Data Structures and Algorithms

Why Sorting?

- Practical application
  - People by last name
  - Countries by population
  - Search engine results by relevance
- Fundamental to other algorithms
- Different algorithms have different asymptotic and constant-factor trade-offs
  - No single ‘best’ sort for all scenarios
  - Knowing one way to sort just isn’t enough
- Many to approaches to sorting which can be used for other problems

Problem statement

- There are *n* comparable elements in an array and we want to rearrange them to be in increasing order

- Pre:
  - An array **`A`** of data records
  - A value in each data record
  - A comparison function
    - `<`, `=`, `>`, `compareTo`

- Post:
  - For each distinct position **`i`** and **`j`** of **`A`**, if **`i<j`** then **`A[i]`** ≤ **`A[j]`**
  - **`A`** has all the same data it started with

Sorting Classification

In memory sorting			External sorting
Comparison sorting $\Omega(N \log N)$		Specialized Sorting	
$O(N^2)$	$O(N \log N)$	$O(N)$	# of tape accesses
• Bubble Sort • Selection Sort • Insertion Sort • Shell Sort	• Merge Sort • Quick Sort • Heap Sort	• Bucket Sort • Radix Sort	• Simple External Merge Sort • Variations

Complexity Of all sorting algorithm

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)
Tim Sort	O(n)	O(nlogn)	O(nlogn)	O(n)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)

Bubble Sort

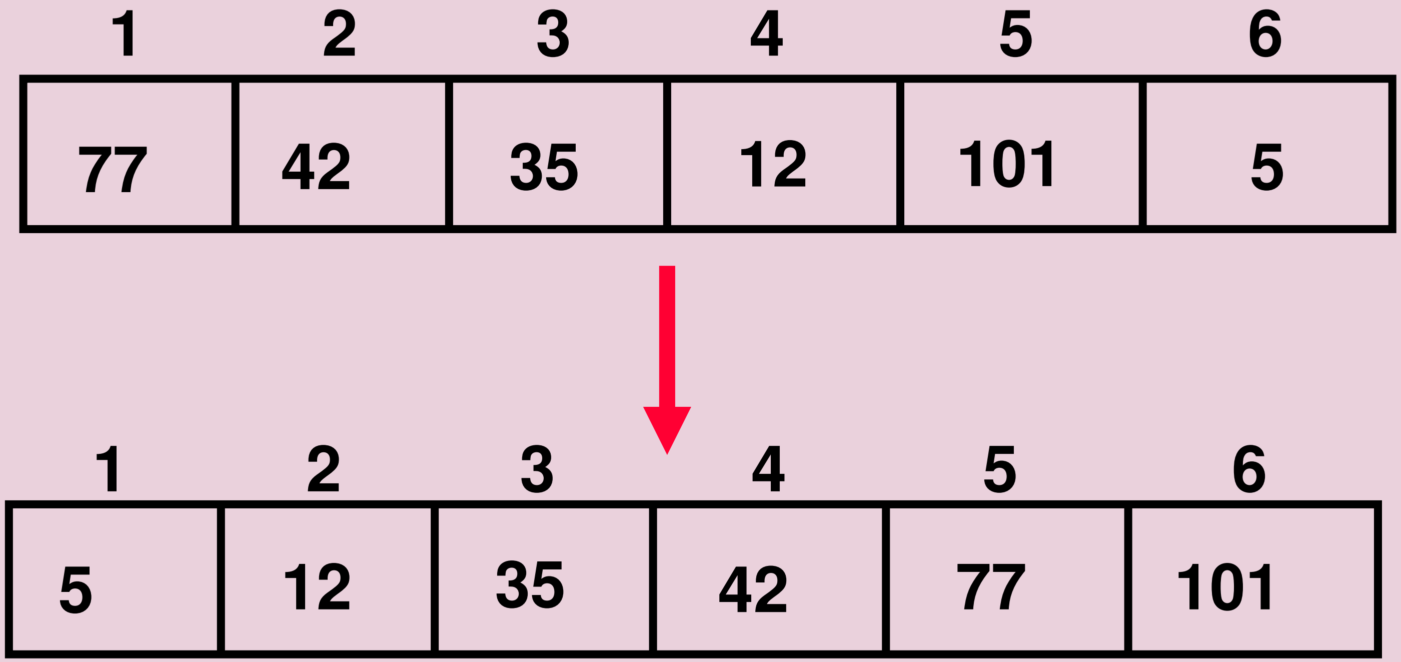
https://www.hackerearth.com/practice/algorithms/sorting/bubble-sort/visualize/

or

https://visualgo.net/bn/sorting

Sorting

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



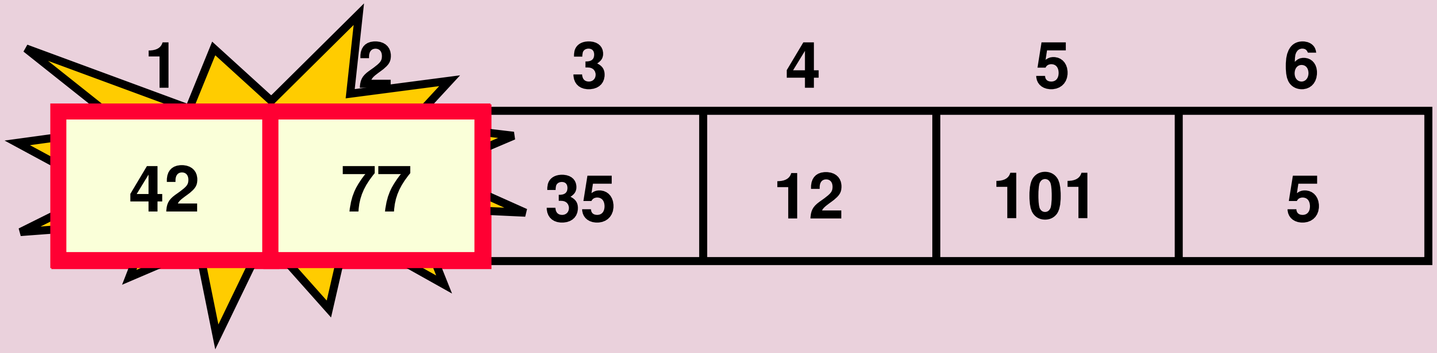
"Bubbling Up" the Largest Element

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

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

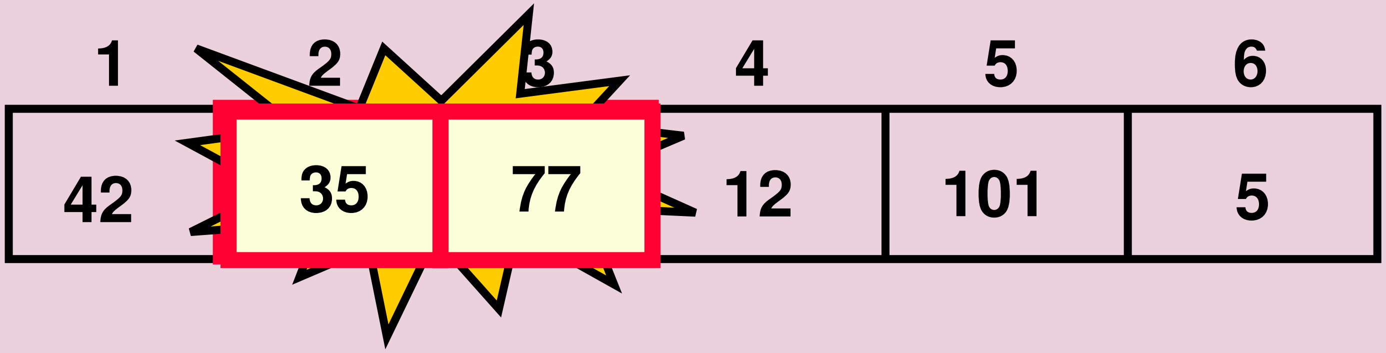
"Bubbling Up" the Largest Element

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



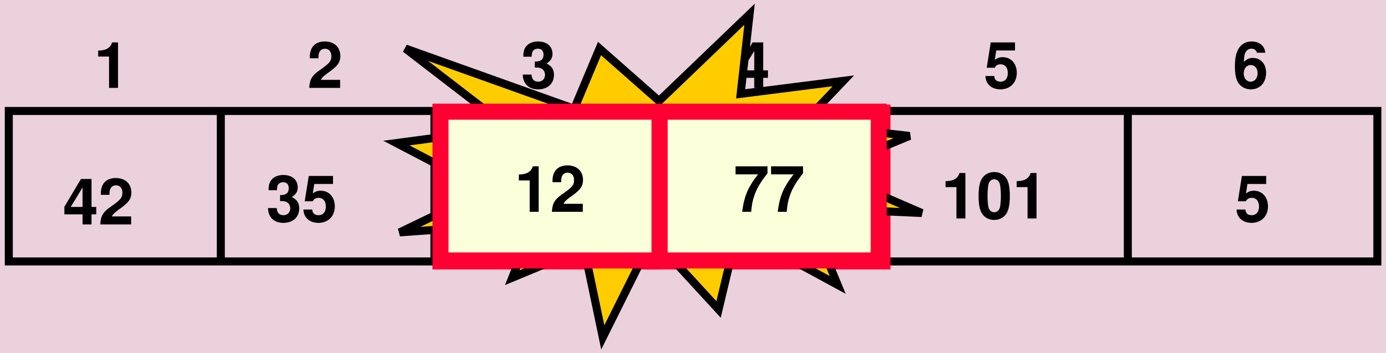
"Bubbling Up" the Largest Element

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



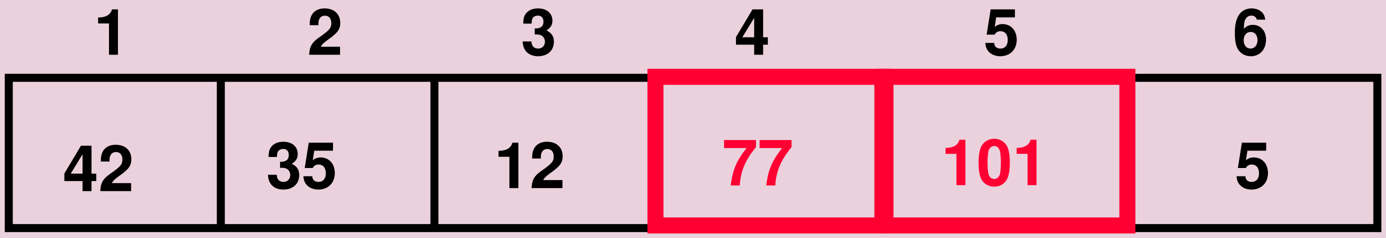
"Bubbling Up" the Largest Element

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



"Bubbling Up" the Largest Element

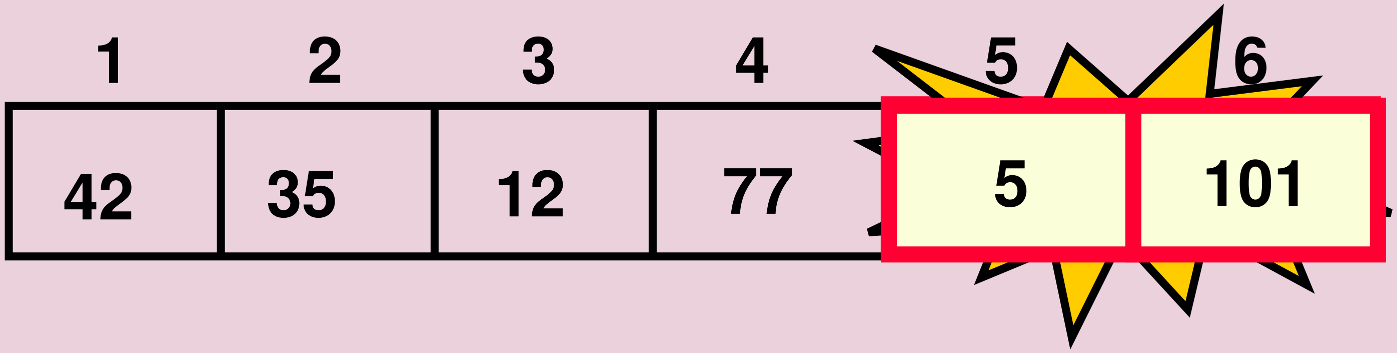
- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



No need to swap

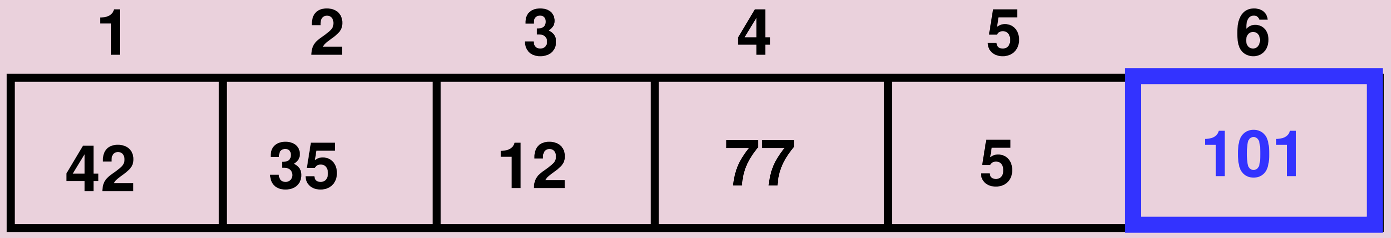
"Bubbling Up" the Largest Element

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



"Bubbling Up" the Largest Element

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



Largest value correctly placed

Items of Interest

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

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

Largest value correctly placed

Repeat “Bubble Up” How Many Times?

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

“Bubbling” All the Elements

N - 1

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

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

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

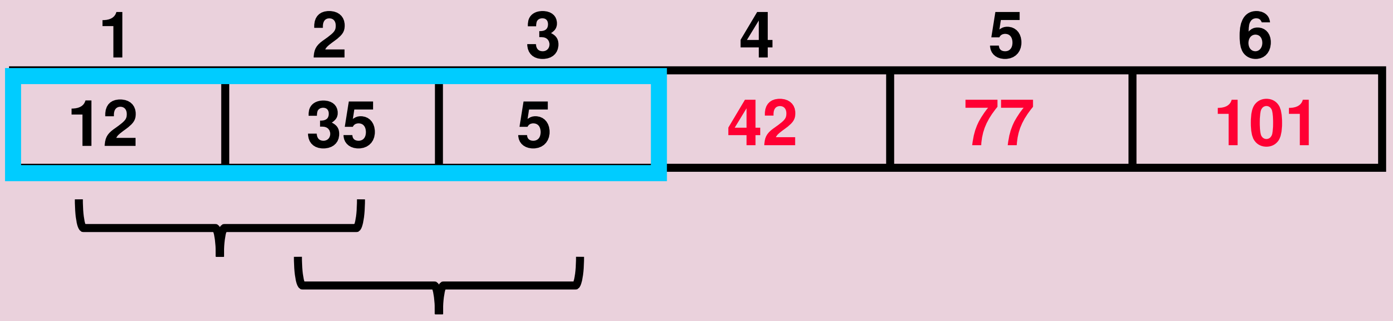
1	2	3	4	5	6
12	5	35	42	77	101

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

Reducing the Number of Comparisons

- On the Nth "bubble up", we only need to do MAX-N comparisons.

- For example:
  - This is the 4th "bubble up"
  - MAX is 6
  - Thus we have 2 comparisons to do



Already Sorted Collections?

- What if the collection was already sorted?
- What if only a few elements were out of place and after a couple of "bubble ups," the collection was sorted?

- We want to be able to detect this and "stop early"!

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

Using a Boolean “Flag”

- We can use a boolean variable to determine if any swapping occurred during the “bubble up.”

- If no swapping occurred, then we know that the collection is already sorted!

- This boolean “flag” needs to be reset after each “bubble up.”

Bubble Sort Algorithm

```
bubbleSort(list,n){
  for i (1 to n-1){
    swapped = False
    for j (0 to n - i - 1){
      if list[j] > list[j+1]{
        t = list[j]
        list[j] = list[j+1]
        list[j+1] = t
        swapped = True } }
    if not swapped:
      break } }
```

Inner loop

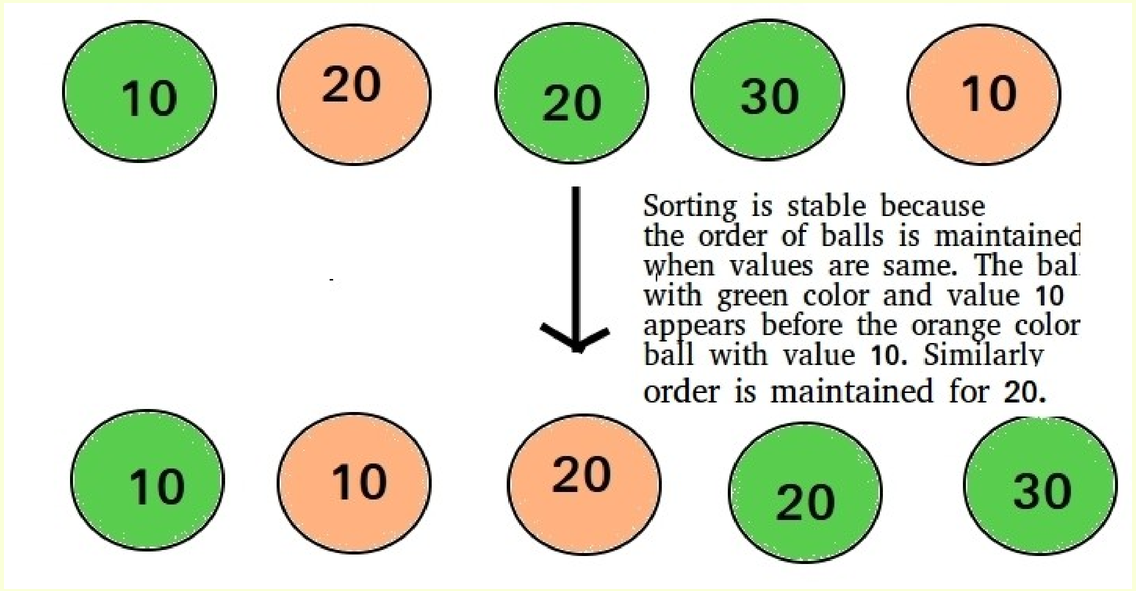
Outer loop

Time Complexity

Worst Case: O(n^2)

PROPERTIES

- Bubble sort is stable and in-place algorithm.
- **In-place** means that the input and output occupy the same memory
- **Stable** means the order of input elements is unchanged except where change is required to satisfy the requirements.



Sorting is stable because the order of balls is maintained when values are same. The ball with green color and value 10 appears before the orange color ball with value 10. Similarly order is maintained for 20.

Summary

- “Bubble Up” algorithm will move largest value to its correct location (to the right)
- Repeat “Bubble Up” until all elements are correctly placed:
  - Maximum of N-1 times
  - Can finish early if no swapping occurs
- We reduce the number of elements we compare each time one is correctly placed

An Animated Example

N	8
to_do	7
index	

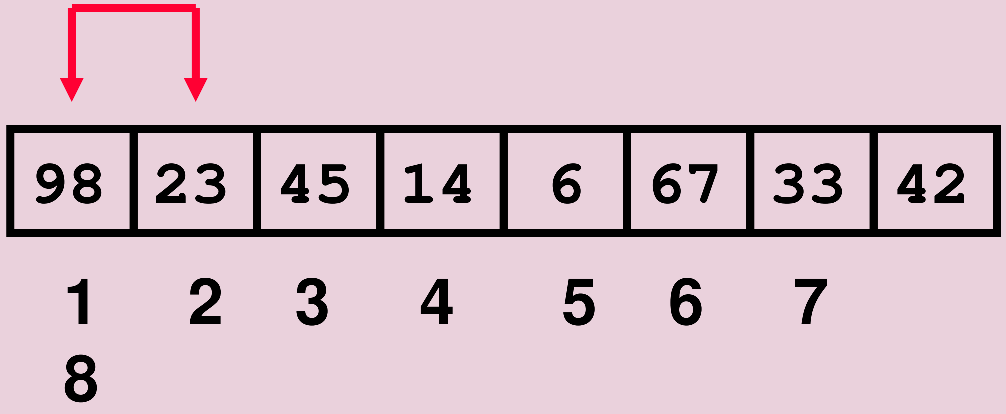
did_swap	true

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

An Animated Example

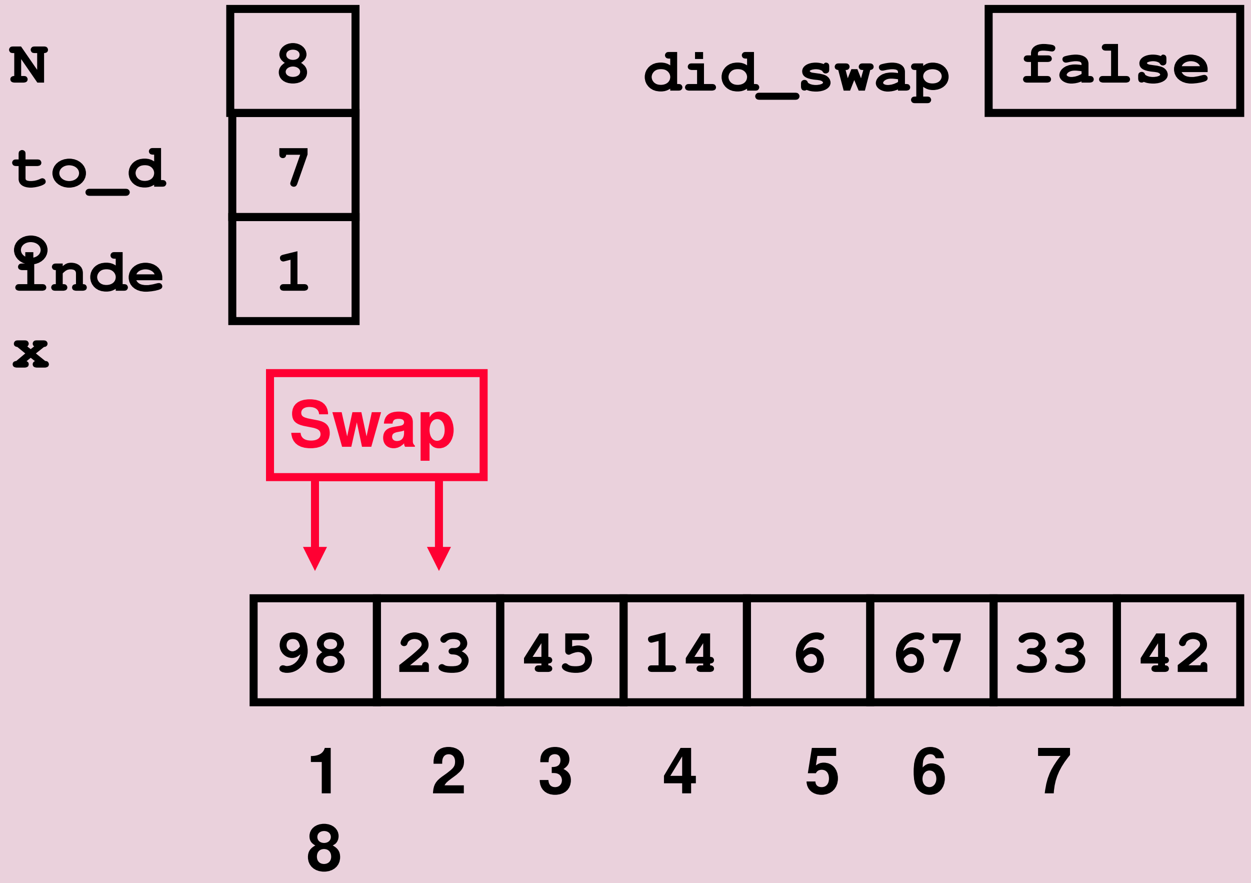
N	8
to_do	7
index	1

did_swap	false

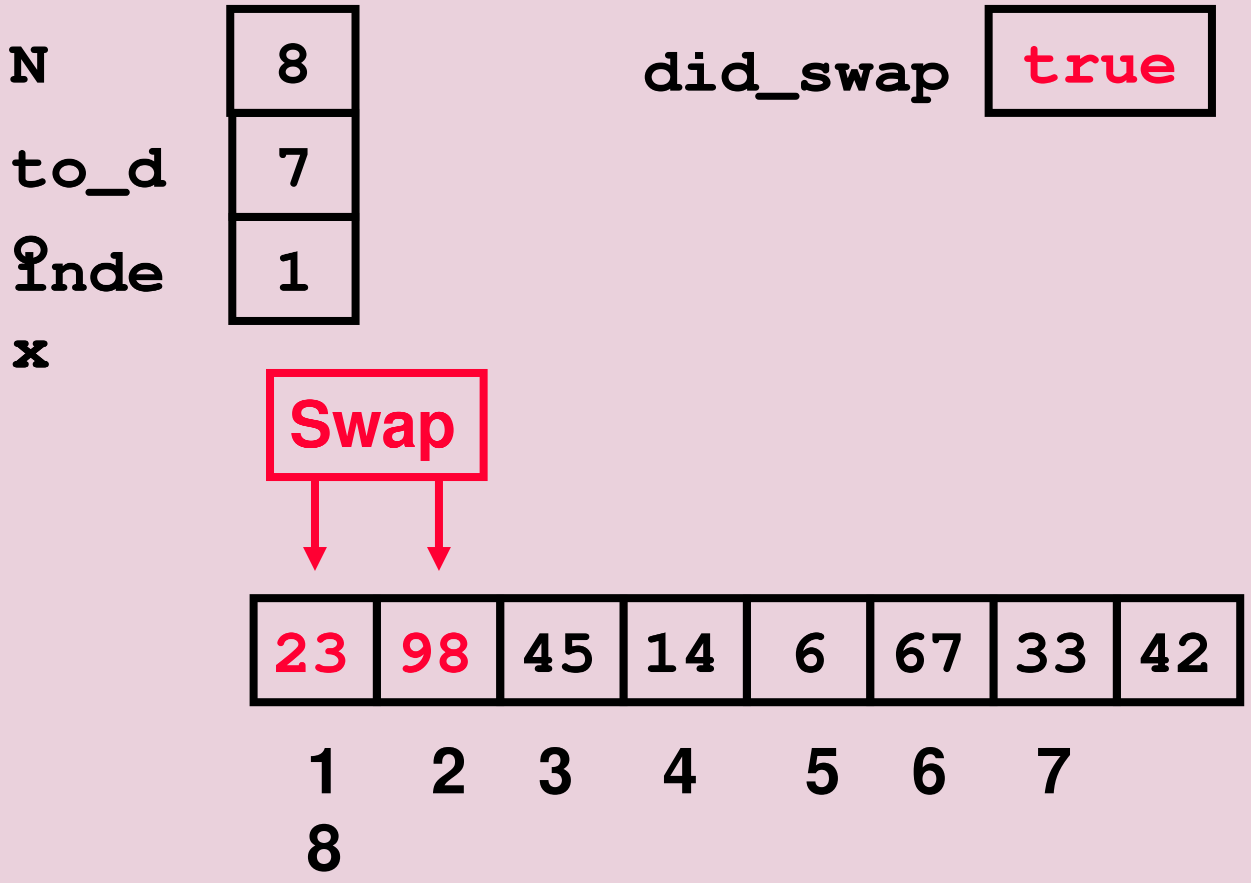


98	23	45	14	6	67	33	42
1	2	3	4	5	6	7	8

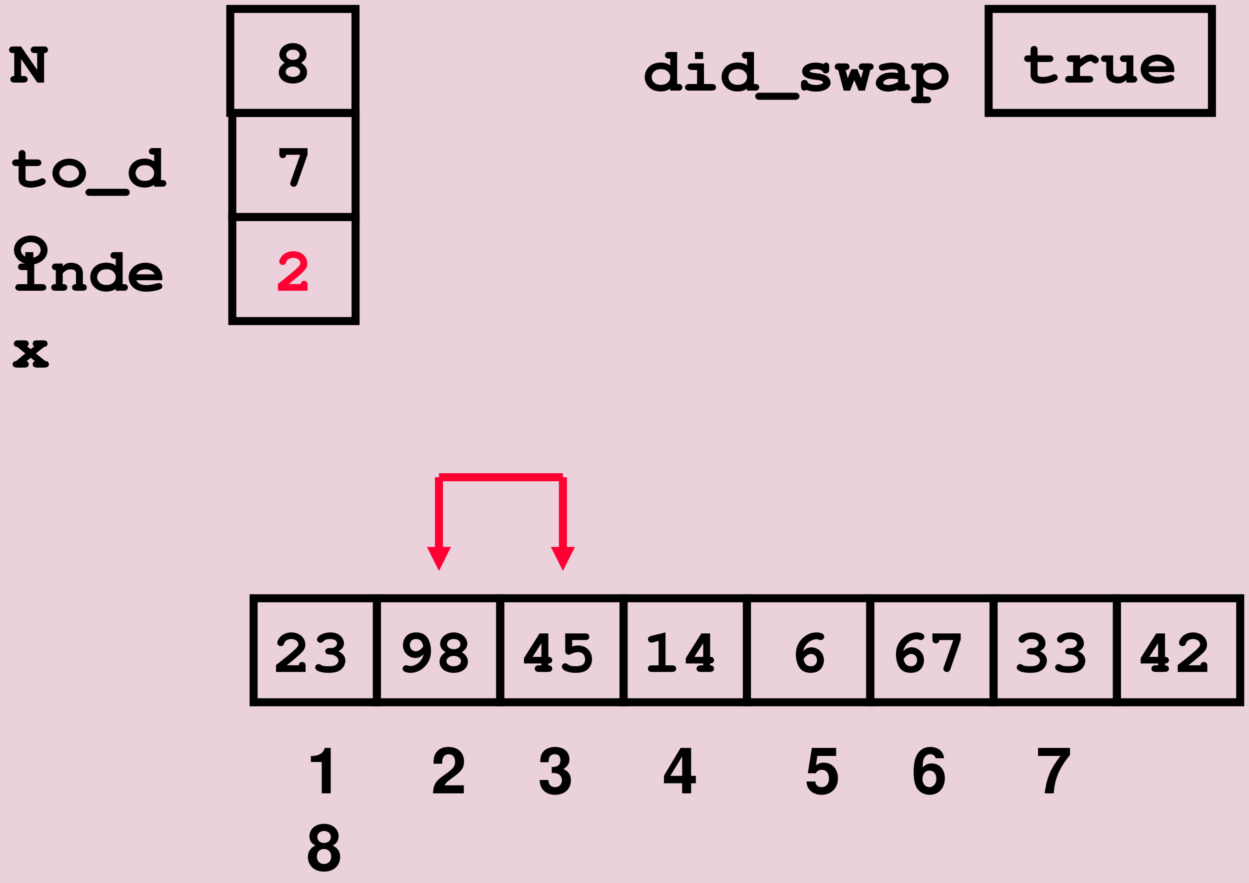
An Animated Example



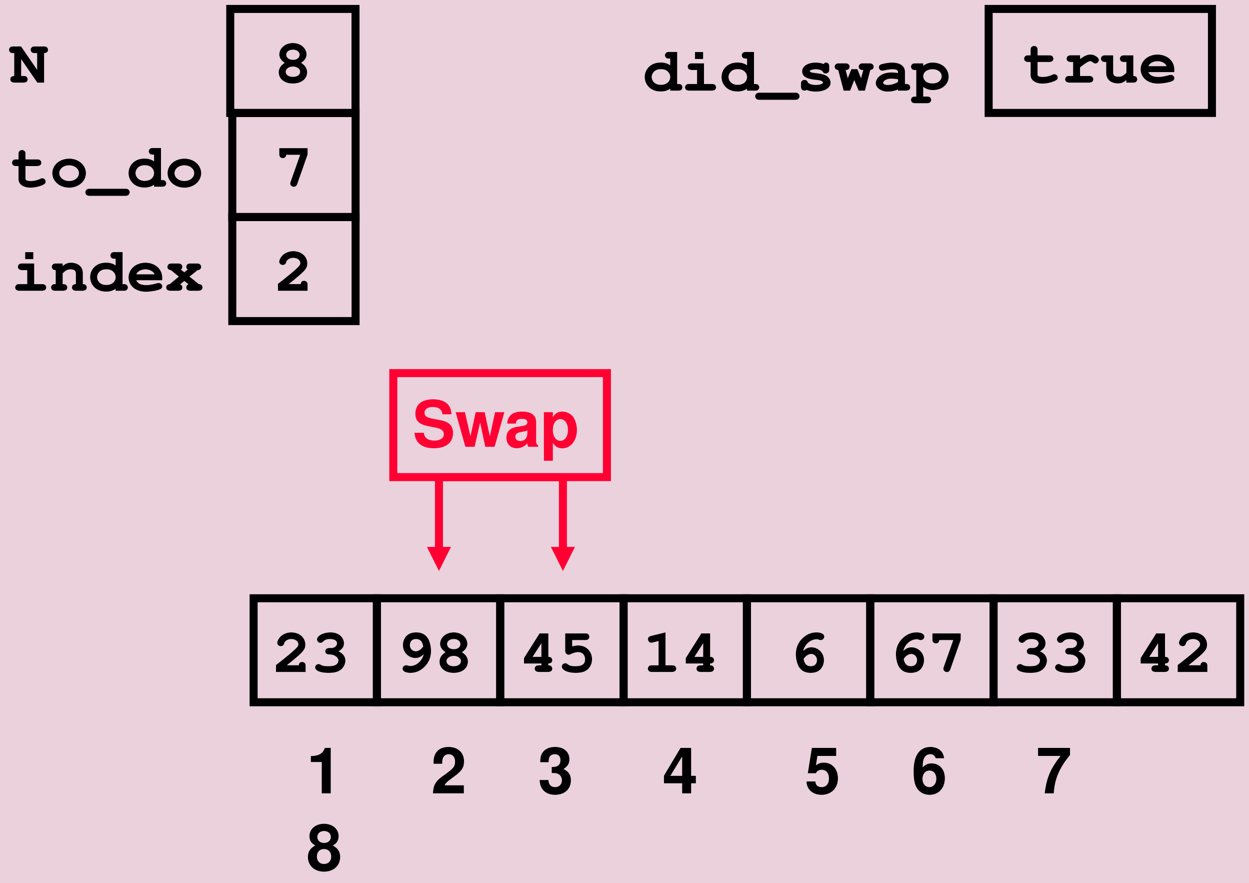
An Animated Example



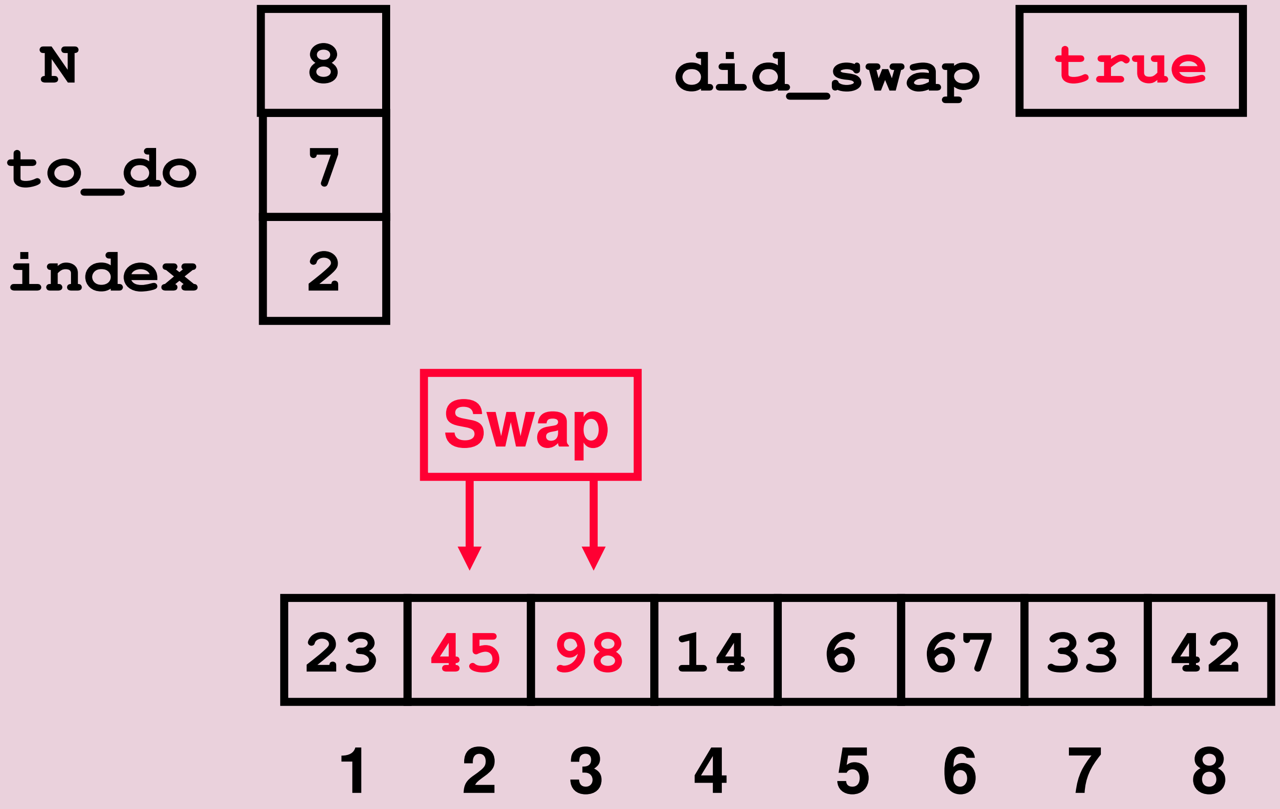
An Animated Example



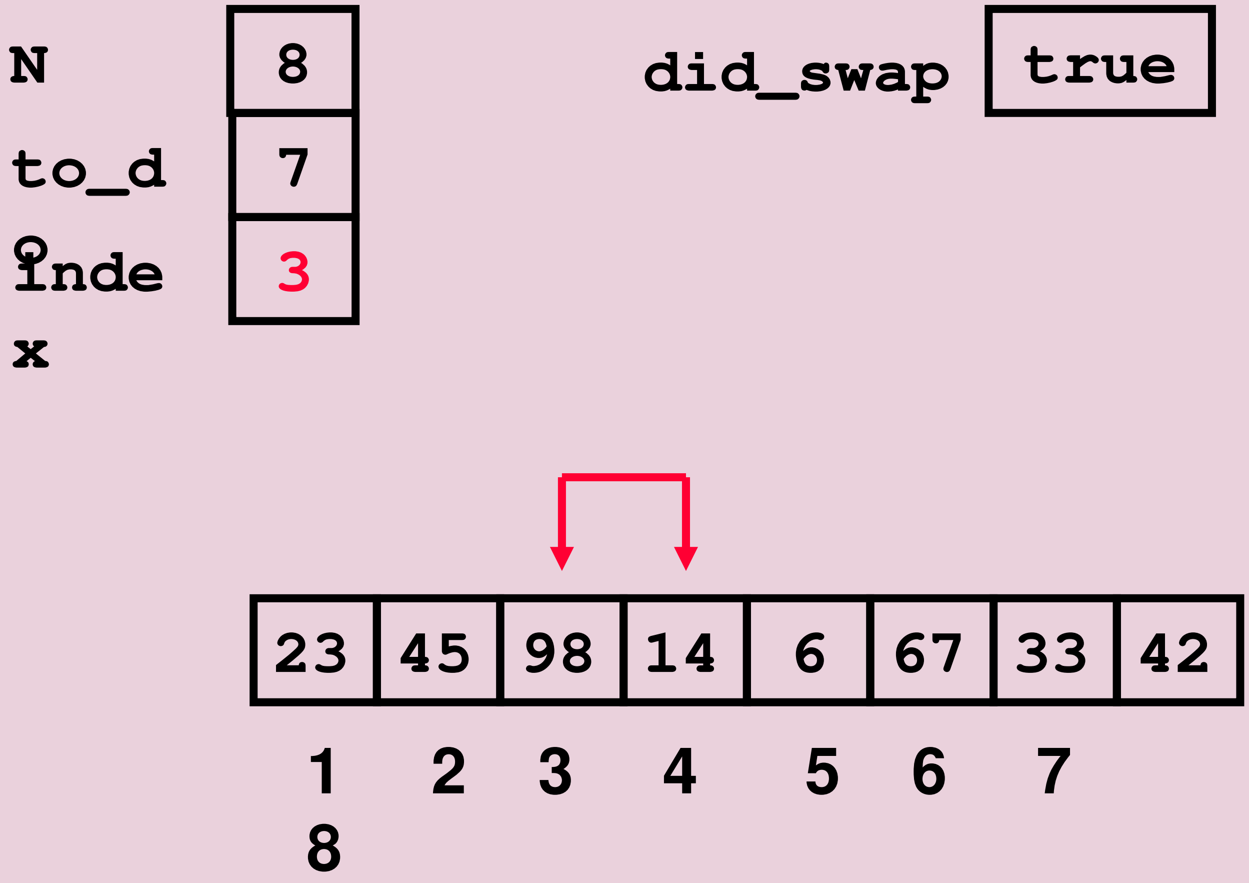
An Animated Example



An Animated Example



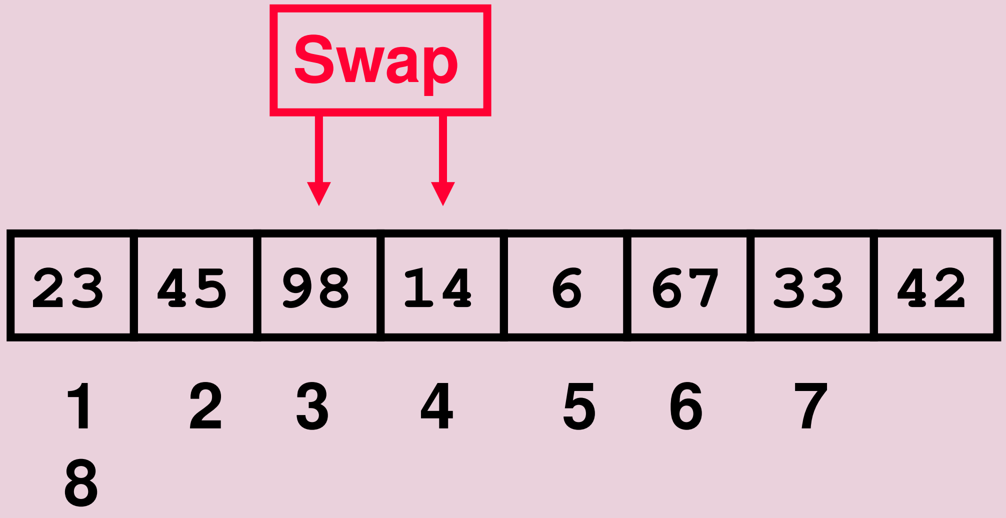
An Animated Example



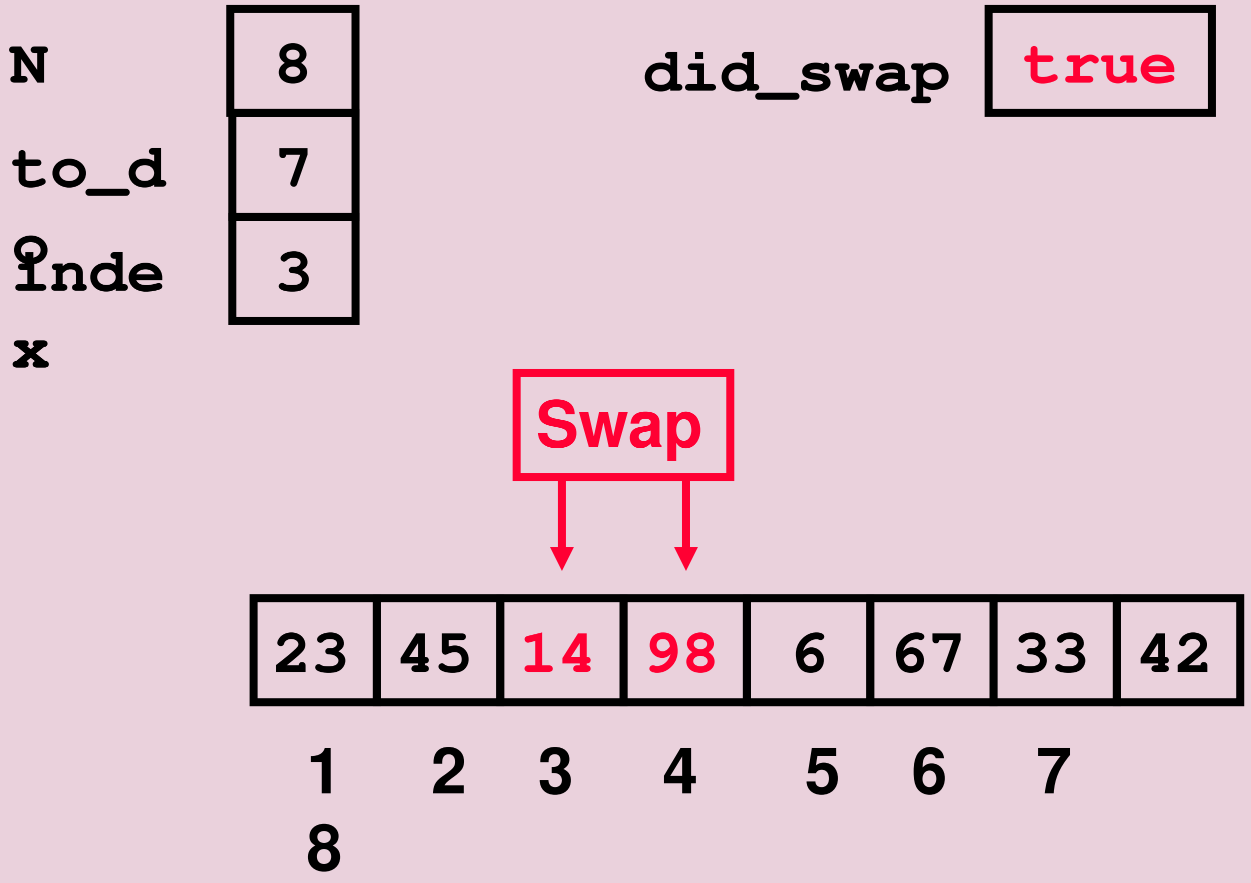
An Animated Example

N	8
to_do	7
index	3

did_swap	true



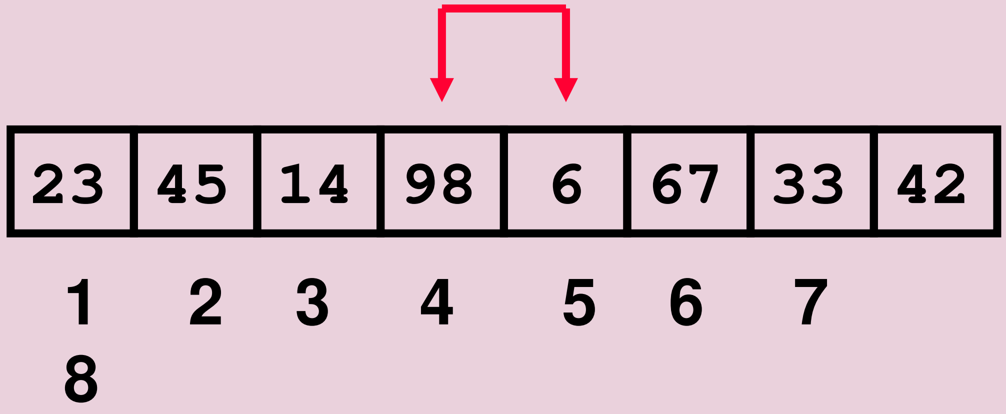
An Animated Example



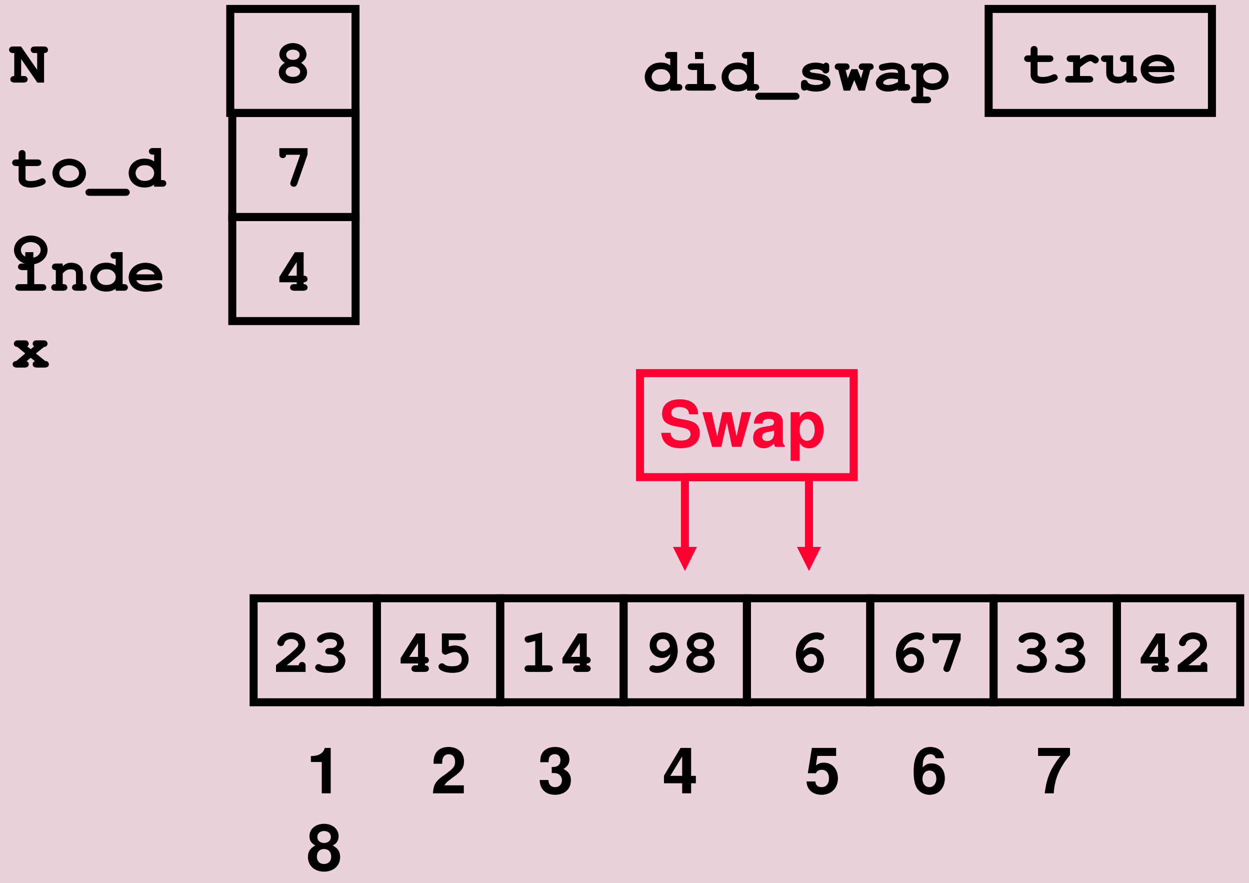
An Animated Example

N	8
to_d	7
inde x	4

did_swap true



An Animated Example



An Animated Example

N	8
to_d	7
inde x	4

did_swap `true`

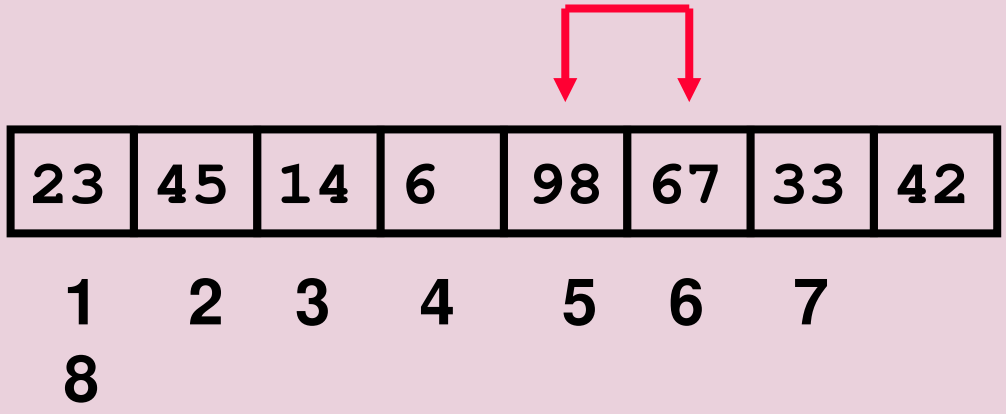
Swap

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

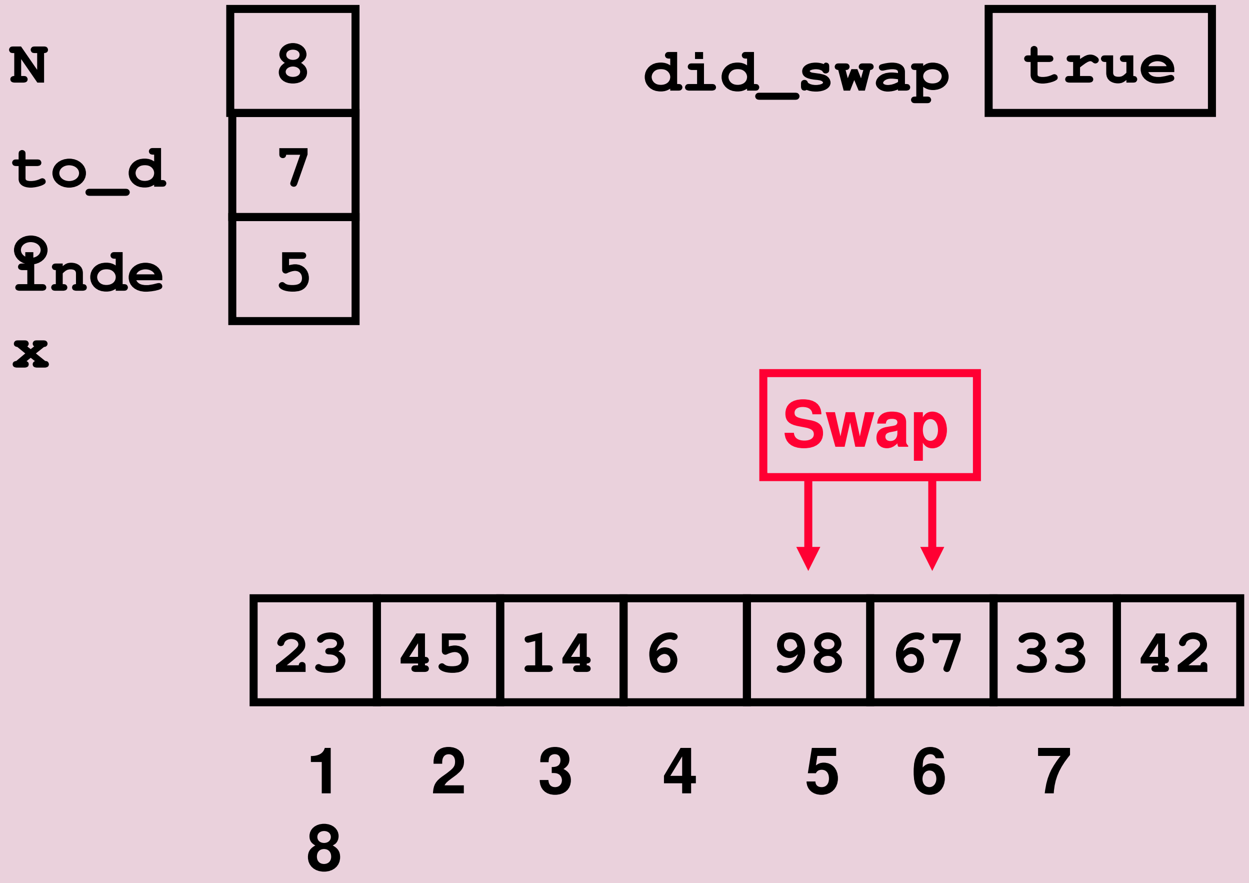
An Animated Example

N	8
to_d	7
inde x	5

did_swap **true**



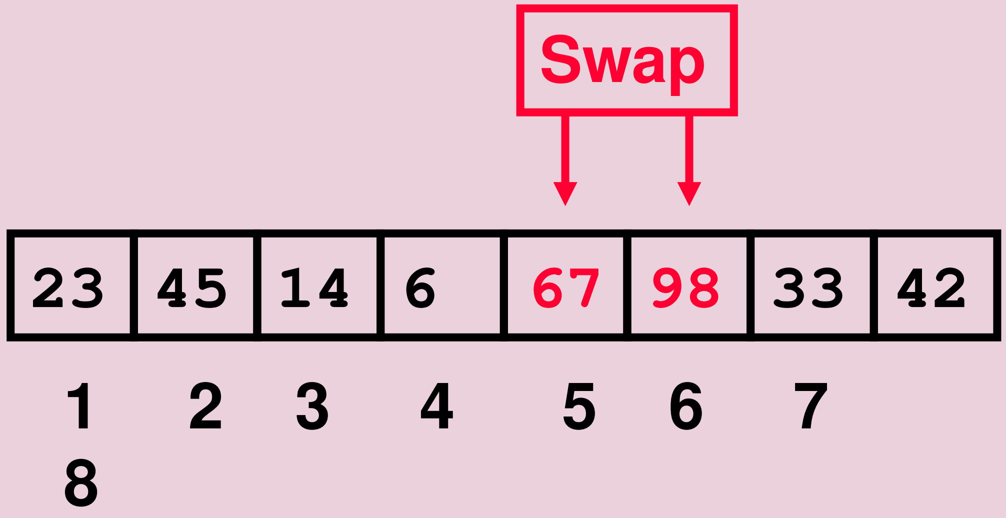
An Animated Example



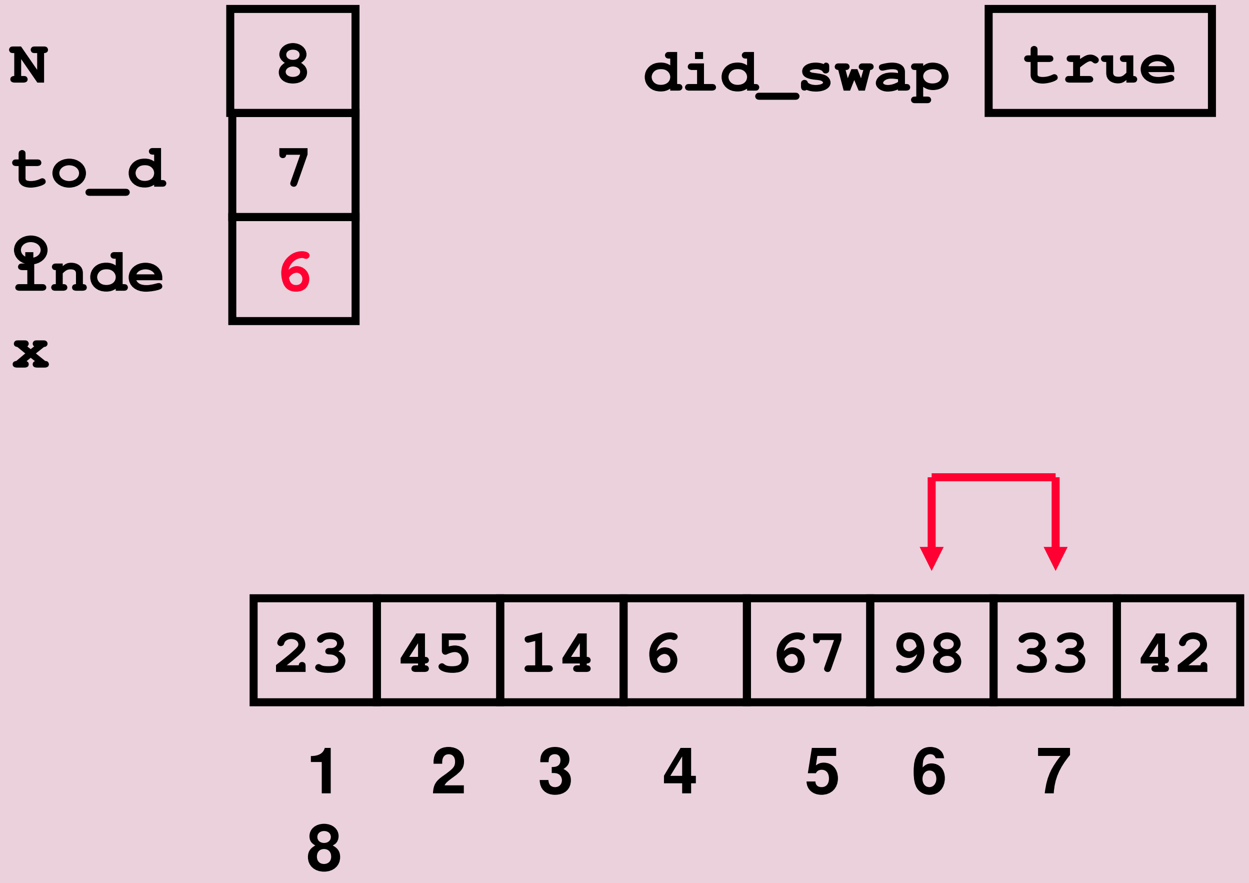
An Animated Example

N	8
to_d	7
inde x	5

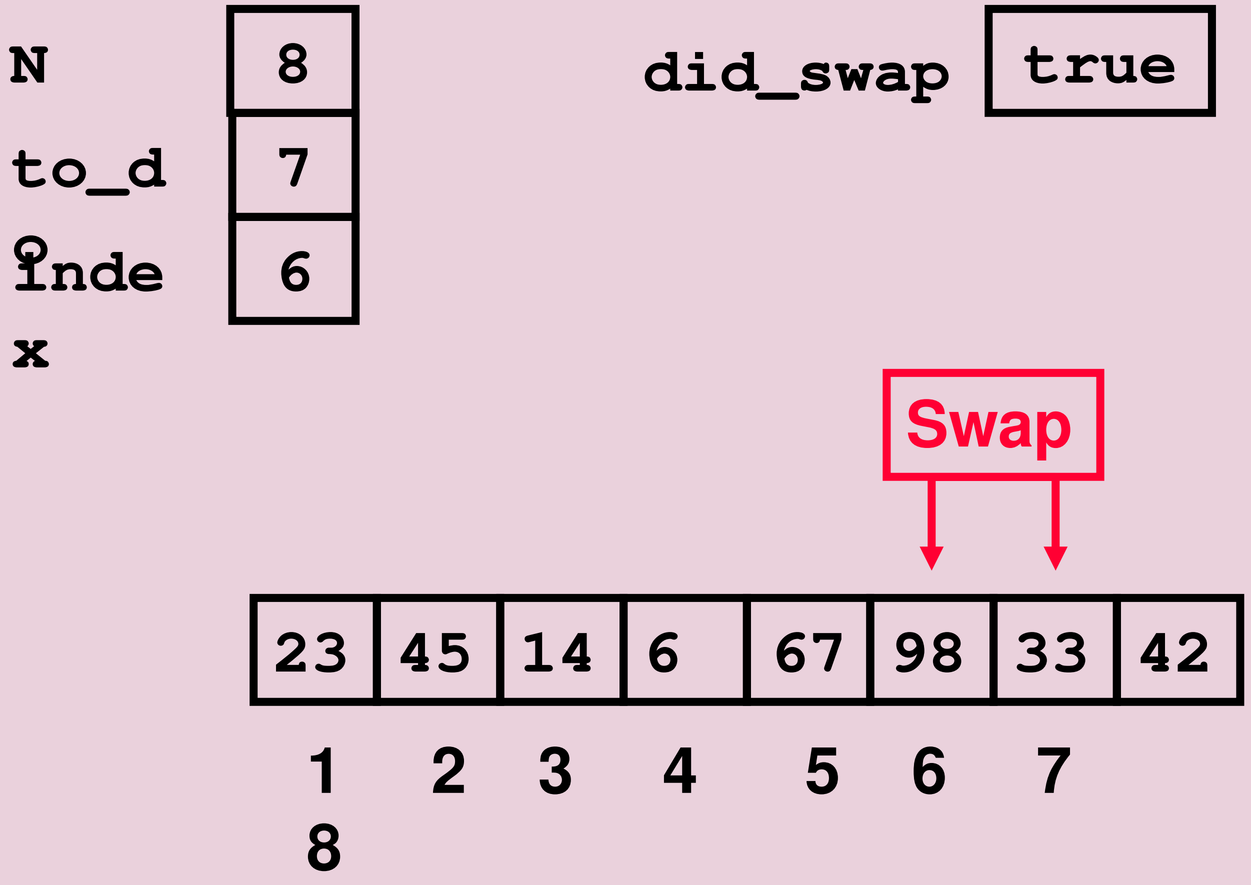
did_swap **true**



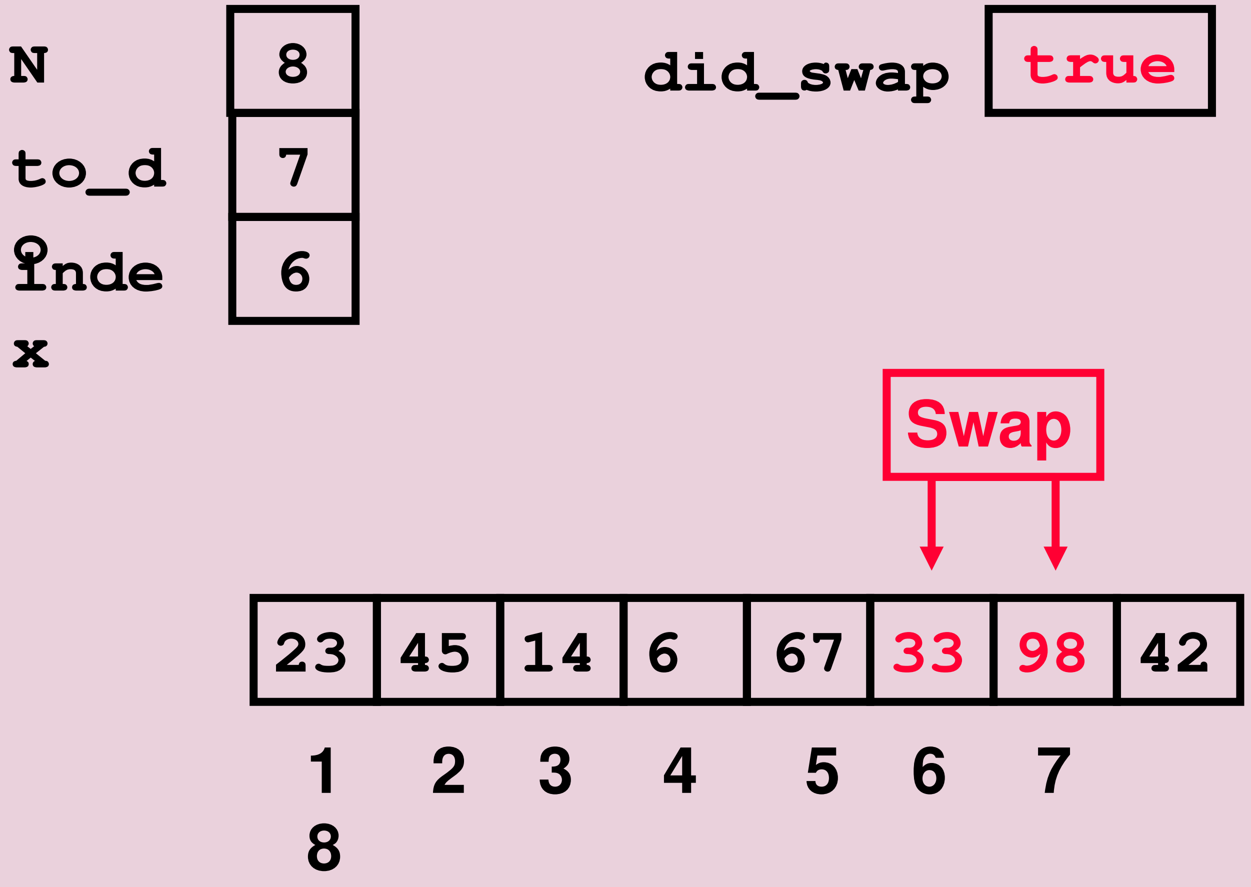
An Animated Example



An Animated Example



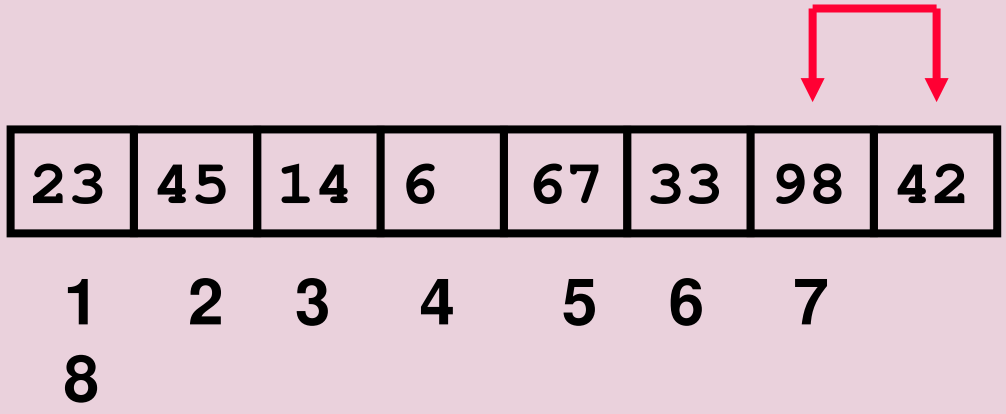
An Animated Example



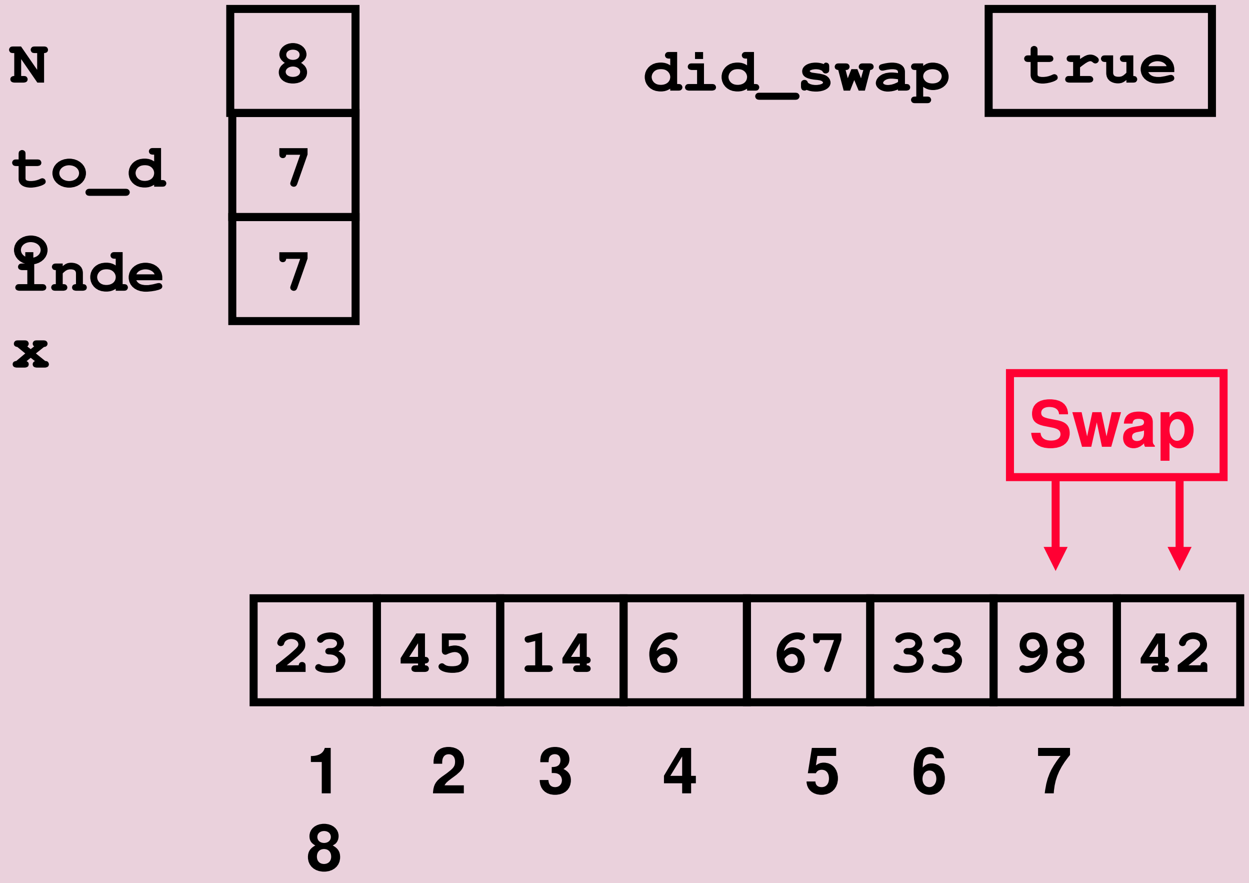
An Animated Example

N	8
to_d	7
inde x	7

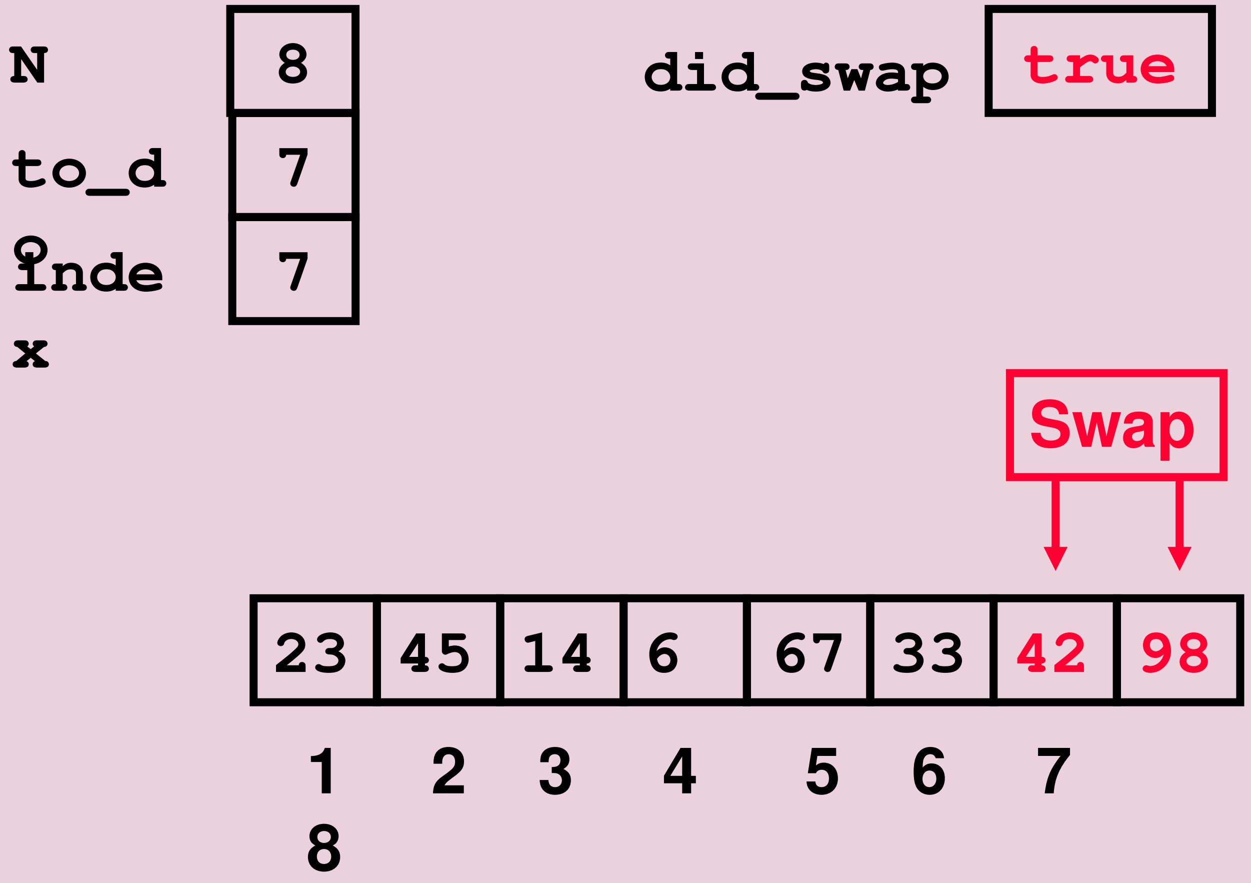
did_swap **true**



An Animated Example



An Animated Example



After First Pass of Outer Loop

N	8
to_d	7
index	8

did_swap `true`

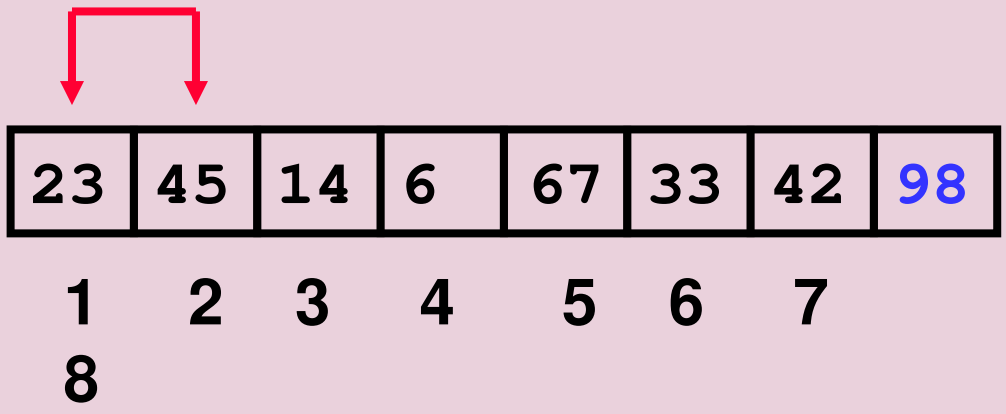
Finished first “Bubble Up”

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

The Second “Bubble Up”

N	8
to_do	6
index	1

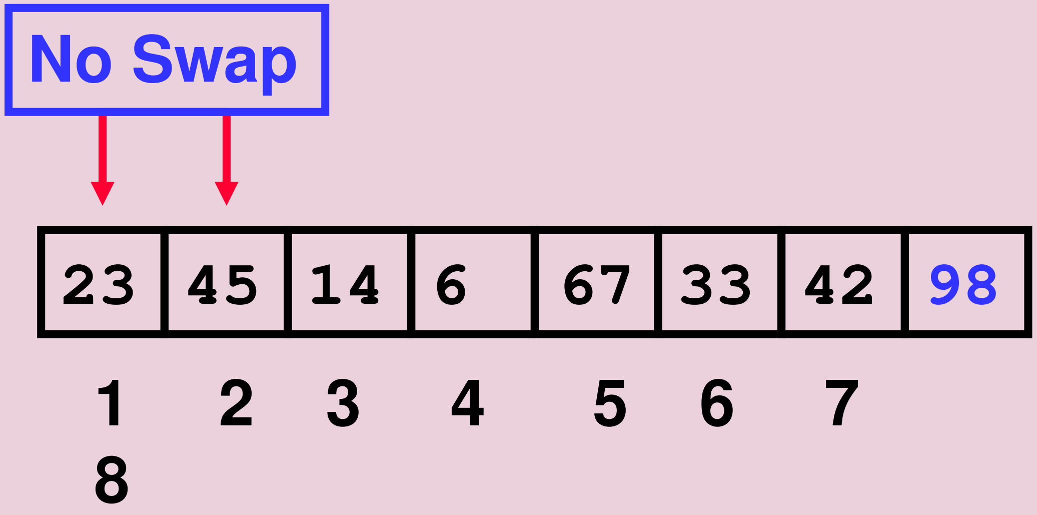
did_swap false



The Second “Bubble Up”

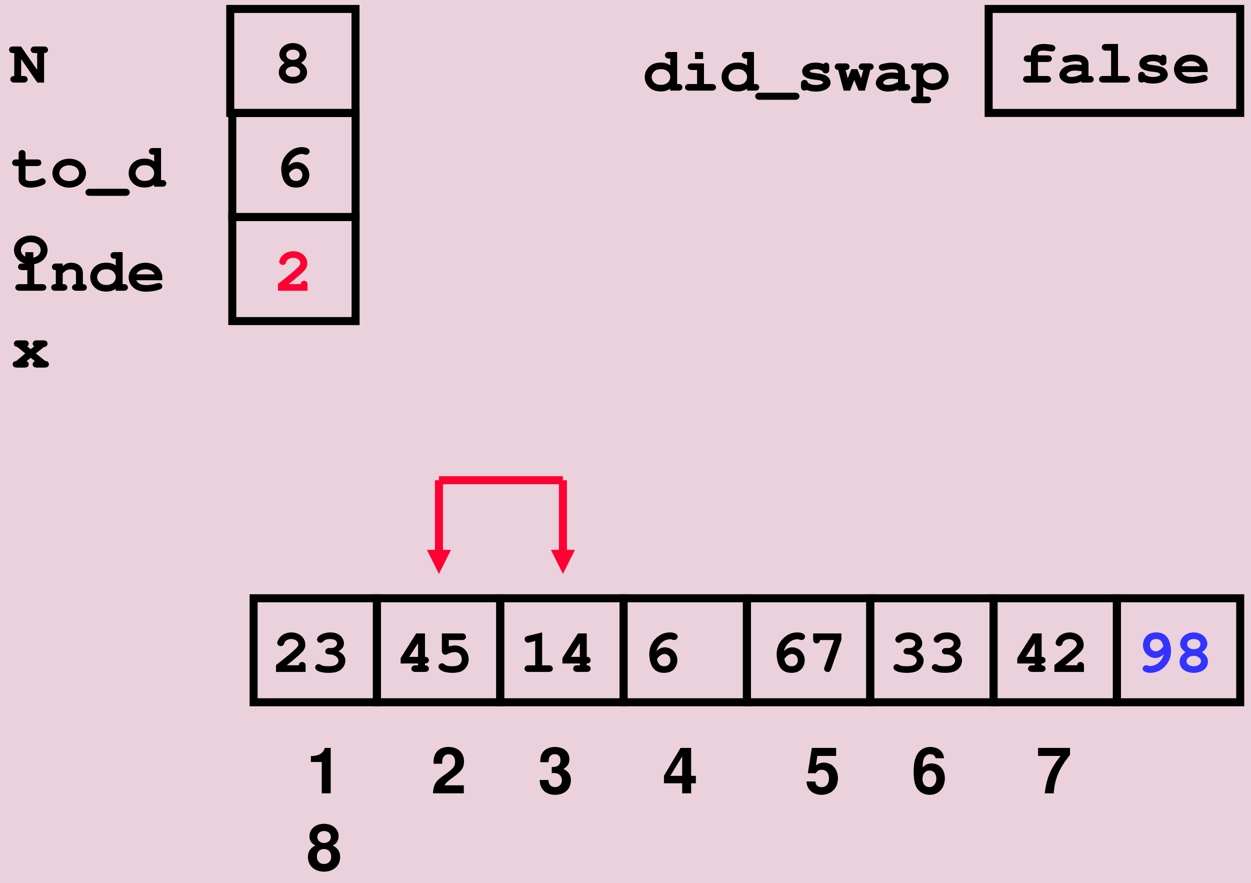
N	8
to_d	6
index	1

did_swap `false`



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

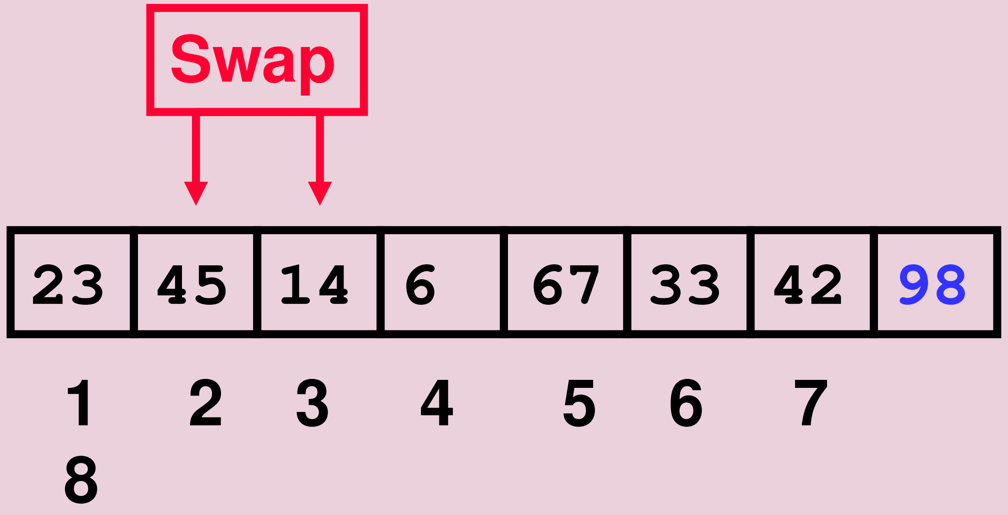
The Second "Bubble Up"



The Second “Bubble Up”

`N`	`8`
`to_do`	`6`
`index`	`2`

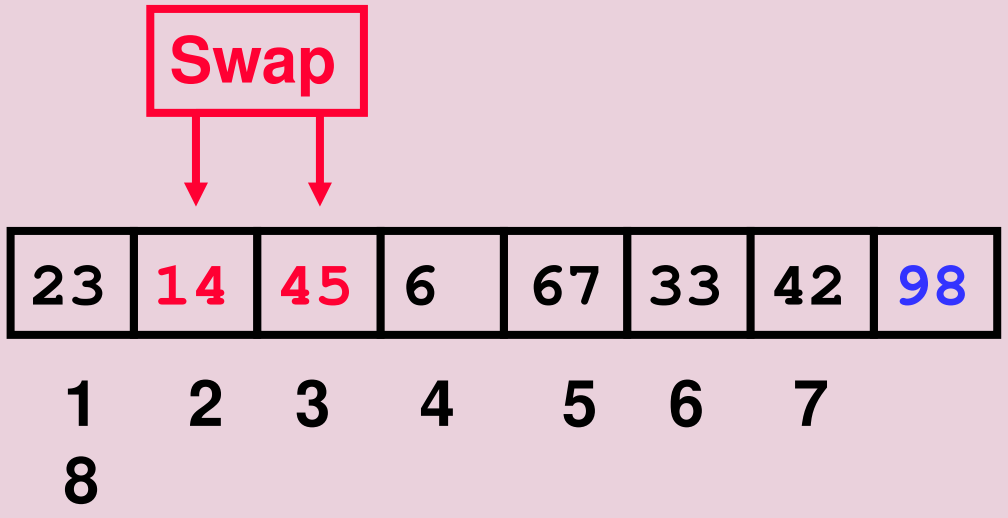
`did_swap` `false`



The Second "Bubble Up"

`N`	8
`to_do`	6
`index`	2

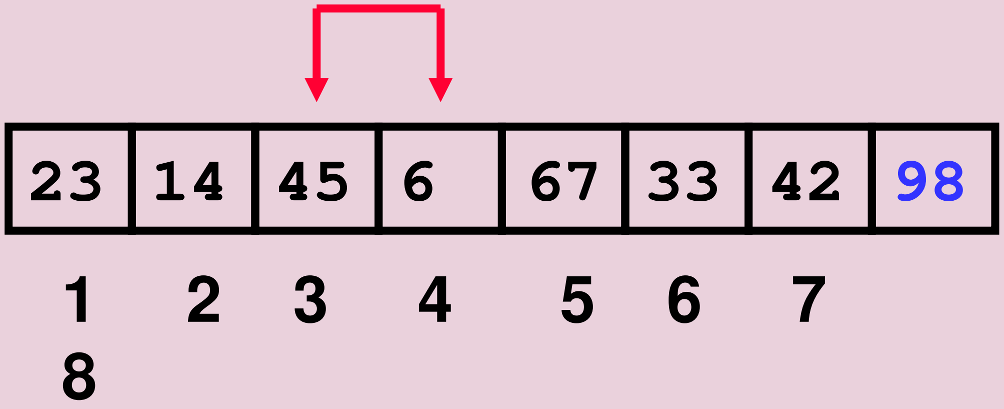
`did_swap` **true**



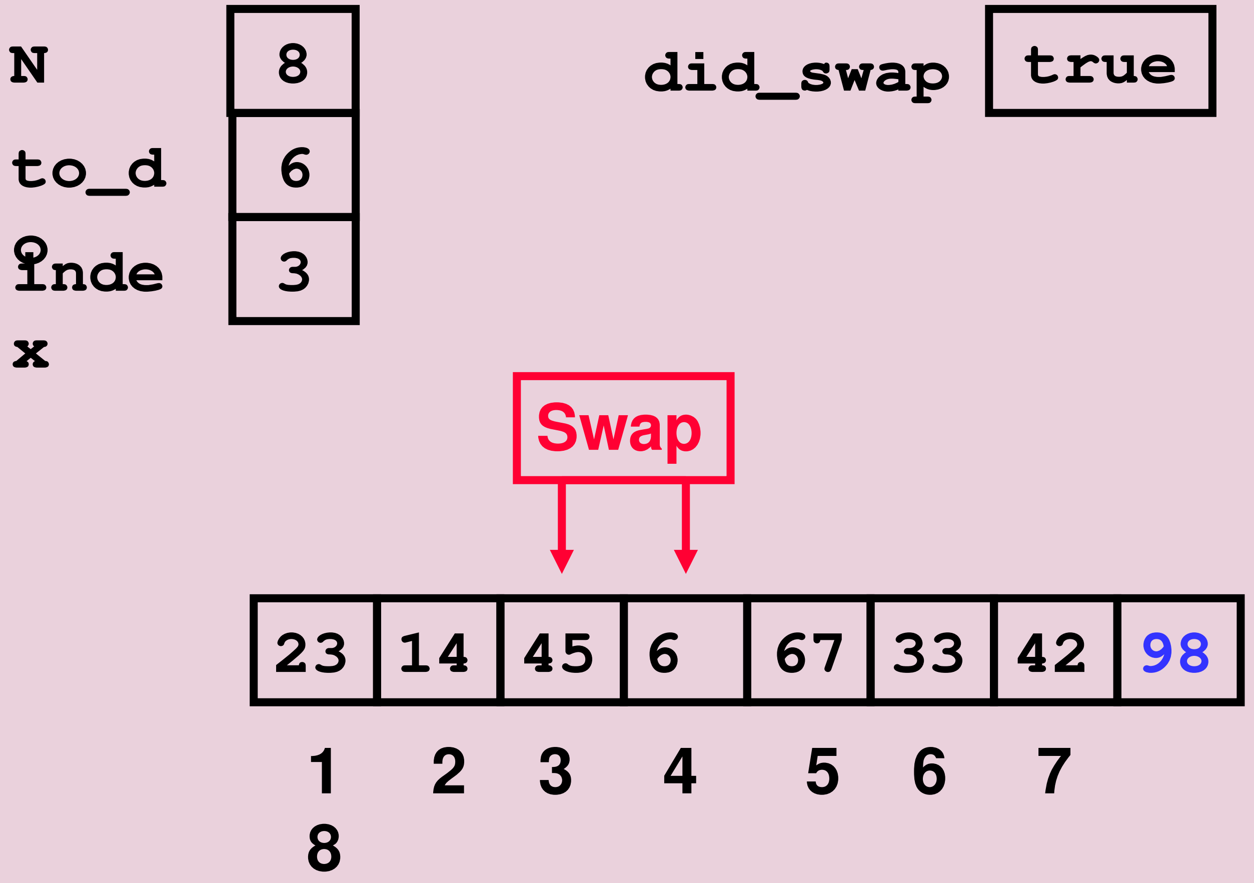
The Second “Bubble Up”

N	8
to_do	6
index	3

did_swap `true`



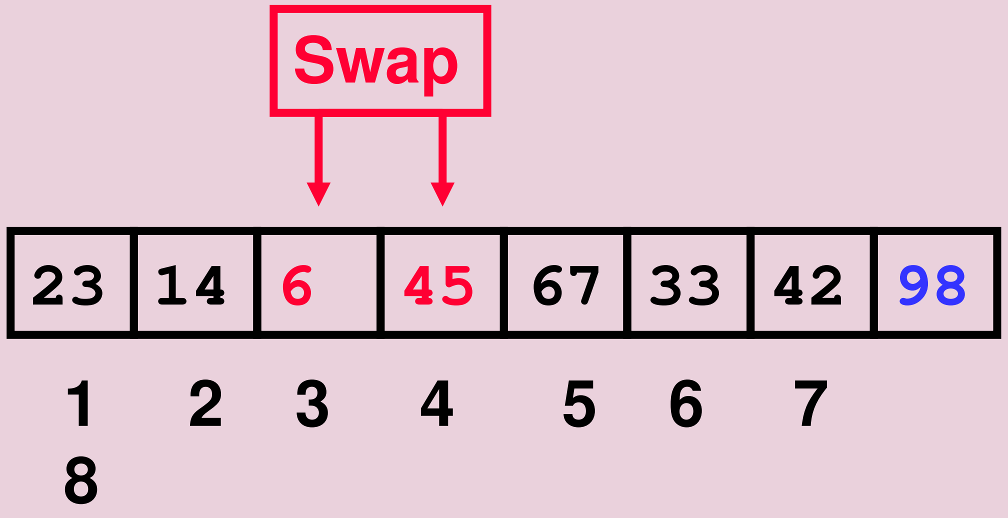
The Second “Bubble Up”



The Second “Bubble Up”

N	8
to_do	6
index	3

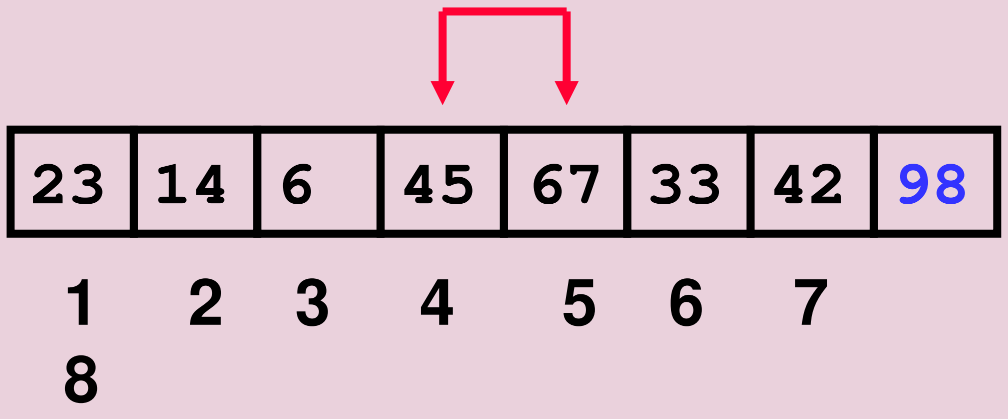
did_swap	true



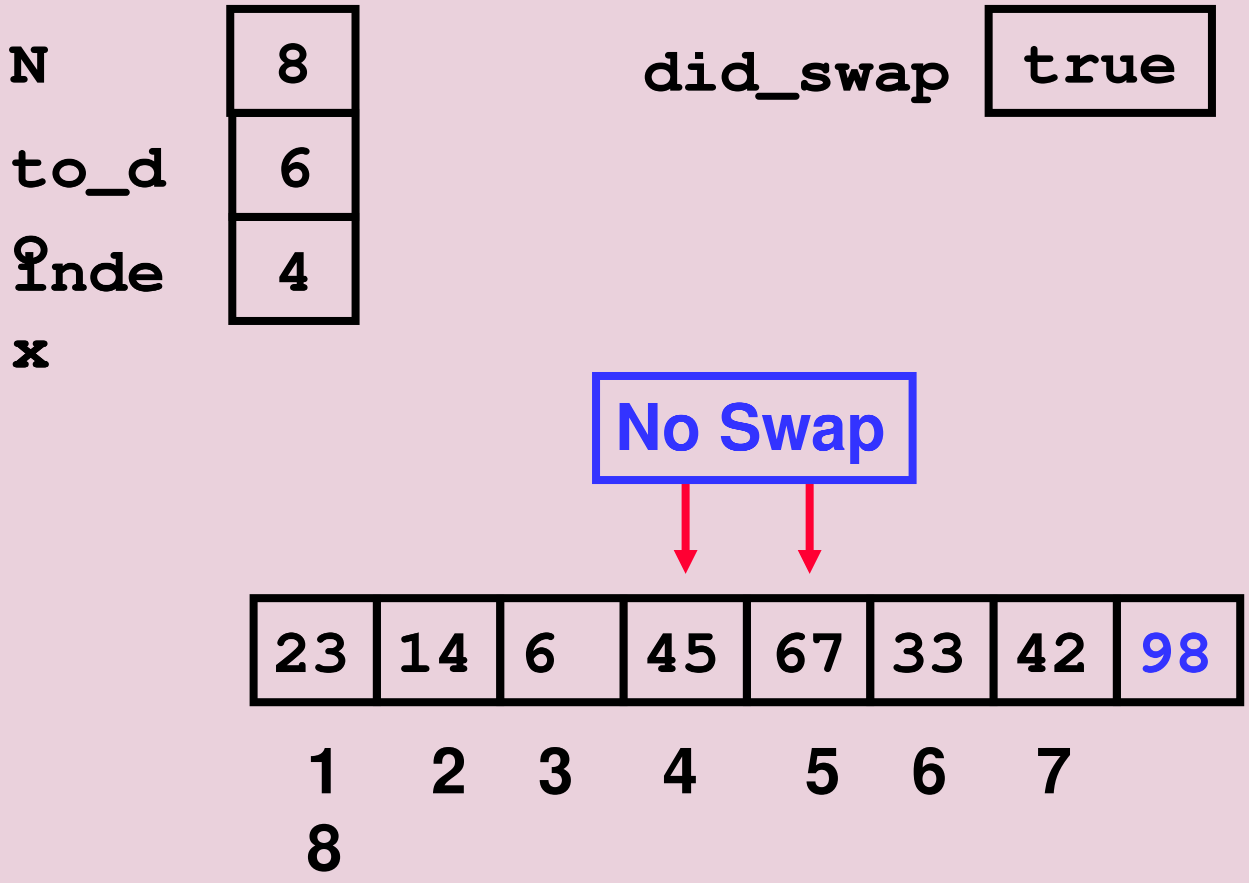
The Second "Bubble Up"

N	8
to_do	6
index	4

did_swap	true



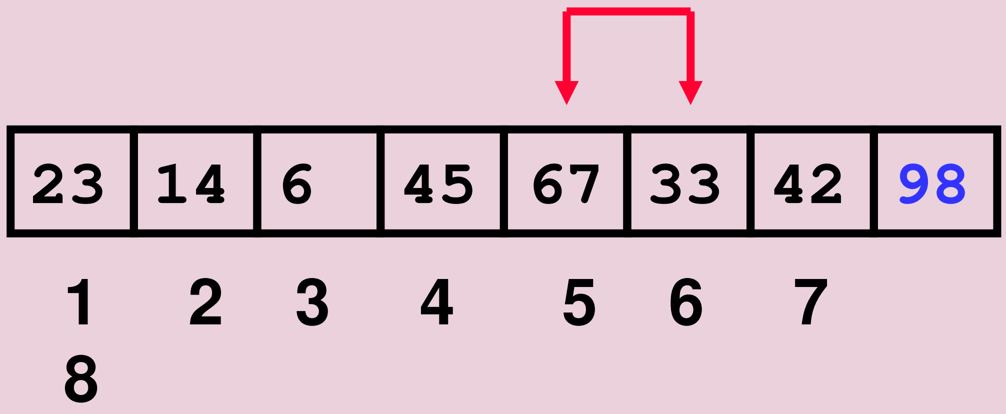
The Second “Bubble Up”



The Second “Bubble Up”

N	8
to_do	6
index	5

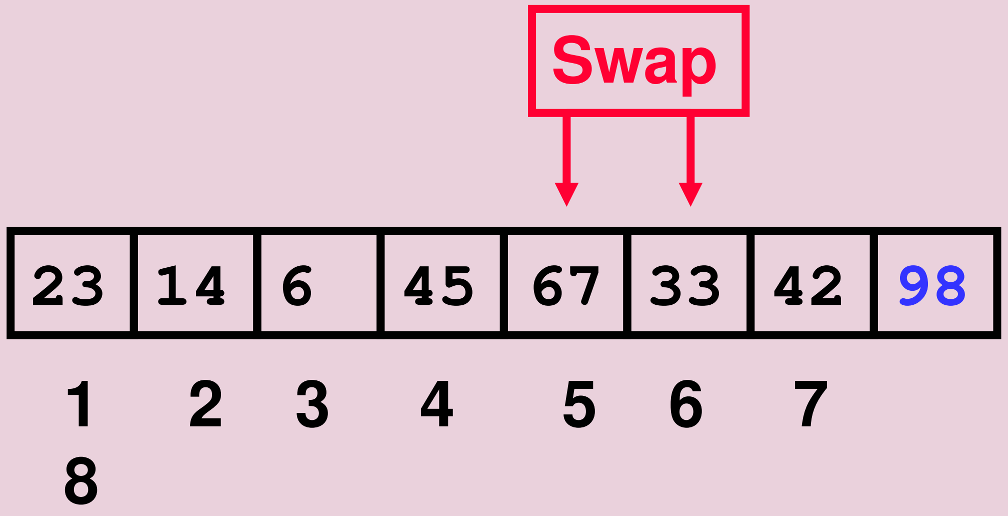
did_swap `true`



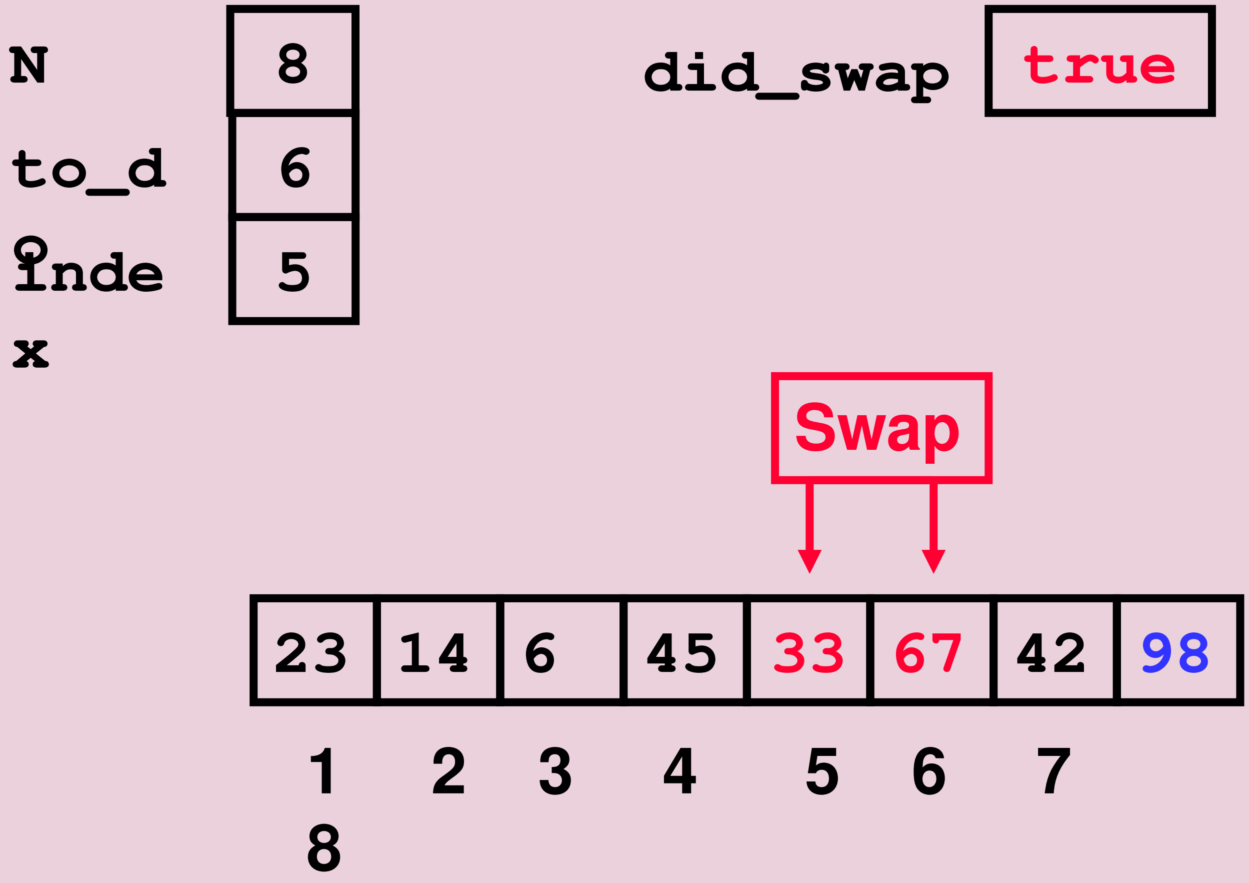
The Second “Bubble Up”

N	8
to_do	6
index	5

did_swap true



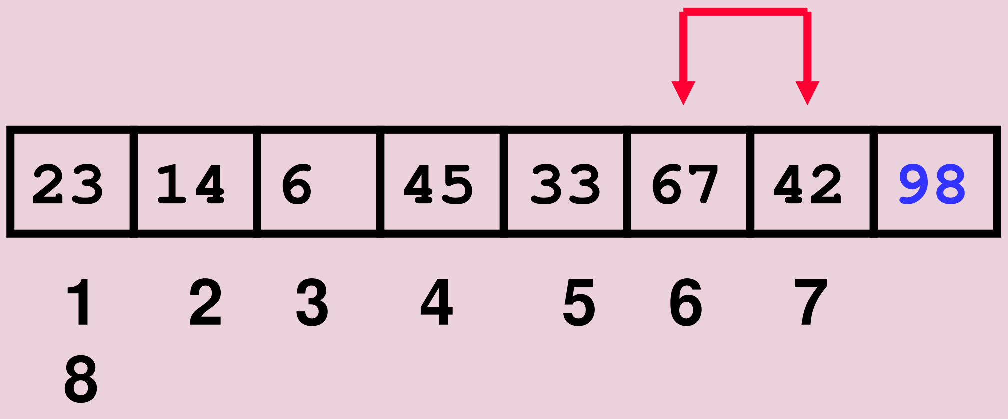
The Second “Bubble Up”



The Second “Bubble Up”

N	8
to_d	6
inde x	6

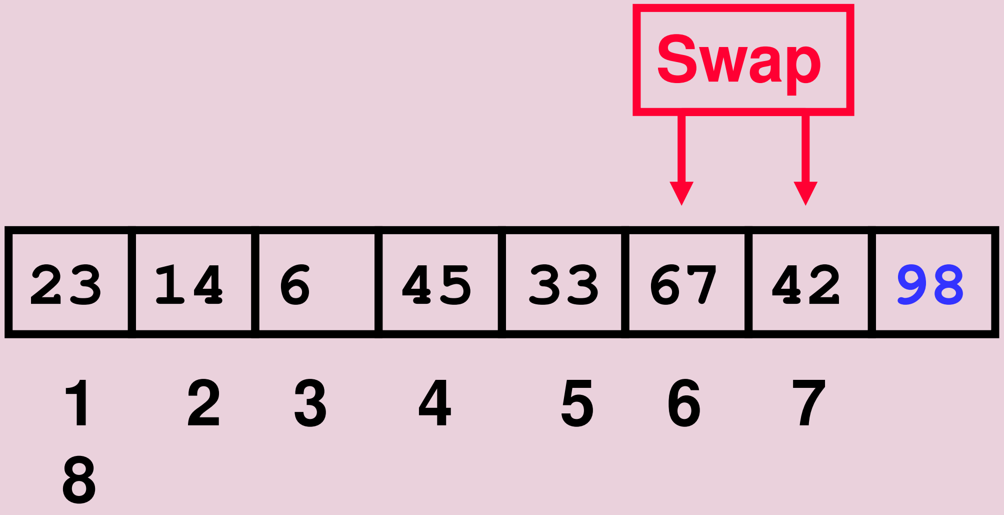
did_swap	true



The Second "Bubble Up"

N	8
to_d o	6
inde x	6

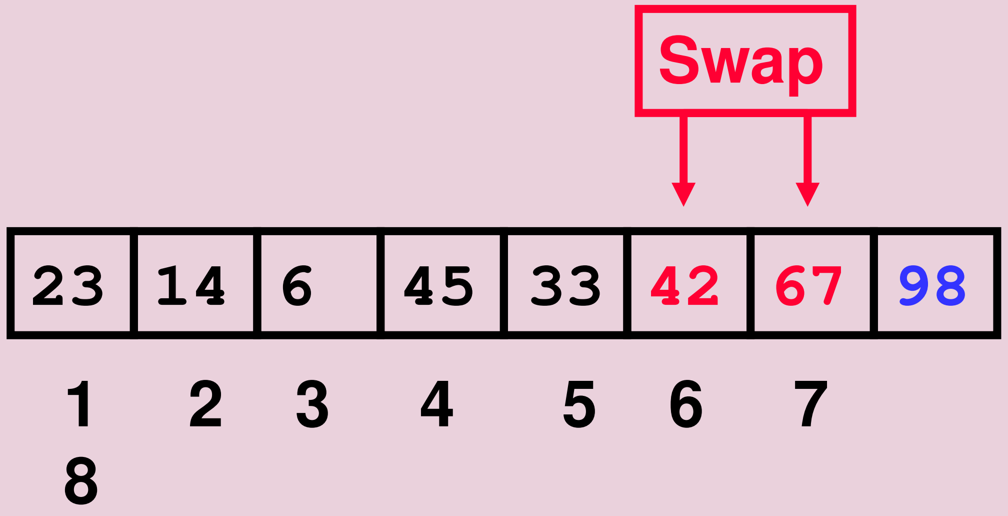
did_swap `true`



The Second “Bubble Up”

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

`did_swap` `true`

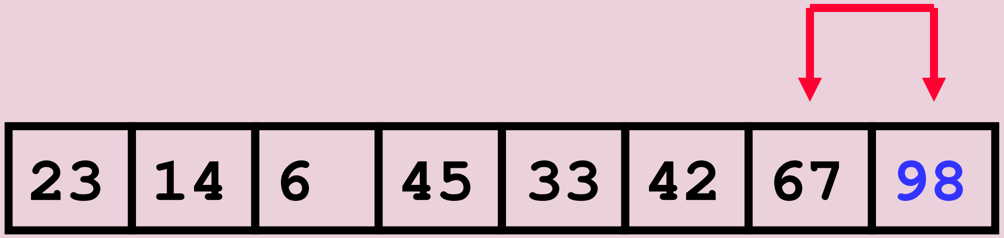


After Second Pass of Outer Loop

N	8
to_do	6
index	7

did_swap `true`

Finished second "Bubble Up"

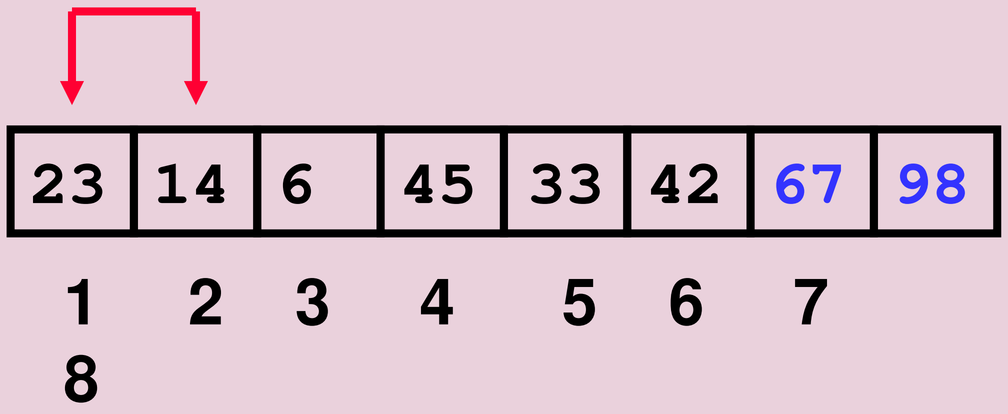


23	14	6	45	33	42	67	98
1	2	3	4	5	6	7	8

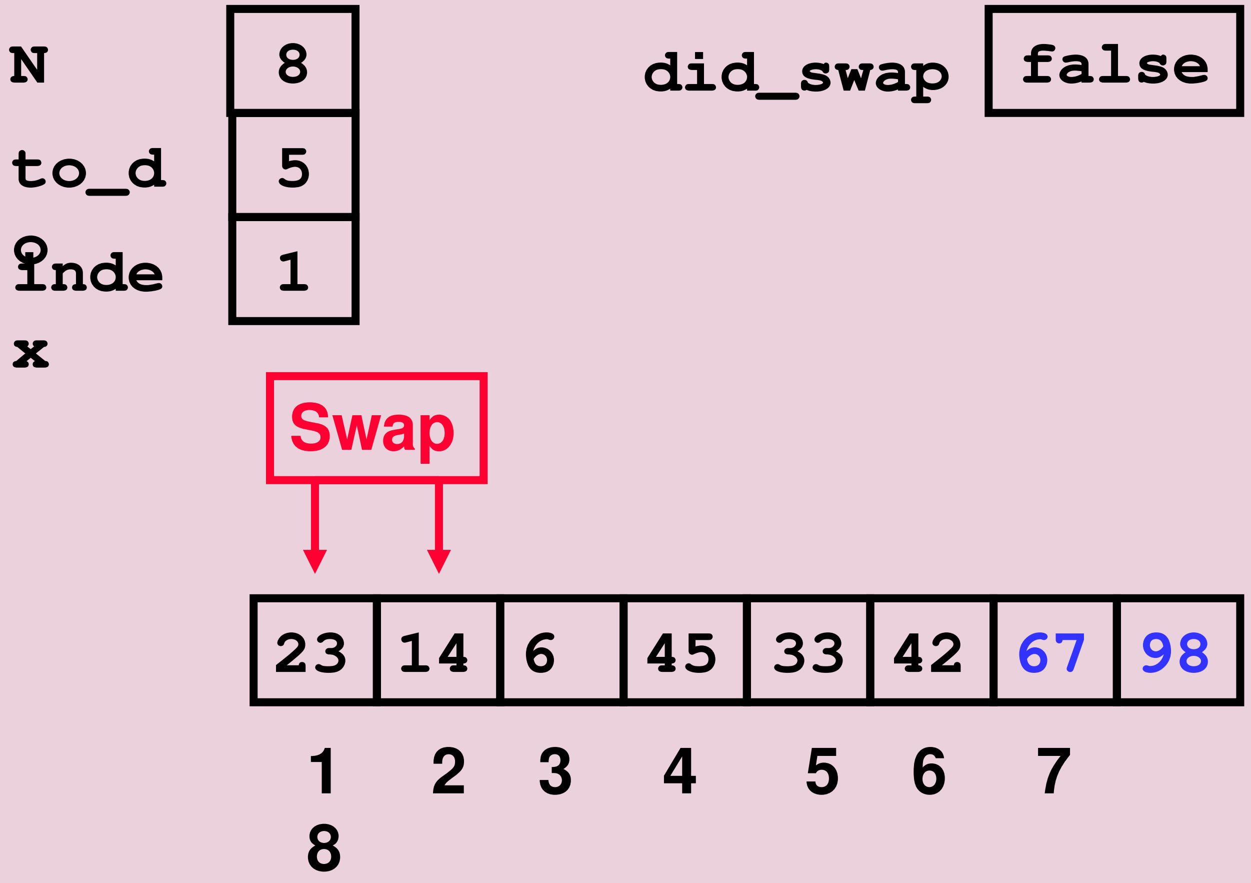
The Third “Bubble Up”

N	8
to_d	5
inde x	1

did_swap false



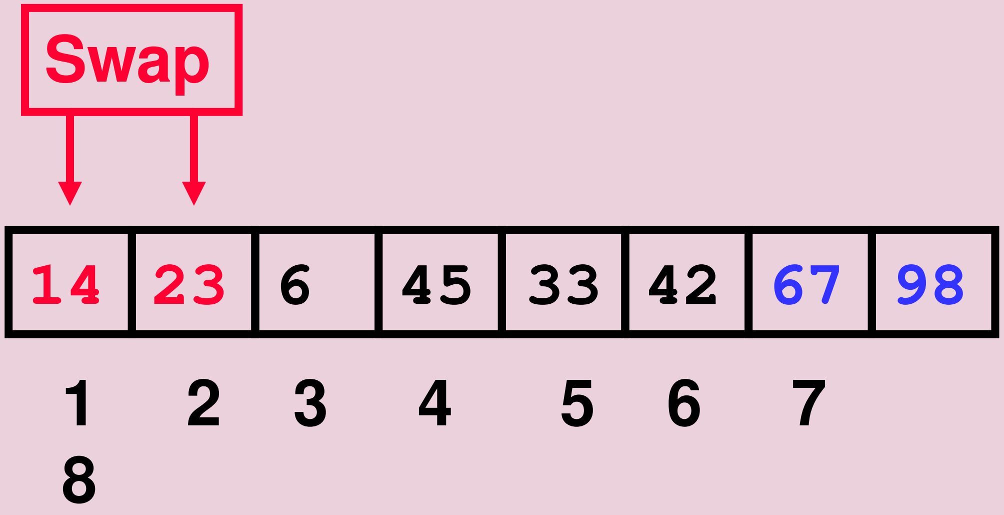
The Third “Bubble Up”



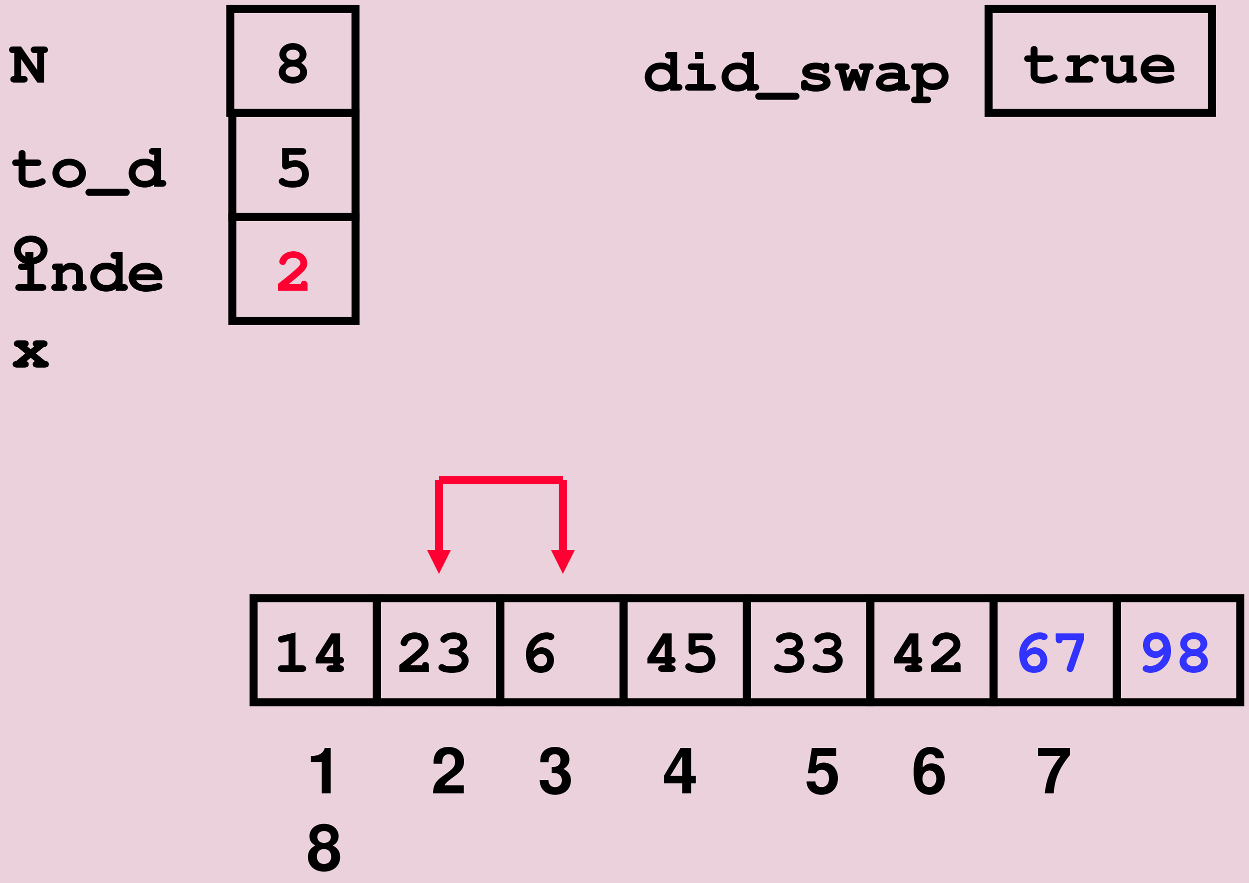
The Third “Bubble Up”

N	8
to_d	5
inde x	1

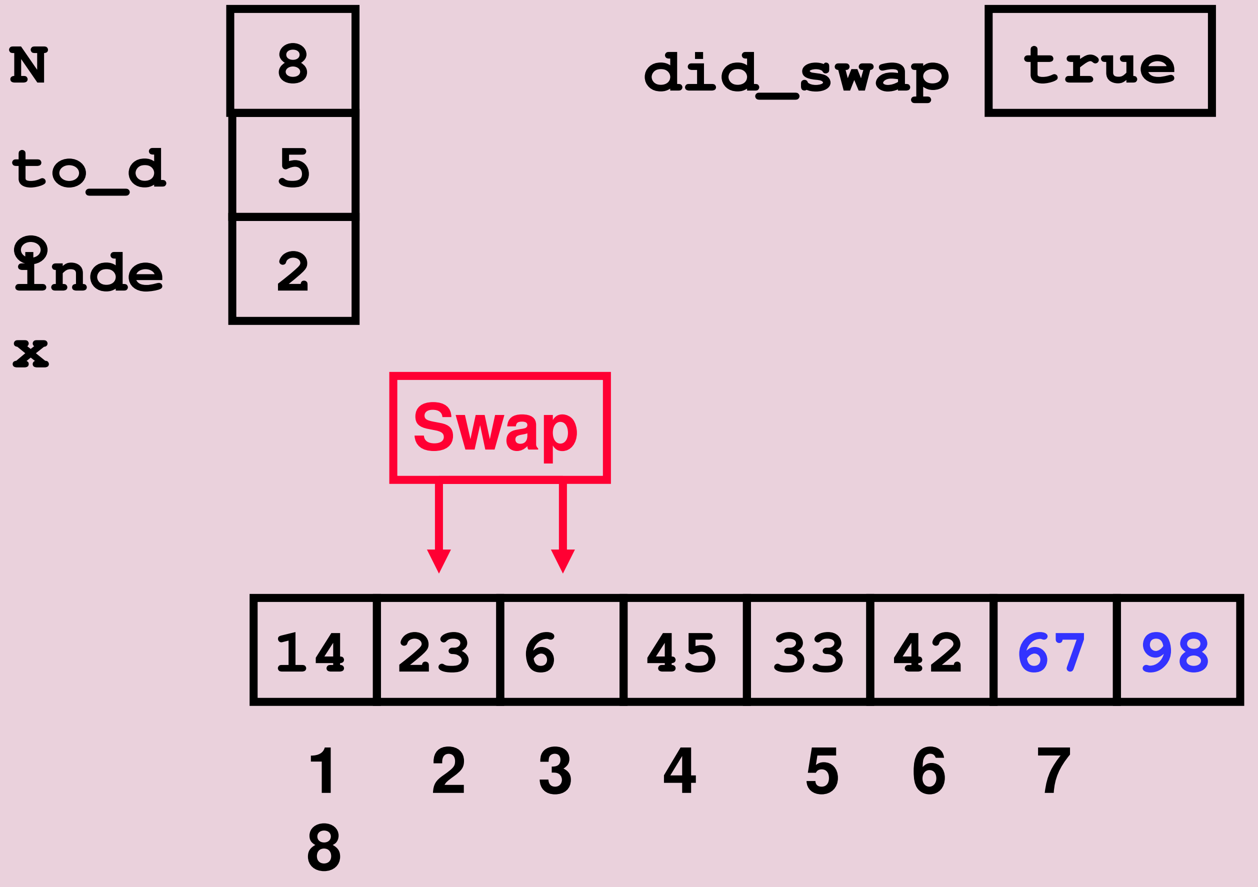
did_swap true



The Third "Bubble Up"



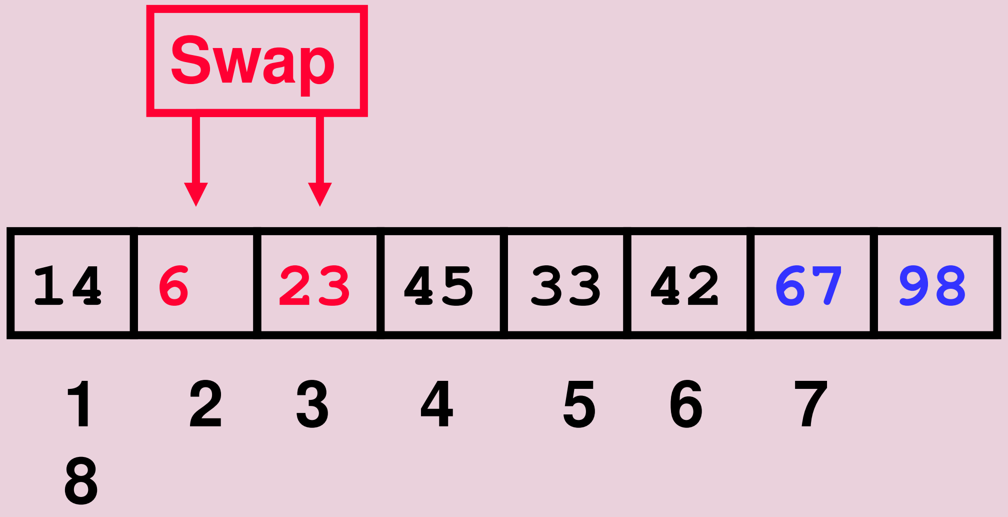
The Third "Bubble Up"



The Third “Bubble Up”

N	8
to_d	5
inde x	2

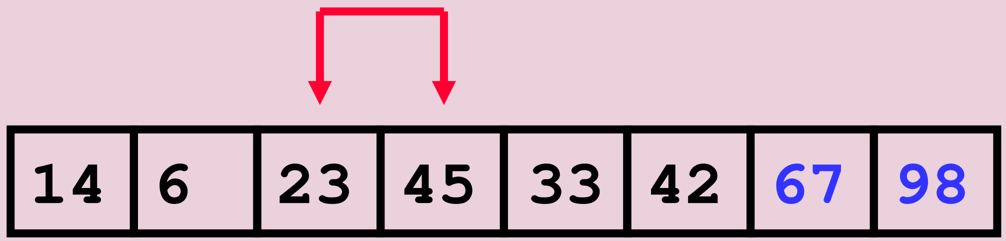
did_swap **true**



The Third “Bubble Up”

N	8
to_d	5
inde x	3

did_swap `true`



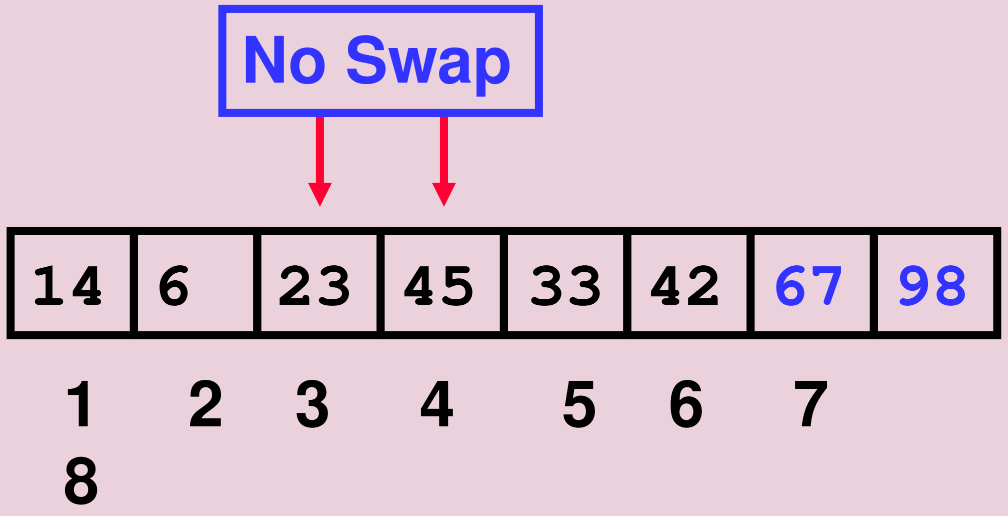
14	6	23	45	33	42	67	98

1 2 3 4 5 6 7
8

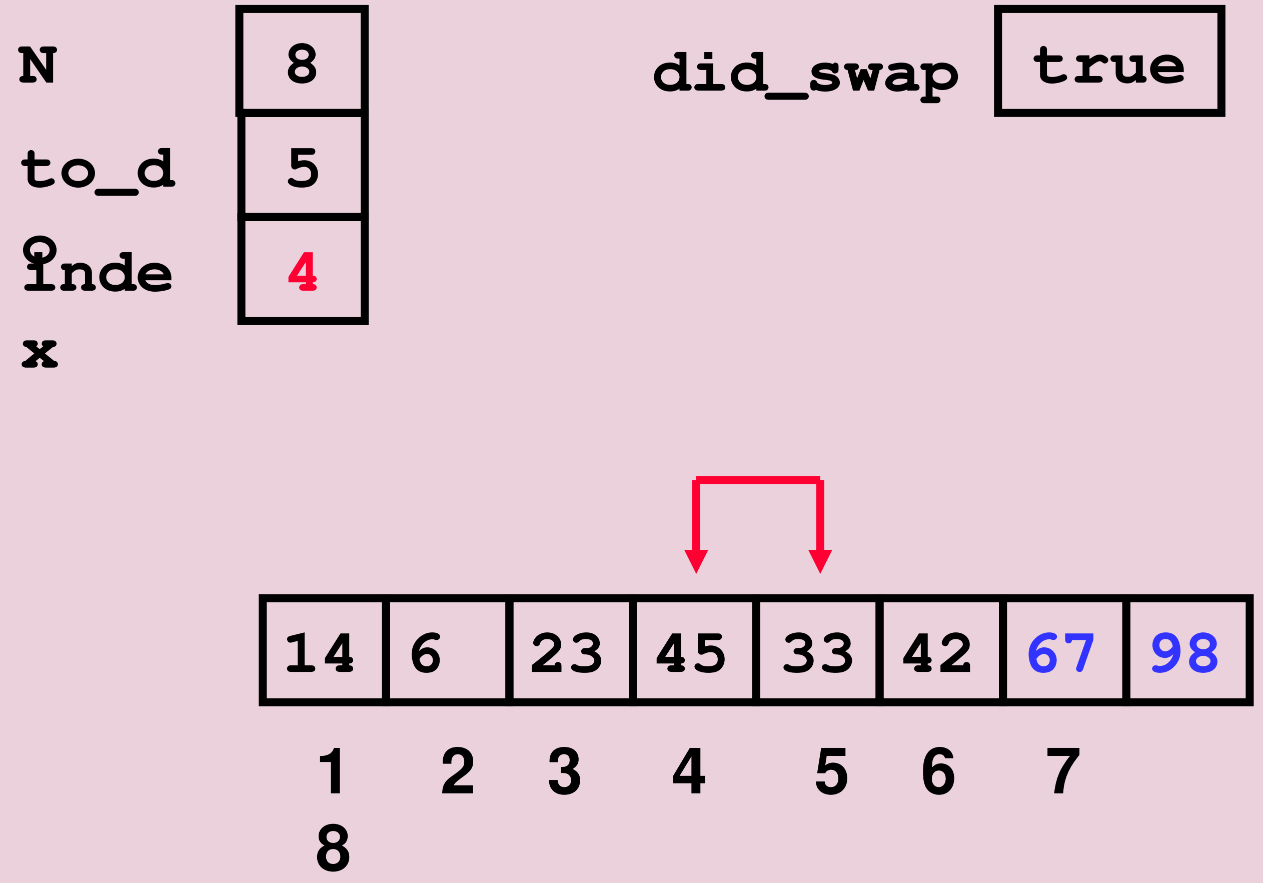
The Third "Bubble Up"

N	8
to_do	5
index	3

did_swap	true



The Third “Bubble Up”



The Third “Bubble Up”

N	8
to_d	5
index	4

did_swap `true`

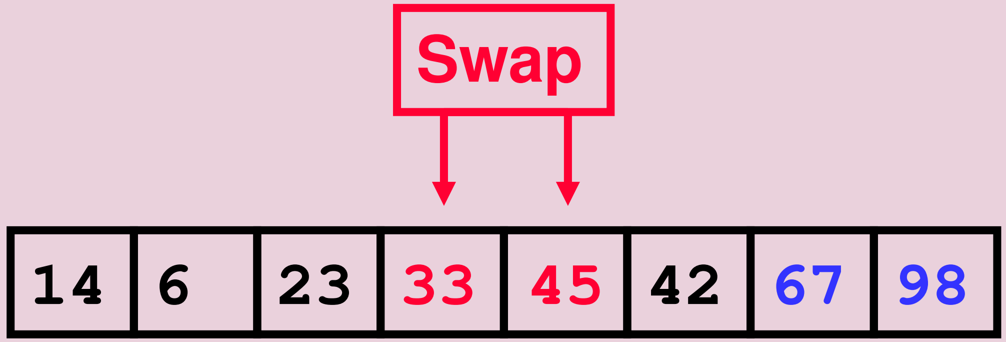
Swap

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

The Third “Bubble Up”

N	8
to_d	5
inde x	4

did_swap	true

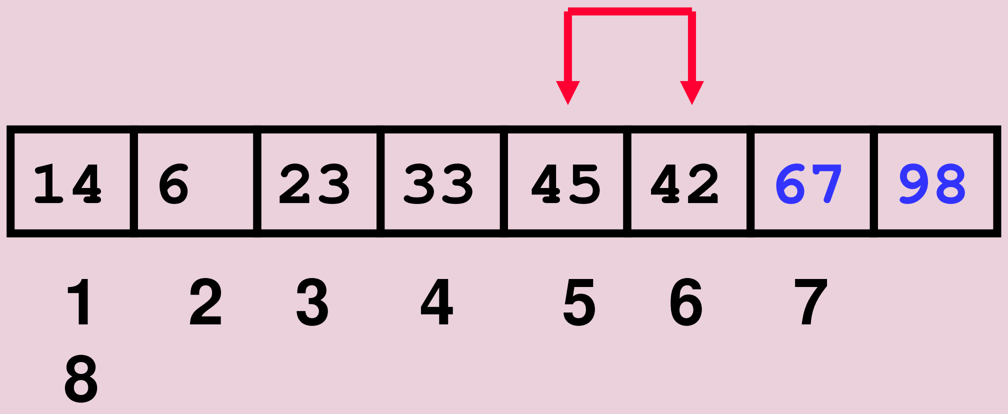


14	6	23	33	45	42	67	98
1	2	3	4	5	6	7	8

The Third “Bubble Up”

N	8
to_d	5
inde x	5

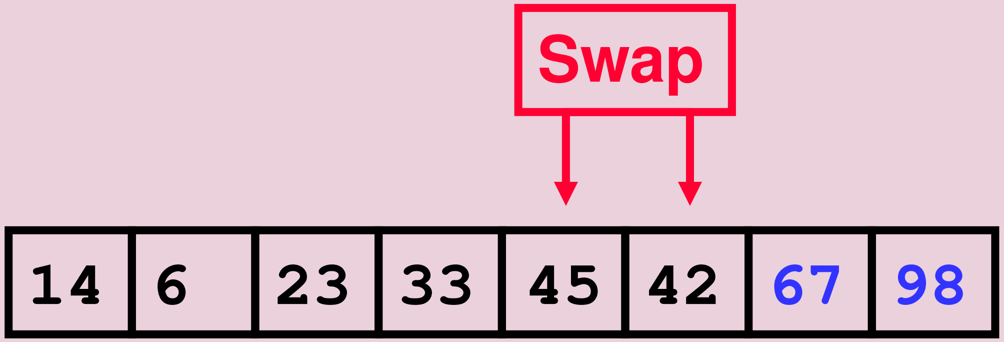
did_swap `true`



The Third “Bubble Up”

N	8
to_d	5
index	5

did_swap `true`

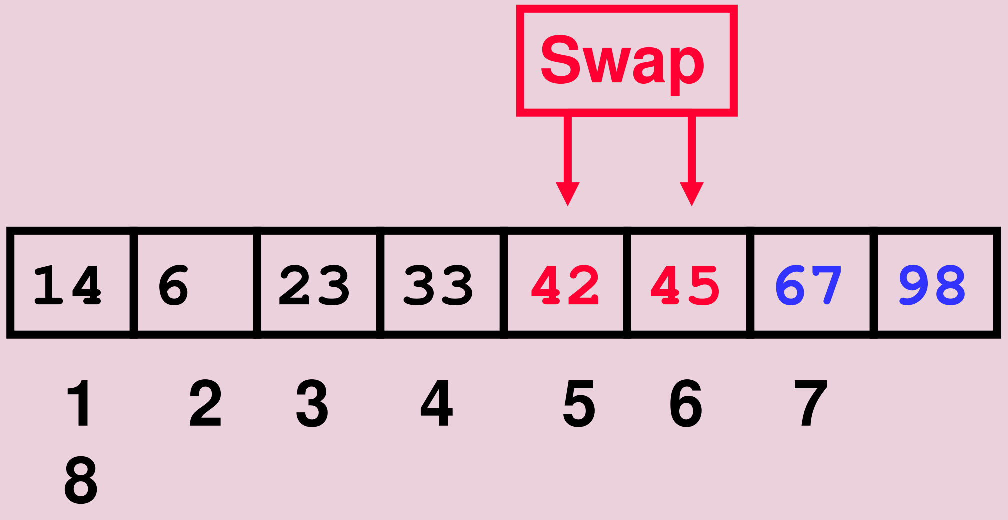


14	6	23	33	45	42	67	98
1	2	3	4	5	6	7	8

The Third “Bubble Up”

N	8
to_d	5
inde x	5

did_swap `true`

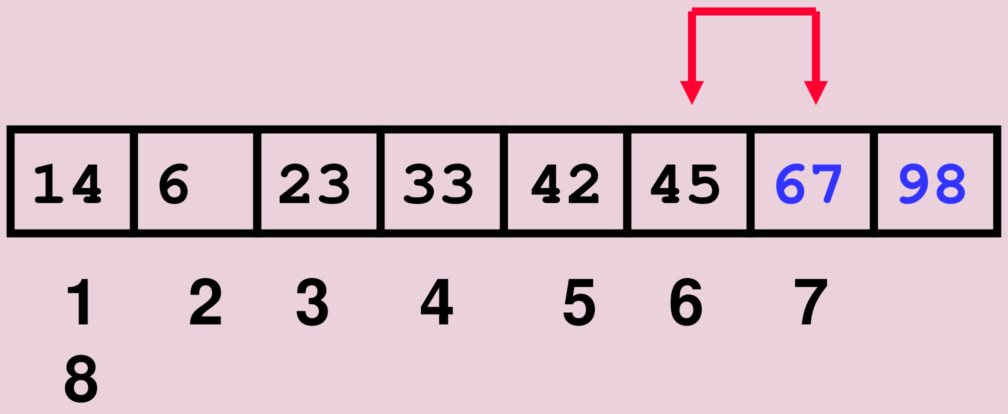


After Third Pass of Outer Loop

`N`	8
`to_d`	5
`inde x`	6

`did_swap` `true`

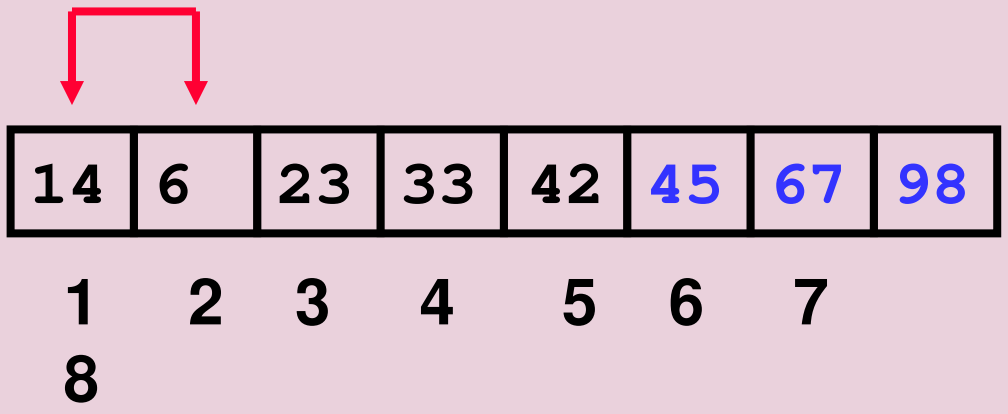
Finished third “Bubble Up”



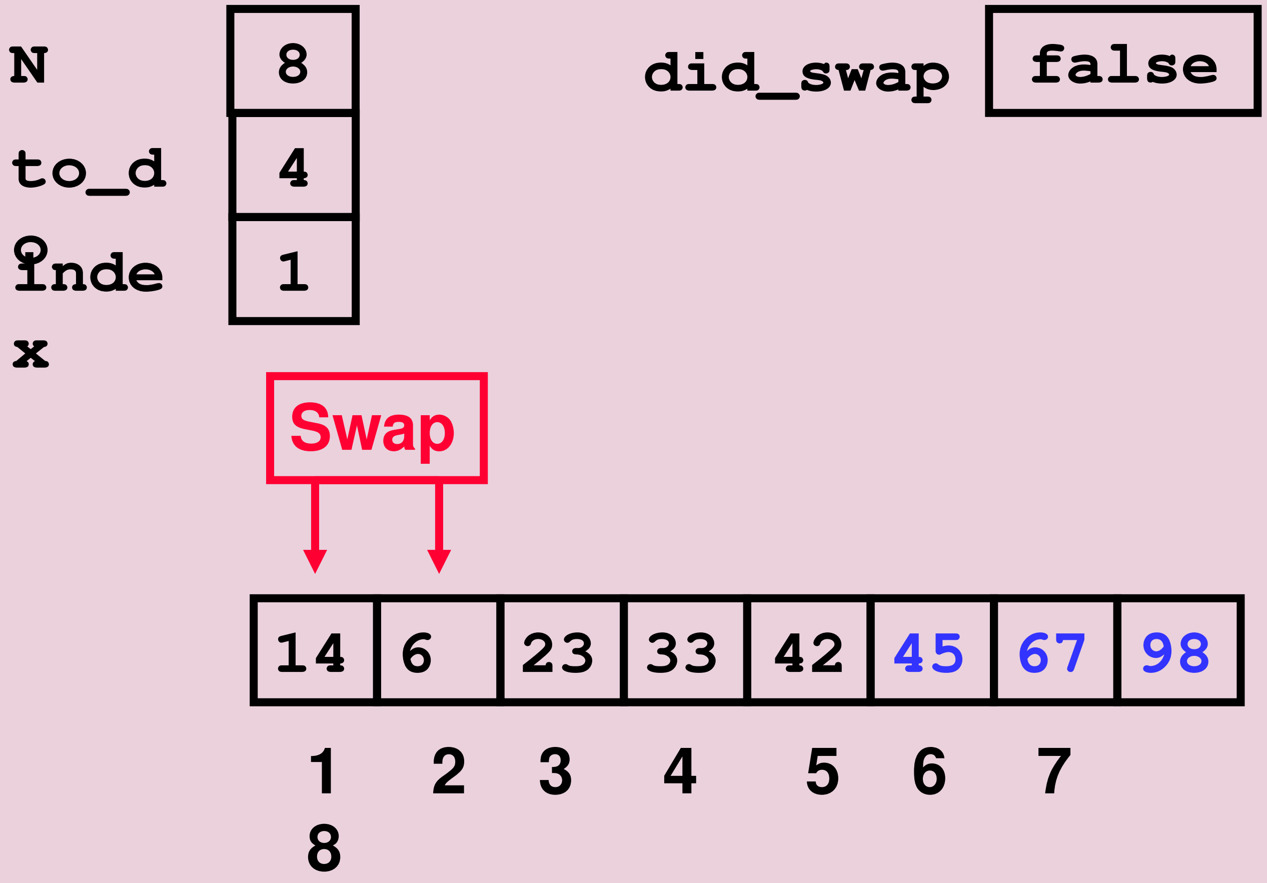
The Fourth “Bubble Up”

N	8
to_d	4
inde x	1

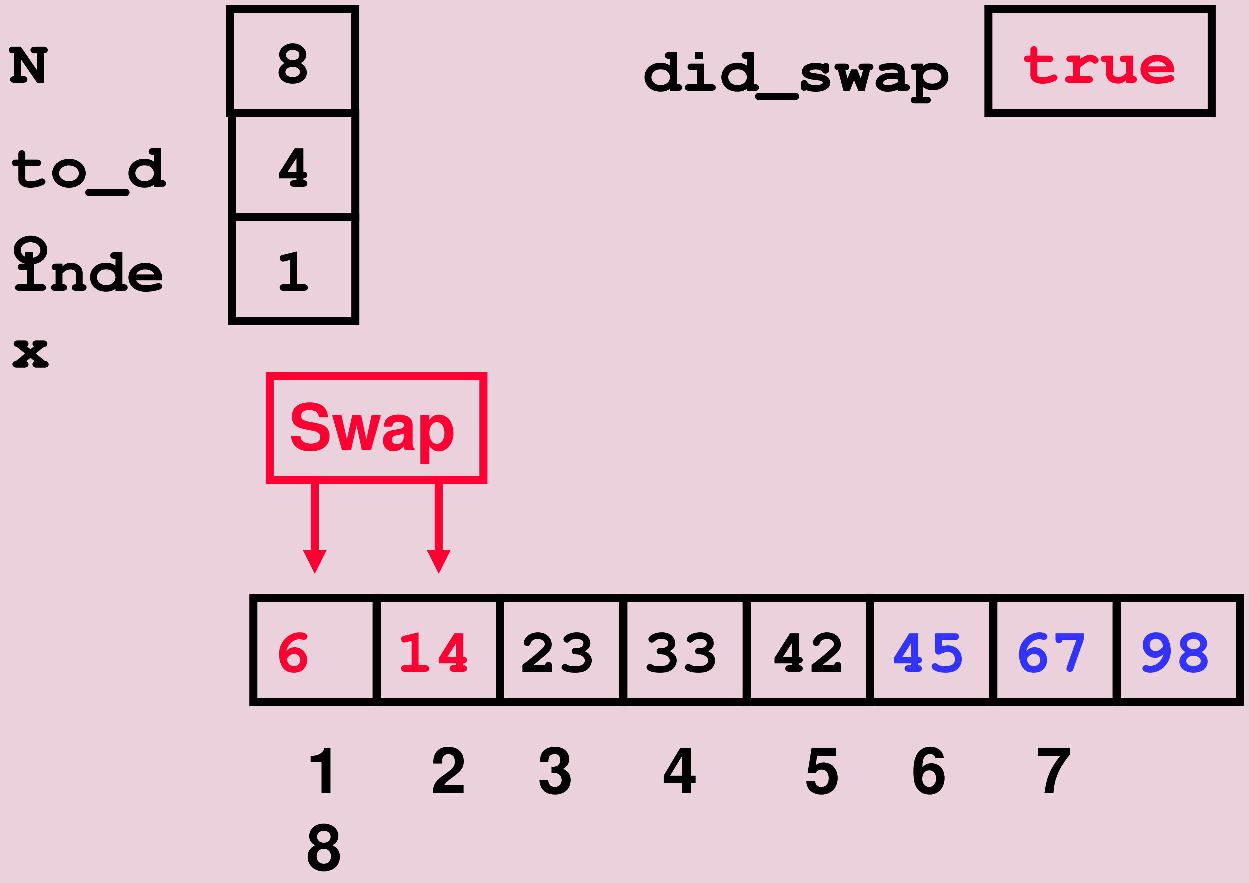
did_swap false



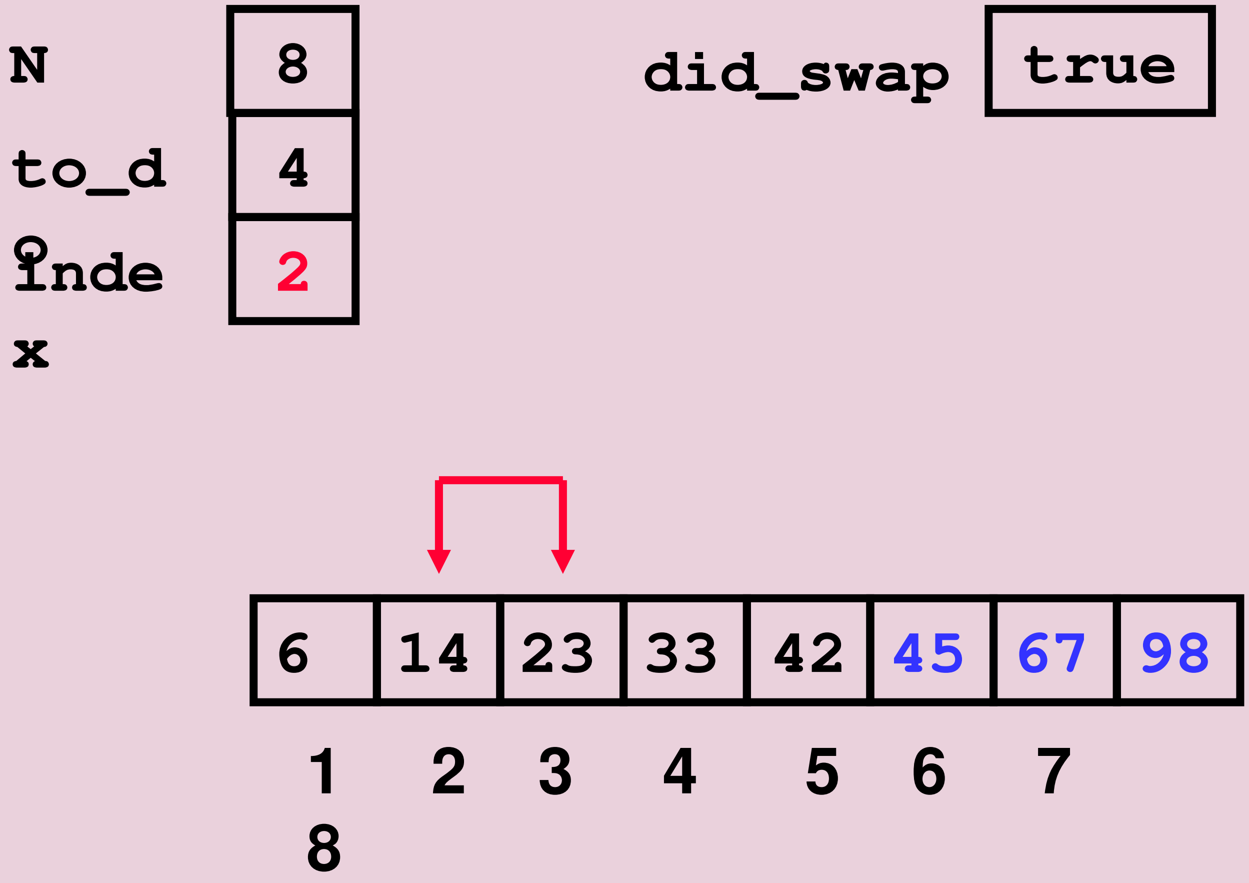
The Fourth "Bubble Up"



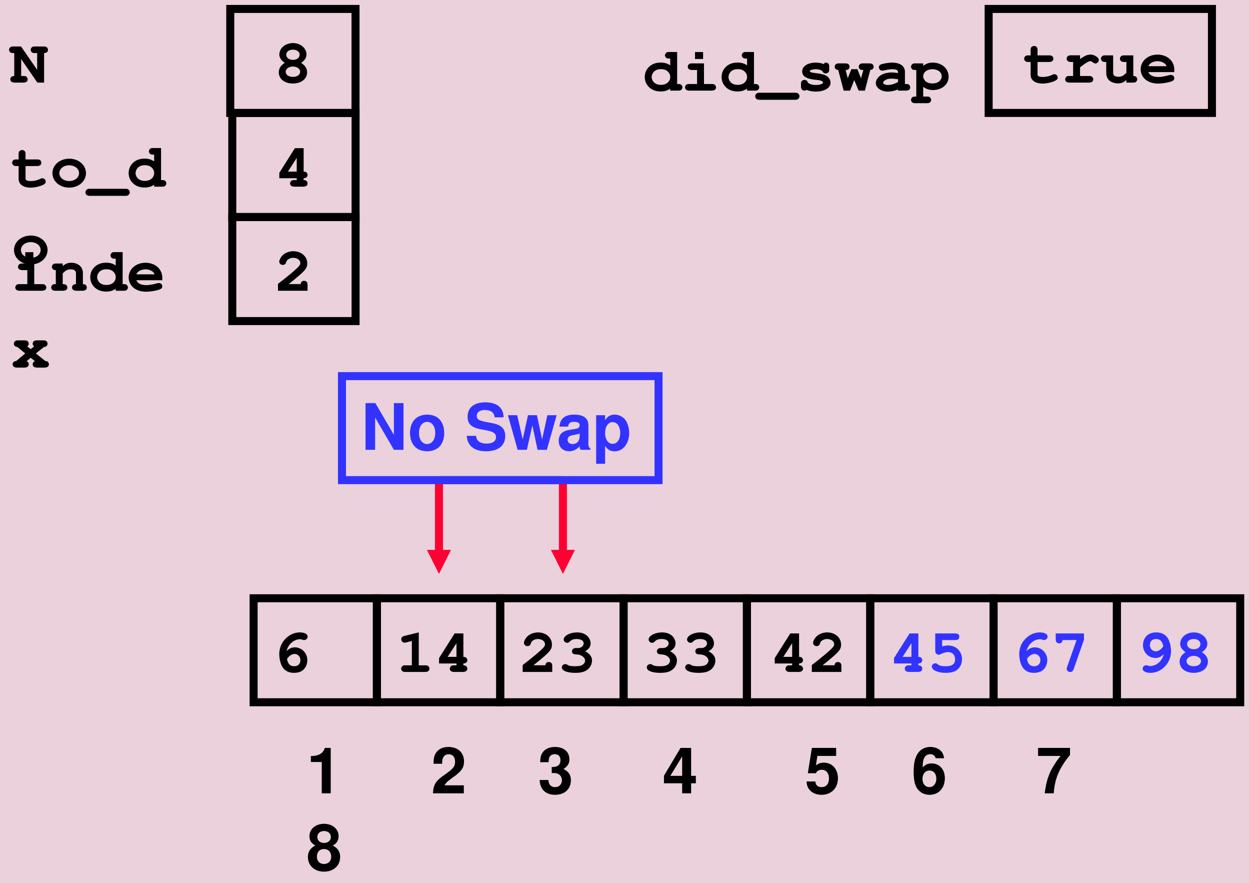
The Fourth "Bubble Up"



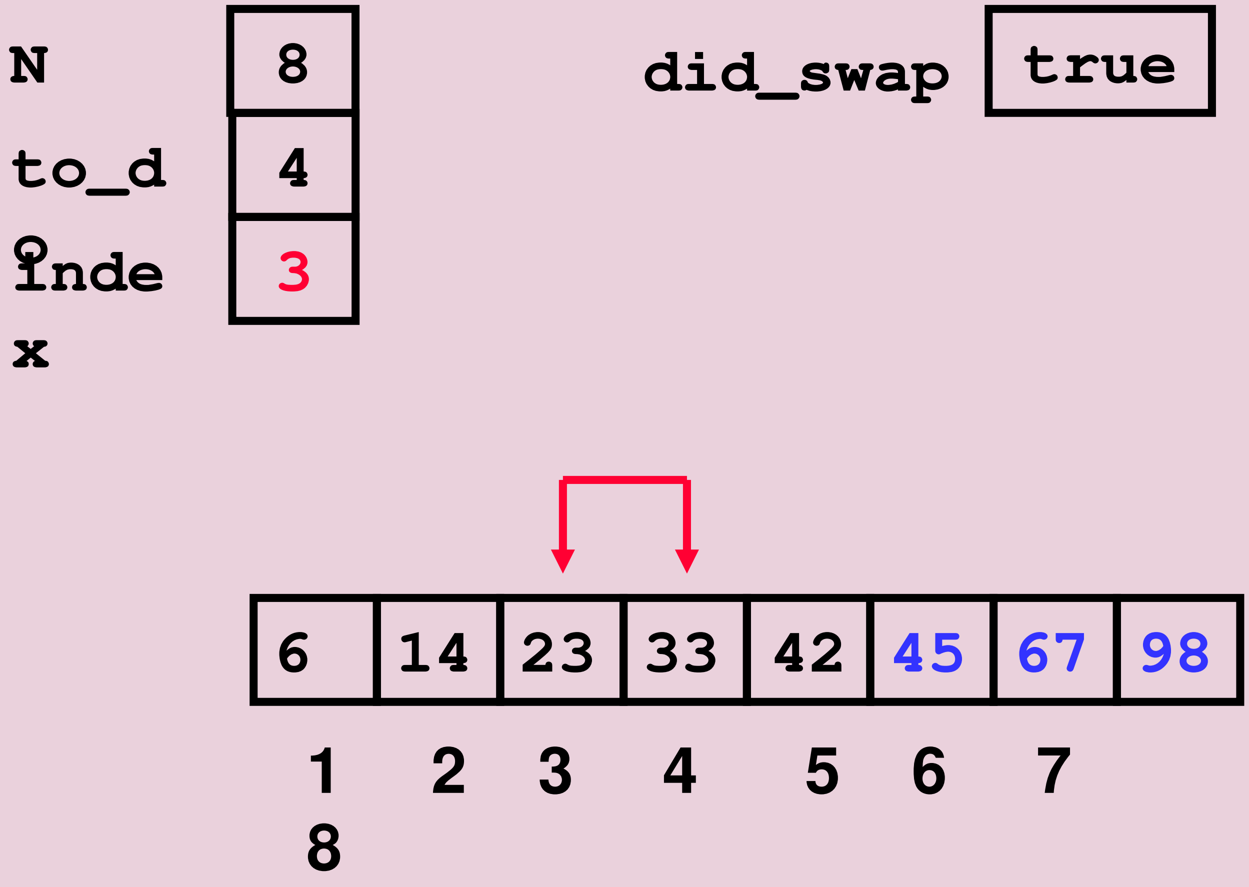
The Fourth "Bubble Up"



The Fourth “Bubble Up”



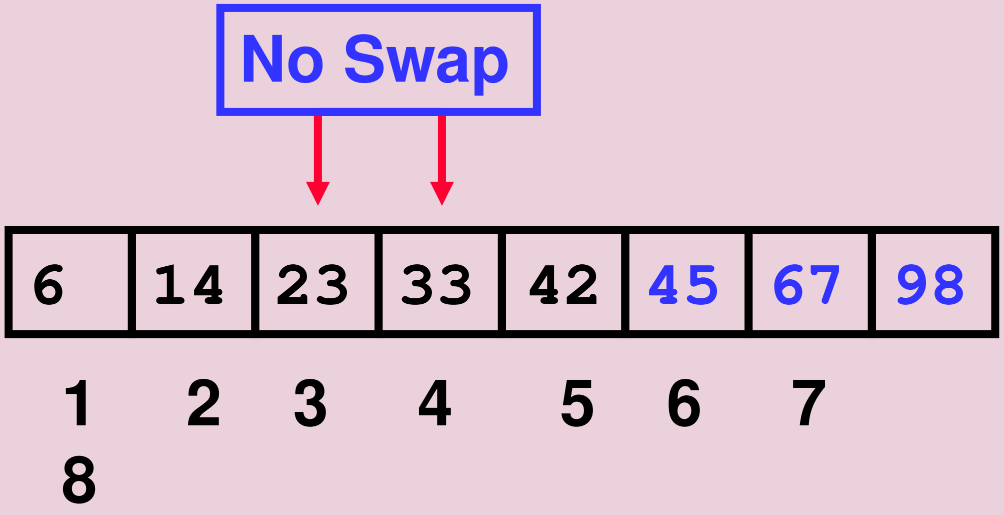
The Fourth “Bubble Up”



The Fourth “Bubble Up”

N	8
to_d o	4
inde x	3

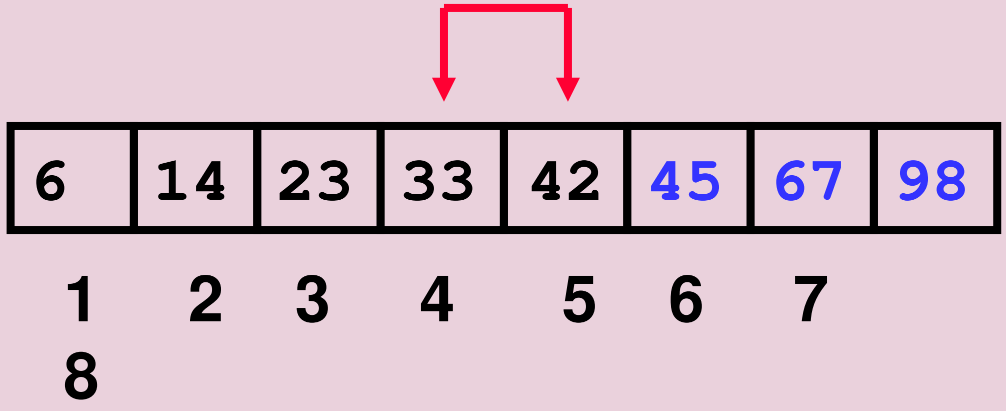
did_swap `true`



The Fourth "Bubble Up"

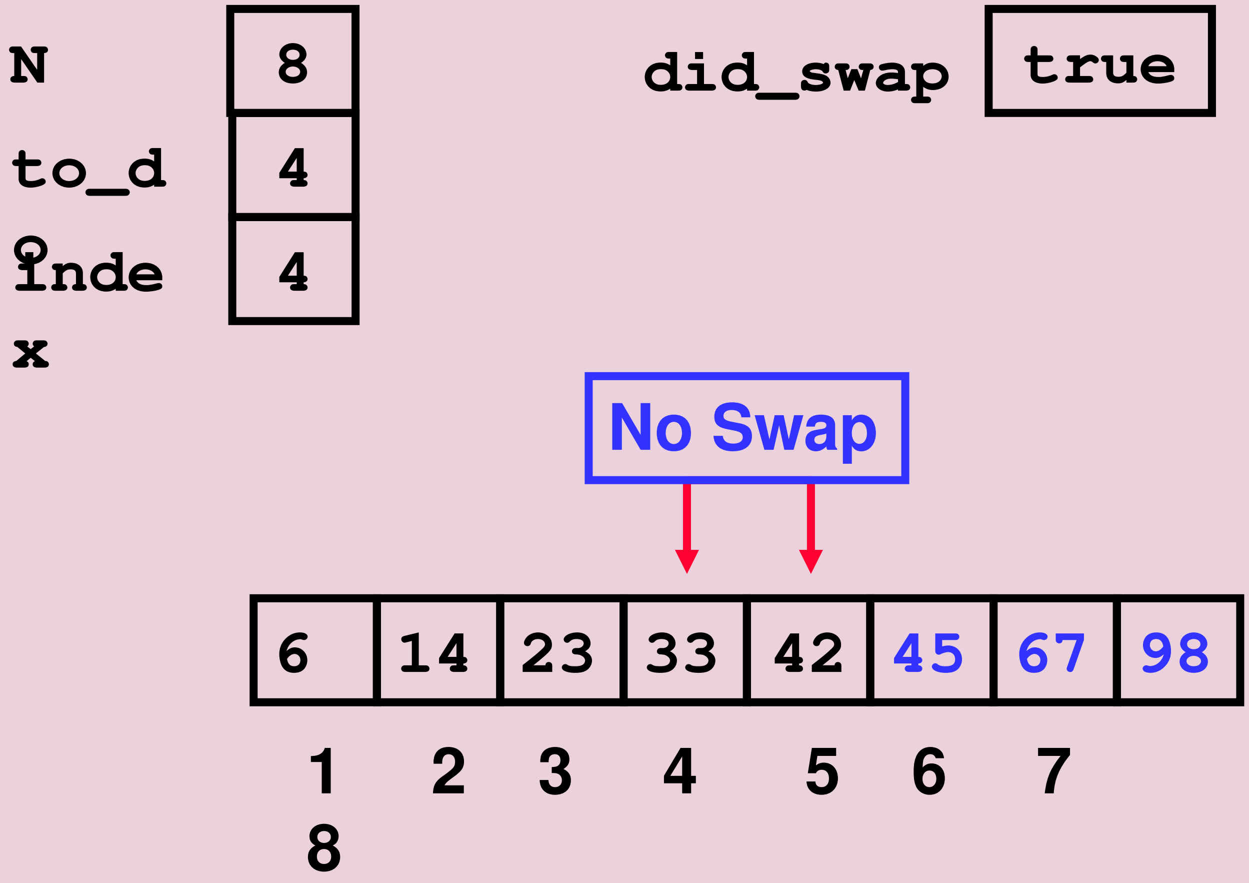
N	8
to_do	4
index	4

did_swap: true



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

The Fourth "Bubble Up"

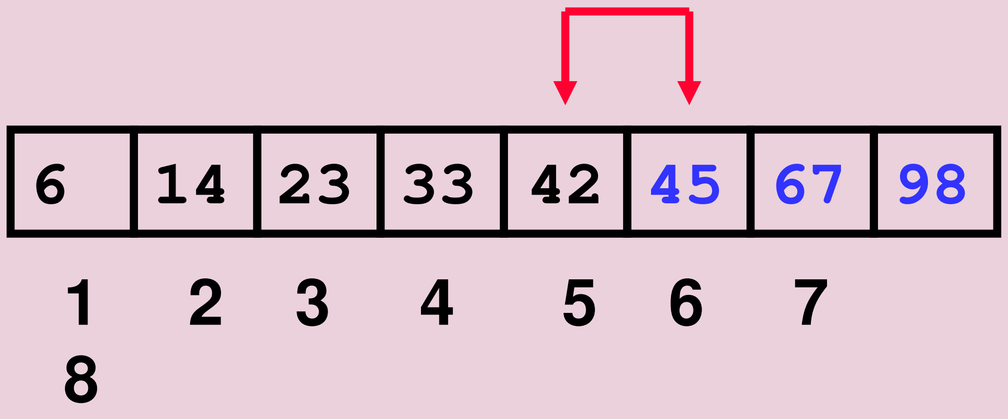


After Fourth Pass of Outer Loop

`N`	8
`to_d`	4
`inde x`	5

`did_swap` `true`

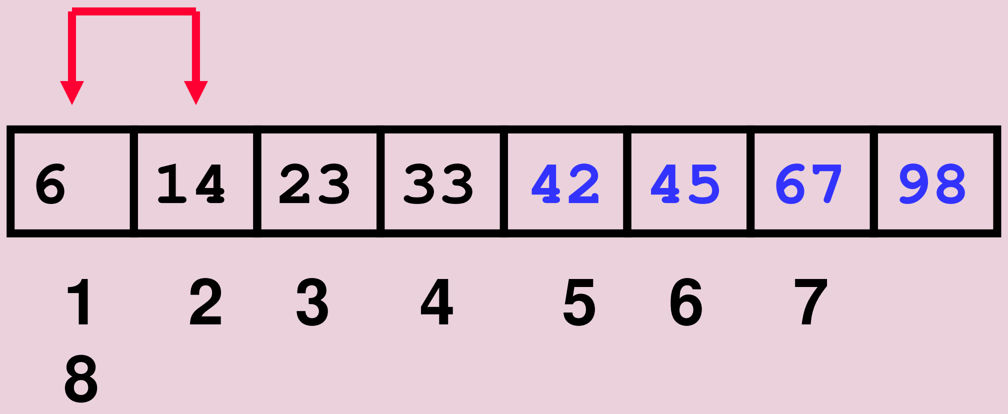
Finished fourth "Bubble Up"



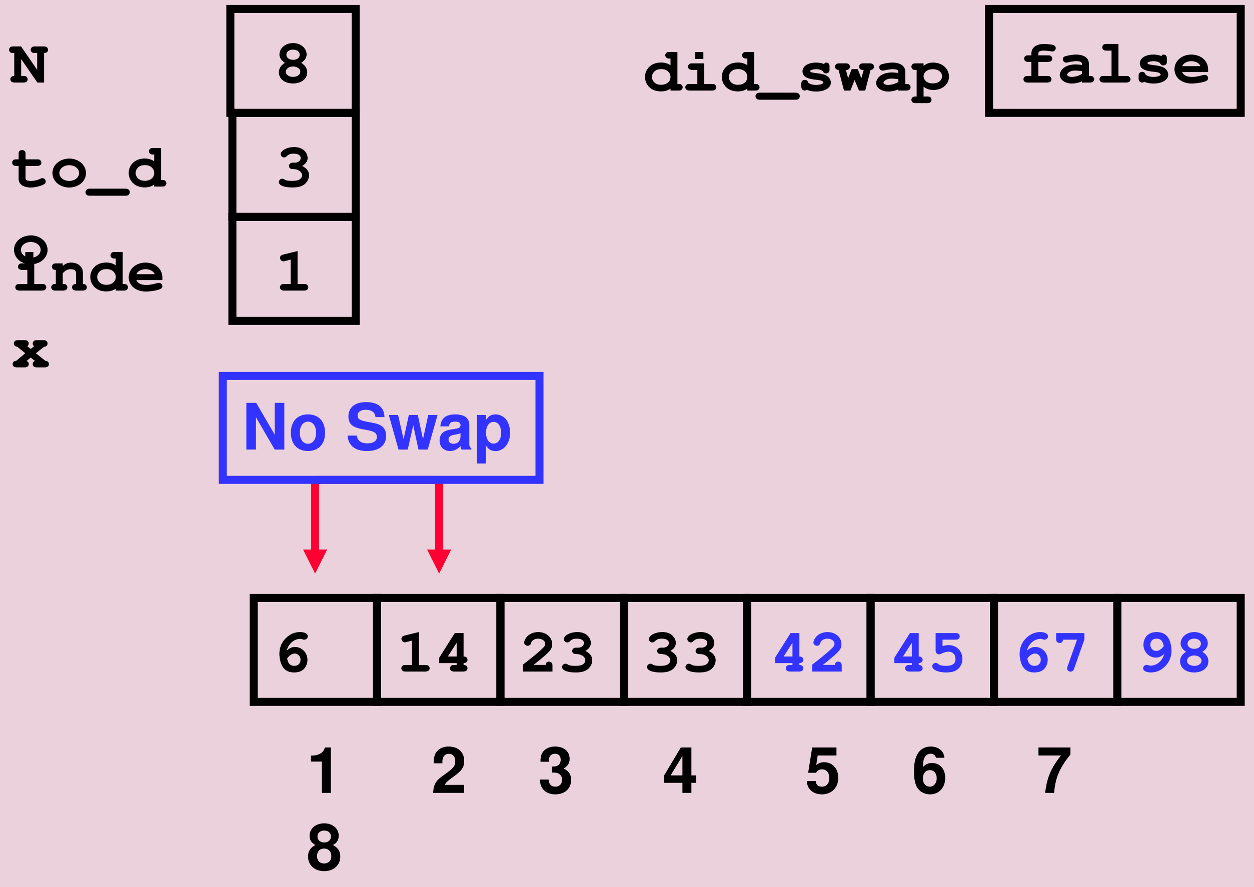
The Fifth “Bubble Up”

N	8
to_d	3
inde x	1

did_swap false



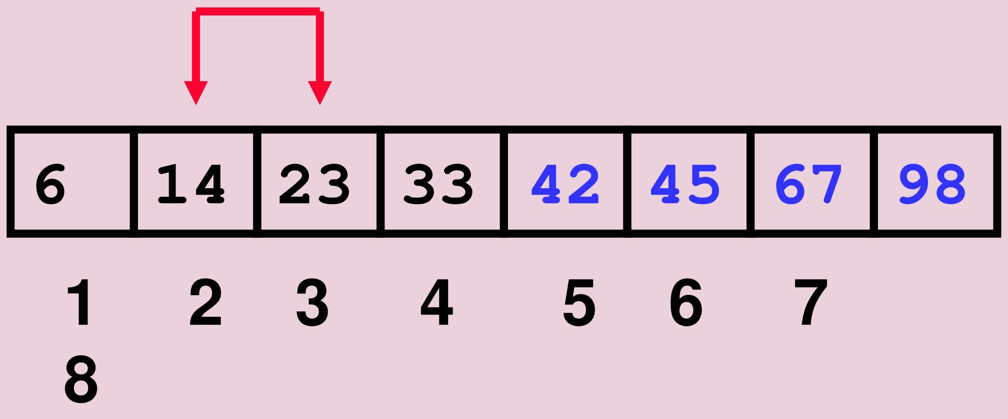
The Fifth “Bubble Up”



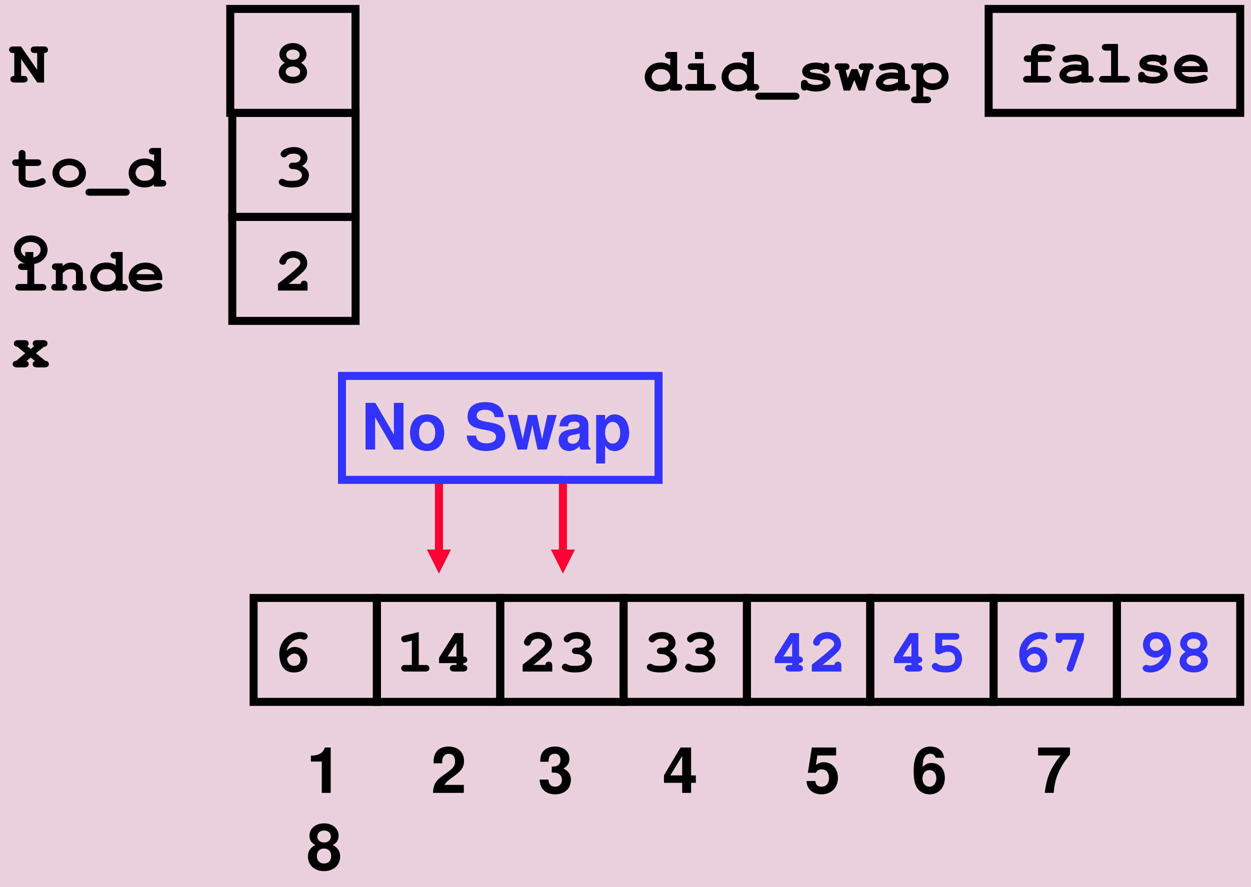
The Fifth “Bubble Up”

N	8
to_d o	3
inde x	2

did_swap false



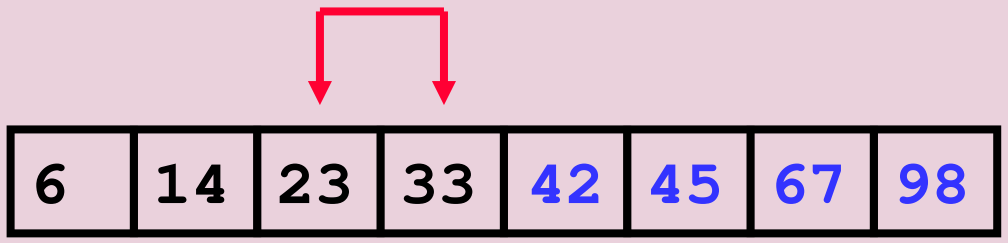
The Fifth “Bubble Up”



The Fifth “Bubble Up”

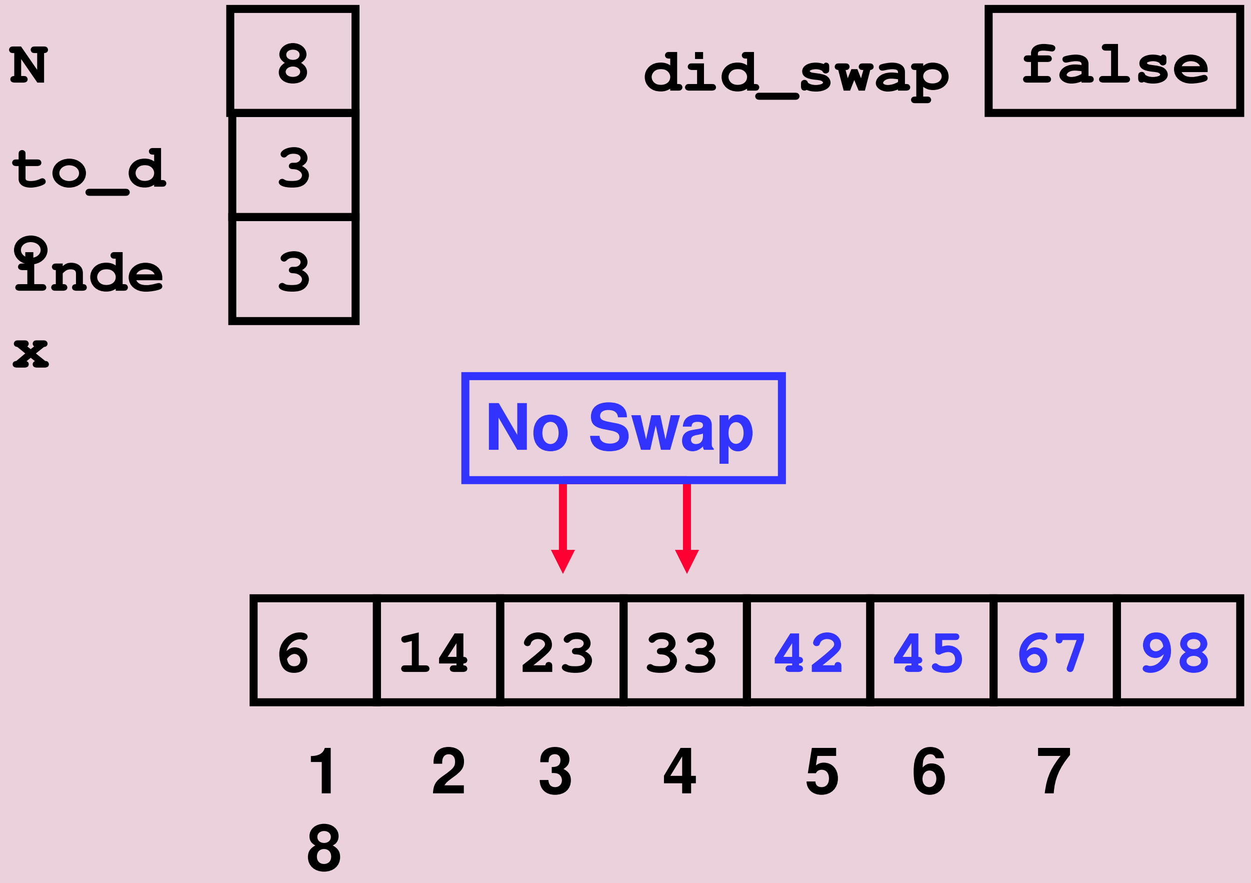
N	8
to_do	3
index	3

did_swap `false`



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

The Fifth “Bubble Up”

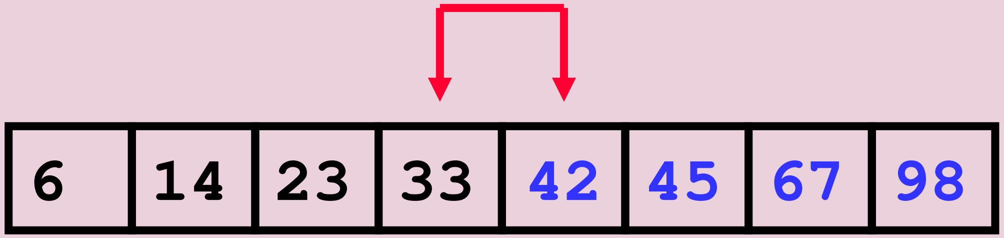


After Fifth Pass of Outer Loop

N	8
to_d	3
oinde	4
x	

did_swap `false`

Finished fifth “Bubble Up”



6	14	23	33	42	45	67	98

1 2 3 4 5 6 7
8

Finished “Early”

`N`	`8`
`to_d`	`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