Data Structure

http://smartlead.hallym.ac.kr

Instructor: Jin Kim
010-6267-8189(033-248-2318)
jinkim@hallym.ac.kr
Office Hours:

Non-linear Data Structure

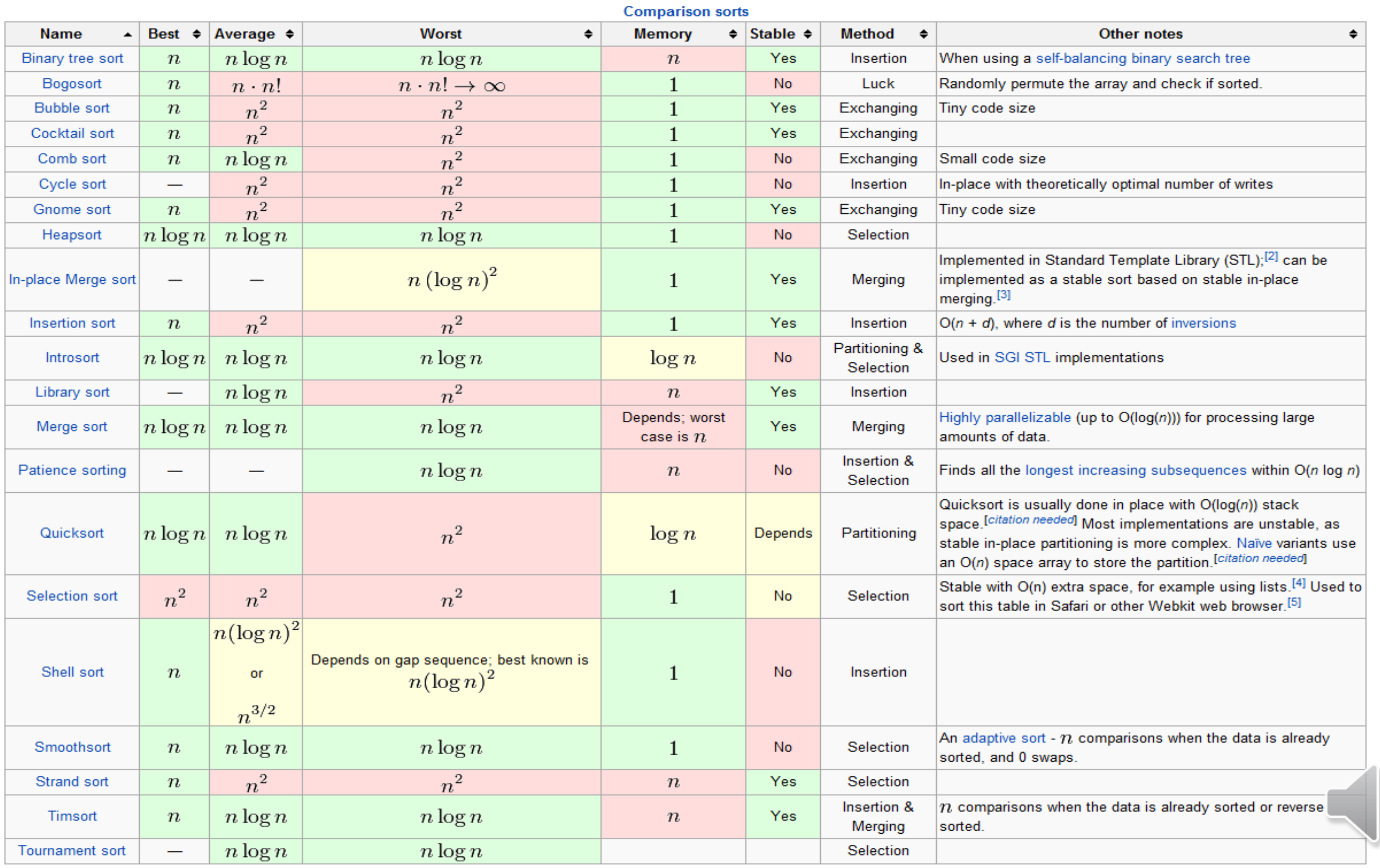
- Data structure we will consider this semister:
  - Tree
  - Binary Search Tree
  - Graph
  - Weighted Graph
  - Sorting
  - Balanced Search Tree

Sorting(정렬)

- ***Sorting*** is a process that organizes a collection of data into either ascending or descending order.
- An ***internal sort*** (내부정렬)requires that the collection of data fit entirely in the computer's main memory. (정렬하고자하는데이터를 몽땅 메인메모리에 복사하여 정렬)
- We can use an ***external sort(외부정렬)*** when the collection of data cannot fit in the computer's main memory all at once but must reside in secondary storage such as on a disk.
- We will analyze only internal sorting algorithms. (내부정렬만을 공부한다)
- An initial sort of the data can significantly enhance the performance of an algorithm.(입력데이터 최초 모양이 정렬속도에 영향을 미침)

Sorting Algorithm

- Sorting algorithms on Wikipedia
- Popular algorithms but not a full list



Comparison sorts

Name	Best	Average	Worst	Memory	Stable	Method	Other notes
Binary tree sort	n	$n \log n$	$n \log n$	n	Yes	Insertion	When using a self-balancing binary search tree
Bogosort	n	$n \cdot n!$	$n \cdot n! \to \infty$	1	No	Luck	Randomly permute the array and check if sorted.
Bubble sort	n	n^2	n^2	1	Yes	Exchanging	Tiny code size
Cocktail sort	n	n^2	n^2	1	Yes	Exchanging	
Comb sort	n	$n \log n$	n^2	1	No	Exchanging	Small code size
Cycle sort	—	n^2	n^2	1	No	Insertion	In-place with theoretically optimal number of writes
Gnome sort	n	n^2	n^2	1	Yes	Exchanging	Tiny code size
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Selection	
In-place Merge sort	—	—	$n (\log n)^2$	1	Yes	Merging	Implemented in Standard Template Library (STL);[2] can be implemented as a stable sort based on stable in-place merging.[3]
Insertion sort	n	n^2	n^2	1	Yes	Insertion	O(n + d), where d is the number of inversions
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No	Partitioning & Selection	Used in SGI STL implementations
Library sort	—	$n \log n$	n^2	n	Yes	Insertion	
Merge sort	$n \log n$	$n \log n$	$n \log n$	Depends; worst case is n	Yes	Merging	Highly parallelizable (up to O(log(n))) for processing large amounts of data.
Patience sorting	—	—	$n \log n$	n	No	Insertion & Selection	Finds all the longest increasing subsequences within O(n log n)
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	Depends	Partitioning	Quicksort is usually done in place with O(log(n)) stack space.[citation needed] Most implementations are unstable, as stable in-place partitioning is more complex. Naïve variants use an O(n) space array to store the partition.[citation needed]
Selection sort	n^2	n^2	n^2	1	No	Selection	Stable with O(n) extra space, for example using lists.[4] Used to sort this table in Safari or other Webkit web browser.[5]
Shell sort	n	$n(\log n)^2$ or $n^{3/2}$	Depends on gap sequence; best known is $n(\log n)^2$	1	No	Insertion	
Smoothsort	n	$n \log n$	$n \log n$	1	No	Selection	An adaptive sort - n comparisons when the data is already sorted, and 0 swaps.
Strand sort	n	n^2	n^2	n	Yes	Selection	
Timsort	n	$n \log n$	$n \log n$	n	Yes	Insertion & Merging	n comparisons when the data is already sorted or reverse sorted.
Tournament sort	—	$n \log n$	$n \log n$			Selection	

Sorting

- *Sorting* is the process of arranging a list of items in a particular order
- The sorting process is based on specific value(s)
  - Sorting a list of test scores in ascending numeric order
  - Sorting a list of people alphabetically by last name
- Sequential sorts O(n^2): Selection, Insertion, Bubble
- Logarithmic sorts O(nlogn): Quick, Merge

Definitions

- Model: In-place sort of an array(제자리정렬)
  - *An in-place algorithm is an algorithm that does not need an extra space and produces an output in the same memory that contains the data by transforming the input 'in-place'. However, a small constant extra space used for variables is allowed.*
- Stable vs. unstable algorithms(안정알고리즘, 불안정알고리즘)
- Time measures:
  - Number of comparisons
  - Number of swaps
- Three "classical" sorting algorithms
  - Insertion sort
  - Bubble sort
  - Selection sort

Inplace sort(제자리정렬)

- Inplace 알고리즘이란 추가적인 메모리 공간을 많이 필요로 하지 않는 혹은 전혀 필요하지 않는 알고리즘을 의미한다.
- 즉, n 길이의 리스트가 있고, 이 리스트를 정렬할 때 추가적으로 메모리 공간을 할당하지 않아도 정렬이 이뤄진다면 in-place 알고리즘이라고 불릴 수 있는 것이다.

Which of the following are inplace sorting algorithms ?

– Bubble sort	yes
– Insertion sort	yes
– Selection sort	yes
– Heap sort	yes
– Merge sort	no
– Quick sort	yes

Stable Sorting Algorithms

안정정렬: 동일한 키가 여러 개 있을때, 원래의 키순으로 정렬하는 알고리즘

예) 3_1,6… 3_2,…5, 3_3 $\longrightarrow$ 3_1, 3_2, 3_3 , 4, 6, …

Which of the following are stable sorting algorithms ?

• Bubble sort	yes	
• Insertion sort	yes	
• Selection sort	yes	
• Heap sort	no	// unstable, 불안정
• Merge sort	no	// unstable, 불안정
• Quick sort	no	// unstable, 불안정

Summary of Sorting Algorithms

Algorithm	Time	Notes
selection-sort	$O(n^2)$	■ in-place(제자리) ■ slow (good for small inputs)
insertion-sort	$O(n^2)$	■ in-place(제자리) ■ slow (good for small inputs)
bubble-sort	$O(n^2)$	■in-place(제자리) ■ slow (good for small inputs)
quick-sort	$O(n \log n)$ expected	■ in-place(제자리), randomized ■ fastest (good for large inputs)
heap-sort	$O(n \log n)$	■ in-place(제자리) ■ fast (good for large inputs)
merge-sort	$O(n \log n)$	■ sequential data access ■ fast (good for huge inputs) ■ needs n additional space

Sorting in linear time(선형시간정렬)

- Comparison sort(compare keys):
  - Lower bound: $\Omega(n\lg n)$.
  - 키를 비교하는 정렬은 $\Theta(n\lg n)$보다 빠를 수 없다.
- Non comparison sort(not compare keys):
  - Bucket sort, counting sort, radix sort
  - They are possible in linear time (under certain assumption).
  - 키를 비교하지 않는 정렬은 선형시간 즉 $\Theta(n)$ 시간에 정렬가능하다.

Some Remarks

Insertion-sort is a good choice for small input size (say, less than 50) and for sequences that are already "almost" sorted.

Merge-sort is difficult to run in-place, is an excellent algorithm for situations where the input can not fit into main memory, but must be stored in blocks on an external memory device, e.g., disks. (외부정렬에 주로사용)

Quick sort is an excellent choice for general-purpose, in-memory sorting. In spite of its slow worst-case running time. The constant factors hidden in O(nlgn) for average case are quite small.

원소가 하나 있다면 정렬된 것인가? yes

$\boxed{5}$

정렬하는데 걸리는 시간?
없음(상수시간)

$O(n^2)$ 정렬 알고리즘

Selection sort(선택정렬)

Bubble sort(버블정렬)

Insertion sort(삽입정렬)

$O(n \log n)$정렬 알고리즘

Quick sort(퀵정렬)

Merge sort(합병정렬)

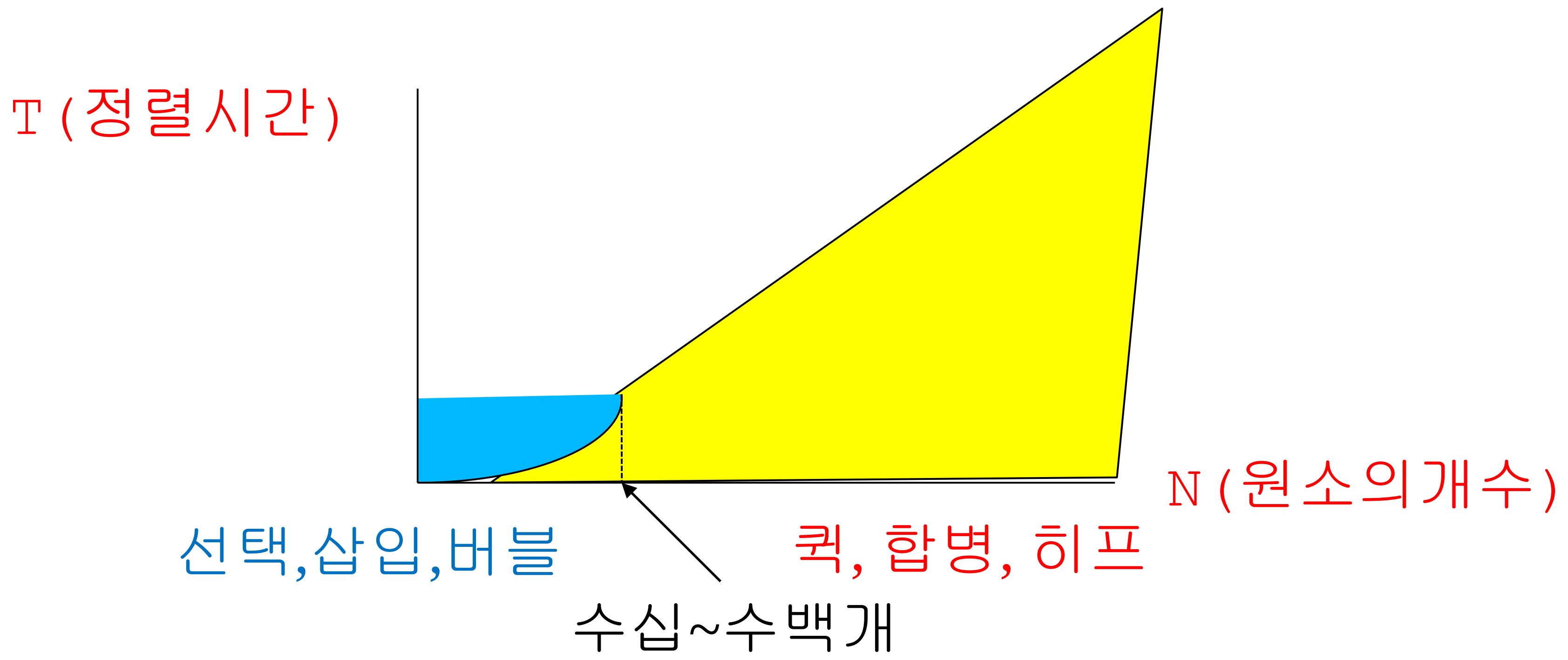
Heap sort(히프정렬)

$O(n)$정렬 알고리즘

Counting sort(계수정렬)

Radix sort(기수정렬)

Bucket sort(버킷정렬)


T (정렬시간)
N (원소의개수)
선택,삽입,버블
퀵, 합병, 히프
수십~수백개

$\boldsymbol{O}(\boldsymbol{n}^2)$ 정렬 알고리즘

Selection sort(선택정렬)

Bubble sort(버블정렬)

Insertion sort(삽입정렬)

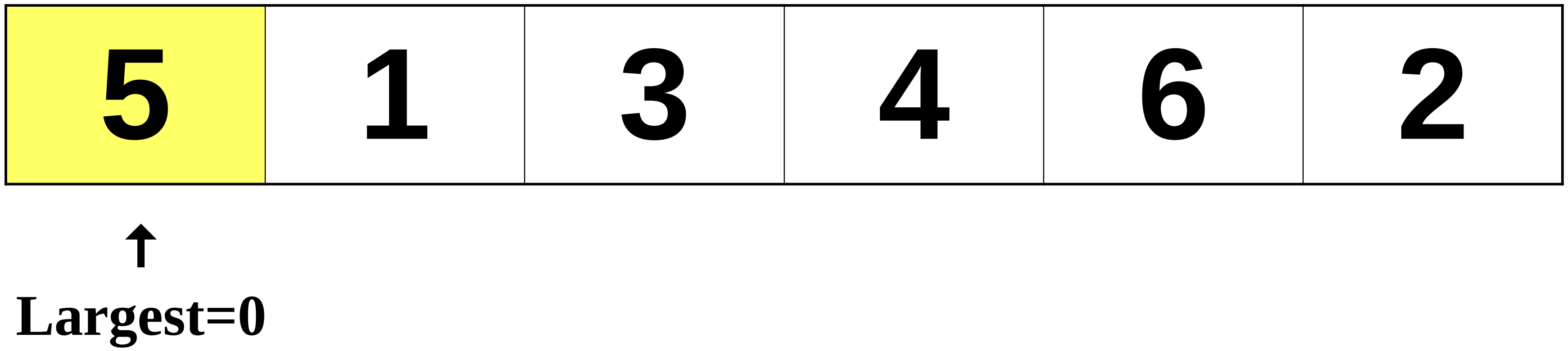
Selection Sort(선택정렬)

Selection Sort(선택정렬)

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

🟨	Comparison
🟩	Data Movement
🟦	Sorted

Selection Sort



🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

5	1	3	4	6	2

↑

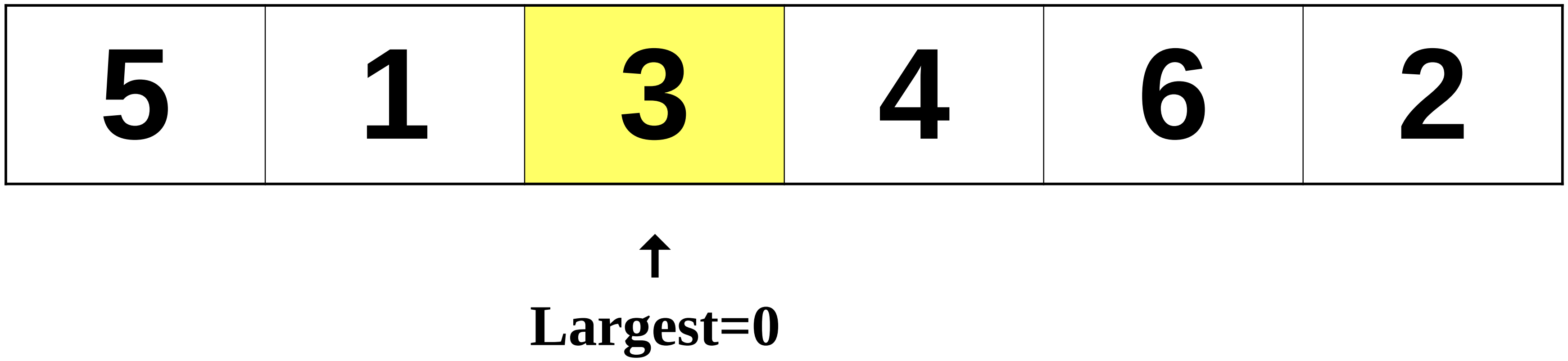
Largest=0

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

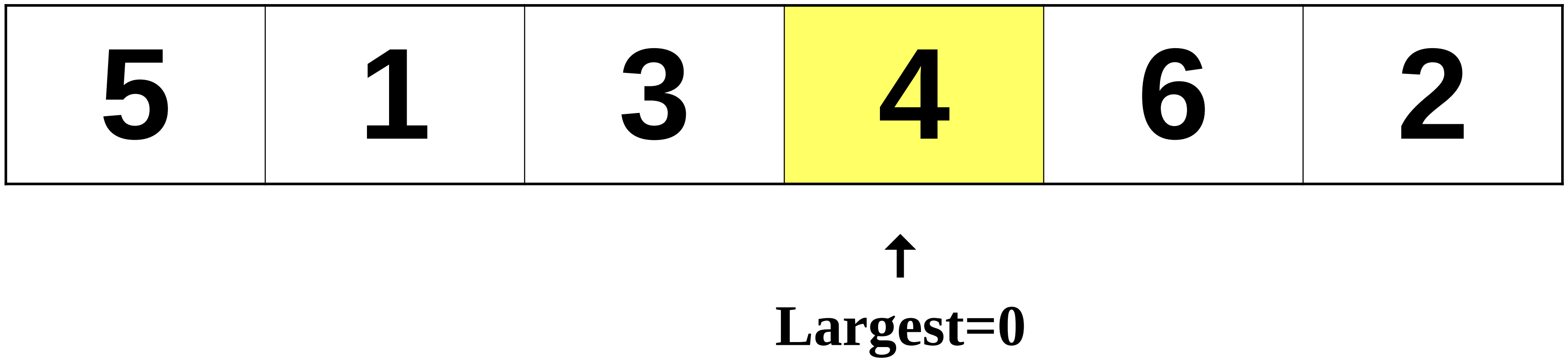


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

5	1	3	4	🟨 6	2

↑

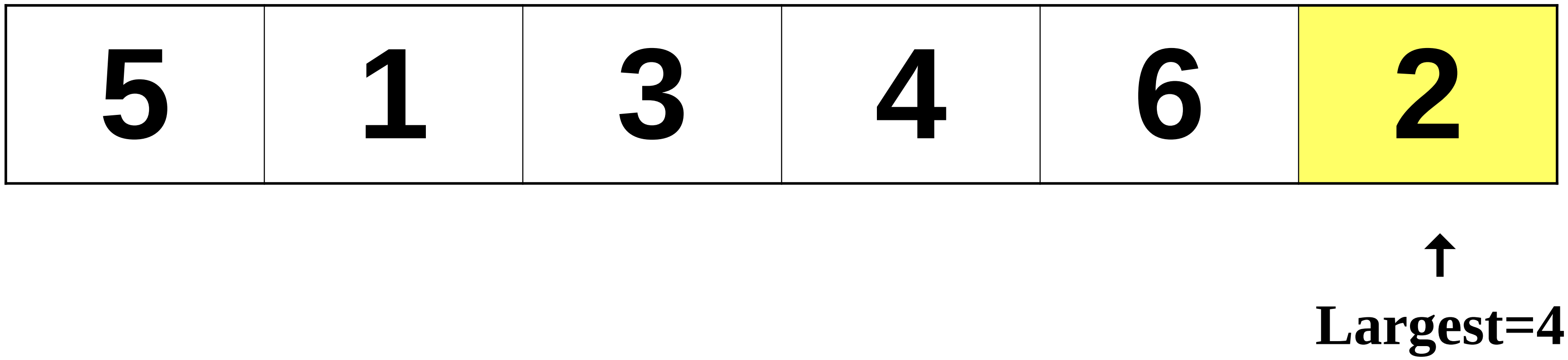
Largest=4

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

5	1	3	4	6	2

↑

Largest=4

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

5	1	3	4	🟩 2	🟩 6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

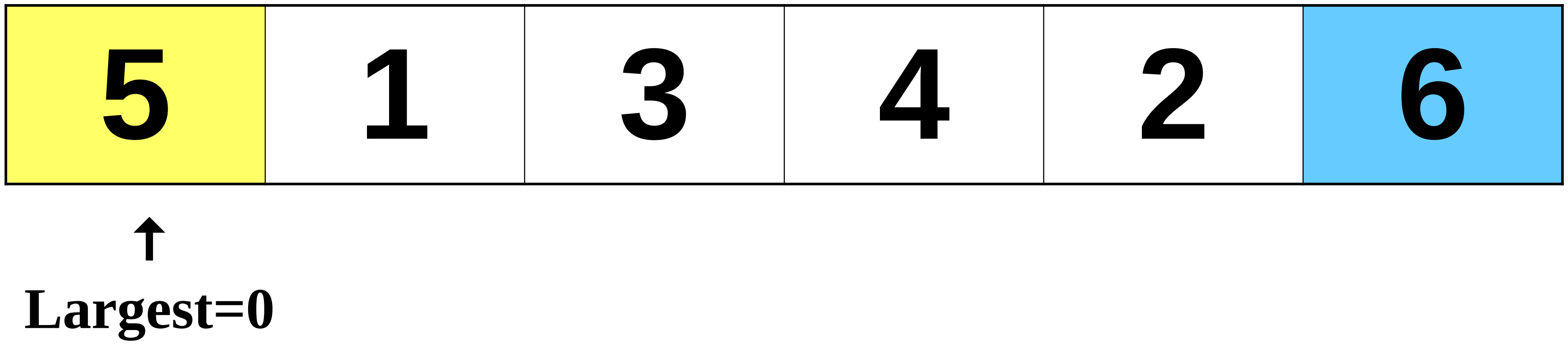
5	1	3	4	2	🟦 6

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

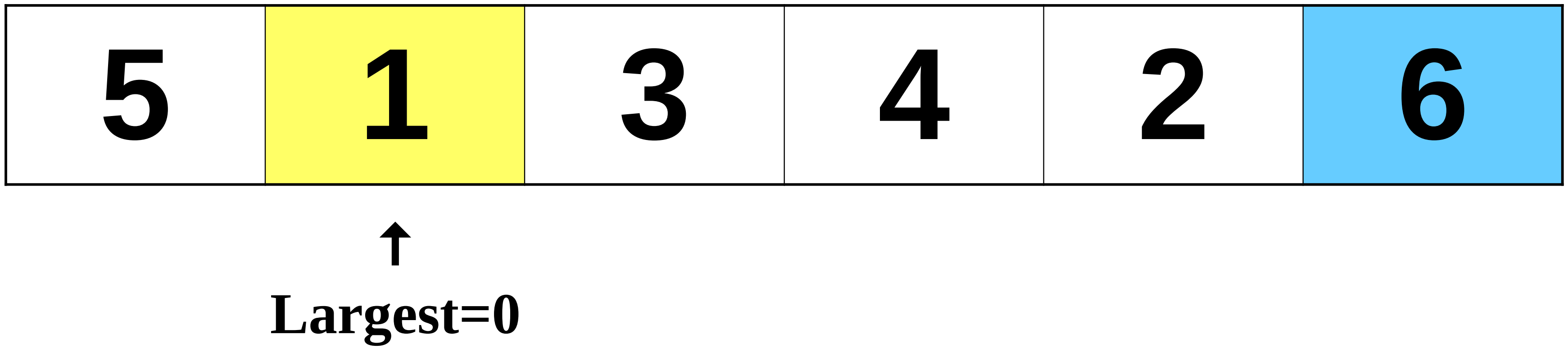


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

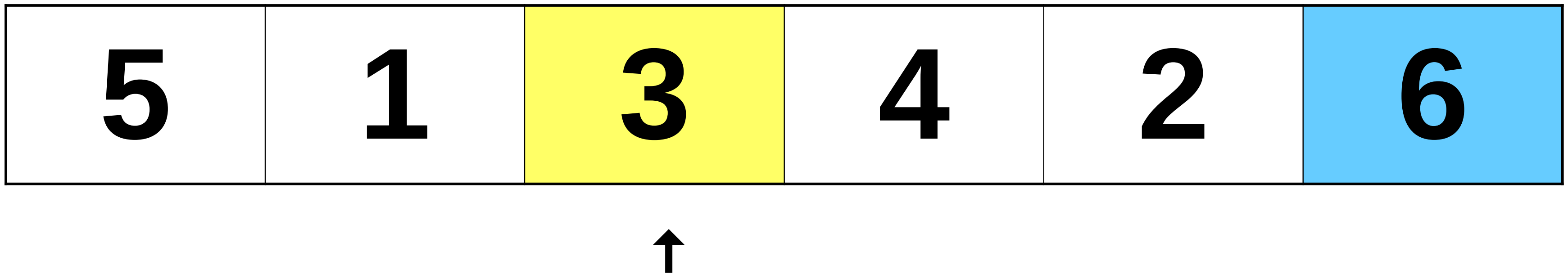


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



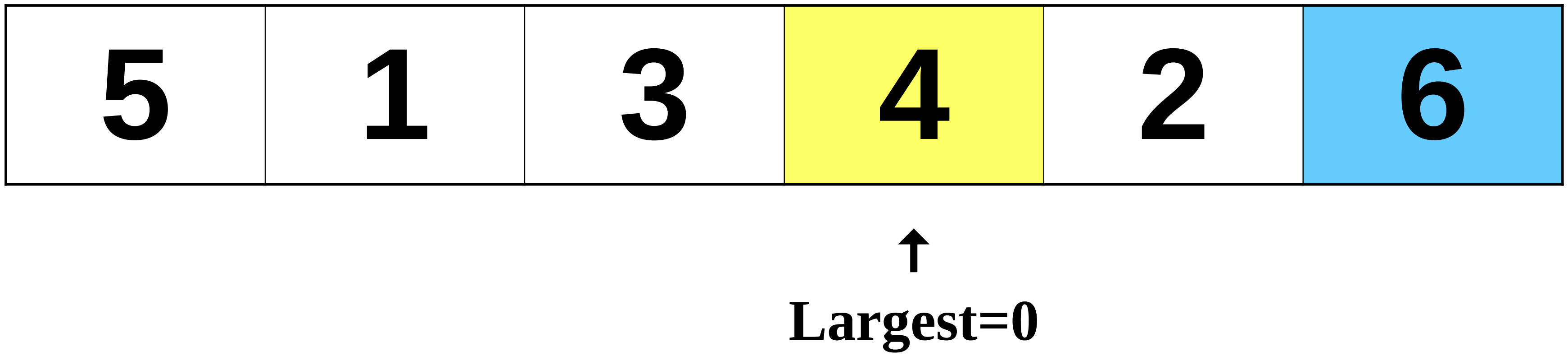
Largest=0

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

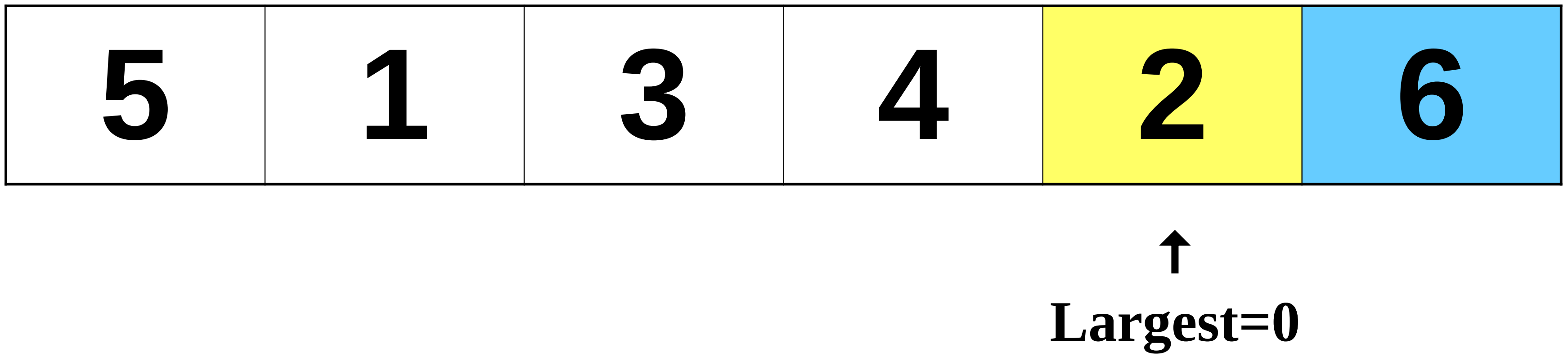


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

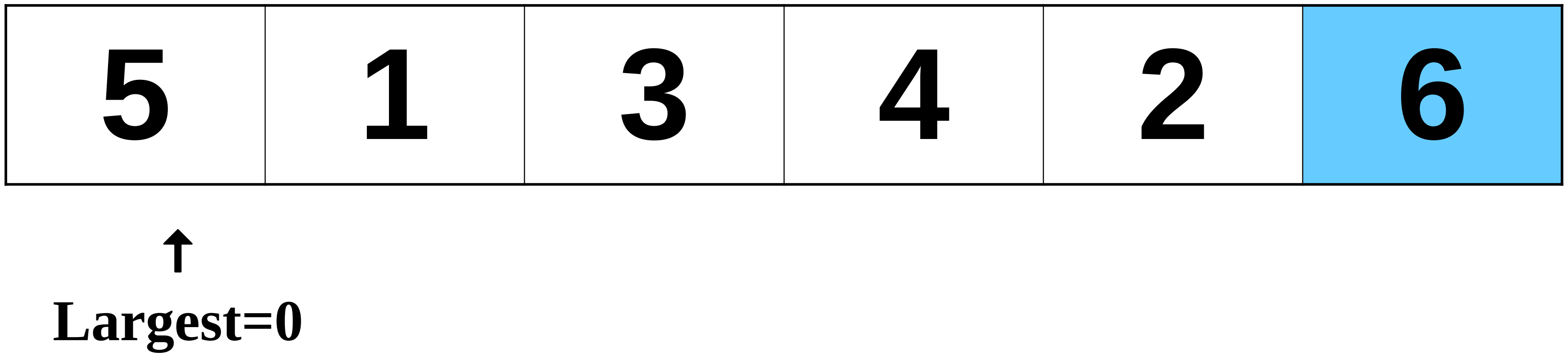


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

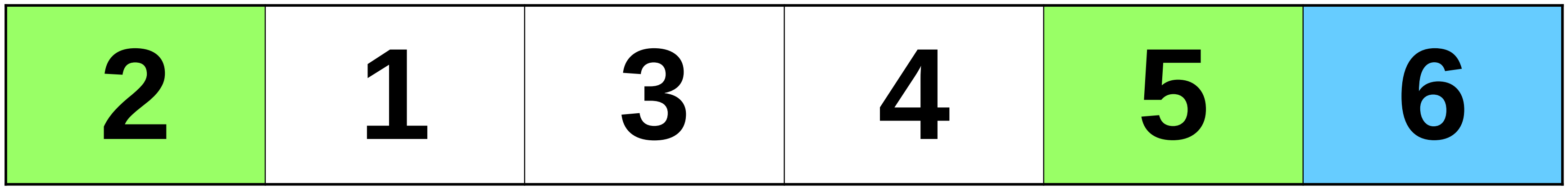


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

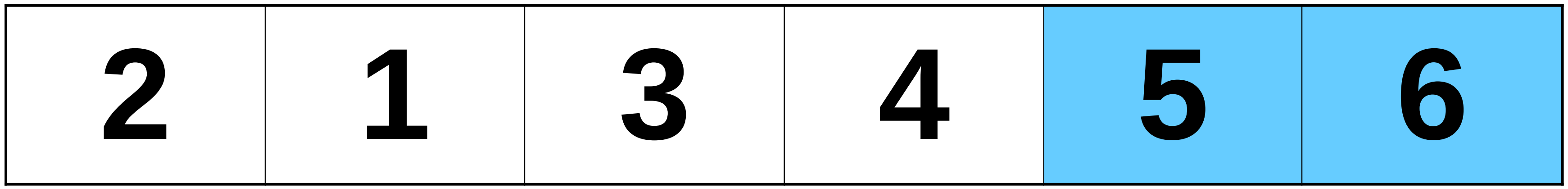


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

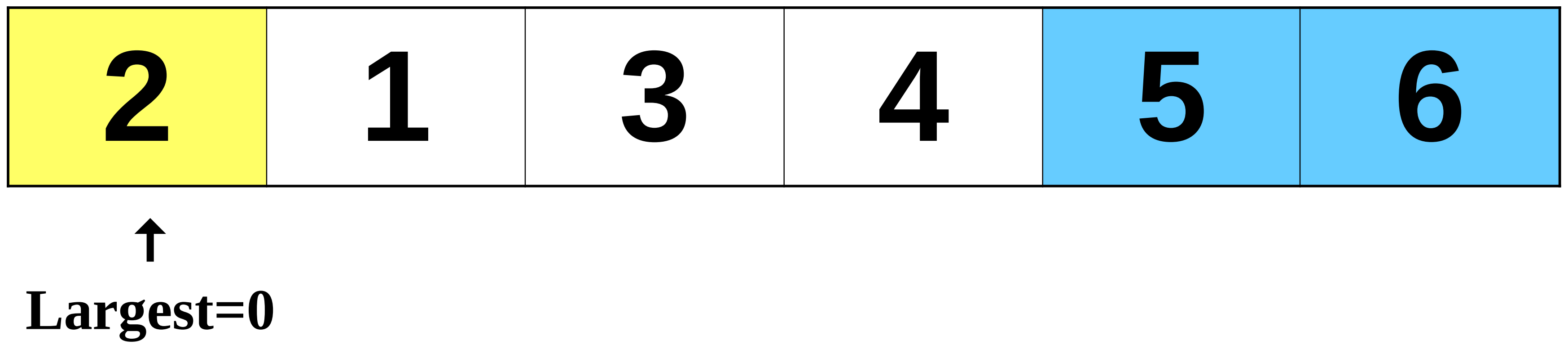


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

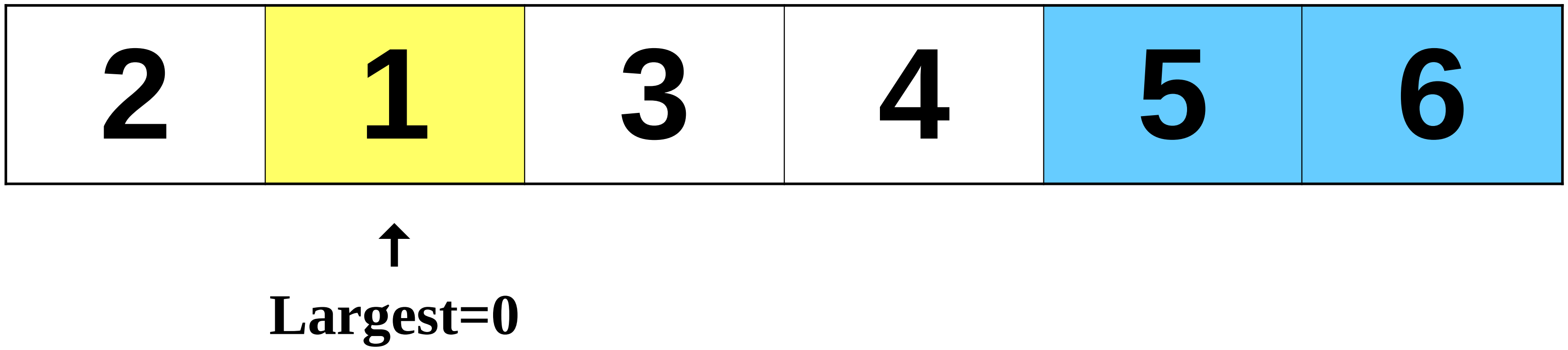


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

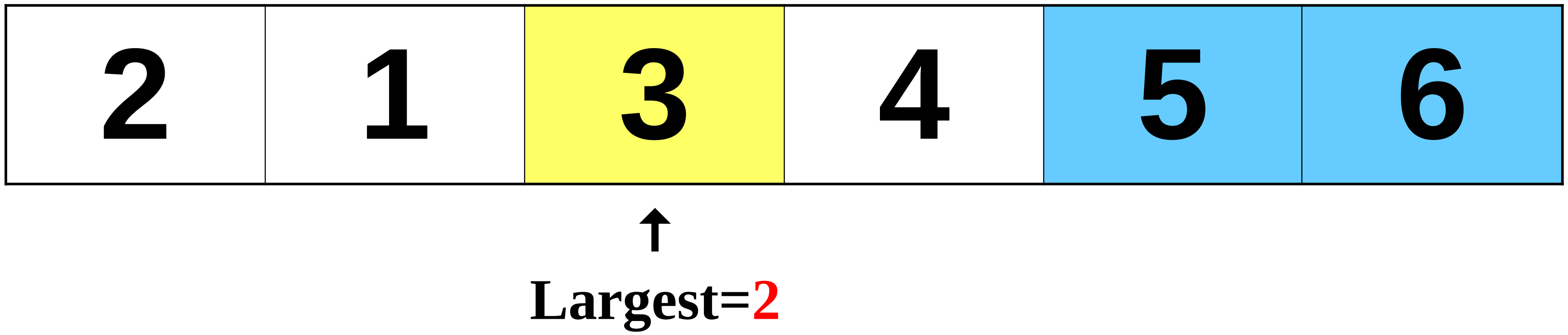


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

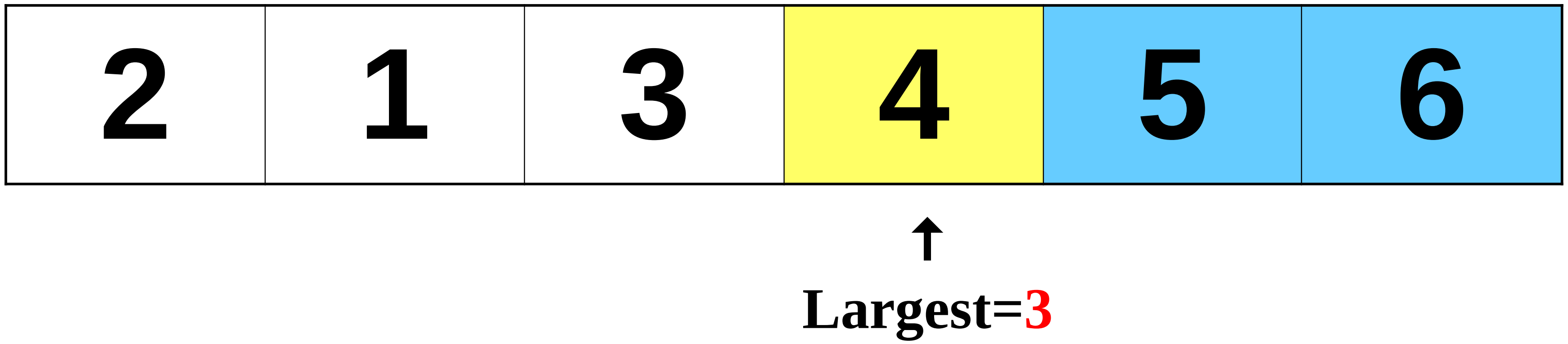


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

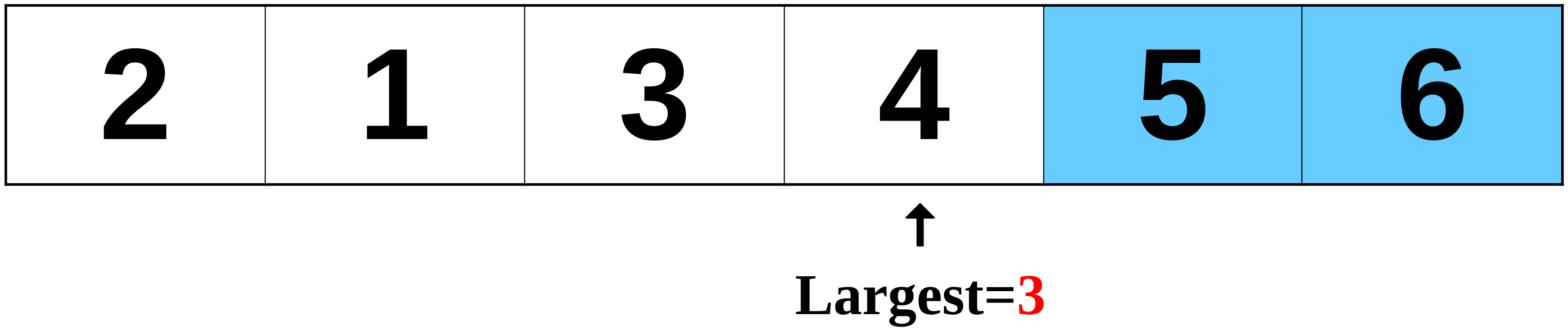


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

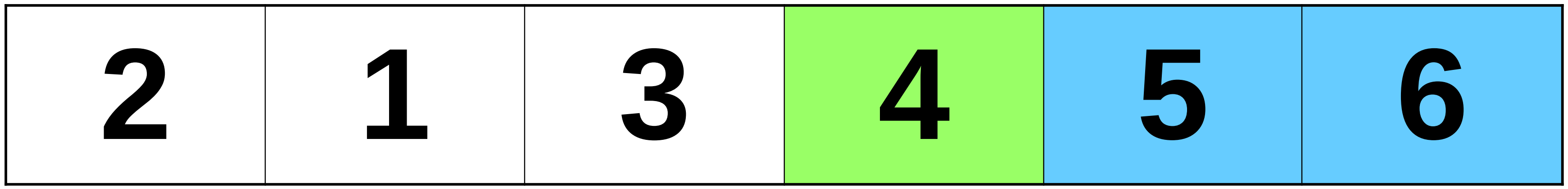


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

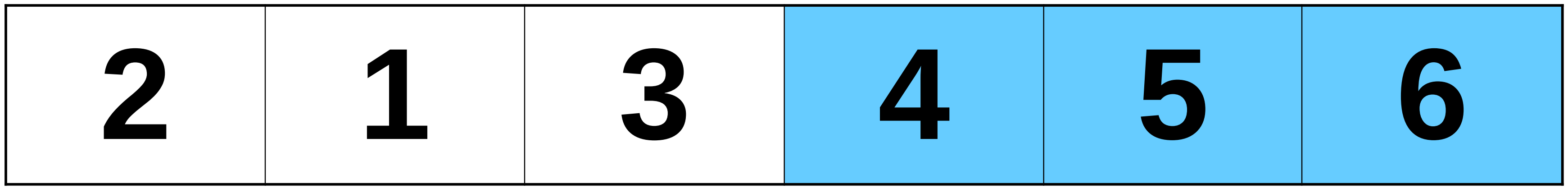


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

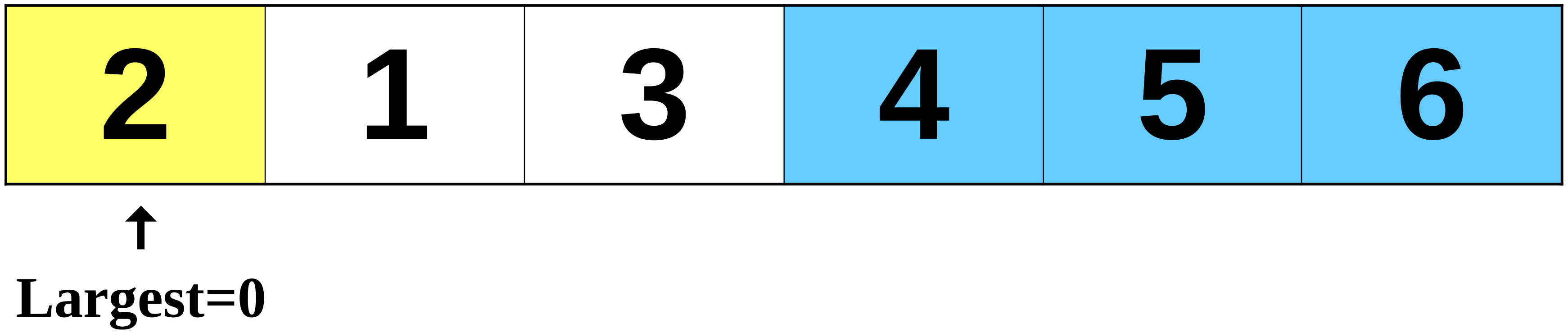


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

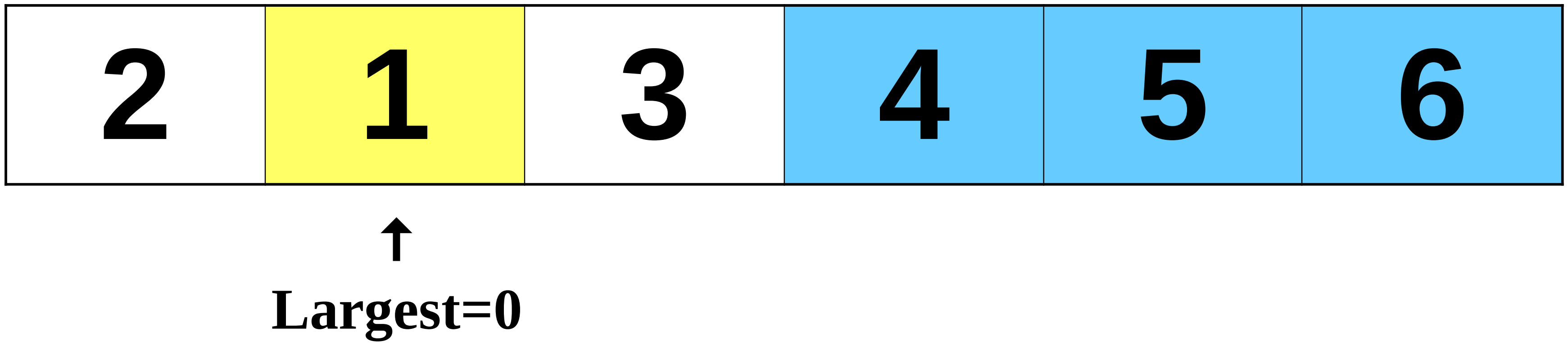


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

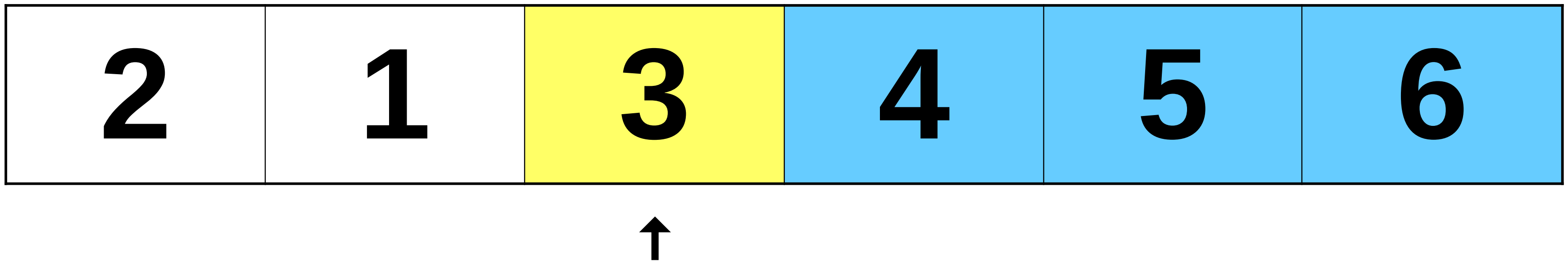


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



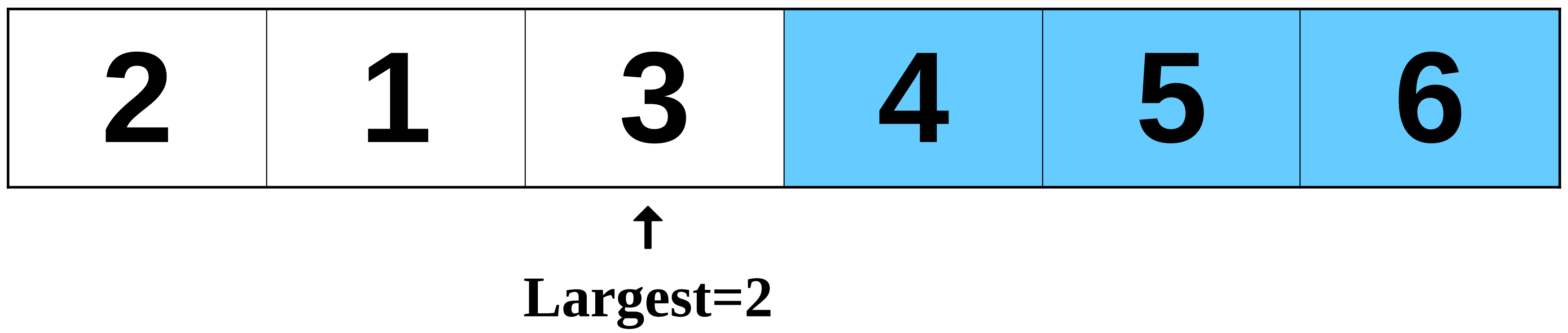
Largest=2

🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

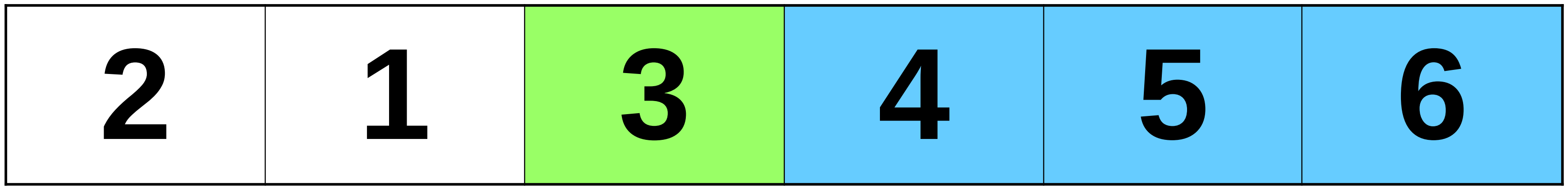


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

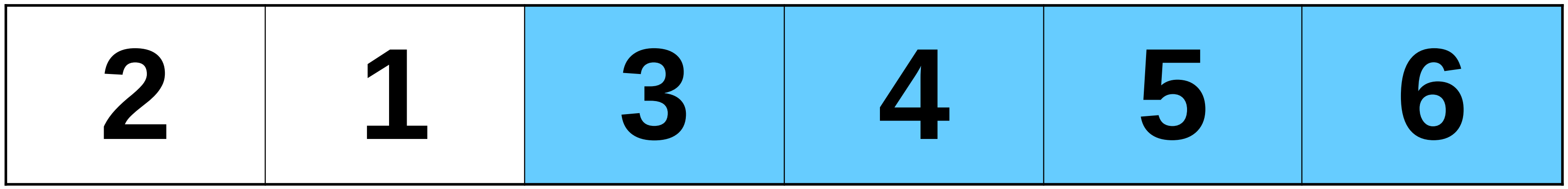


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

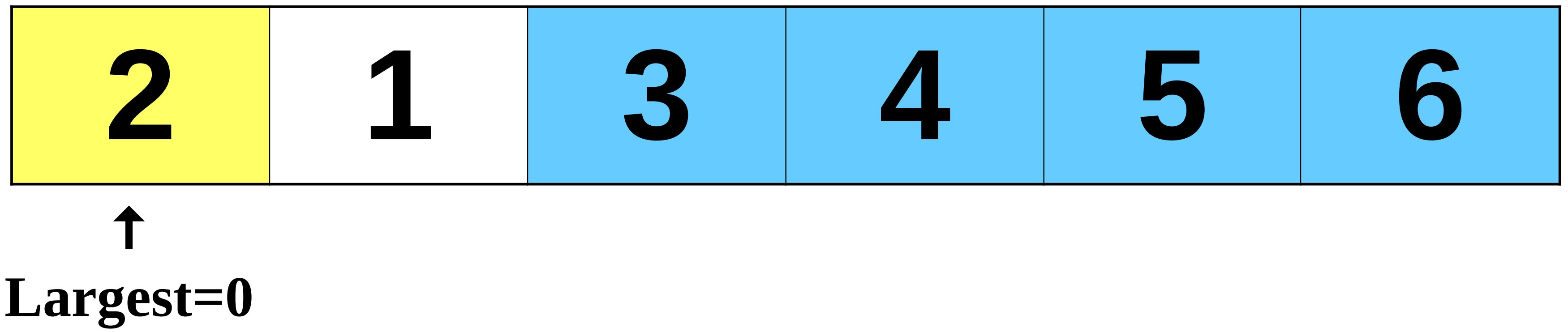


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

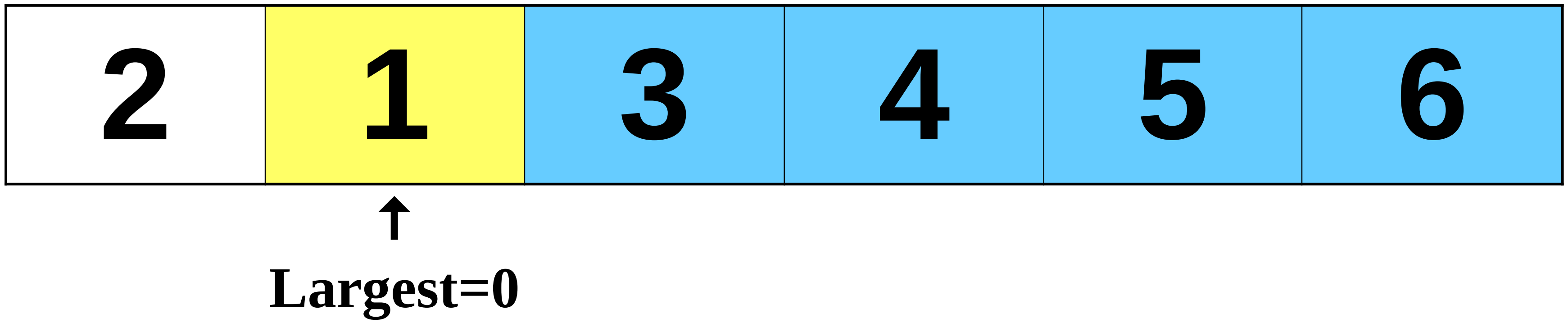


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

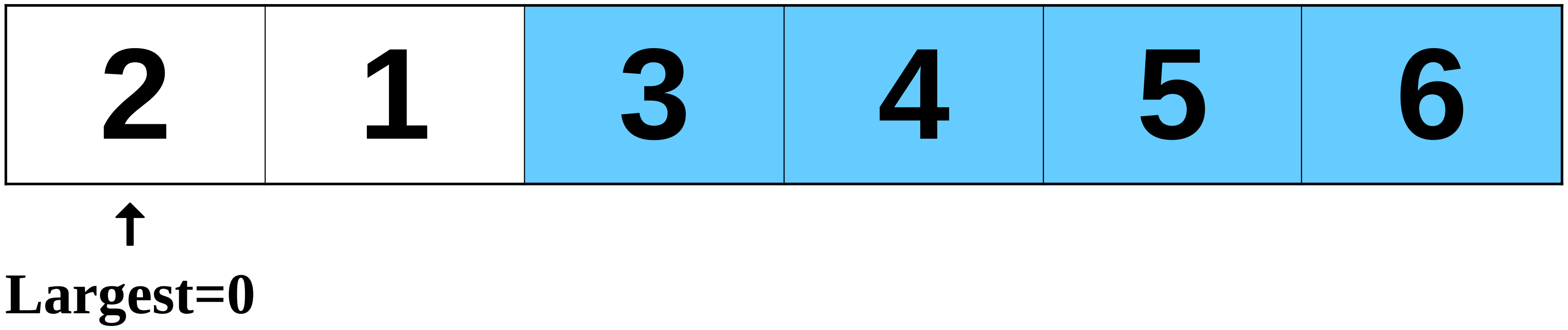


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

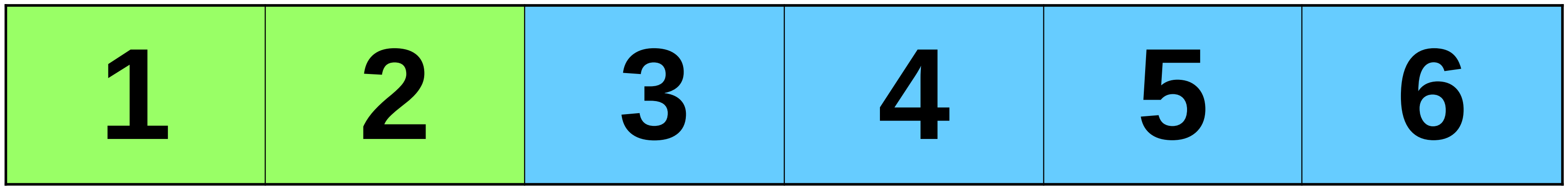


🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort



🟨 Comparison

🟩 Data Movement

🟦 Sorted

Selection Sort

1	2	3	4	5	6

DONE!

🟨	Comparison
🟩	Data Movement
🟦	Sorted

선택 정렬 (3)

◆ Sorting 클래스 메소드 멤버 구현

```
public class Sorting  {
    public static void selectionSort(int[] a)  {
      int i, j, min;
      for (i = 0; i < a.length – 1; i++)  {
            for (j = i+1, min = i; j < a.length; j++)  {     // a[j] … a[a.length] 사이에
                                                            // 가장 작은 원소의 인덱스(min)을 찾음
               if (a[j] < a[min]) min = j;
       }
       swap(a, min, i);      // a[i]와 a[min]을 교환
    }

    public static void swap(int[] a, int j, int k)  {     // a[j]와 a[k]를 교환
       int temp = a[j];   a[j] = a[k];   a[k] = temp;
    }

    // • • • • • •
    // bubleSort(), insertionSort(), quickSort() 등 기타 다른 메소드들을 추가로 포함
    // • • • • • •
}
```

선택 정렬 (4)

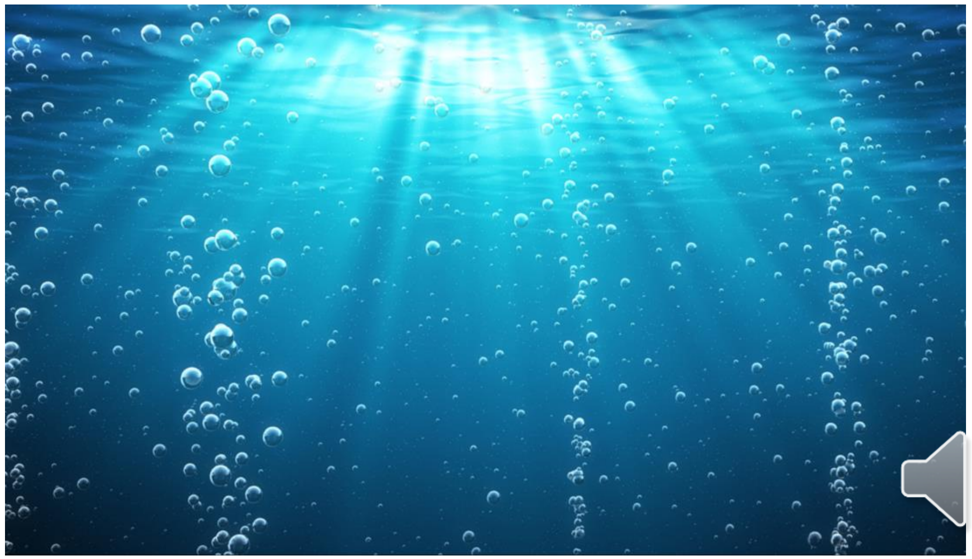
◆ SortMain 클래스 (샘플 프로그램)

```
public class SortMain  {
    public static void main(String[] args)  {
        int[] a = { 5, 2, 8, 3, 1};
        System.out.println("정렬전 배열 원소 : ");
        int i;
        for (i = 0; i < a.length; i++)
            System.out.print(a[i] + "  ");
        System.out.println();
        //Sorting.selectionSort(a);     // 원하는 정렬 메소드를 선택
        Sorting a1 = new Sorting();   //   ?
        a1.selectionSort(a);    //?
        System.out.println("정렬된 배열 원소 : ");
        for (i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();
    }
}
```

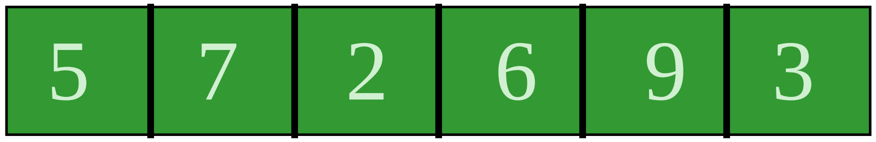
<실행결과>

```
정렬전 배열 원소 :
5  2  8  3  1
정렬된 배열 원소 :
1  2  3  5  8
```

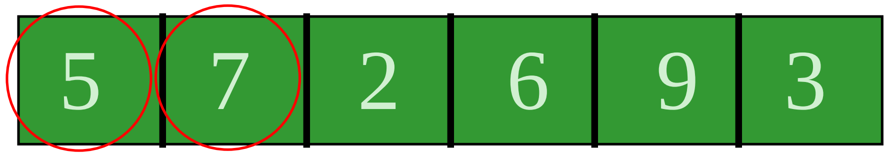
Bubble Sort(버블정렬)



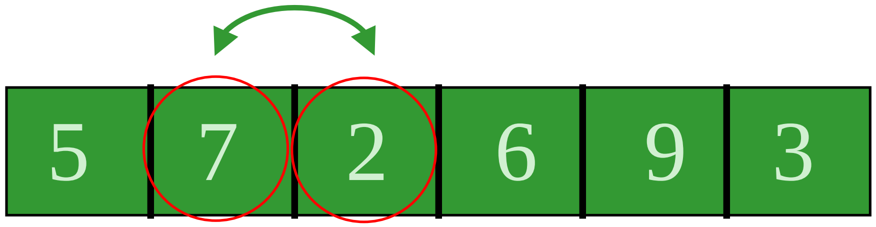
Bubble Sort



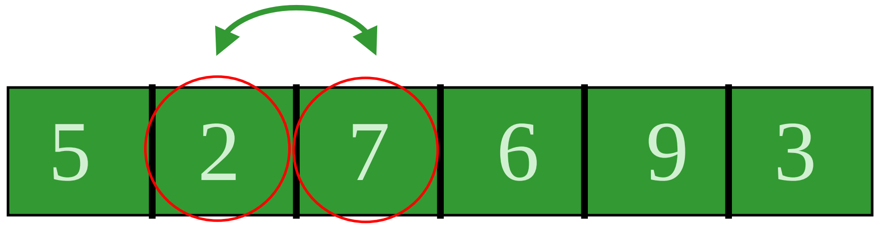
Bubble Sort



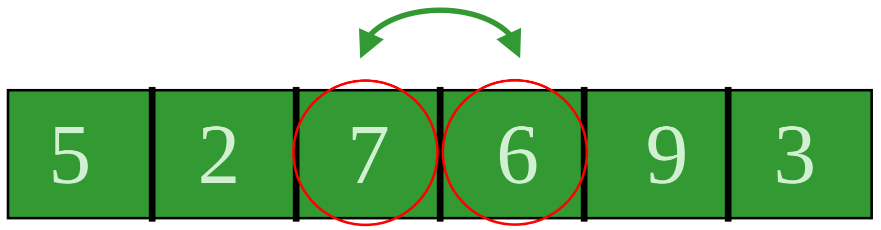
Bubble Sort



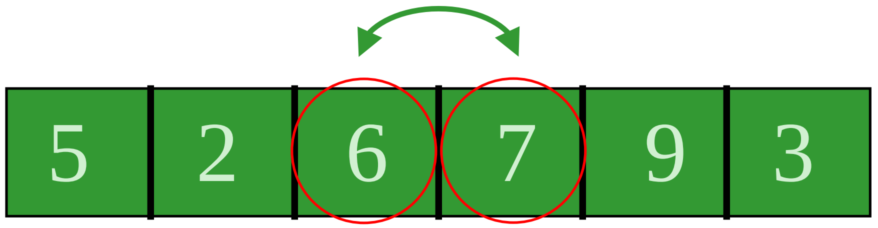
Bubble Sort



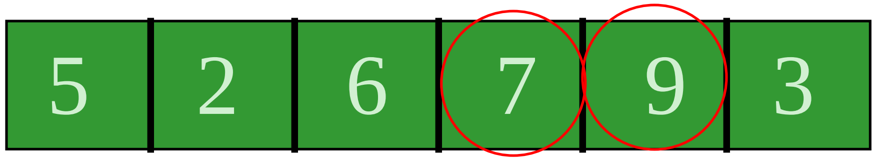
Bubble Sort



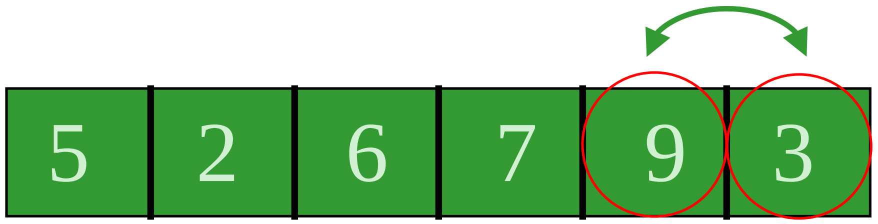
Bubble Sort



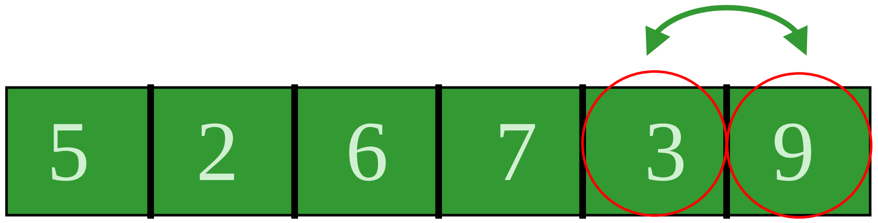
Bubble Sort



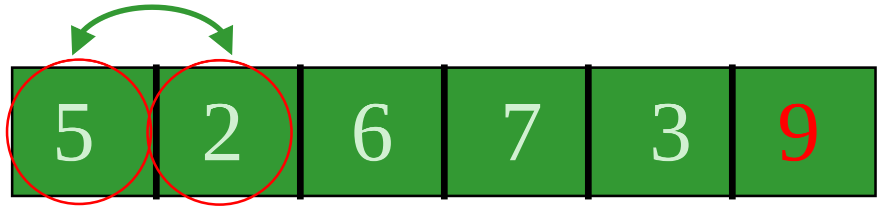
Bubble Sort



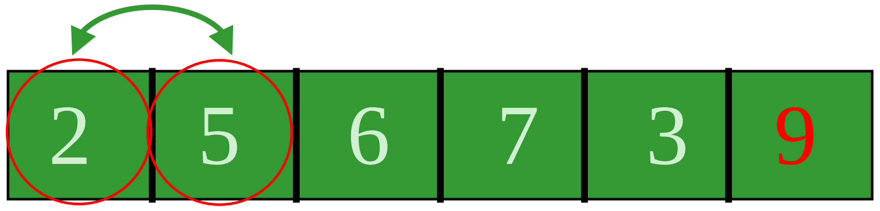
Bubble Sort



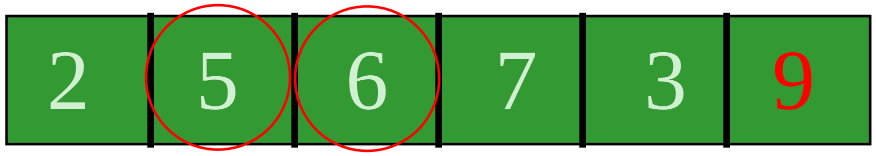
Bubble Sort



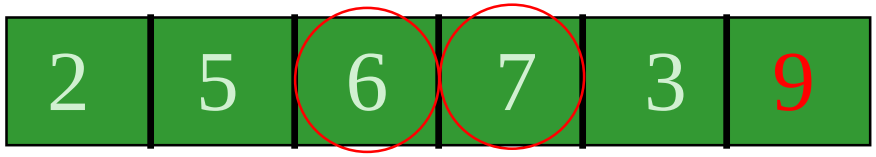
Bubble Sort



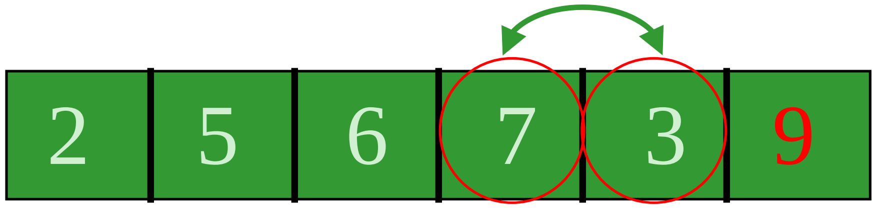
Bubble Sort



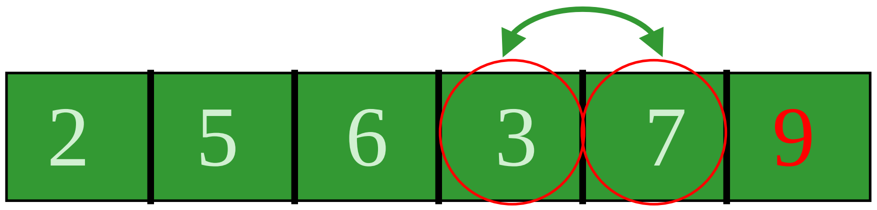
Bubble Sort



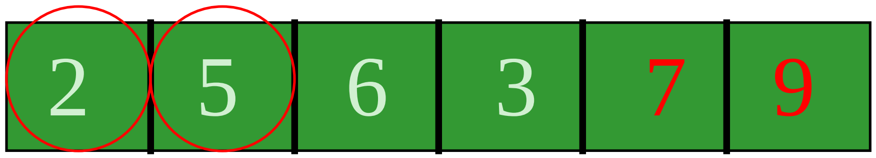
Bubble Sort



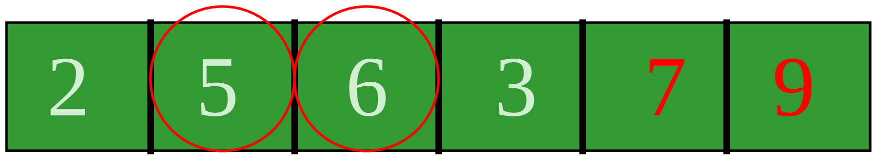
Bubble Sort



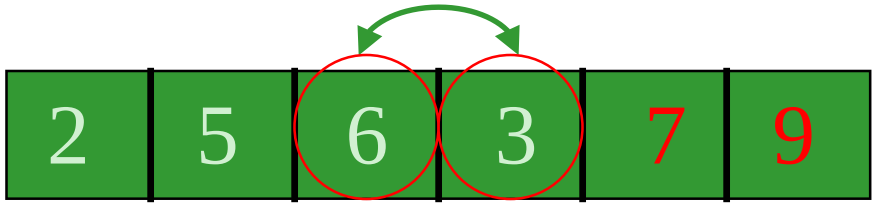
Bubble Sort



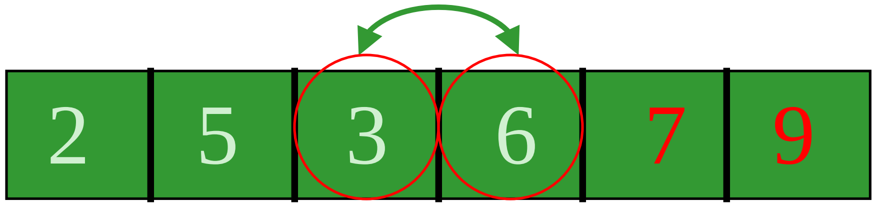
Bubble Sort



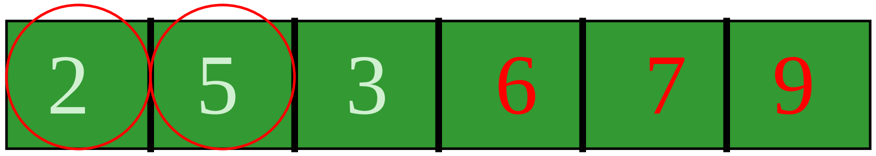
Bubble Sort



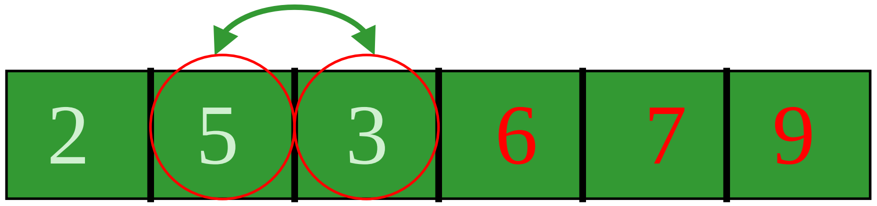
Bubble Sort



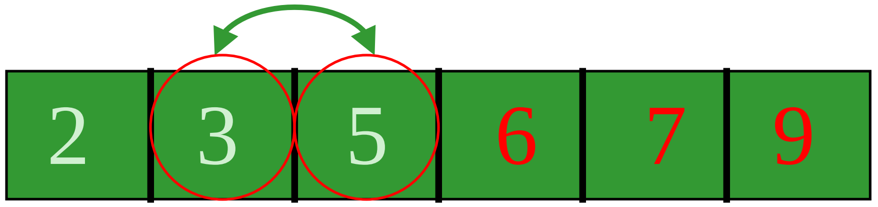
Bubble Sort



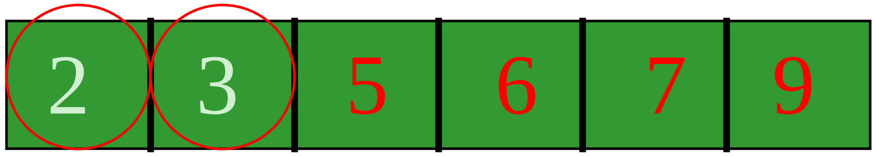
Bubble Sort



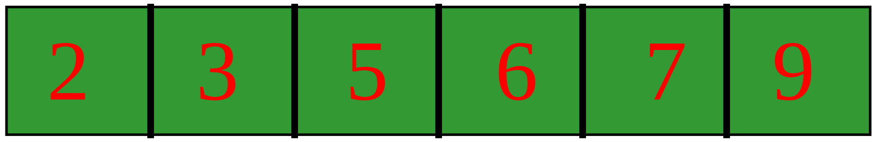
Bubble Sort



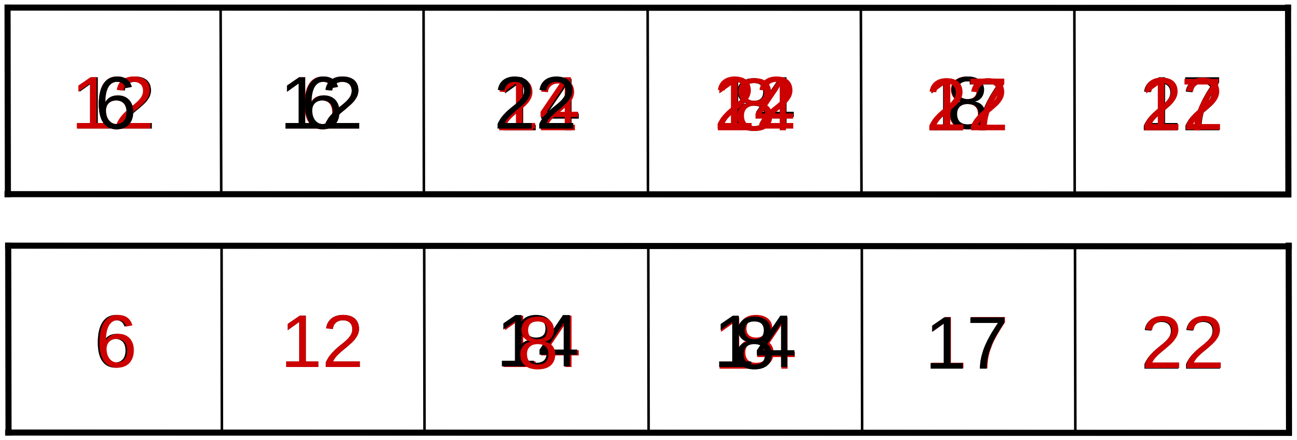
Bubble Sort



Bubble Sort



Bubble Sort



- Given n numbers to sort:
- Repeat the following n-1 times:
    - For each pair of adjacent numbers:
        - If the number on the left is greater than the number on the right, swap them.

Bubble Sort

6	12 8	12 8	14	17	22

6	8	12	14	17	22

- Given n numbers to sort:
- Repeat the following n-1 times:
  - For each pair of adjacent numbers:
    - If the number on the left is greater than the number on the right, swap them.

More Explanation?

Enough!

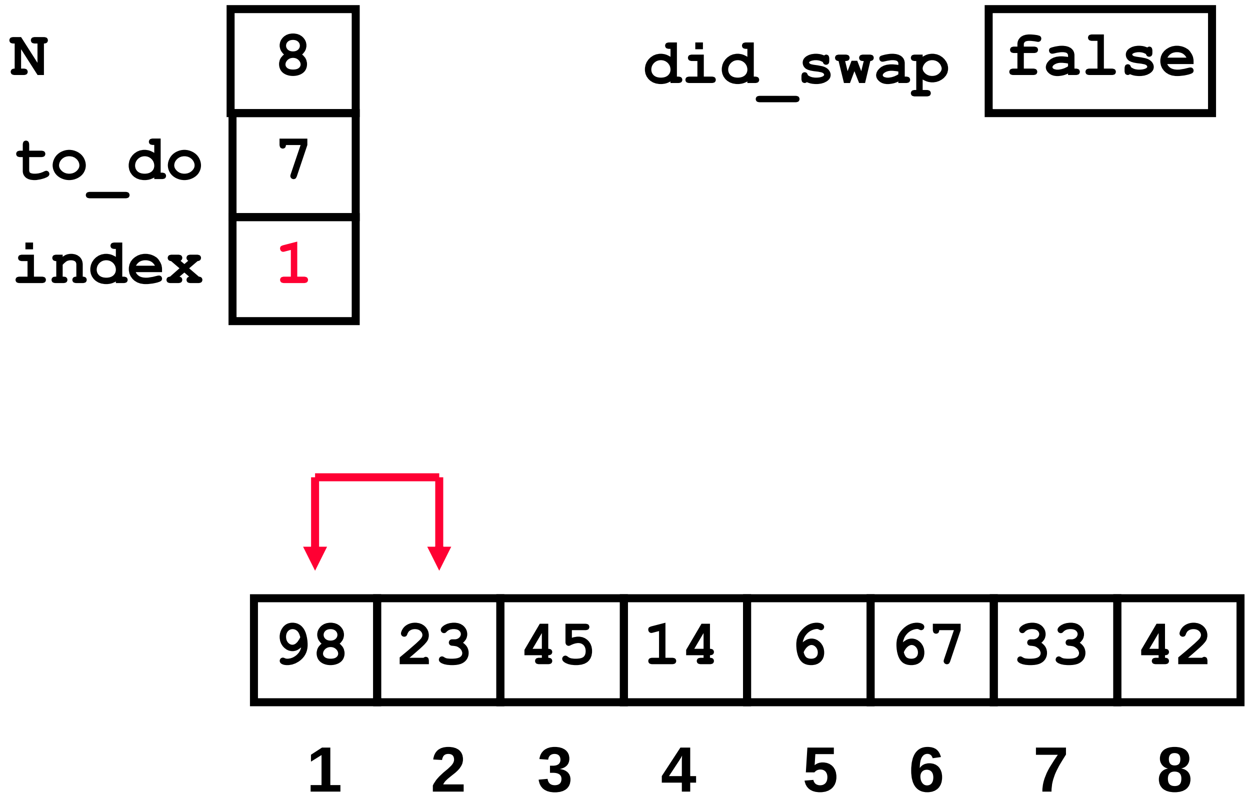
An Animated Example

N	8
to_do	7
index	

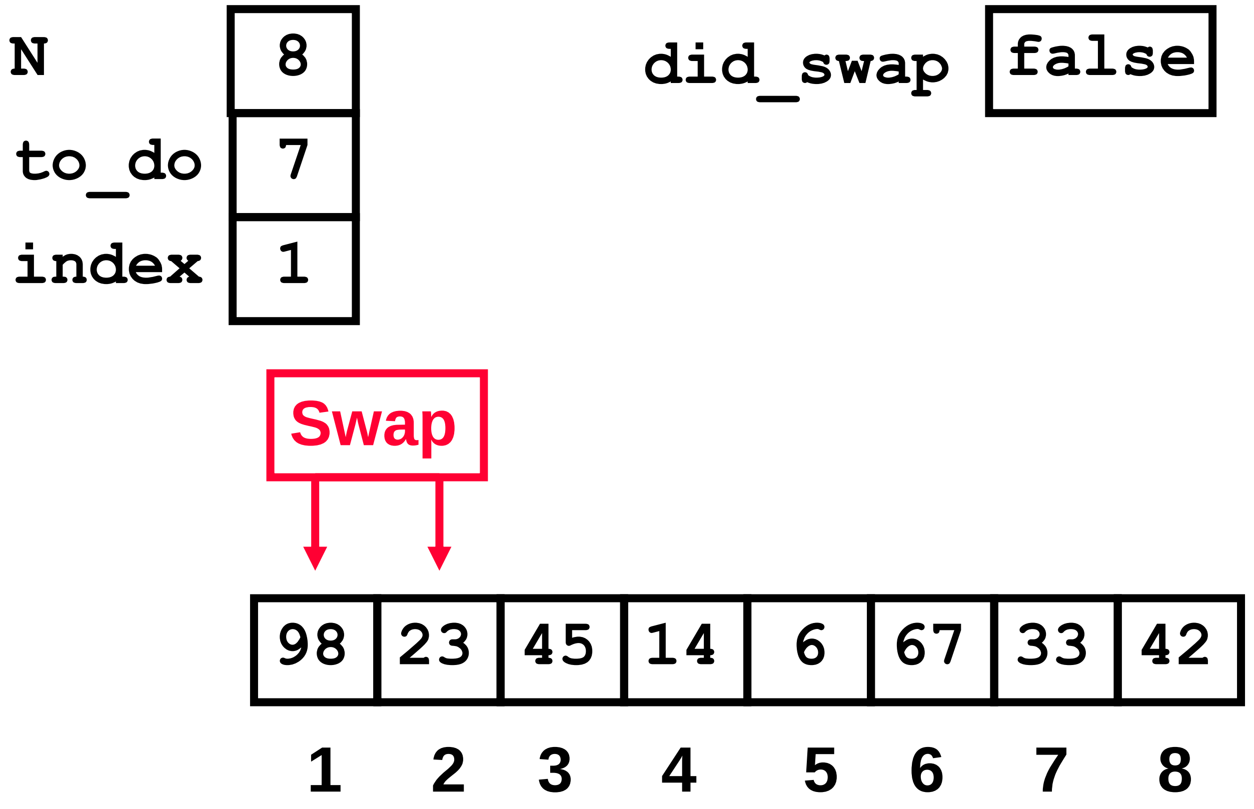
did_swap	true

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

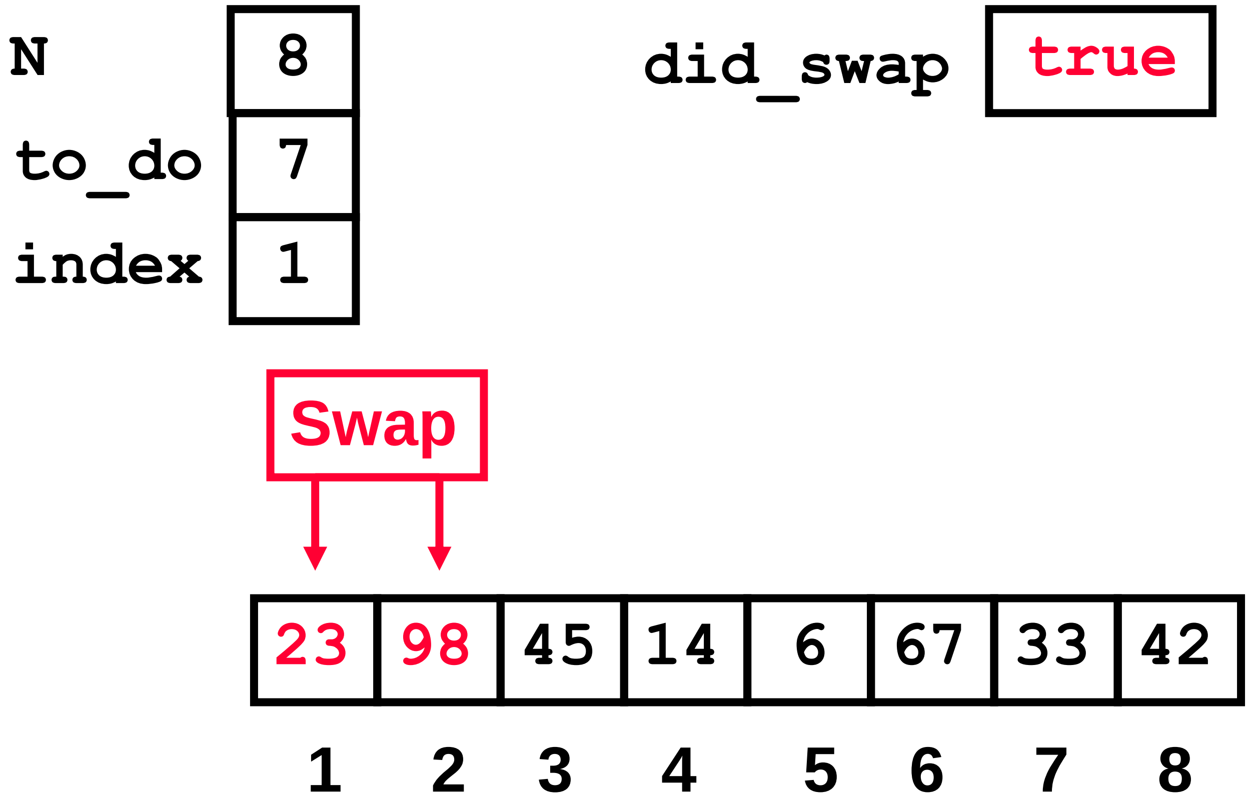
An Animated Example



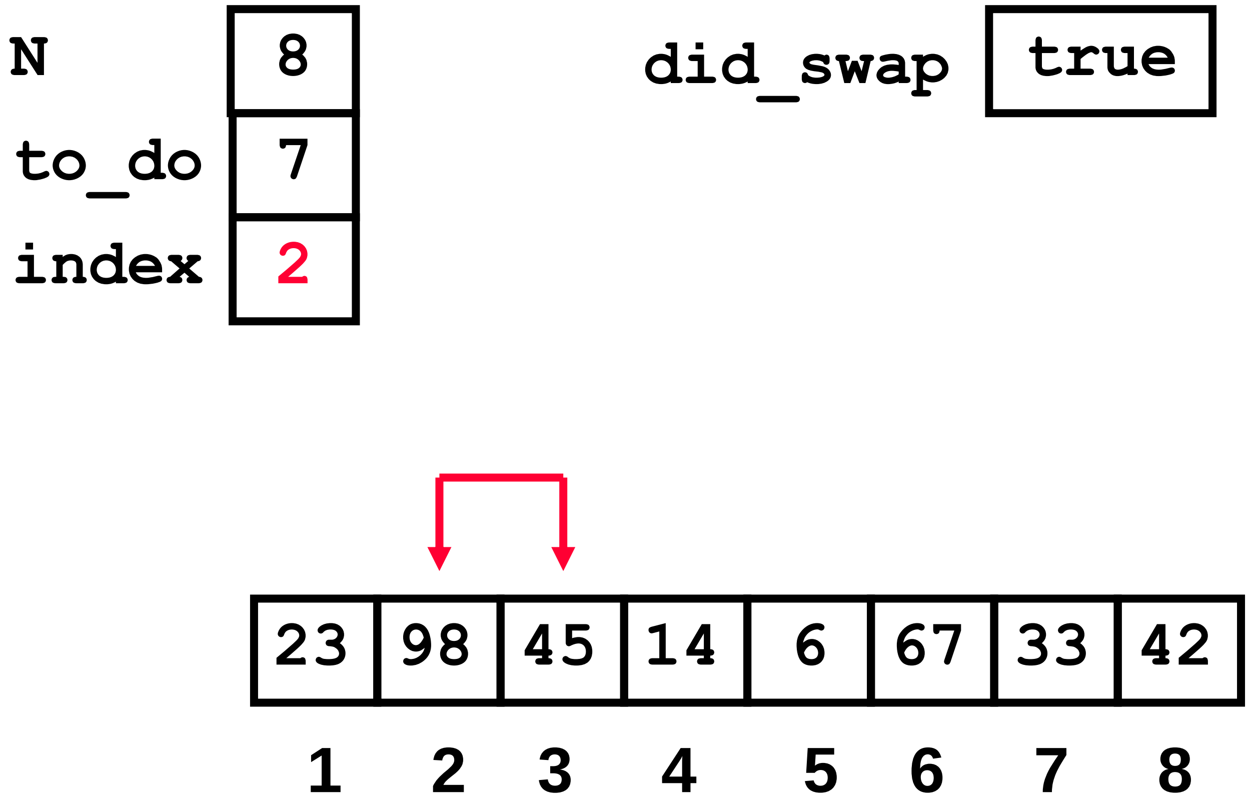
An Animated Example



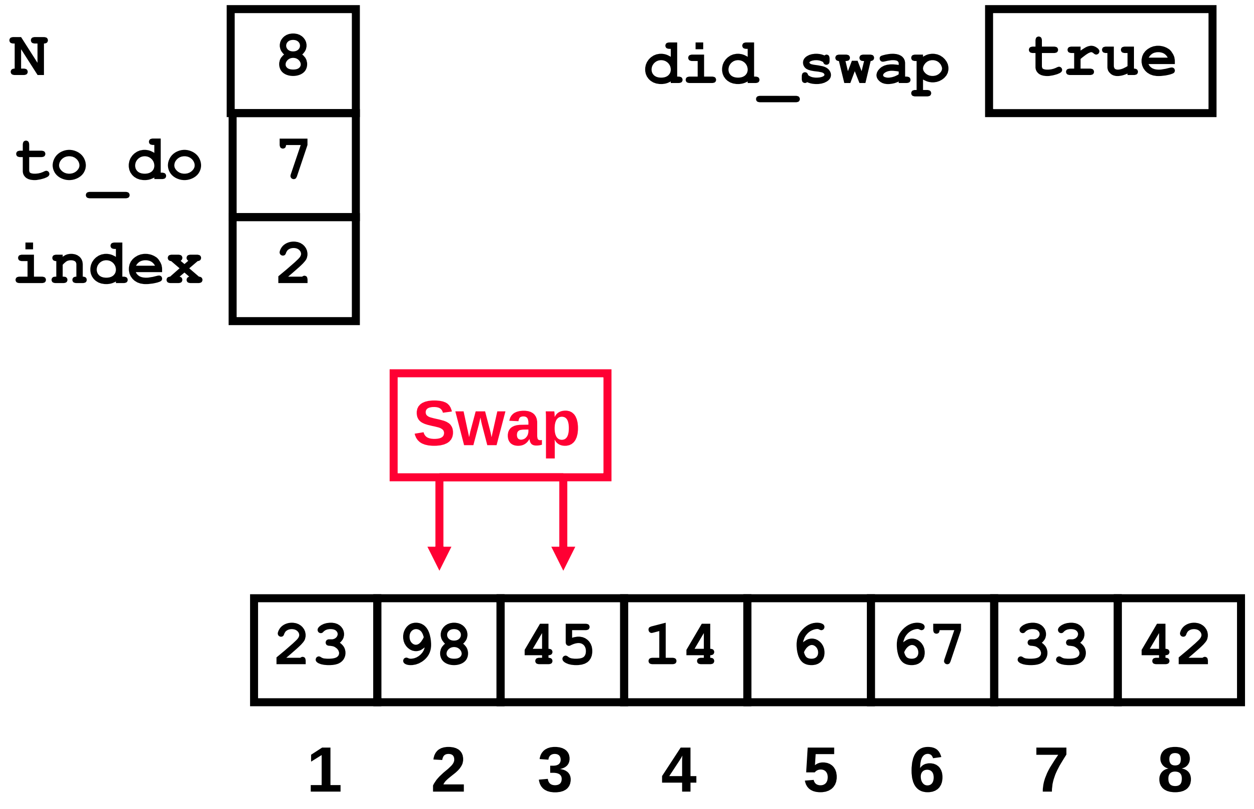
An Animated Example



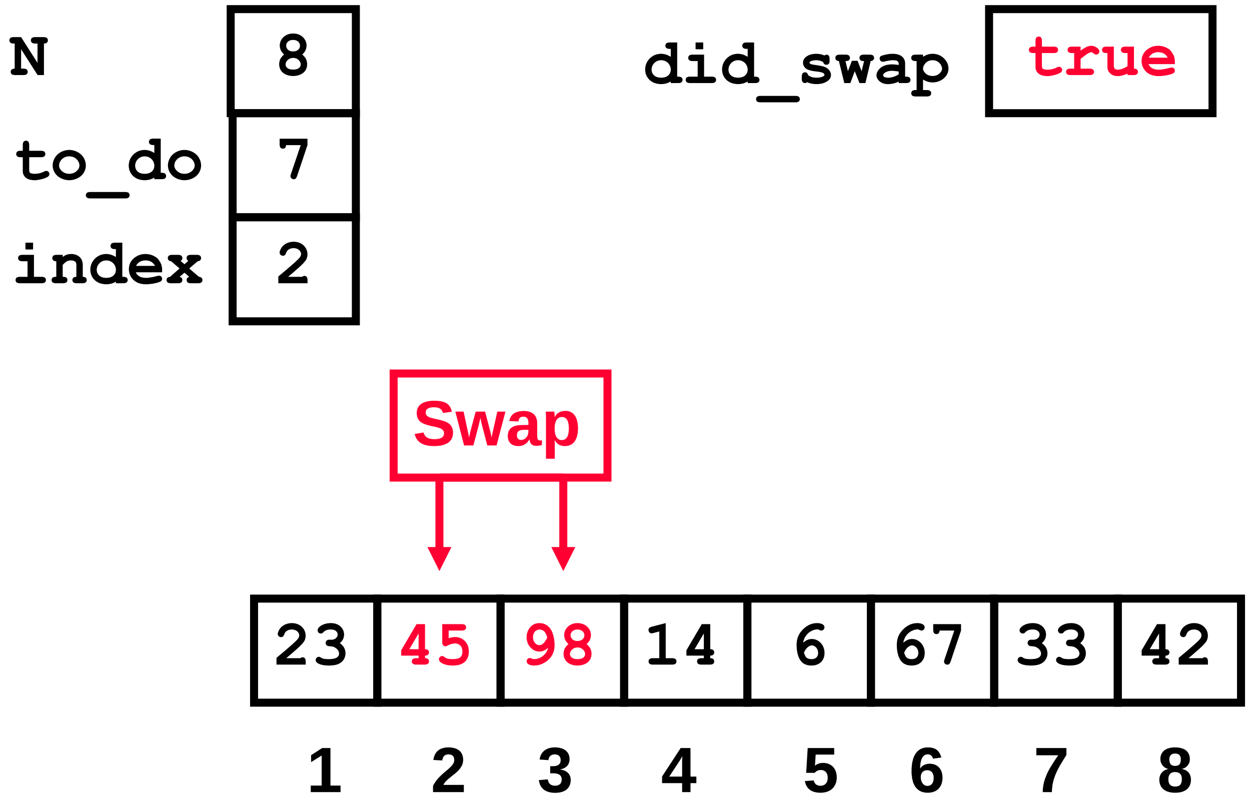
An Animated Example



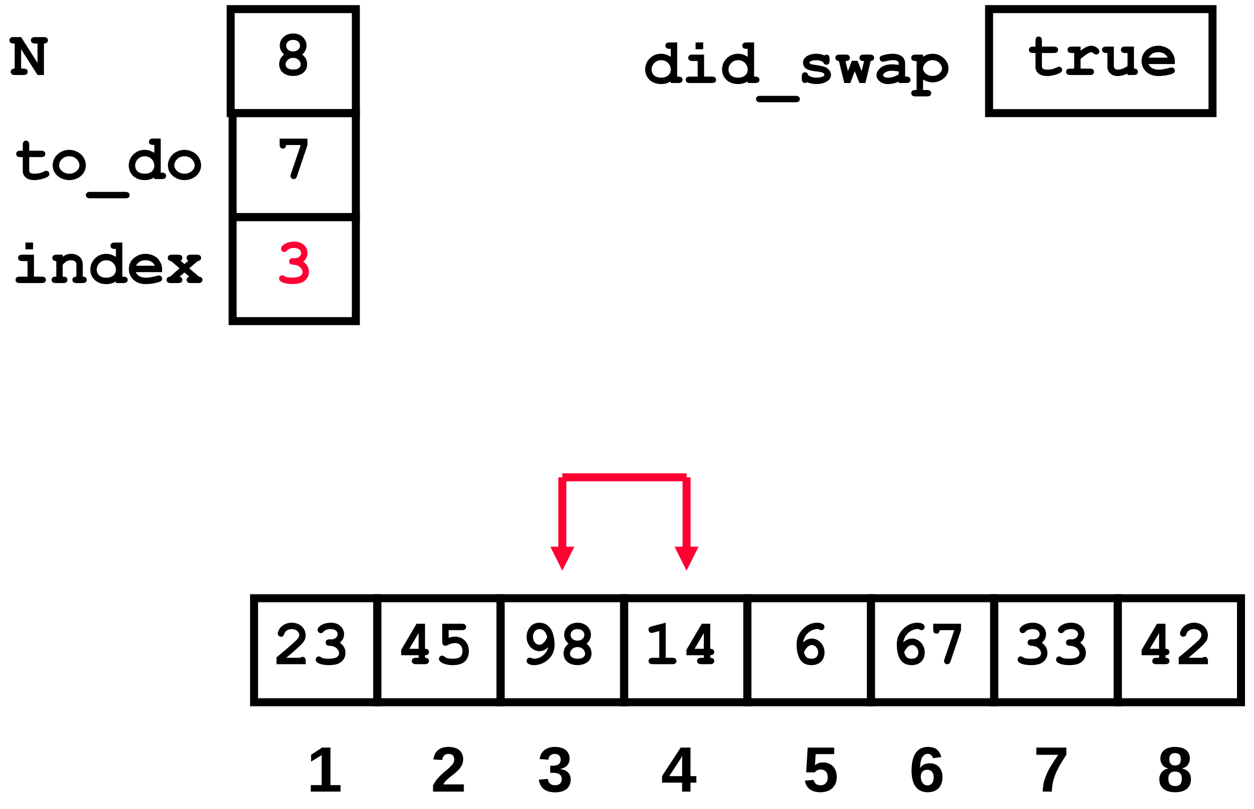
An Animated Example



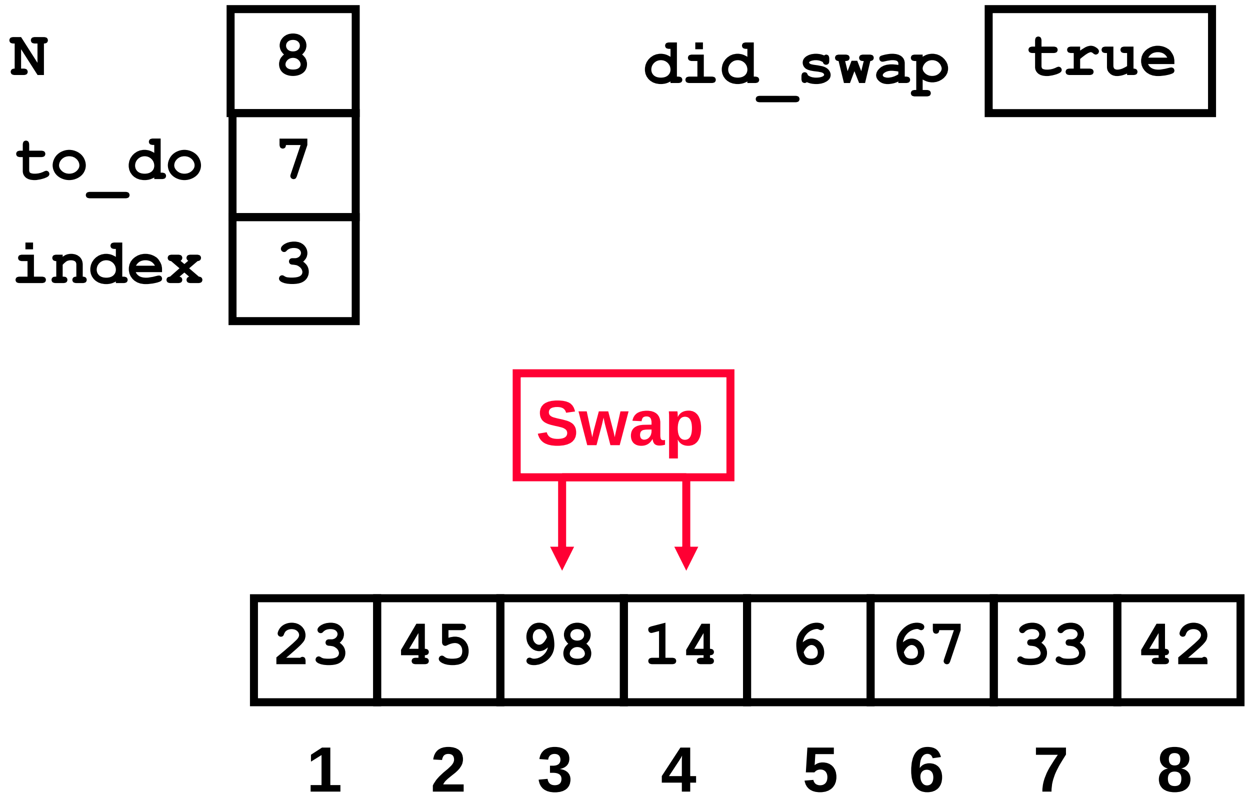
An Animated Example



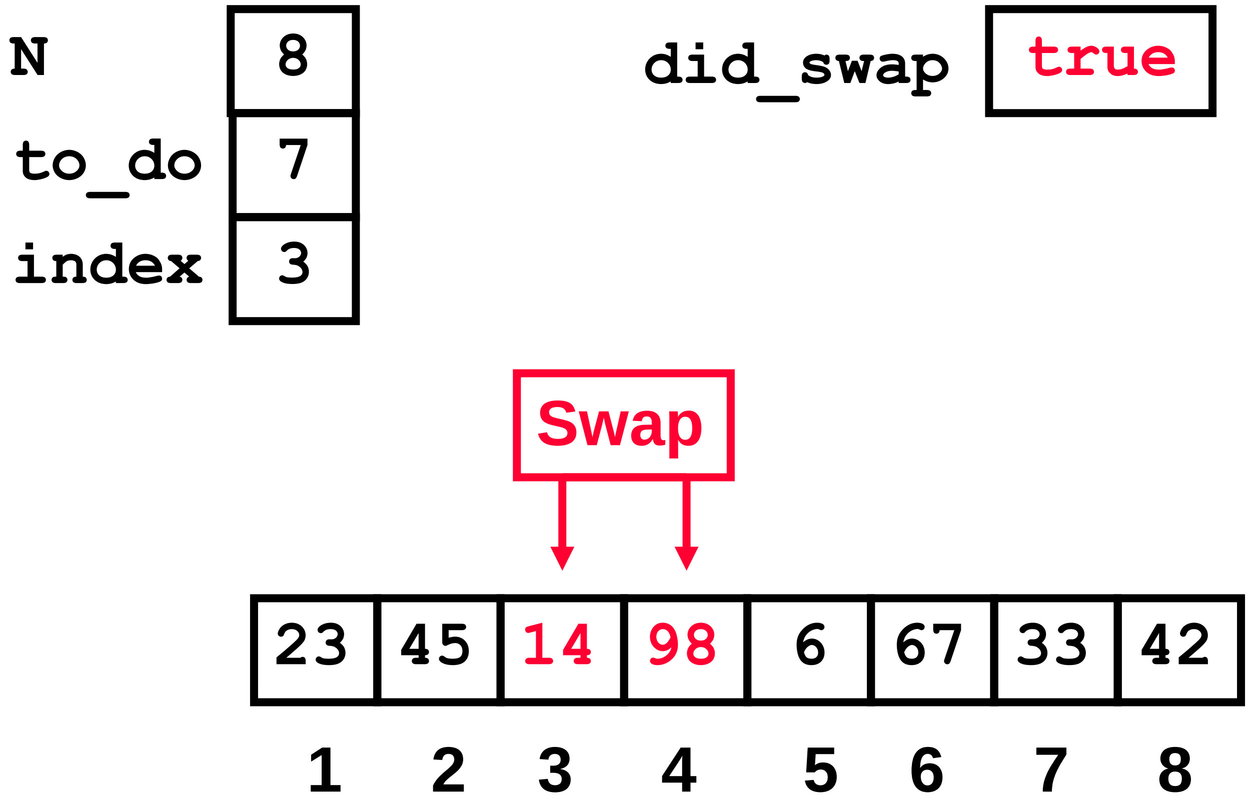
An Animated Example



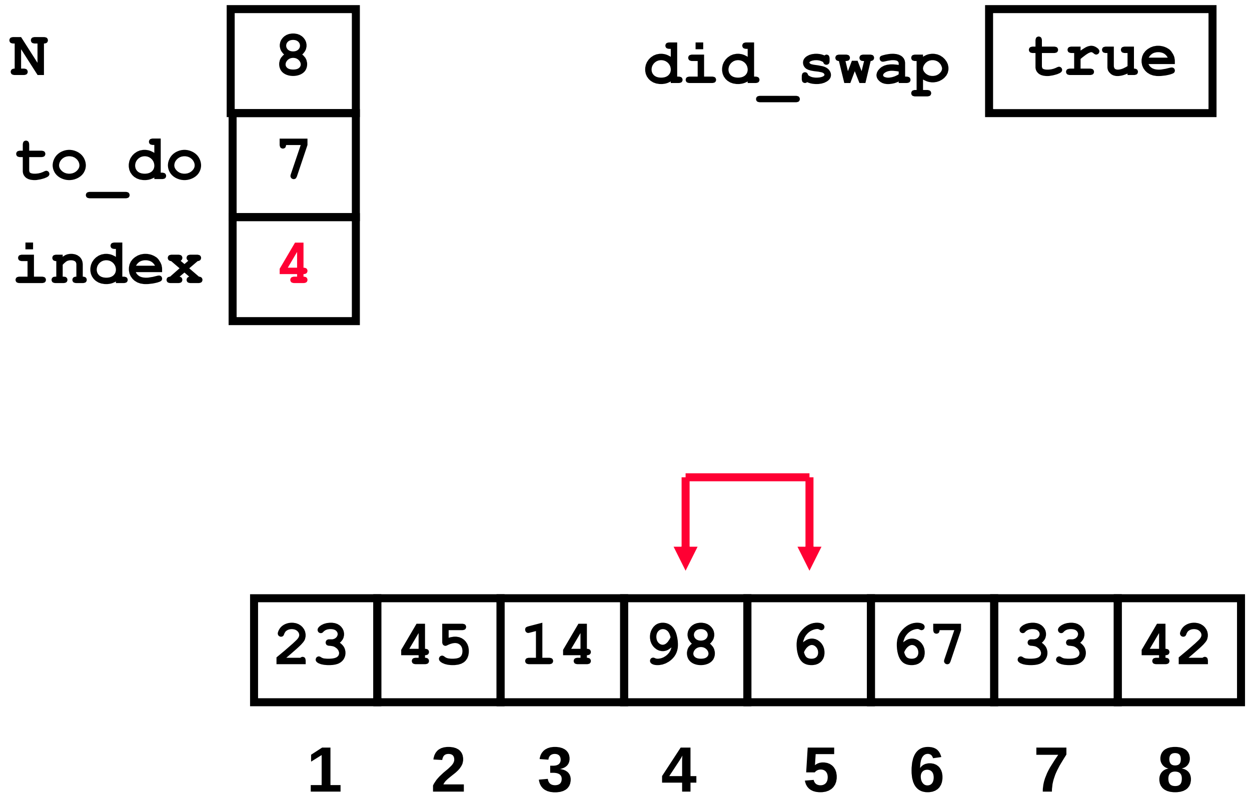
An Animated Example



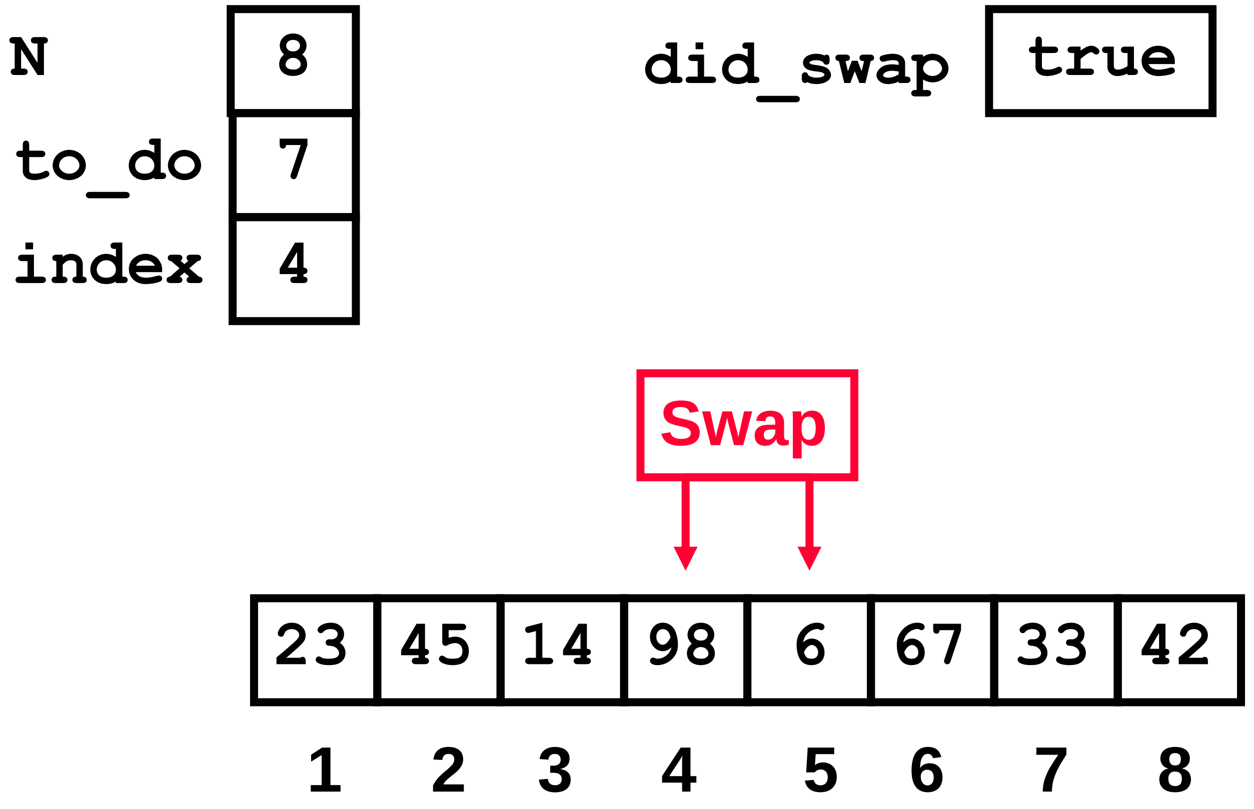
An Animated Example



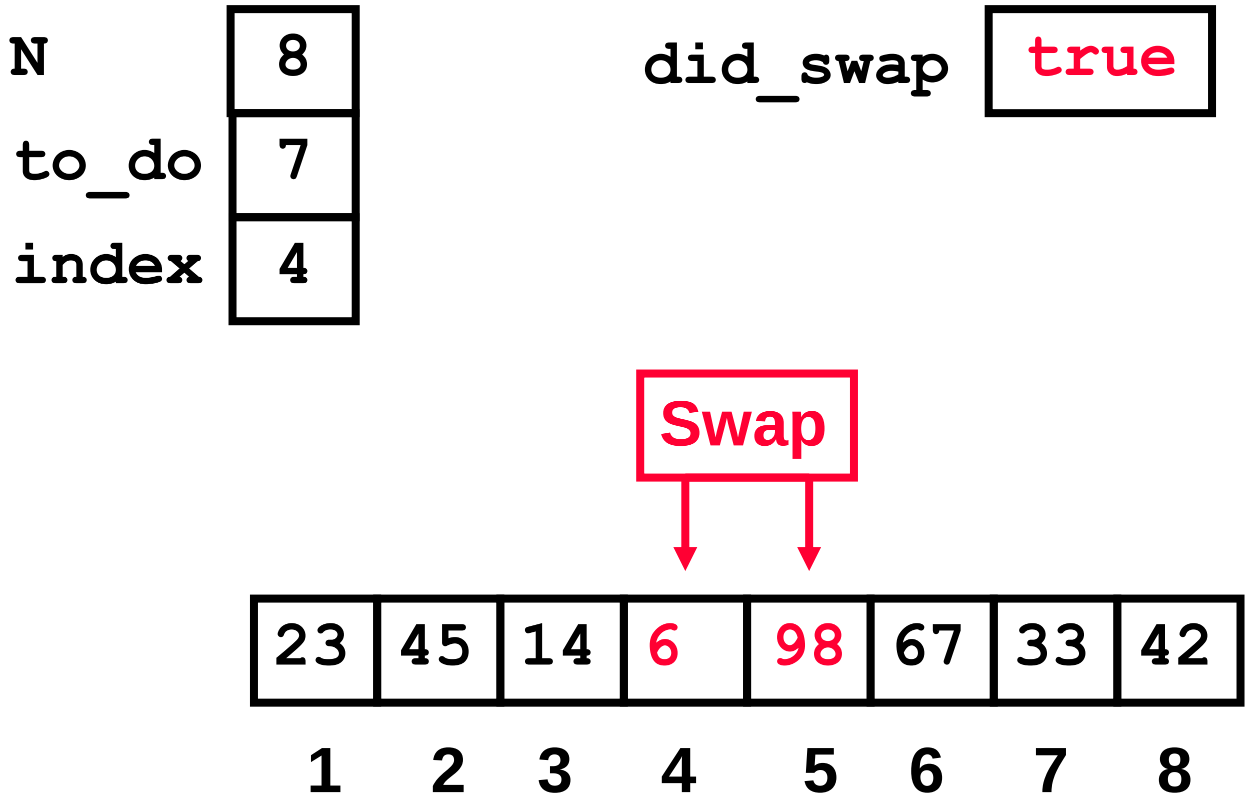
An Animated Example



An Animated Example



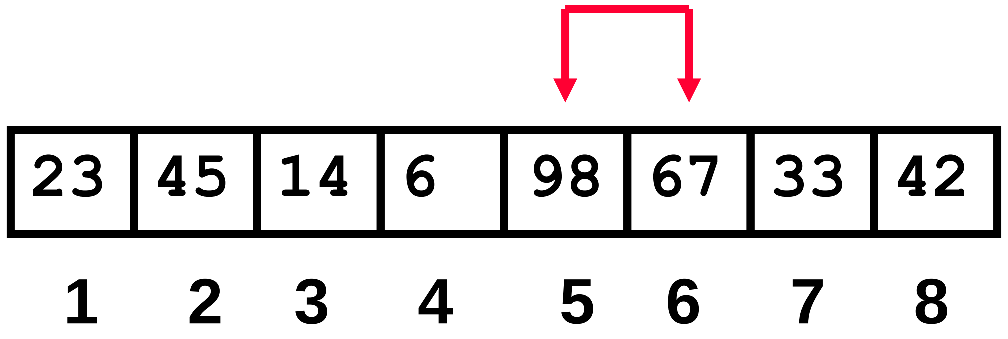
An Animated Example



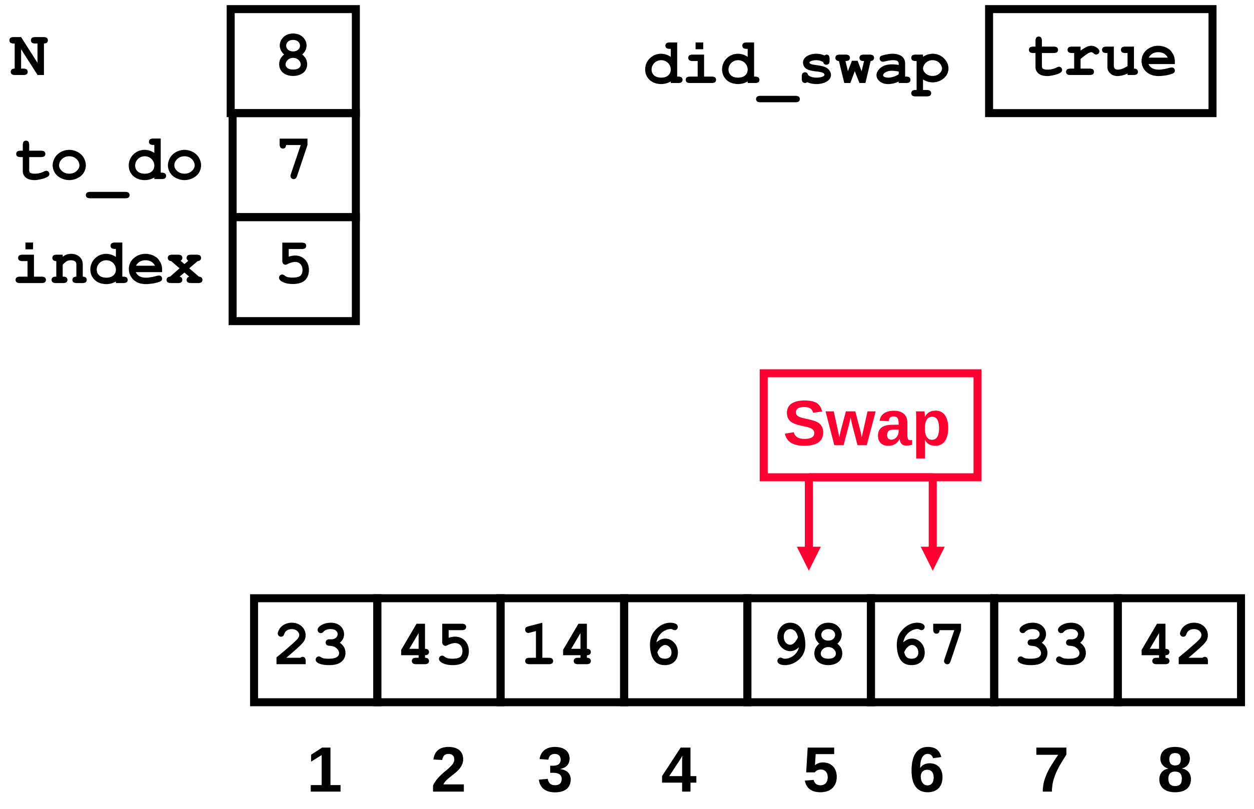
An Animated Example

`N`	`8`
`to_do`	`7`
`index`	`5`

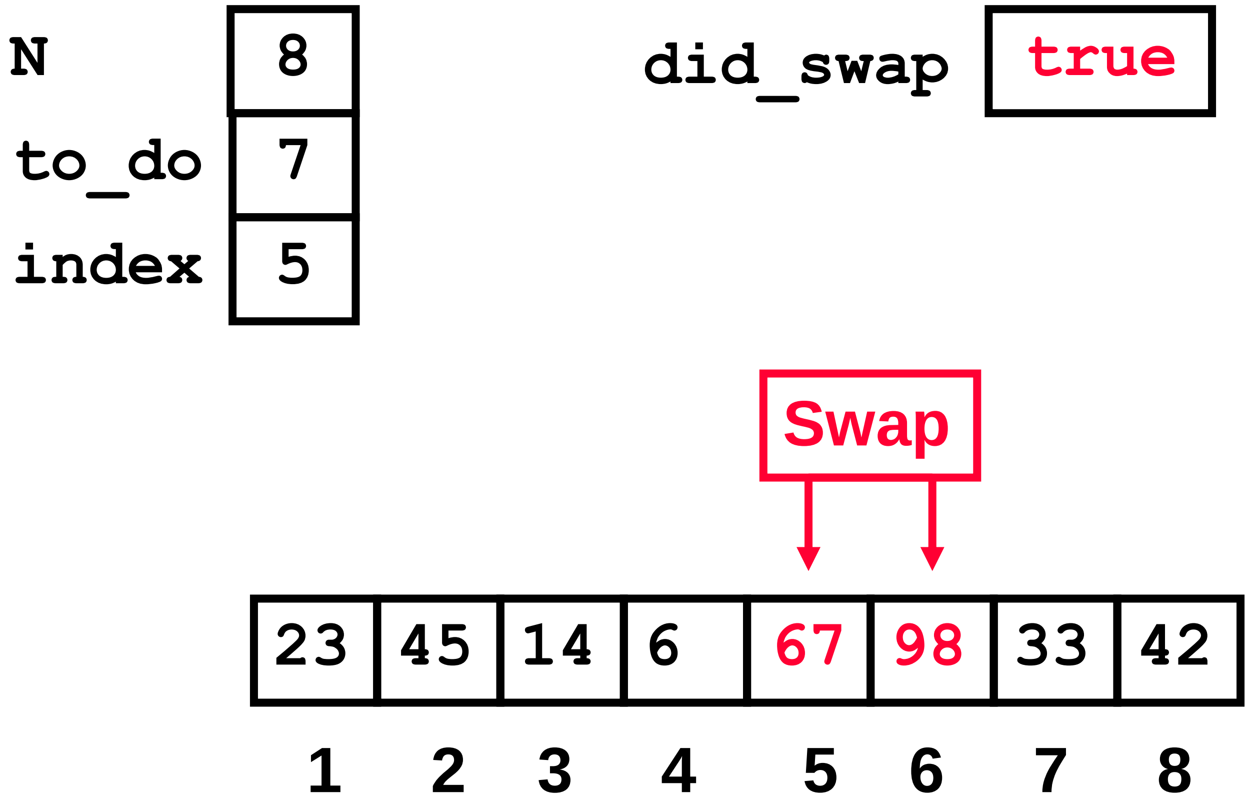
`did_swap` `true`



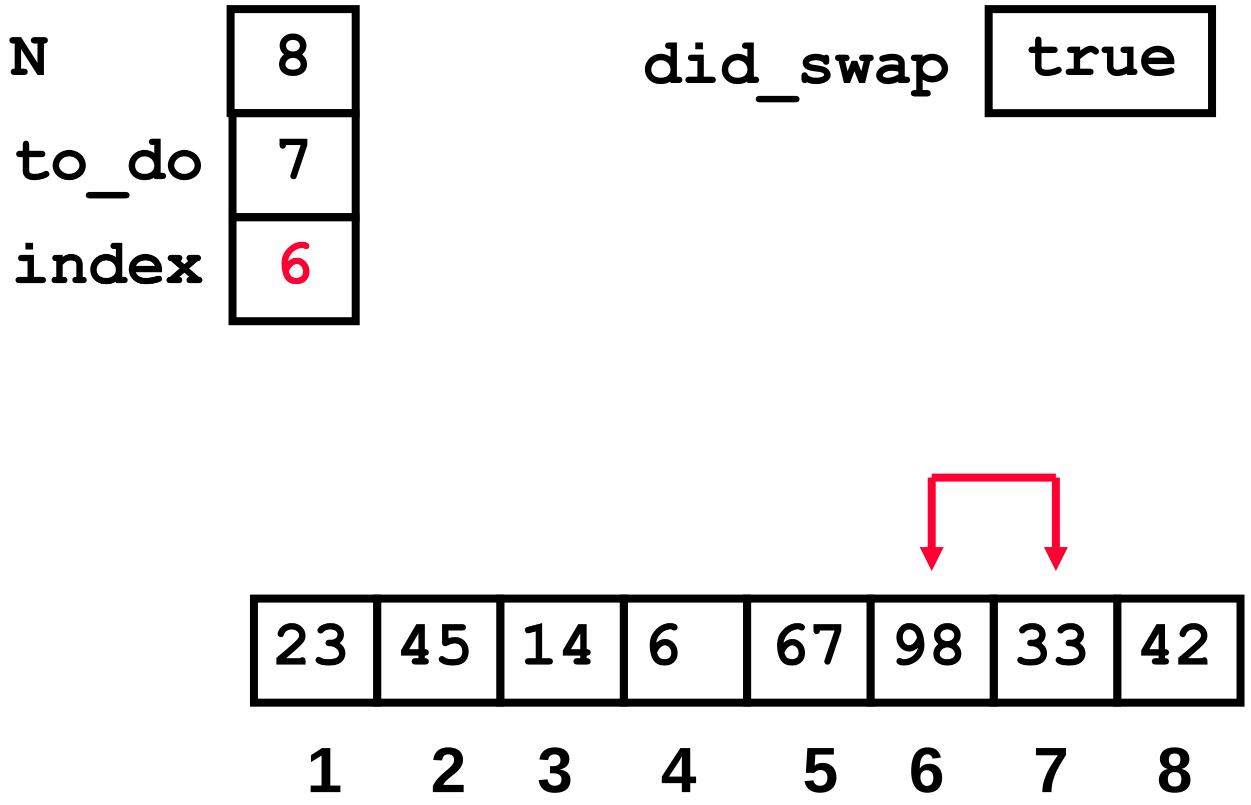
An Animated Example



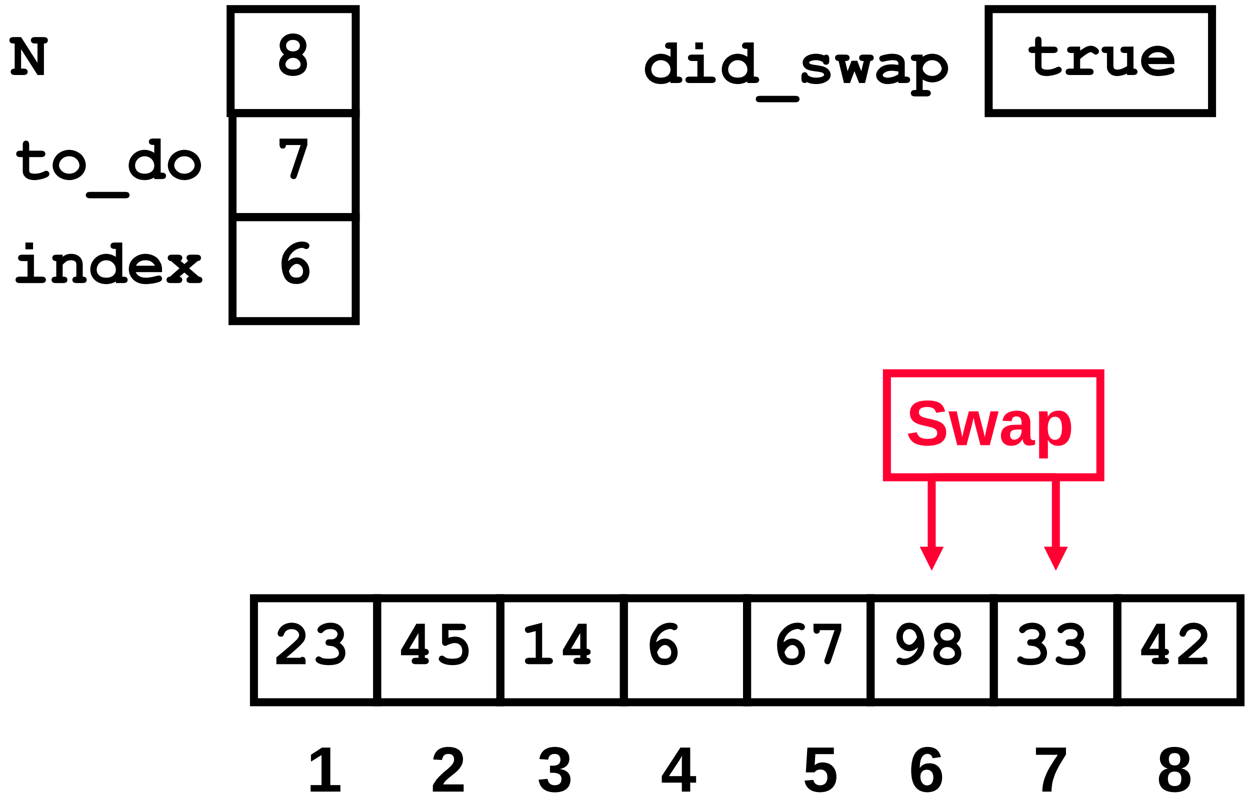
An Animated Example



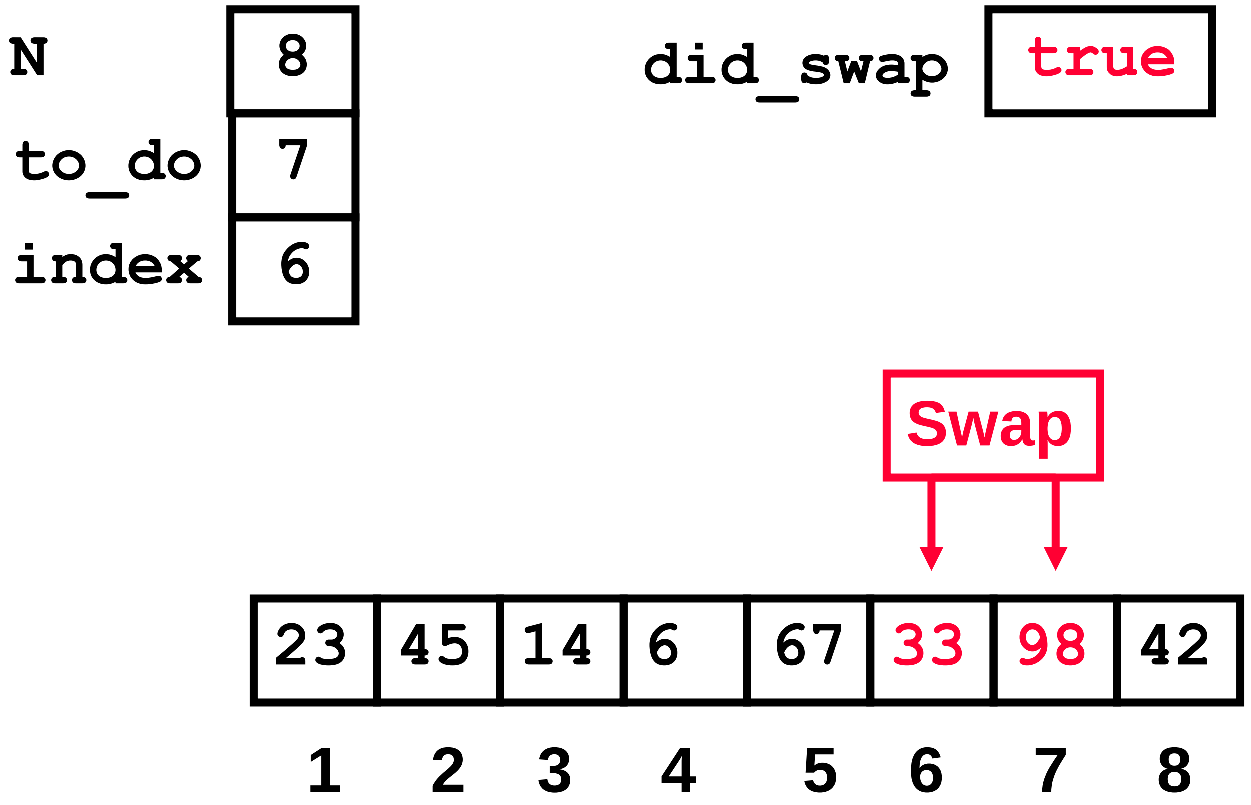
An Animated Example



An Animated Example



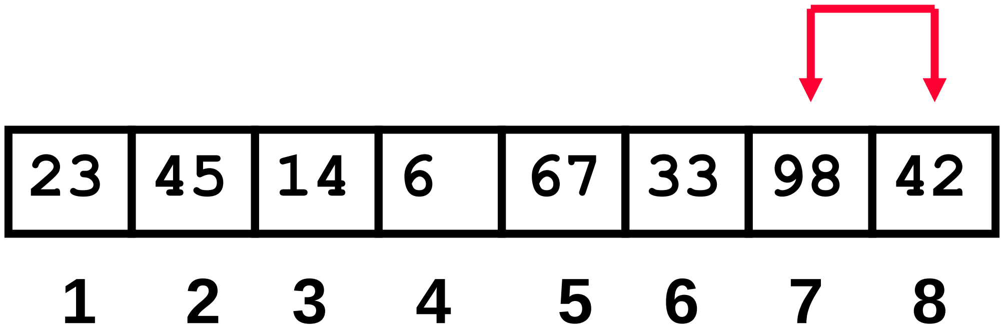
An Animated Example



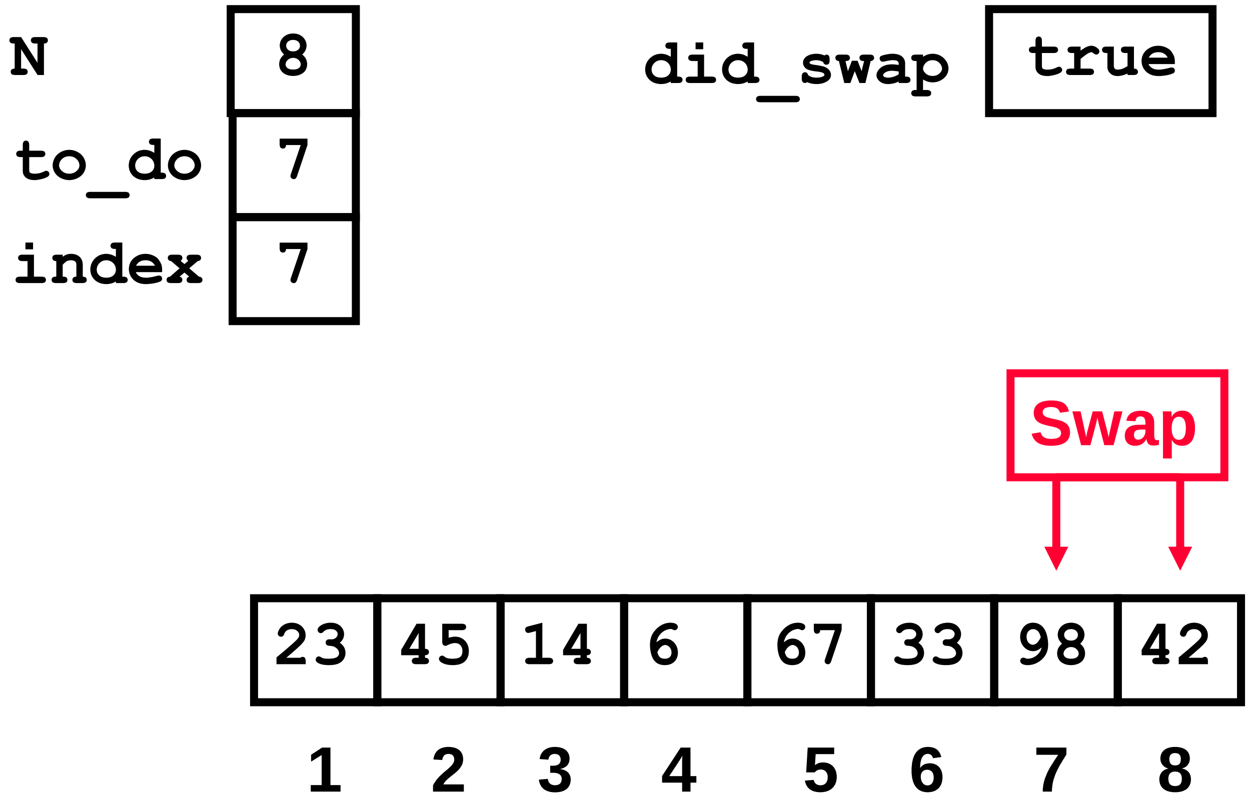
An Animated Example

N	8
to_do	7
index	7

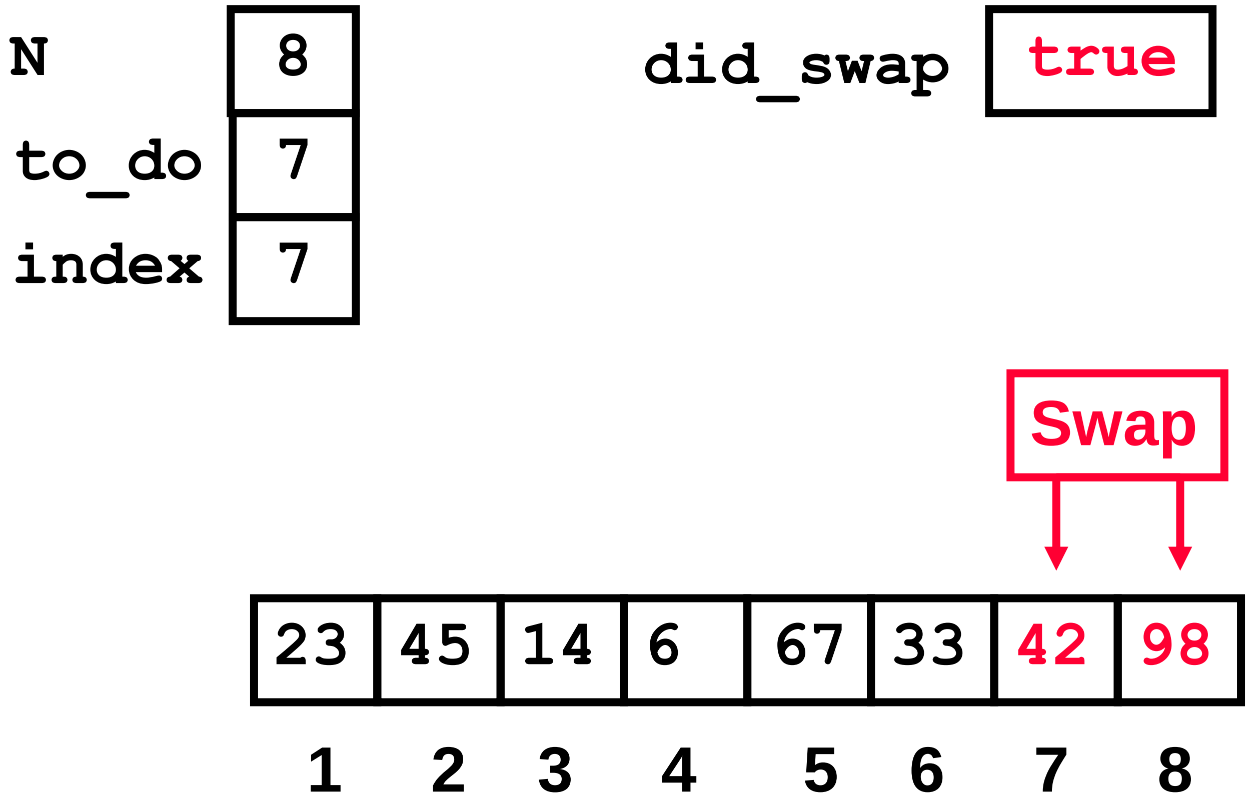
did_swap	true



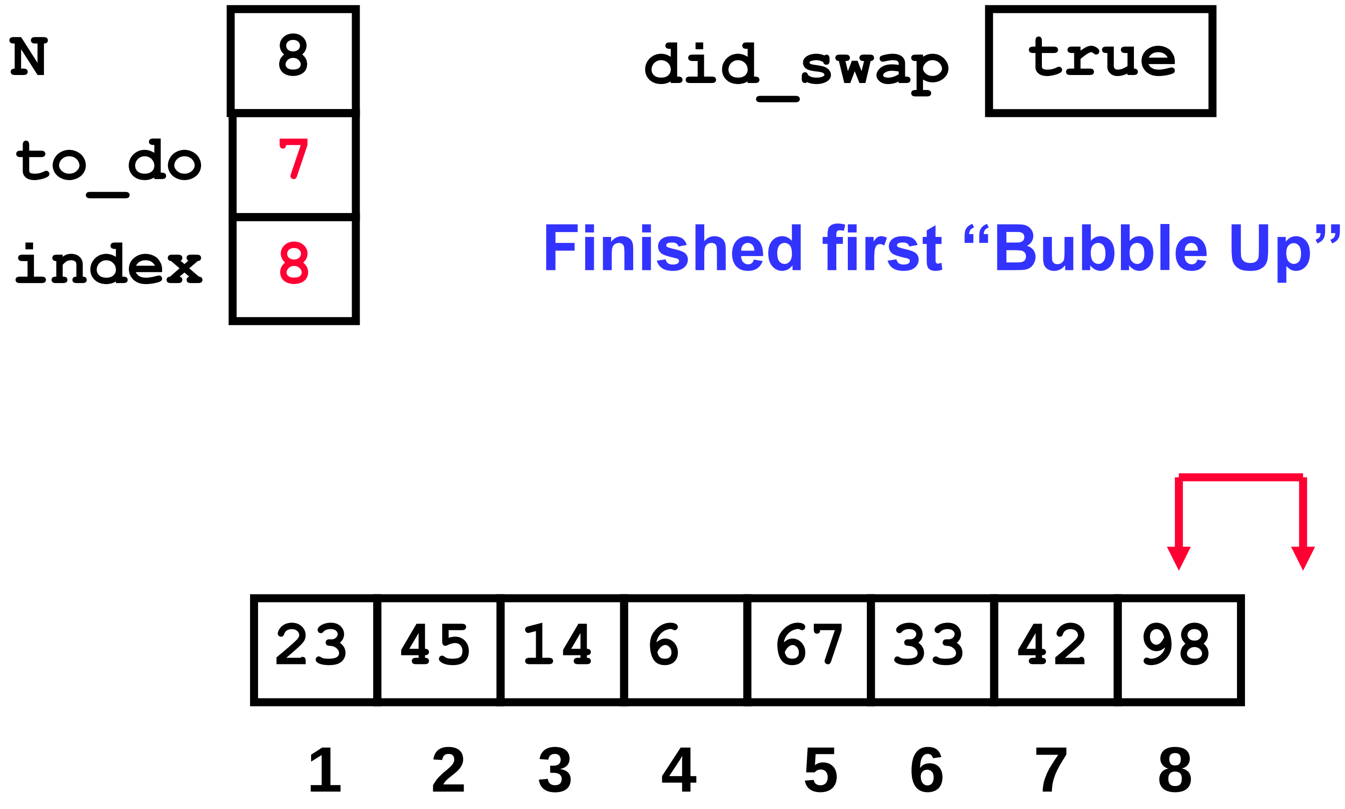
An Animated Example



An Animated Example



After First Pass of Outer Loop

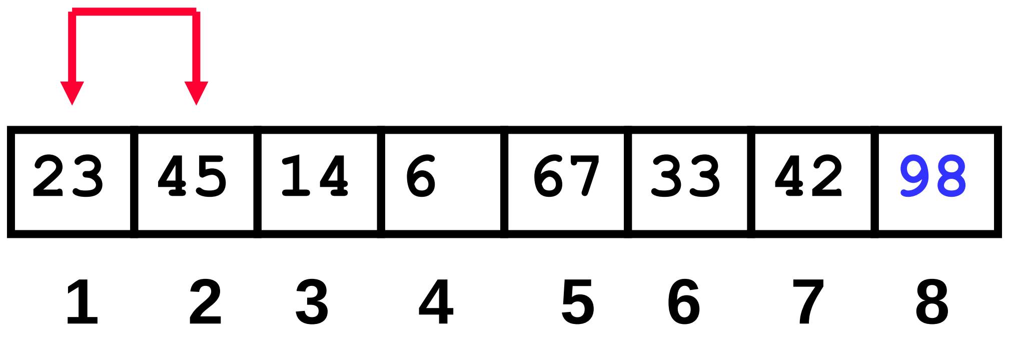


Finished first “Bubble Up”

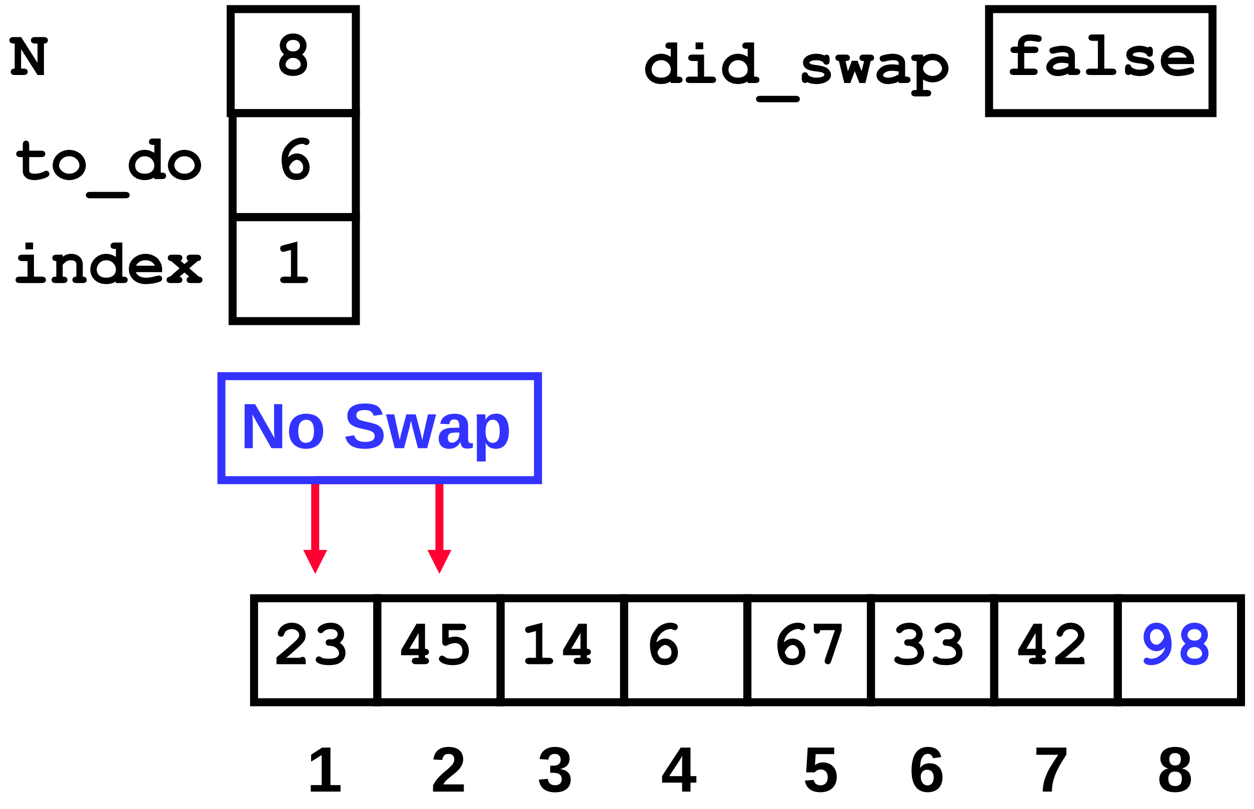
The Second "Bubble Up"

N	8
to_do	6
index	1

did_swap false



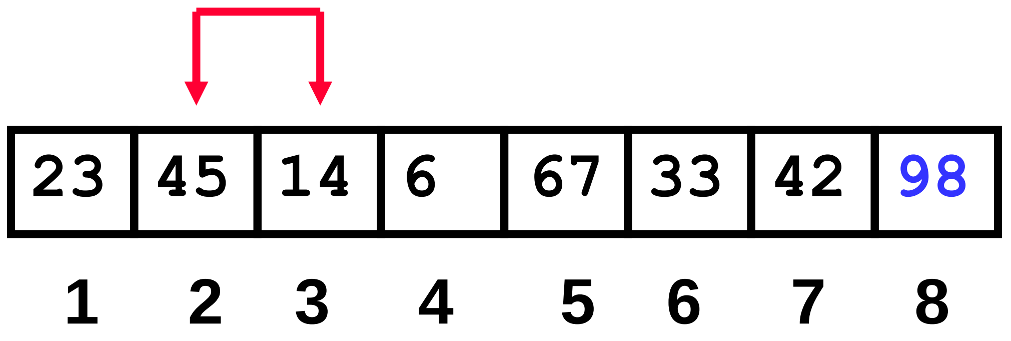
The Second "Bubble Up"



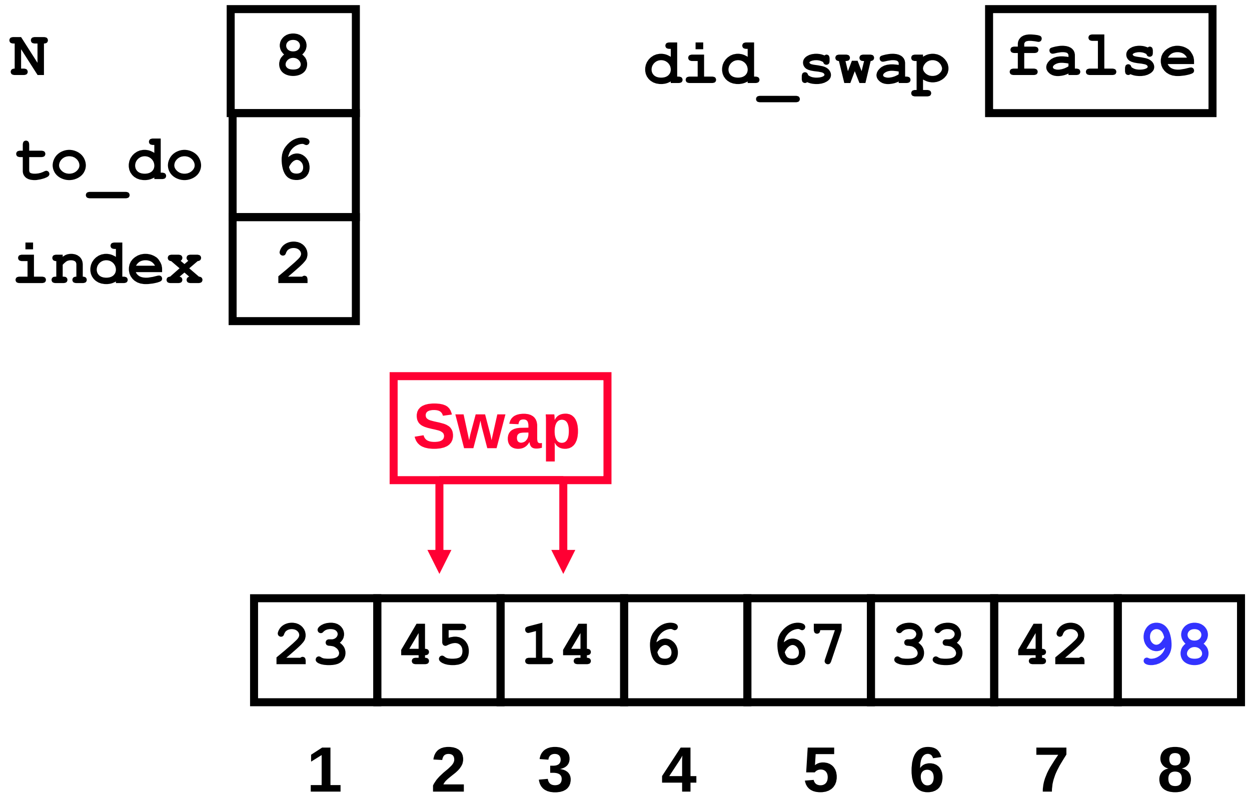
The Second “Bubble Up”

N	8
to_do	6
index	2

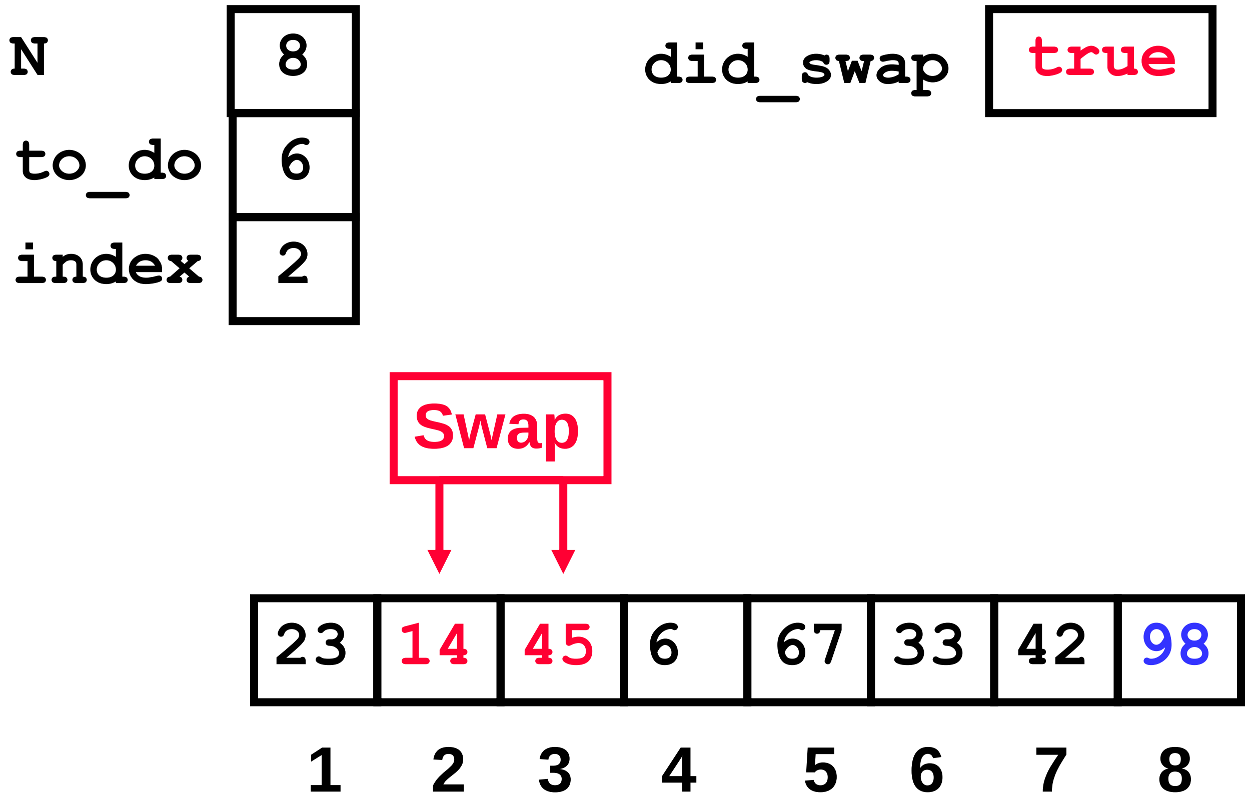
did_swap	false



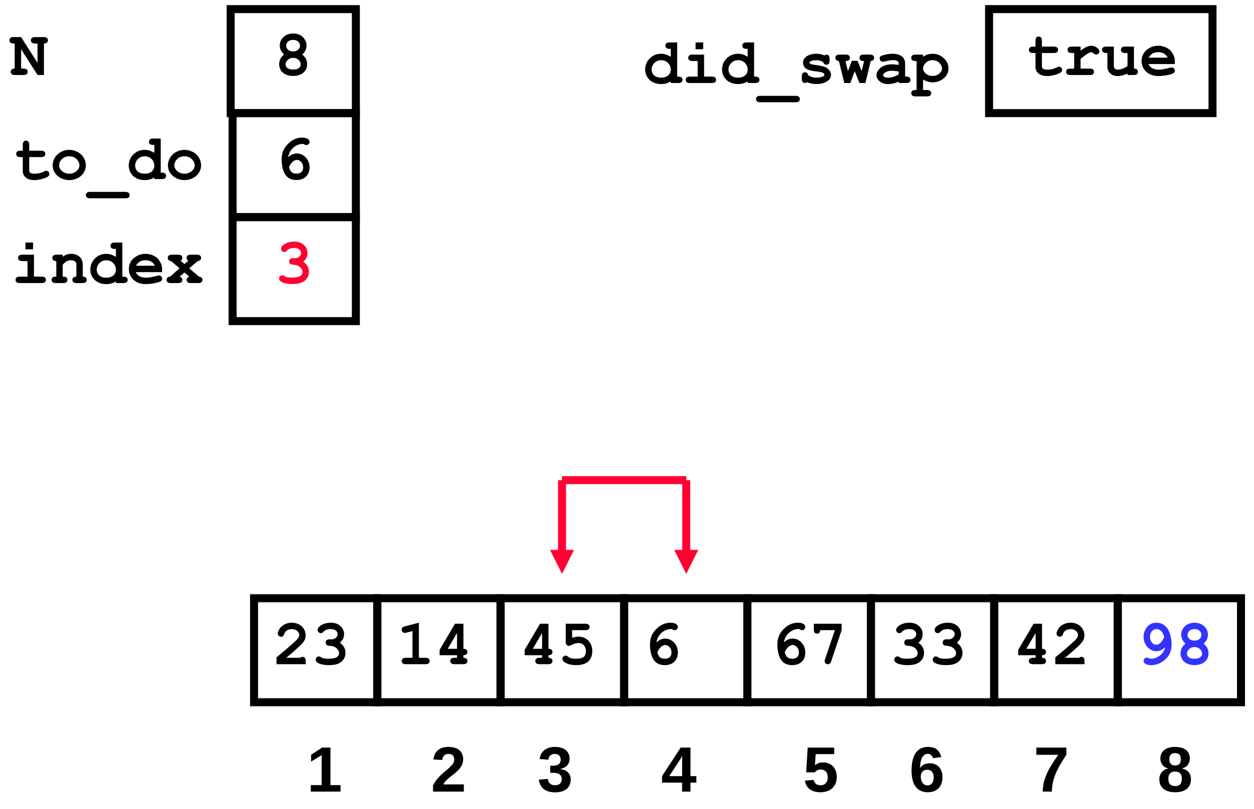
The Second "Bubble Up"



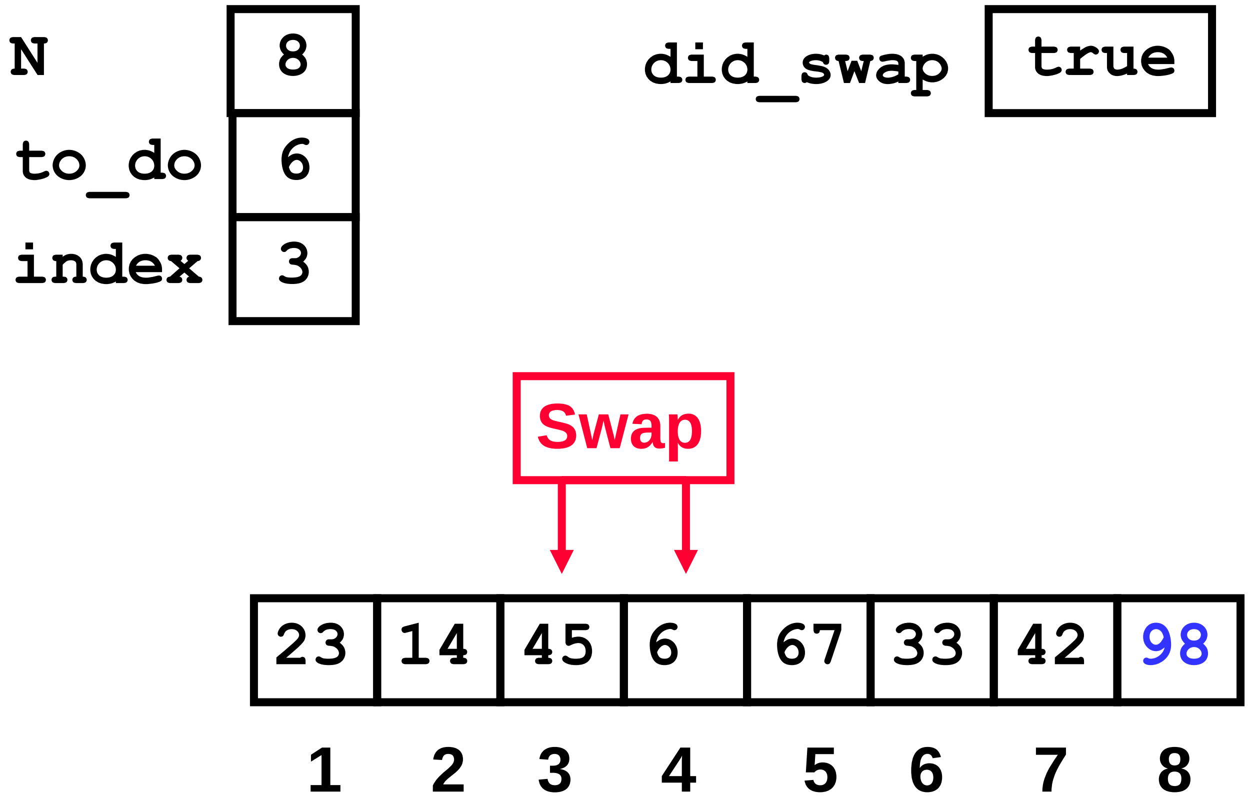
The Second "Bubble Up"



The Second "Bubble Up"



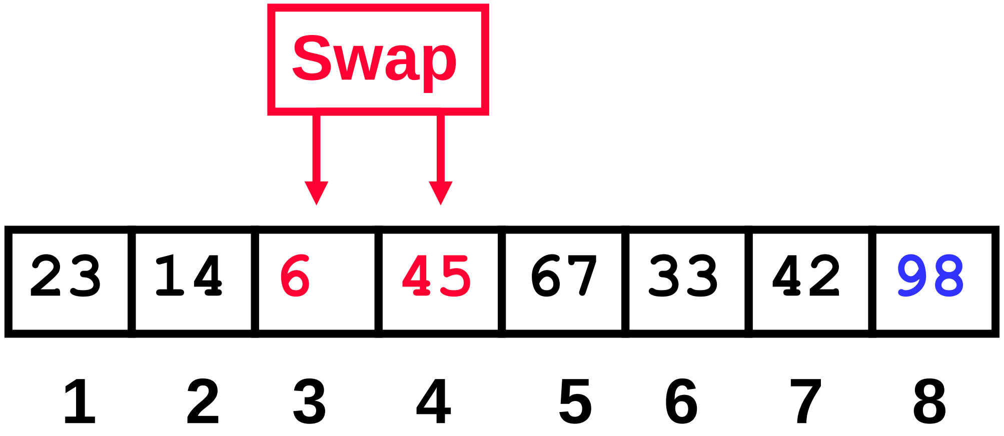
The Second "Bubble Up"



The Second “Bubble Up”

`N`	8
`to_do`	6
`index`	3

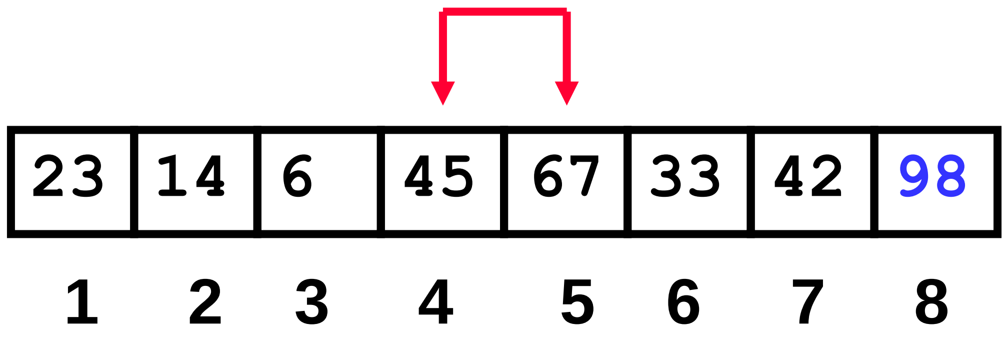
`did_swap` `true`



The Second "Bubble Up"

N	8
to_do	6
index	4

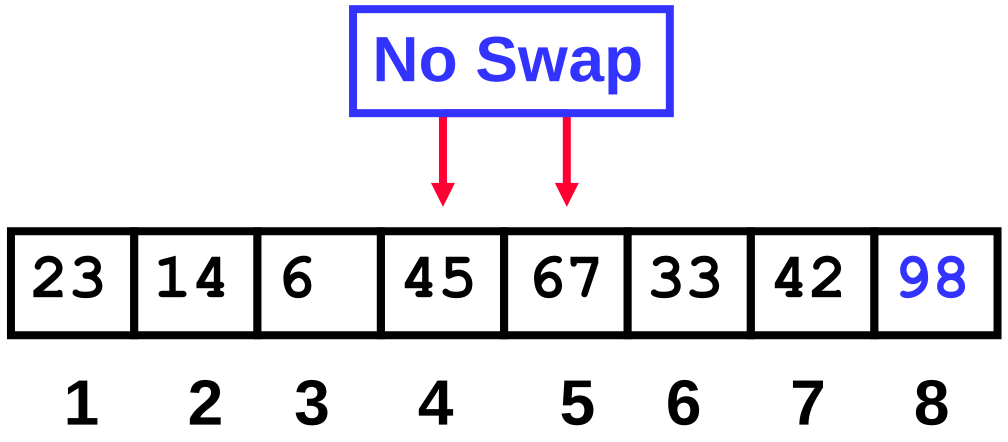
did_swap	true



The Second "Bubble Up"

N	8
to_do	6
index	4

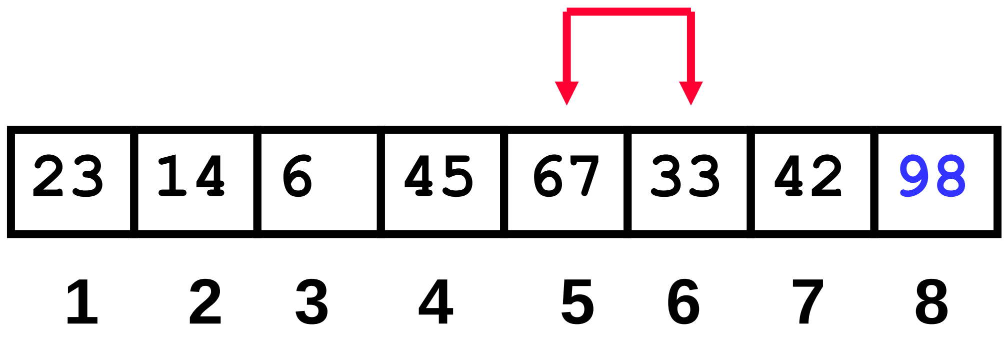
did_swap	true



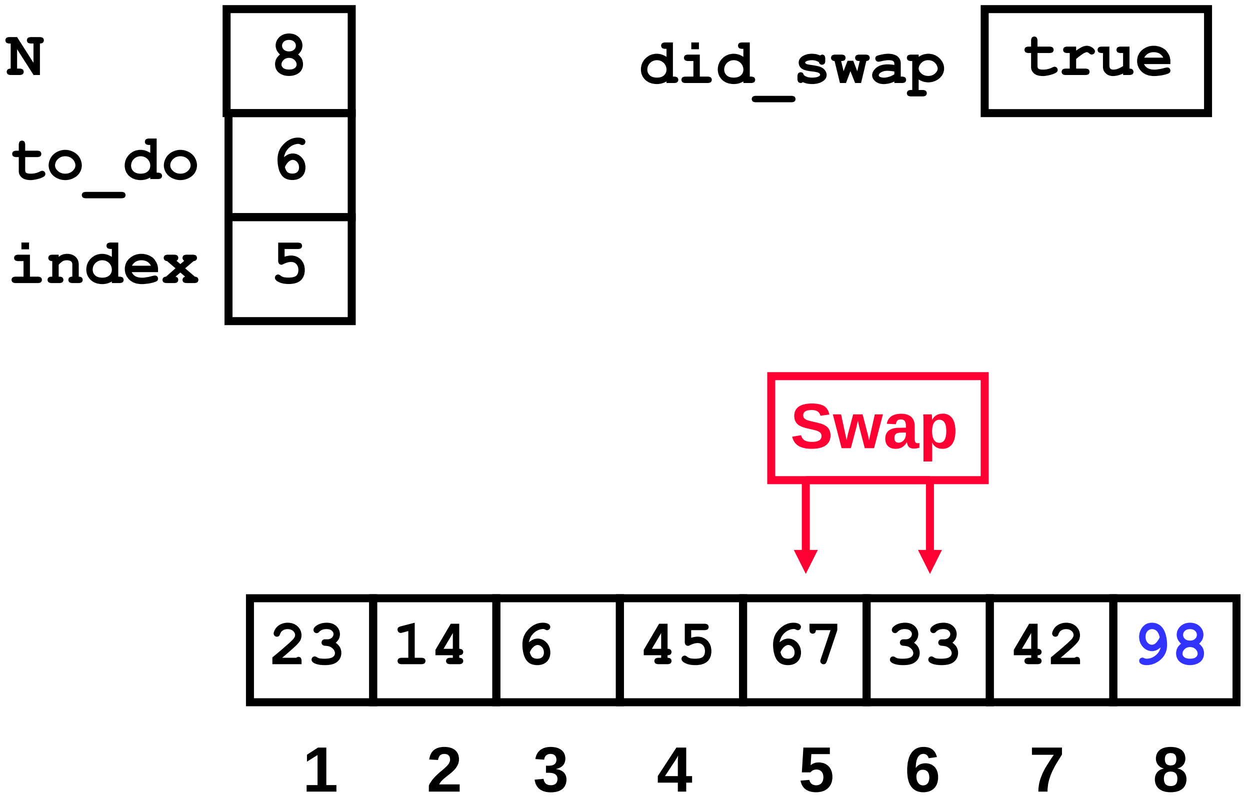
The Second "Bubble Up"

N	8
to_do	6
index	5

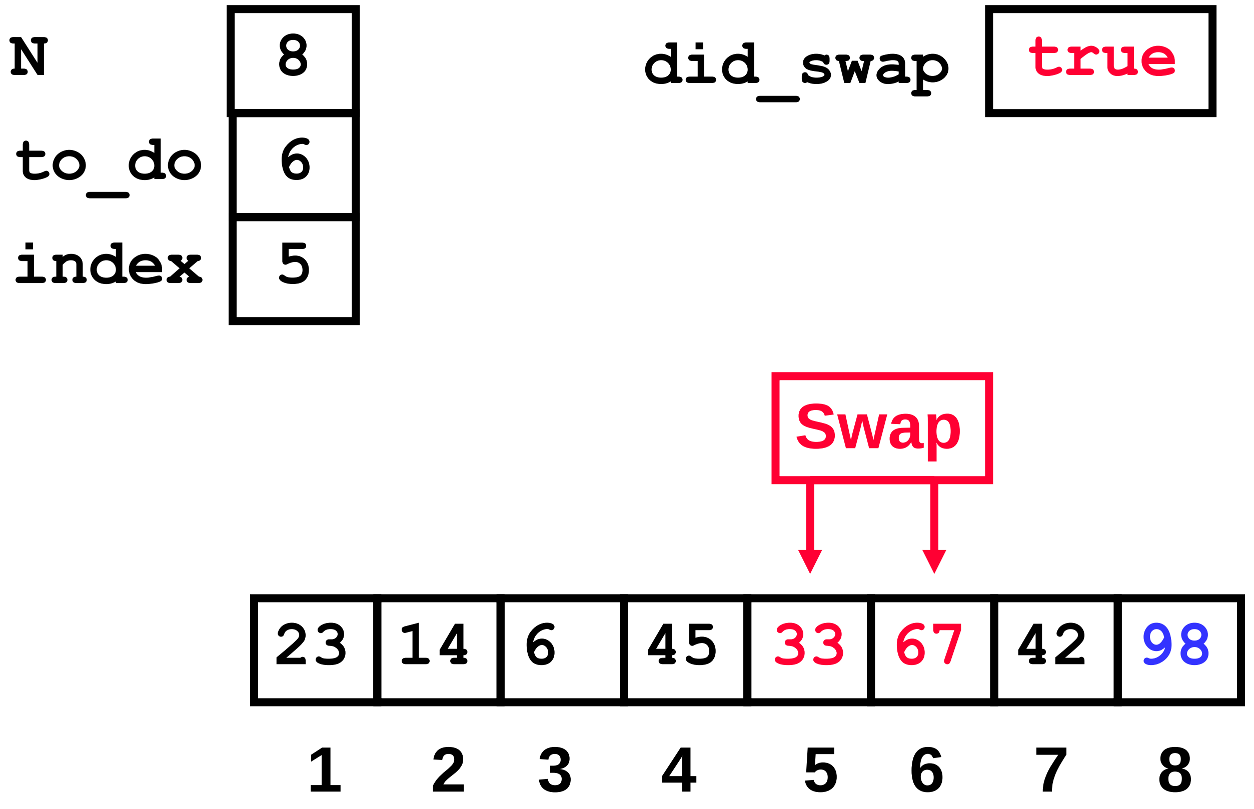
did_swap	true



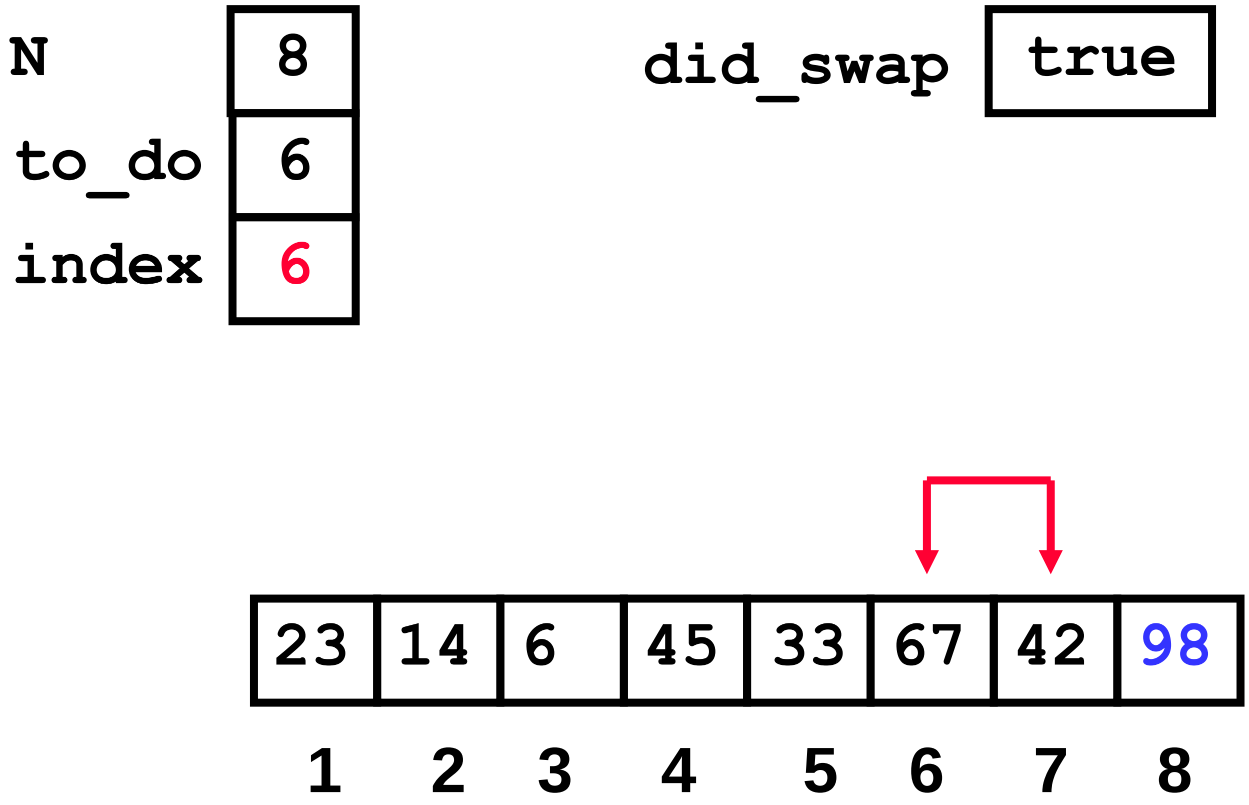
The Second "Bubble Up"



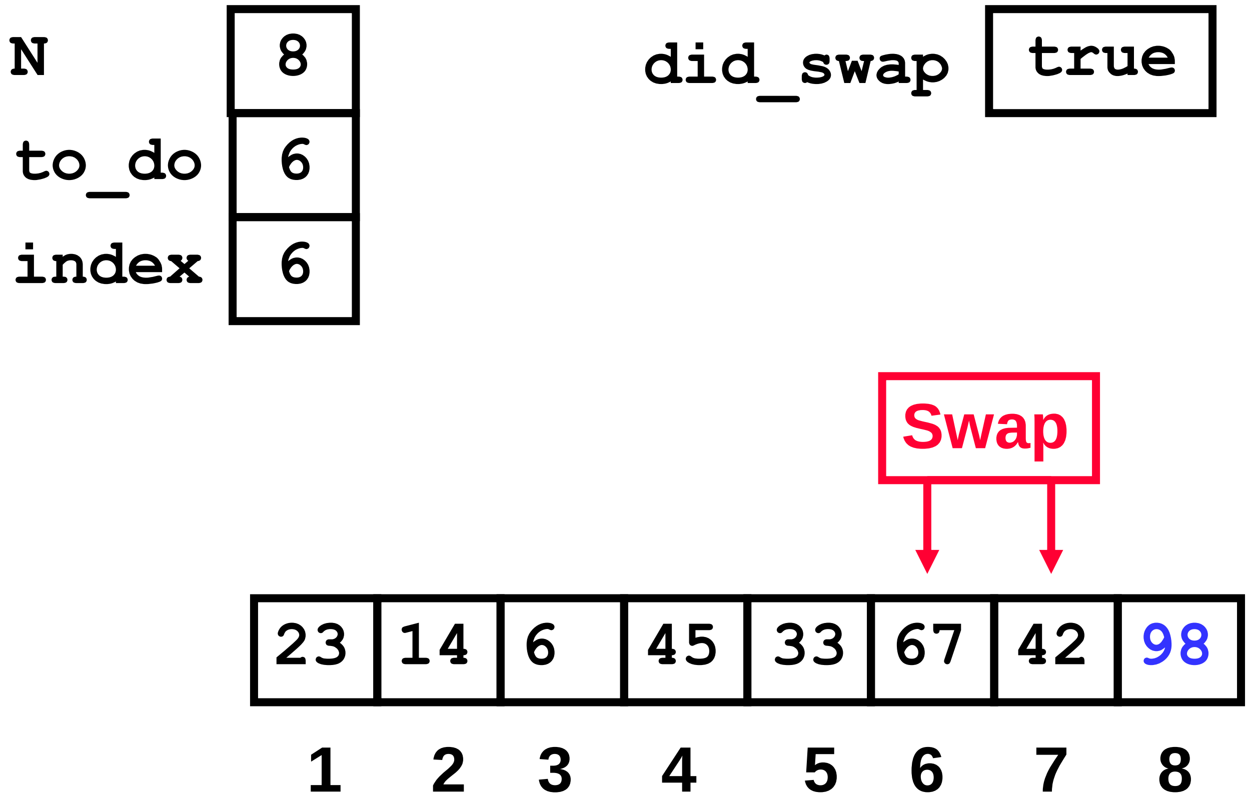
The Second "Bubble Up"



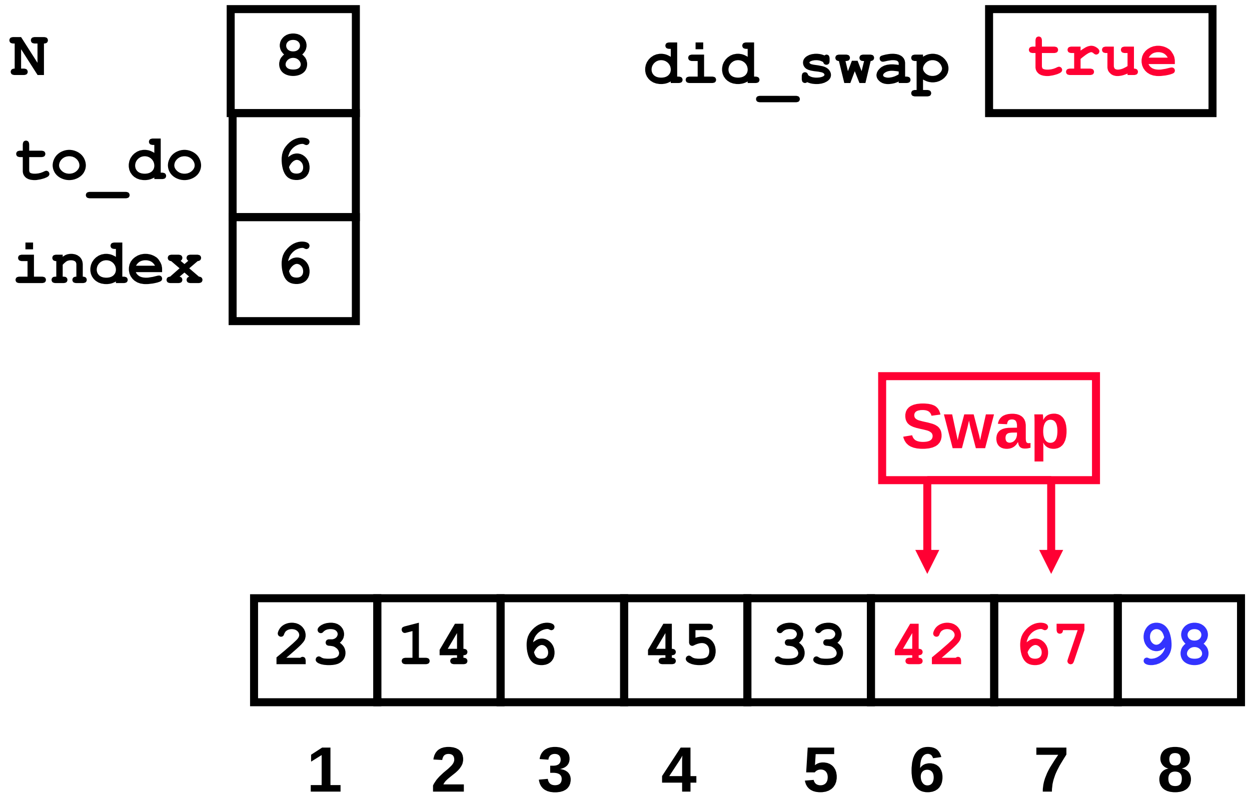
The Second "Bubble Up"



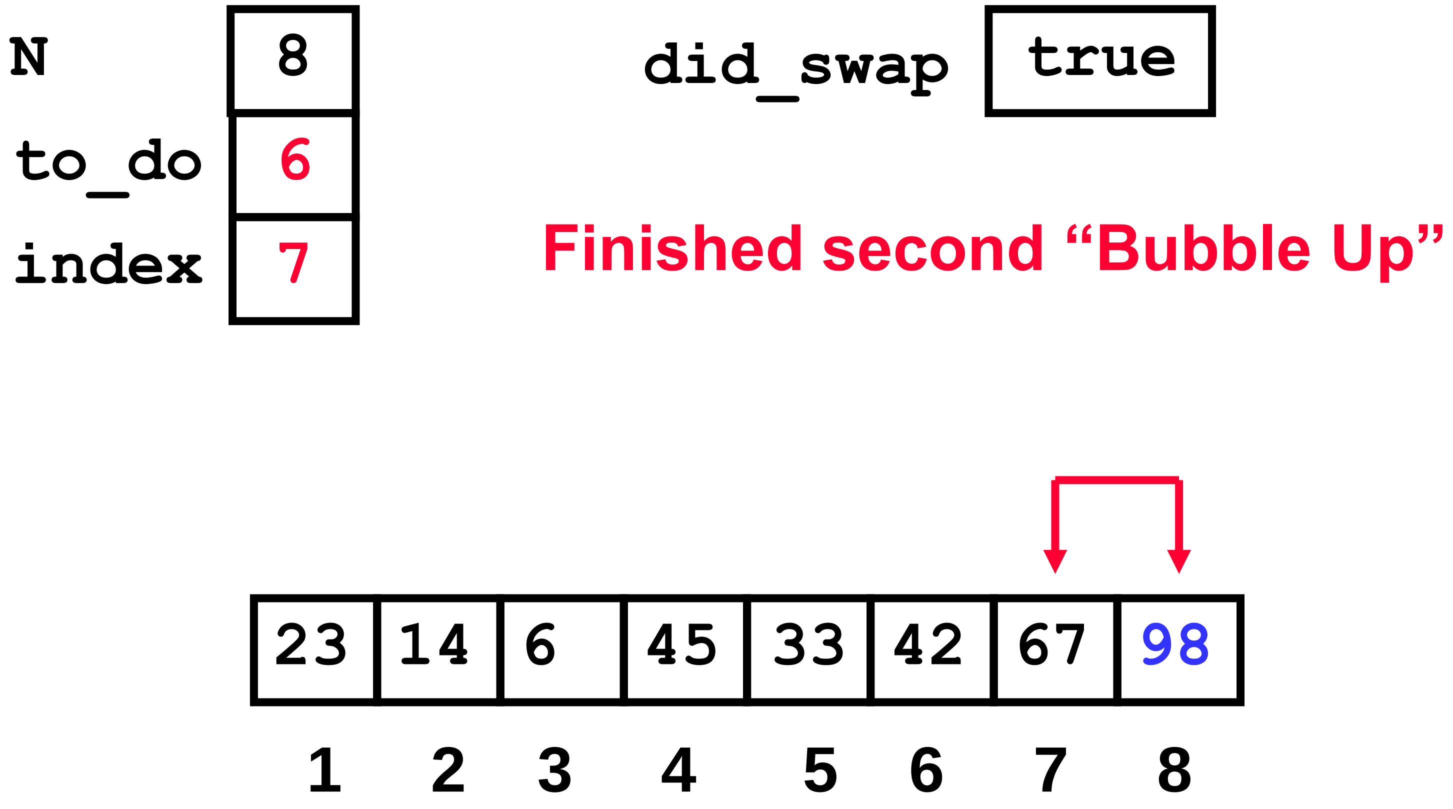
The Second "Bubble Up"



The Second "Bubble Up"



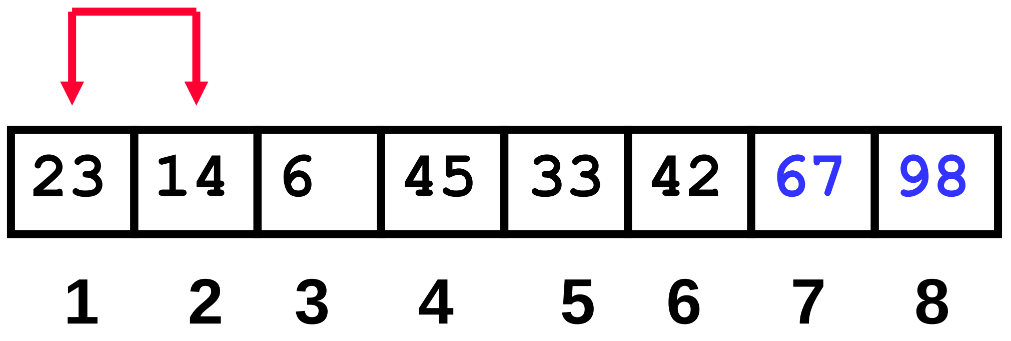
After Second Pass of Outer Loop



The Third "Bubble Up"

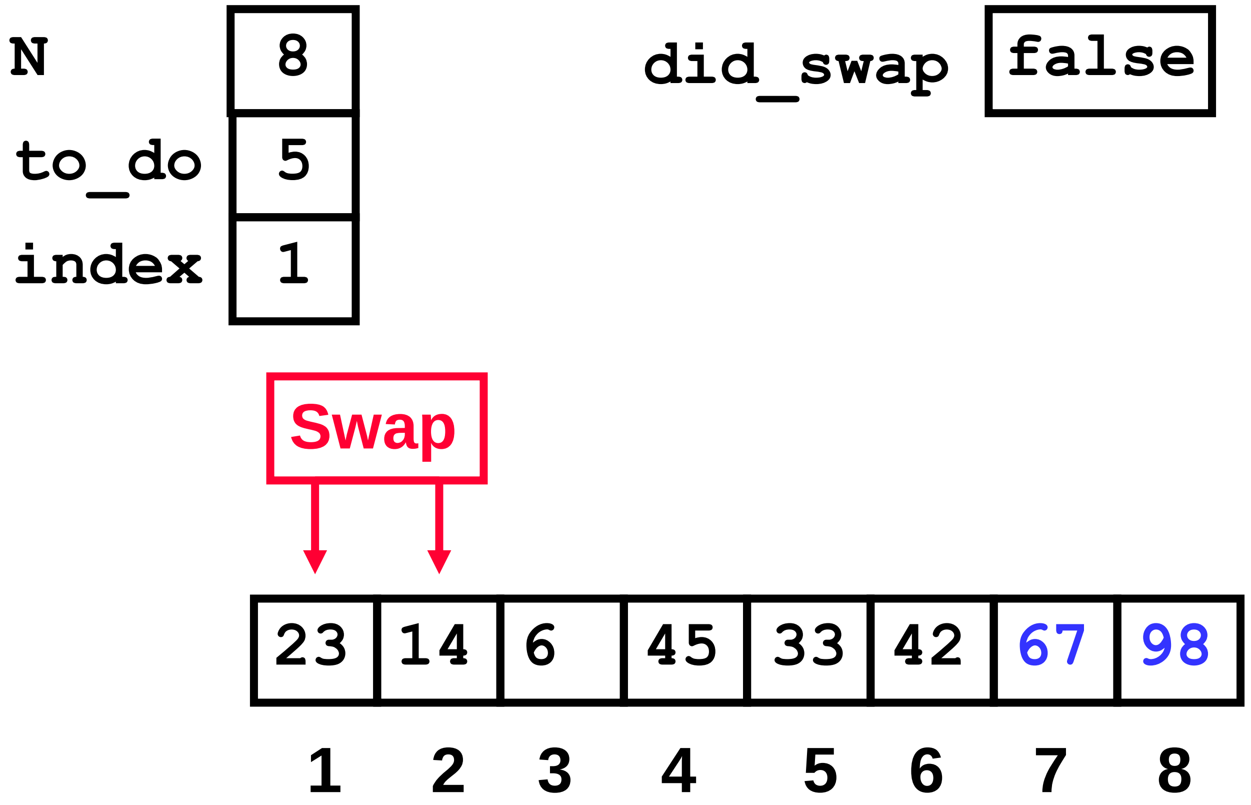
N	8
to_do	5
index	1

did_swap	false

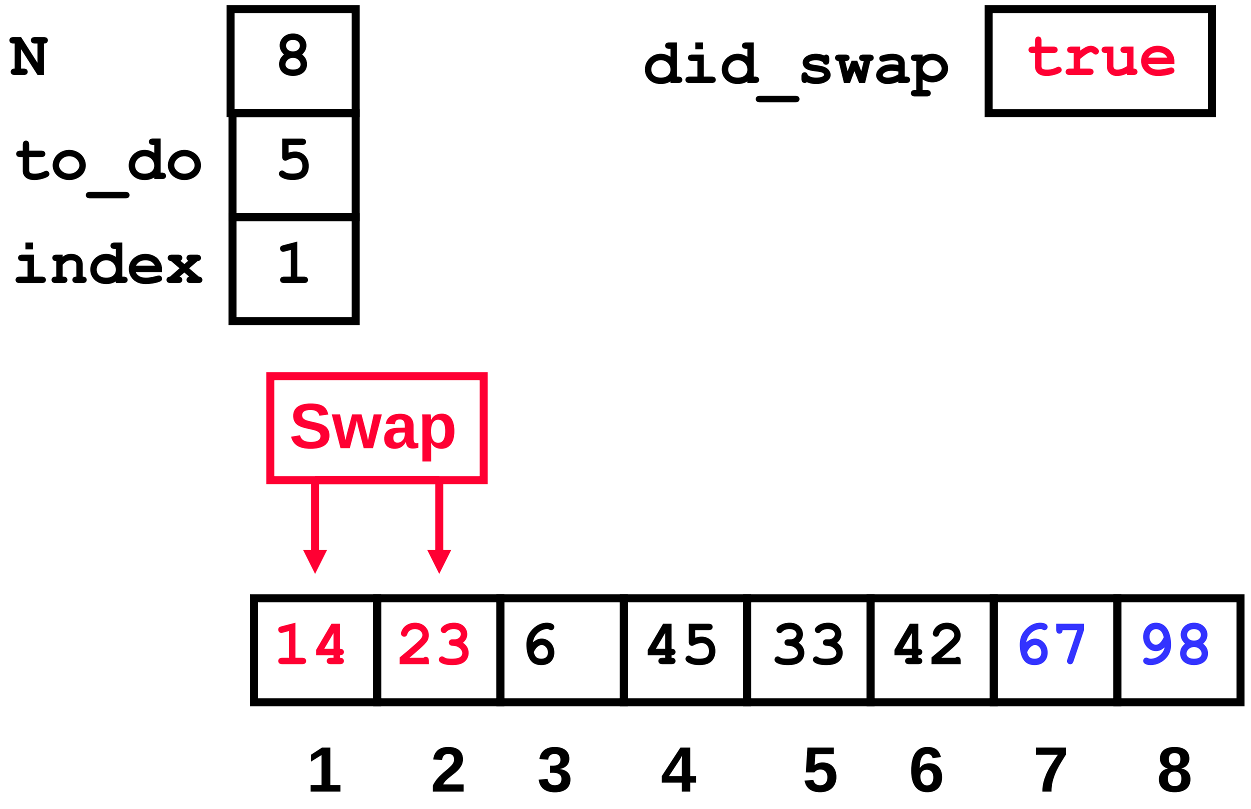


23	14	6	45	33	42	67	98
1	2	3	4	5	6	7	8

The Third “Bubble Up”



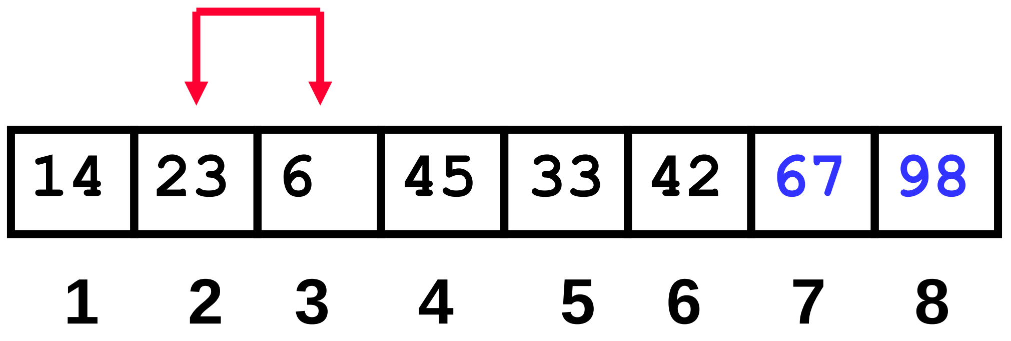
The Third "Bubble Up"



The Third "Bubble Up"

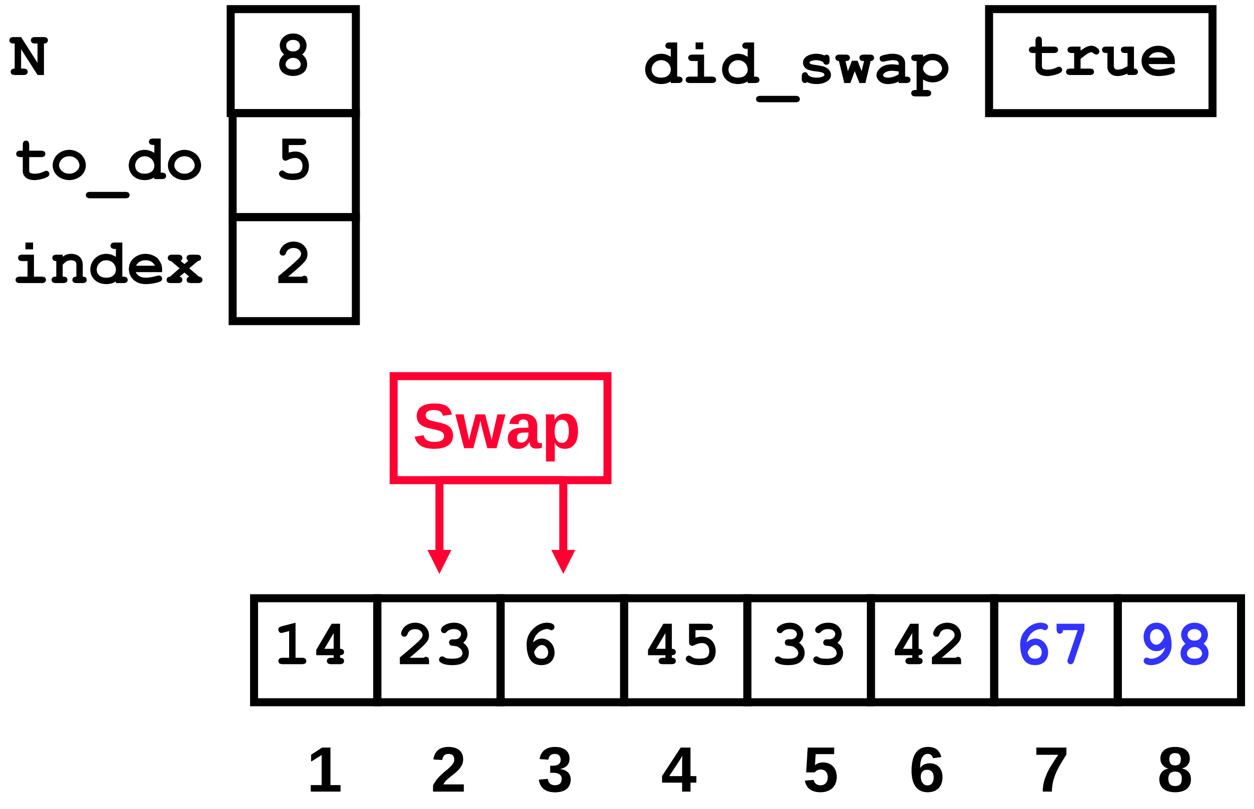
N	8
to_do	5
index	2

did_swap	true

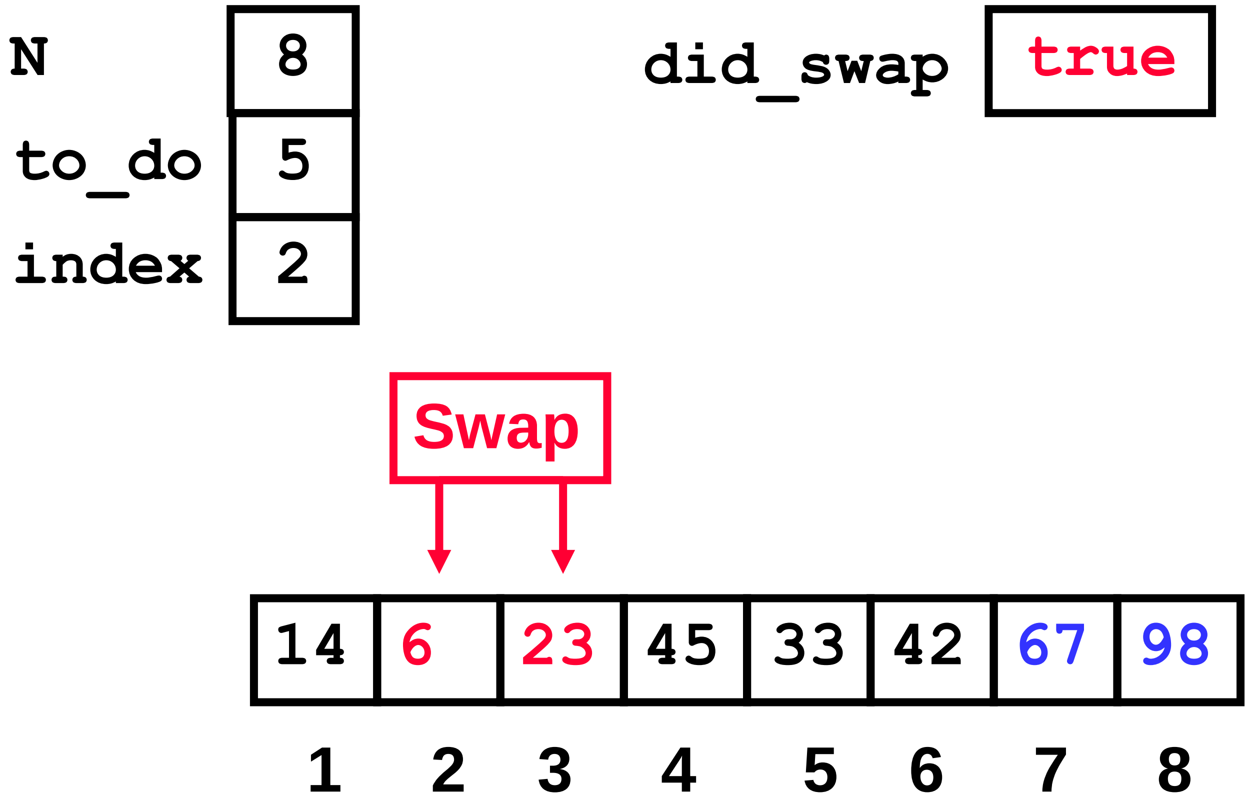


14	23	6	45	33	42	67	98
1	2	3	4	5	6	7	8

The Third "Bubble Up"



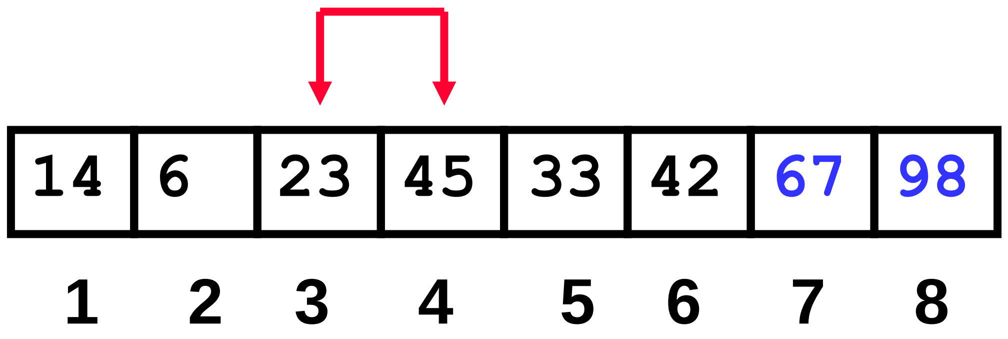
The Third "Bubble Up"



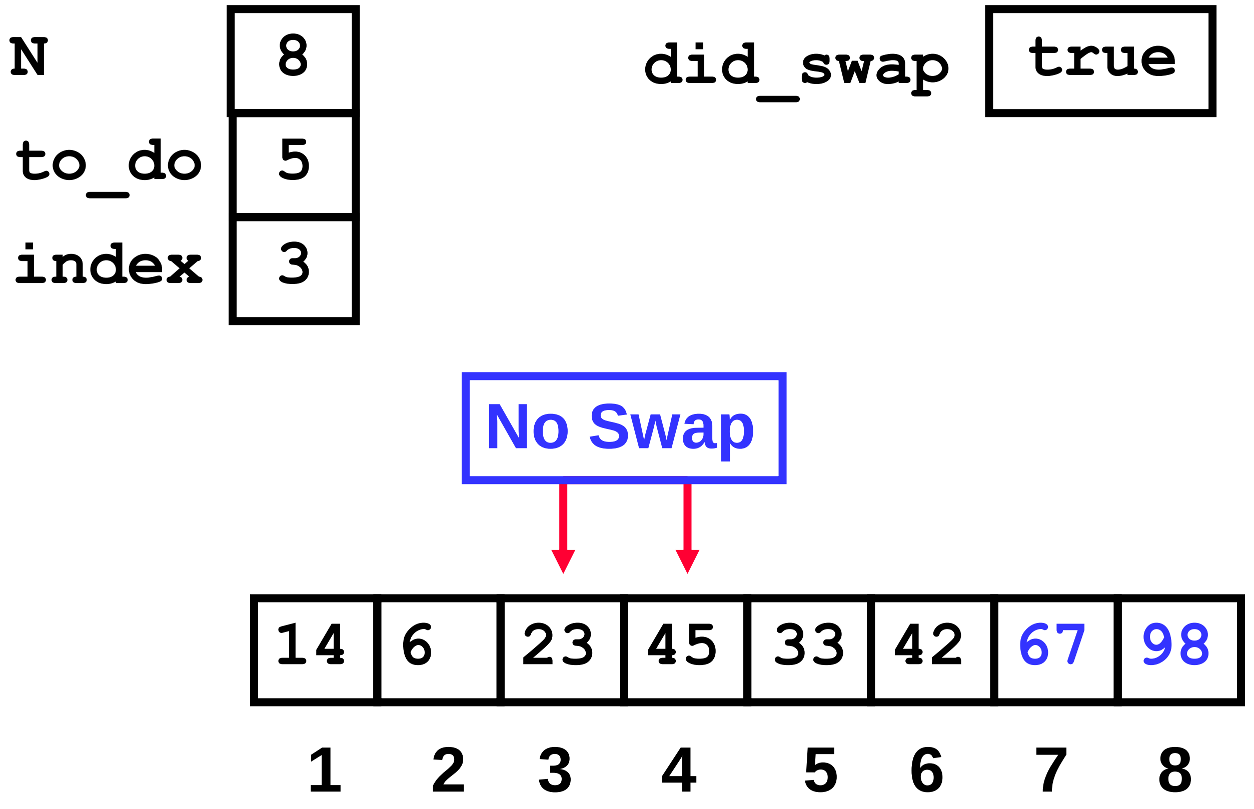
The Third "Bubble Up"

N	8
to_do	5
index	3

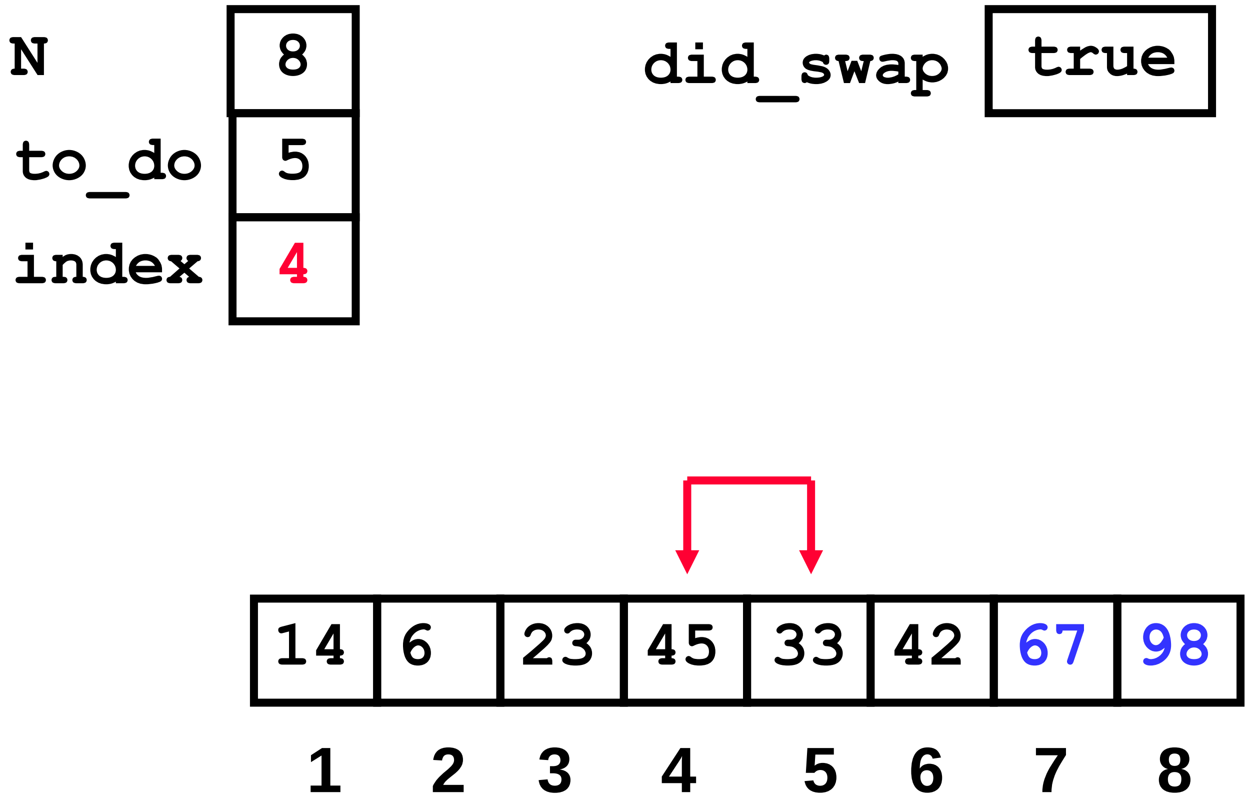
did_swap `true`



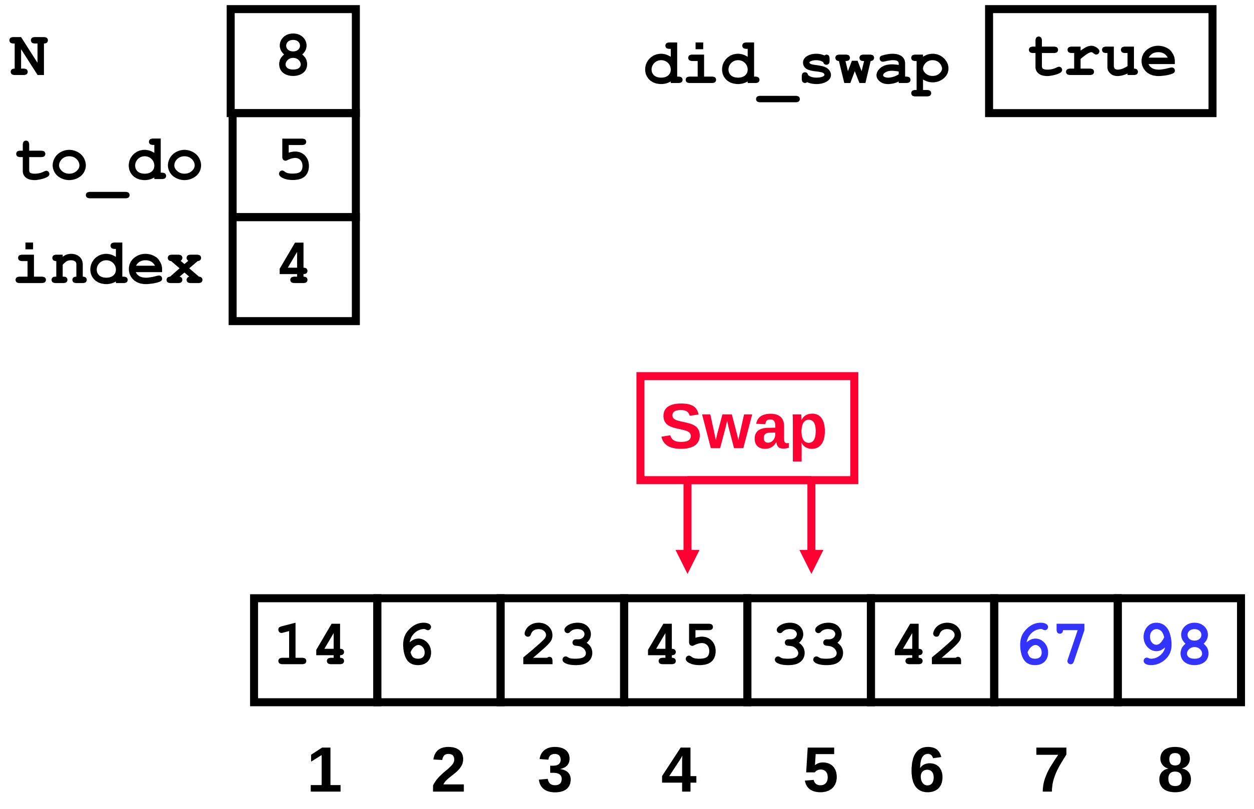
The Third “Bubble Up”



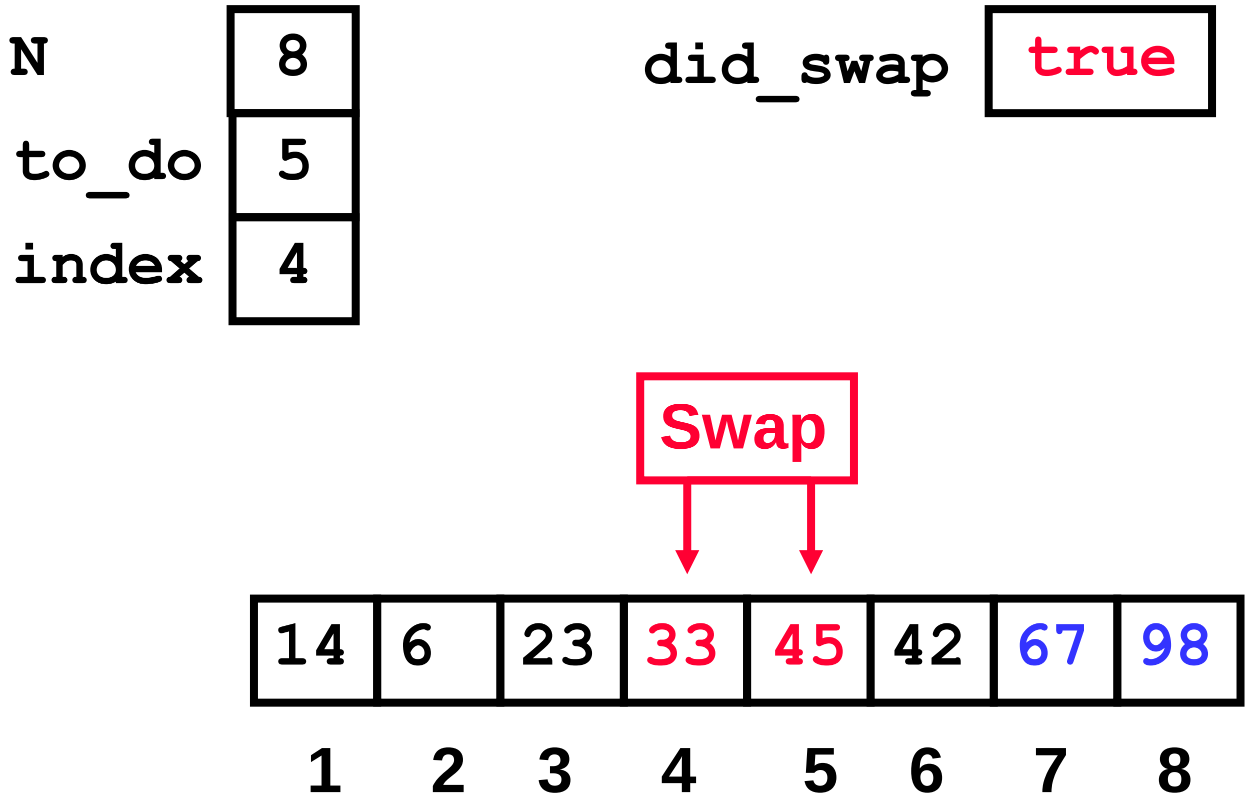
The Third "Bubble Up"



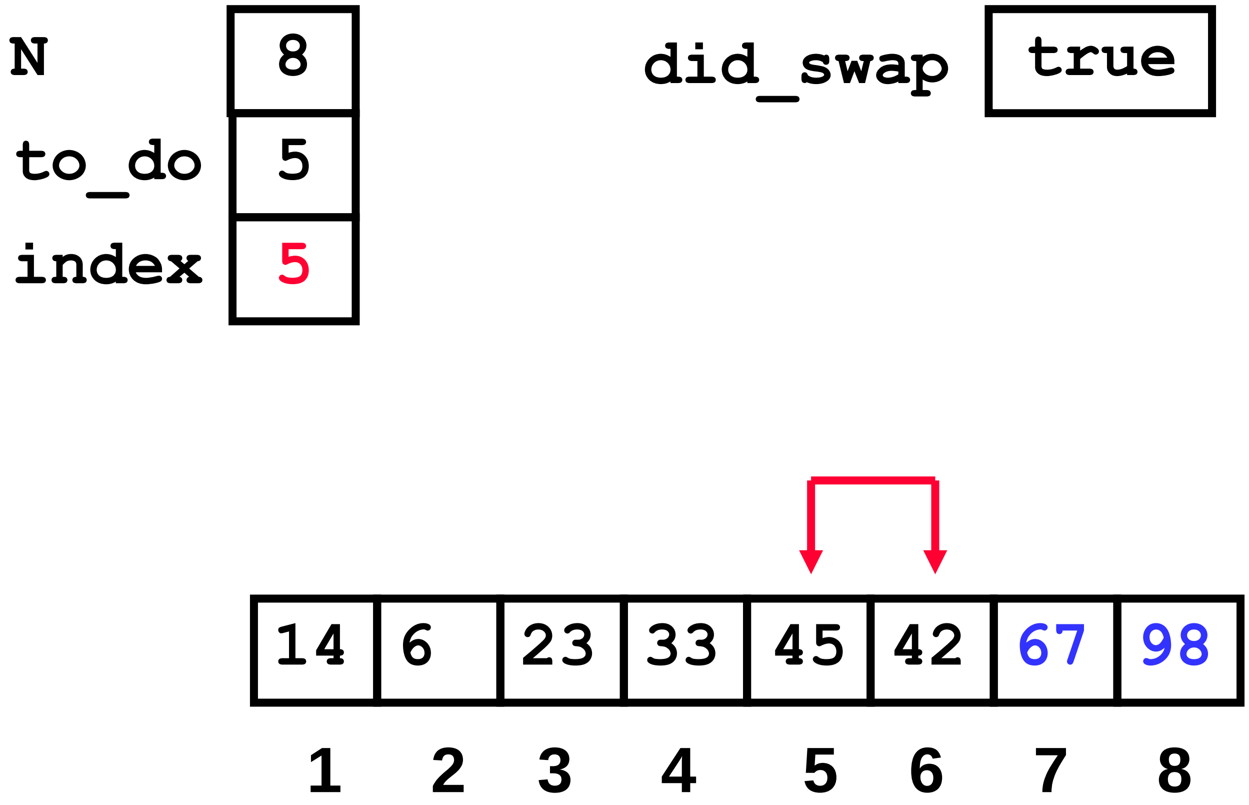
The Third "Bubble Up"



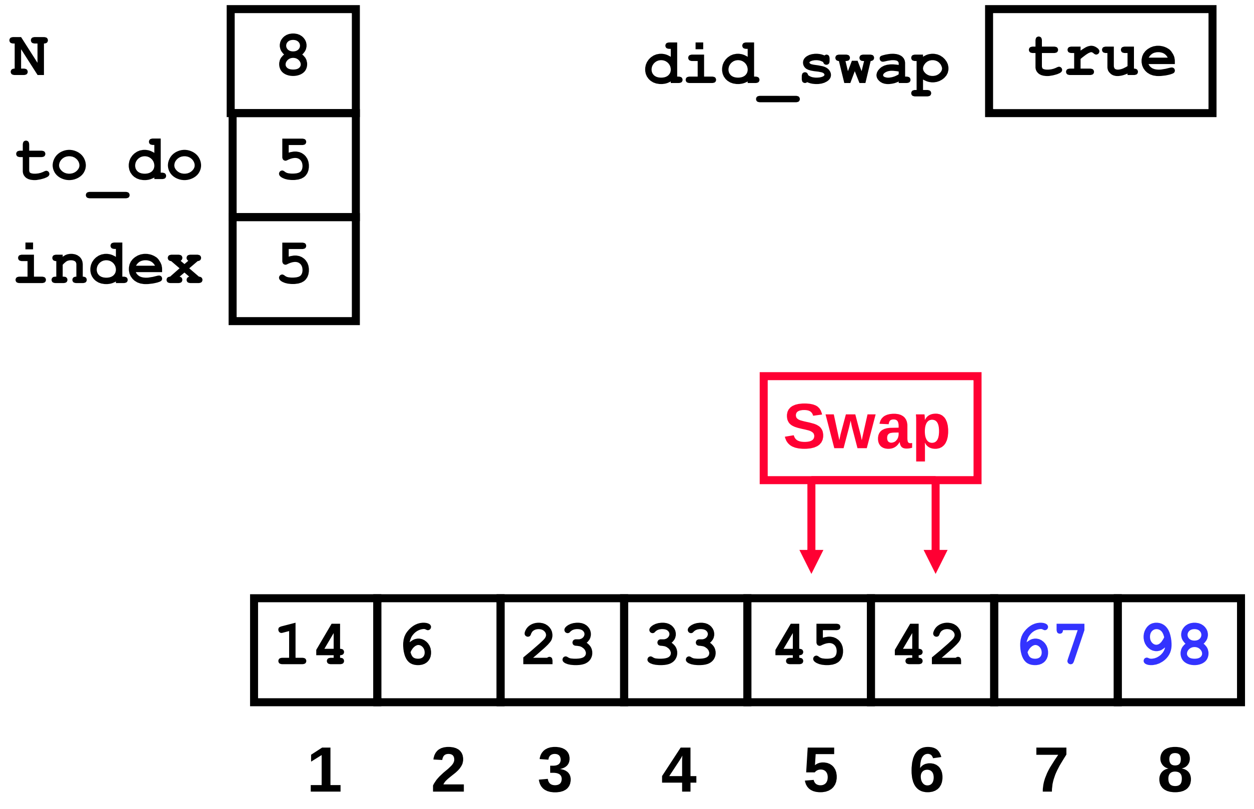
The Third “Bubble Up”



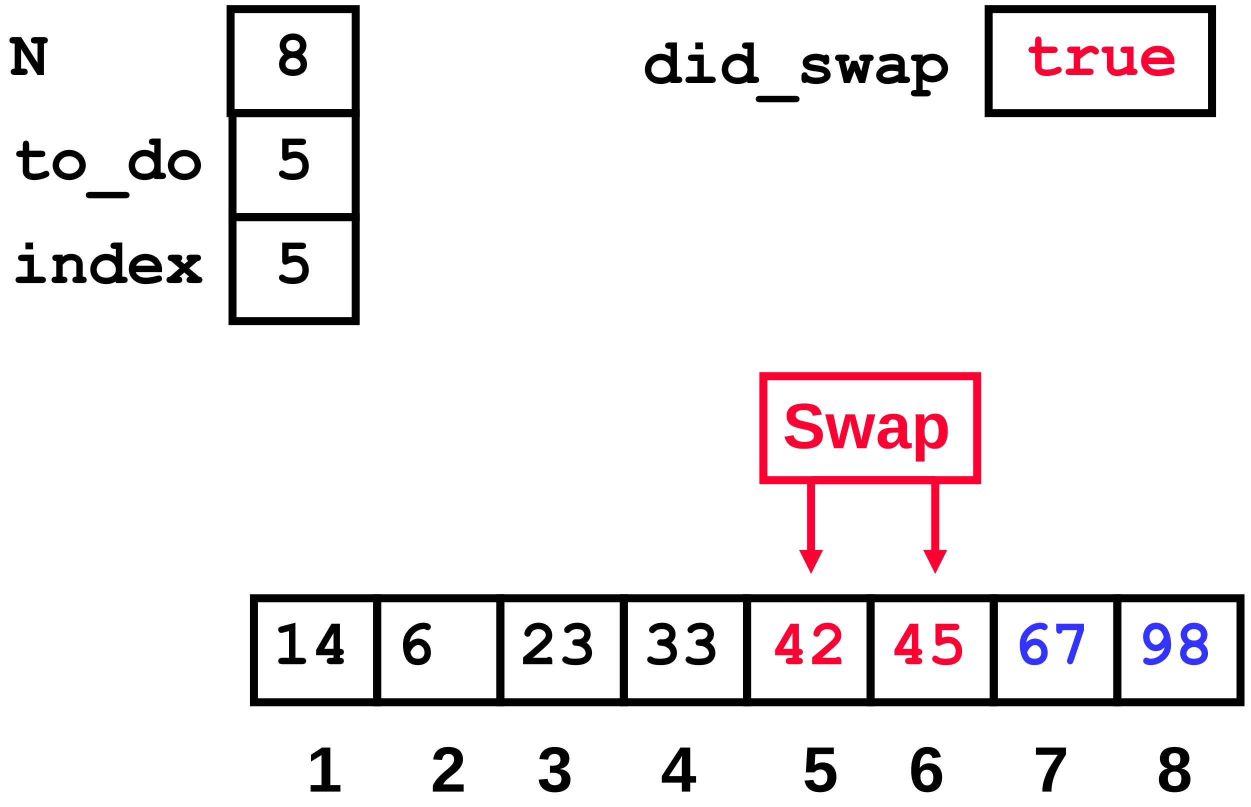
The Third "Bubble Up"



The Third "Bubble Up"



The Third “Bubble Up”

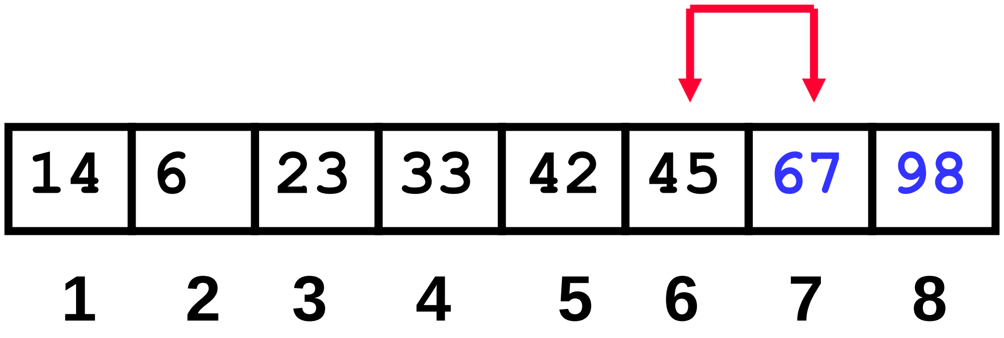


After Third Pass of Outer Loop

N	8
to_do	5
index	6

did_swap	true

Finished third “Bubble Up”

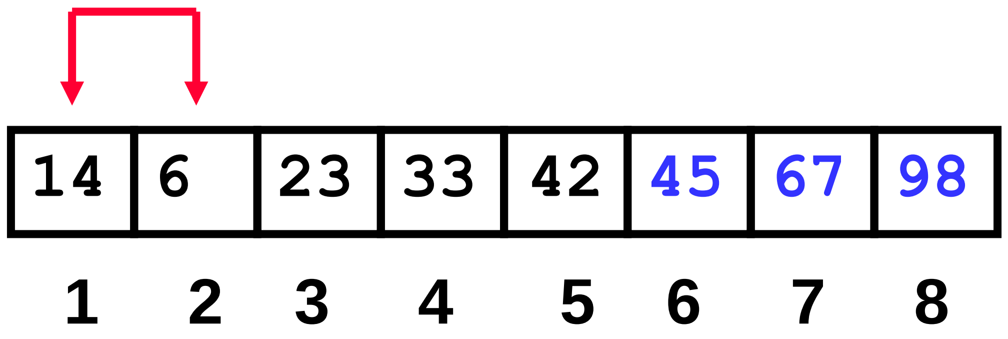


14	6	23	33	42	45	67	98
1	2	3	4	5	6	7	8

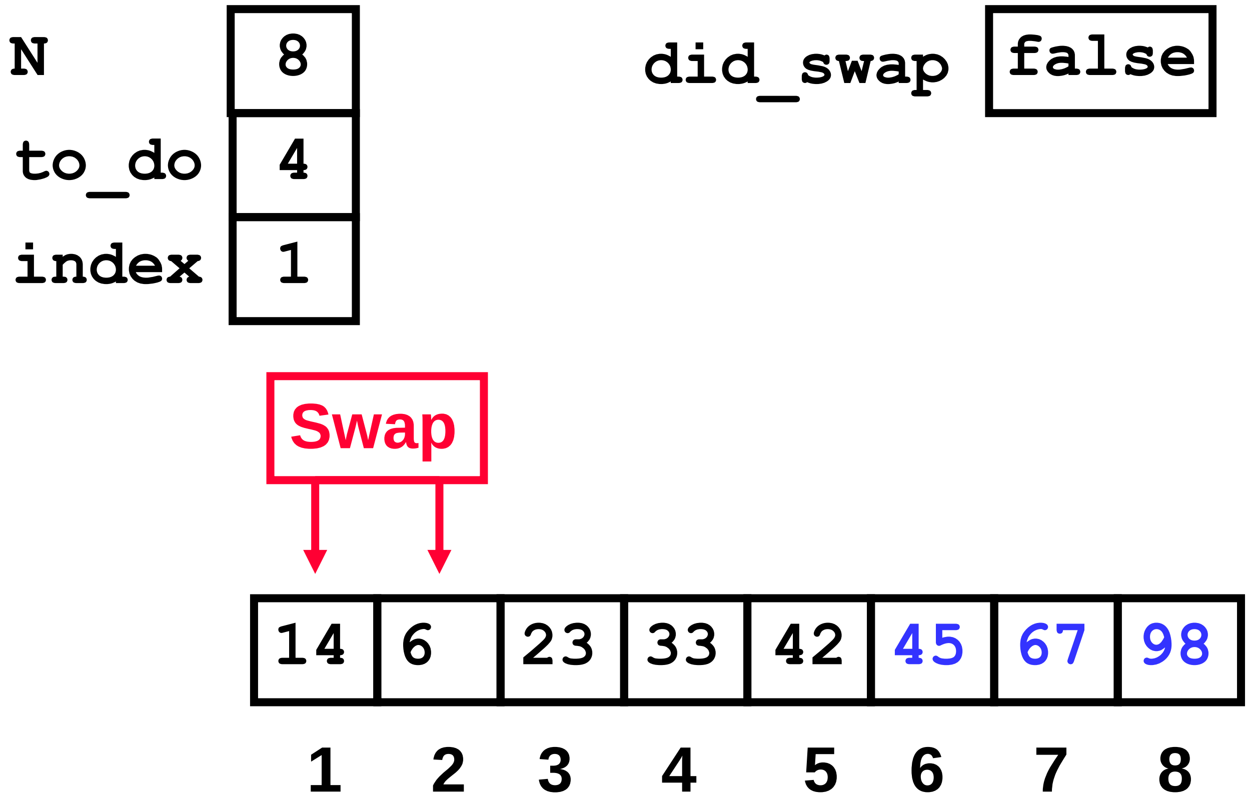
The Fourth “Bubble Up”

N	8
to_do	4
index	1

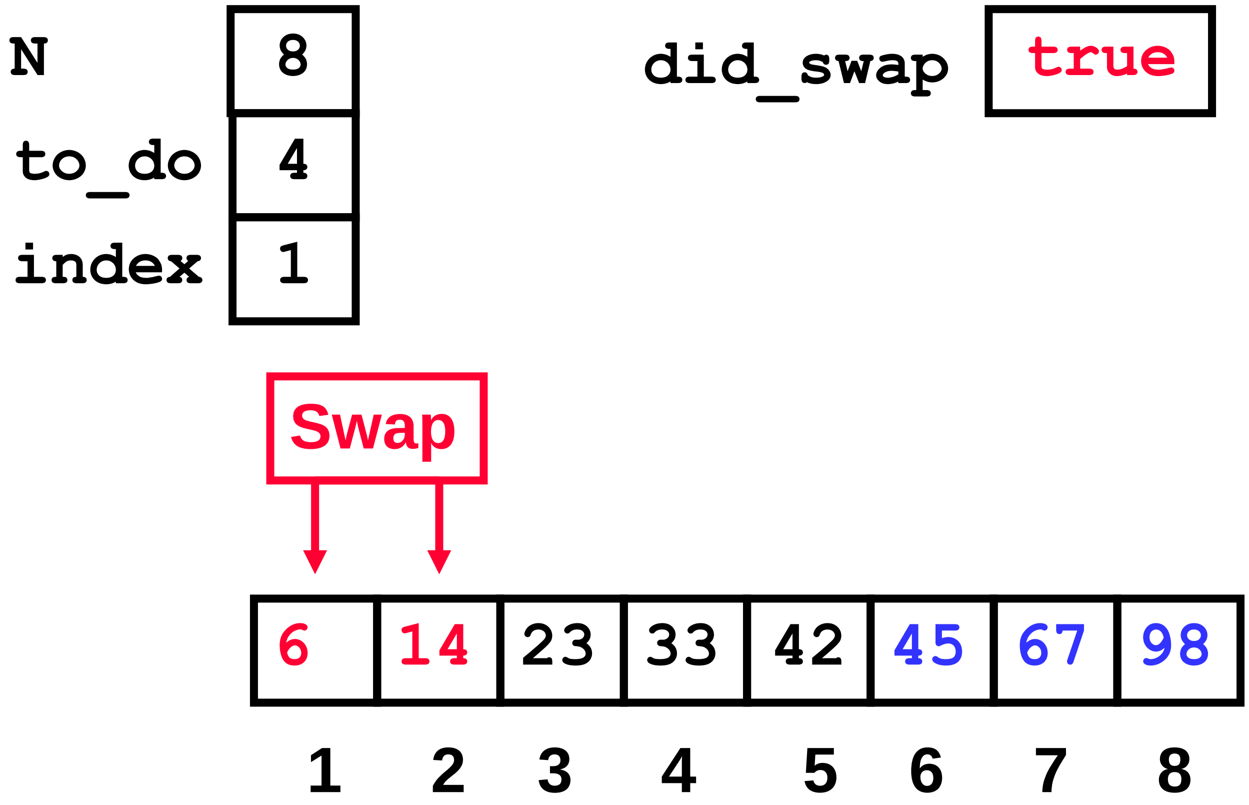
did_swap	false



The Fourth "Bubble Up"



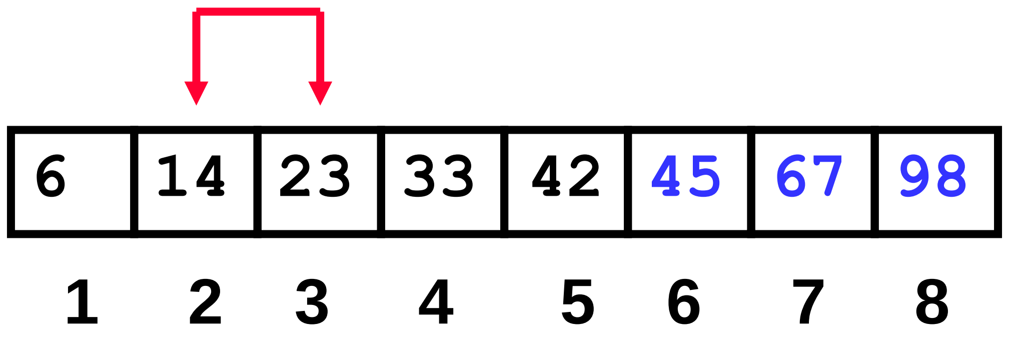
The Fourth "Bubble Up"



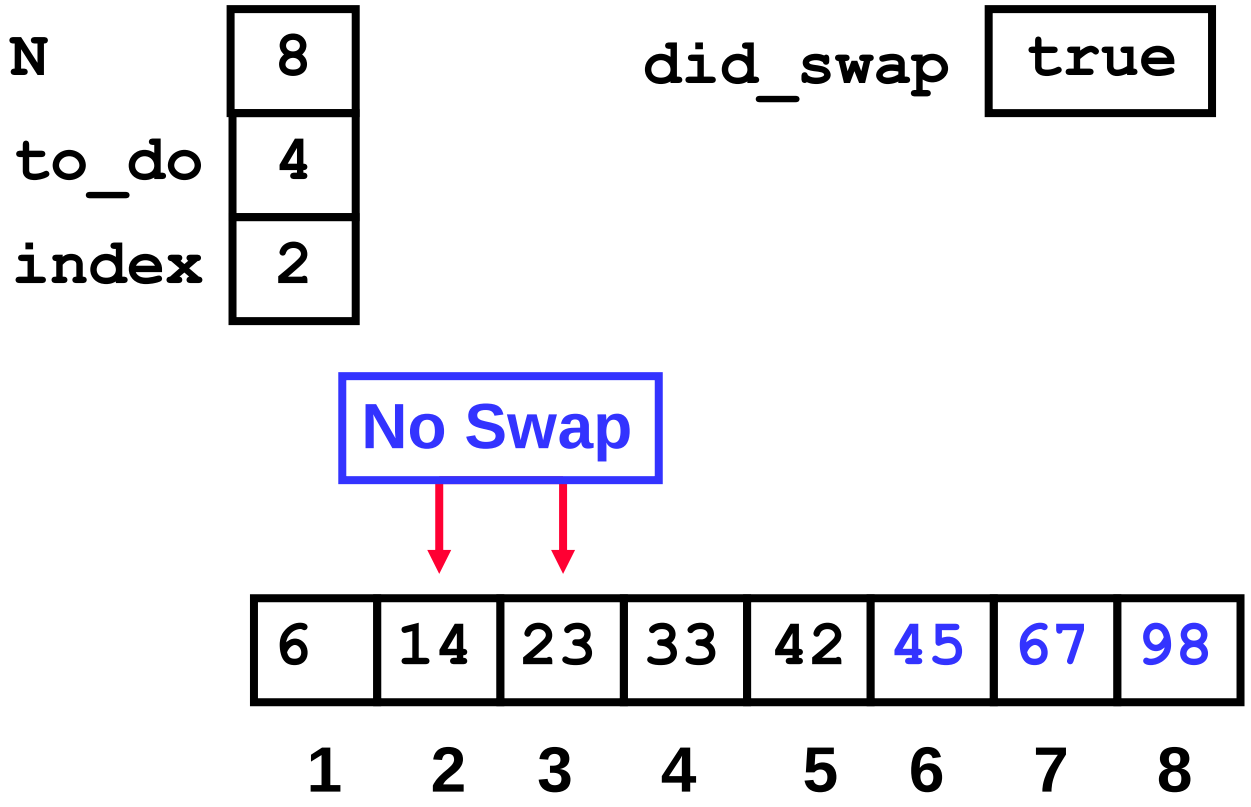
The Fourth "Bubble Up"

N	8
to_do	4
index	2

did_swap	true



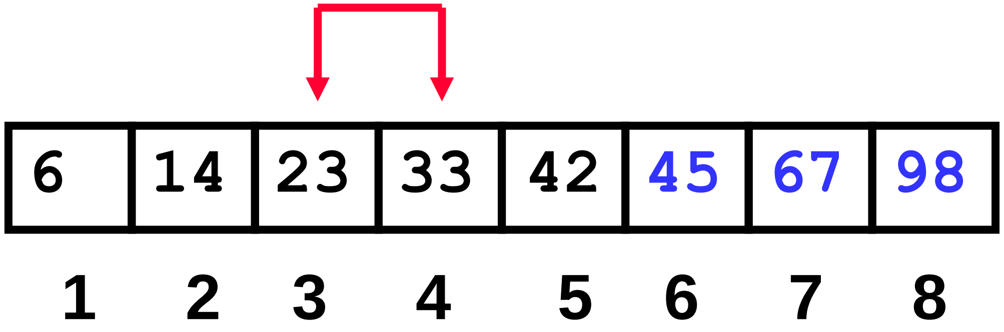
The Fourth "Bubble Up"



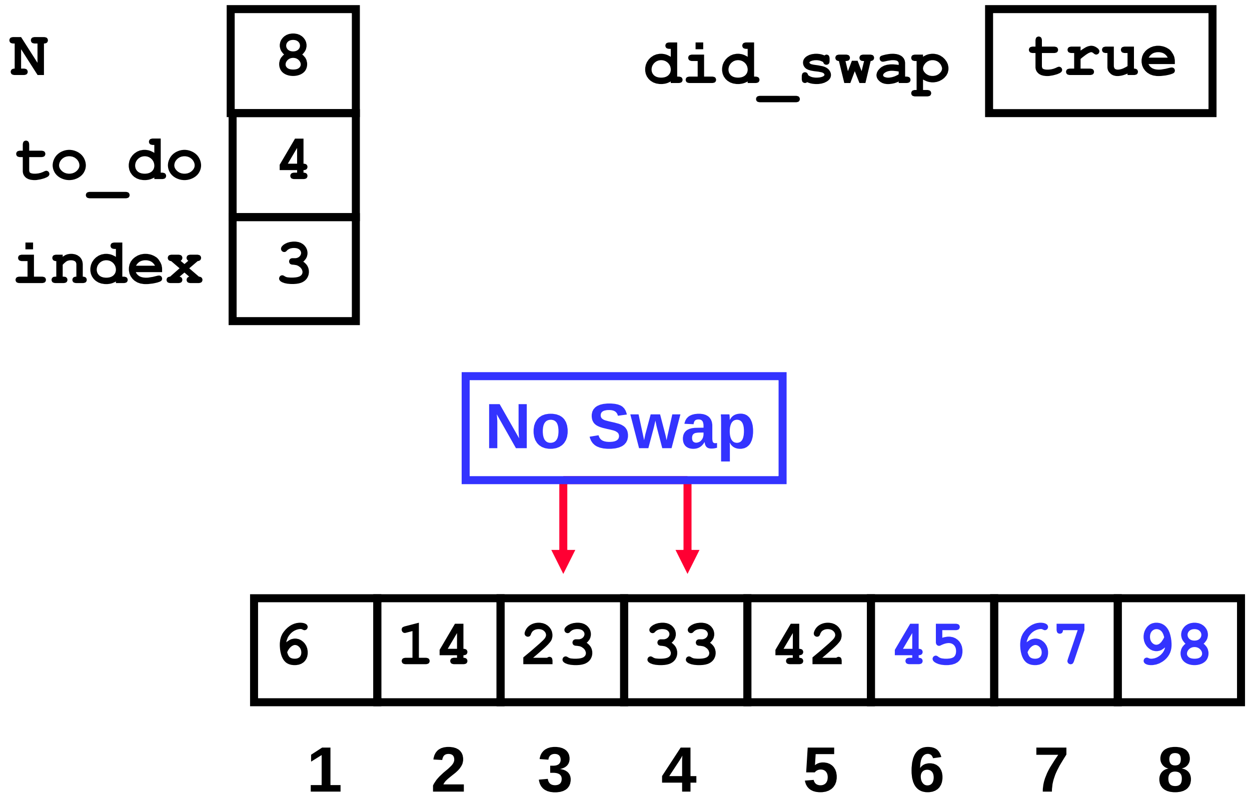
The Fourth “Bubble Up”

N	8
to_do	4
index	3

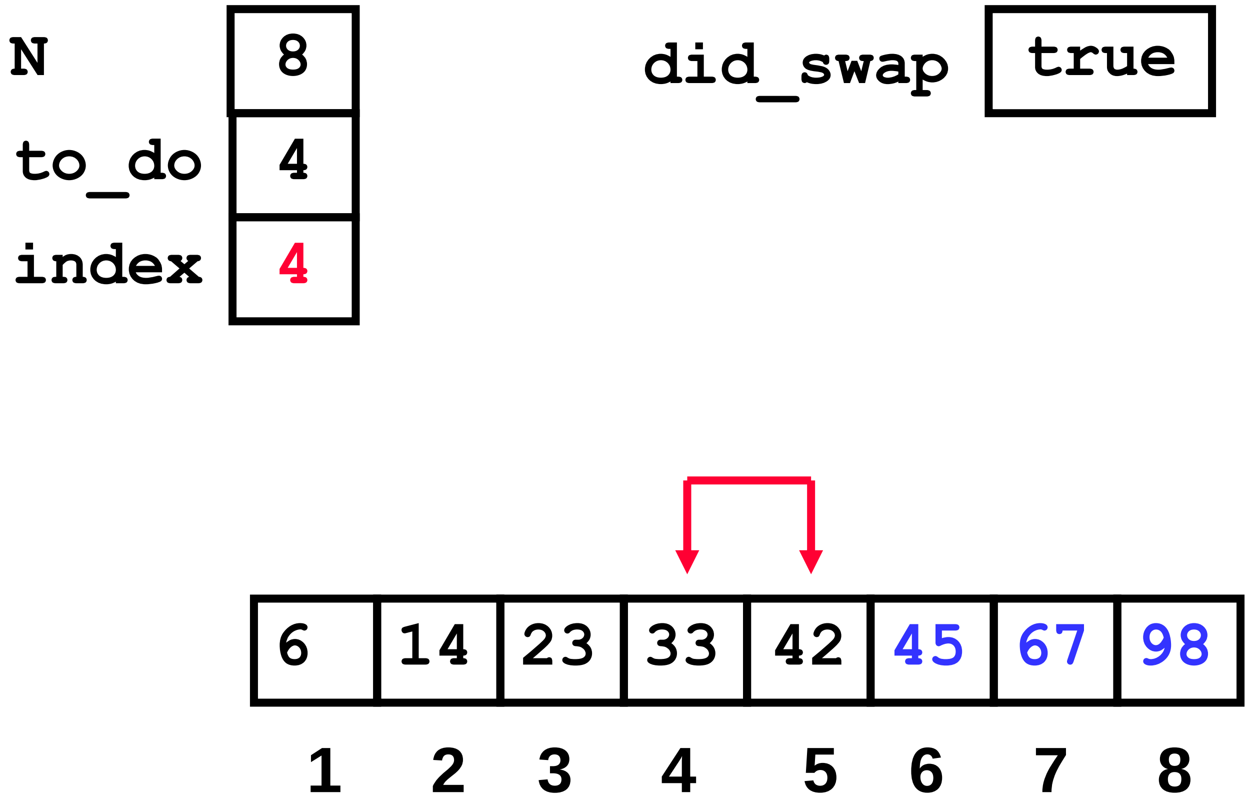
did_swap	true



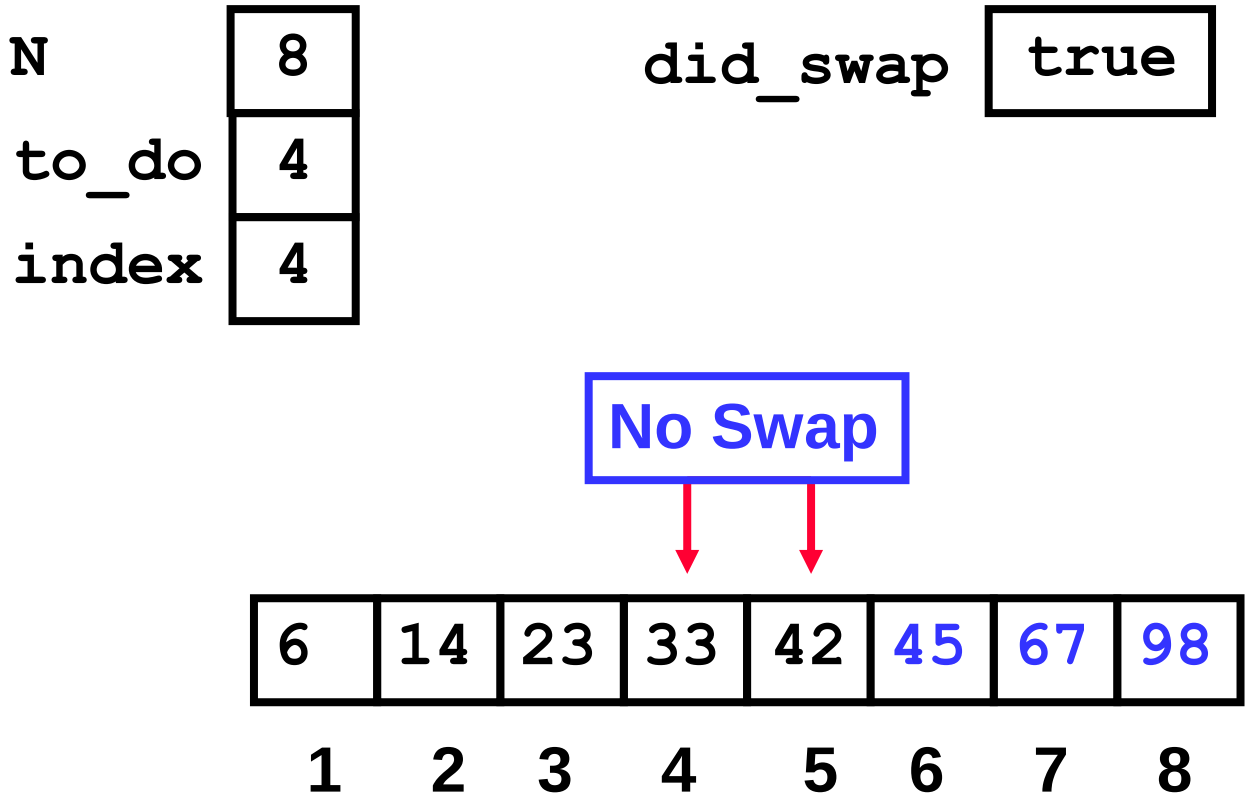
The Fourth “Bubble Up”



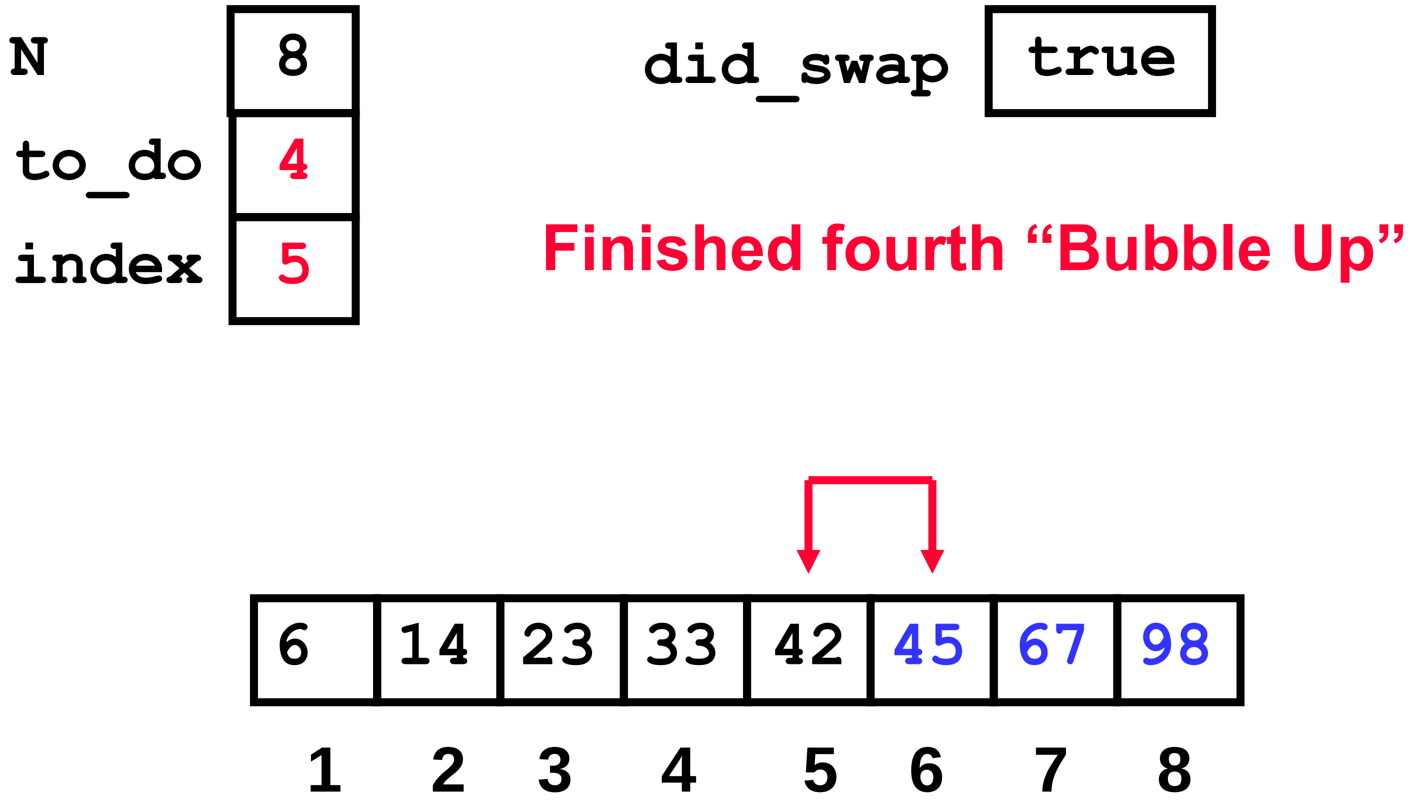
The Fourth “Bubble Up”



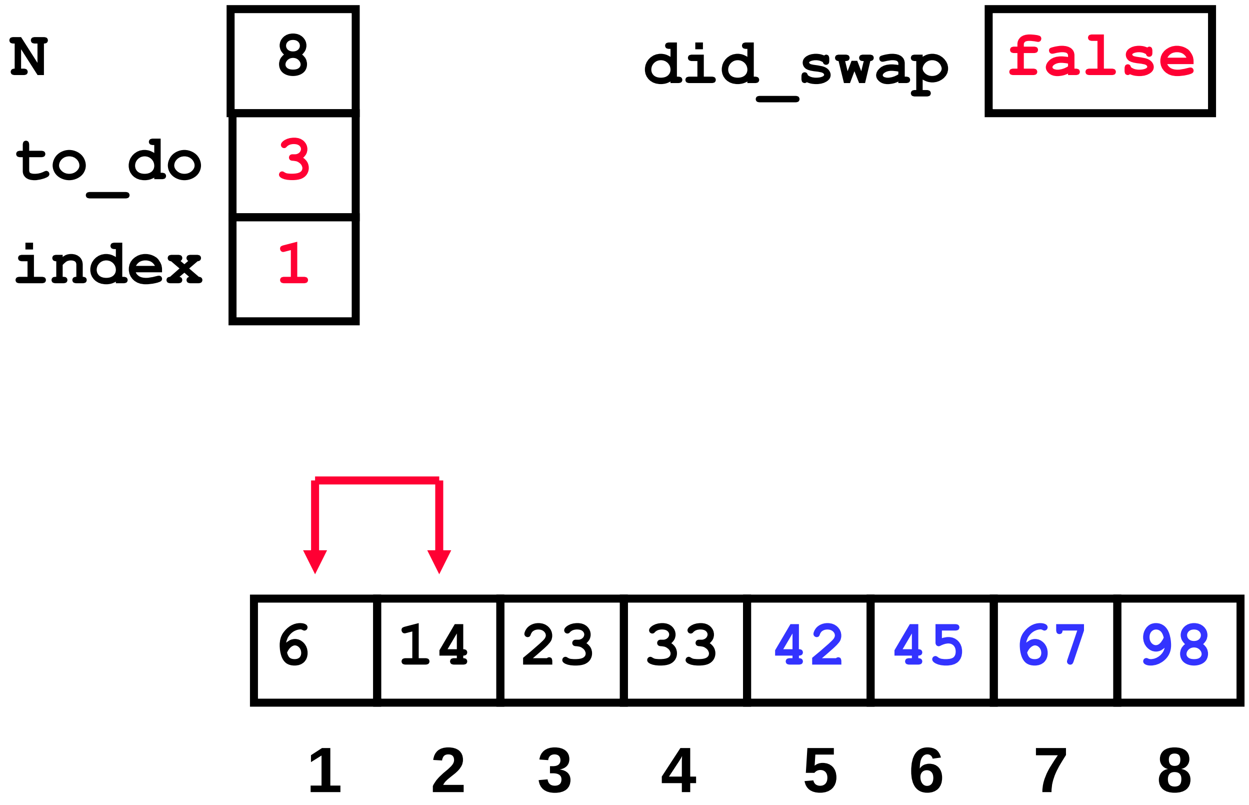
The Fourth "Bubble Up"



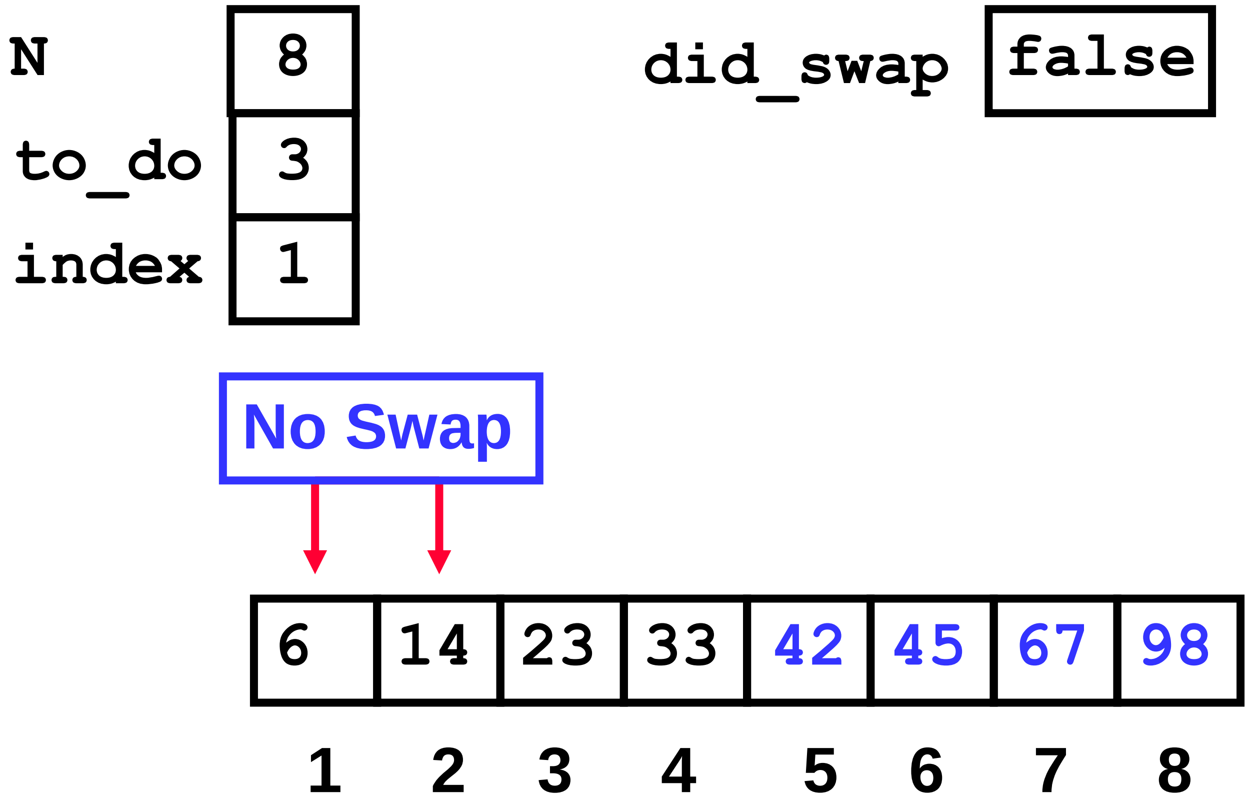
After Fourth Pass of Outer Loop



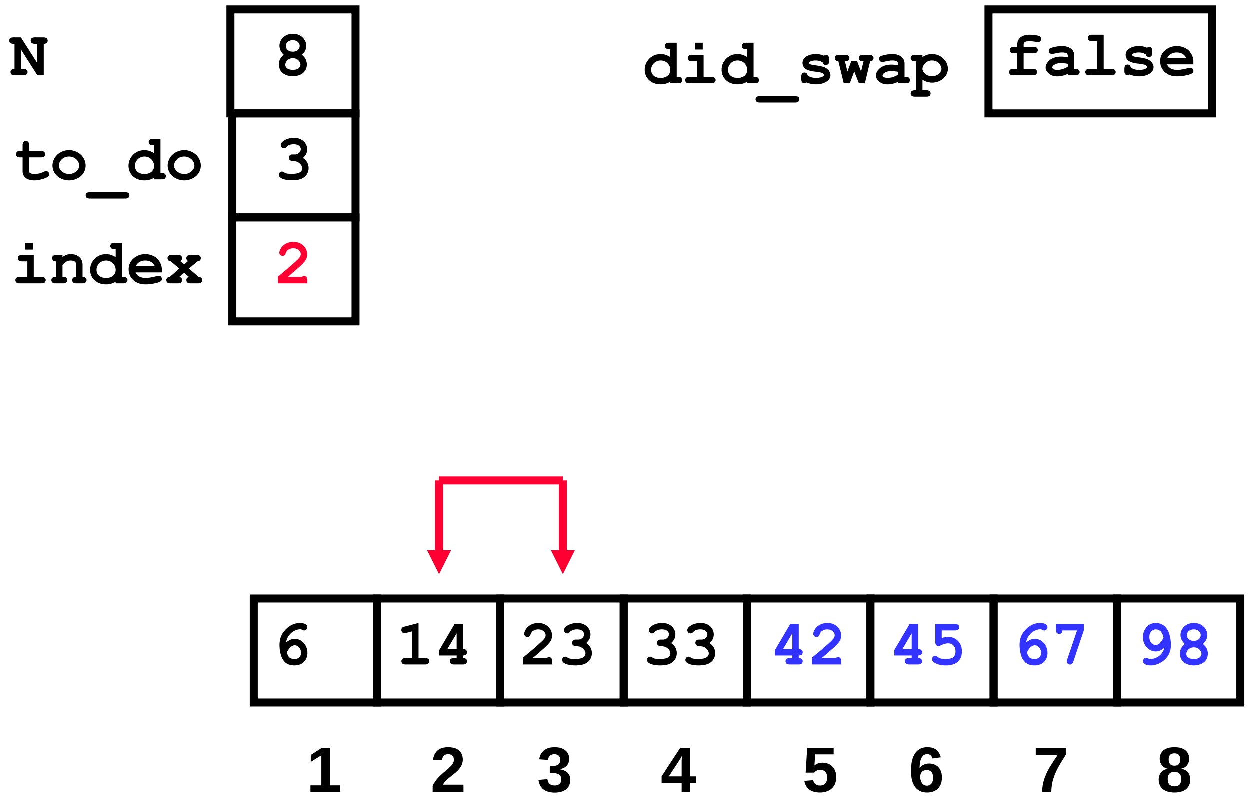
The Fifth "Bubble Up"



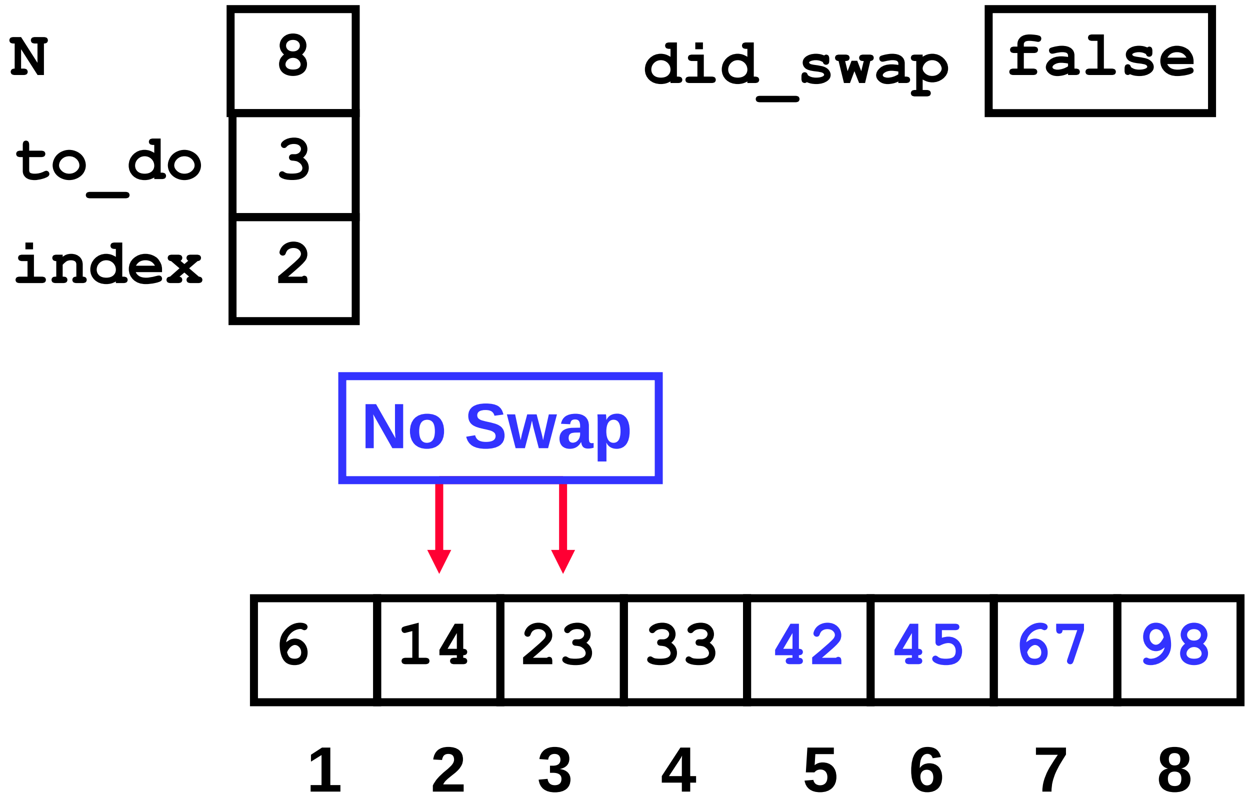
The Fifth "Bubble Up"



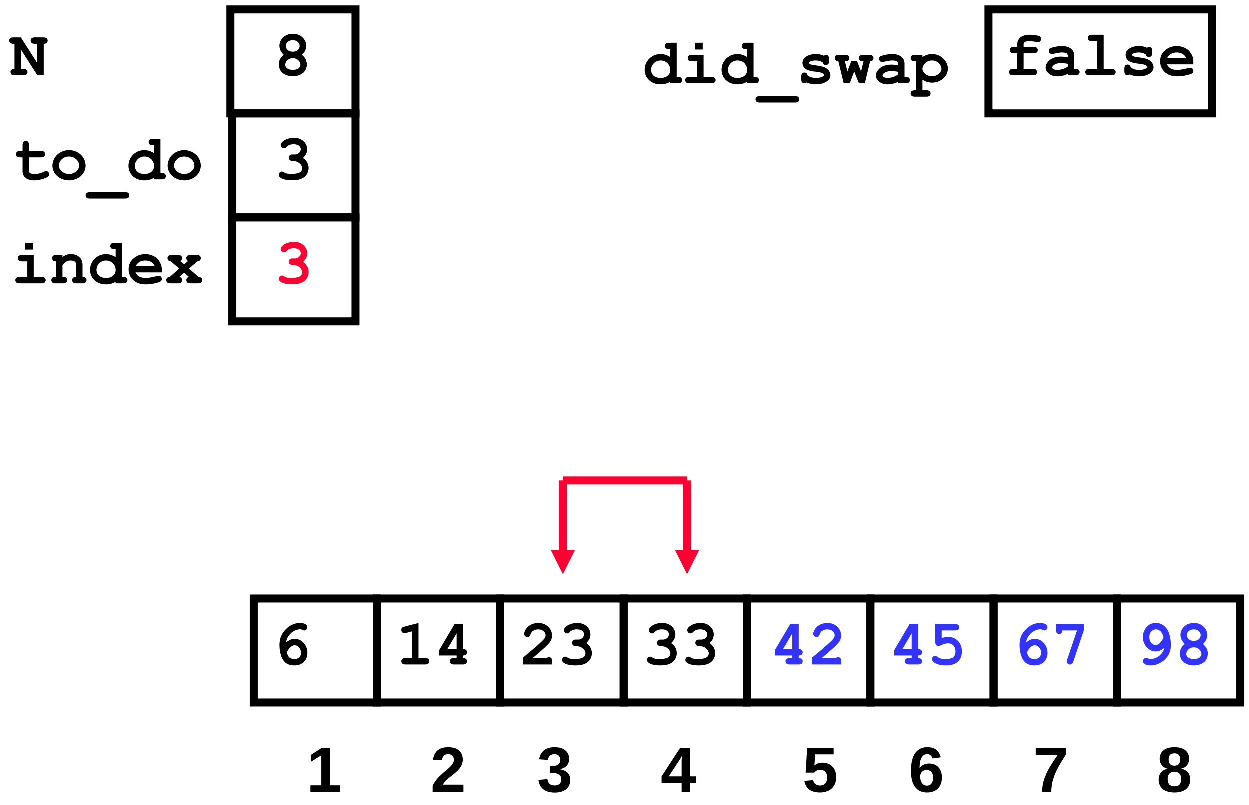
The Fifth “Bubble Up”



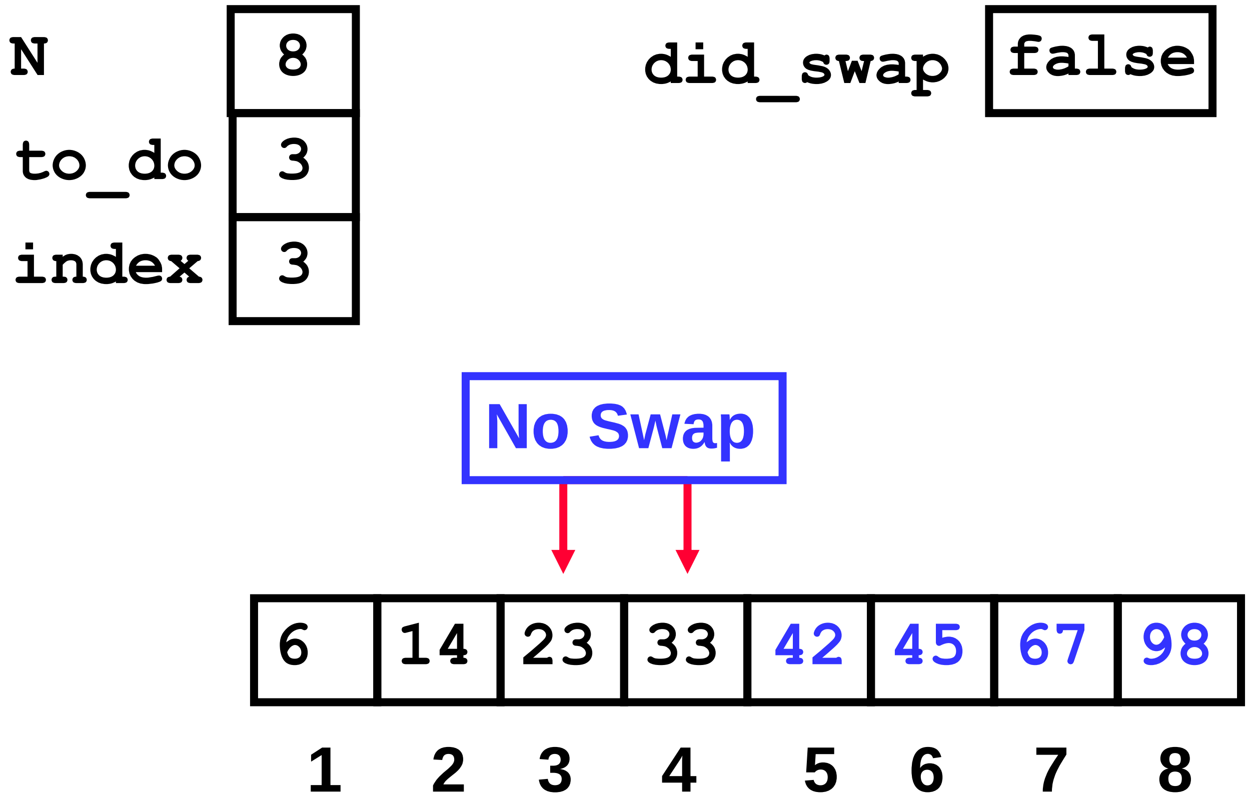
The Fifth "Bubble Up"



The Fifth "Bubble Up"



The Fifth "Bubble Up"

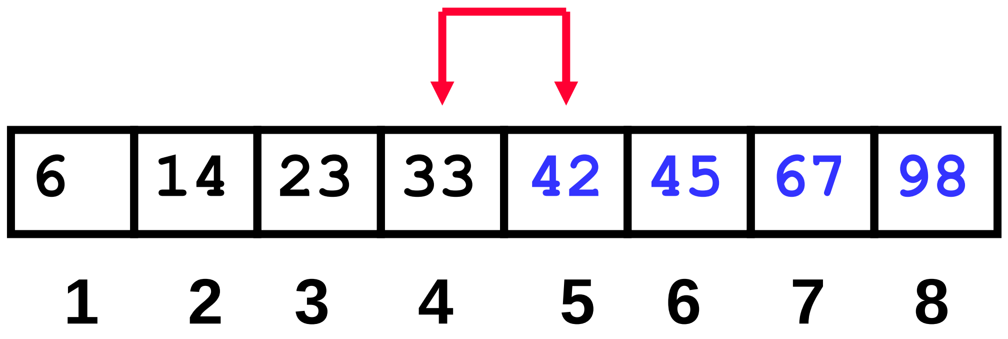


After Fifth Pass of Outer Loop

`N`	`8`
`to_do`	`3`
`index`	`4`

`did_swap` `false`

Finished fifth “Bubble Up”



Finished "Early"

N	8
to_do	3
index	4

did_swap	false

We didn't do any swapping, so all of the other elements must be correctly placed.

We can "skip" the last two passes of the outer loop.

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

- 한번의 패스가 끝나도 swap된 키들이 없다면 정렬되었음을 의미
- 이경우 정렬 알고리즘을 더 수행할 필요없이 끝나도 됨. 시간절약.

Complexity of Bubble Sort

- Two loops each proportional to n

  $T(n) = O(n^2)$
- It is possible to speed up bubble sort using the last index swapped, but doesn't affect worst case complexity

```
Algorithm bubbleSort(S, C)
Input sequence S, comparator C
Output sequence S sorted according to C
    for ( i = 0; i < S.size(); i++ )
        for ( j = 1; j < S.size() – i; j++ )
            if ( S.atIndex ( j – 1 ) > S.atIndex ( j ) )
                S.swap ( j-1, j );
    return(S)
```

만일 입력이 정렬되었다면

```
Algorithm bubbleSort(S, C)
Input sequence S, comparator C
Output sequence S sorted according to C
    for ( i = 0; i < S.size(); i++ ){
        flag=0;
        for ( j = 1; j < S.size() – i; j++ ){
            if ( S.atIndex ( j – 1 ) > S.atIndex ( j ) ){
                S.swap ( j-1, j );  flag=1;
            }
        }
        if(flag==0){ break;}//sorted
    }
    return(S)
```

버블 정렬 (2)

- BubbleSort 알고리즘
  - 의사코드

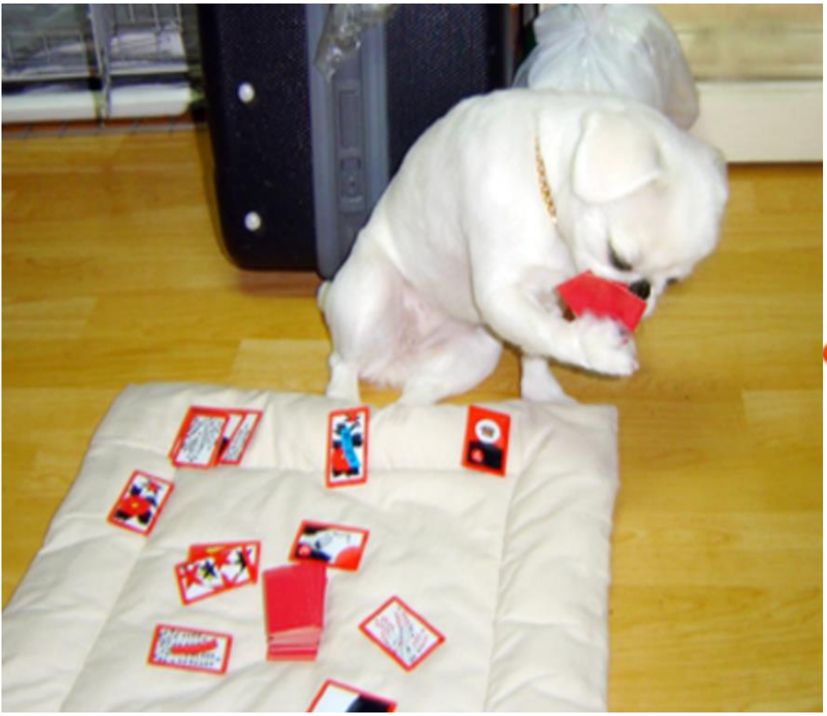
```
bubbleSort(a[])
    for (i ← a.length - 1;  i ≥ 0;  i ← i - 1) do  {
        for (j ← 0;  j < i;  j ← j+1) do  {
            if (a[j] > a[j+1]) then a[j]와 a[j+1]을 교환;
        }
    }
```

  - Java 코드

```
public static void bubbleSort(int[] a)  {
    int i, j;
    for (i = a.length – 1;  i ≥ 0;  --i)
        for (j = 0;  j < i; j++)
            if (a[j] > a[j+1])
                swap(a, j, j+1);
    }
```

  - 전체 비교 연산 수 = n(n-1)/2, 시간 복잡도 = $O(n^2)$

Insertion Sort(삽입정렬)



Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.

B	R	U	T	E	F	O	R	C	E

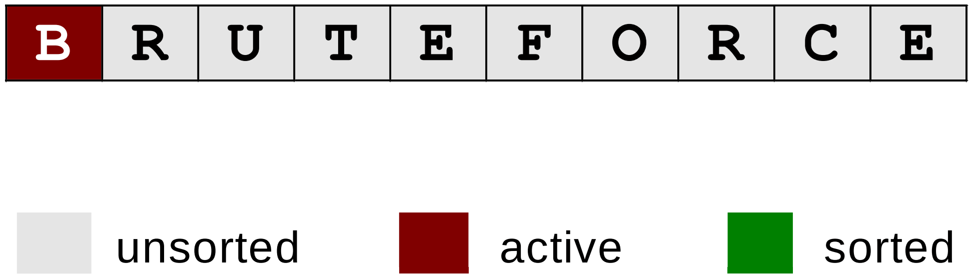
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



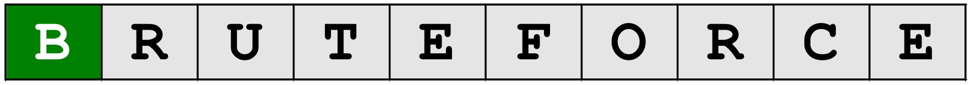
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

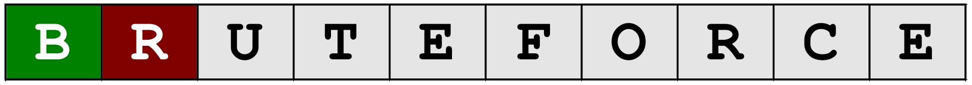
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

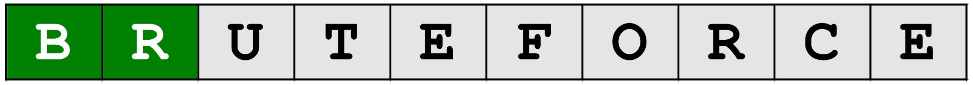
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

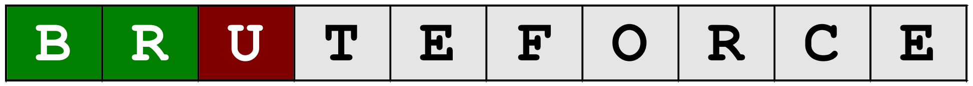
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

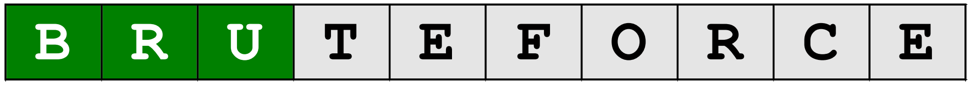
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

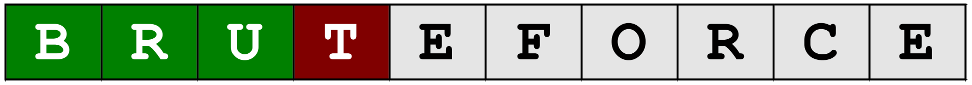
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

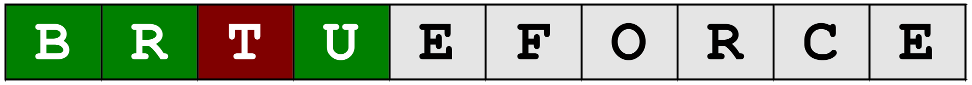
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

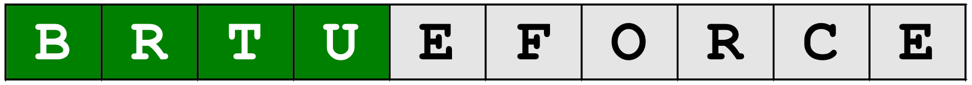
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

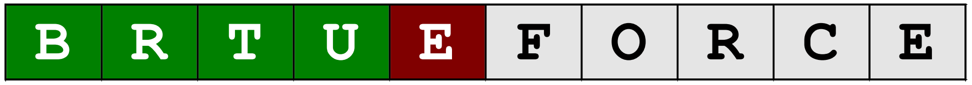
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

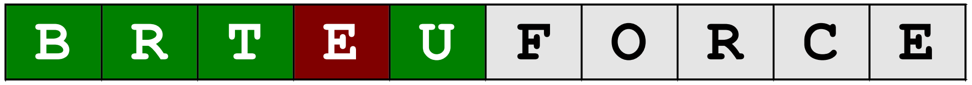
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

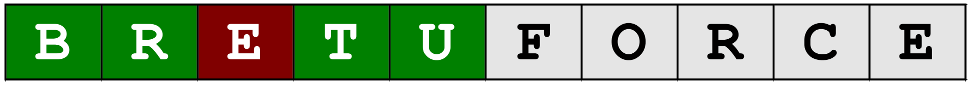
Insertion Sort Demo

Sorting problem:

- **Given an array of N integers, rearrange them so that they are in increasing order.**

Insertion sort

- **Brute-force sorting solution.**
- **Move left-to-right through array.**
- **Exchange next element with larger elements to its left, one-by-one.**



⬜ unsorted 🟥 active 🟩 sorted

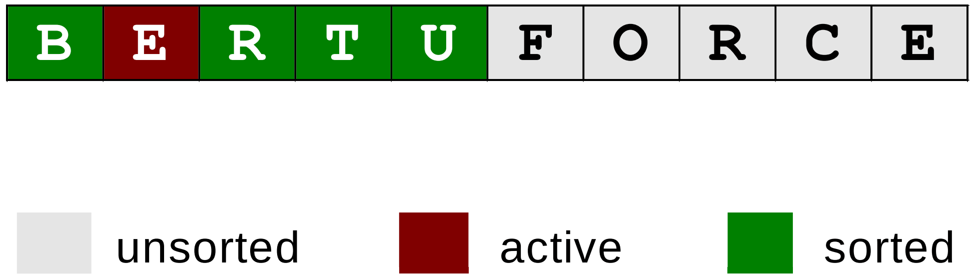
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



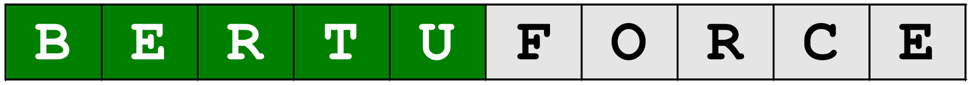
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

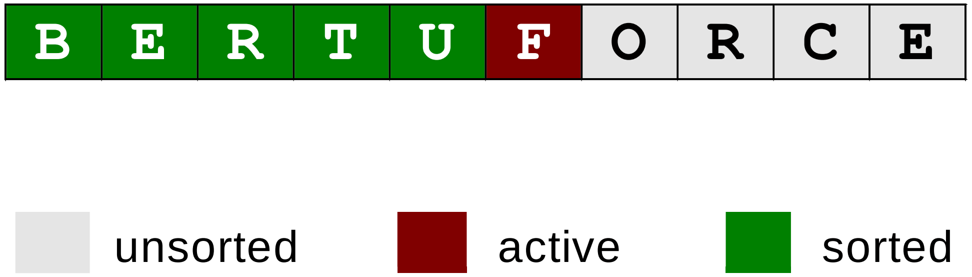
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



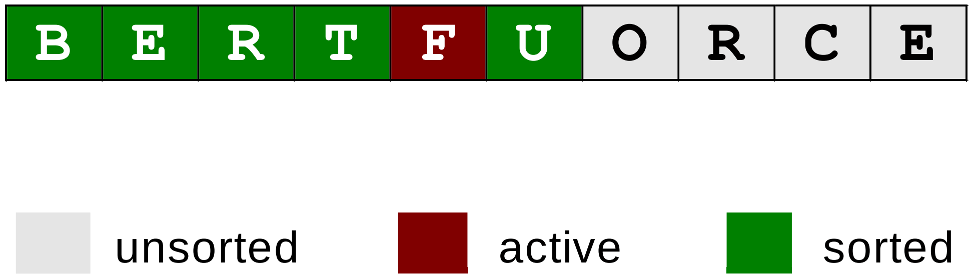
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



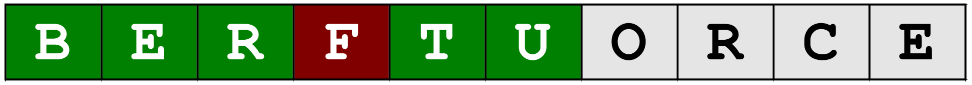
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

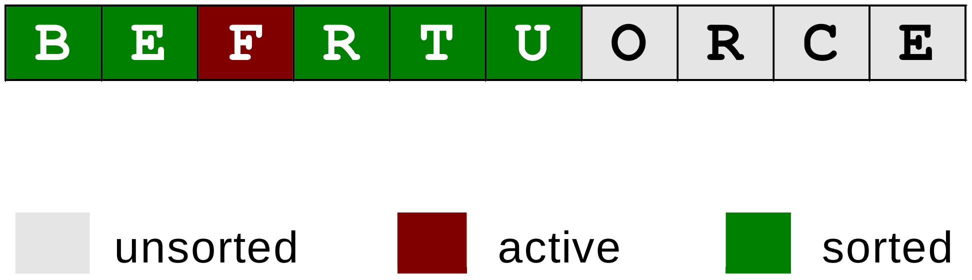
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



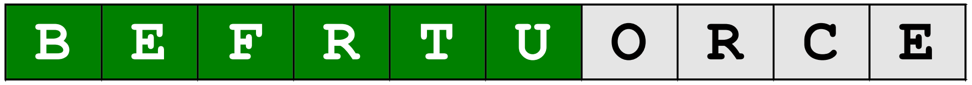
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

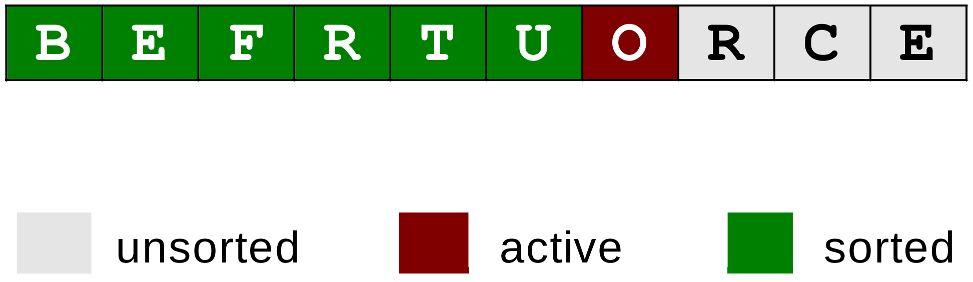
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



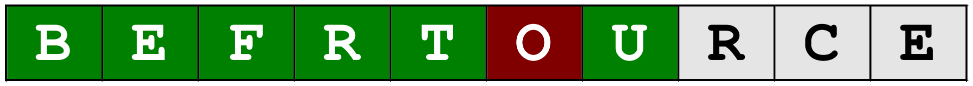
Insertion Sort Demo

Sorting problem:

- **Given an array of N integers, rearrange them so that they are in increasing order.**

Insertion sort

- **Brute-force sorting solution.**
- **Move left-to-right through array.**
- **Exchange next element with larger elements to its left, one-by-one.**



⬜ unsorted 🟥 active 🟩 sorted

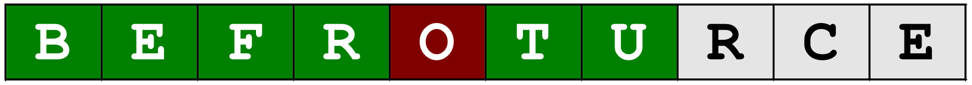
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

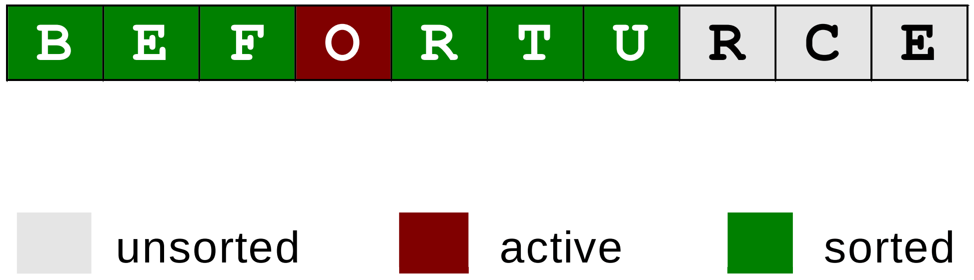
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.

B	E	F	O	R	T	U	R	C	E

⬜ unsorted 🟥 active 🟩 sorted

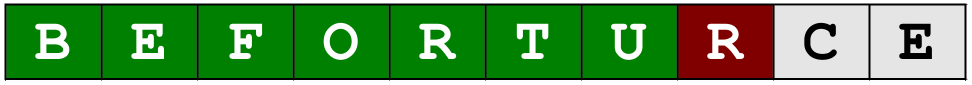
Insertion Sort Demo

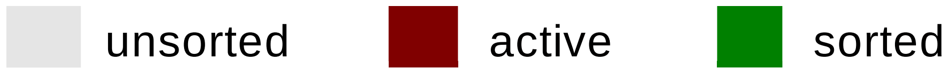
Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.





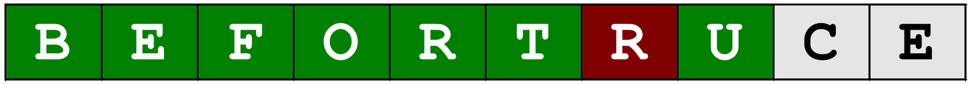
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

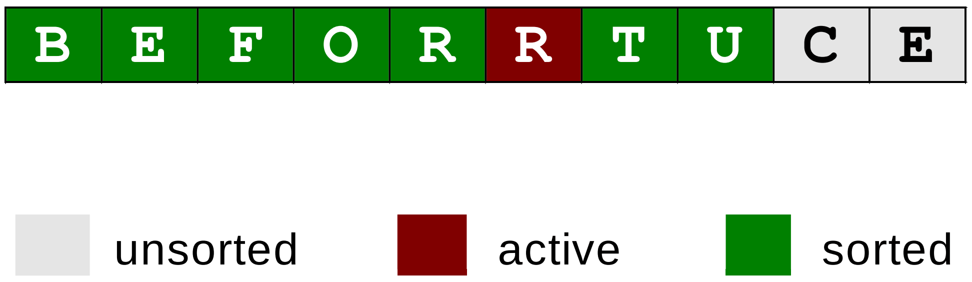
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



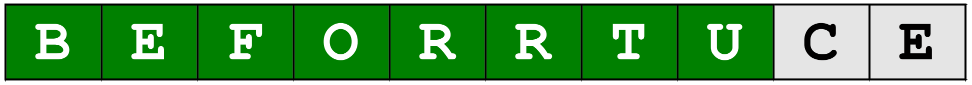
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

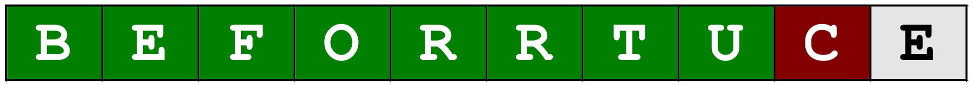
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

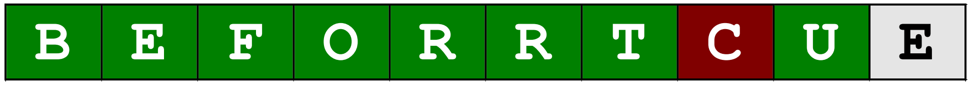
Insertion Sort Demo

Sorting problem:

- **Given an array of N integers, rearrange them so that they are in increasing order.**

Insertion sort

- **Brute-force sorting solution.**
- **Move left-to-right through array.**
- **Exchange next element with larger elements to its left, one-by-one.**



⬜ unsorted 🟥 active 🟩 sorted

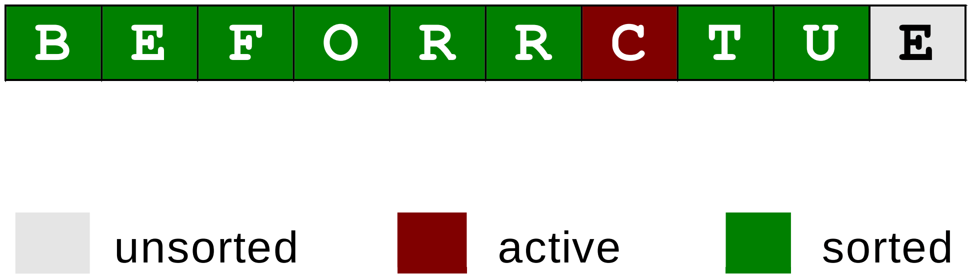
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



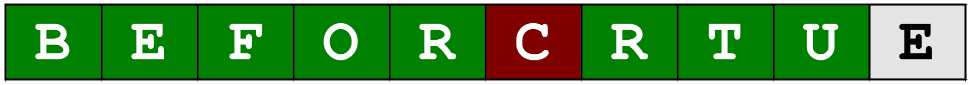
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

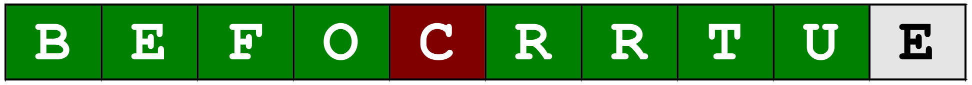
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

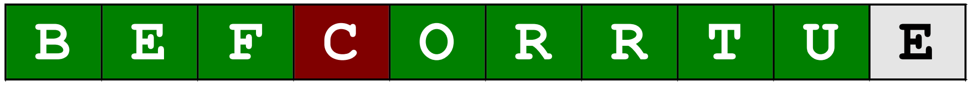
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

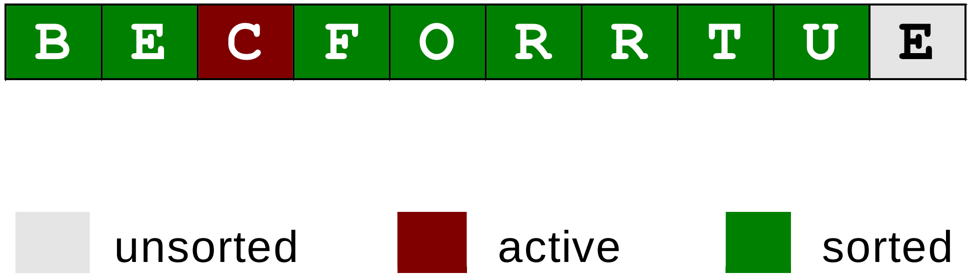
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



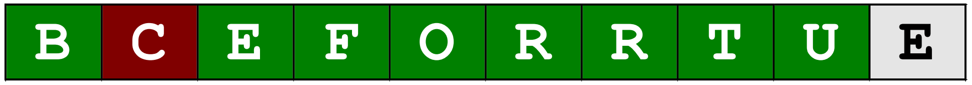
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

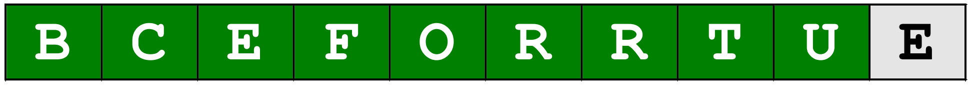
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.

B	C	E	F	O	R	R	T	U	E

⬜ unsorted 🟥 active 🟩 sorted

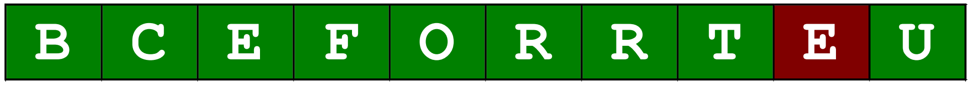
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

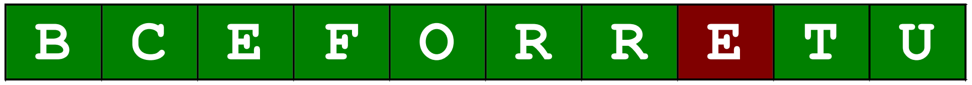
Insertion Sort Demo

Sorting problem:

- **Given an array of N integers, rearrange them so that they are in increasing order.**

Insertion sort

- **Brute-force sorting solution.**
- **Move left-to-right through array.**
- **Exchange next element with larger elements to its left, one-by-one.**



⬜ unsorted 🟥 active 🟩 sorted

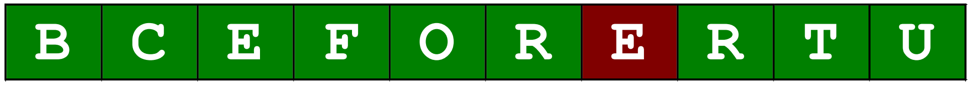
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

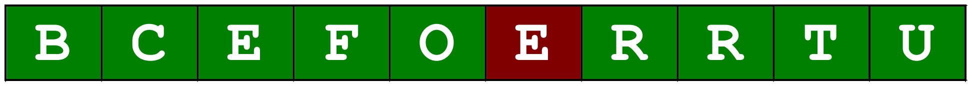
Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.



⬜ unsorted 🟥 active 🟩 sorted

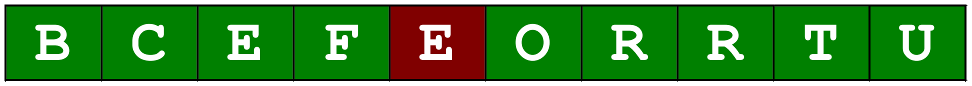
Insertion Sort Demo

Sorting problem:

- **Given an array of N integers, rearrange them so that they are in increasing order.**

Insertion sort

- **Brute-force sorting solution.**
- **Move left-to-right through array.**
- **Exchange next element with larger elements to its left, one-by-one.**



⬜ unsorted 🟥 active 🟩 sorted

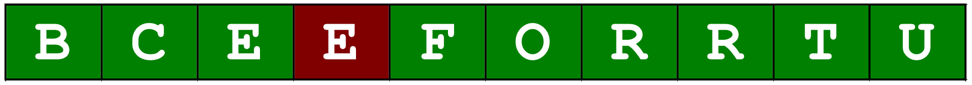
Insertion Sort Demo

Sorting problem:

- **Given an array of N integers, rearrange them so that they are in increasing order.**

Insertion sort

- **Brute-force sorting solution.**
- **Move left-to-right through array.**
- **Exchange next element with larger elements to its left, one-by-one.**



⬜ unsorted 🟥 active 🟩 sorted

Insertion Sort Demo

Sorting problem:

- Given an array of N integers, rearrange them so that they are in increasing order.

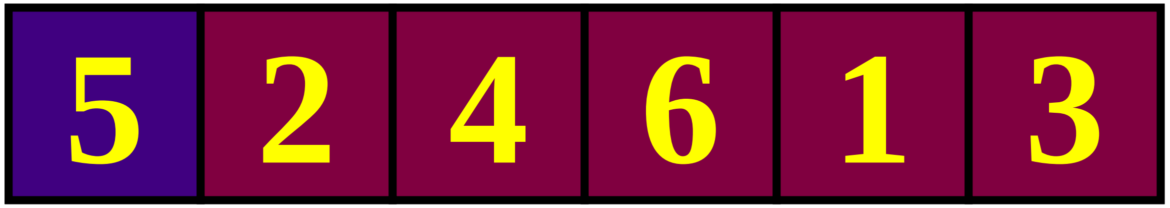
Insertion sort

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.

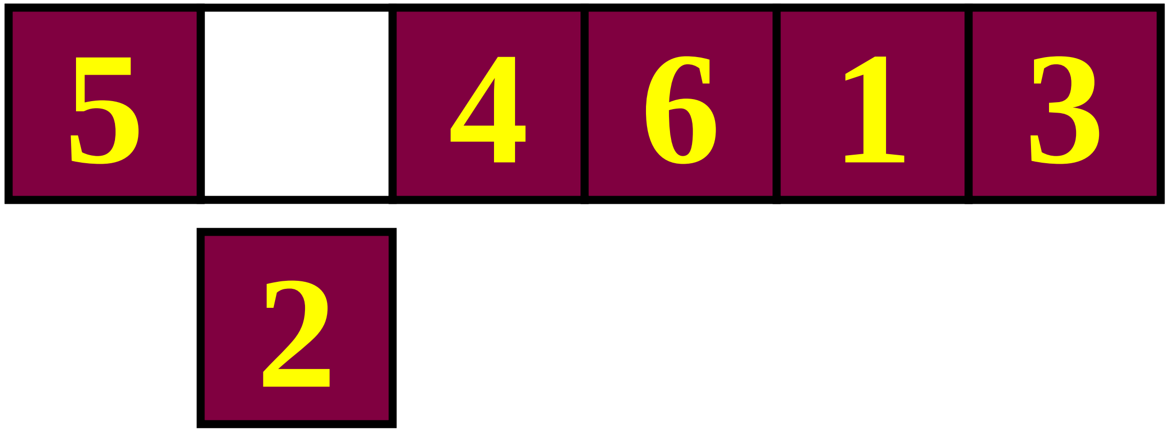
B	C	E	E	F	O	R	R	T	U

⬜ unsorted 🟥 active 🟩 sorted

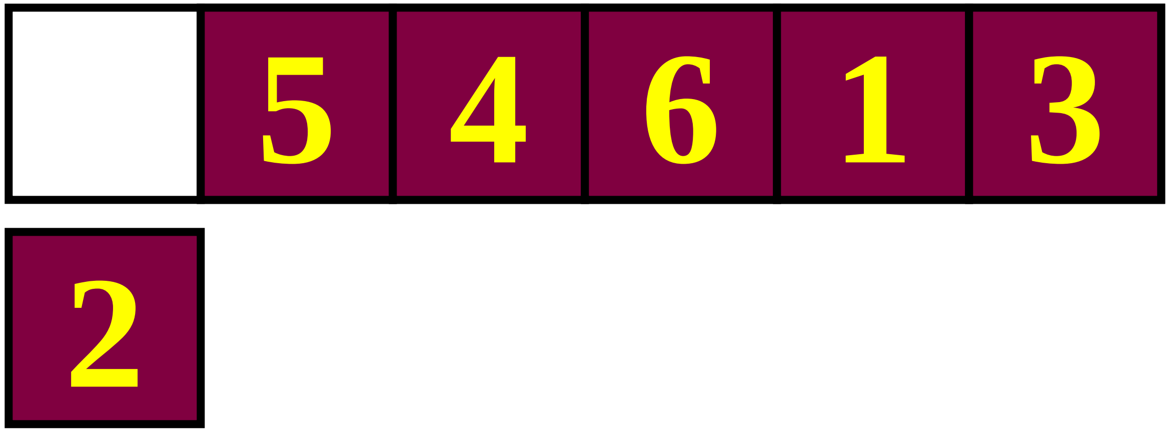
Insertion Sort



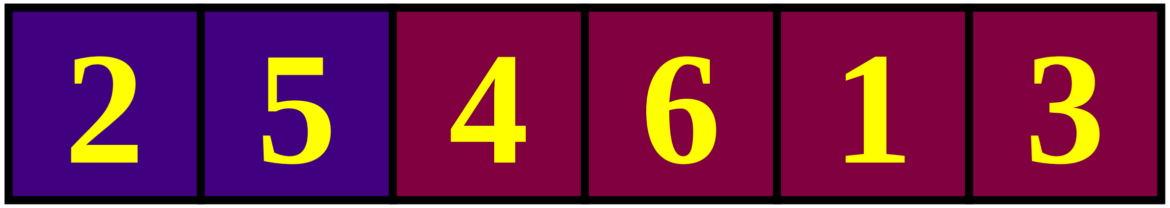
Insertion Sort



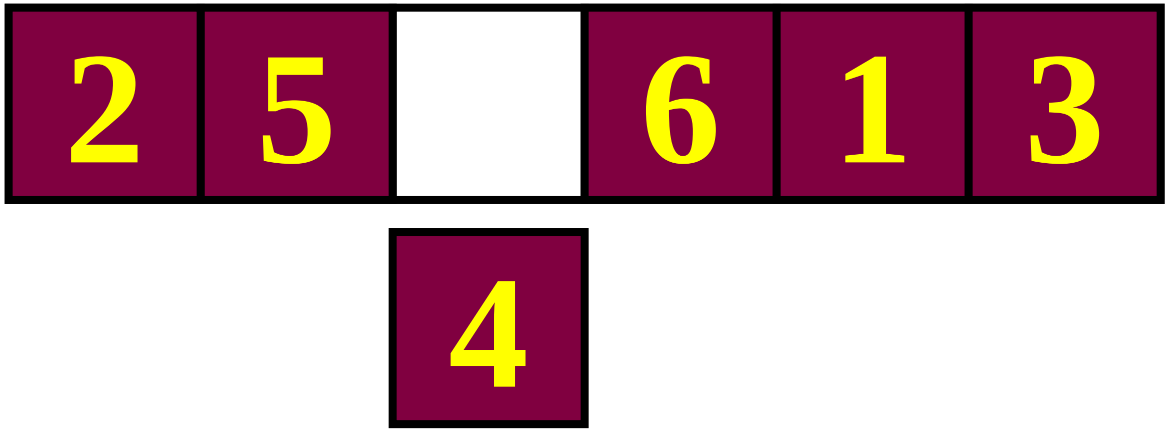
Insertion Sort



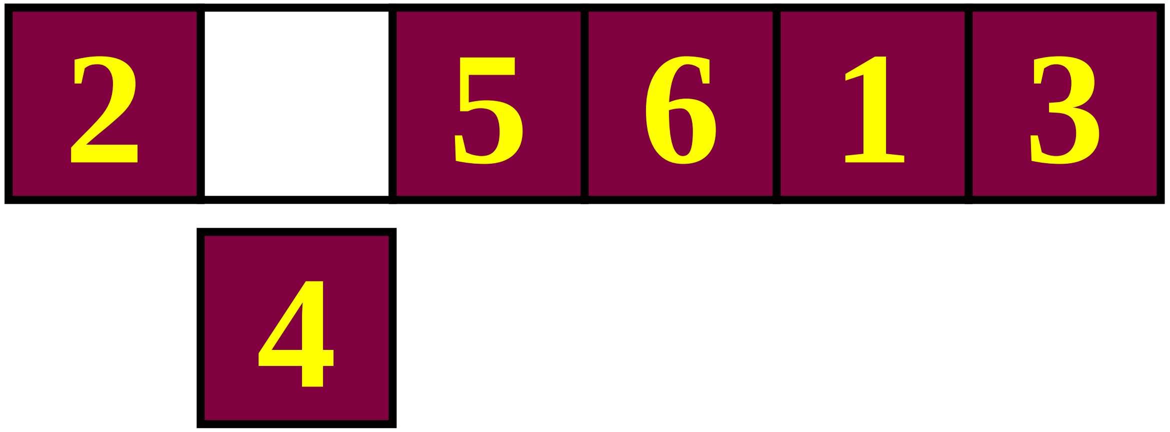
Insertion Sort



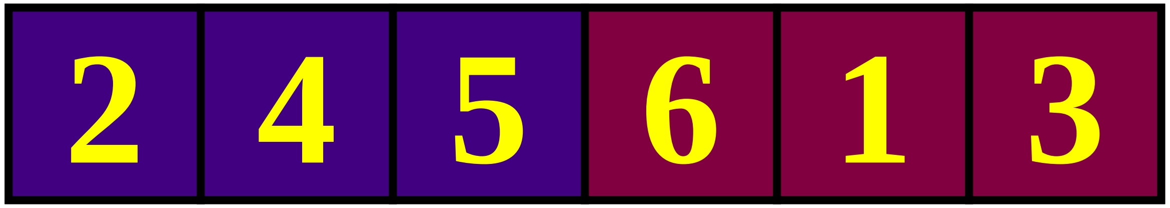
Insertion Sort



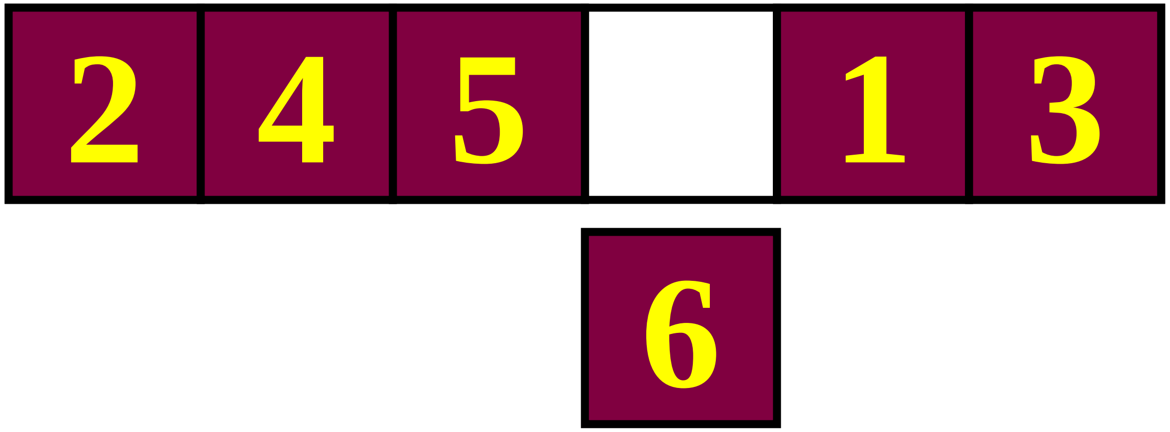
Insertion Sort



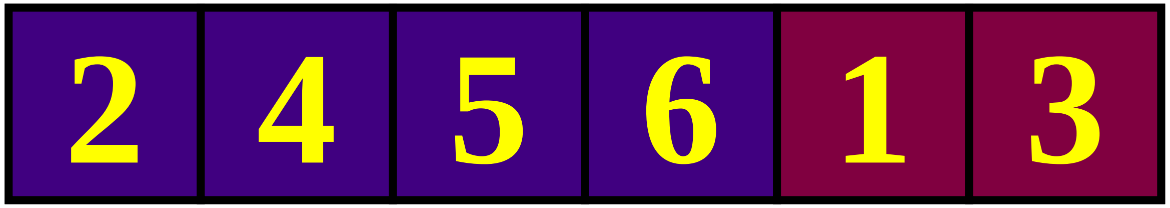
Insertion Sort



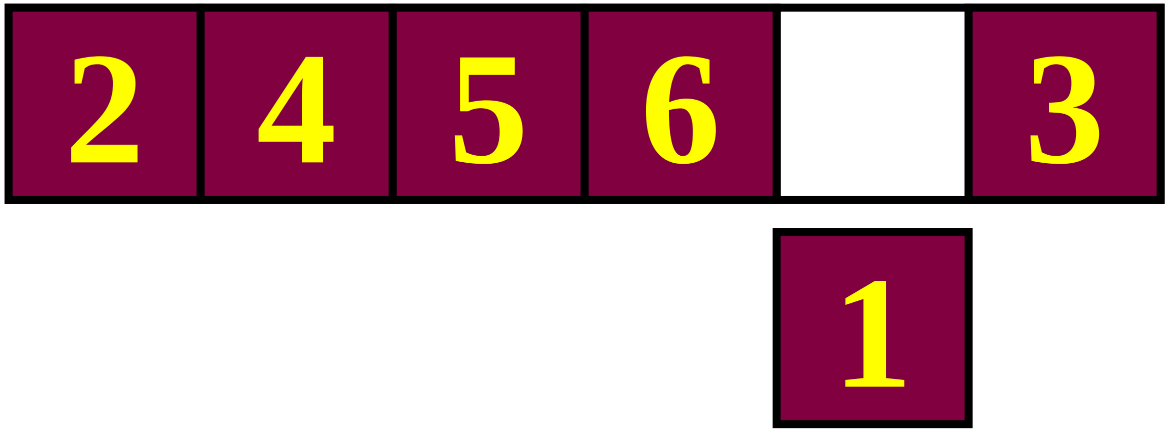
Insertion Sort



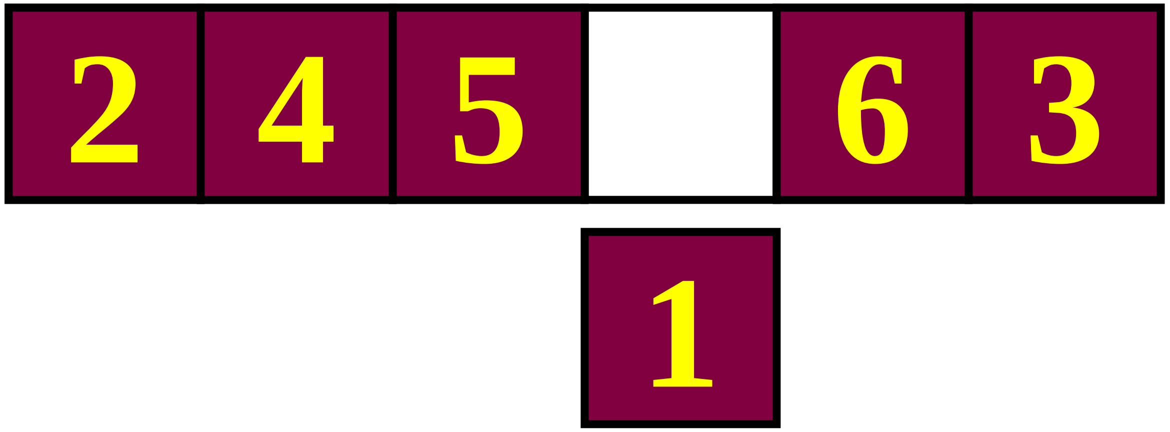
Insertion Sort



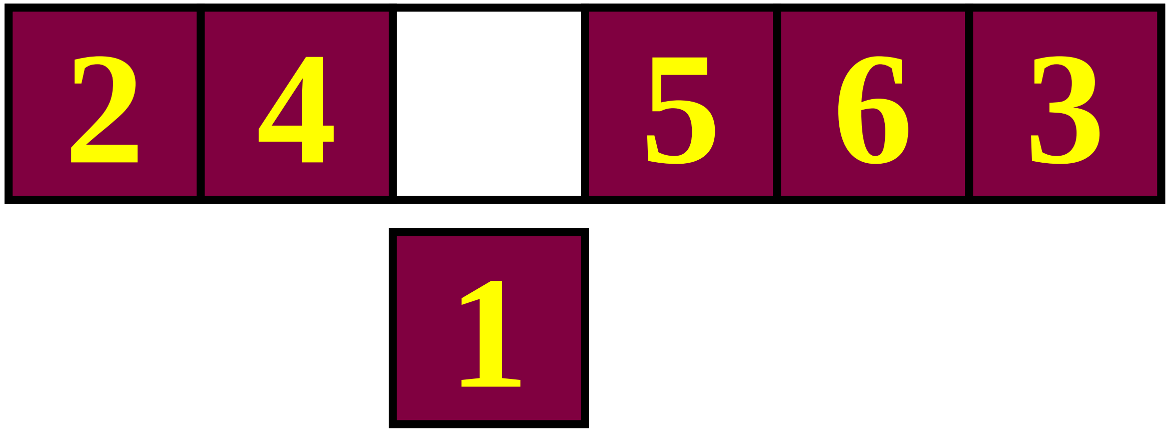
Insertion Sort



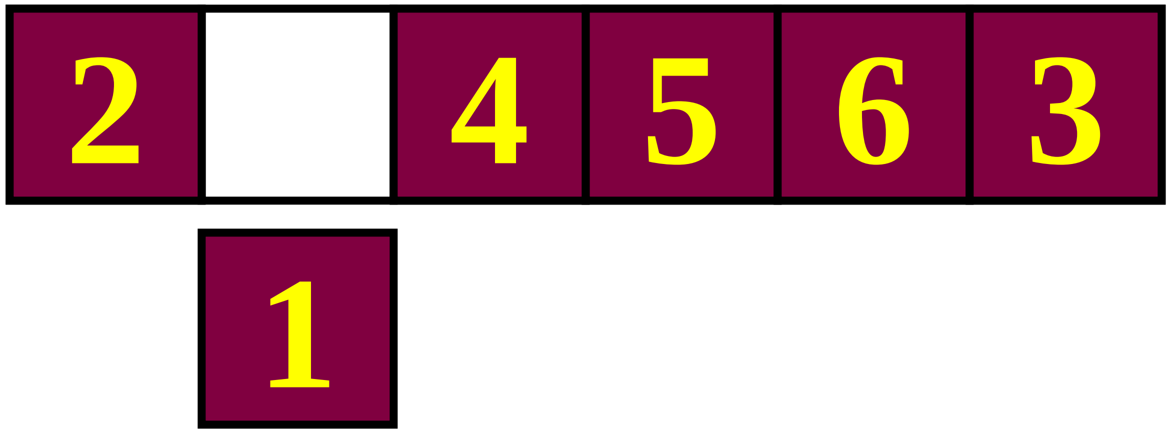
Insertion Sort



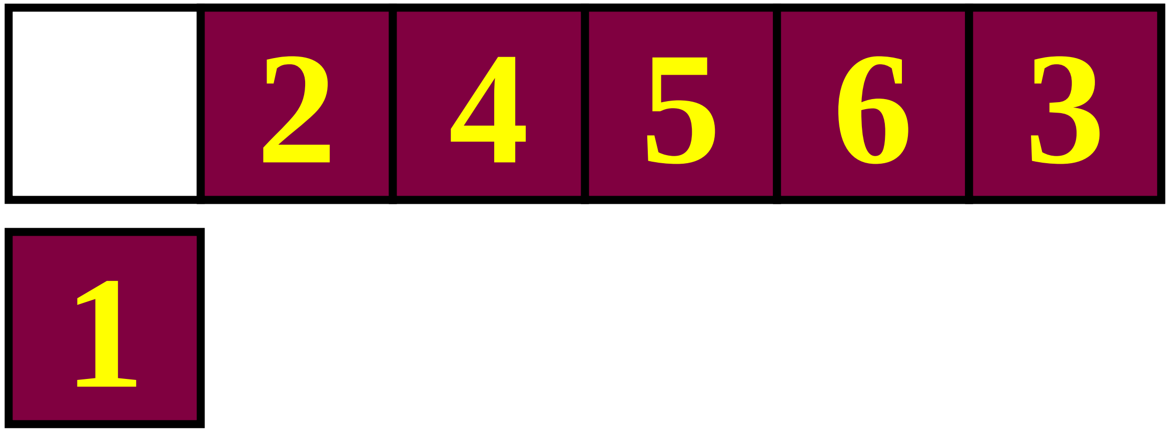
Insertion Sort



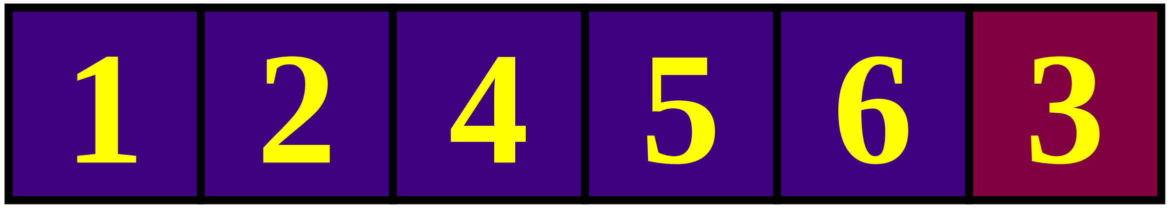
Insertion Sort



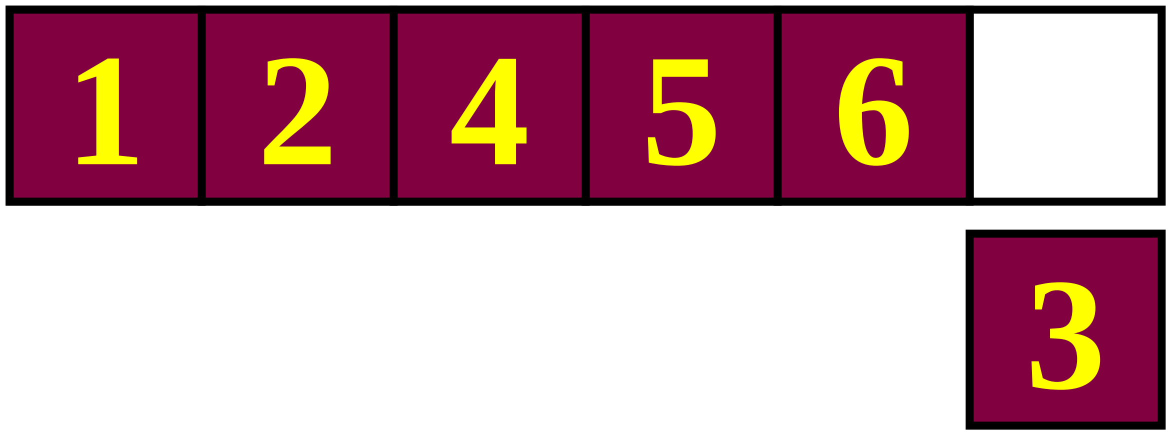
Insertion Sort



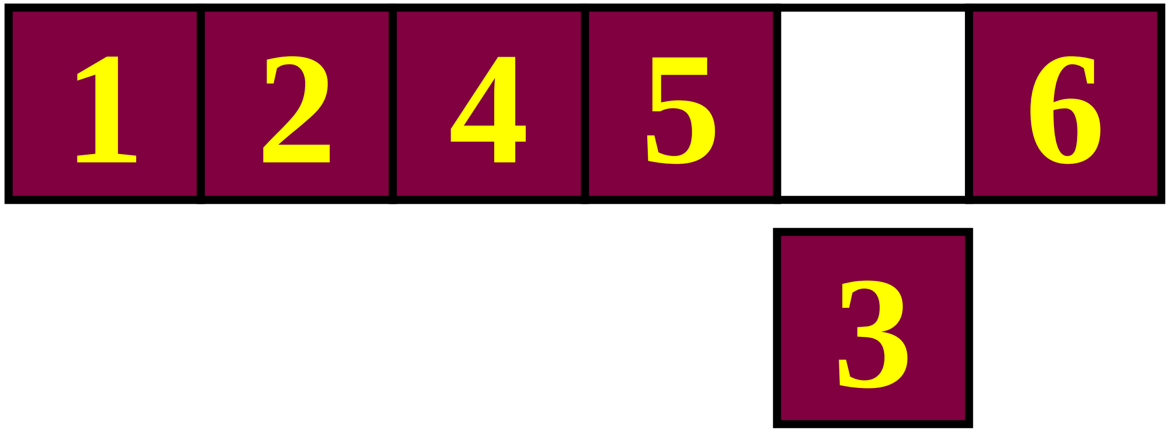
Insertion Sort



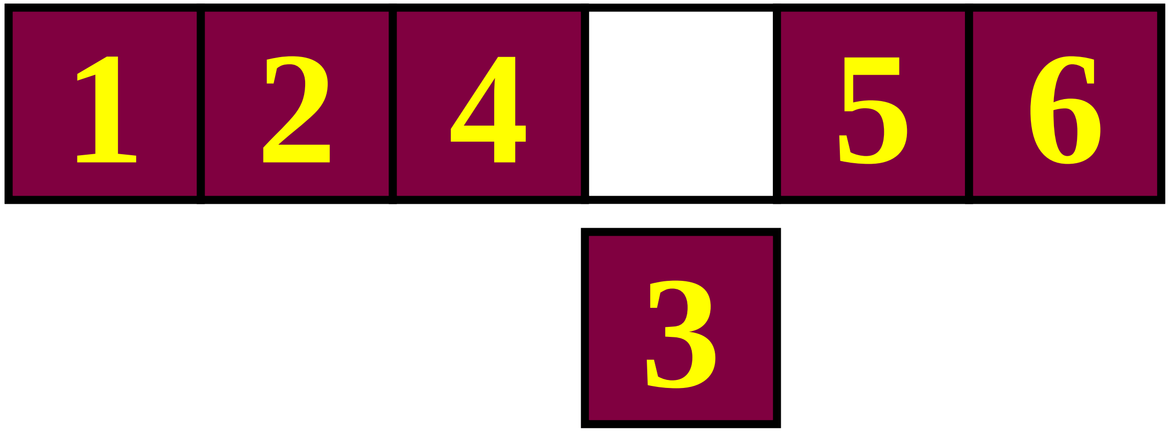
Insertion Sort



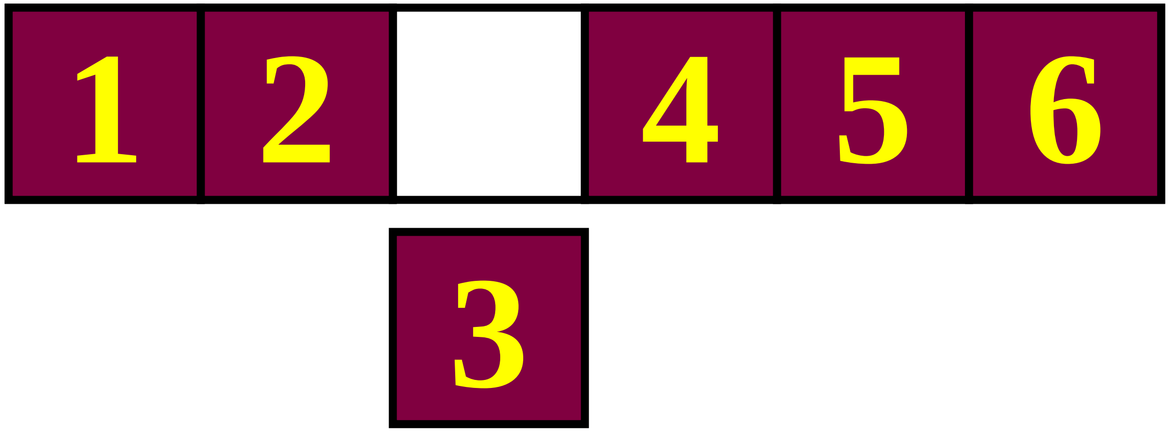
Insertion Sort



Insertion Sort



Insertion Sort



Insertion Sort

1	2	3	4	5	6

삽입 정렬

- InsertionSort 알고리즘

```
insertionSort(a[])
        // 원소 ak는 a[i], 0 < i < n,
        // 원소 k(=a[i], 0 < i < n)를 부분 배열 a[0 : i-1]에 오름차순으로 삽입
    for (i = 1;  i < n;  i ← i + 1)  {   // 두 번째 원소 a[1]부터
        k ← a[i];     // k는 임시 저장소
        j ← i;
        if (a[j-1] > k) then move ← true
        else move ← false;
        while (move) do  {
                a[j] ← a[j-1];   // a[j-1]을 오른쪽으로 한자리 이동시켜 빈자리를 만듦
                j ← j-1;
                if (j > 0 and a[j-1] > k)
                then move ← true
                else move ← false;
        }
        a[j] ← k;   //k를 빈자리에 삽입
    }   // for
end InsertionSort()
```

Next topics

$\boldsymbol{O}(\boldsymbol{n} \log \boldsymbol{n})$정렬 알고리즘

Quick sort(퀵정렬)

Merge sort(합병정렬)

Heap sort(히프정렬)

감사합니다.