

Bubble Sort

Sorting

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

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



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

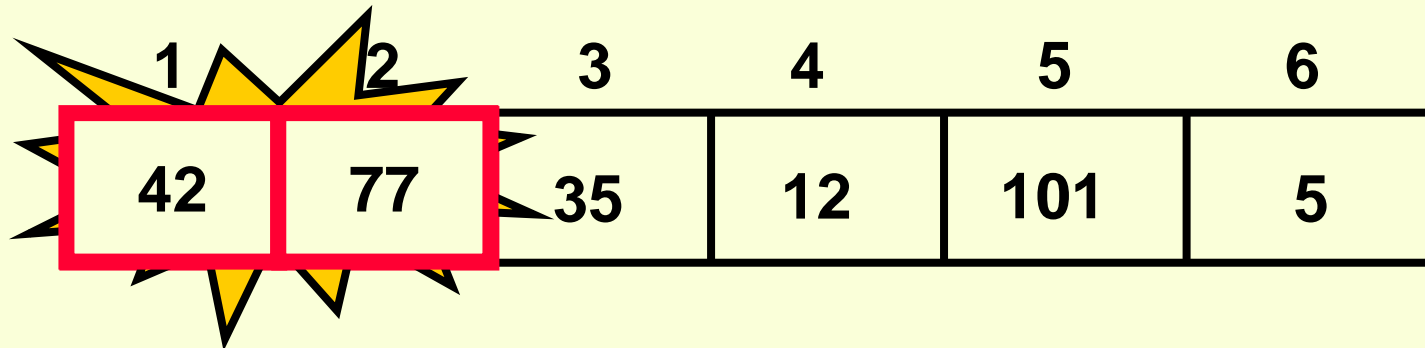
"Bubbling Up" the Largest Element

- 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

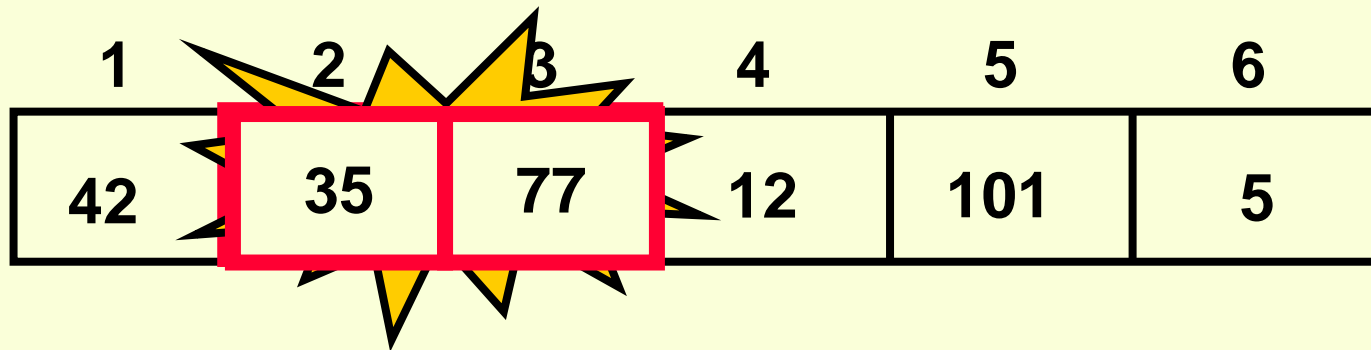
"Bubbling Up" the Largest Element

- 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



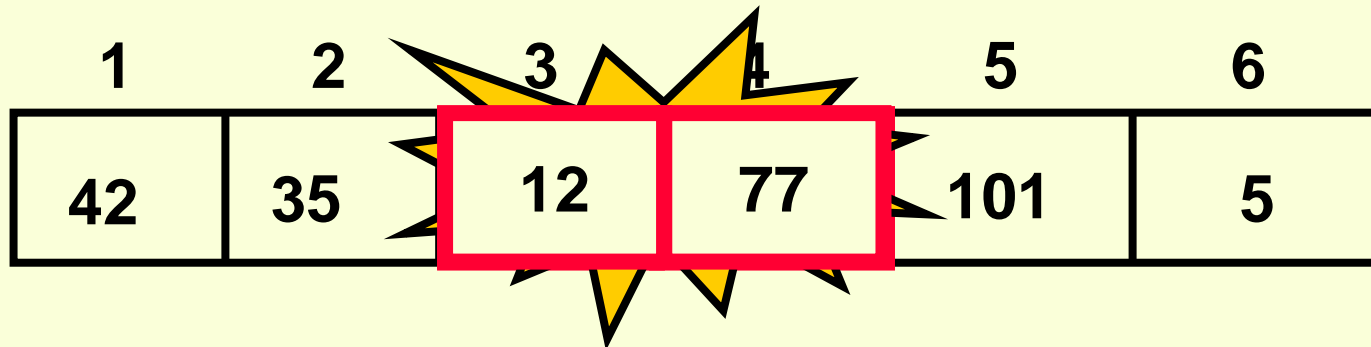
"Bubbling Up" the Largest Element

- 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

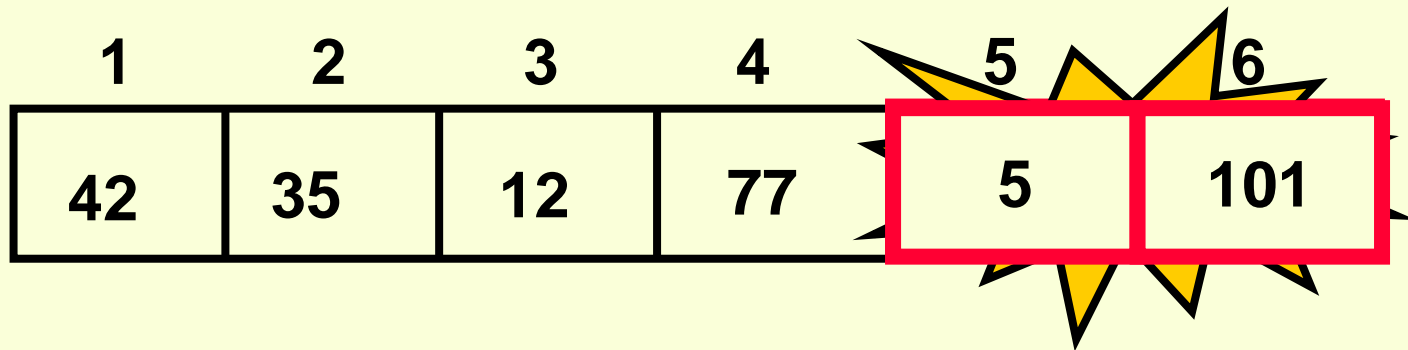
- Traverse a collection of elements
 - Move from the front to the end
 - “Bubble” the largest value to the end using pair-wise comparisons and swapping

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

No need to swap

"Bubbling Up" the Largest Element

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



"Bubbling Up" the Largest Element

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

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

Largest value correctly placed

The “Bubble Up” Algorithm

```
index <- 1
last_compare_at <- n - 1

loop
  exitif(index > last_compare_at)
  if(A[index] > A[index + 1]) then
    Swap(A[index], A[index + 1])
  endif
  index <- index + 1
endloop
```

No, Swap isn't built in.

```
Procedure Swap(a, b isoftype in/out Num)
  t isoftype Num
  t <- a
  a <- b
  b <- t
endprocedure // Swap
```

```
void sort(int b[], int n){  
    int t;  
    for(int k=n-1; k>=1; k--)  
    { for(int s=0;s<k;s++)  
      { if(b[s]>b[s+1])  
        {t=b[s+1];  
         b[s+1]=b[s];  
         b[s]=t; }          }      }  
    }
```

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

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

Reducing the Number of Comparisons

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

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

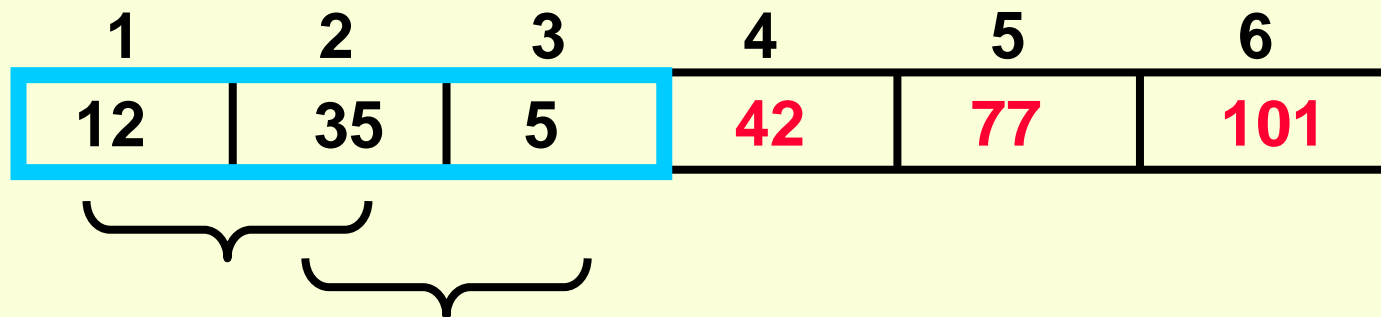
1	2	3	4	5	6
35	12	42	5	77	101

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

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

Reducing the Number of Comparisons

- On the N^{th} “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



Putting It All Together

N is ... // Size of Array

Arr_Type definesa Array[1..N] of Num

Procedure Swap(n1, n2 isoftype in/out Num)

temp isoftype Num





temp <- n1

n1 <- n2

n2 <- temp

endprocedure // Swap

```
procedure Bubblesort(A isoftype in/out Arr_Type)
  to_do, index isoftype Num
  to_do <- N - 1
```

```
  loop ← 
    exitif(to_do = 0)
    index <- 1
    loop ← 
      exitif(index > to_do)
      if(A[index] > A[index + 1]) then
        Swap(A[index], A[index + 1])
      endif
      index <- index + 1
    endloop ← 
    to_do <- to_do - 1
  endloop ← 
endprocedure // Bubblesort
```

Inner loop

Outer loop

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.”

```
did_swap isoftype Boolean  
did_swap <- true
```

```
loop  
  exitif ((to_do = 0) OR NOT(did_swap))  
  index <- 1  
  did_swap <- false  
  loop  
    exitif(index > to_do)  
    if(A[index] > A[index + 1]) then  
      Swap(A[index], A[index + 1])  
      did_swap <- true  
    endif  
    index <- index + 1  
  endloop  
  to_do <- to_do - 1  
endloop
```

An Animated Example

N

8

 did_swap

true

to_do

7

index

--

98	23	45	14	6	67	33	42
----	----	----	----	---	----	----	----

1 2 3 4 5 6 7 8

An Animated Example

N

8

 did_swap

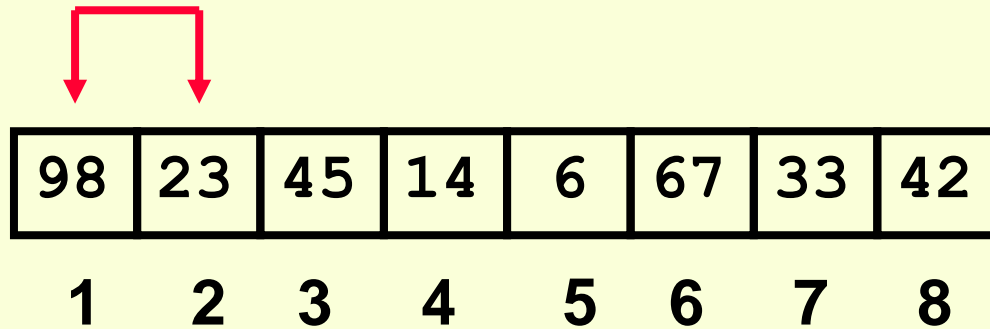
false

to_do

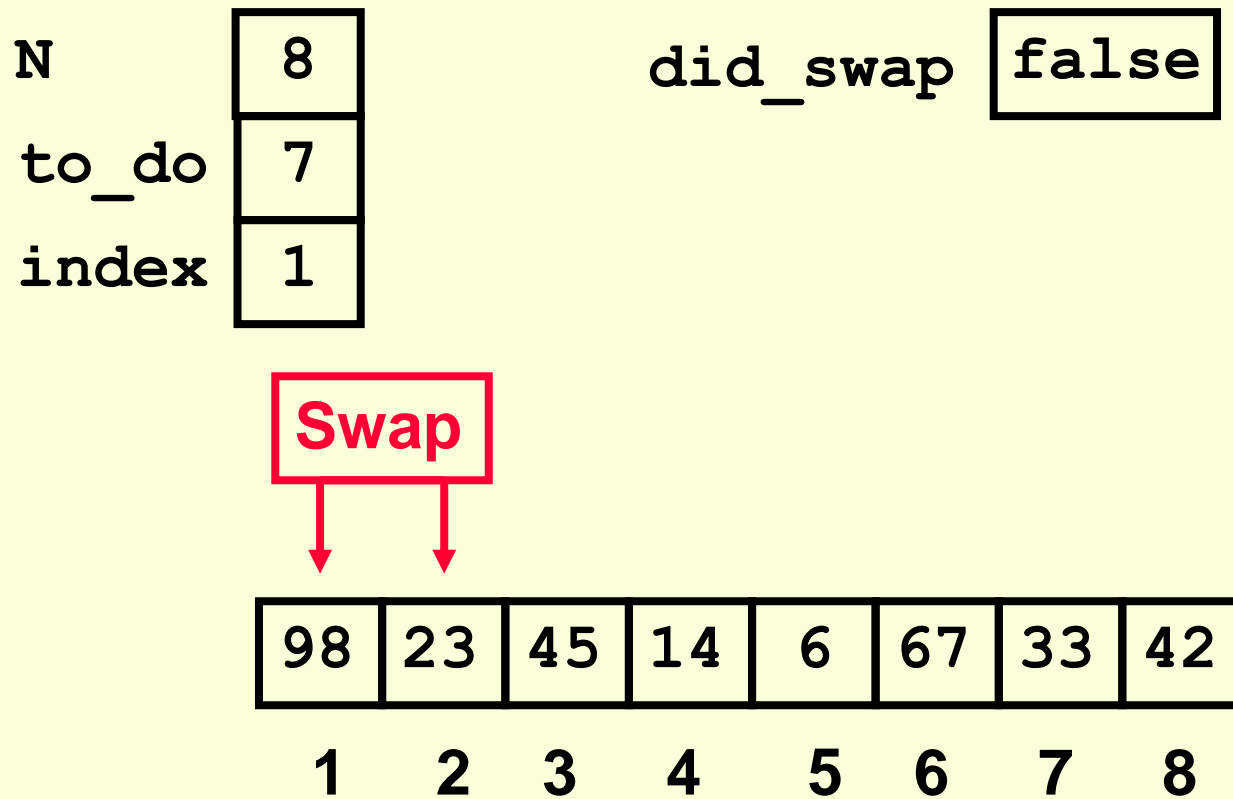
7

index

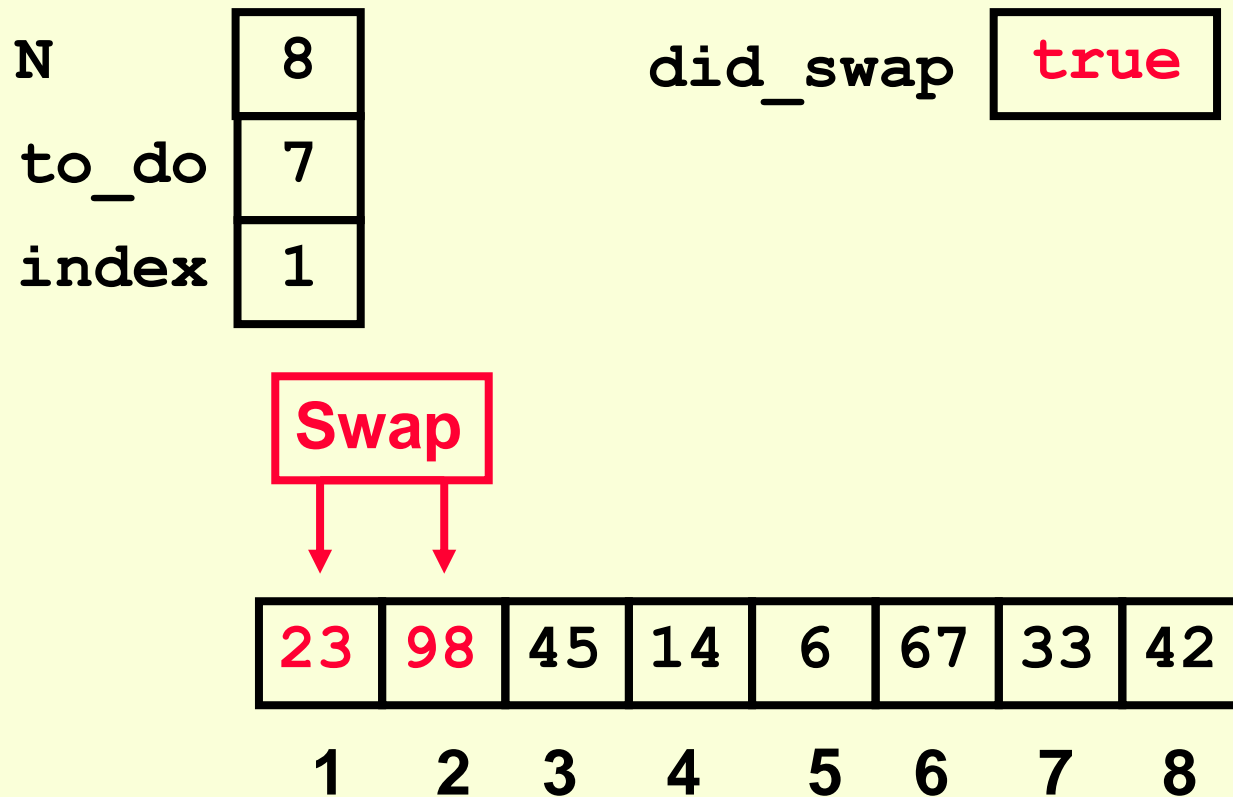
1



An Animated Example



An Animated Example



An Animated Example

N

8

 did_swap

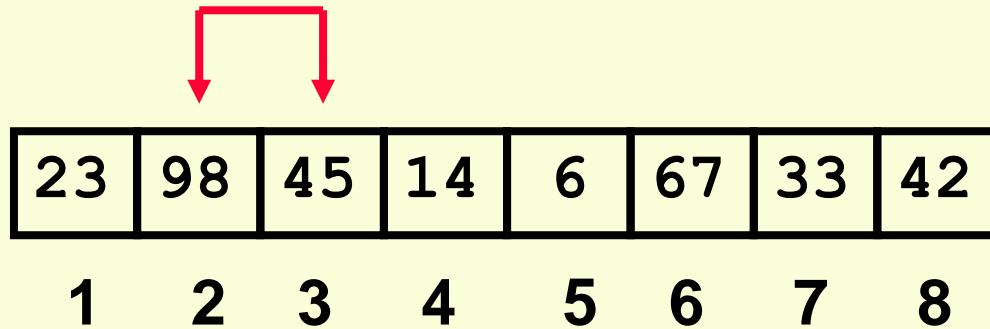
true

to_do

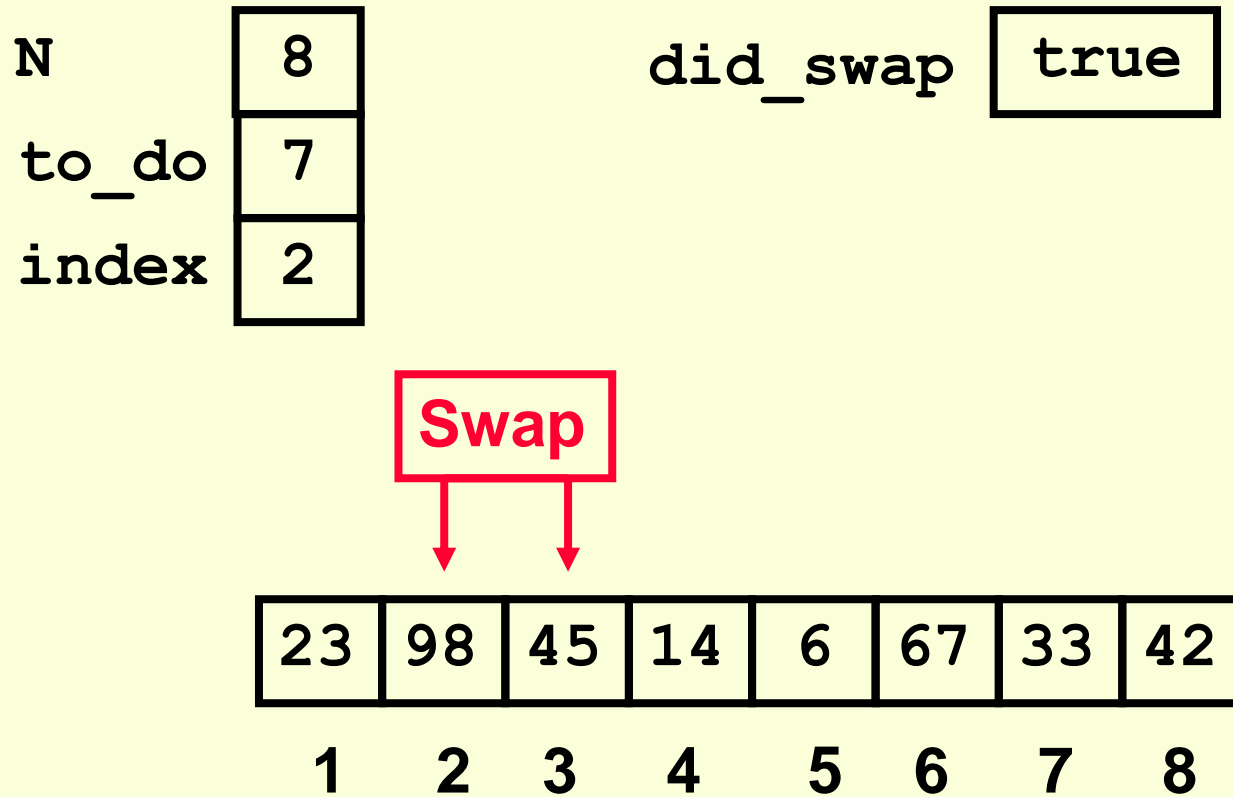
7

index

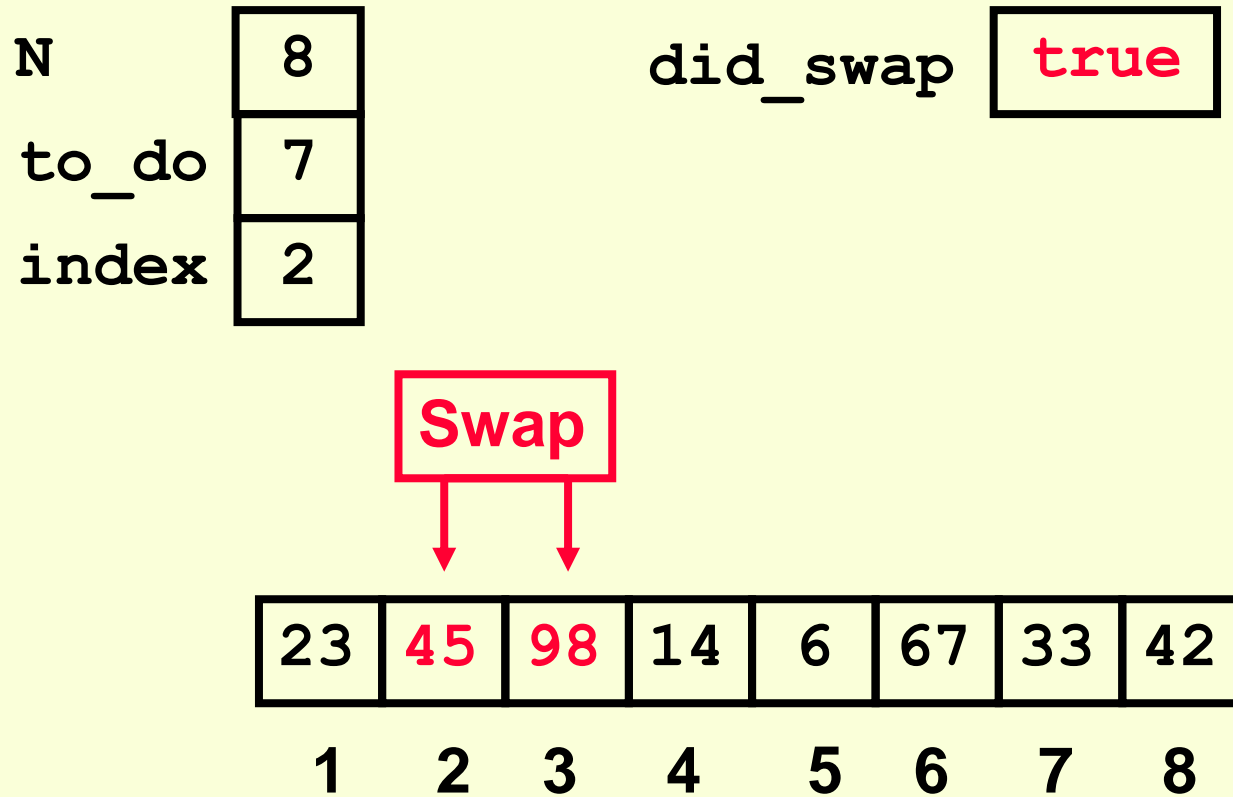
2



An Animated Example



An Animated Example



An Animated Example

N

8

 did_swap

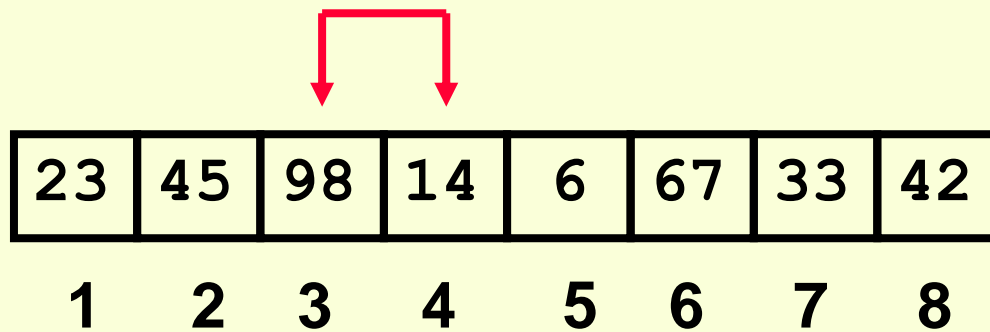
true

to_do

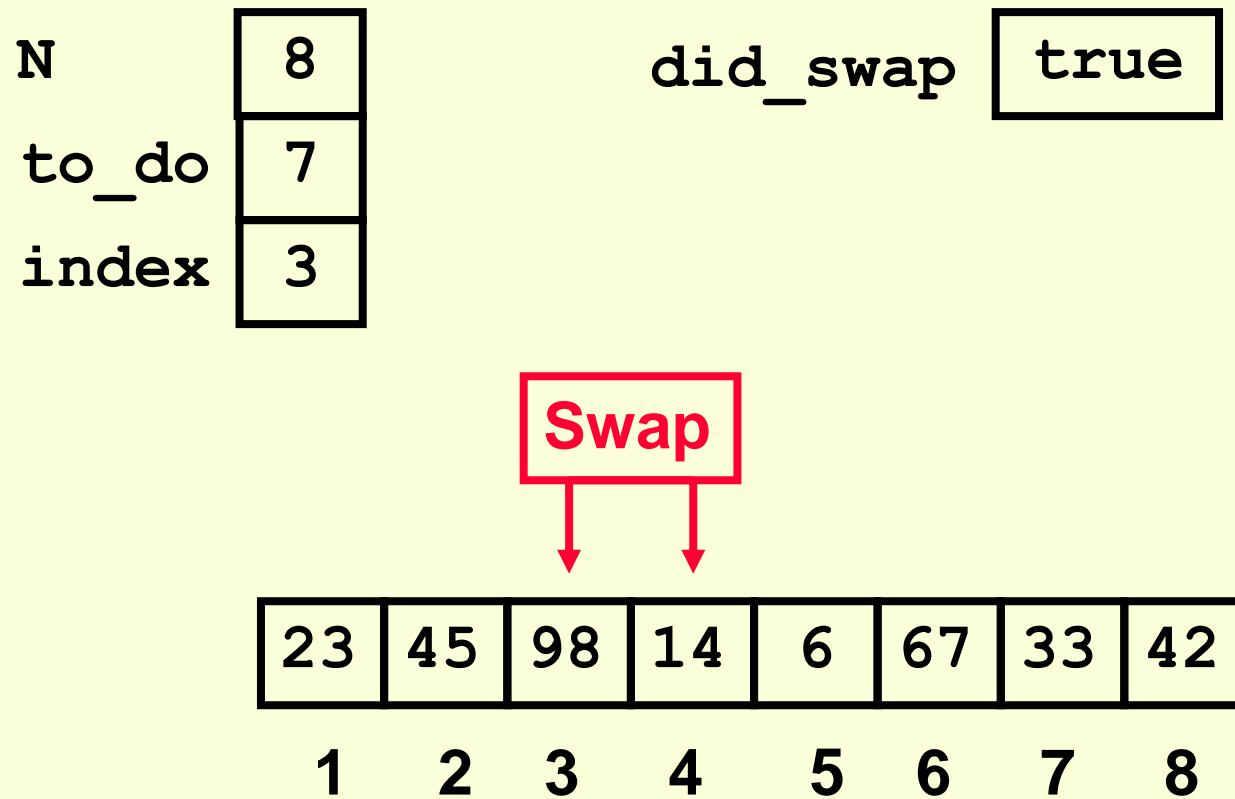
7

index

3



An Animated Example



An Animated Example

N

8

to_do

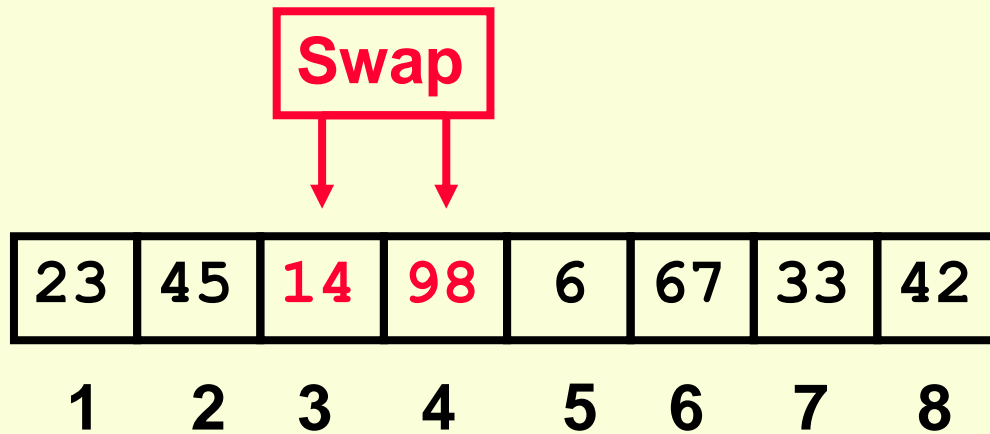
7

index

3

did_swap

true



An Animated Example

N

8

to_do

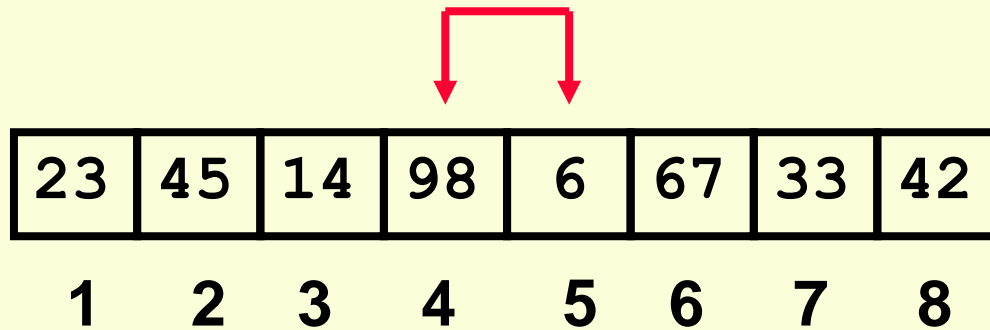
7

index

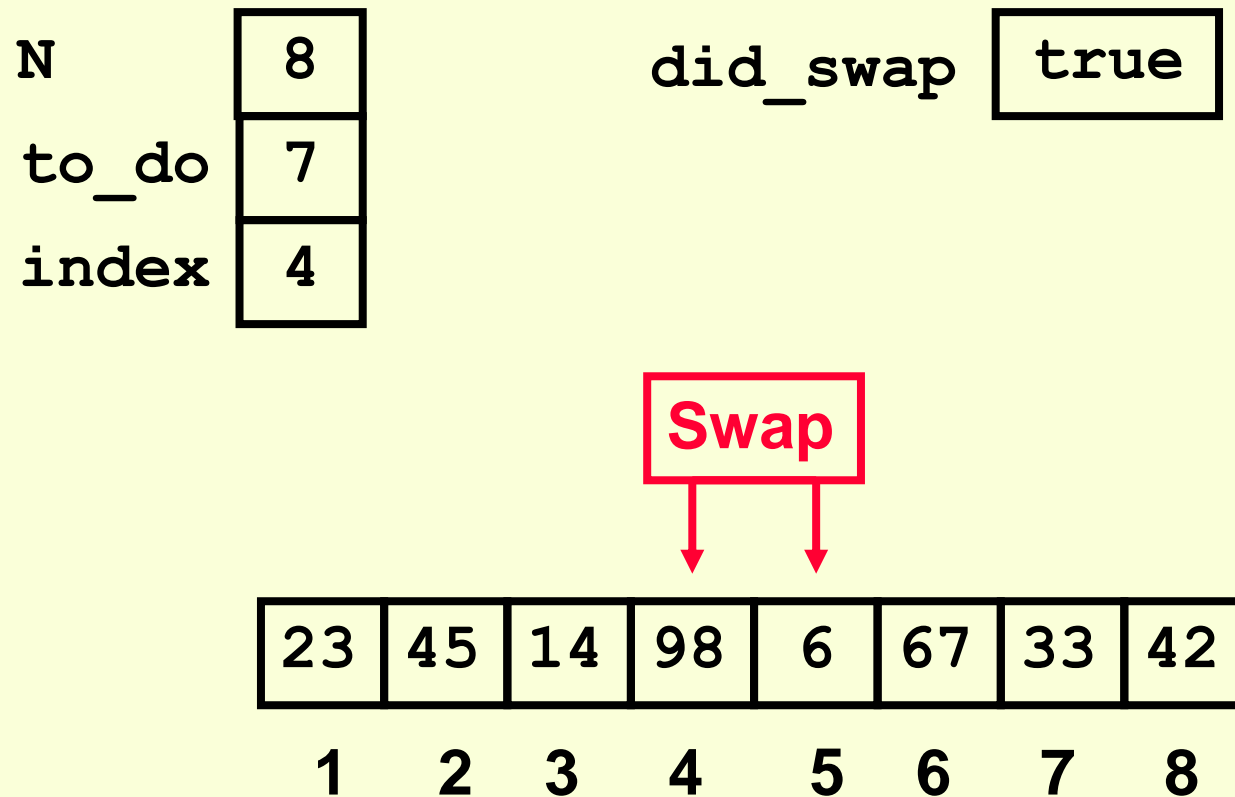
4

did_swap

true



An Animated Example



An Animated Example

N

8

to_do

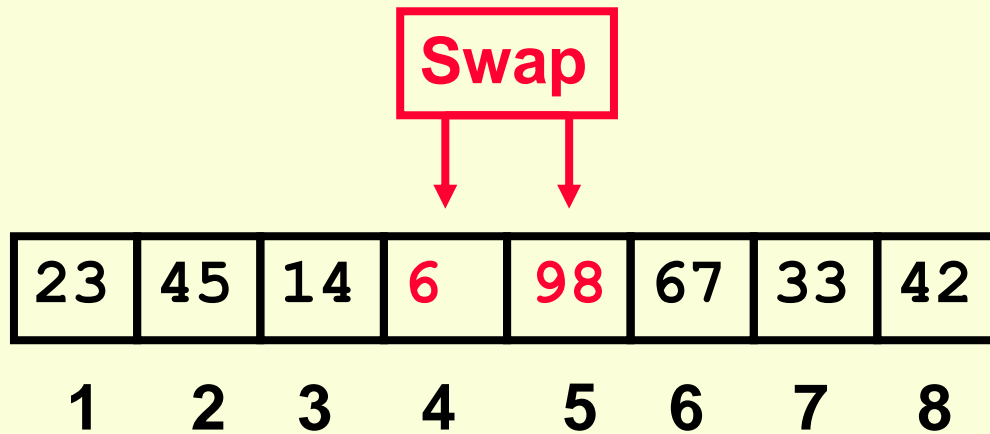
7

index

4

did_swap

true



An Animated Example

N

8

to_do

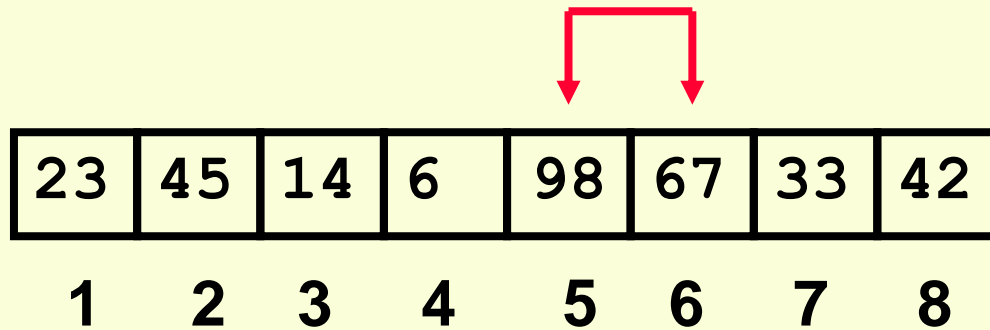
7

index

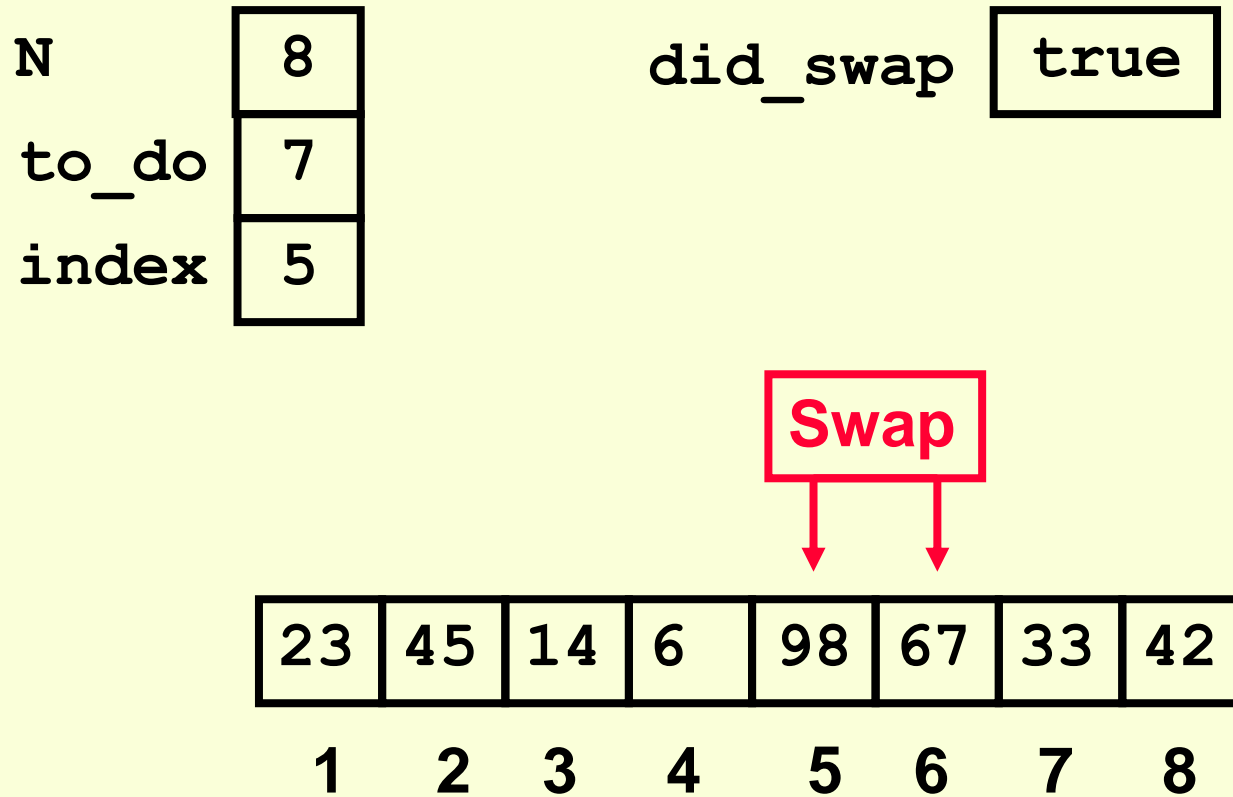
5

did_swap

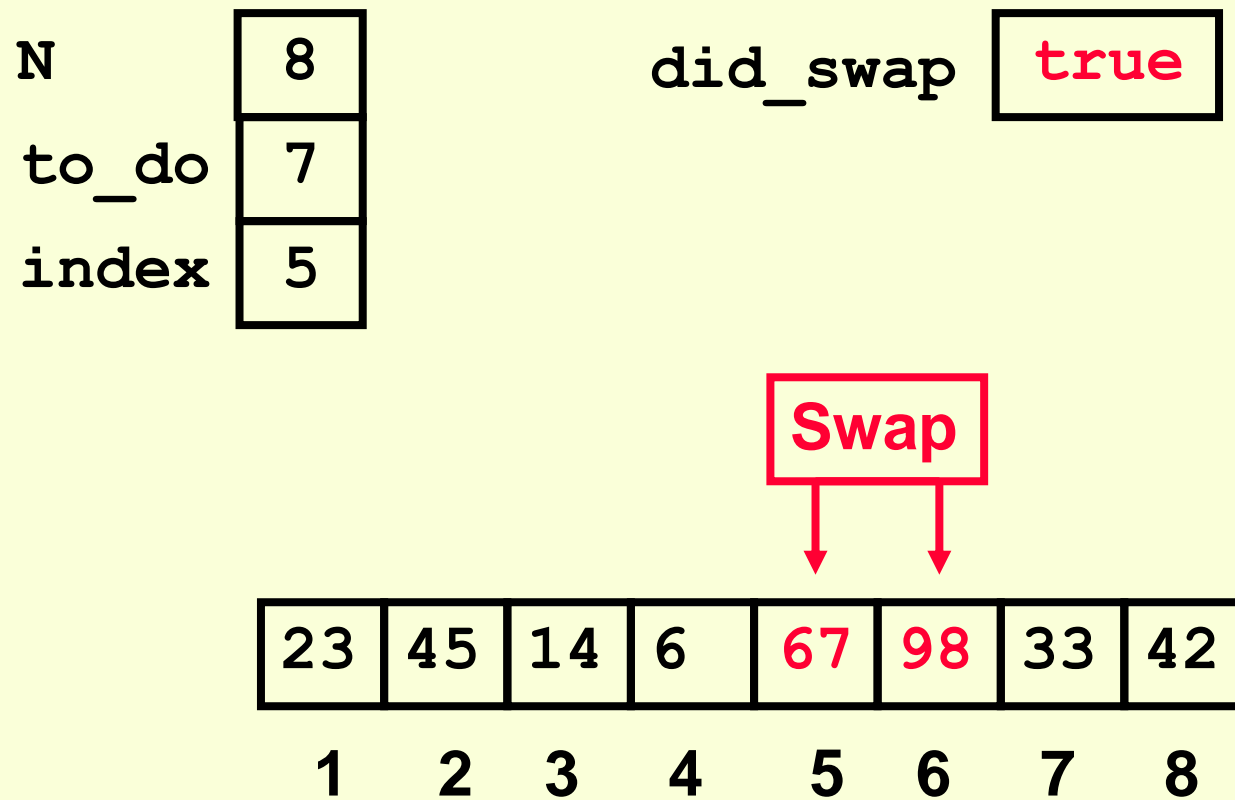
true



An Animated Example



An Animated Example



An Animated Example

N

8

to_do

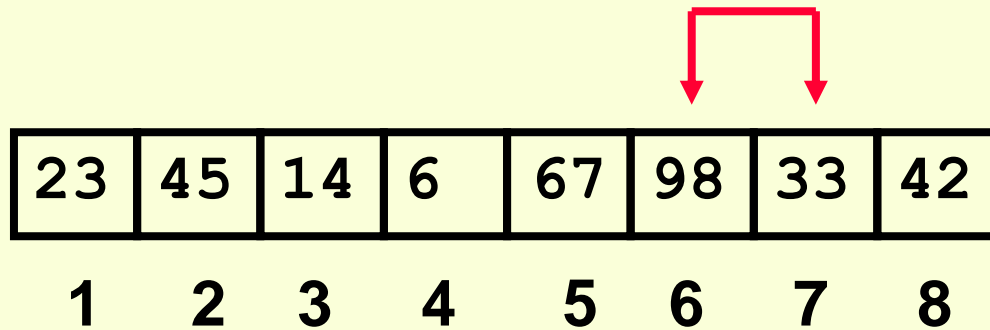
7

index

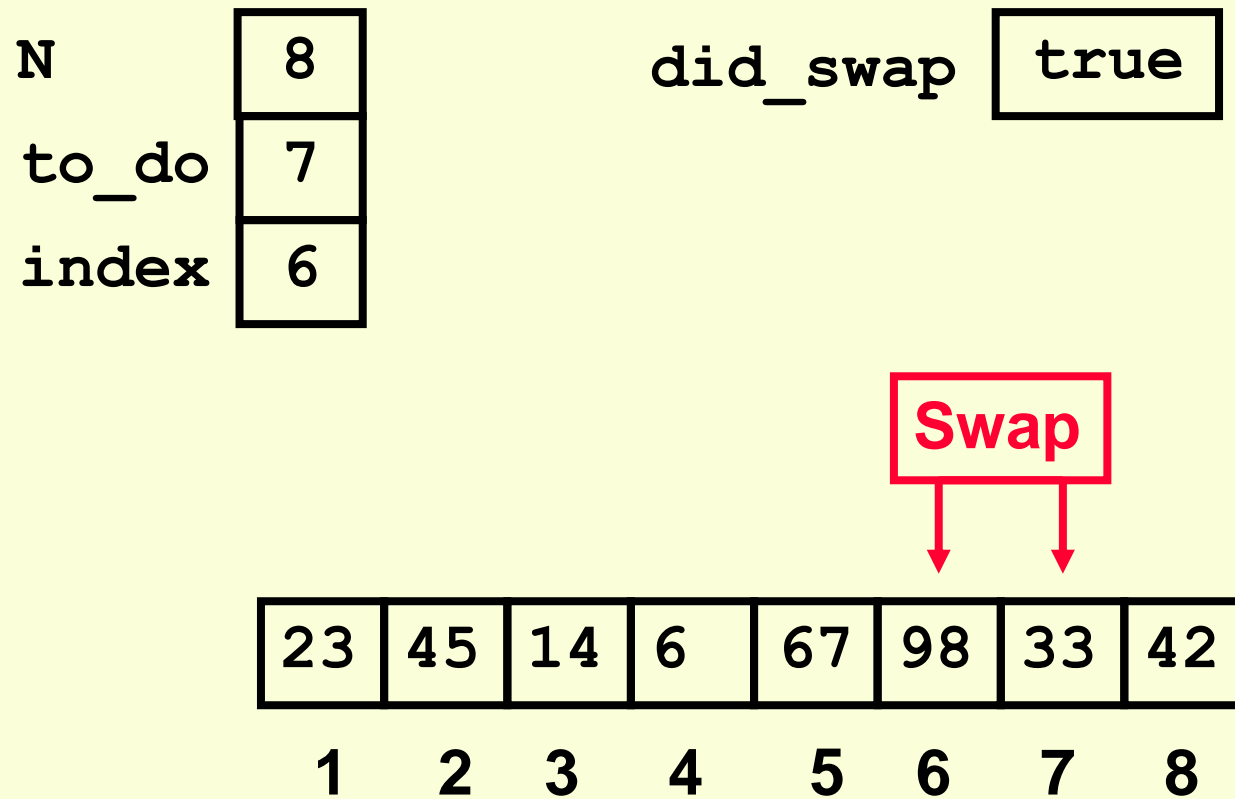
6

did_swap

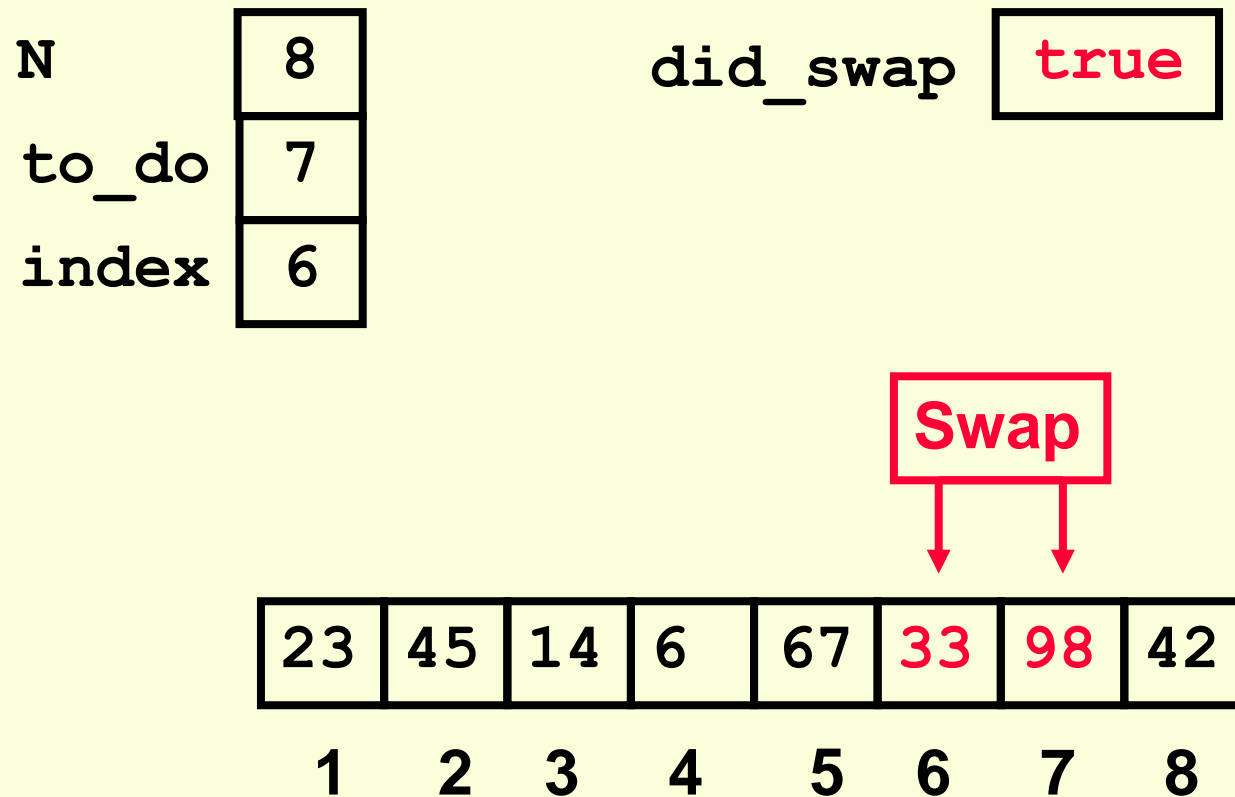
true



An Animated Example



An Animated Example



An Animated Example

N

8

 did_swap

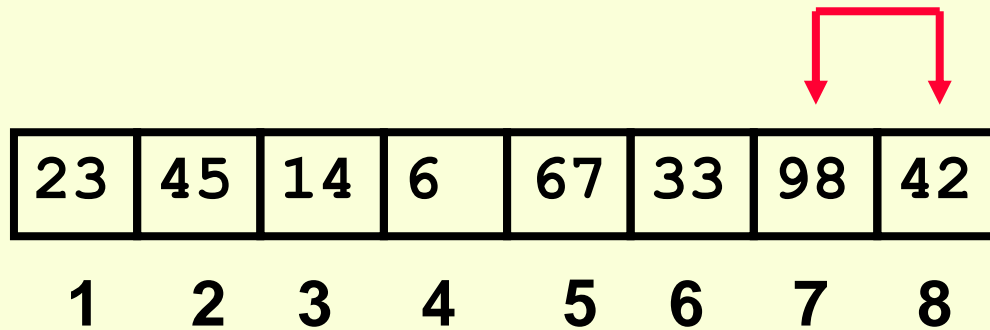
true

to_do

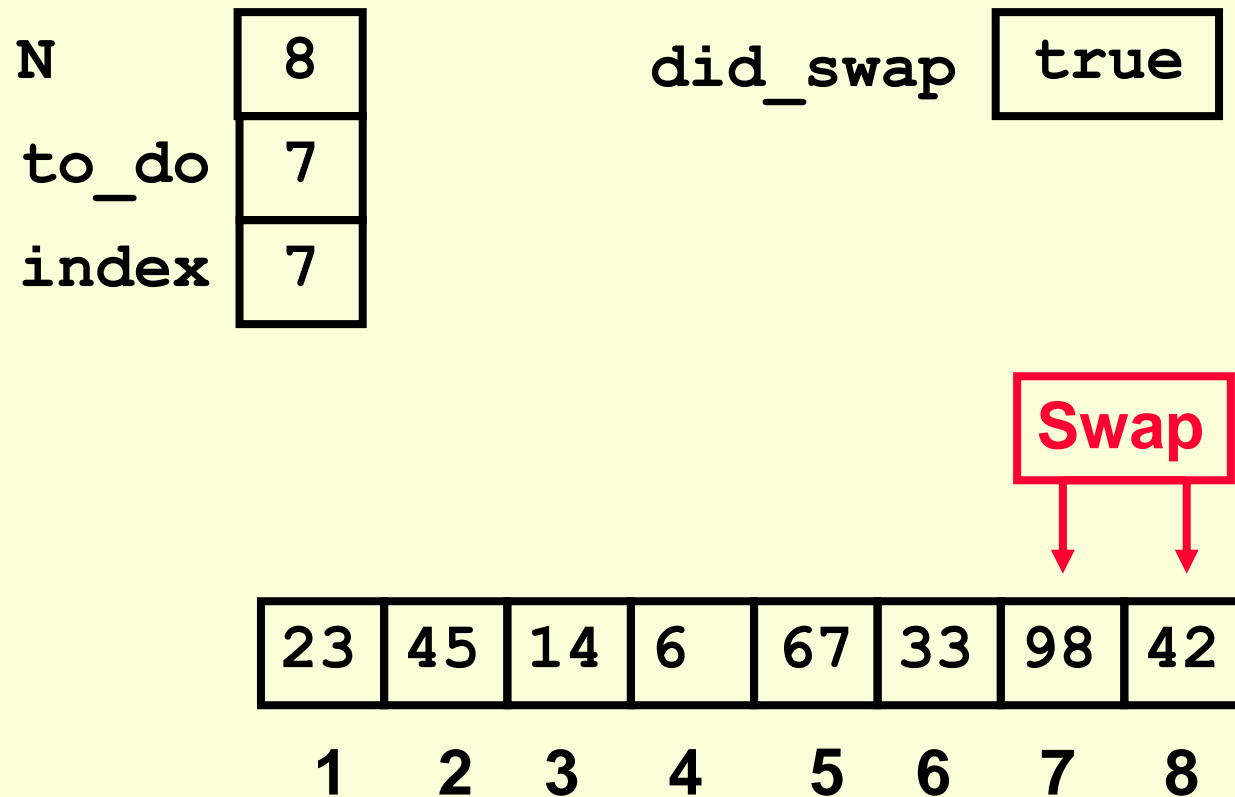
7

index

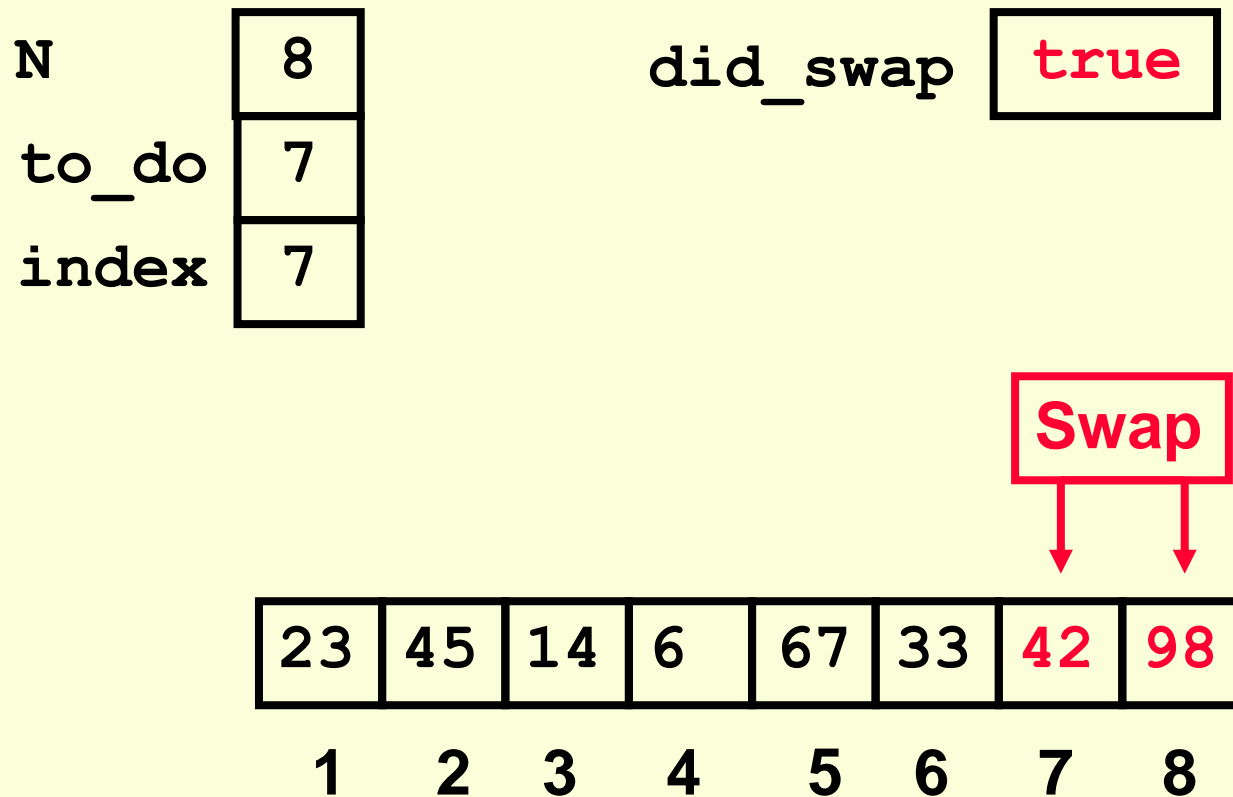
7



An Animated Example



An Animated Example



After First Pass of Outer Loop

N

8

 did_swap

true

to_do

7

index

8

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

 did_swap

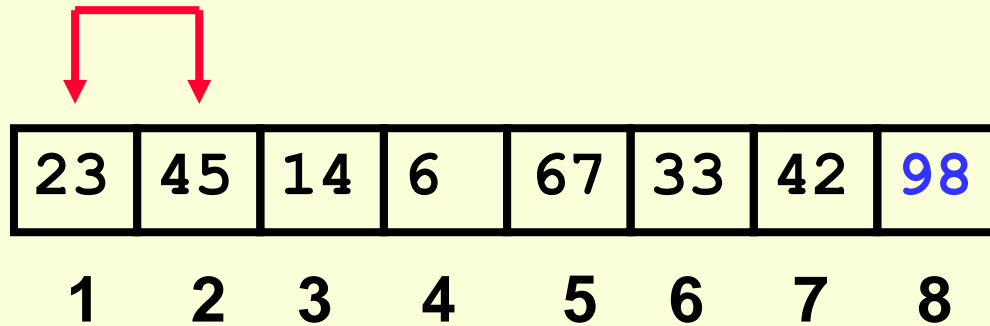
false

to_do

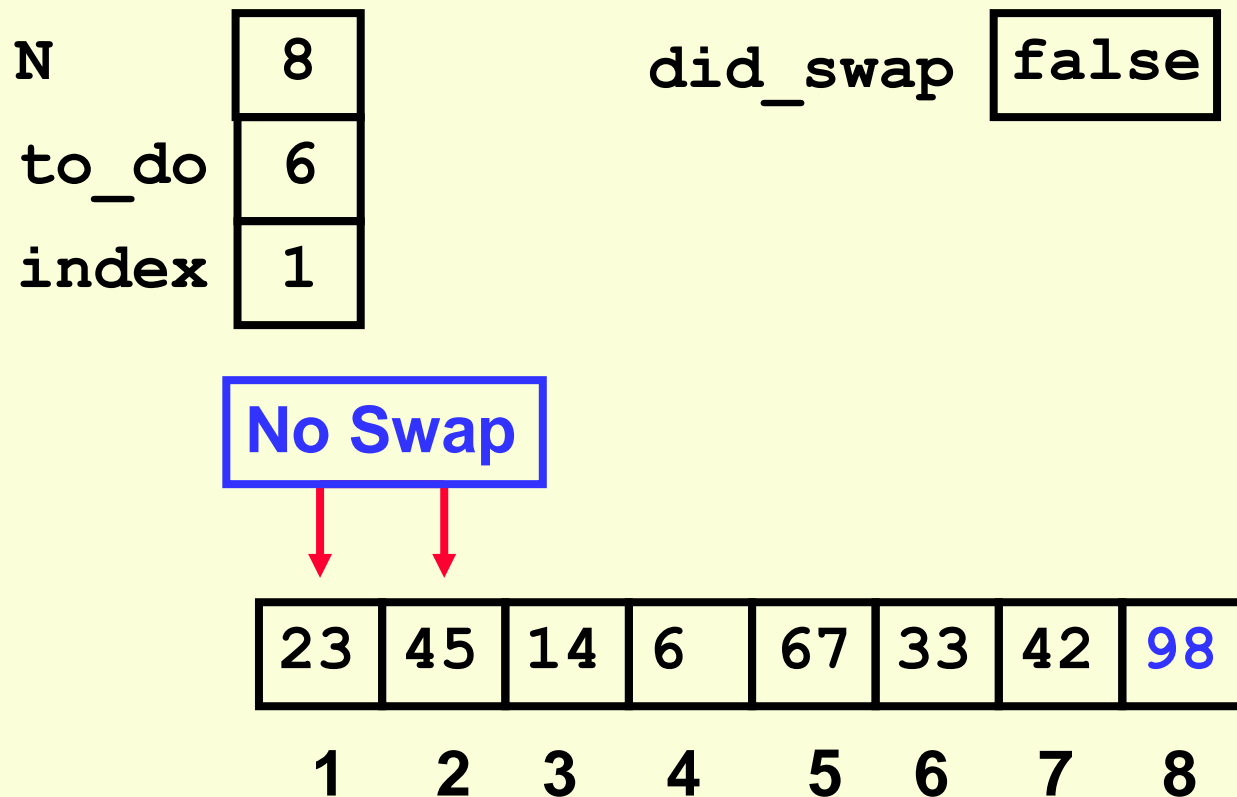
6

index

1



The Second “Bubble Up”



The Second “Bubble Up”

N

8

 did_swap

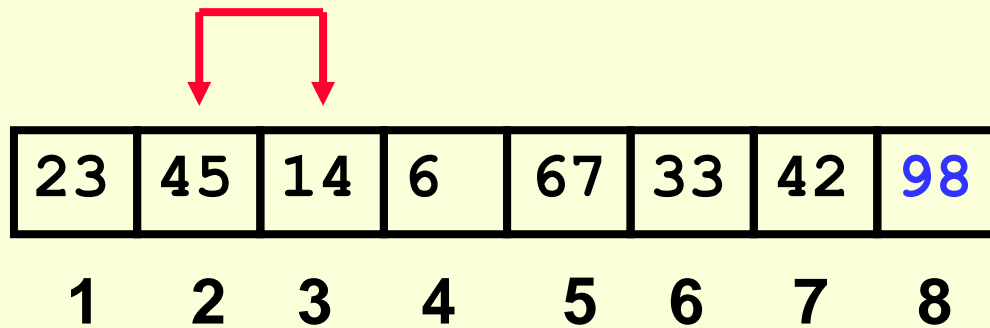
false

to_do

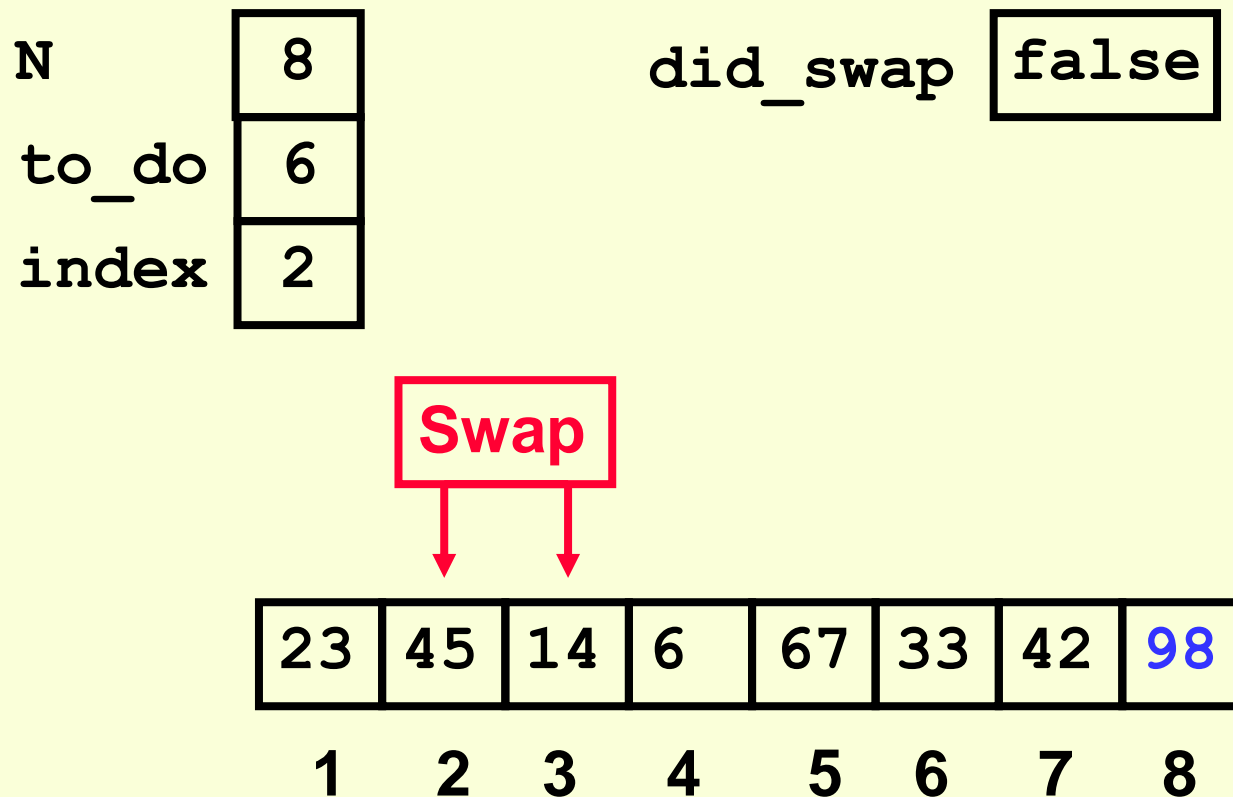
6

index

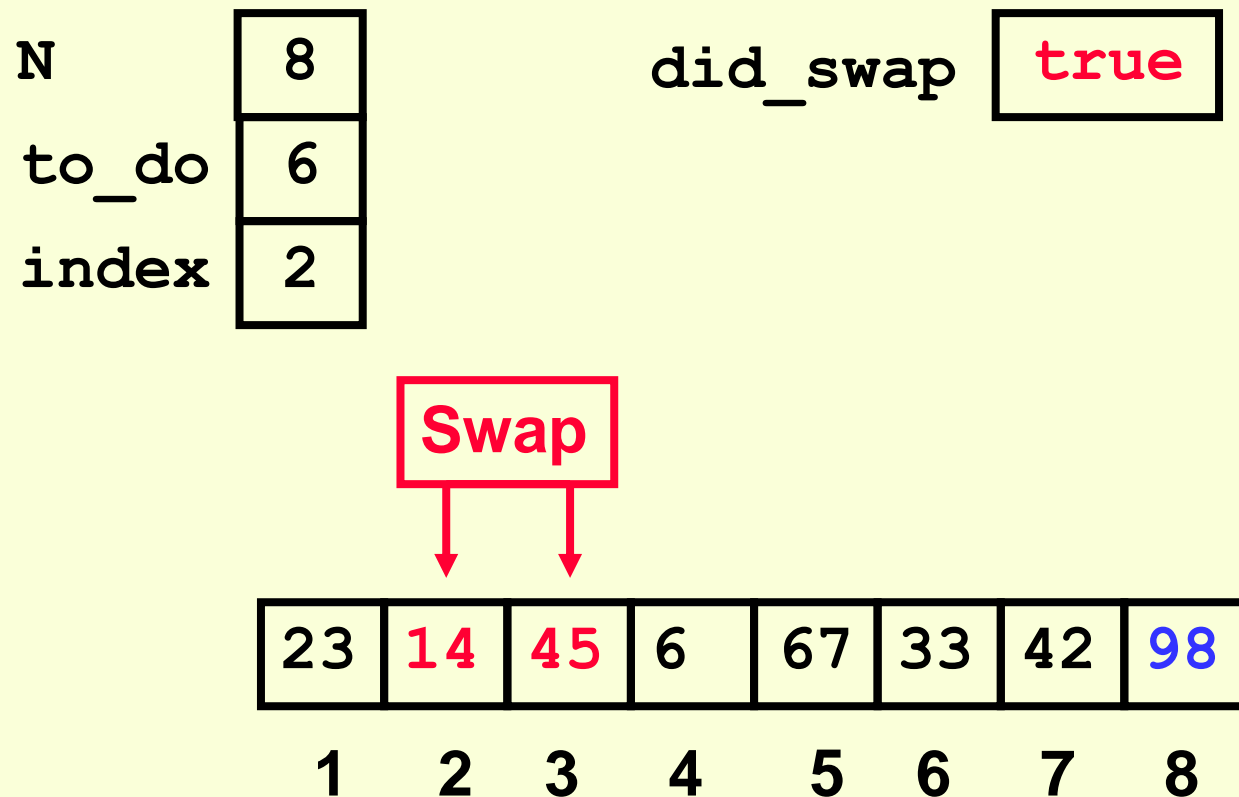
2



The Second “Bubble Up”



The Second “Bubble Up”



The Second “Bubble Up”

N

8

 did_swap

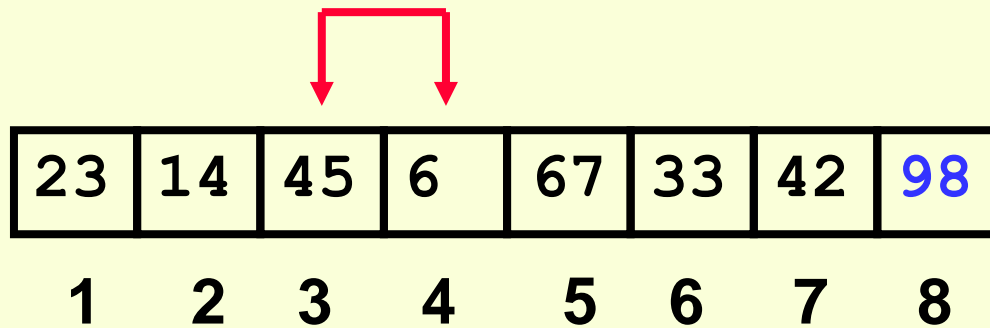
true

to_do

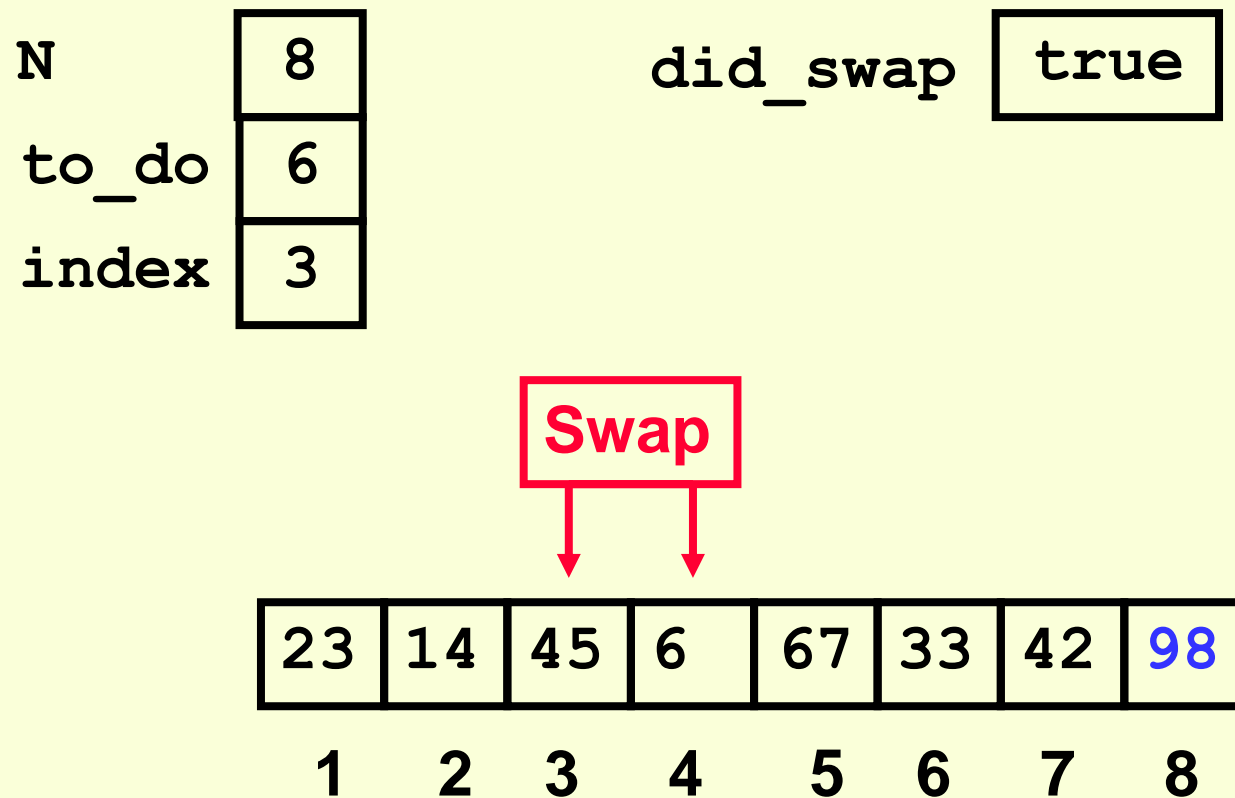
6

index

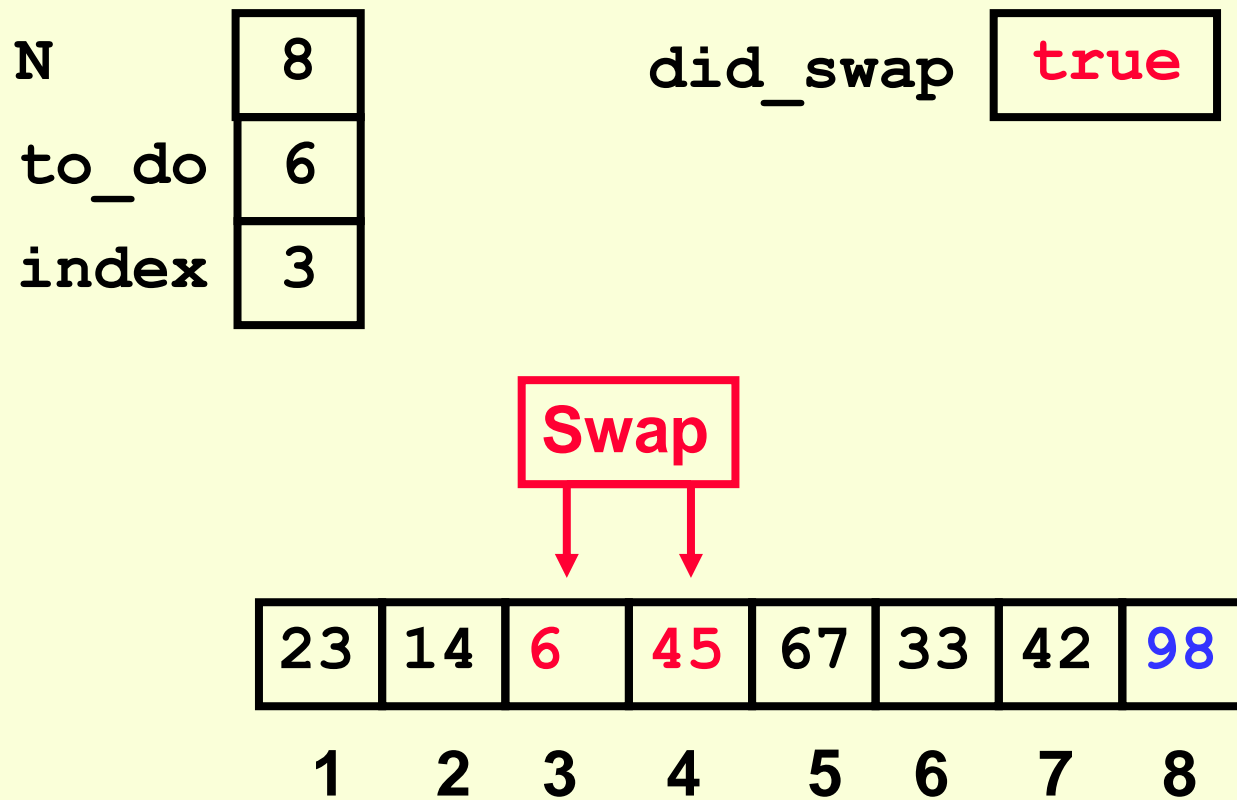
3



The Second “Bubble Up”



The Second “Bubble Up”



The Second “Bubble Up”

N

8

 did_swap

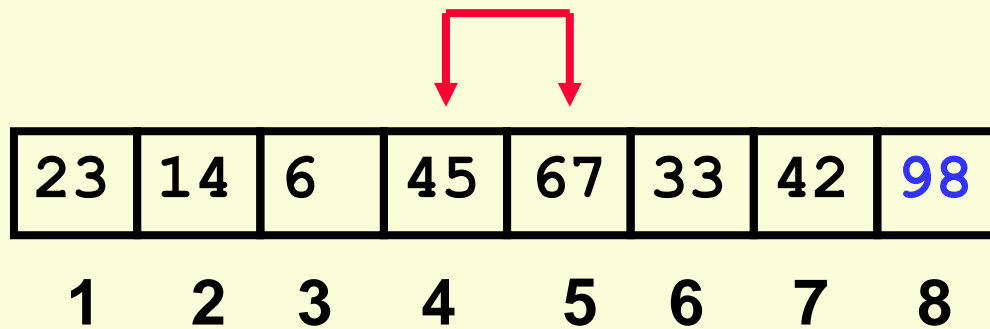
true

to_do

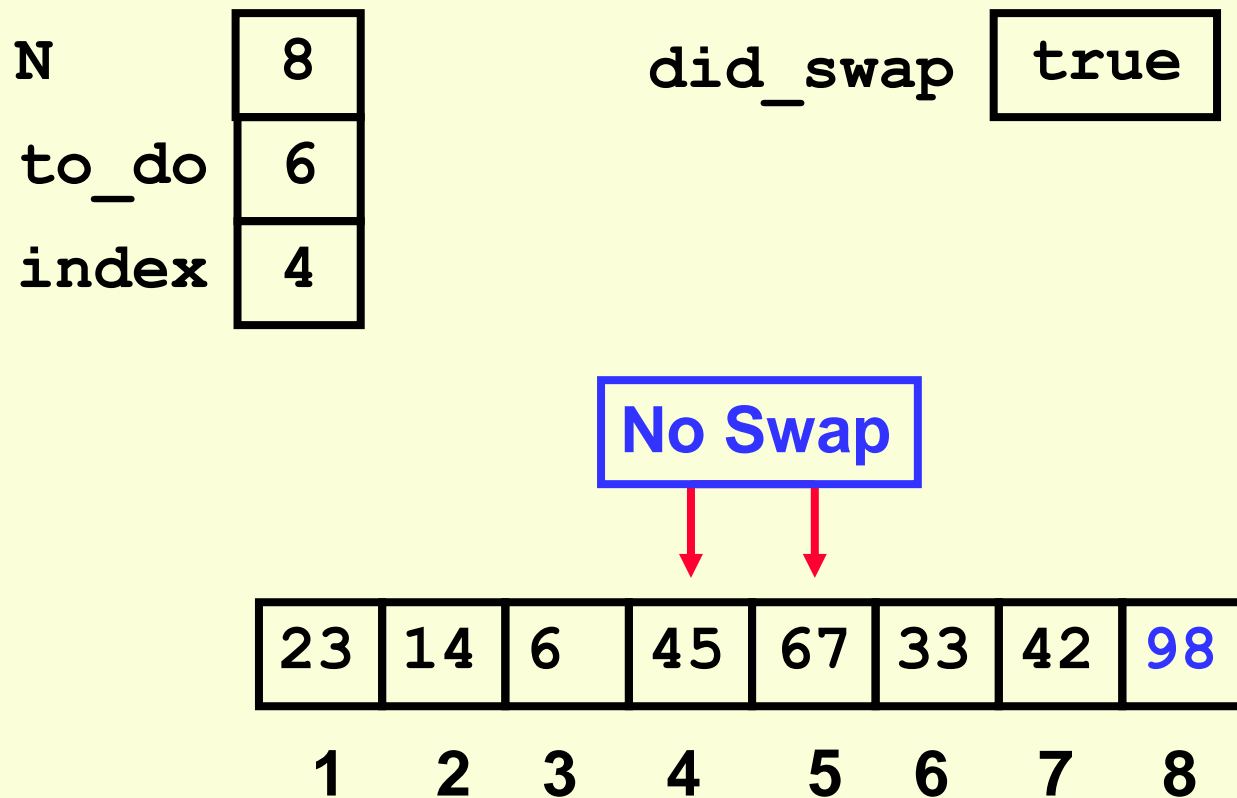
6

index

4



The Second “Bubble Up”



The Second “Bubble Up”

N

8

to_do

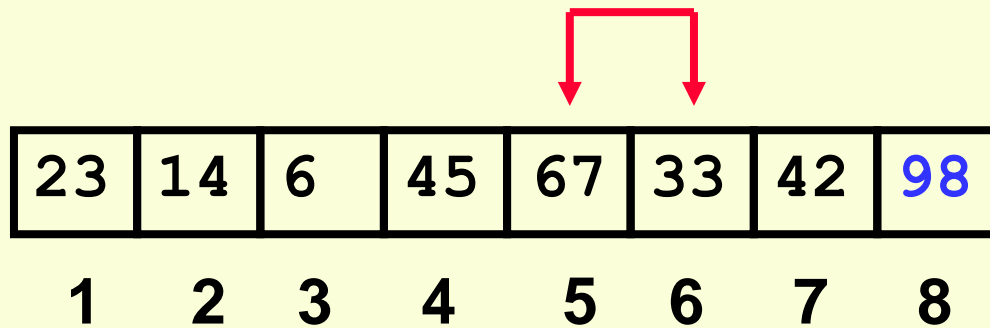
6

index

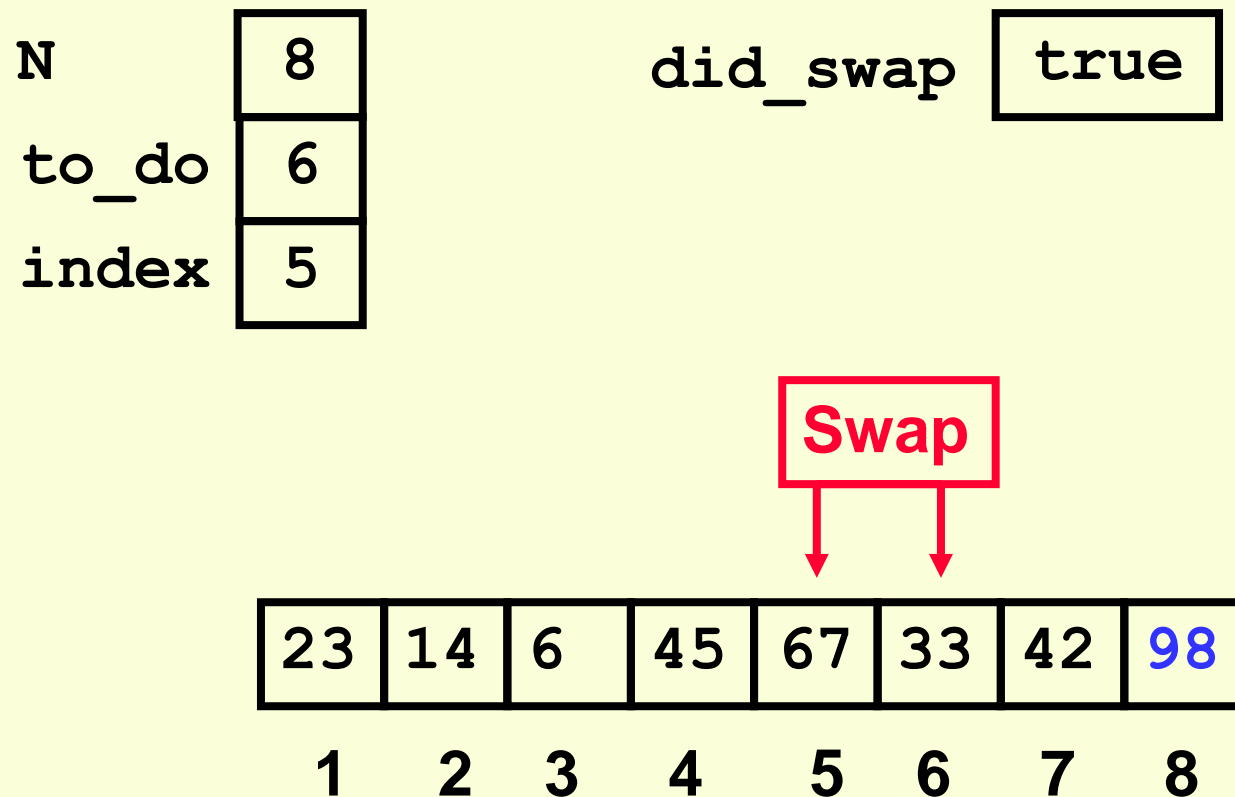
5

did_swap

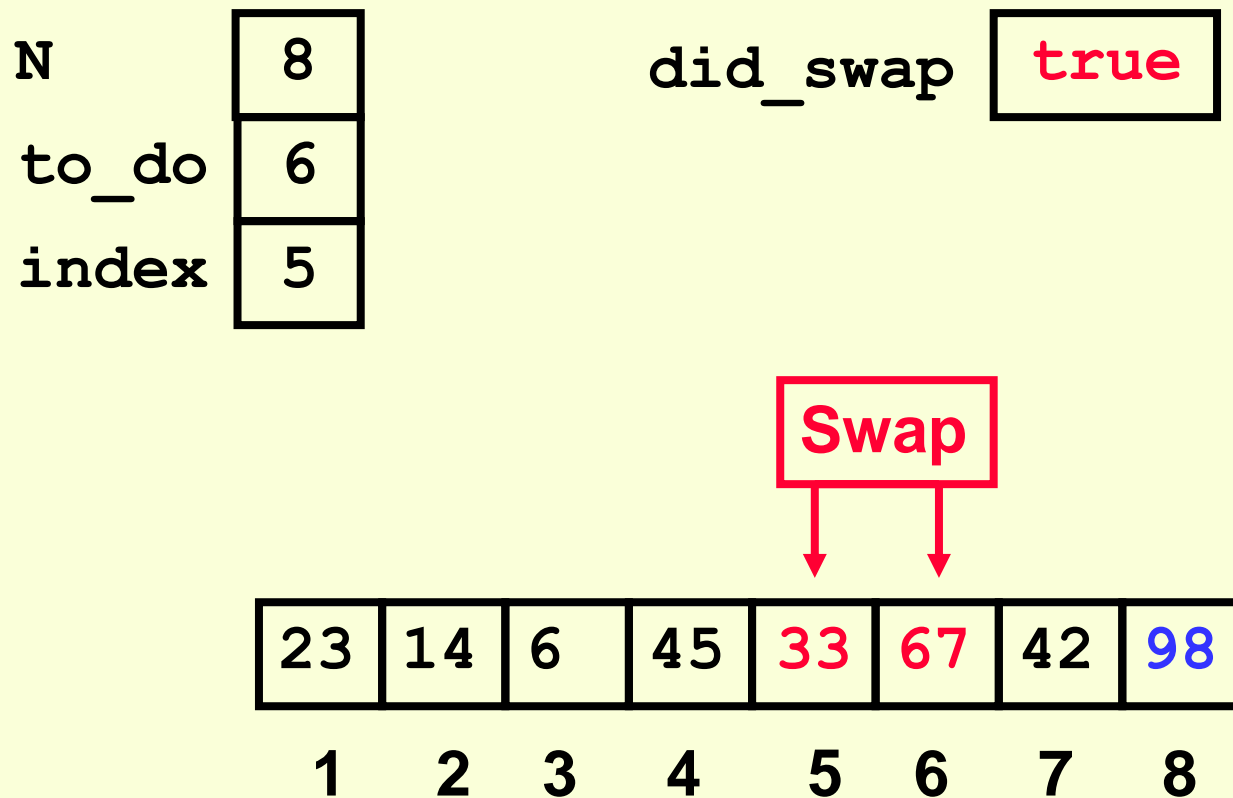
true



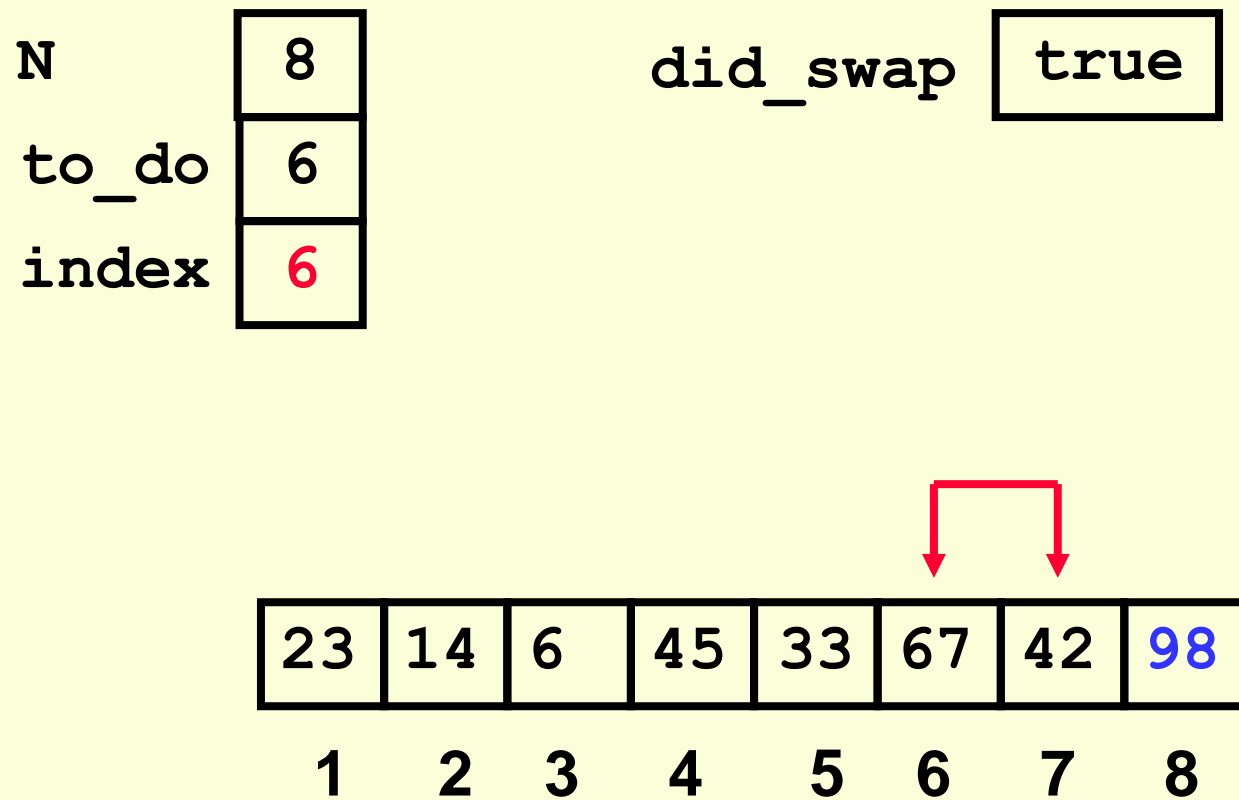
The Second “Bubble Up”



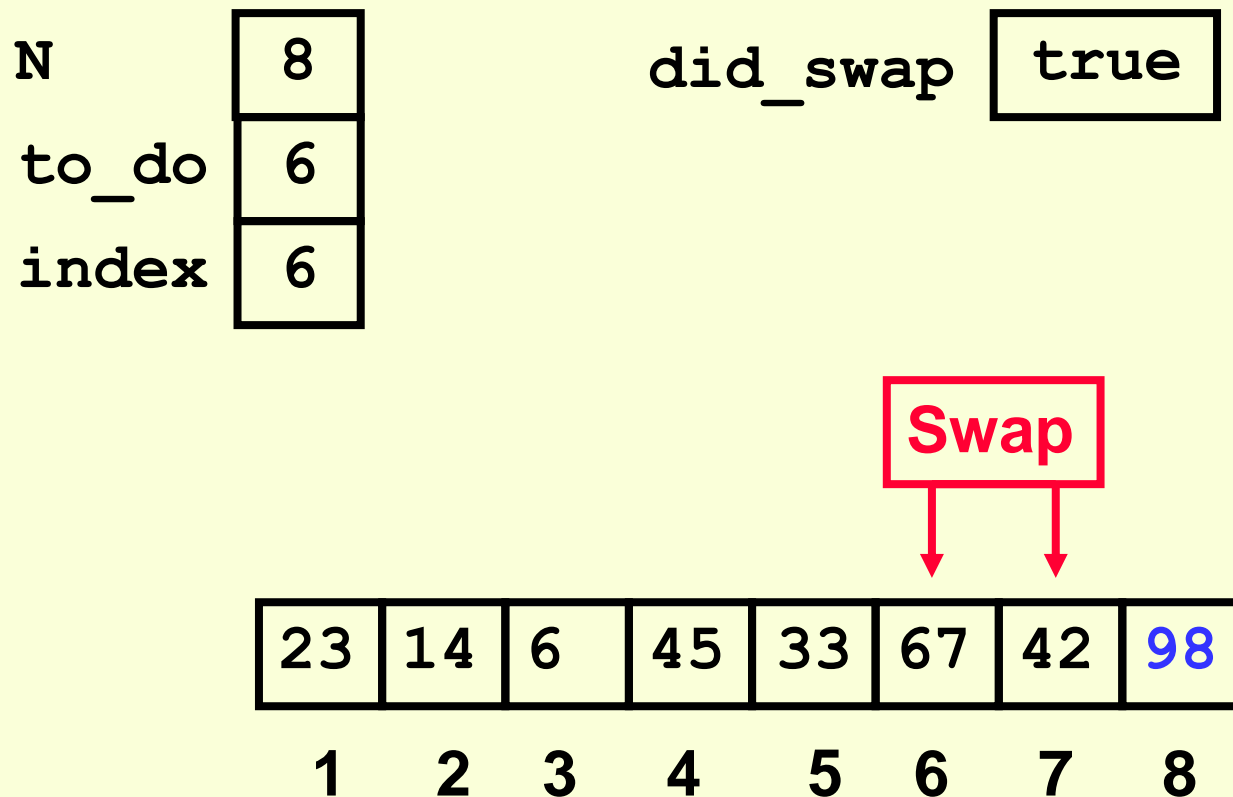
The Second “Bubble Up”



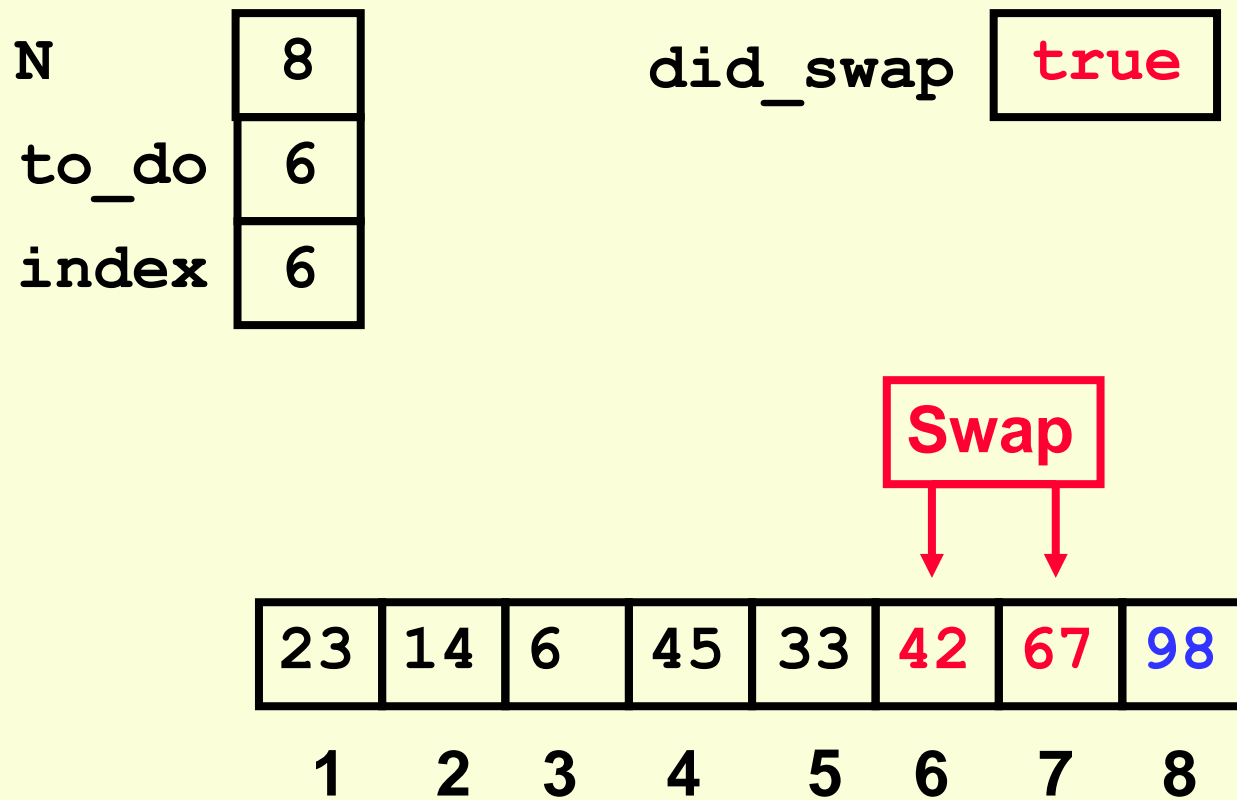
The Second “Bubble Up”



The Second “Bubble Up”



The Second “Bubble Up”



After Second Pass of Outer Loop

N

8

to_do

6

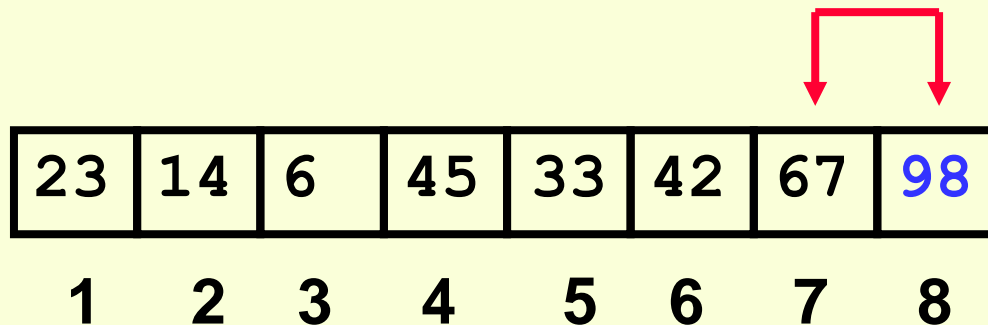
index

7

did_swap

true

Finished second “Bubble Up”



The Third “Bubble Up”

N

8

 did_swap

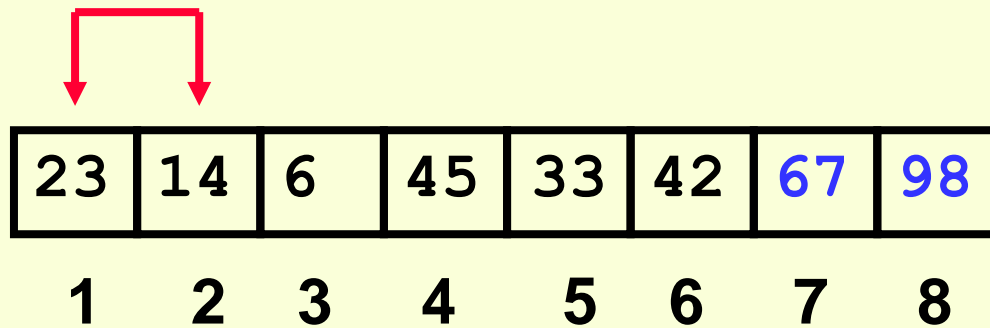
false

to_do

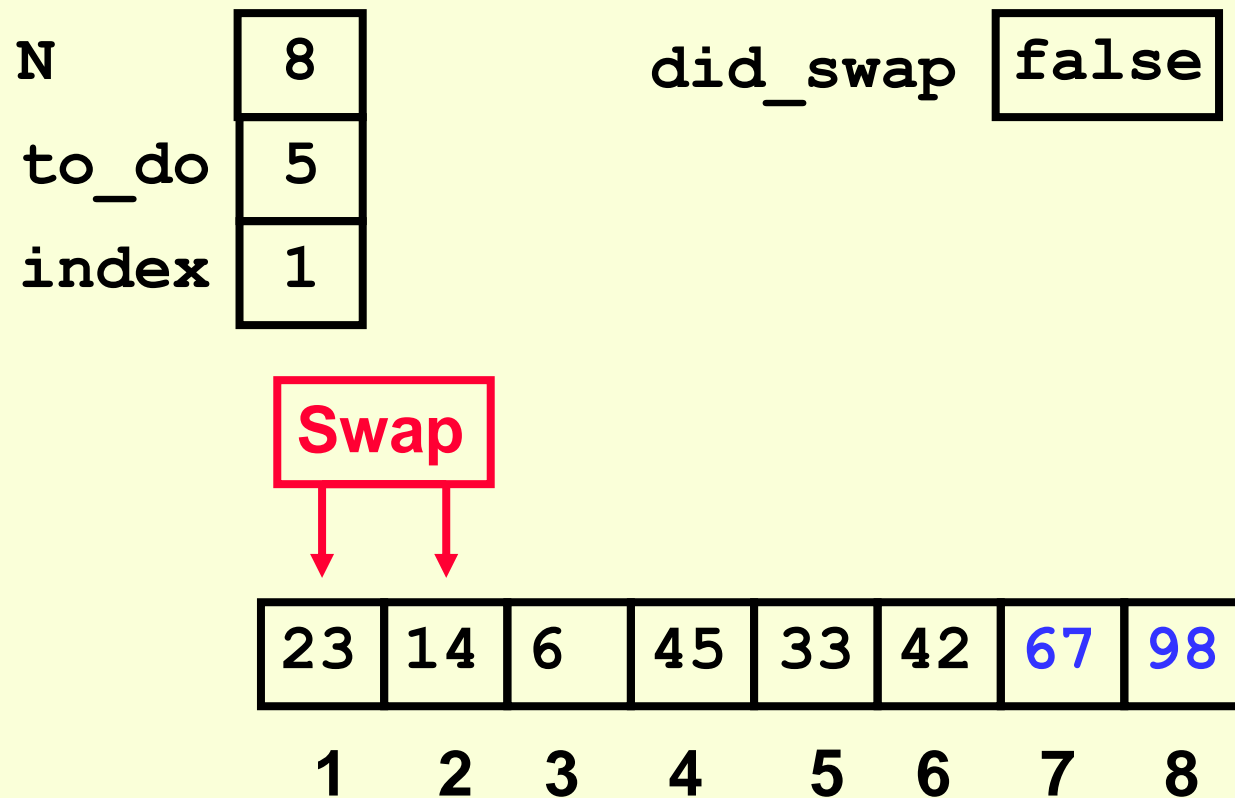
5

index

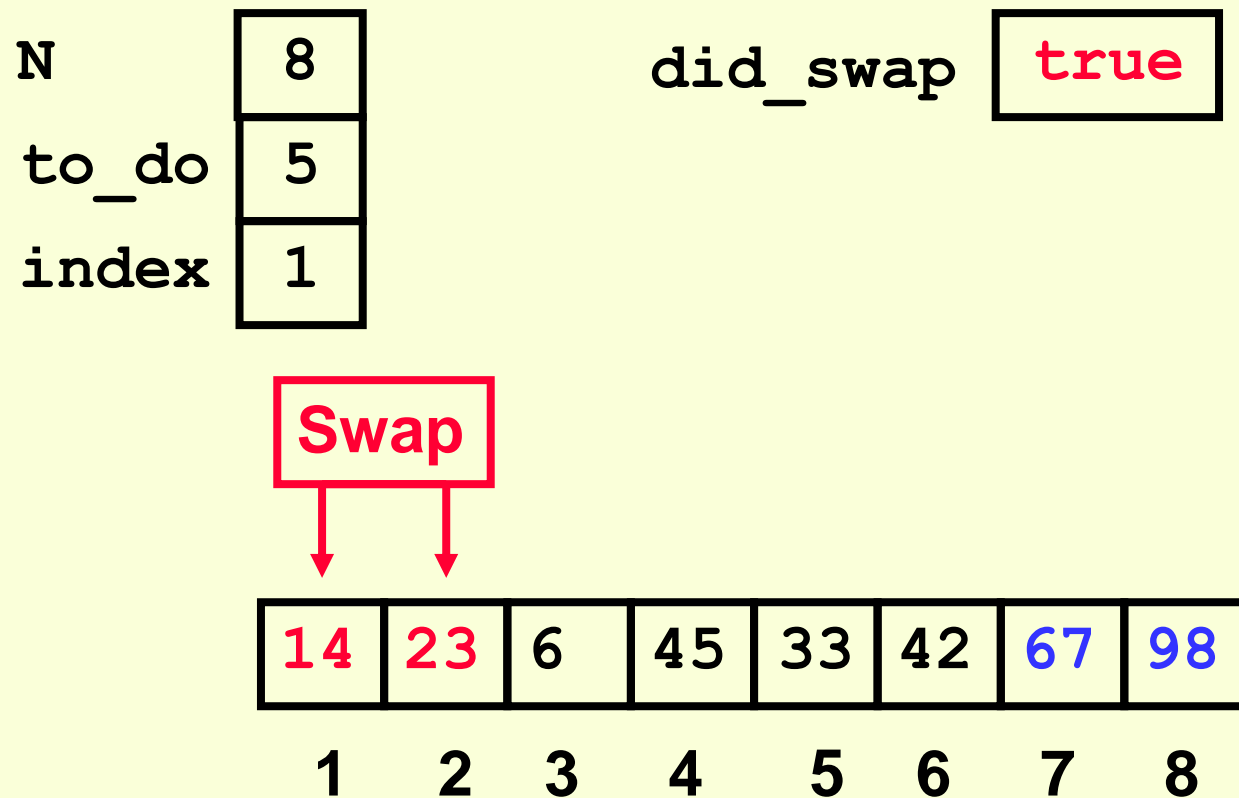
1



The Third “Bubble Up”



The Third “Bubble Up”



The Third “Bubble Up”

N

8

 did_swap

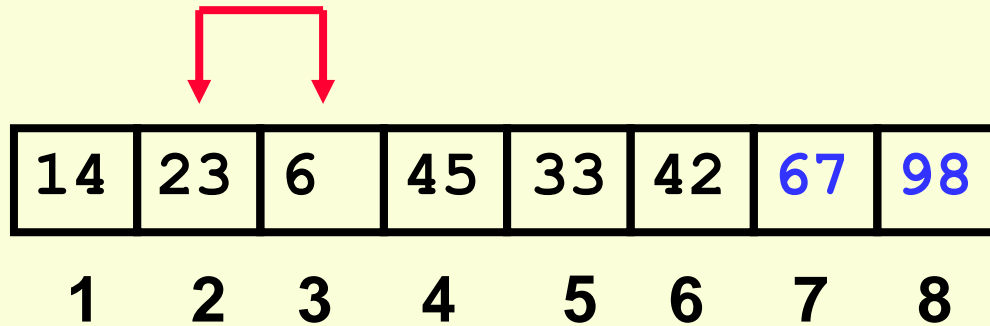
true

to_do

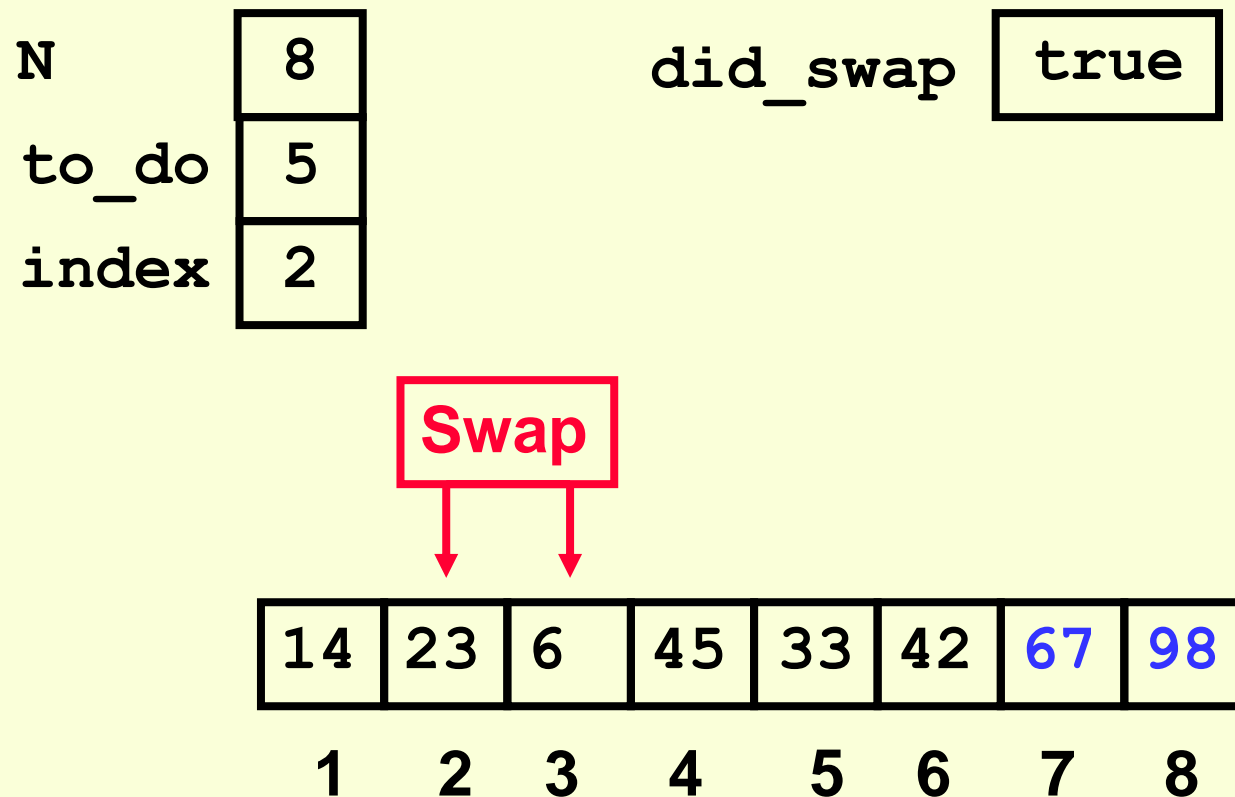
5

index

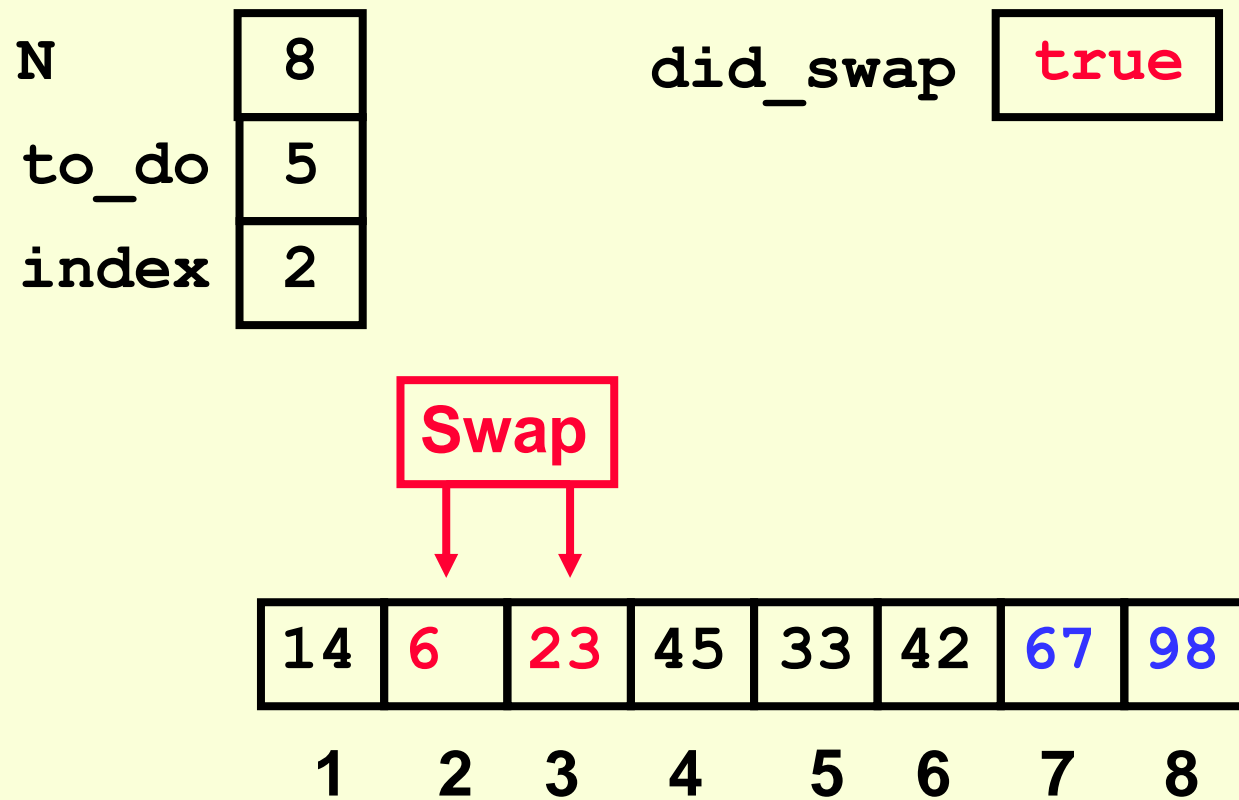
2



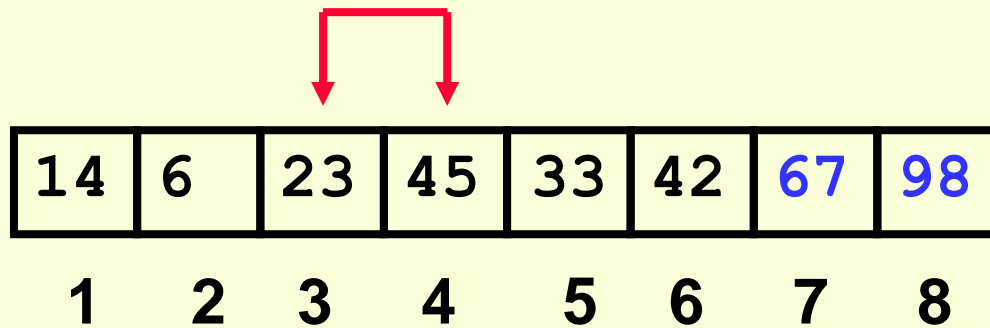
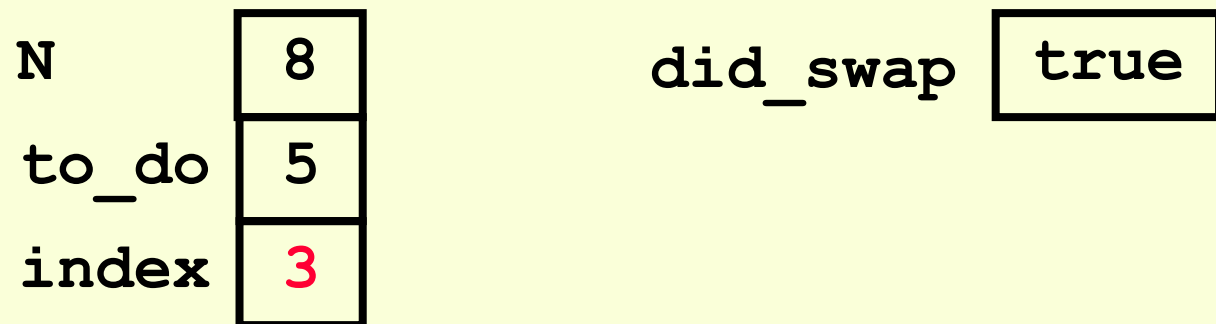
The Third “Bubble Up”



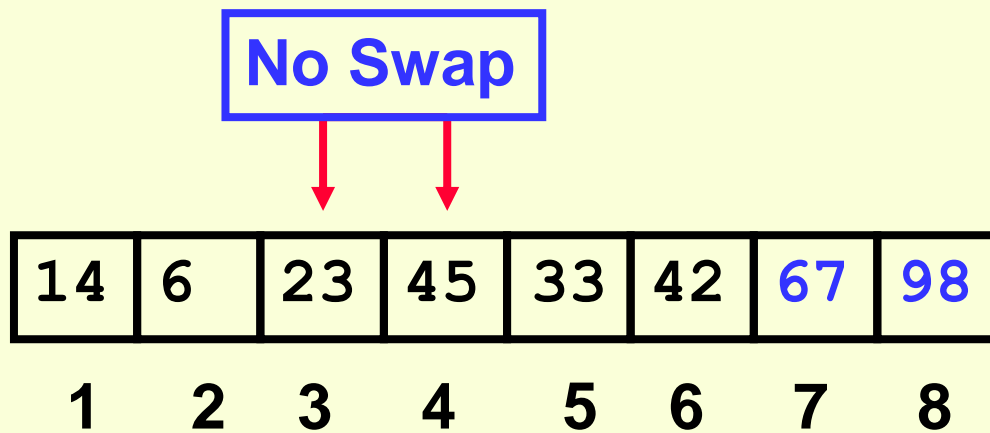
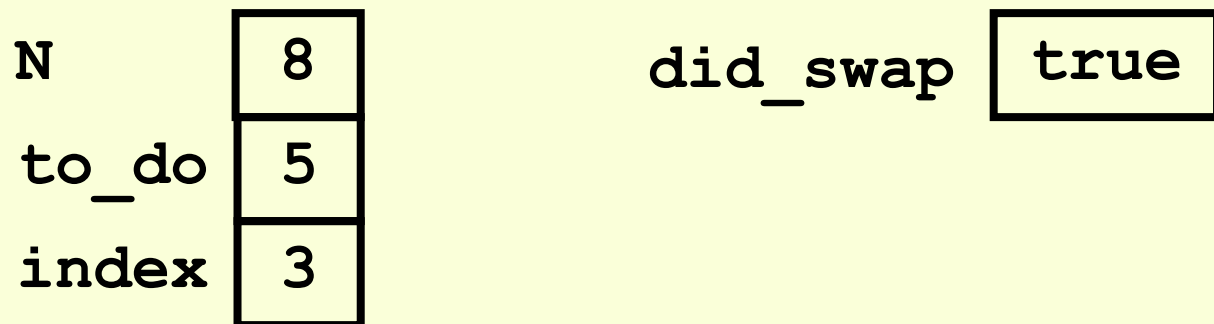
The Third “Bubble Up”



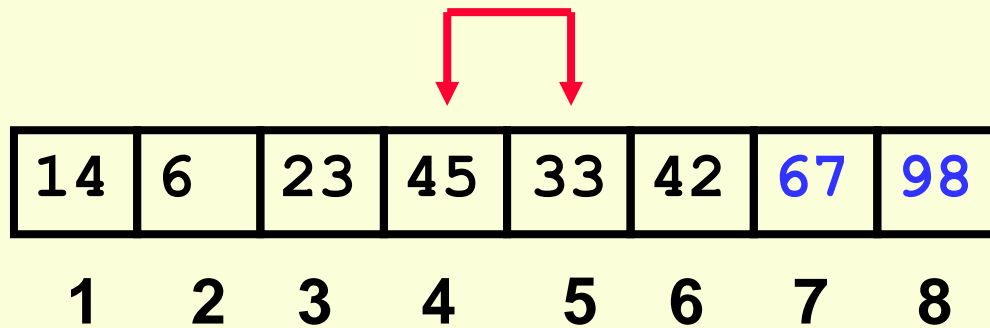
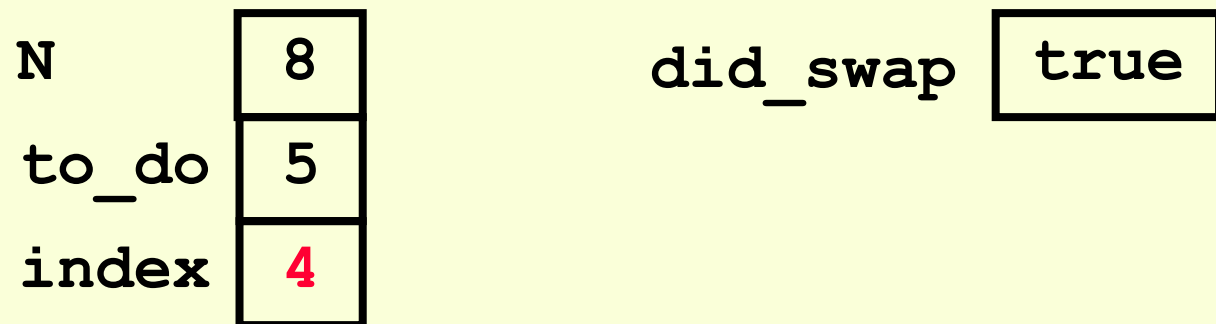
The Third “Bubble Up”



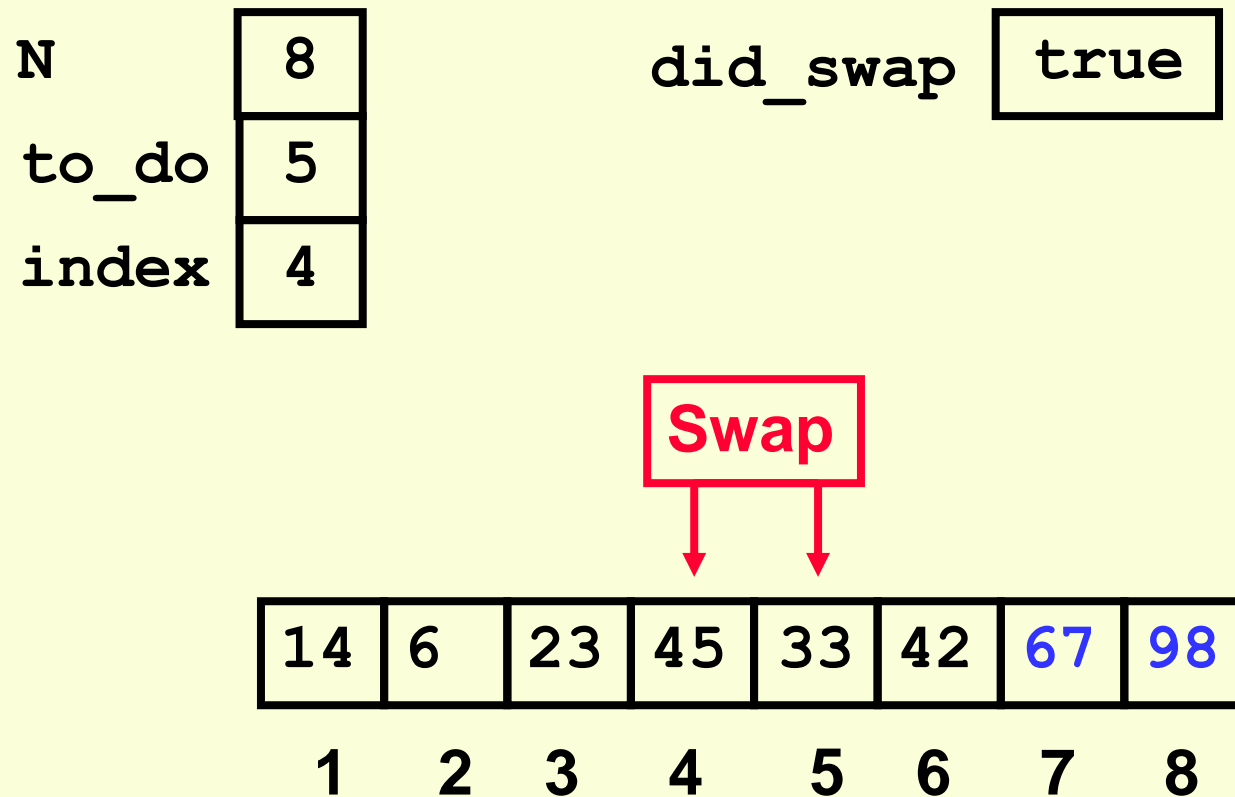
The Third “Bubble Up”



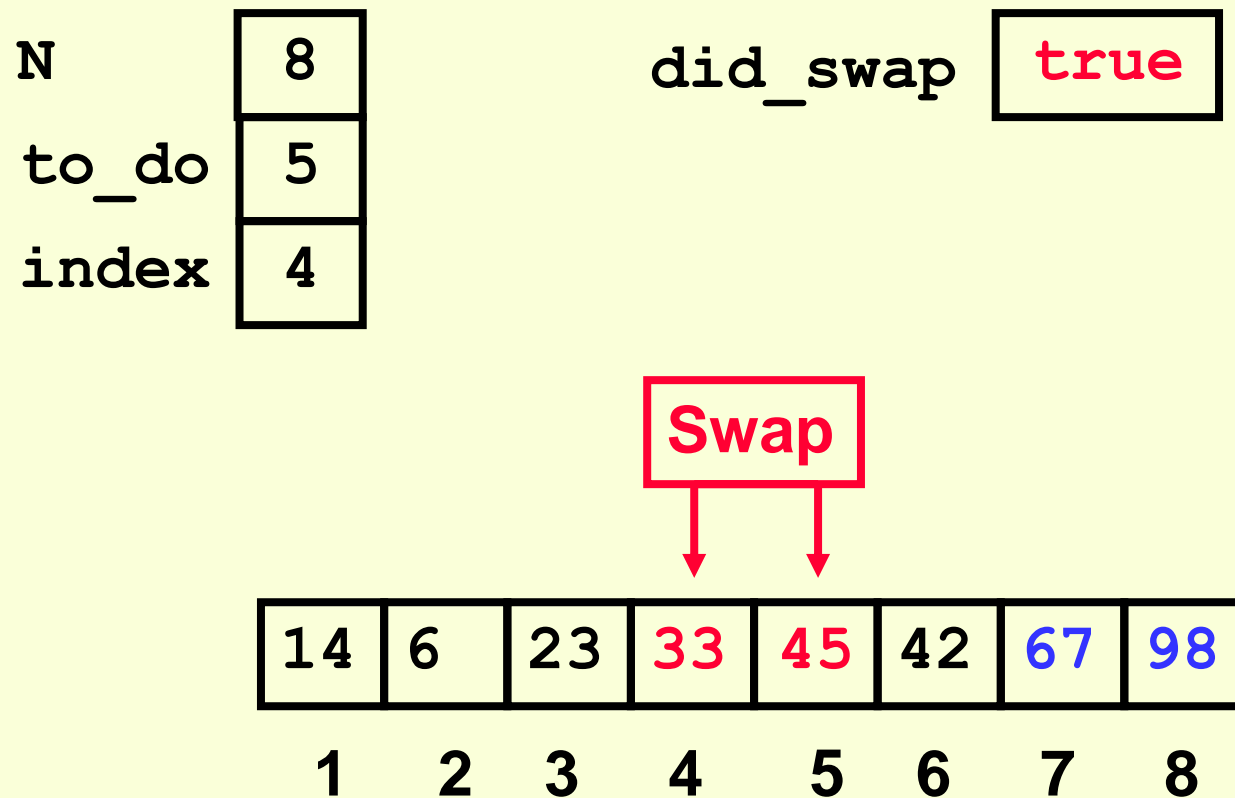
The Third “Bubble Up”



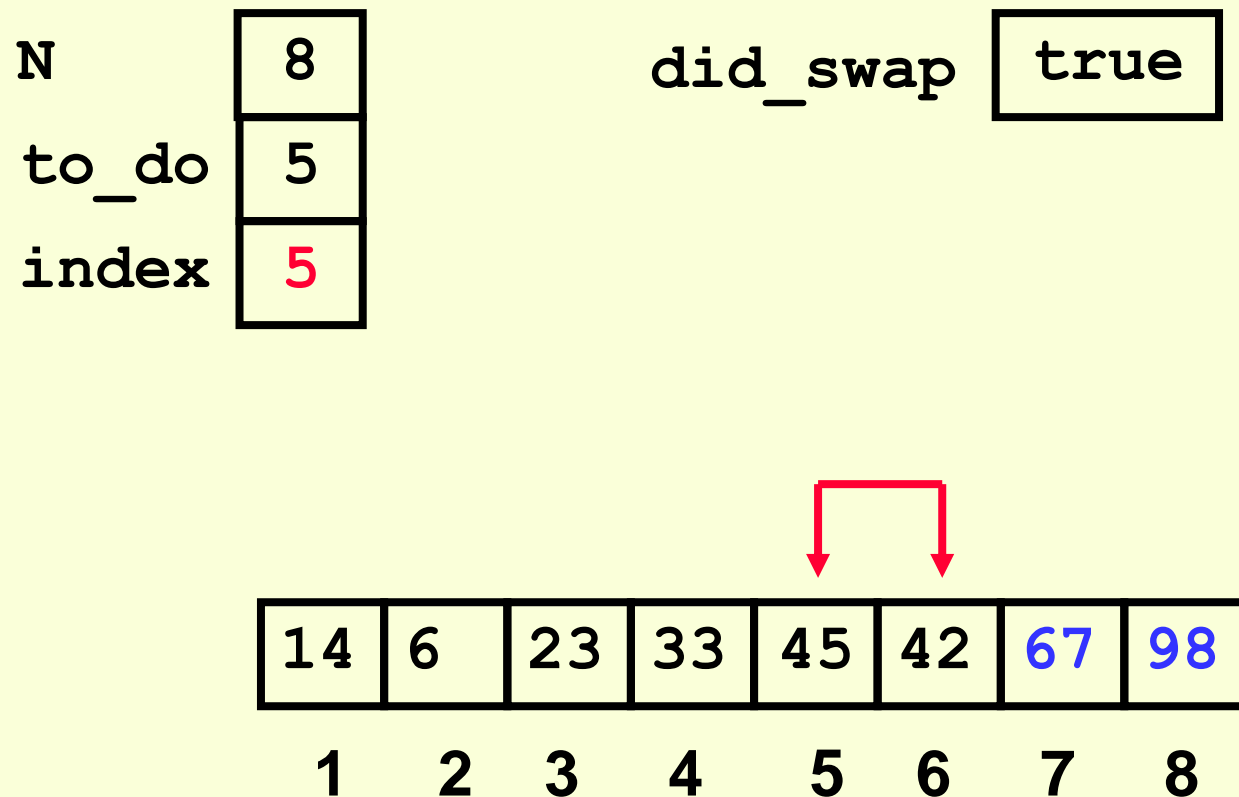
The Third “Bubble Up”



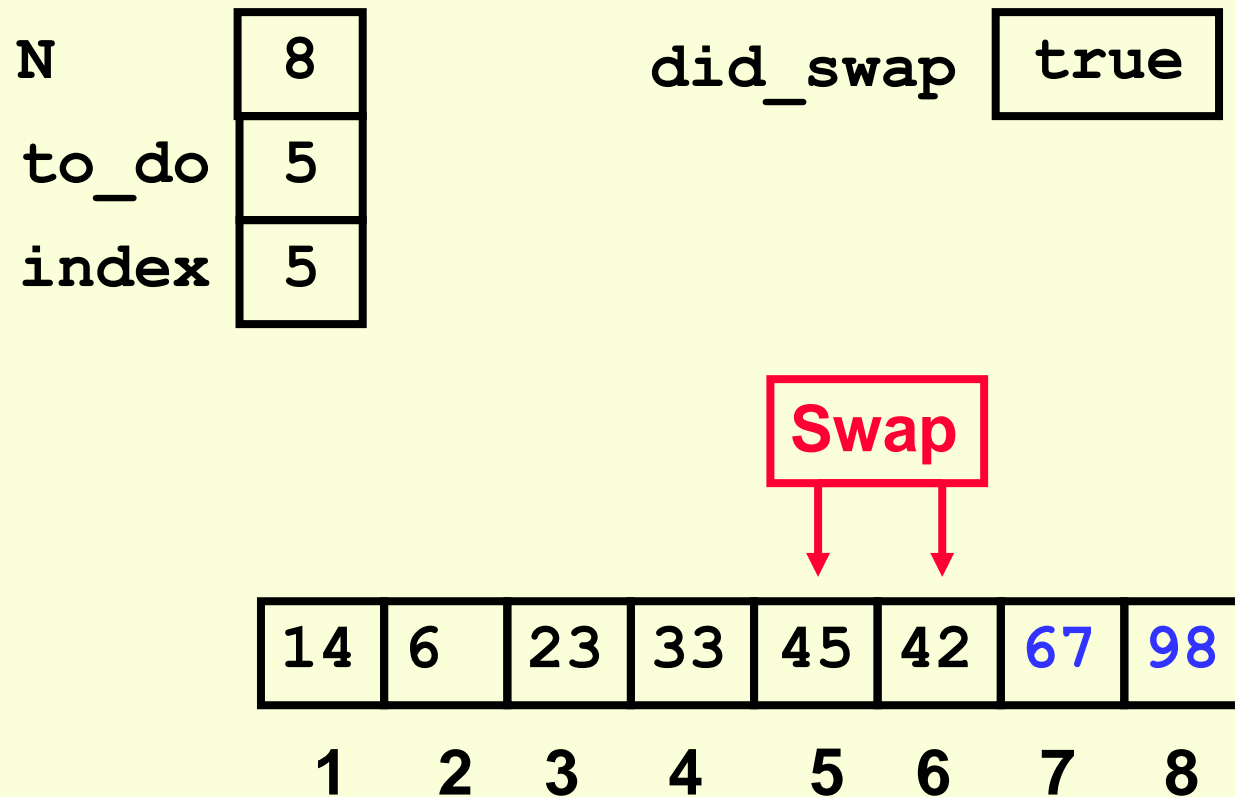
The Third “Bubble Up”



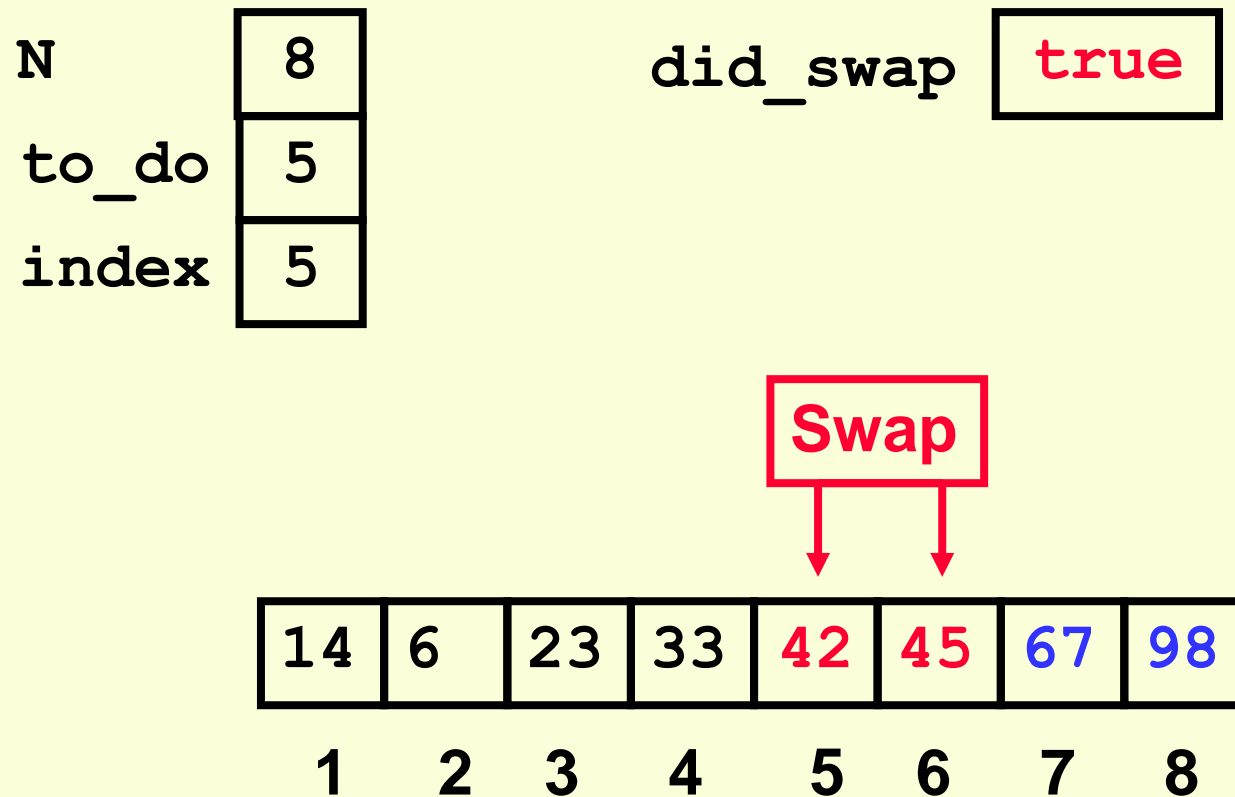
The Third “Bubble Up”



The Third “Bubble Up”



The Third “Bubble Up”



After Third Pass of Outer Loop

N

8

 did_swap

true

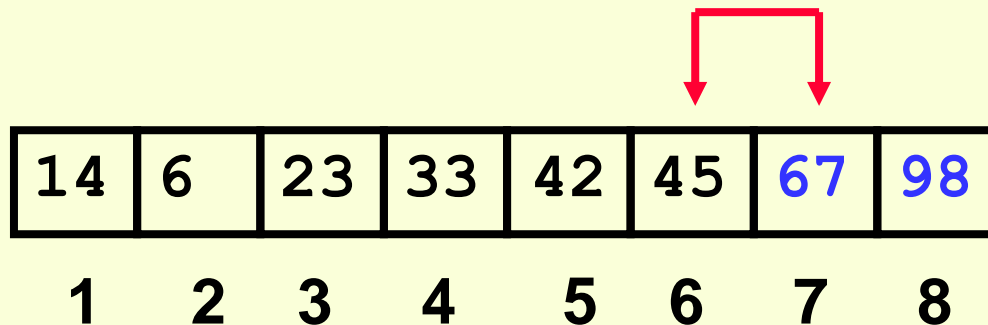
to_do

5

index

6

Finished third “Bubble Up”



The Fourth “Bubble Up”

N

8

 did_swap

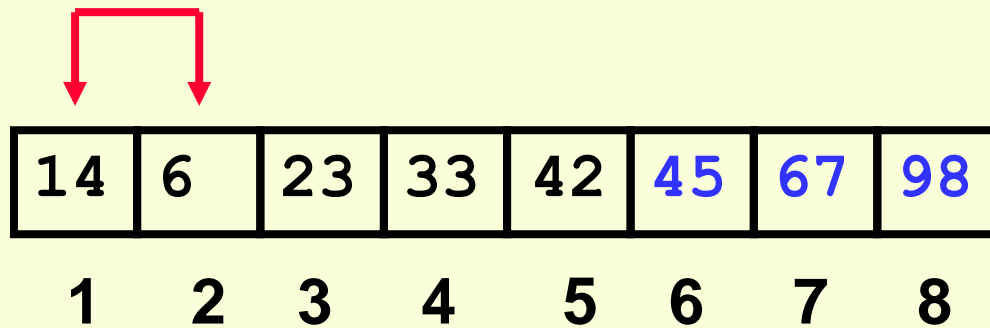
false

to_do

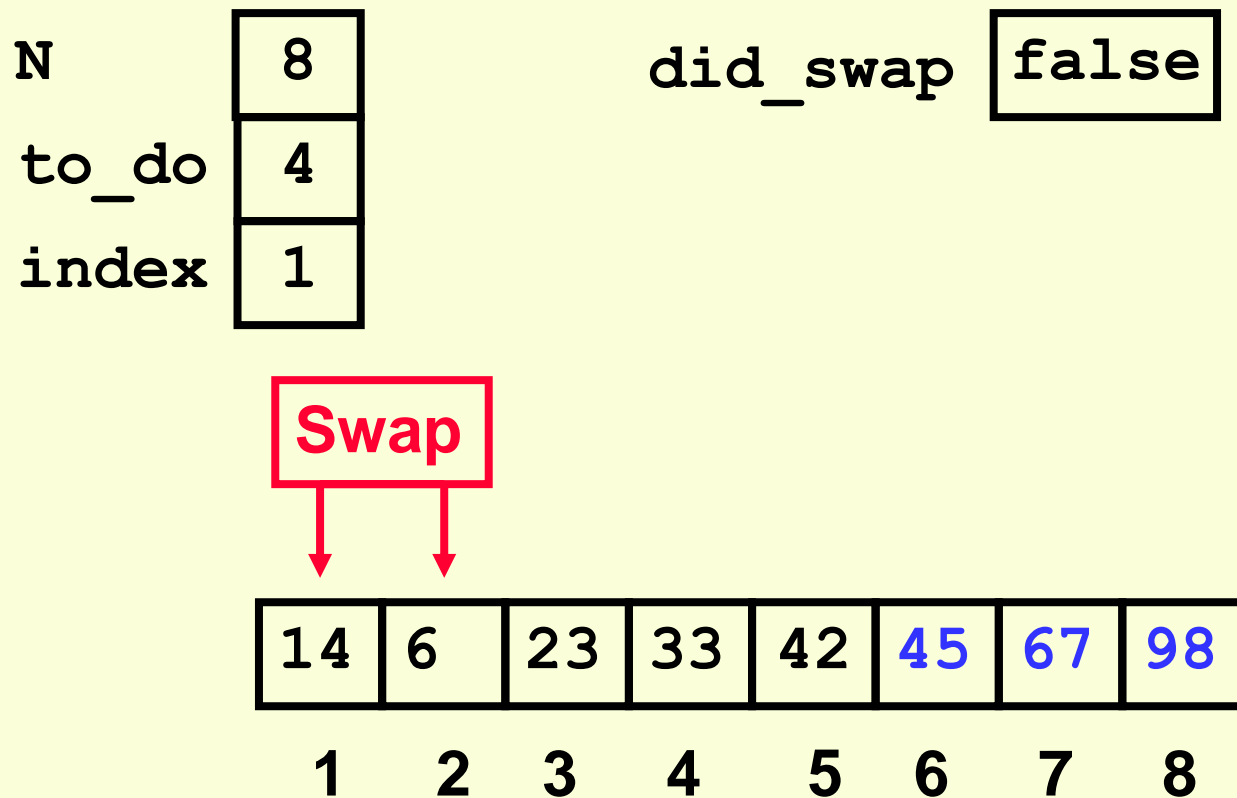
4

index

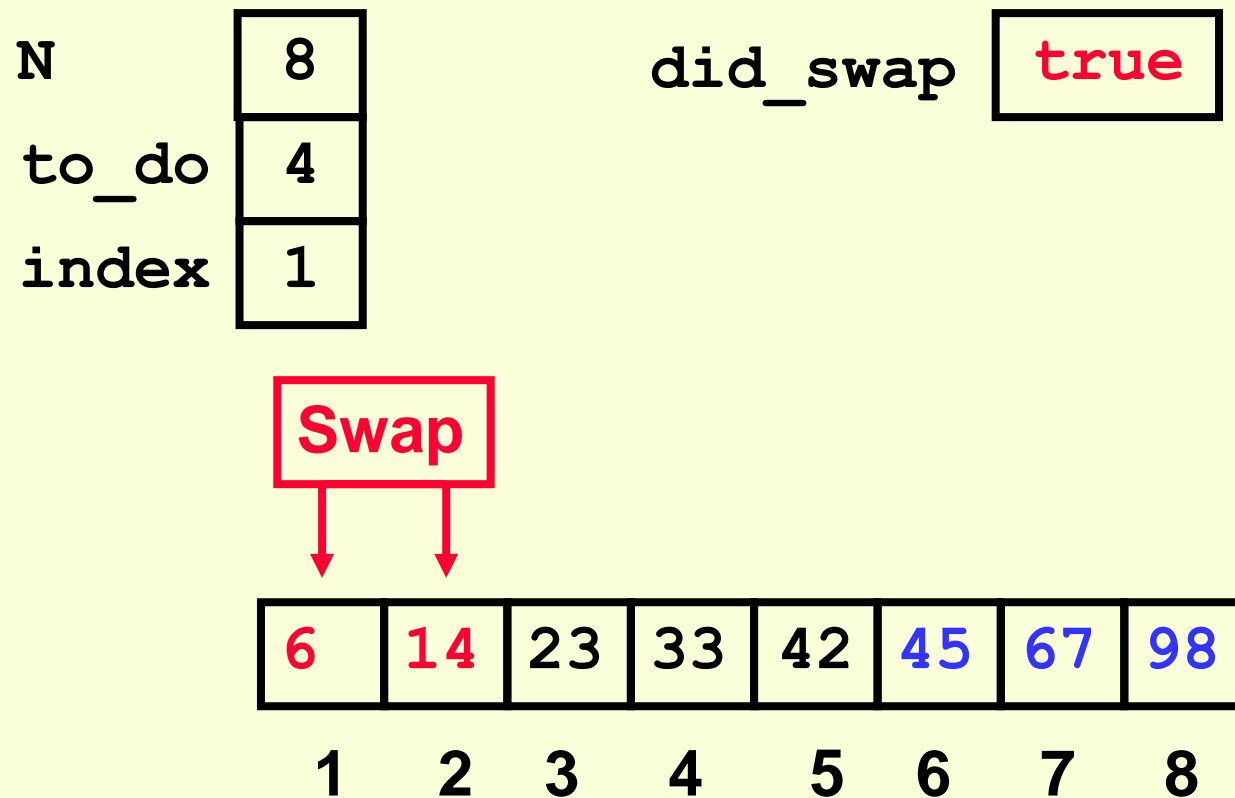
1



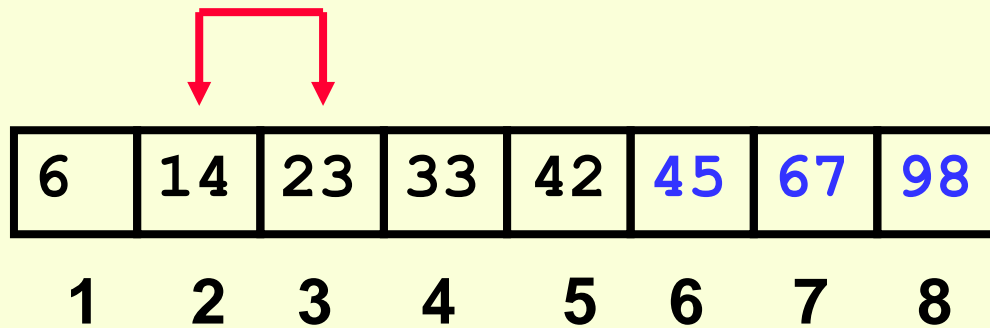
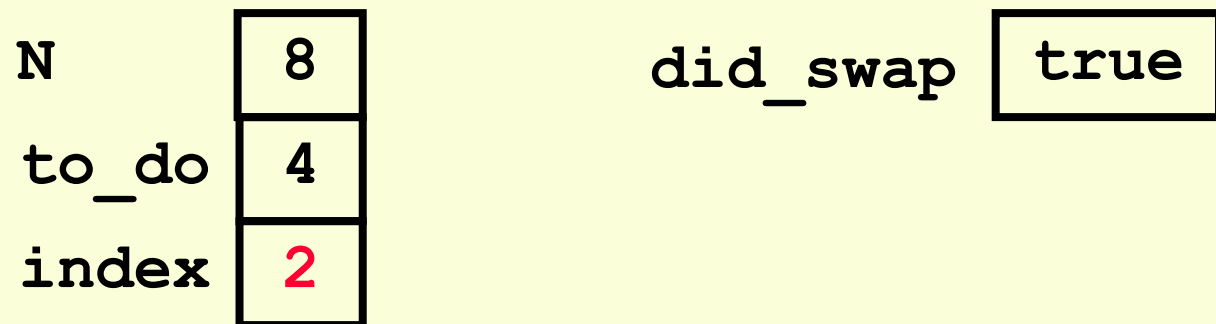
The Fourth “Bubble Up”



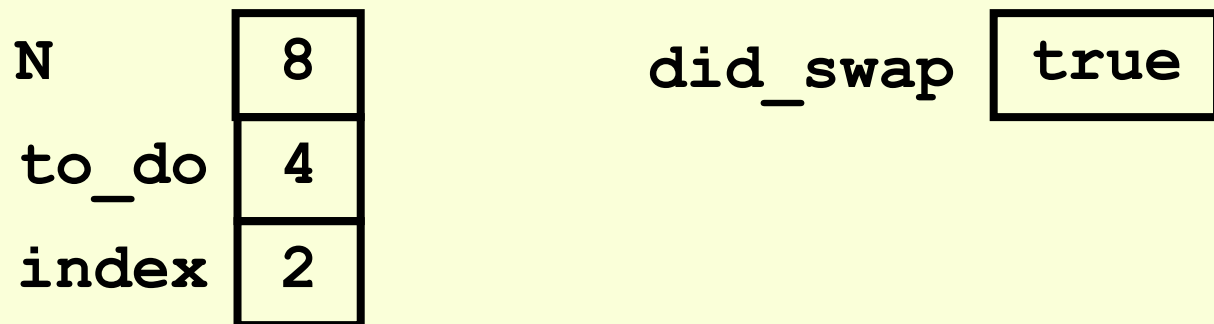
The Fourth “Bubble Up”



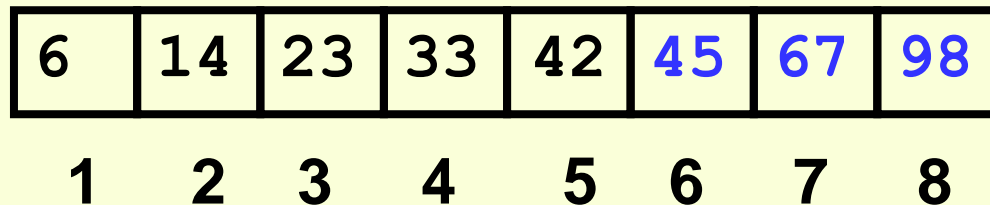
The Fourth “Bubble Up”



The Fourth “Bubble Up”



No Swap



The Fourth “Bubble Up”

N

8

to_do

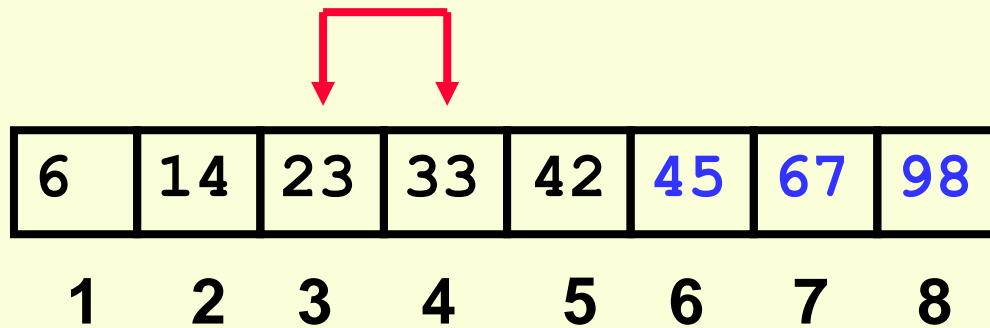
4

index

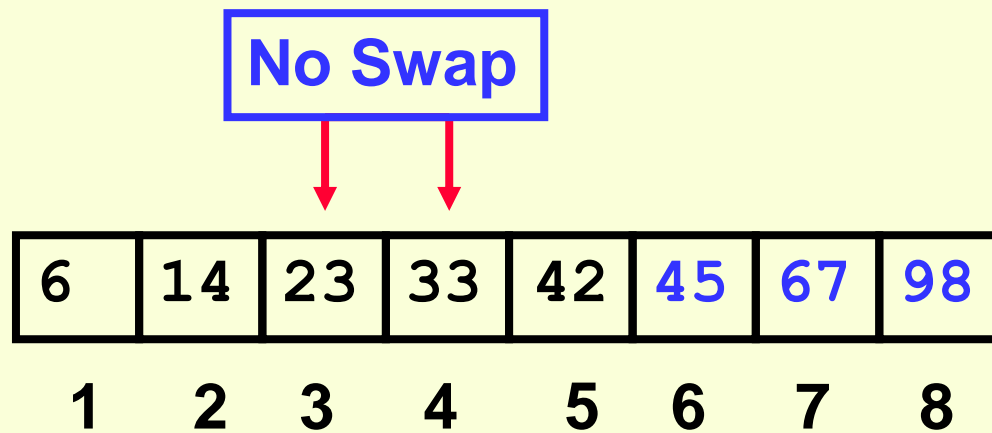
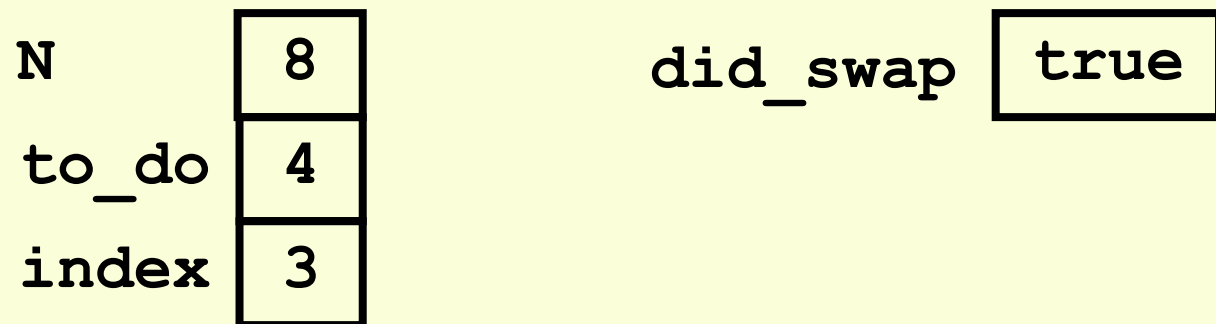
3

did_swap

true



The Fourth “Bubble Up”



The Fourth “Bubble Up”

N

8

to_do

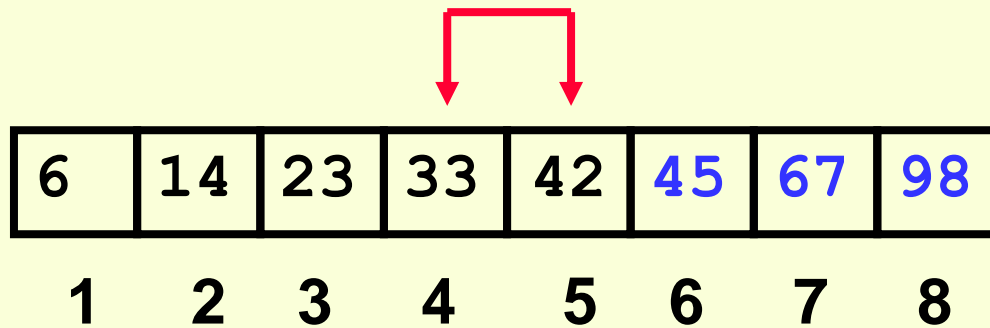
4

index

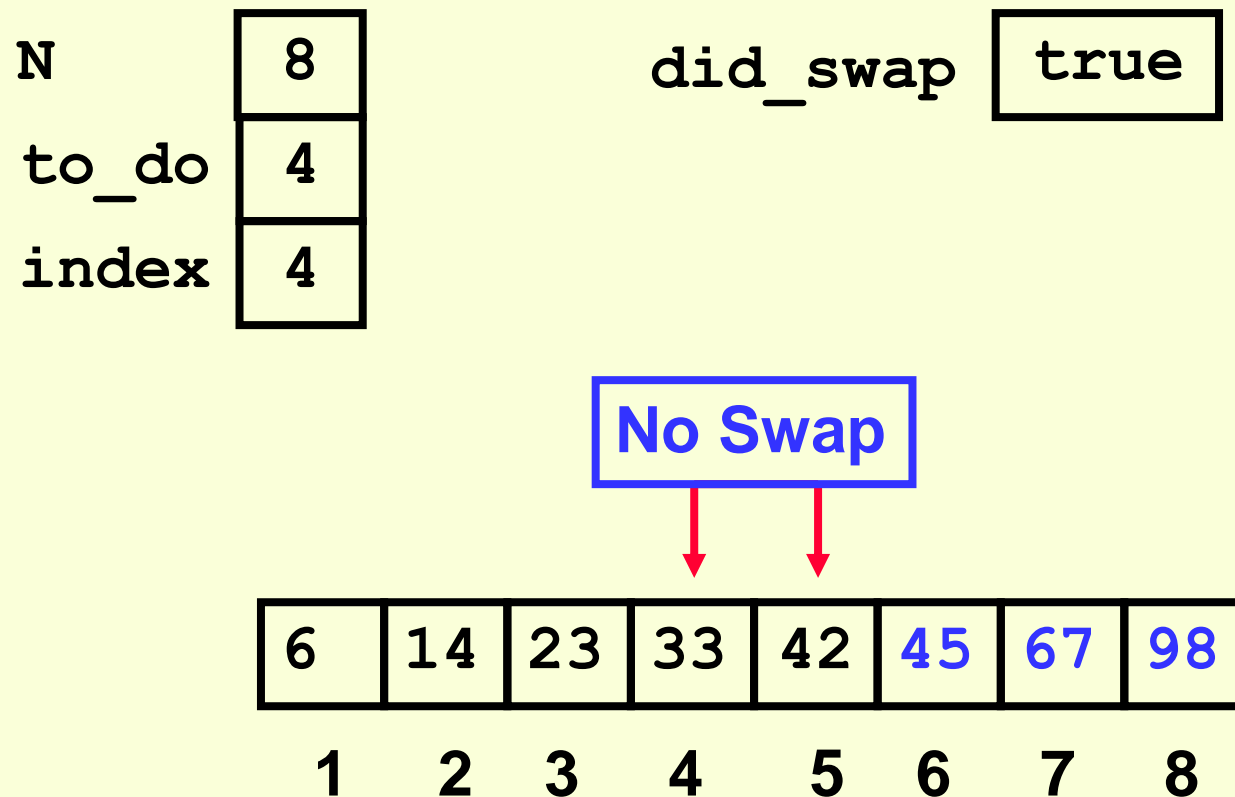
4

did_swap

true



The Fourth “Bubble Up”



After Fourth Pass of Outer Loop

N

8
4
5

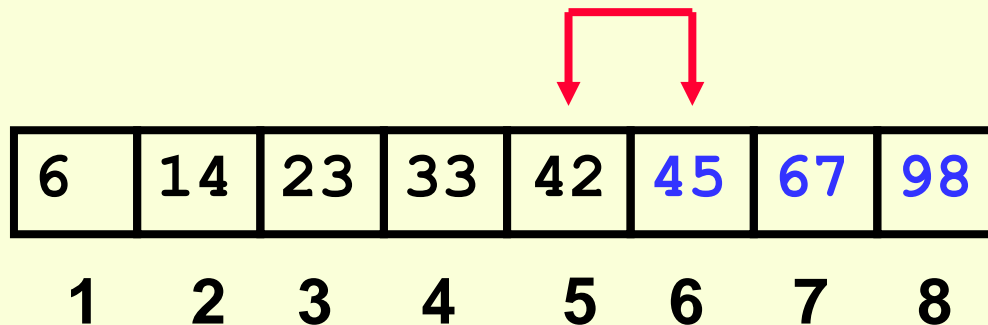
to_do

index

did_swap

true

Finished fourth "Bubble Up"



The Fifth “Bubble Up”

N

8

 did_swap

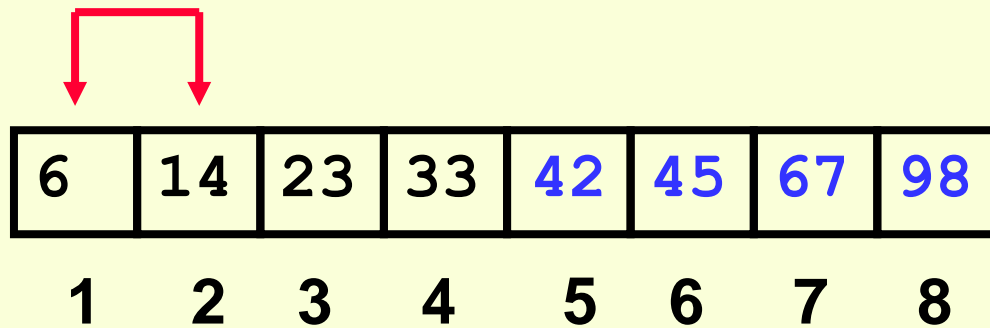
false

to_do

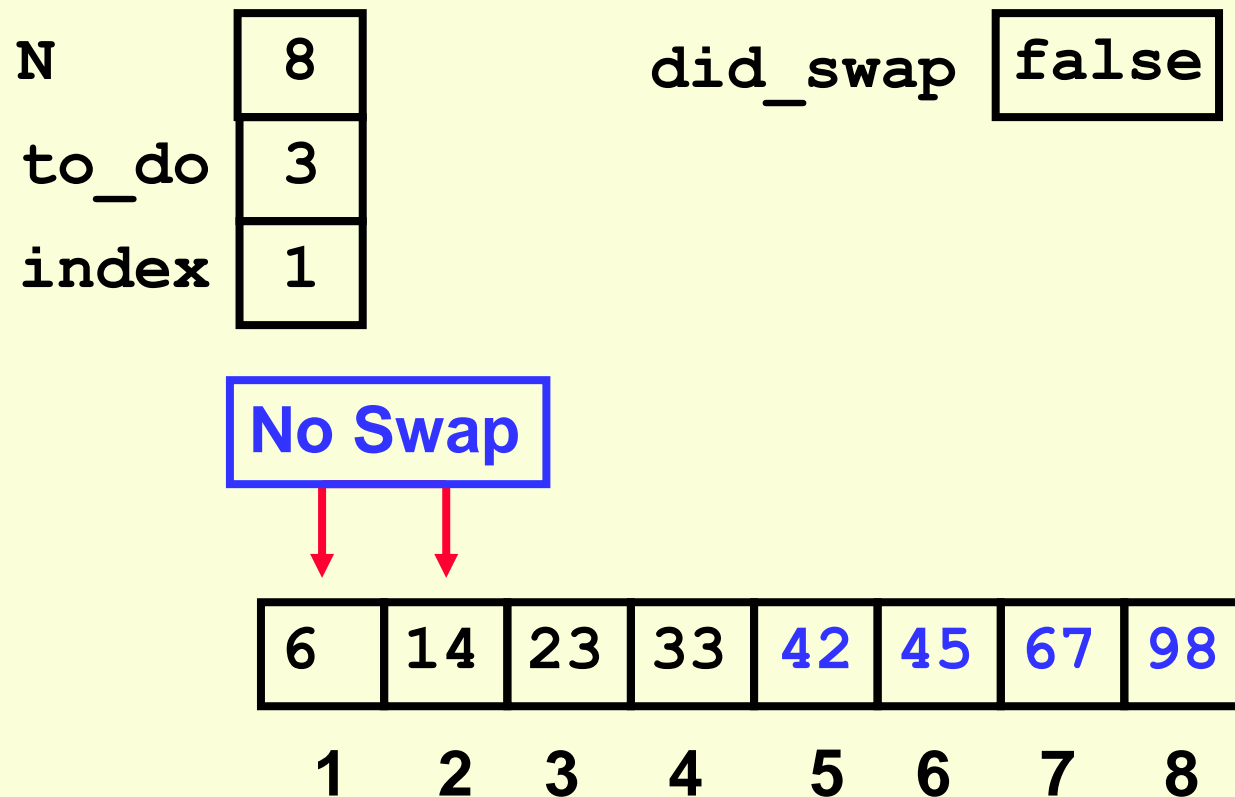
3

index

1



The Fifth “Bubble Up”



The Fifth “Bubble Up”

N

8

 did_swap

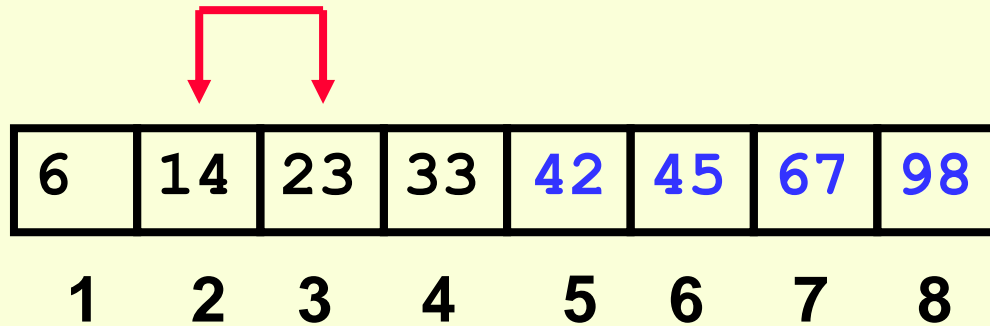
false

to_do

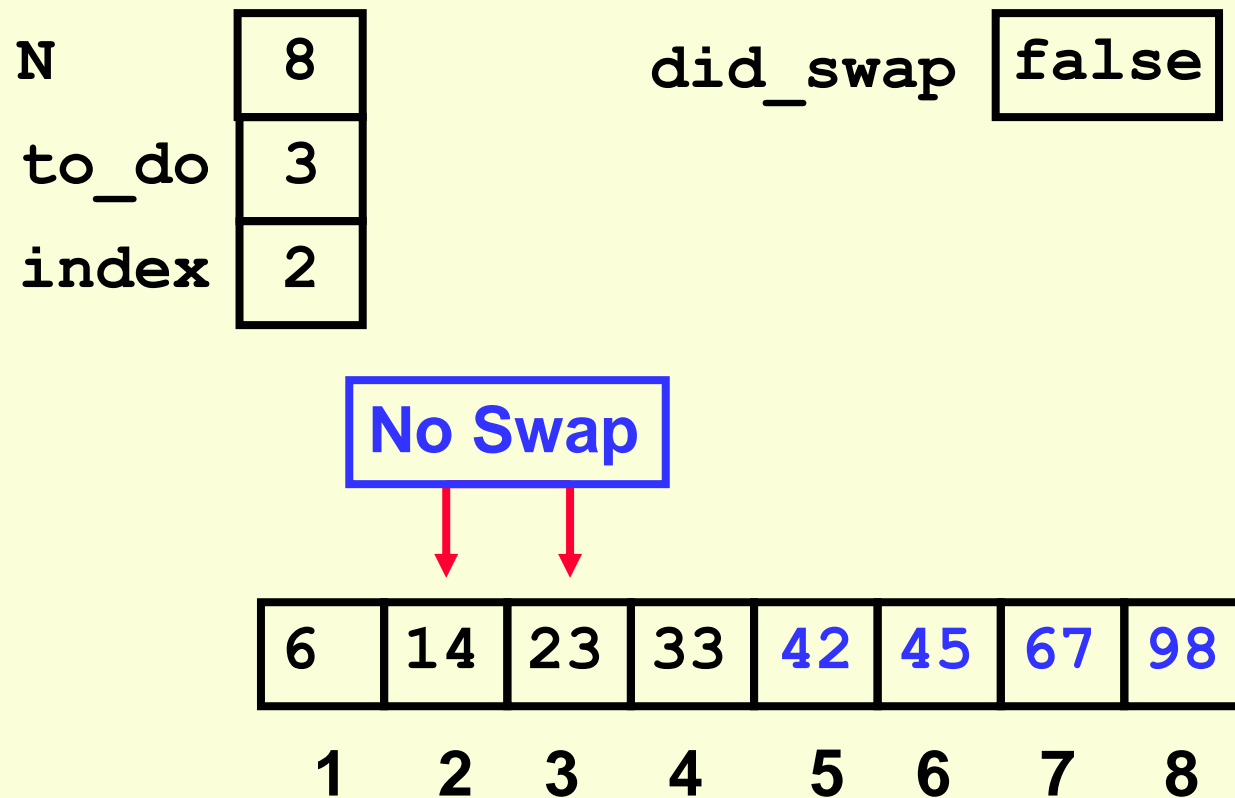
3

index

2



The Fifth “Bubble Up”



The Fifth “Bubble Up”

N

8

 did_swap

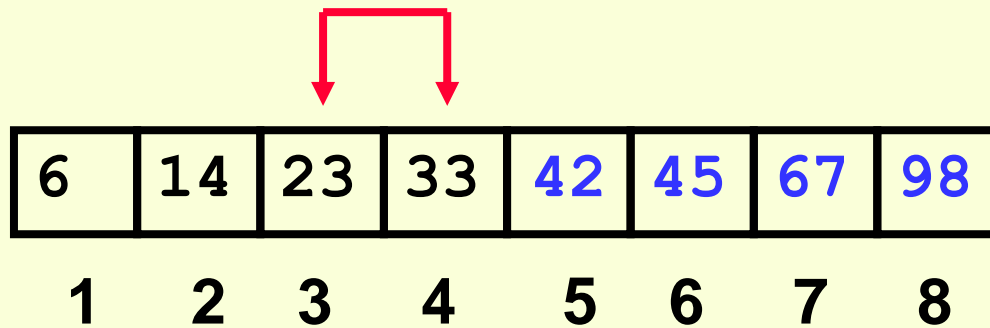
false

to_do

3

index

3



The Fifth “Bubble Up”

N

8

to_do

3

index

3

did_swap

false

No Swap

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

After Fifth Pass of Outer Loop

N

8

 did_swap

false

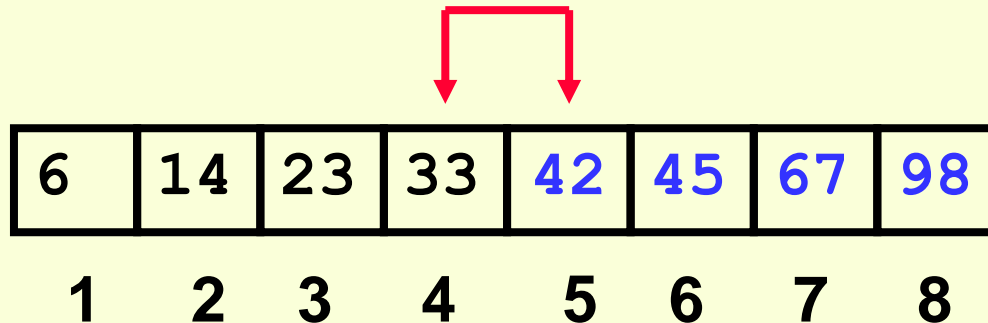
to_do

3

index

4

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

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