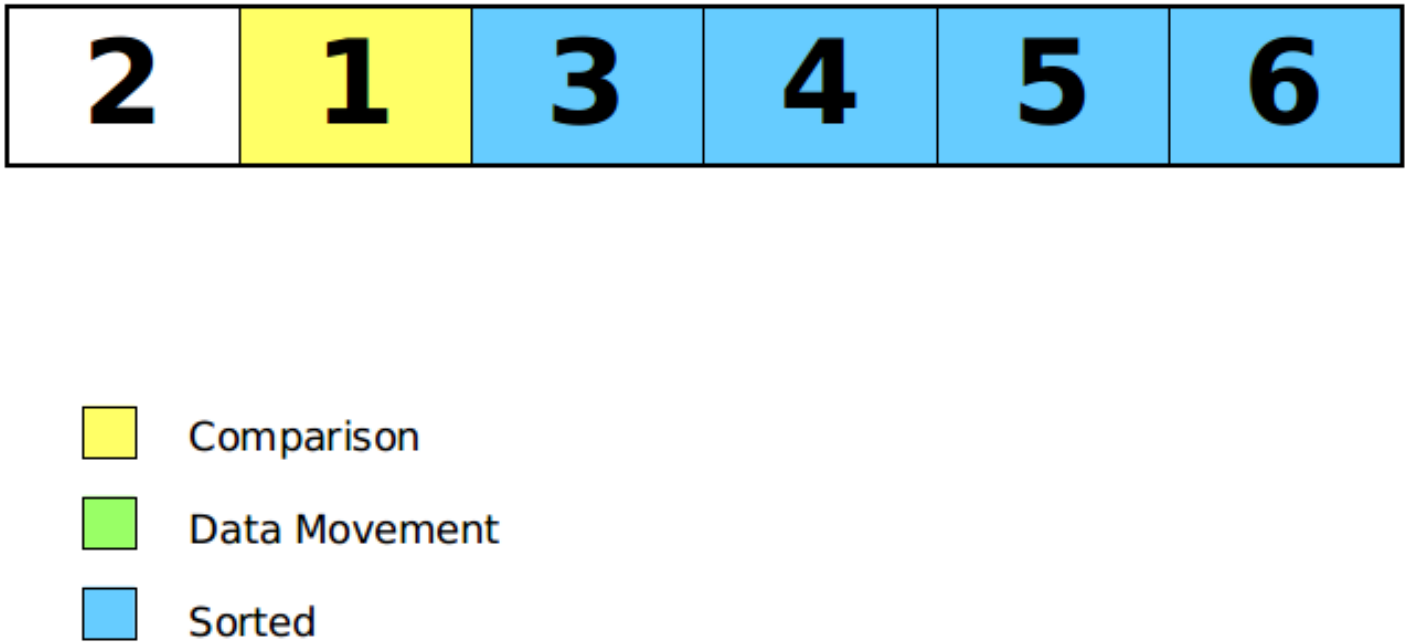
2	1	3	4	5	6

- Comparison
- Data Movement
- Sorted

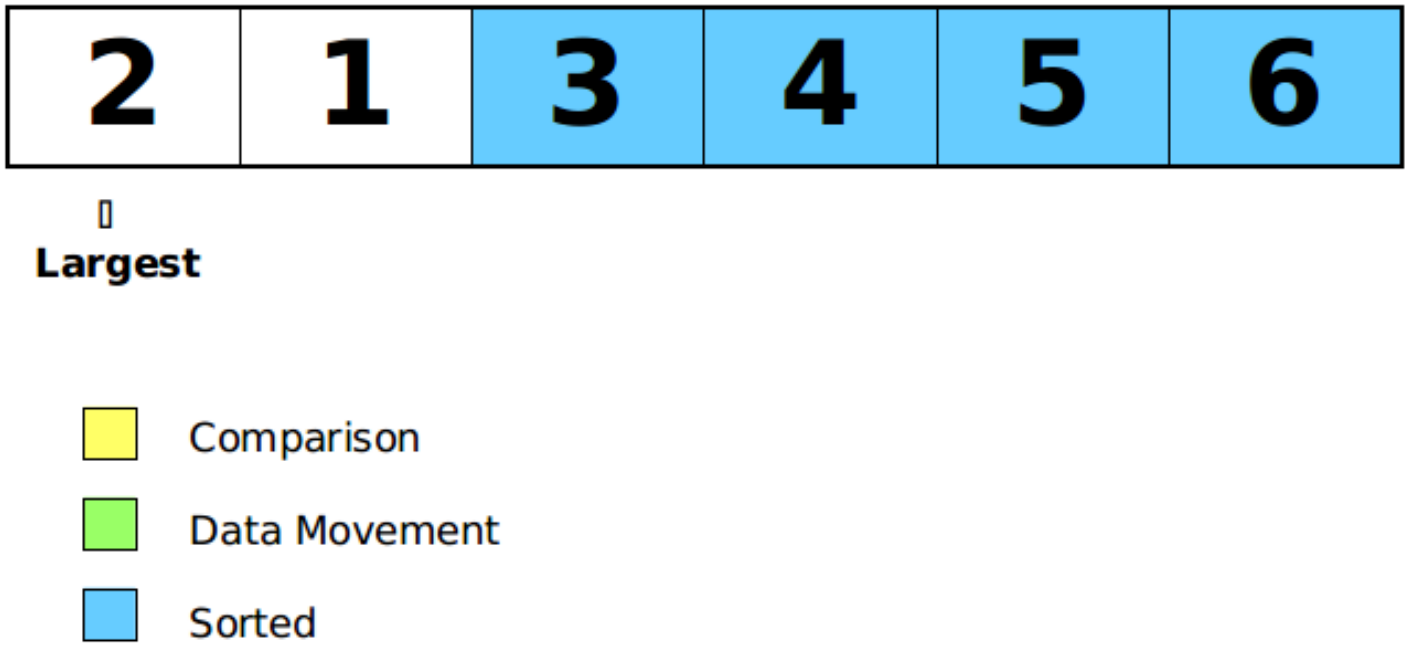
Selection Sort



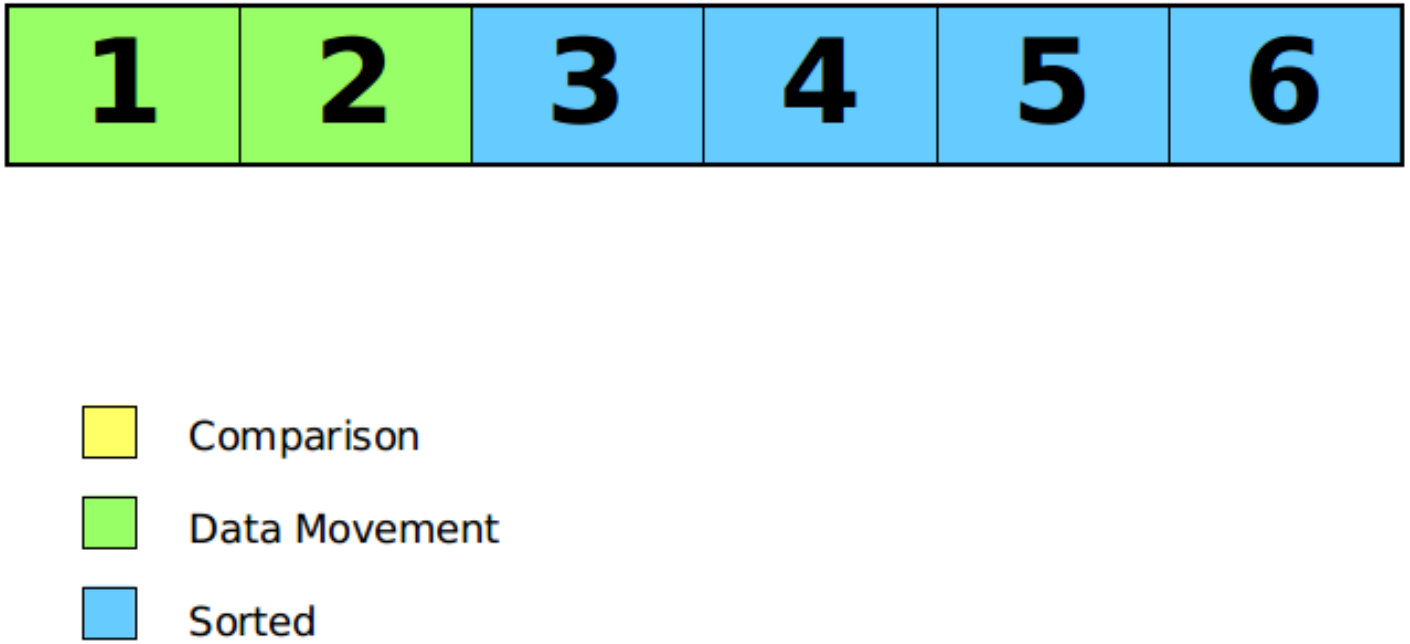
Selection Sort



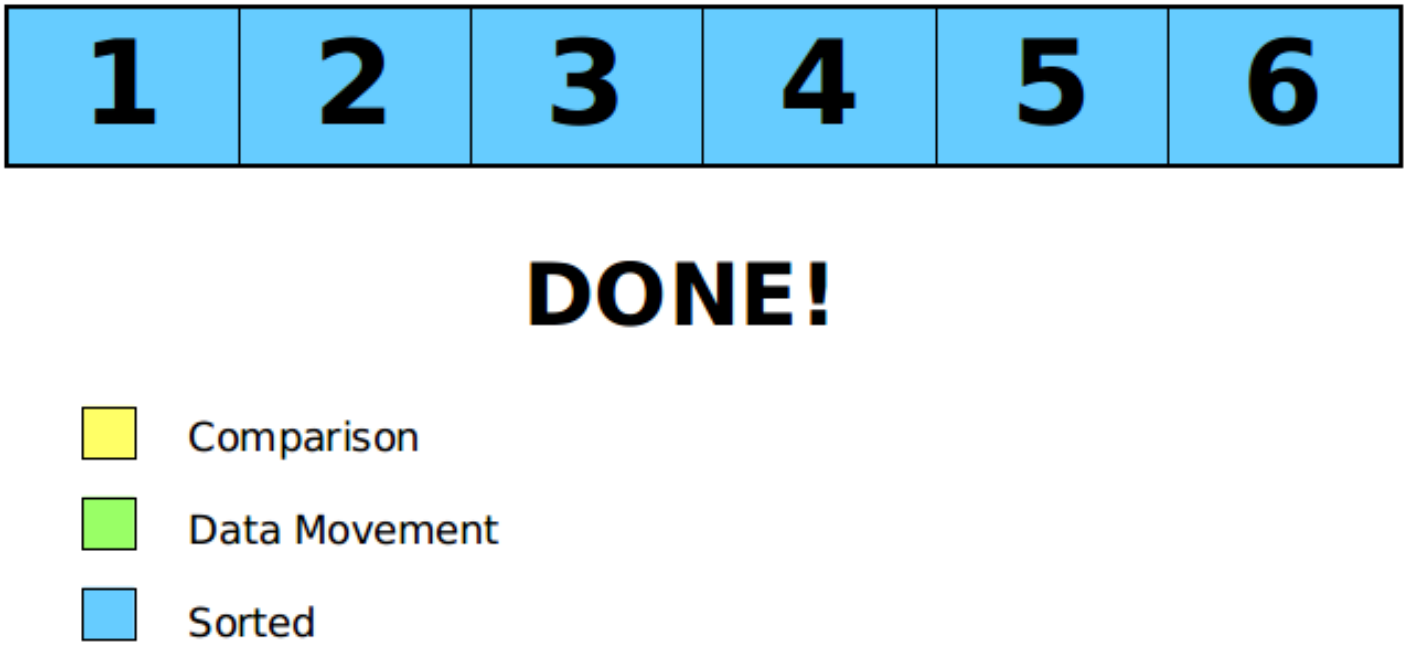
Selection Sort



Selection Sort



Selection Sort



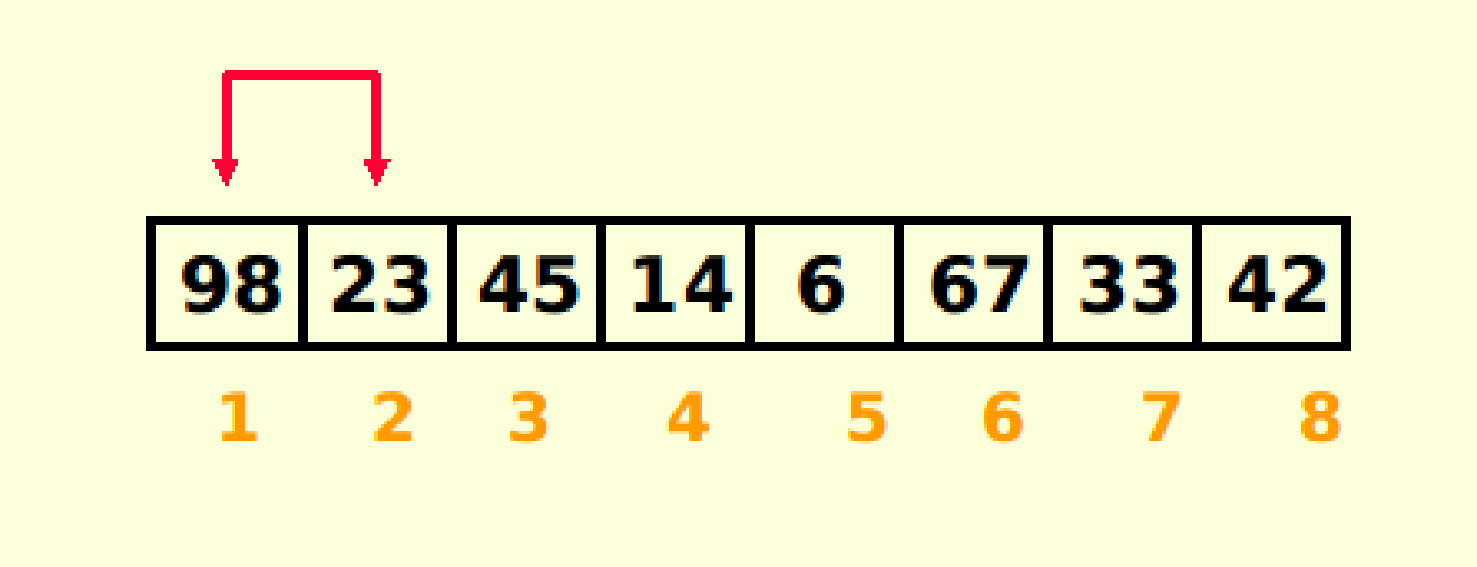
Bubble Sort

- It is a popular and simple algorithm for sorting data.
- This algorithm is not so efficient.
- **Iverson** was the first to use name "bubble sort" in 1962, even though used earlier.
- Unfortunately it is commonly used where the number of elements is too large.
- **Procedure:**
  - Starts at one end of the list and make repeated scans through the list comparing successive pairs of elements.
  - If the first element is larger than the second, called an "inversion", then the values are swapped.
  - Each scan will push the maximum element to the top.
  - This is the "bubbling" effect $\rightarrow$ name $\rightarrow$ bubble sort.
  - This process is continued until the list is sorted.
  - More swaps $\rightarrow$ More time for sorting.

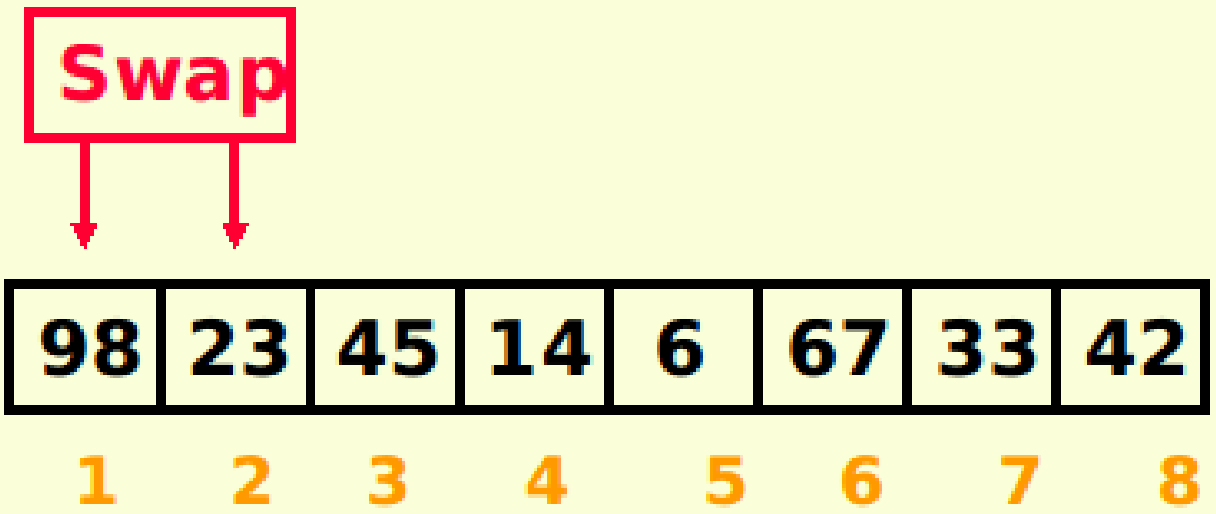
Bubble Sort Example:

98	23	45	14	6	67	33	42
1	2	3	4	5	6	7	8

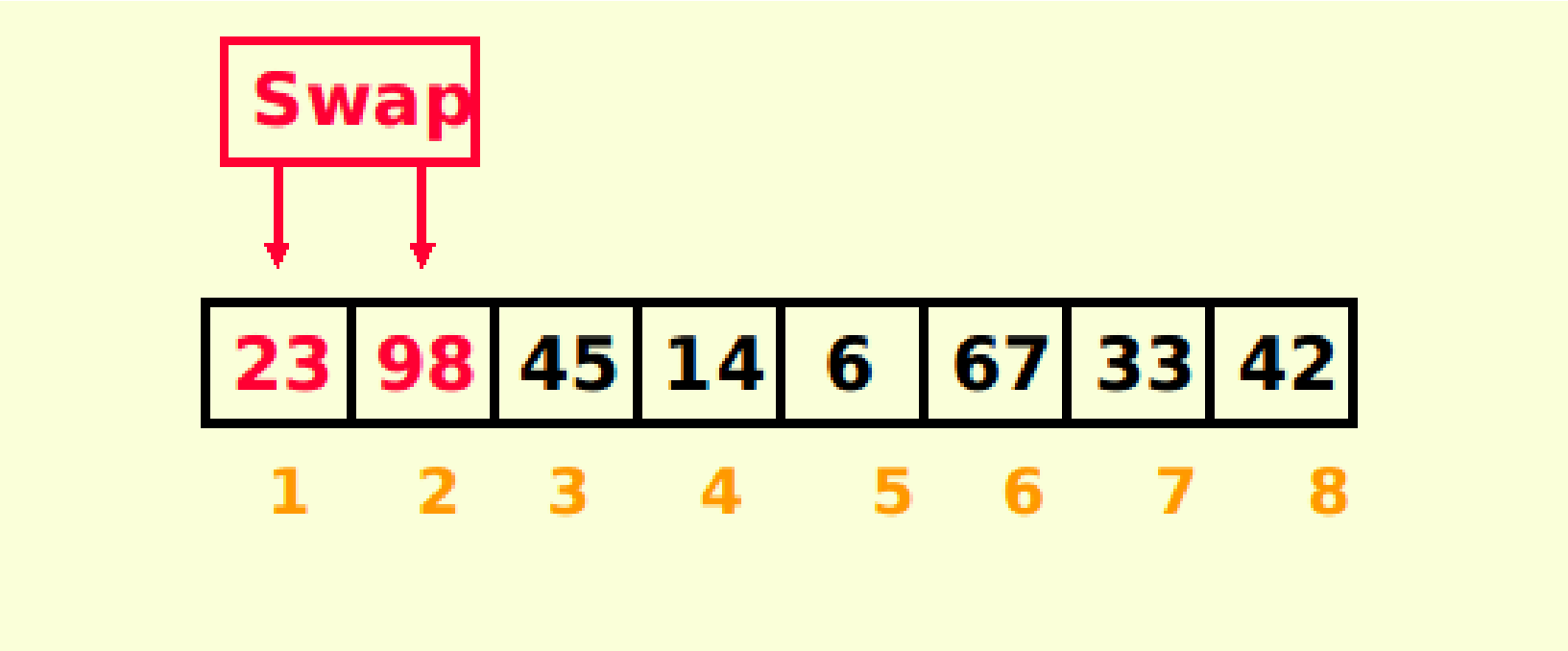
Bubble Sort Example:



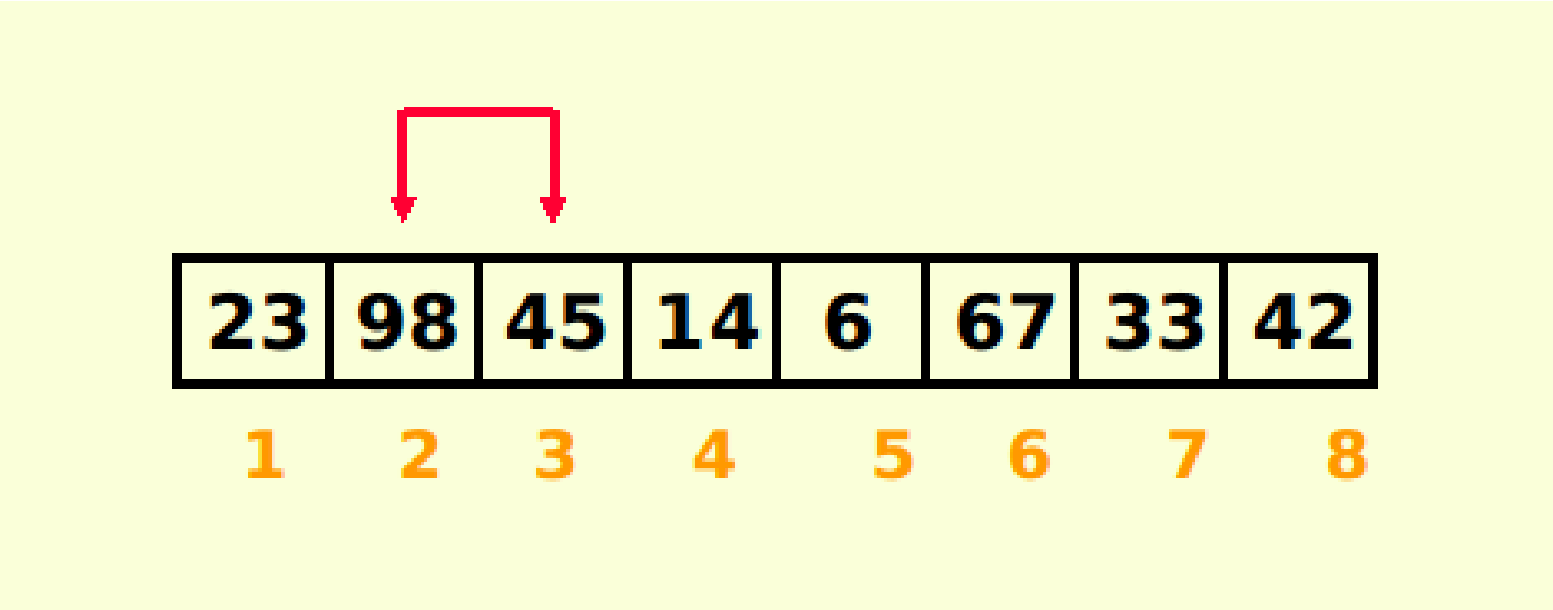
Bubble Sort Example:



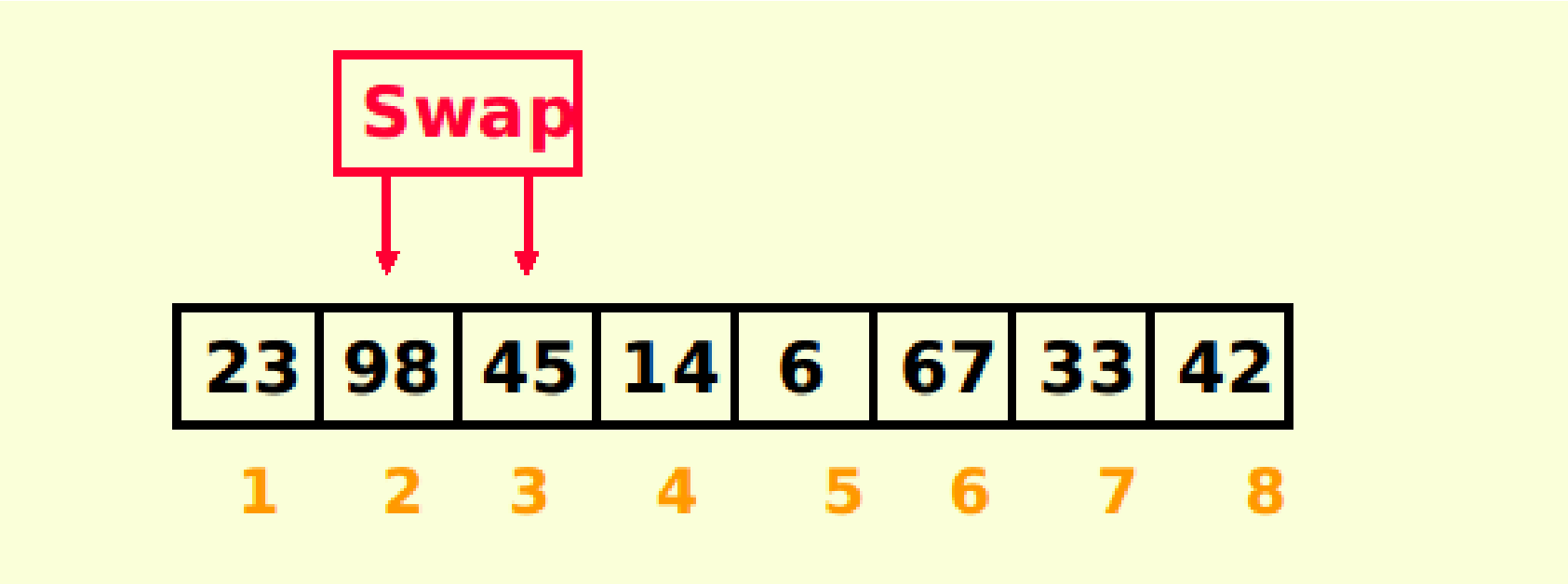
Bubble Sort Example:



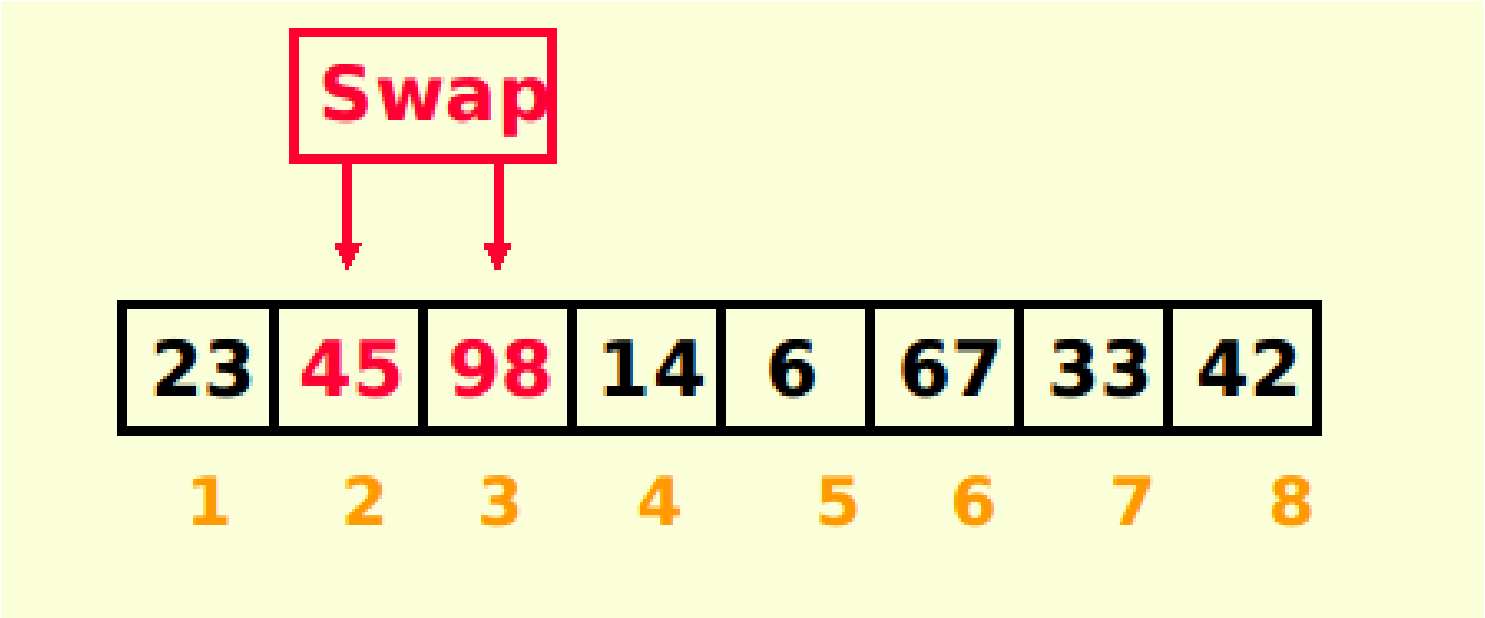
Bubble Sort Example:



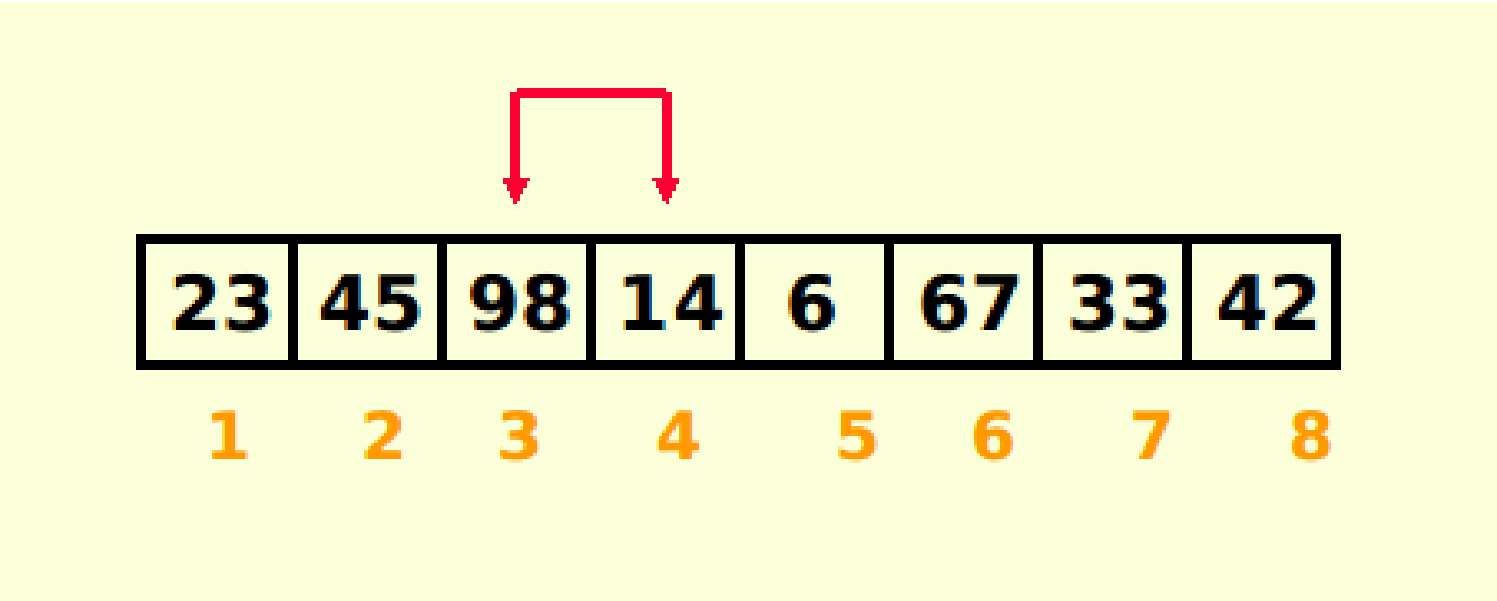
Bubble Sort Example:



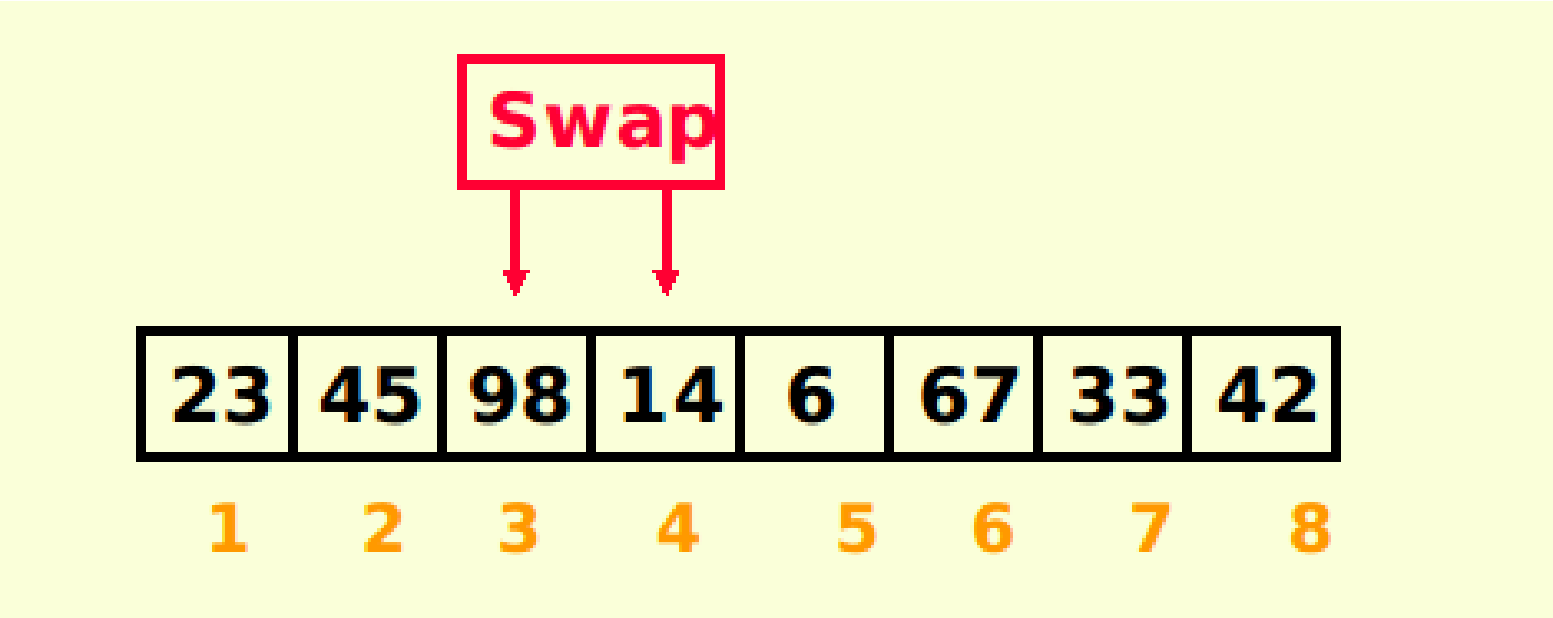
Bubble Sort Example:



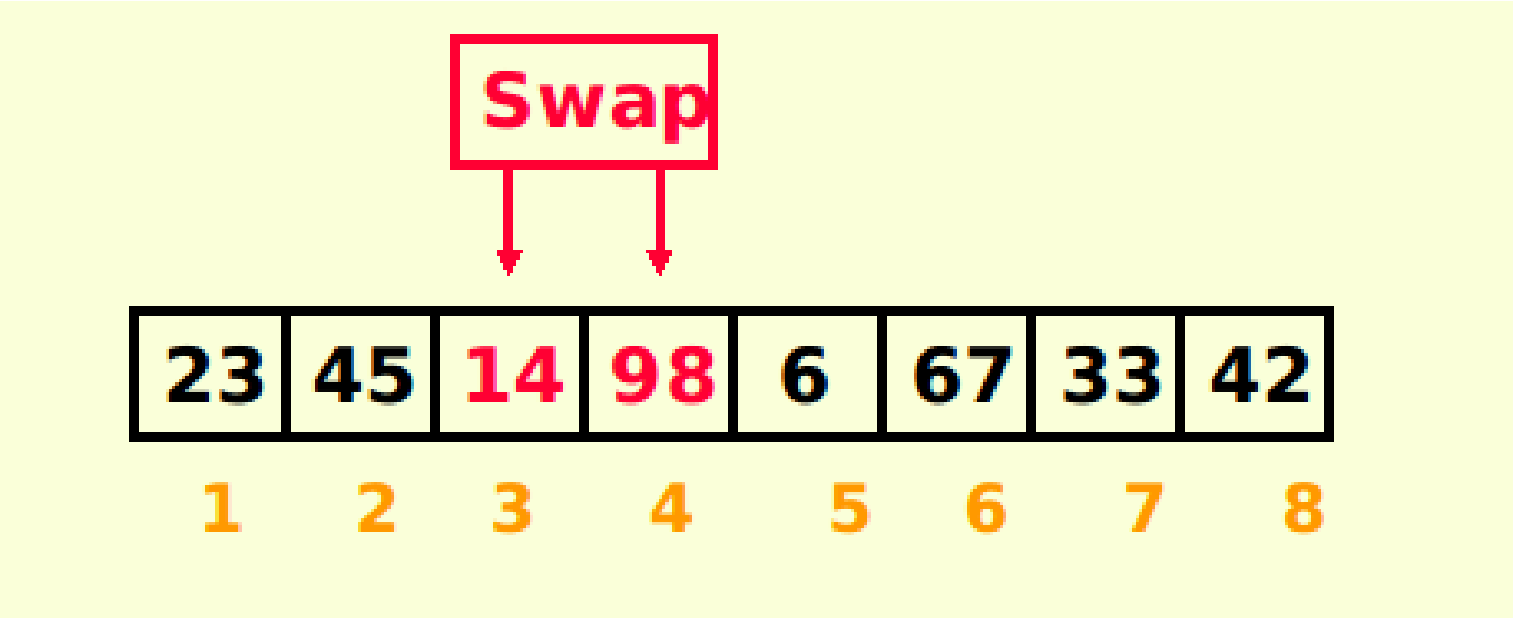
Bubble Sort Example:



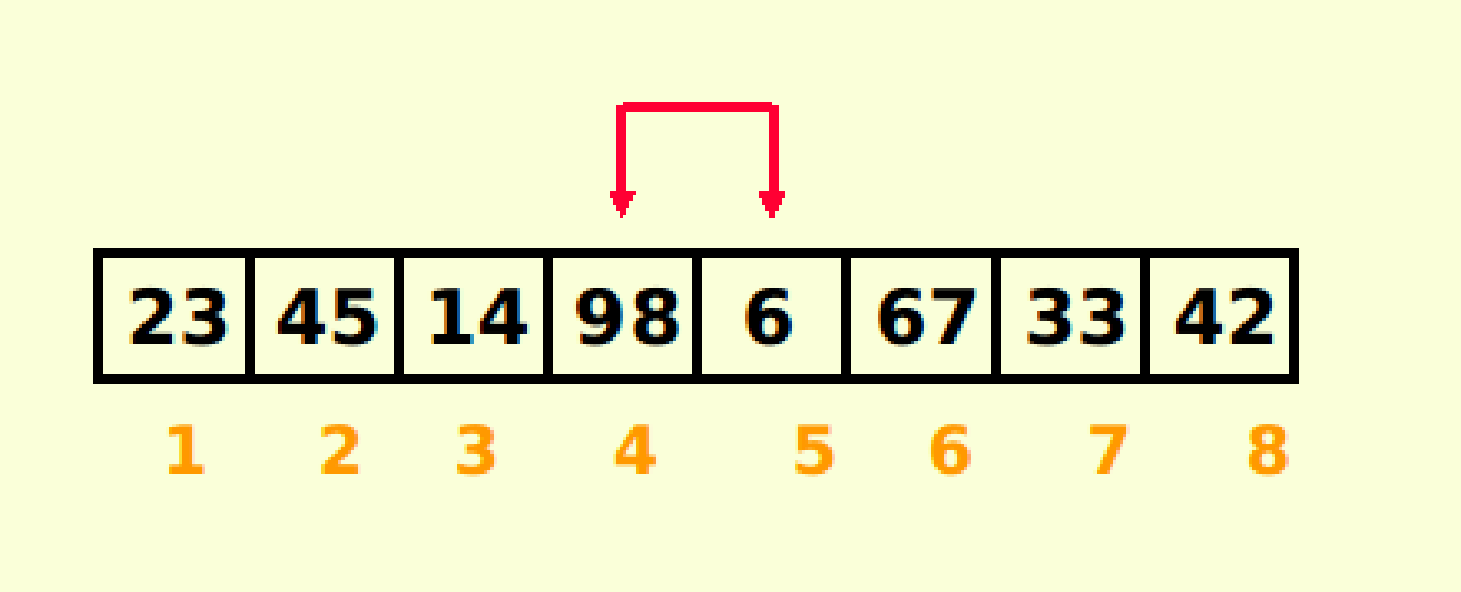
Bubble Sort Example:



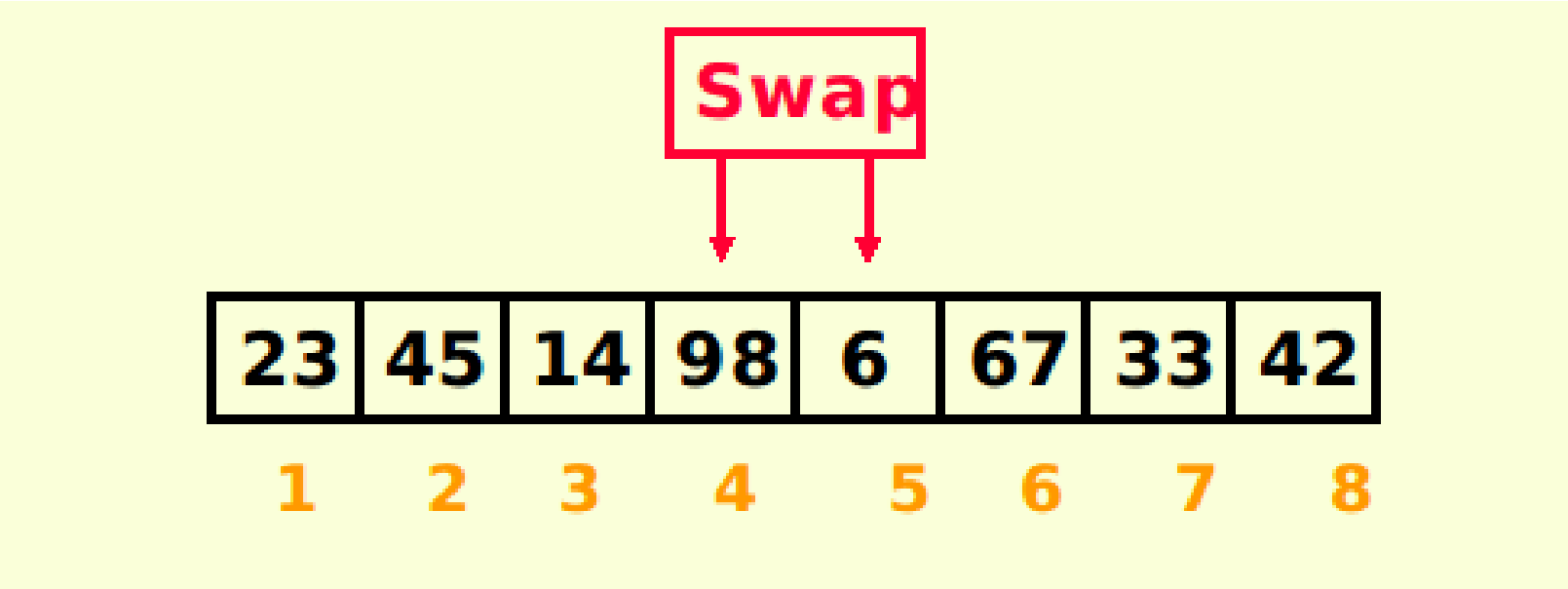
Bubble Sort Example:



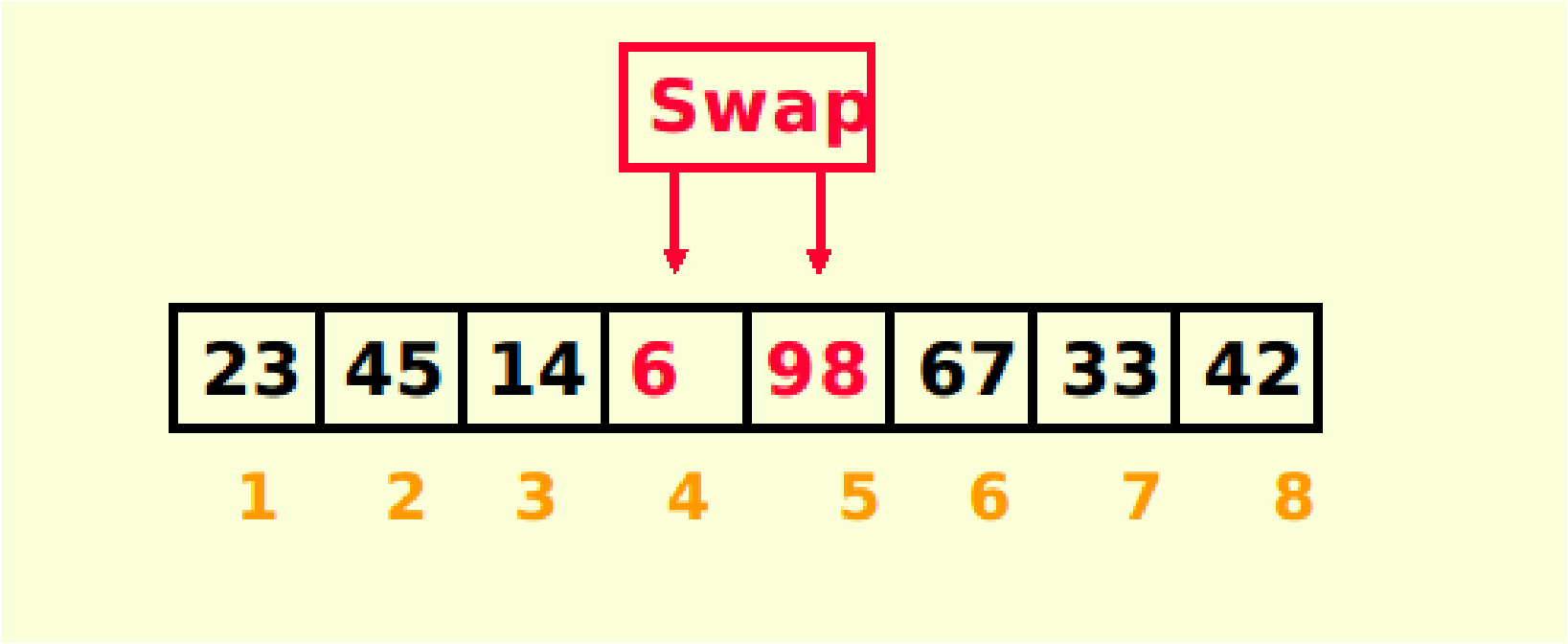
Bubble Sort Example:



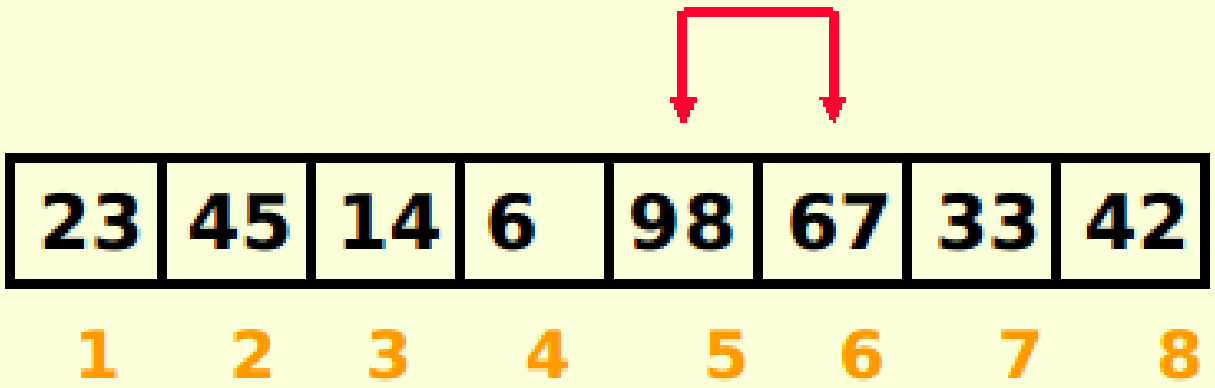
Bubble Sort Example:



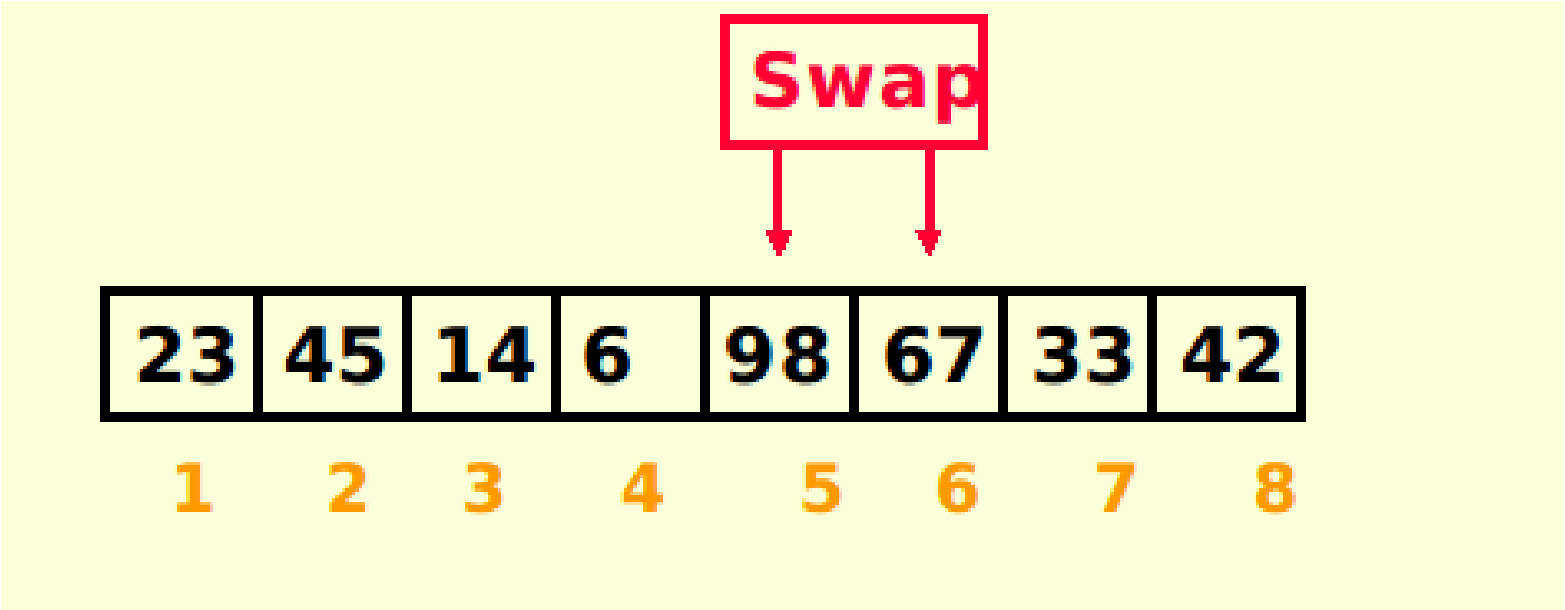
Bubble Sort Example:



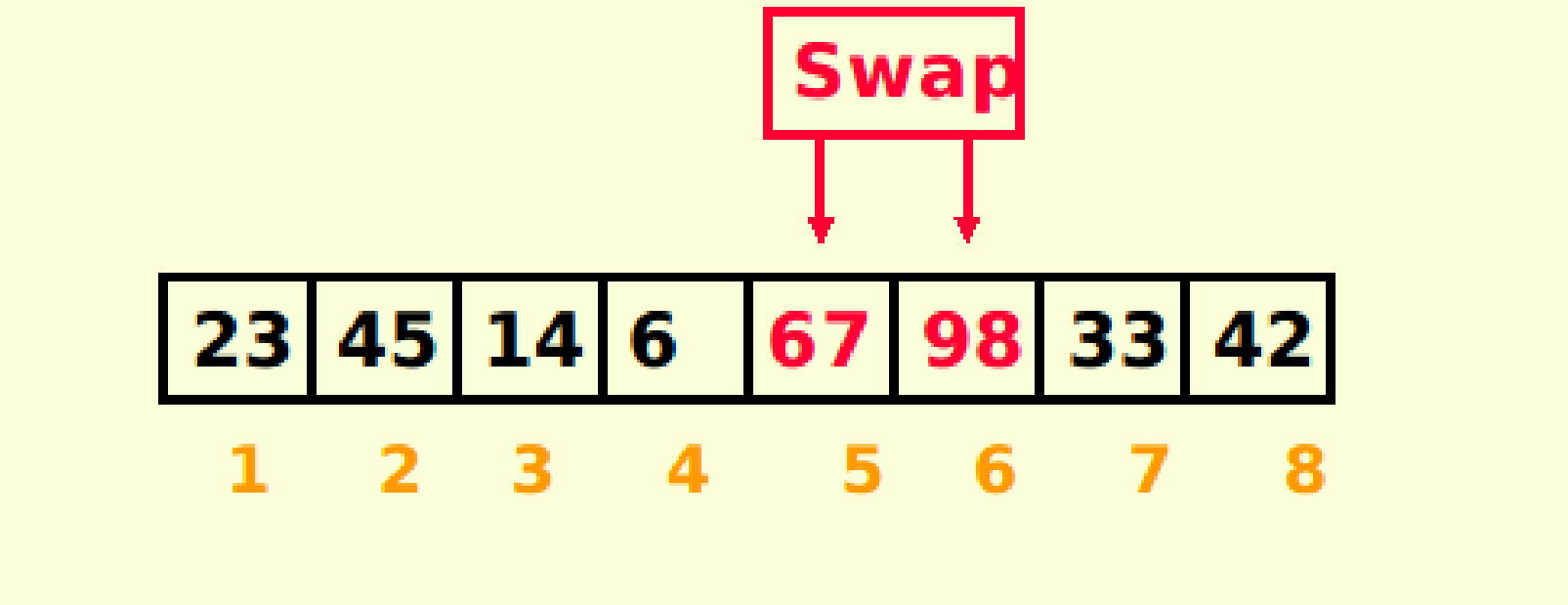
Bubble Sort Example:



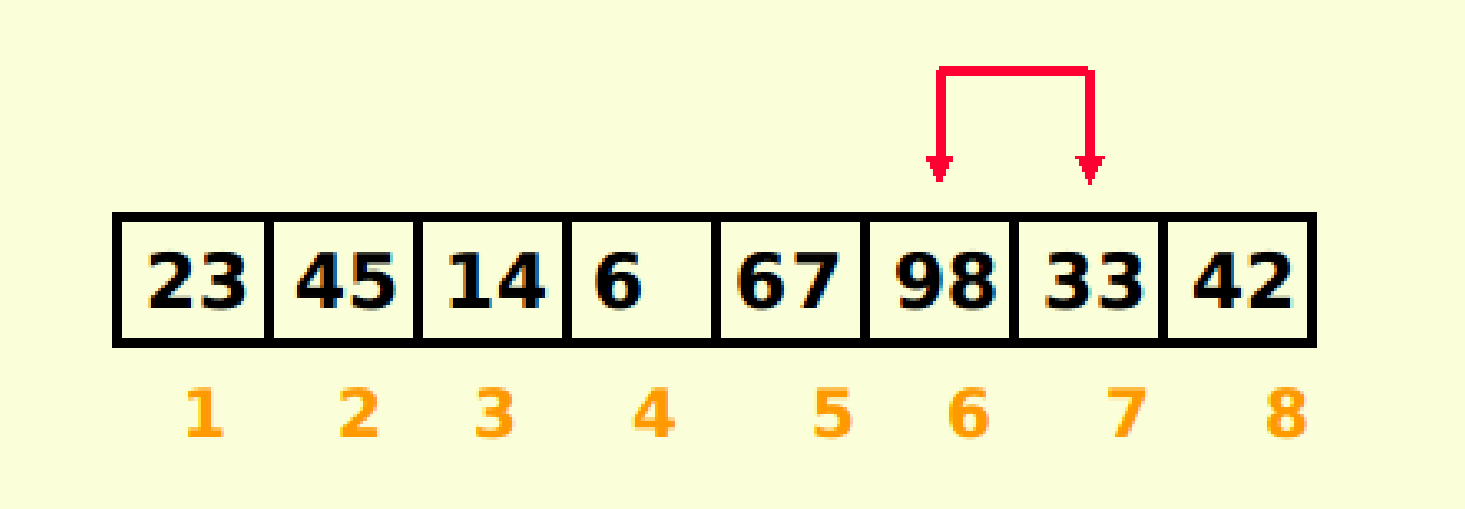
Bubble Sort Example:



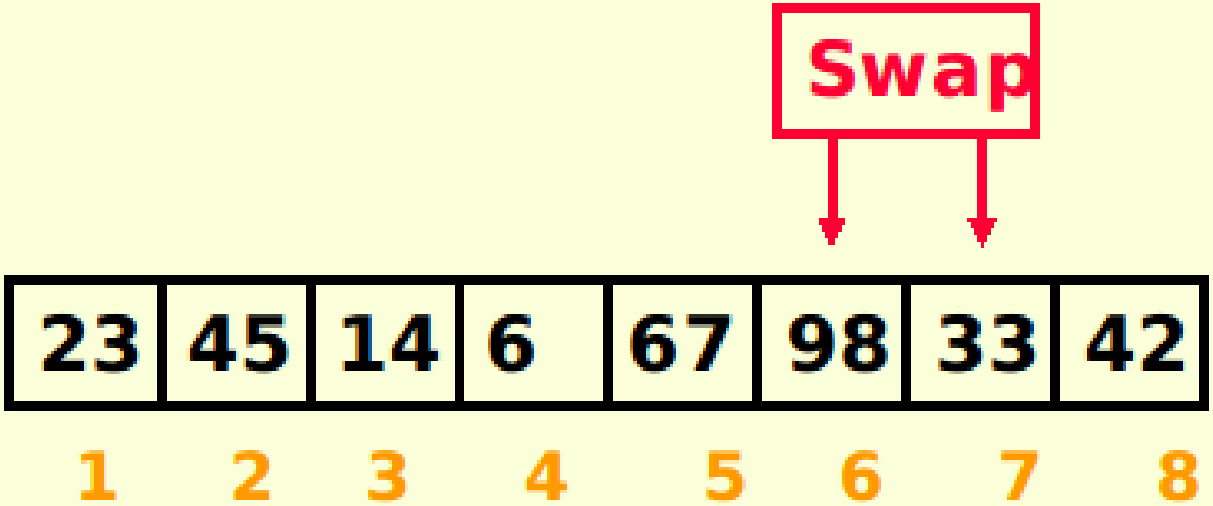
Bubble Sort Example:



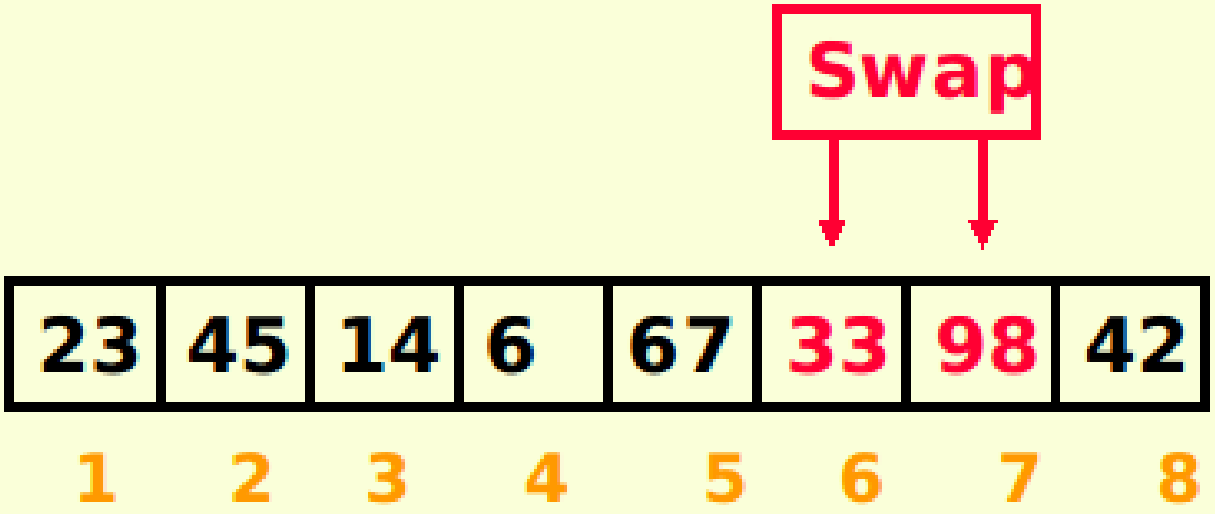
Bubble Sort Example:



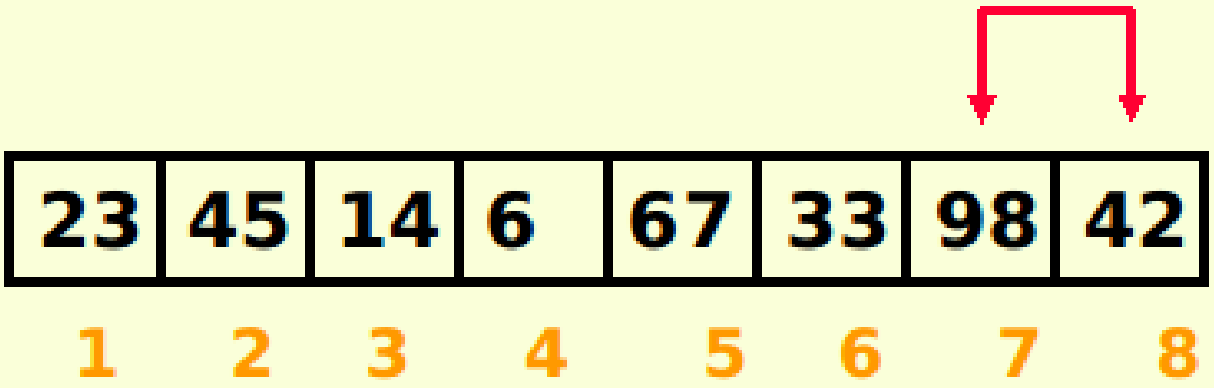
Bubble Sort Example:



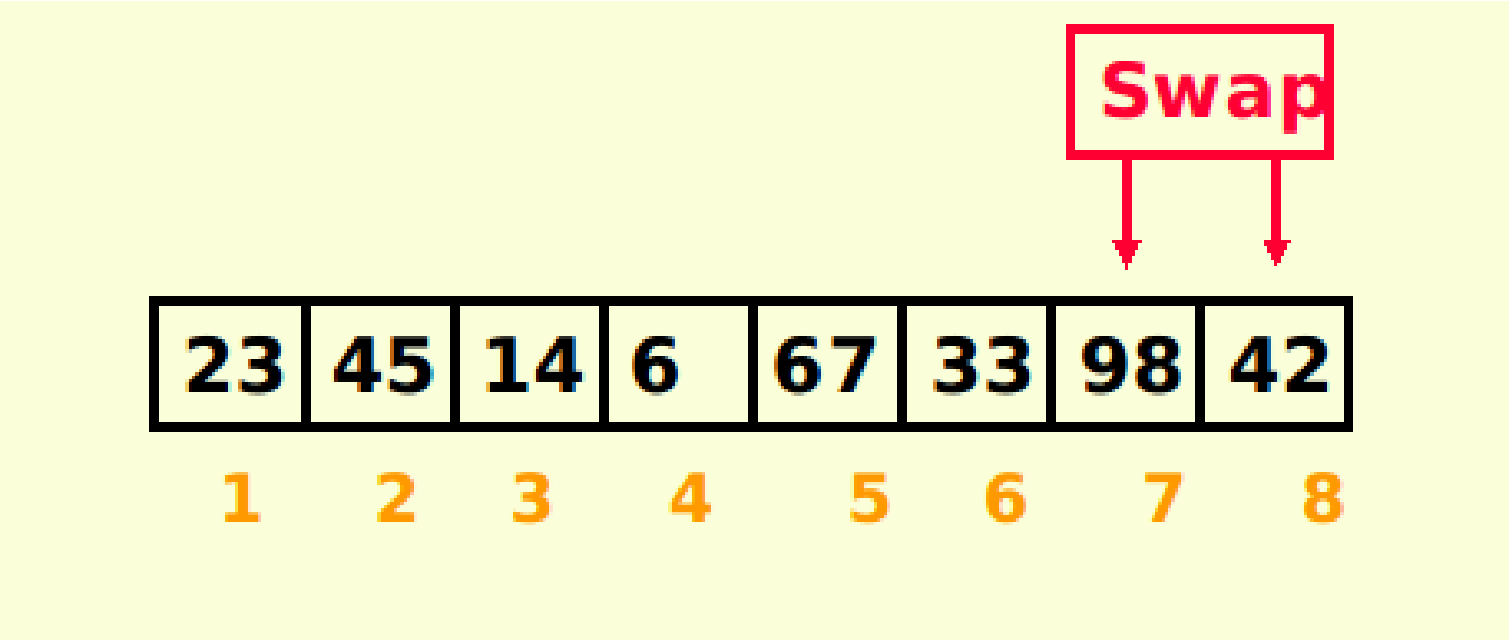
Bubble Sort Example:



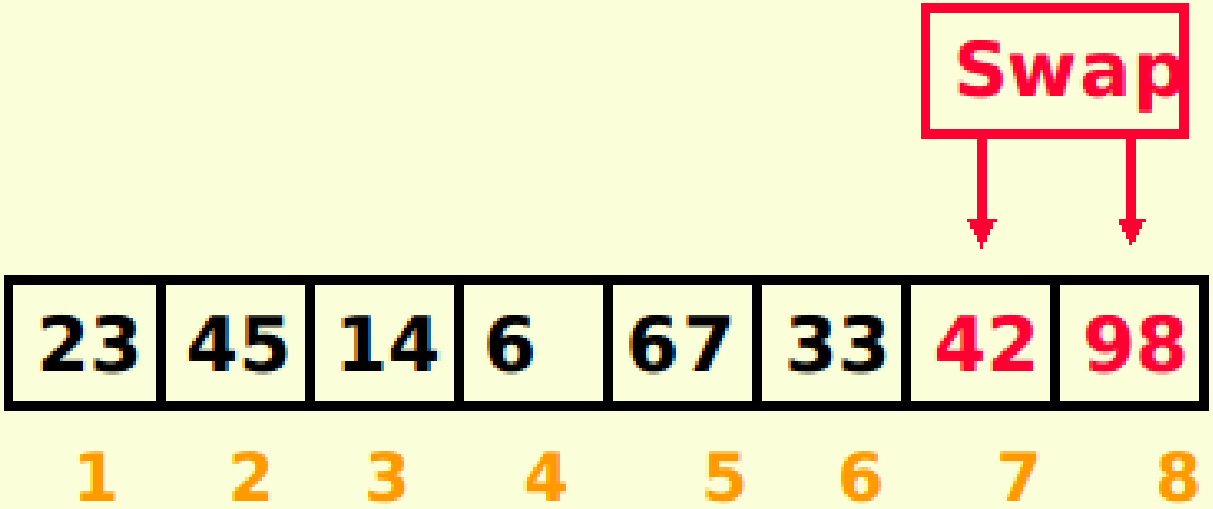
Bubble Sort Example:



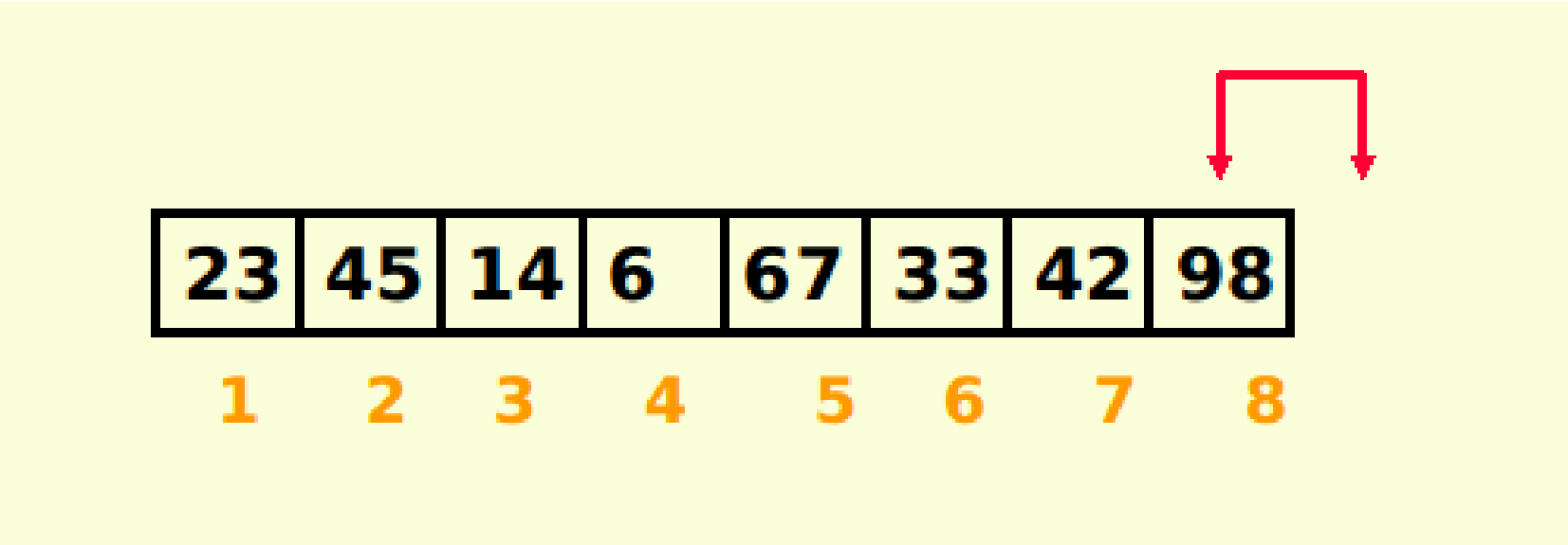
Bubble Sort Example:



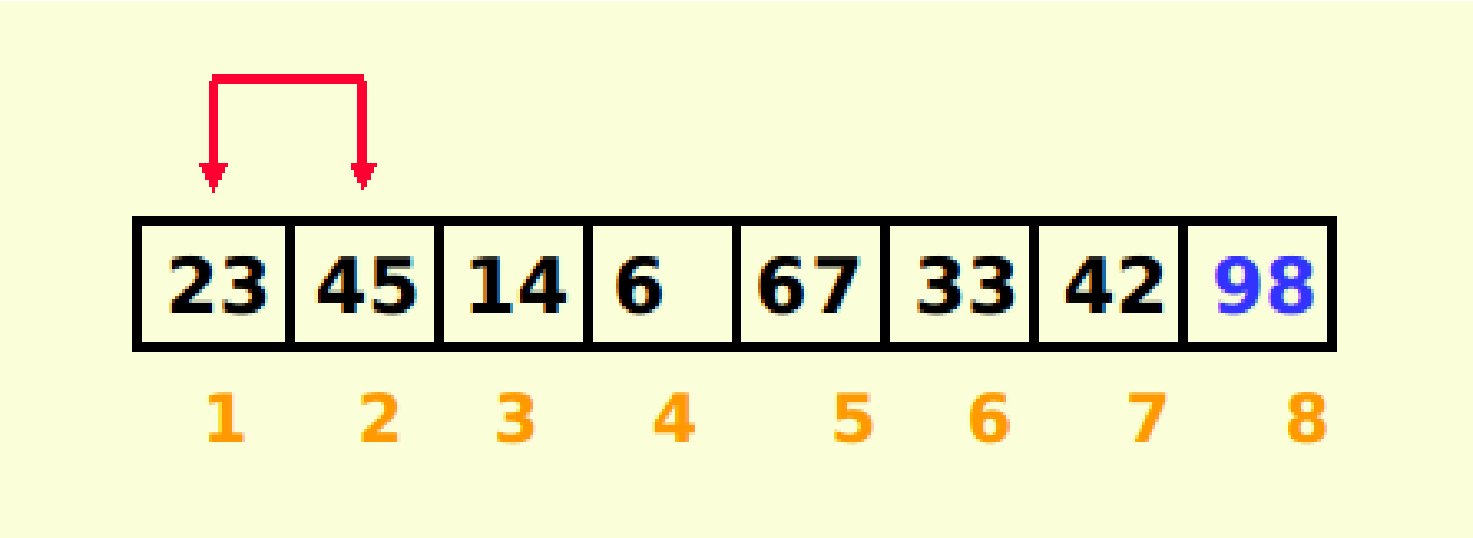
Bubble Sort Example:



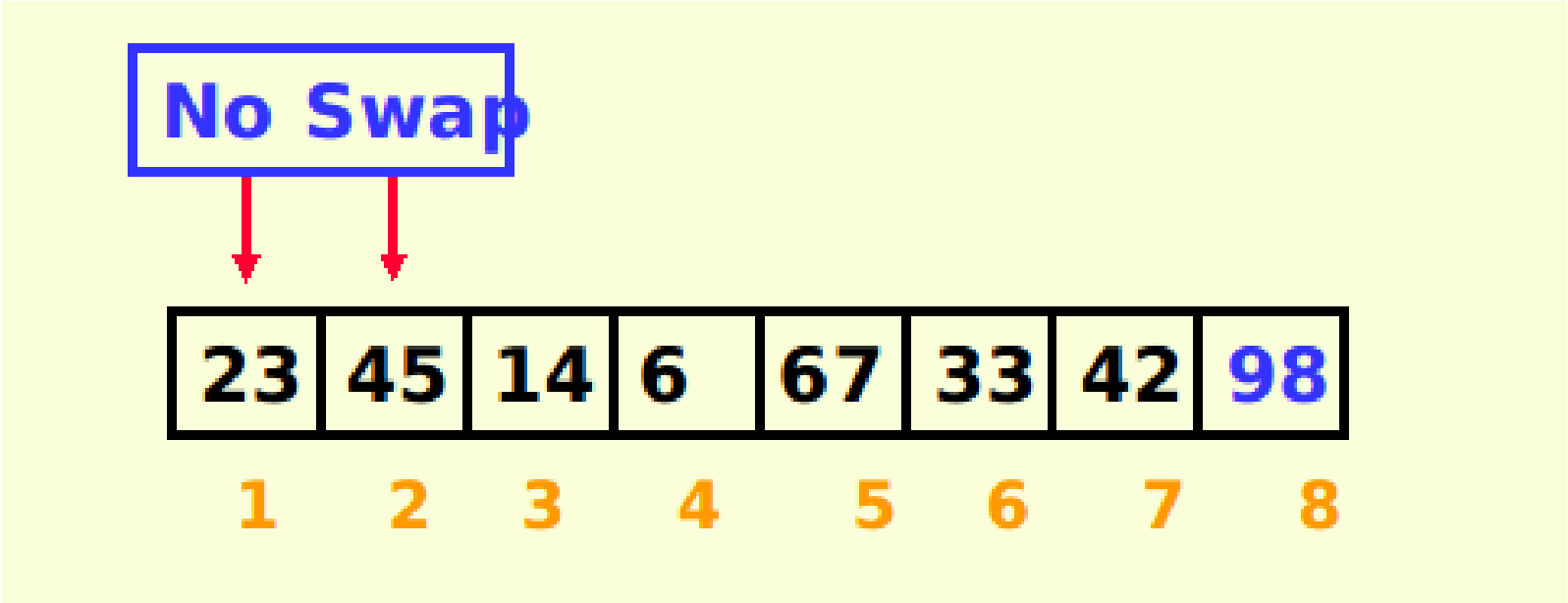
Bubble Sort Example:



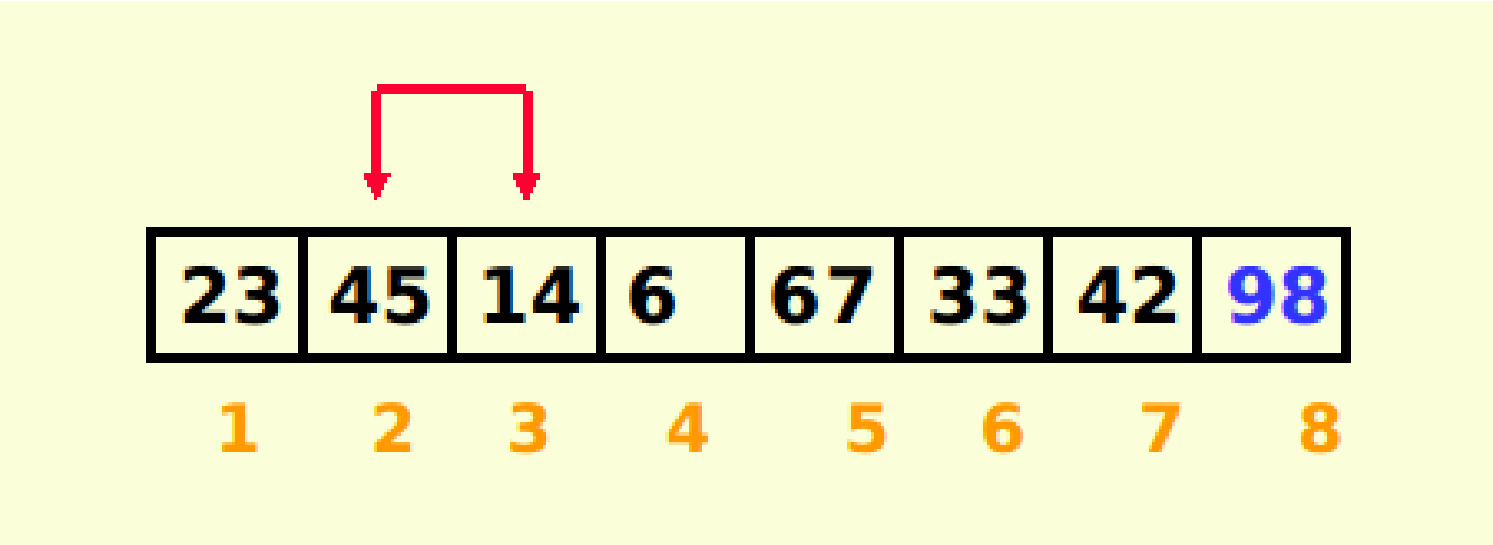
Bubble Sort Example:



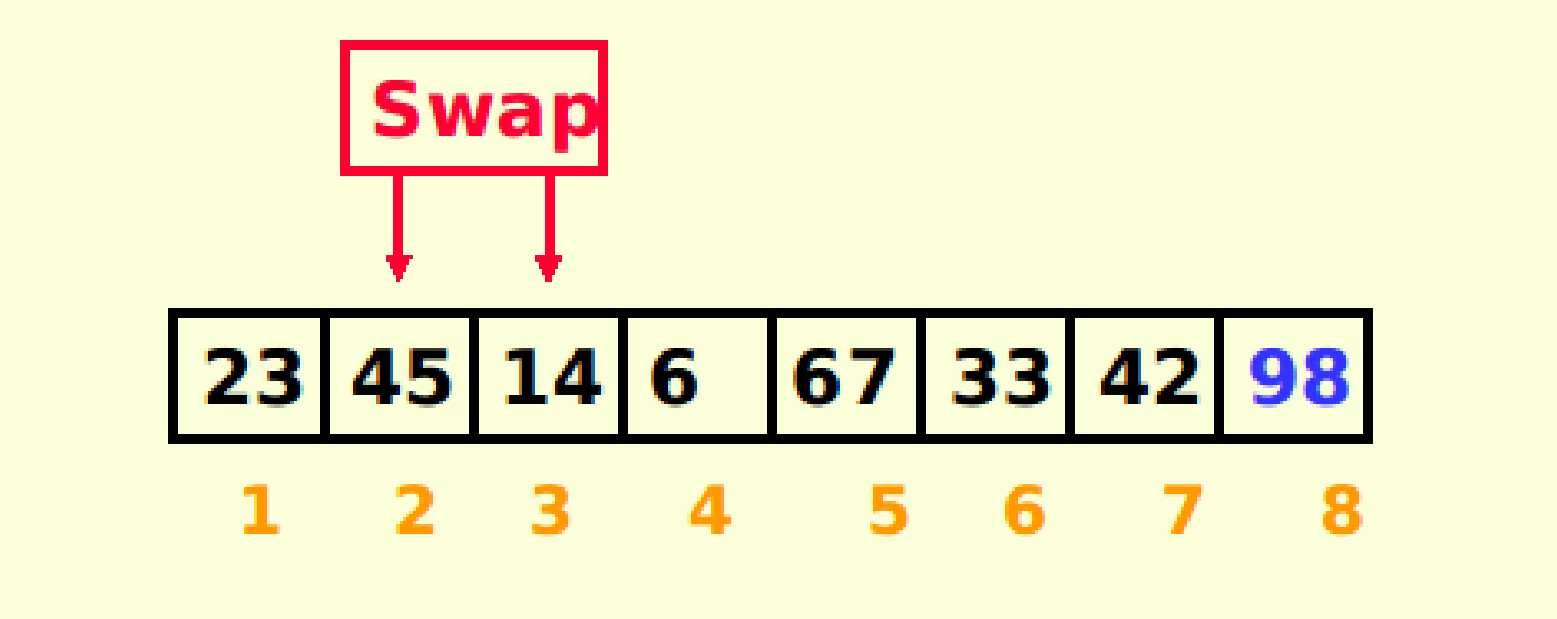
Bubble Sort Example:



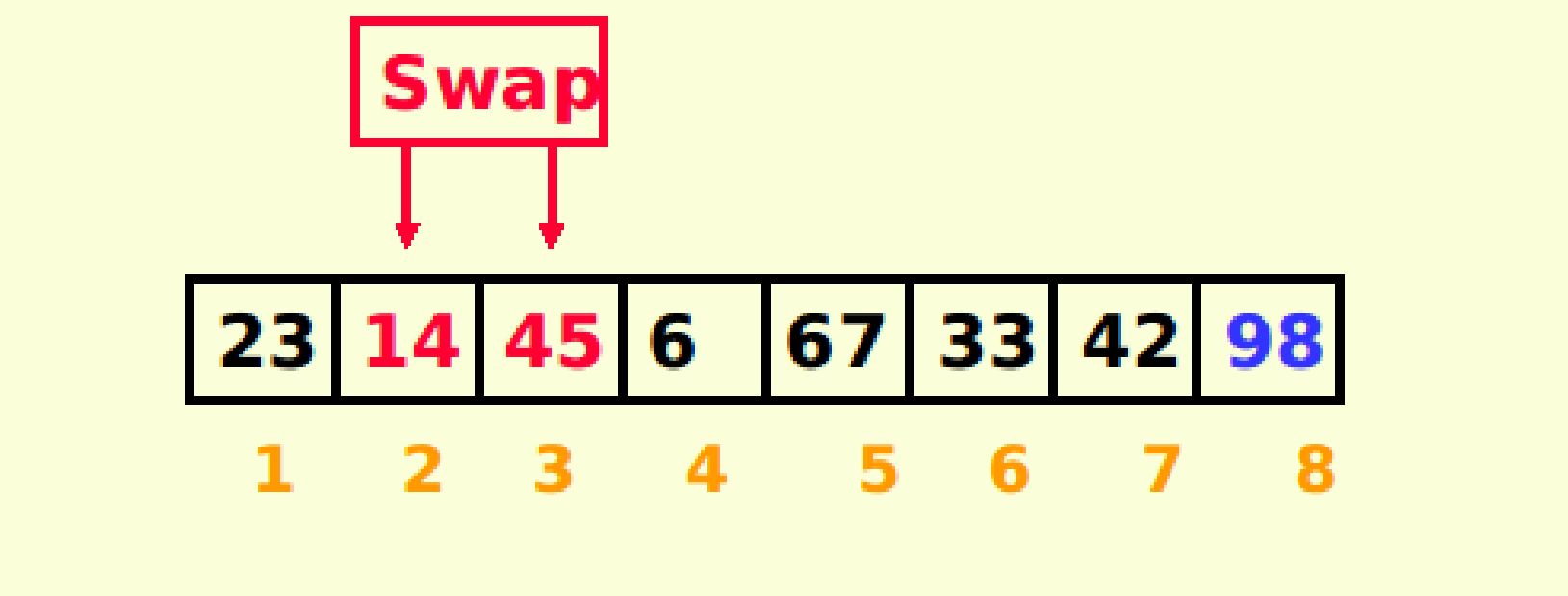
Bubble Sort Example:



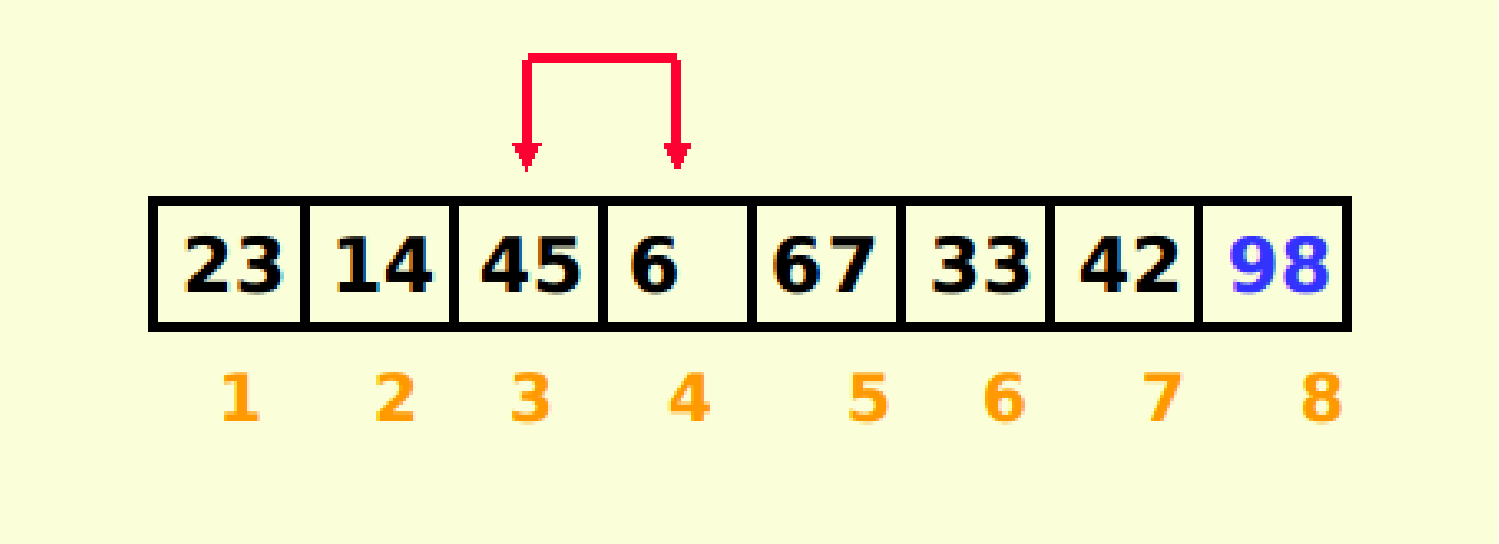
Bubble Sort Example:



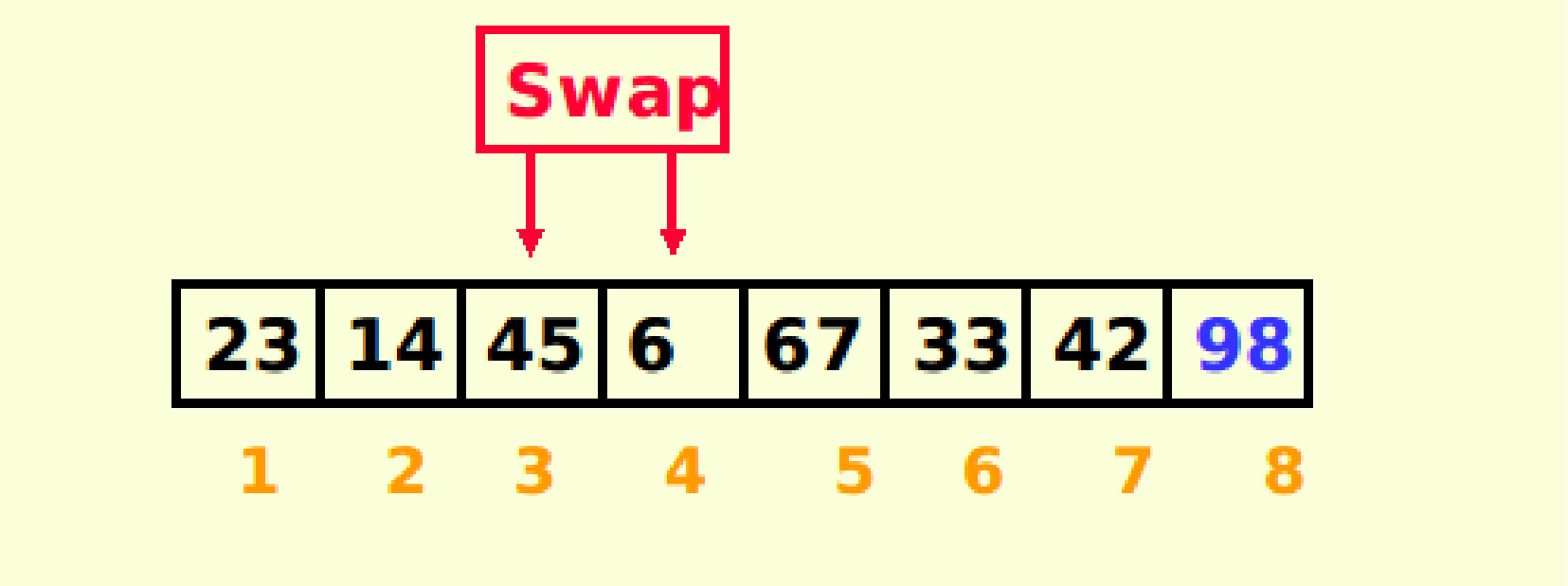
Bubble Sort Example:



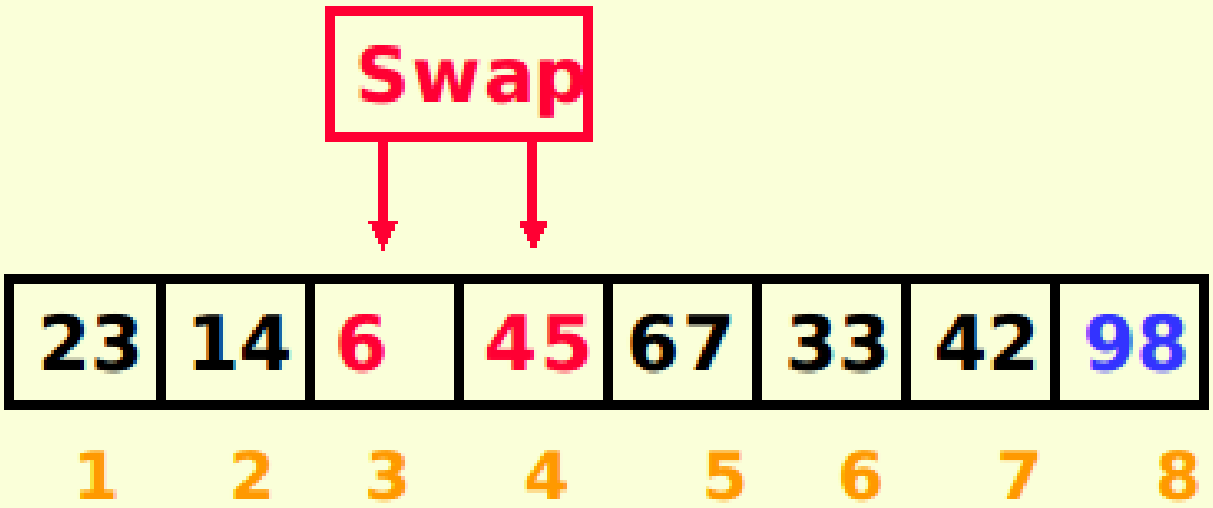
Bubble Sort Example:



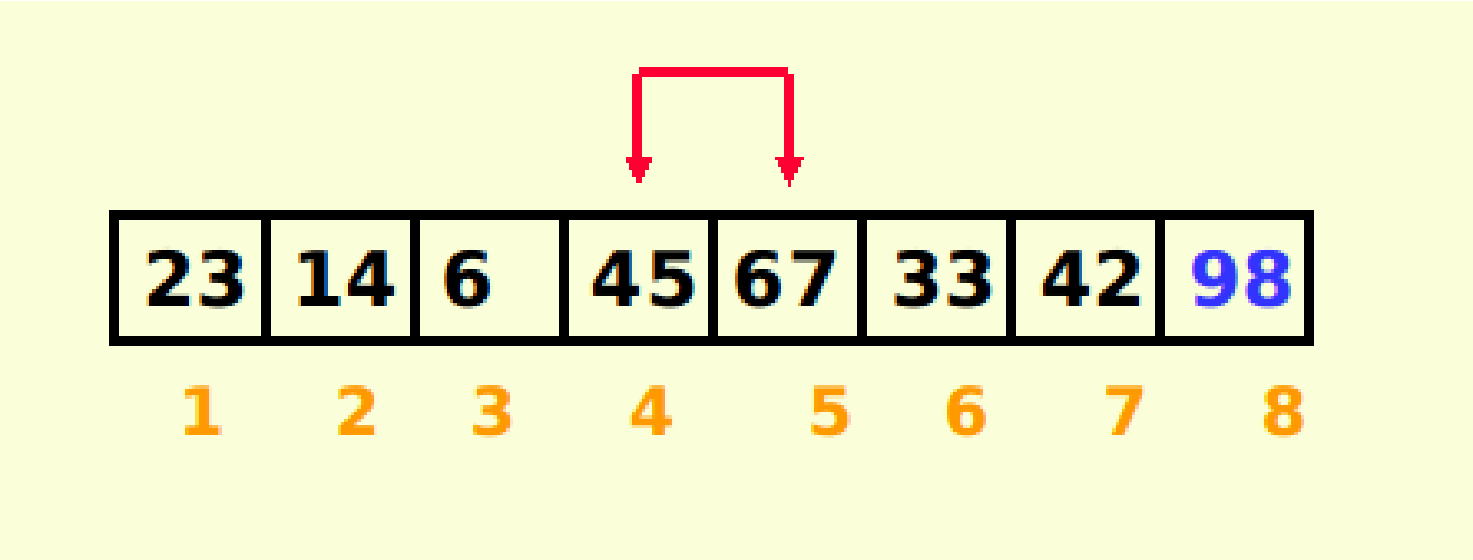
Bubble Sort Example:



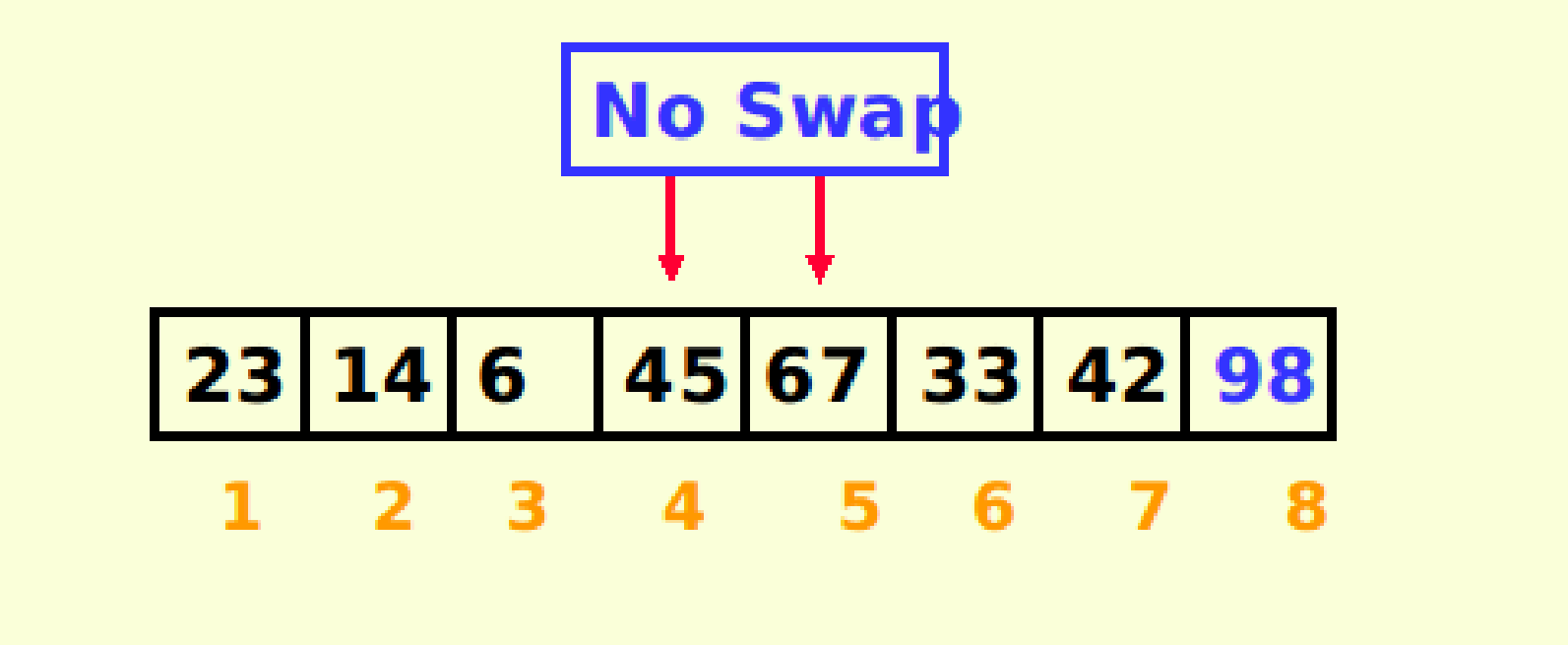
Bubble Sort Example:



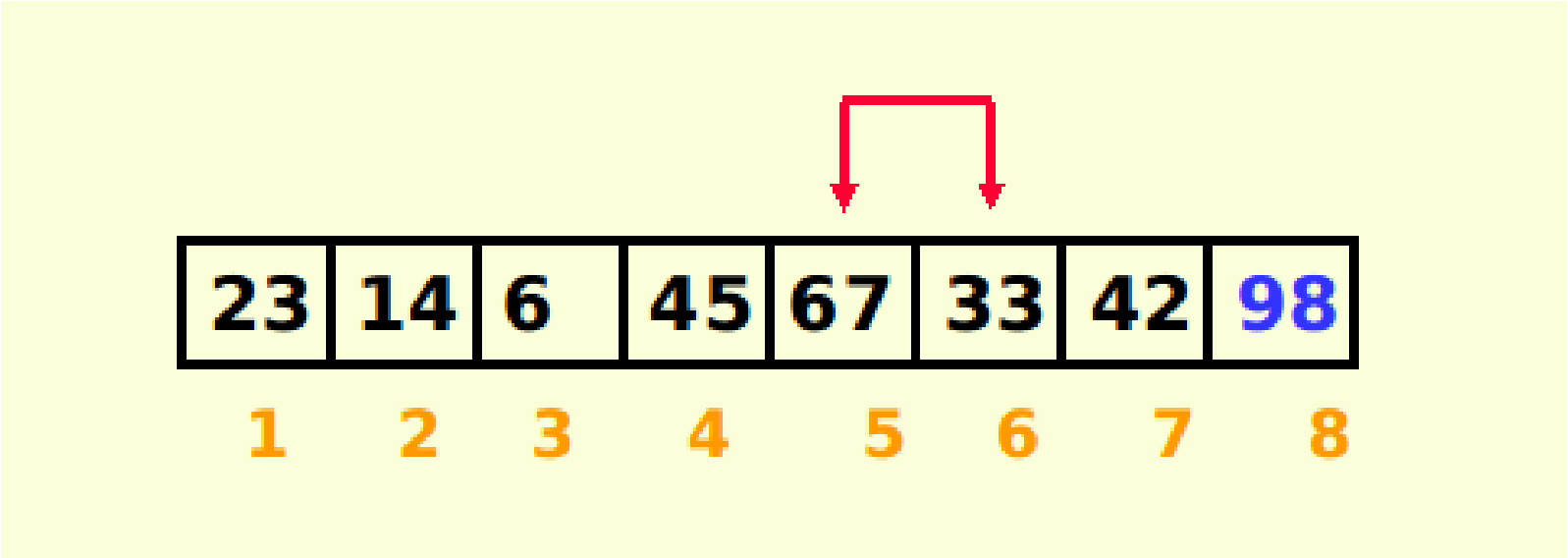
Bubble Sort Example:



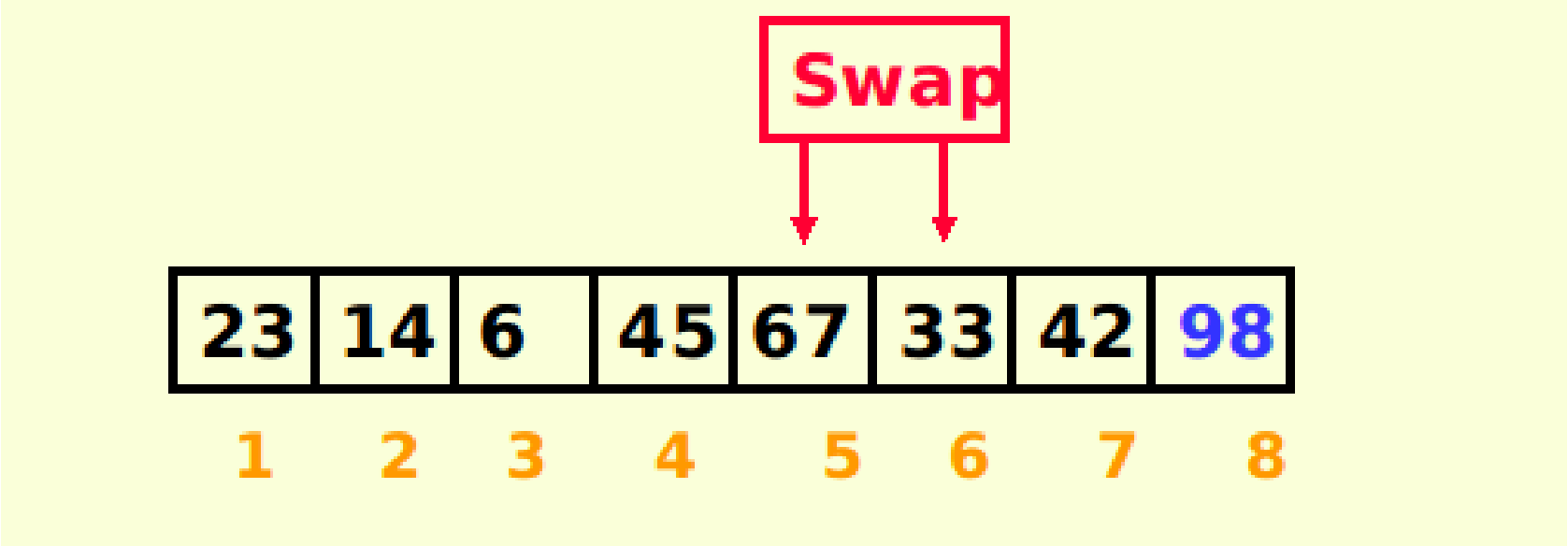
Bubble Sort Example:



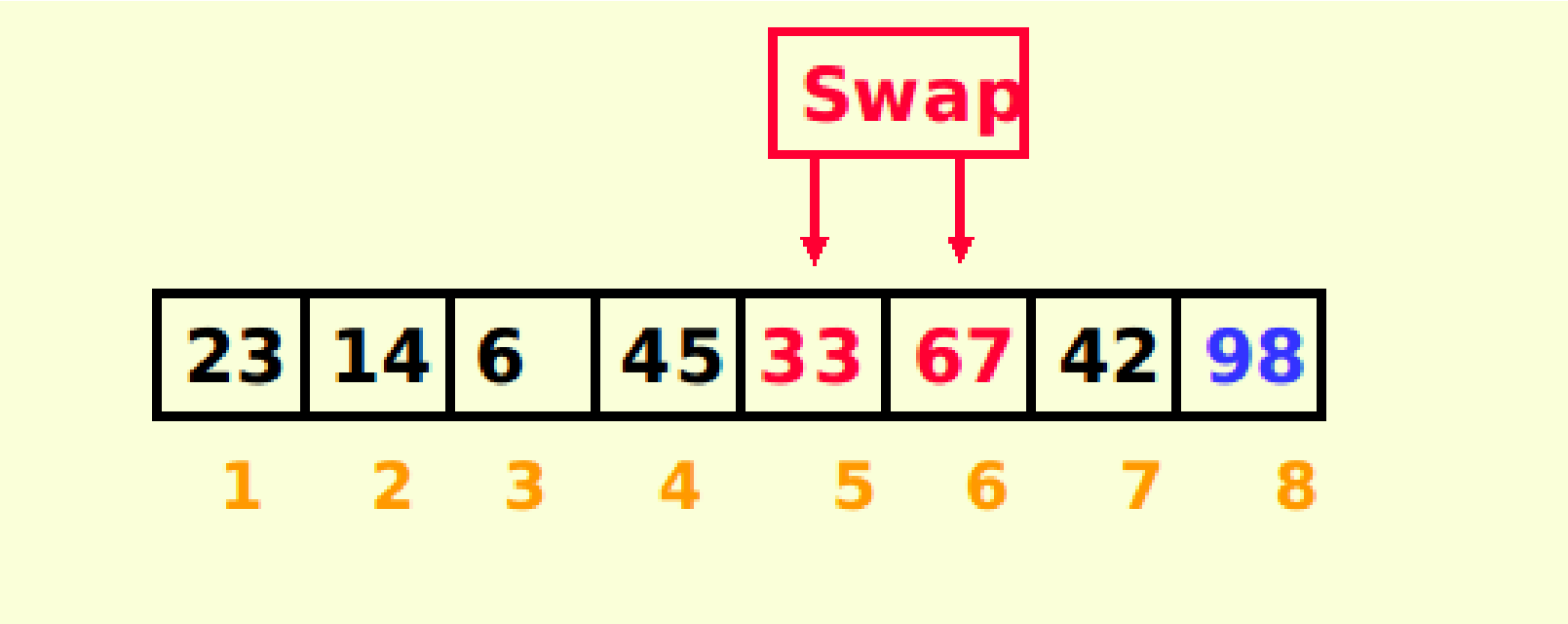
Bubble Sort Example:



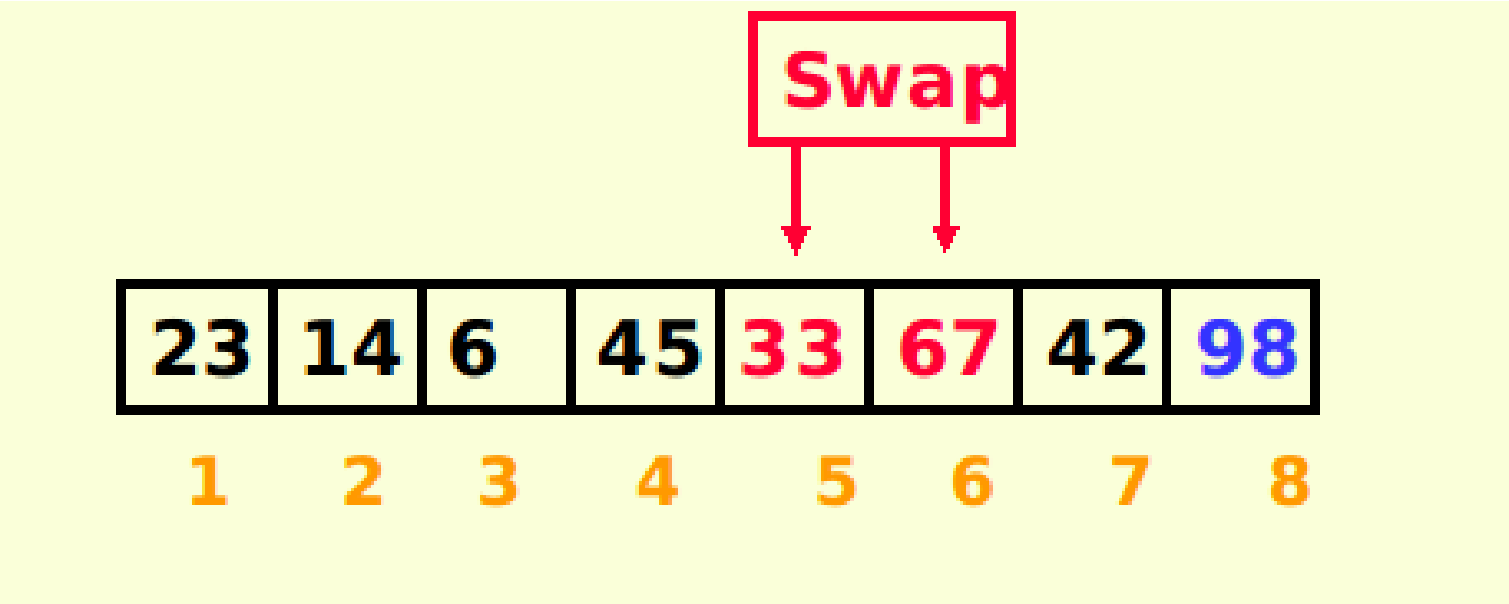
Bubble Sort Example:



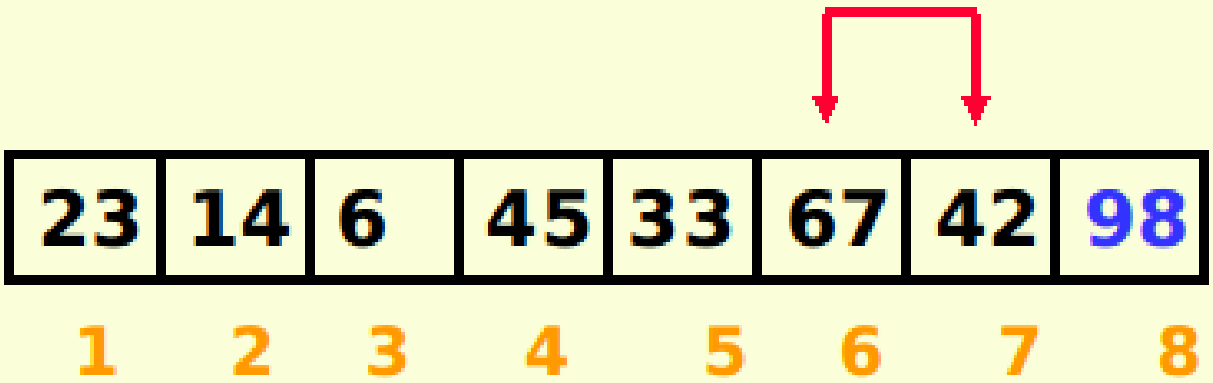
Bubble Sort Example:



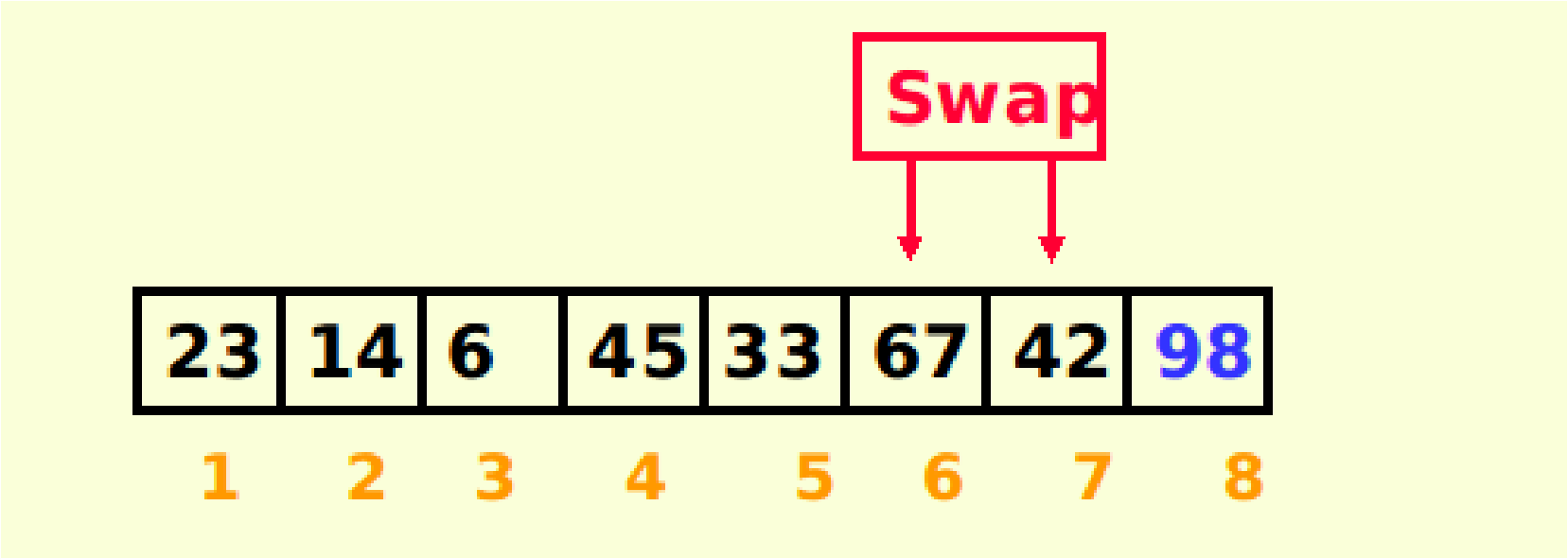
Bubble Sort Example:



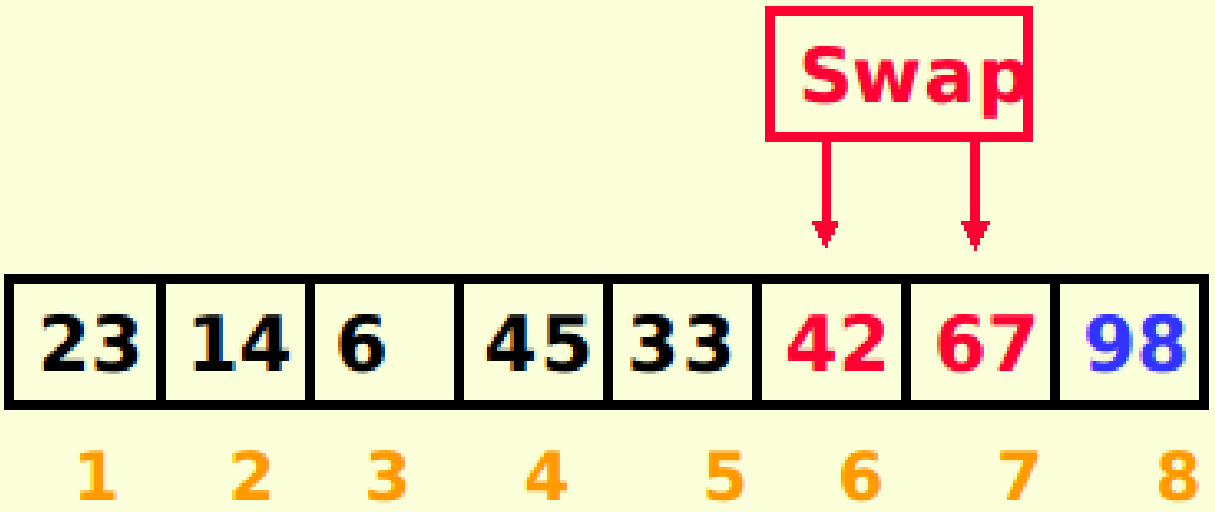
Bubble Sort Example:



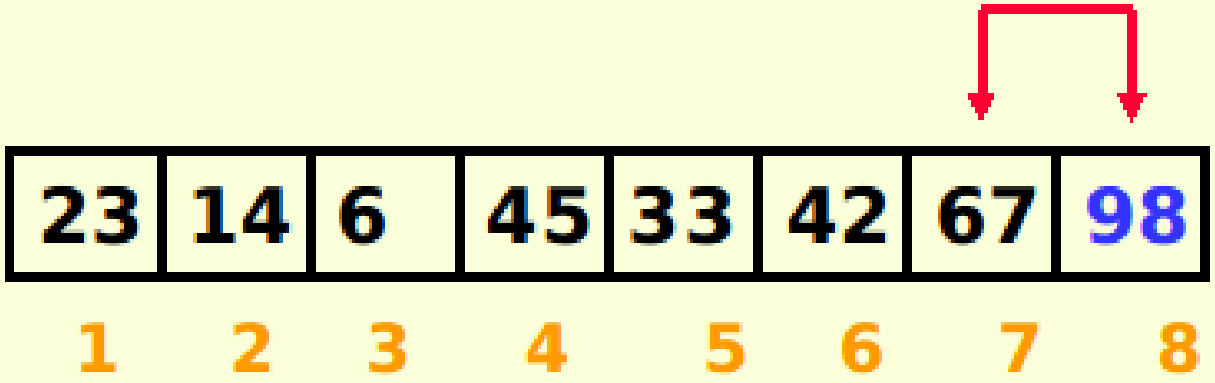
Bubble Sort Example:



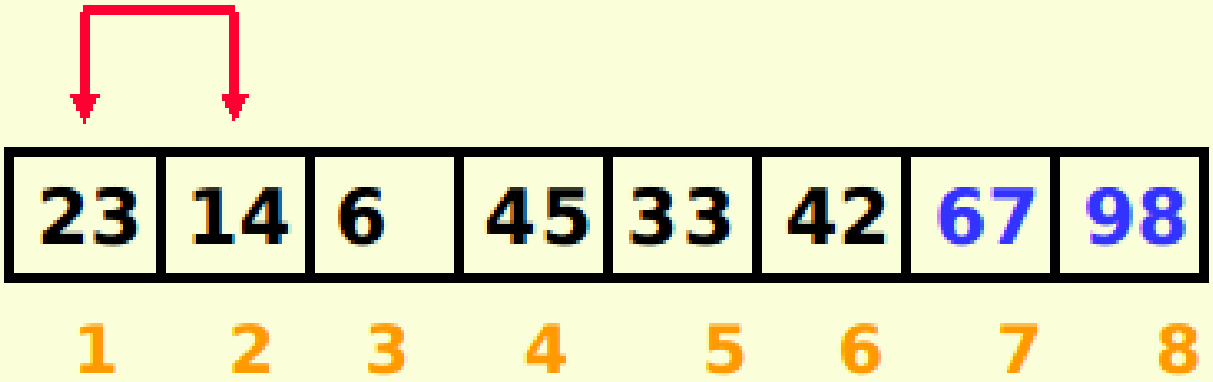
Bubble Sort Example:



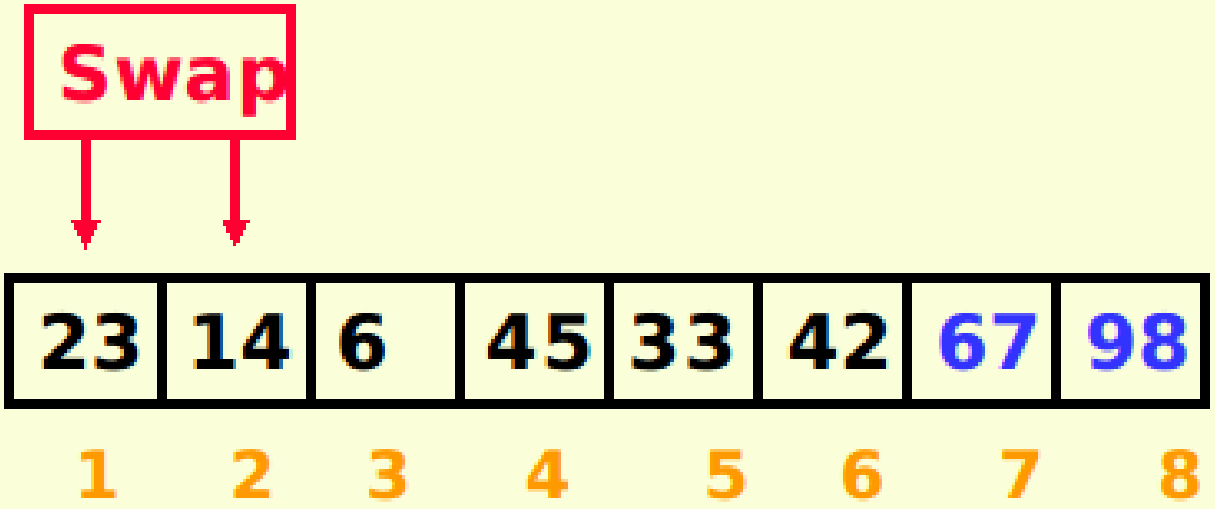
Bubble Sort Example:



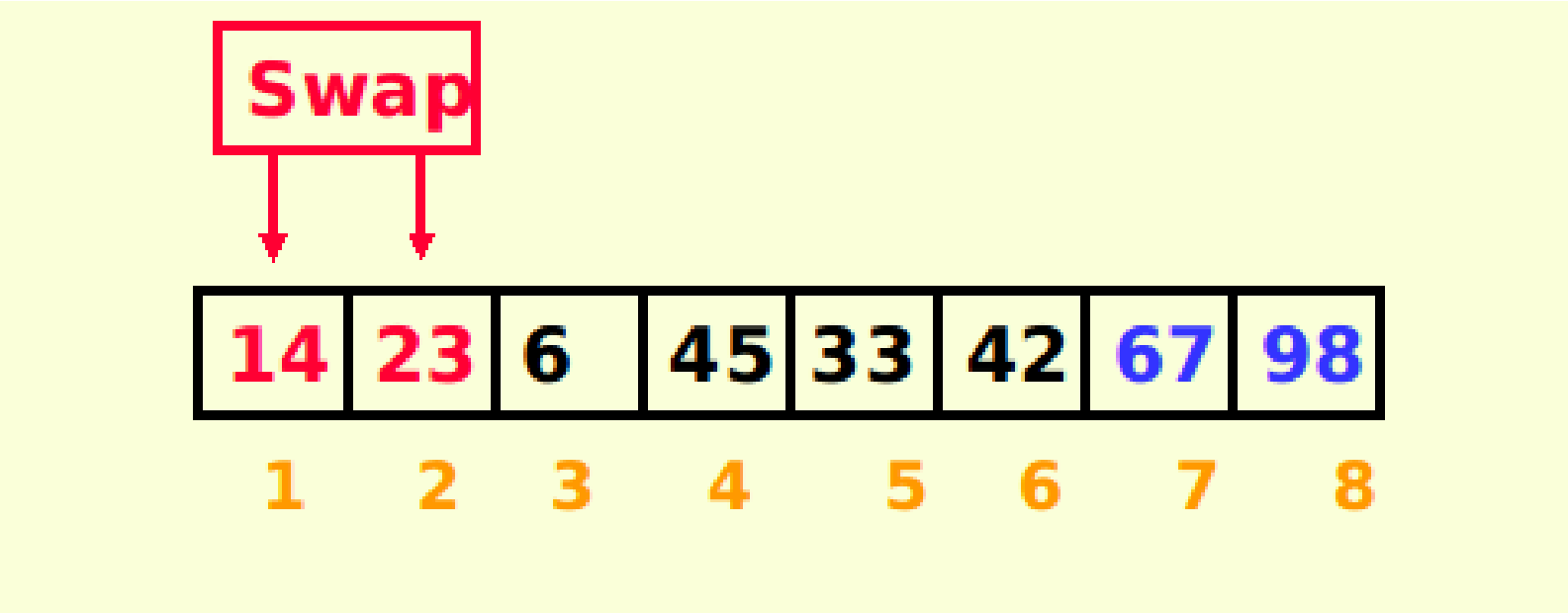
Bubble Sort Example:



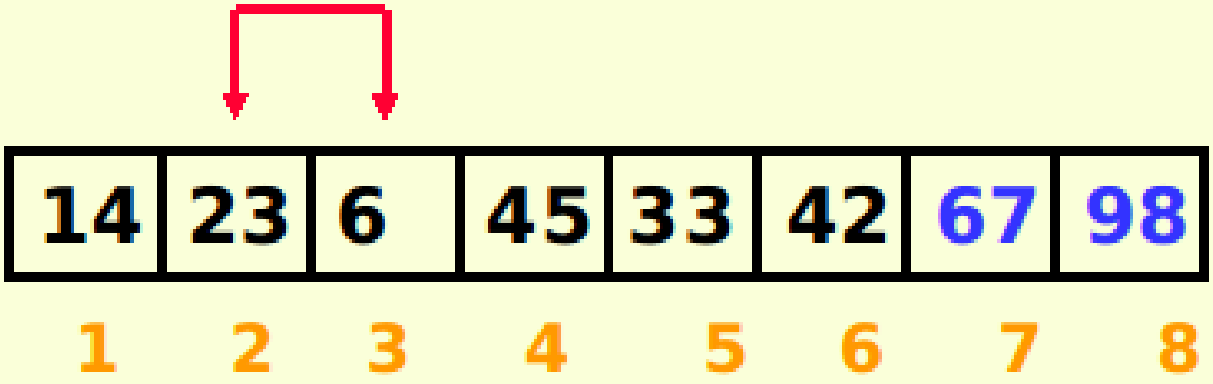
Bubble Sort Example:



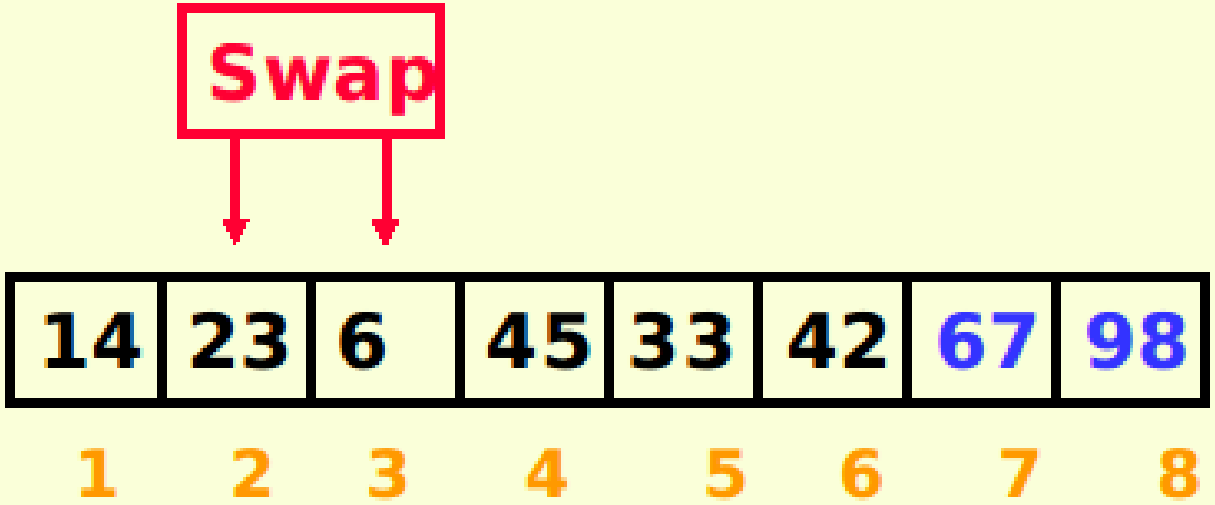
Bubble Sort Example:



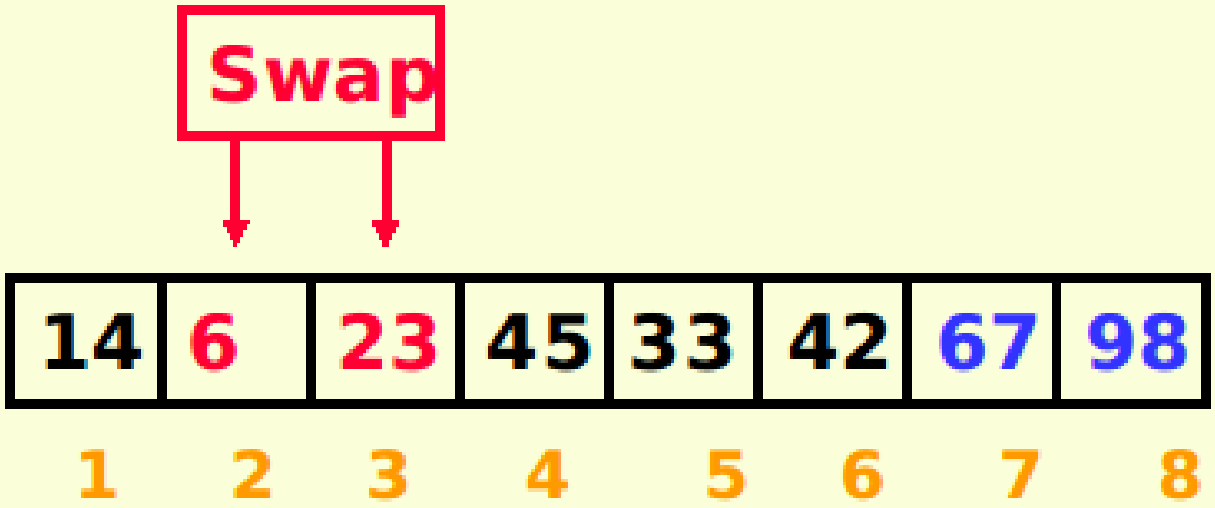
Bubble Sort Example:



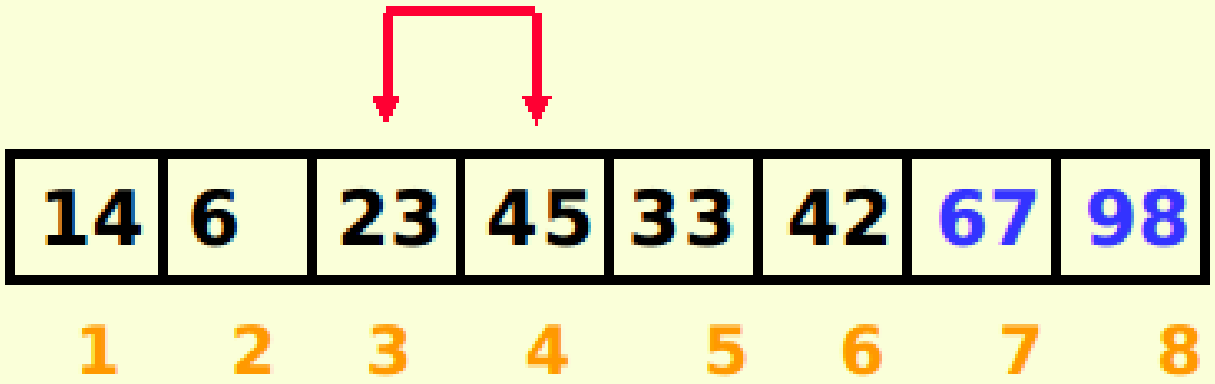
Bubble Sort Example:



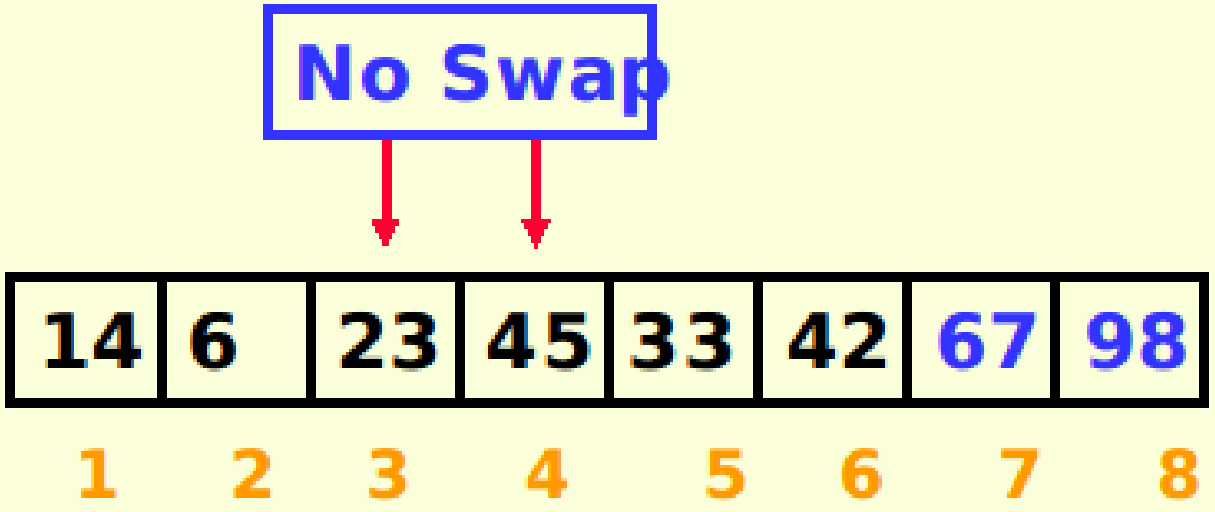
Bubble Sort Example:



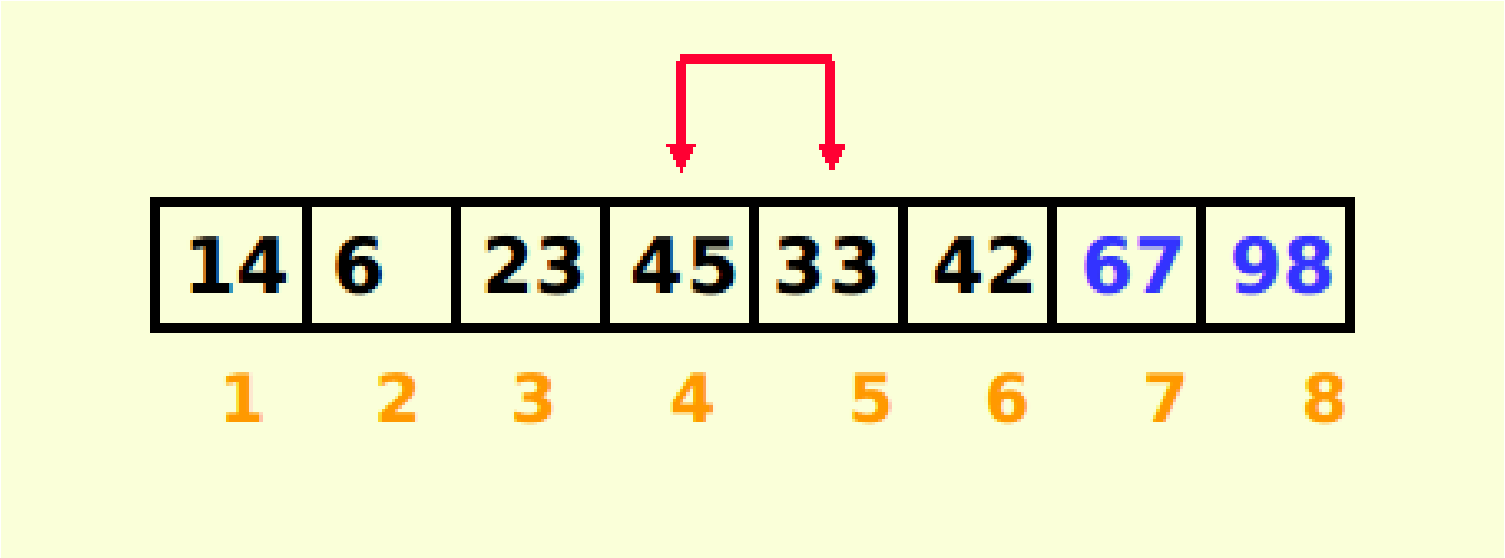
Bubble Sort Example:



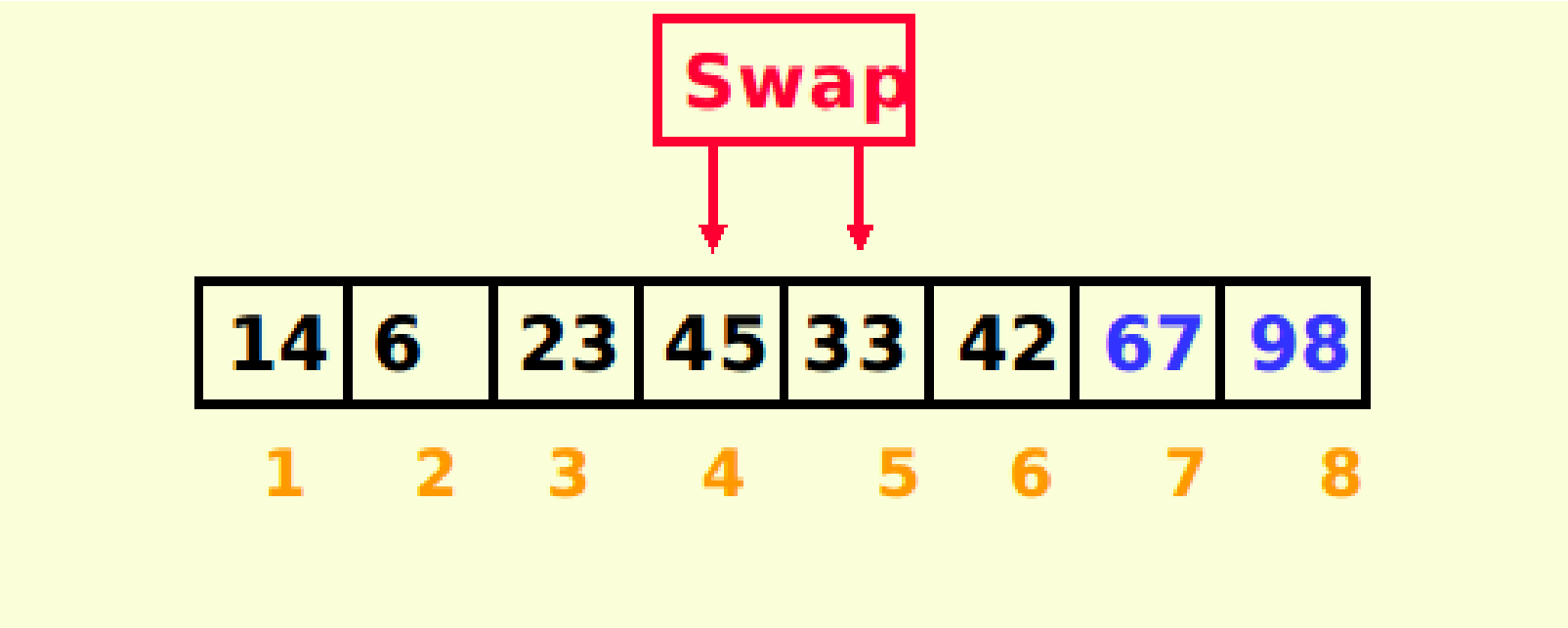
Bubble Sort Example:



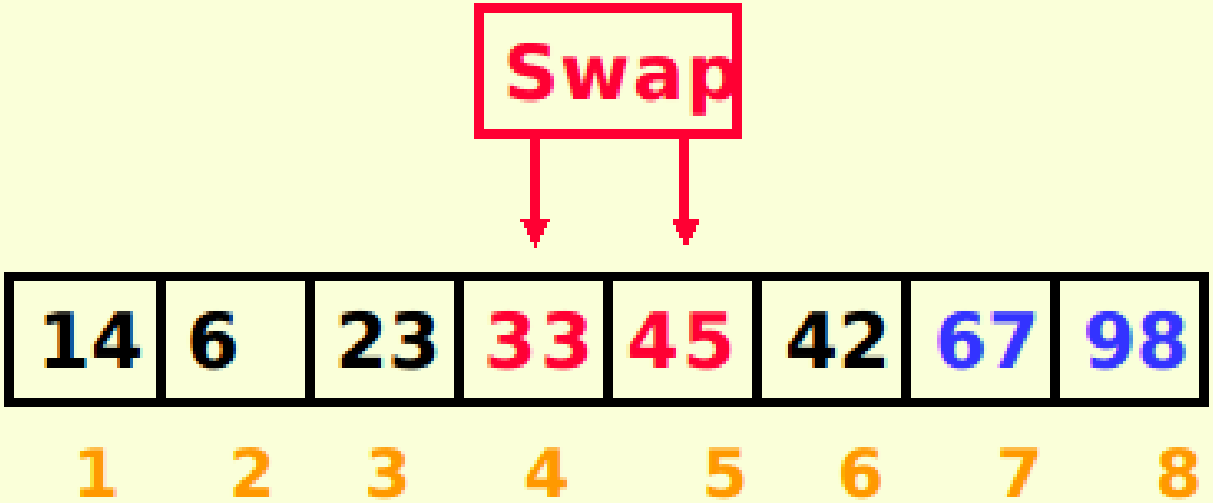
Bubble Sort Example:



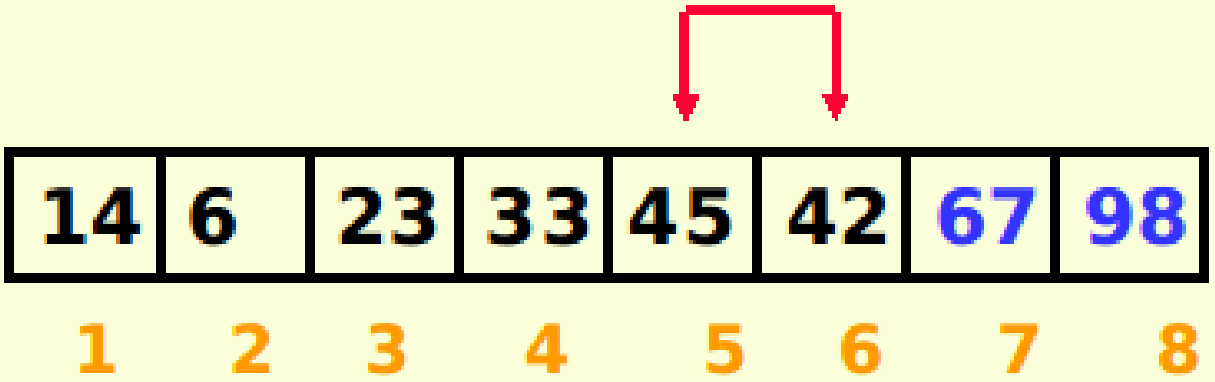
Bubble Sort Example:



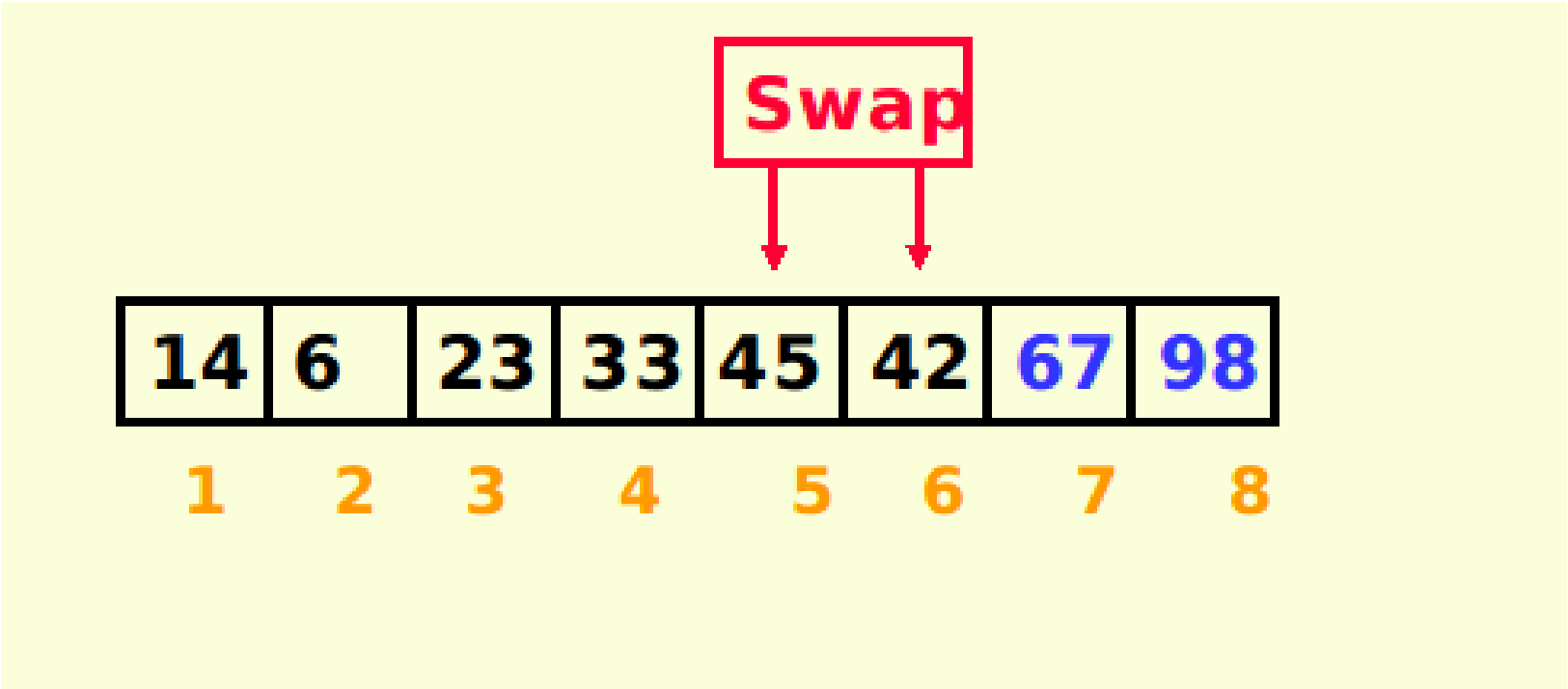
Bubble Sort Example:



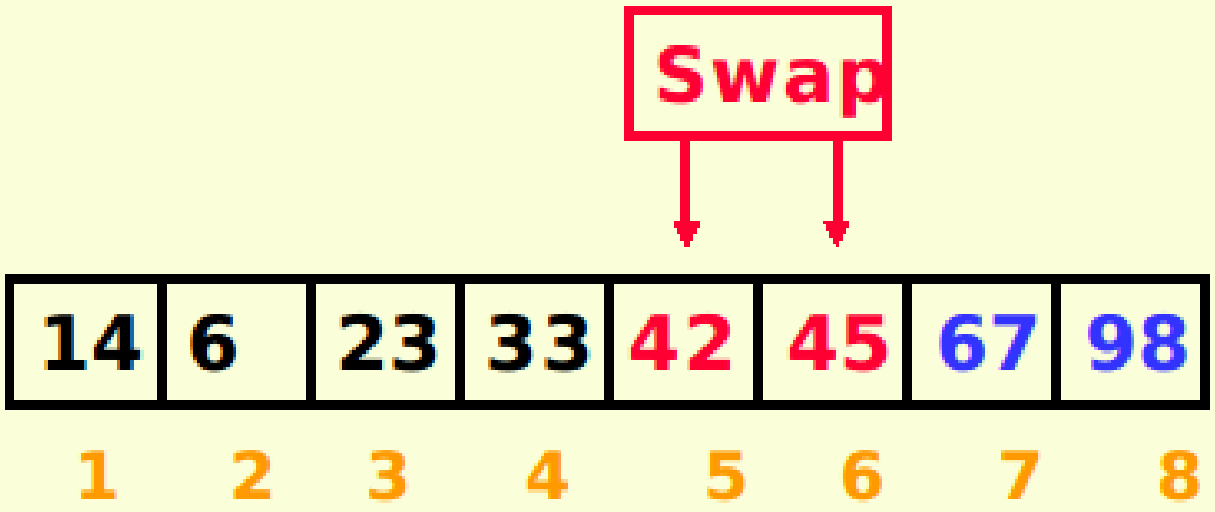
Bubble Sort Example:



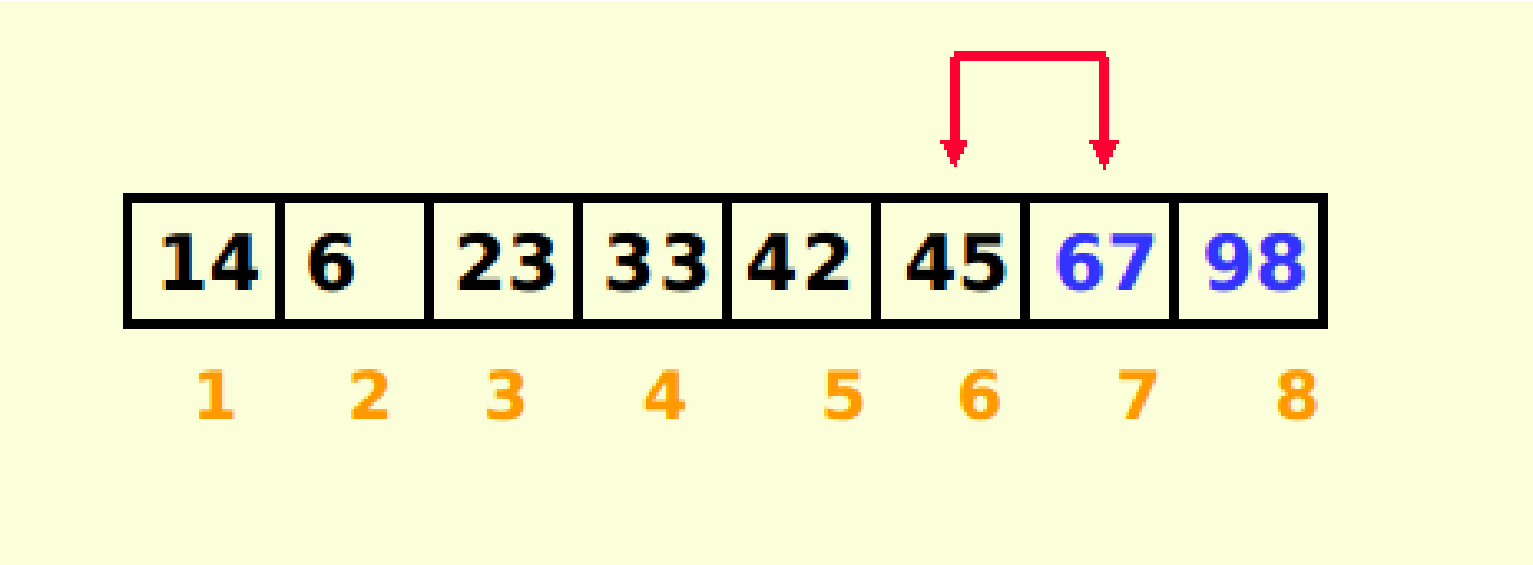
Bubble Sort Example:



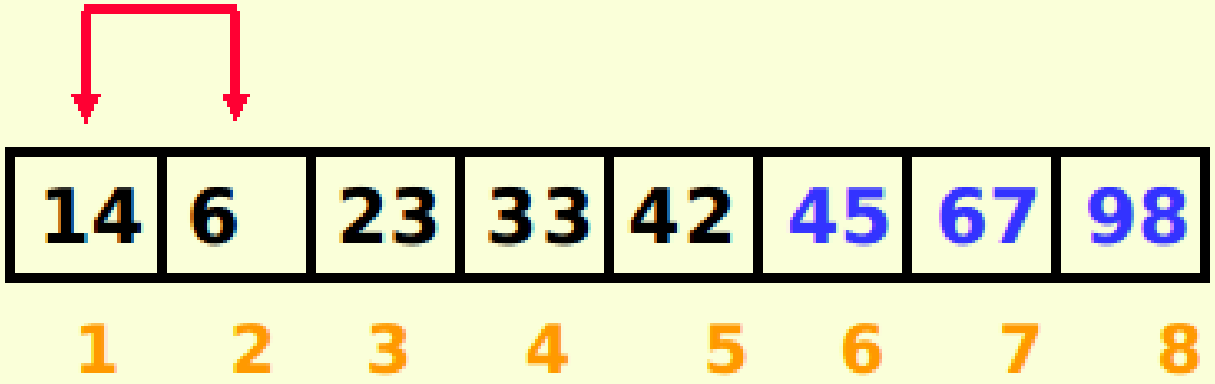
Bubble Sort Example:



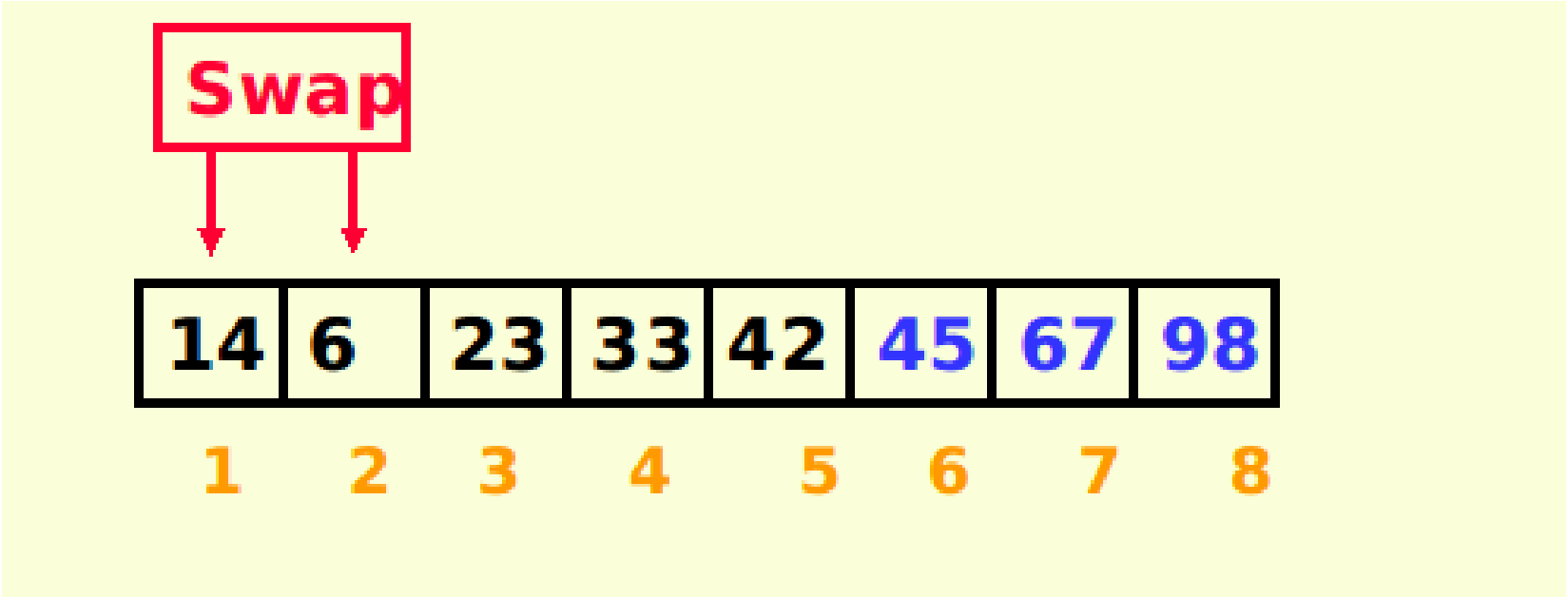
Bubble Sort Example:



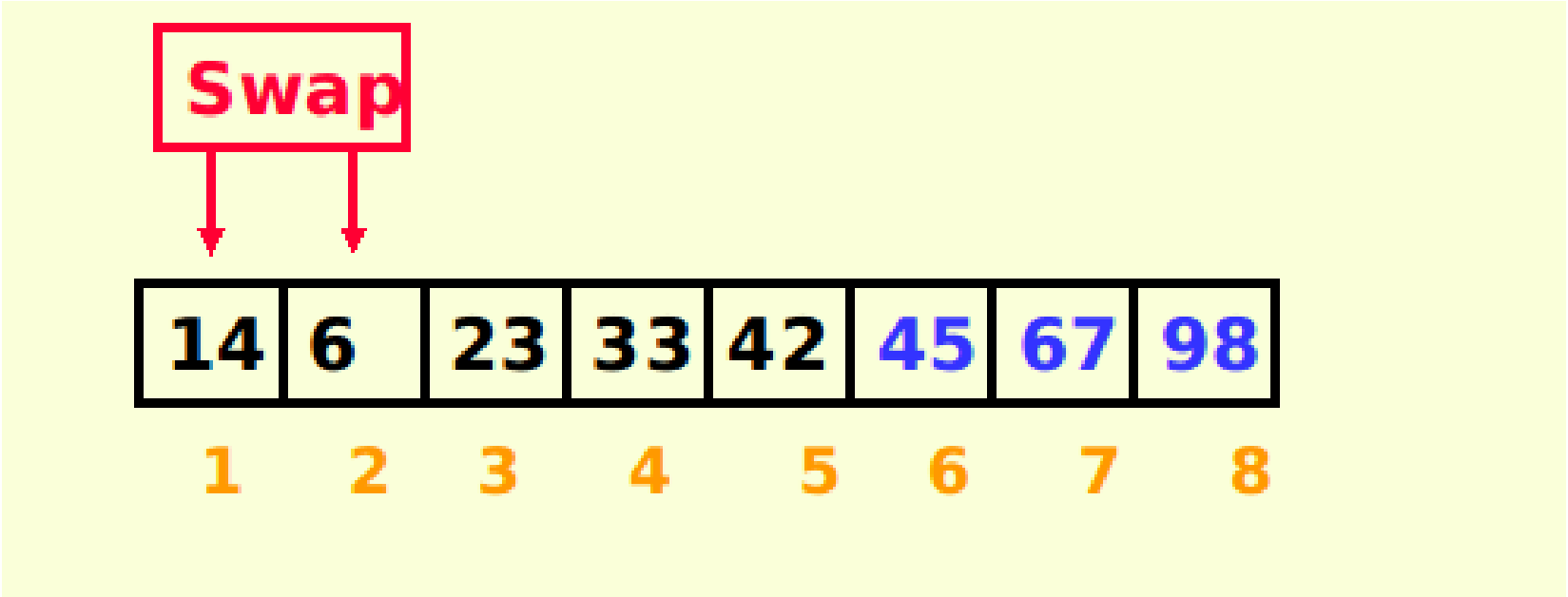
Bubble Sort Example:



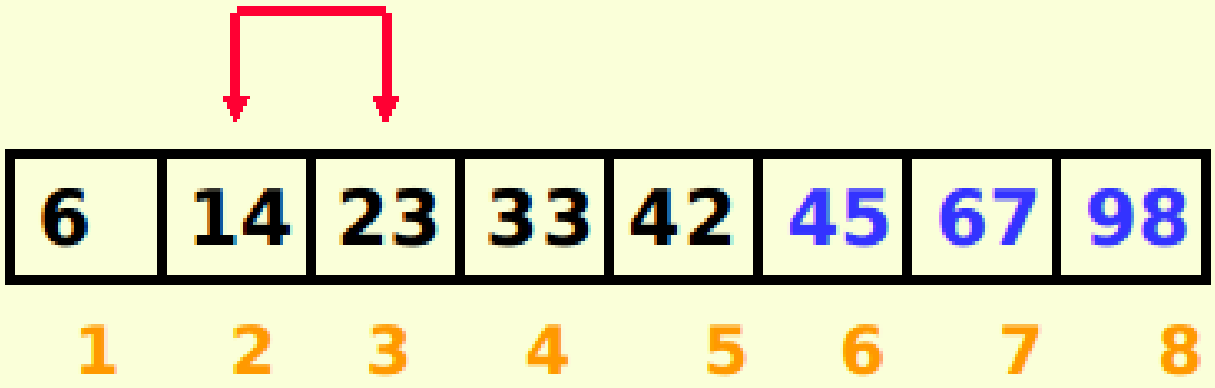
Bubble Sort Example:



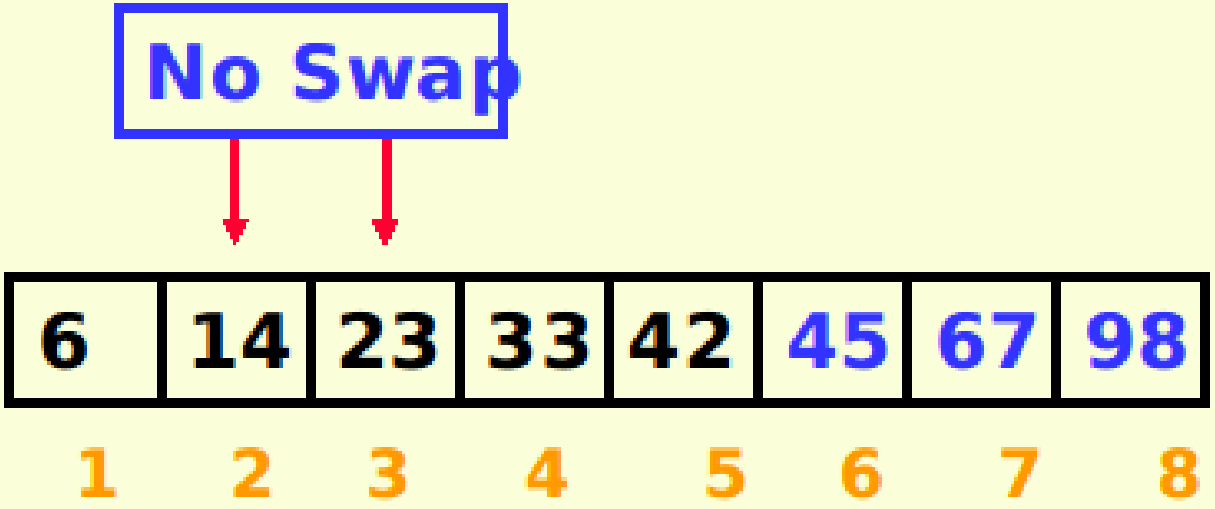
Bubble Sort Example:



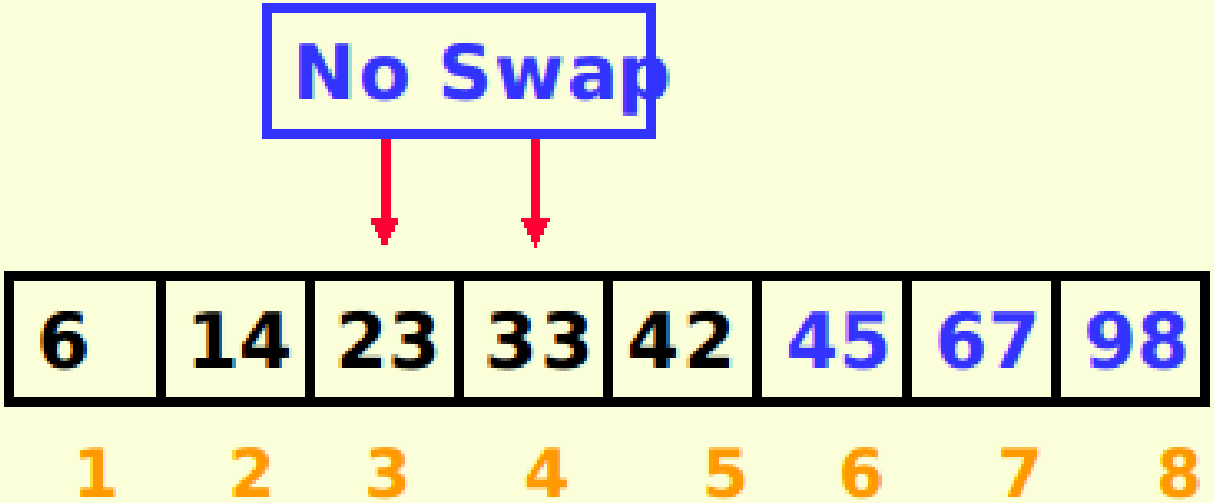
Bubble Sort Example:



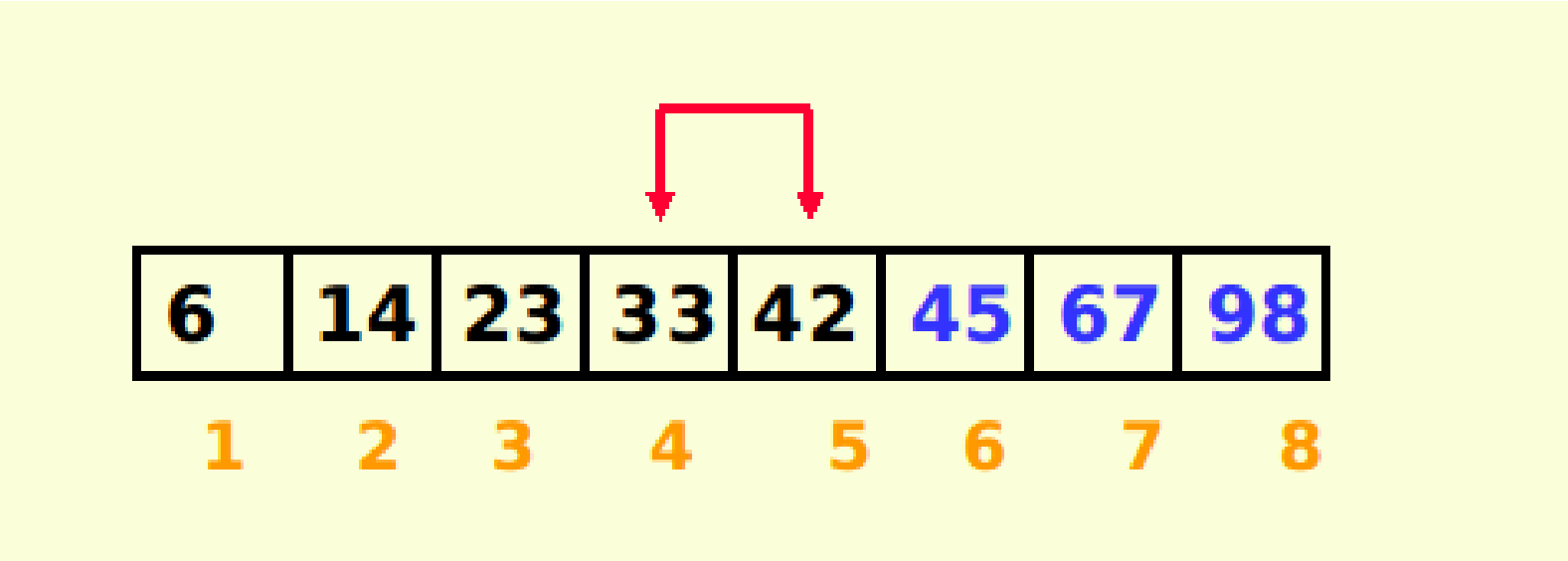
Bubble Sort Example:



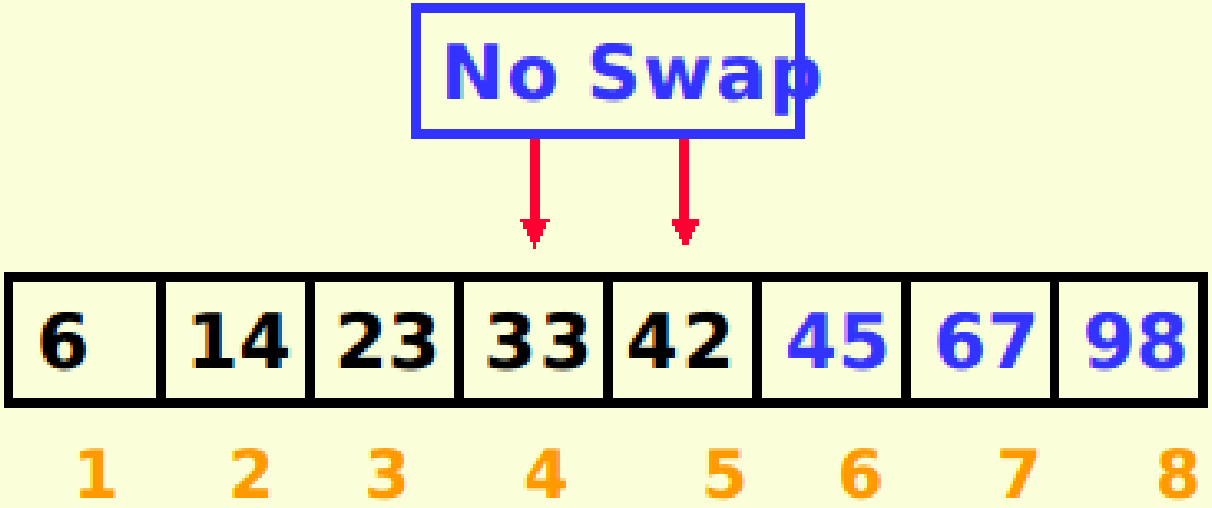
Bubble Sort Example:



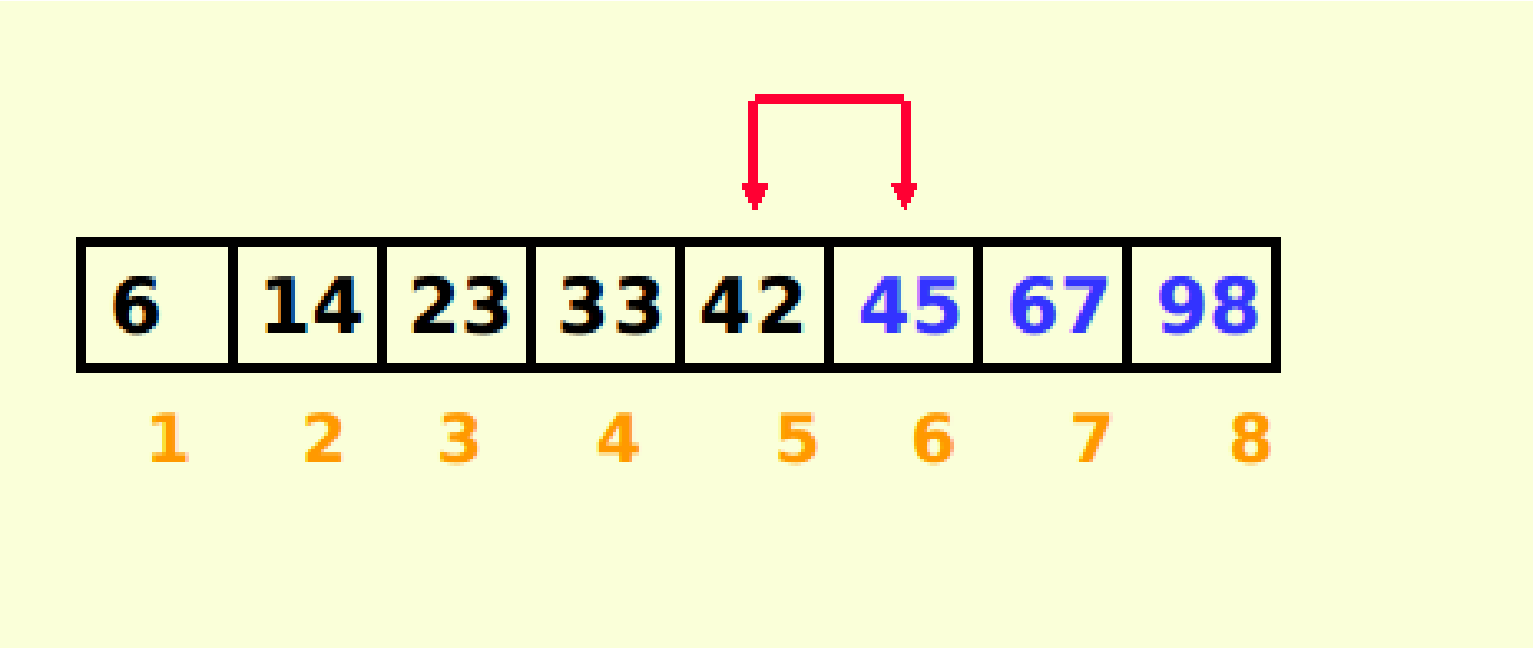
Bubble Sort Example:



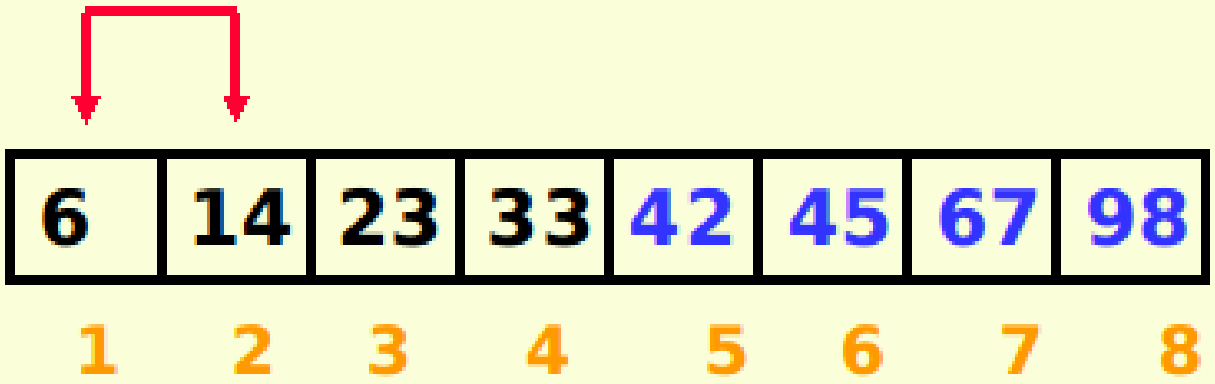
Bubble Sort Example:



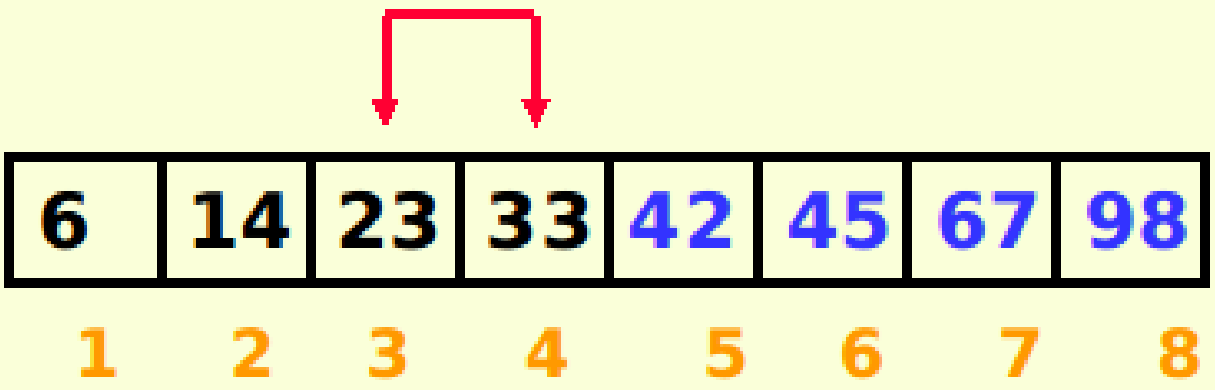
Bubble Sort Example:



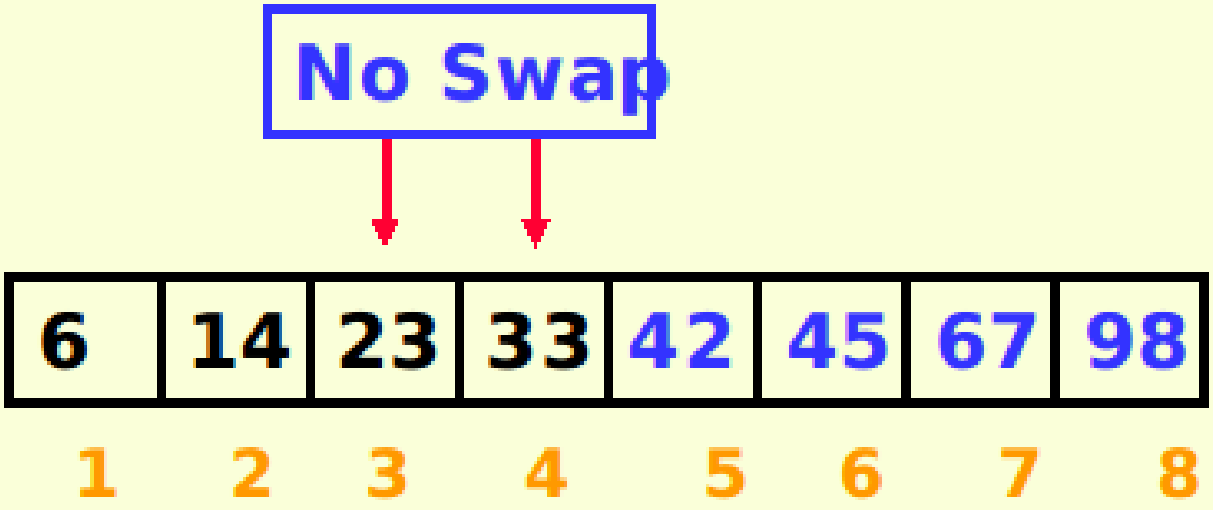
Bubble Sort Example:



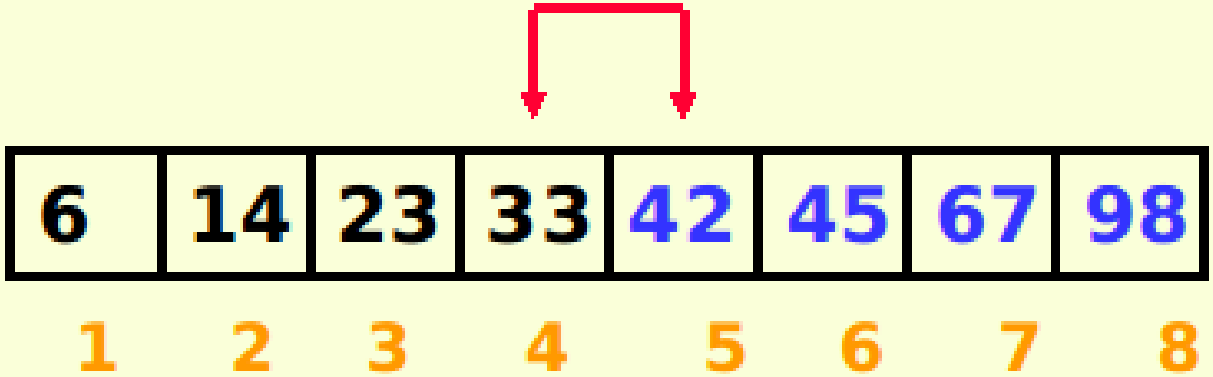
Bubble Sort Example:



Bubble Sort Example:



Bubble Sort Example:



Bubble Sort Example:

6	14	23	33	42	45	67	98
1	2	3	4	5	6	7	8

Bubble Sort Algorithm

- Algorithm for Bubble Sort:

Algorithm BubbleSort(A,n)

```
{
for i ← 0 to n − 2 do
    for j ← 0 to n − 2 − i do
        if (A[j + 1] < A[j]) then
            {
            temp ← A[j]
            A[j] ← A[j + 1]
            A[j + 1] ← temp
            }
        end if
    end for
end for
}
```

Bubble Sort Analysis

- Analysis of Bubble Sort
  - Input Size is given bye number of elements
  - Basic Operation: Key Comparisons

$$
\begin{aligned}
C(n) &= \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 \\
&= \sum_{i=0}^{n-2} [(n-2-i) - 0 + 1] \\
&= \sum_{i=0}^{n-2} (n-1-i) \\
&= \frac{(n-1)n}{2} \\
&\in \Theta(n^2)
\end{aligned}
$$

Merge Sort

- Algorithm for merge sort:

Algorithm MergeSort(low,high)

```
{
if (low < high) then
    {
    mid ← ⌊(low + high)/2⌋;
    MergeSort(low,mid);
    MergeSort(mid+1,high);
    Merge(a,low,mid,high);
    }
end if
}
```