Selection Sort

- Selection sort is relatively easy to understand
- It sorts an array by making several passes through the array, selecting a next smallest item in the array each time and placing it where it belongs in the array
    - While the sort algorithms are not limited to arrays, we will sort arrays for simplicity

Trace of Selection Sort

n = number of elements in the array

```
for fill = 0 to n - 2 do
    Set posMin to the subscript of a smallest item in the subarray starting at subscript fill
    Exchange the item at posMin with the one at fill
```

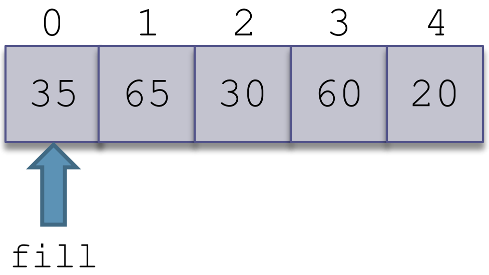
0	1	2	3	4
35	65	30	60	20

n	5
fill	
posMin	

Trace of Selection Sort (cont.)

n = number of elements in the array

```
► 1. for fill = 0 to n - 2 do
  2.     Set posMin to the subscript of a smallest item in the
         subarray starting at subscript fill
  3.     Exchange the item at posMin with the one at fill
```



0	1	2	3	4
35	65	30	60	20

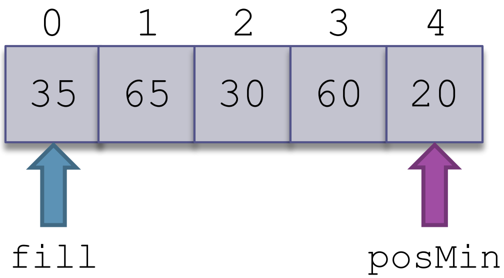
fill

n	5
fill	0
posMin	

Trace of Selection Sort (cont.)

n = number of elements in the array

```
for fill = 0 to n – 2 do
        Set posMin to the subscript of a smallest item in the
        subarray starting at subscript fill
        Exchange the item at posMin with the one at fill
```



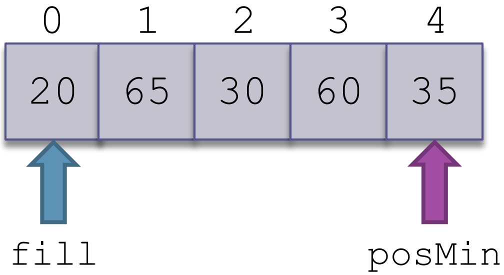
0	1	2	3	4
35	65	30	60	20

n	5
fill	0
posMin	4

Trace of Selection Sort (cont.)

n = number of elements in the array

```
1. for fill = 0 to n – 2 do
2.       Set posMin to the subscript of a smallest item in the
         subarray starting at subscript fill
▶ 3.     Exchange the item at posMin with the one at fill
```



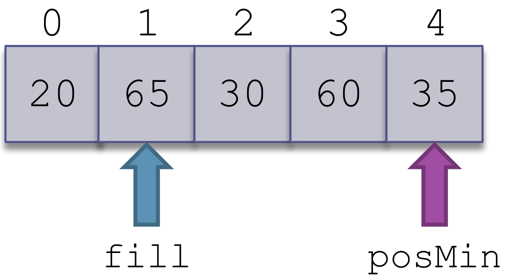
0	1	2	3	4
20	65	30	60	35

n	5
fill	0
posMin	4

Trace of Selection Sort (cont.)

n = number of elements in the array

```
►1. for fill = 0 to n - 2 do
 2.        Set posMin to the subscript of a smallest item in the
           subarray starting at subscript fill
 3.        Exchange the item at posMin with the one at fill
```



0	1	2	3	4
20	65	30	60	35

fill

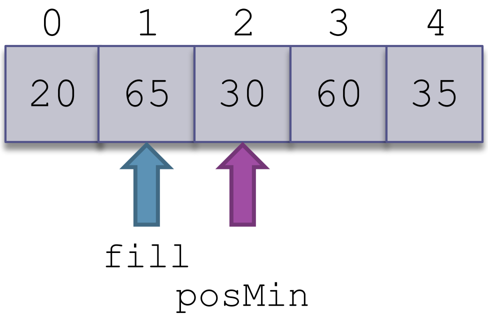
posMin

n	5
fill	1
posMin	4

Trace of Selection Sort (cont.)

n = number of elements in the array

```
  1. for fill = 0 to n - 2 do
▶ 2.     Set posMin to the subscript of a smallest item in the
         subarray starting at subscript fill
  3.     Exchange the item at posMin with the one at fill
```



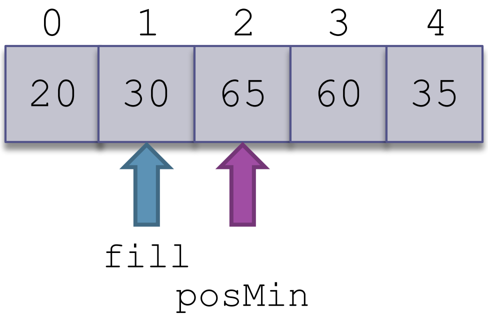
0	1	2	3	4
20	65	30	60	35

n	5
fill	1
posMin	2

Trace of Selection Sort (cont.)

n = number of elements in the array

```
1. for fill = 0 to n - 2 do
2.     Set posMin to the subscript of a smallest item in the
       subarray starting at subscript fill
3.     Exchange the item at posMin with the one at fill
```



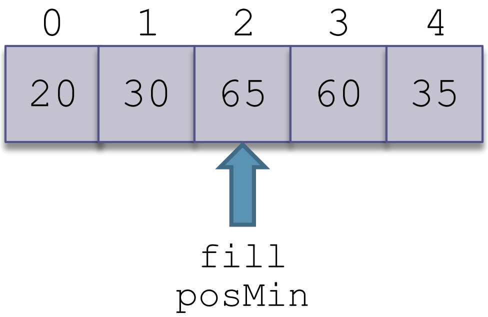
0	1	2	3	4
20	30	65	60	35

n	5
fill	1
posMin	2

Trace of Selection Sort (cont.)

n = number of elements in the array

```
► 1. for fill = 0 to n - 2 do
  2.     Set posMin to the subscript of a smallest item in the
         subarray starting at subscript fill
  3.     Exchange the item at posMin with the one at fill
```



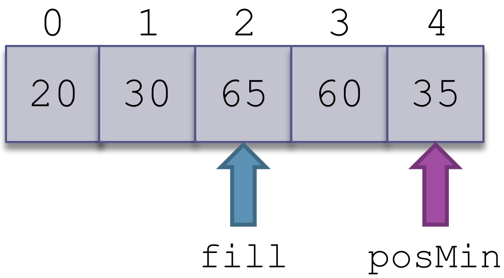
0	1	2	3	4
20	30	65	60	35

n	5
fill	2
posMin	2

Trace of Selection Sort (cont.)

`n` = number of elements in the array

```
for fill = 0 to n - 2 do
    Set posMin to the subscript of a smallest item in the subarray starting at subscript fill
    Exchange the item at posMin with the one at fill
```

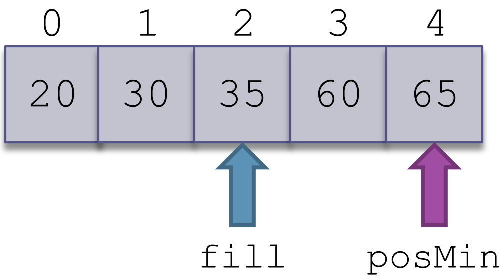


n	5
fill	2
posMin	4

Trace of Selection Sort (cont.)

```
n  = number of elements in the array

1. for fill = 0 to n - 2 do
2.        Set posMin to the subscript of a smallest item in the
          subarray starting at subscript fill
3.        Exchange the item at posMin with the one at fill
```



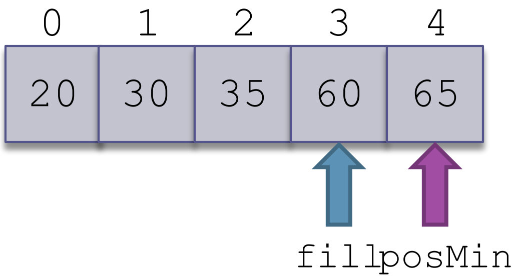
0	1	2	3	4
20	30	35	60	65

n	5
fill	2
posMin	4

Trace of Selection Sort (cont.)

n = number of elements in the array

```
1. for fill = 0 to n - 2 do
2.         Set posMin to the subscript of a smallest item in the
           subarray starting at subscript fill
3.         Exchange the item at posMin with the one at fill
```



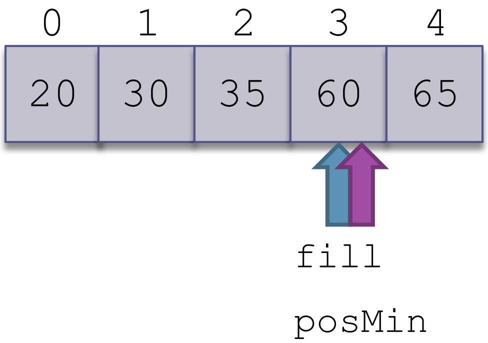
0	1	2	3	4
20	30	35	60	65

n	5
fill	3
posMin	4

Trace of Selection Sort (cont.)

n = number of elements in the array

```
1. for fill = 0 to n - 2 do
2.     Set posMin to the subscript of a smallest item in the
       subarray starting at subscript fill
3.     Exchange the item at posMin with the one at fill
```

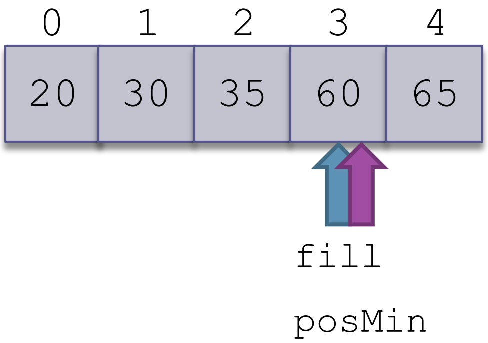


n	5
fill	3
posMin	3

Trace of Selection Sort (cont.)

n = number of elements in the array

```
1. for fill = 0 to n - 2 do
2.     Set posMin to the subscript of a smallest item in the
       subarray starting at subscript fill
3.     Exchange the item at posMin with the one at fill
```



0	1	2	3	4
20	30	35	60	65

n	5
fill	3
posMin	3

Trace of Selection Sort (cont.)

n = number of elements in the array

```
1. for fill = 0 to n - 2 do
2.     Set posMin to the subscript of a smallest item in the
       subarray starting at subscript fill
3.     Exchange the item at posMin with the one at fill
```

0	1	2	3	4
20	30	35	60	65

n	5
fill	3
posMin	3

Trace of Selection Sort Refinement

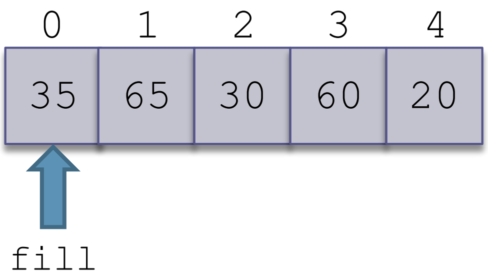
n	5
fill	
posMin	
next	

0	1	2	3	4
35	65	30	60	20

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	0
posMin	
next	

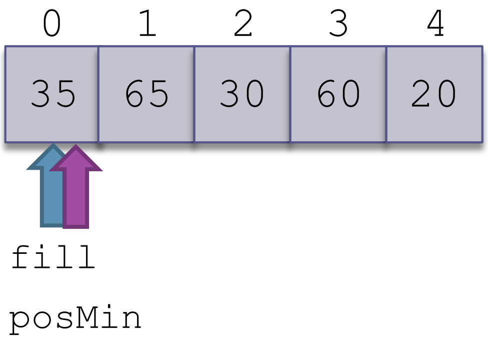


0	1	2	3	4
35	65	30	60	20

```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the
           item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```

Trace of Selection Sort Refinement (cont.)

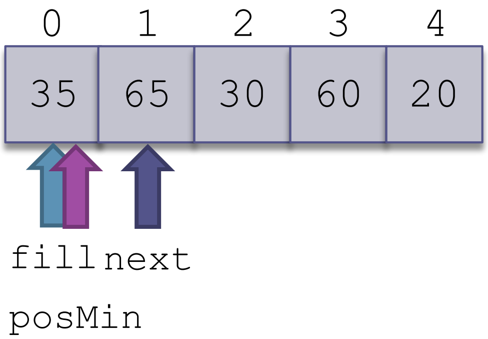
n	5
fill	0
posMin	0
next	



```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	0
posMin	0
next	1

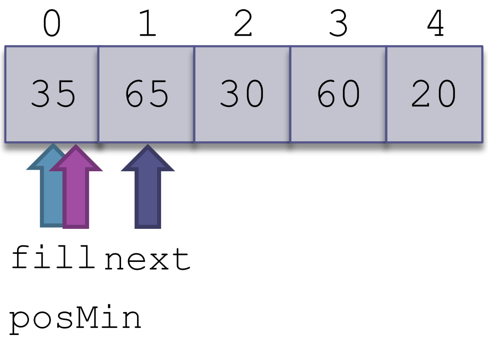


0	1	2	3	4
35	65	30	60	20

```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	0
posMin	0
next	1

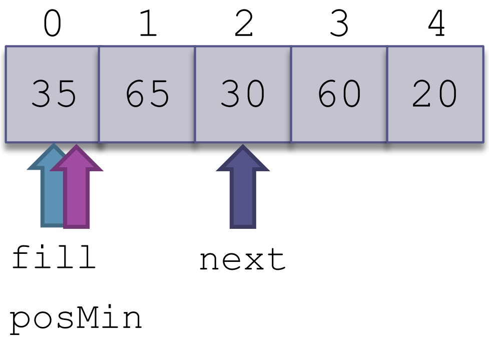


0	1	2	3	4
35	65	30	60	20

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

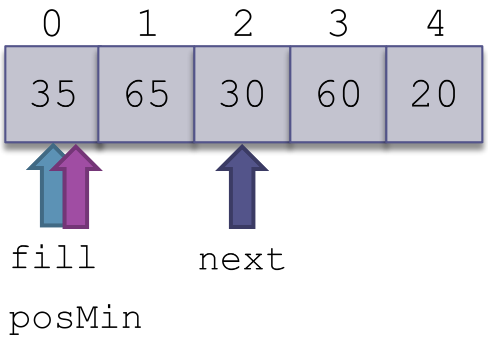
n	5
fill	0
posMin	0
next	2



```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	0
posMin	0
next	2

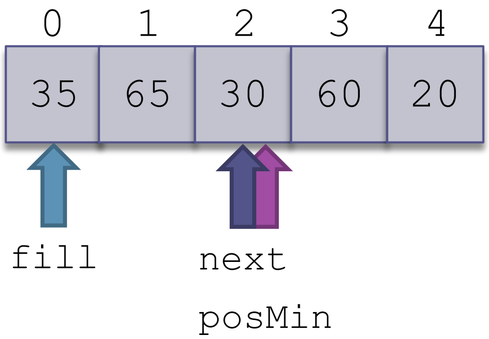


0	1	2	3	4
35	65	30	60	20

```
1. for fill = 0 to n – 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n – 1 do
4.         if the item at next is less than the
           item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	0
posMin	2
next	2

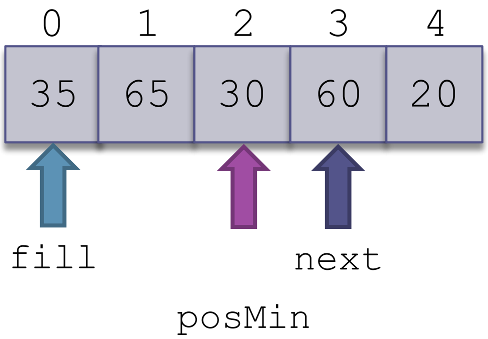


0	1	2	3	4
35	65	30	60	20

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	0
posMin	2
next	3

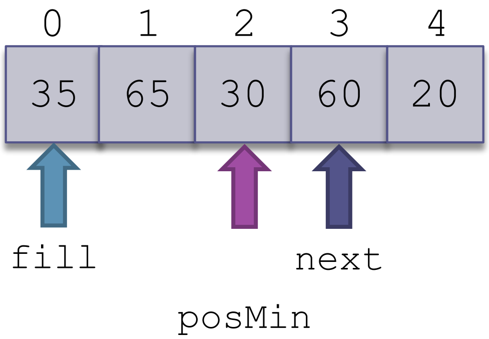


0	1	2	3	4
35	65	30	60	20

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	0
posMin	2
next	3

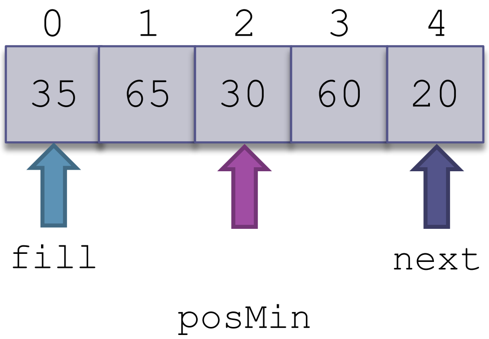


0	1	2	3	4
35	65	30	60	20

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

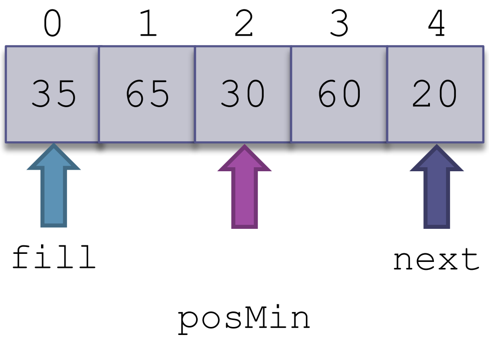
n	5
fill	0
posMin	2
next	4



```
  for fill = 0 to n - 2 do
      Initialize posMin to fill
► for next = fill + 1 to n - 1 do
          if the item at next is less than the item at posMin
              Reset posMin to next
      Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

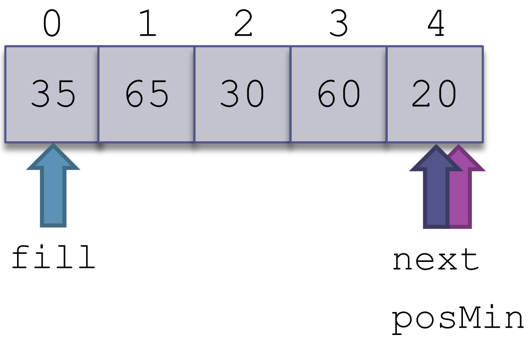
n	5
fill	0
posMin	2
next	4



```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

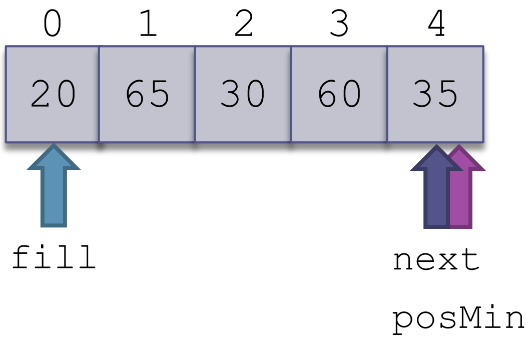
n	5
fill	0
posMin	4
next	4



```
  for fill = 0 to n - 2 do
      Initialize posMin to fill
      for next = fill + 1 to n - 1 do
          if the item at next is less than the item at posMin
►             Reset posMin to next
      Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	0
posMin	4
next	4

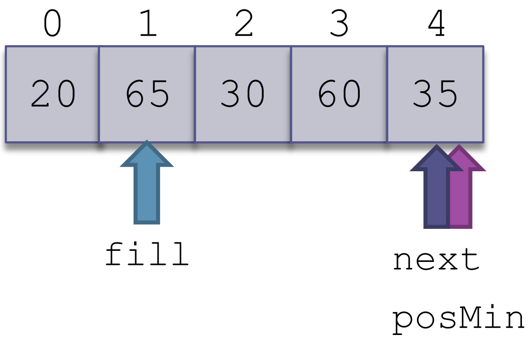


0	1	2	3	4
20	65	30	60	35

```
  1. for fill = 0 to n - 2 do
  2.     Initialize posMin to fill
  3.     for next = fill + 1 to n - 1 do
  4.         if the item at next is less than the
             item at posMin
  5.             Reset posMin to next
► 6.     Exchange the item at posMin with the one
         at fill
```

Trace of Selection Sort Refinement (cont.)

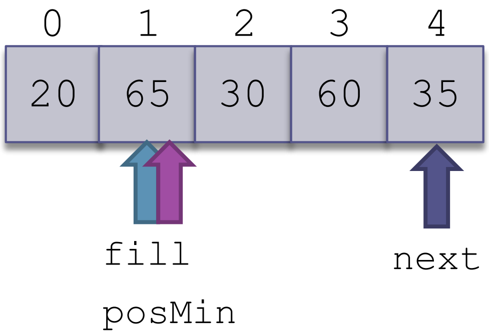
n	5
fill	1
posMin	4
next	4



```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the
           item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```

Trace of Selection Sort Refinement (cont.)

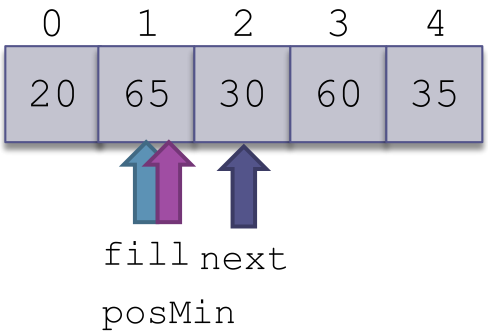
n	5
fill	1
posMin	1
next	4



```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	1
posMin	1
next	2

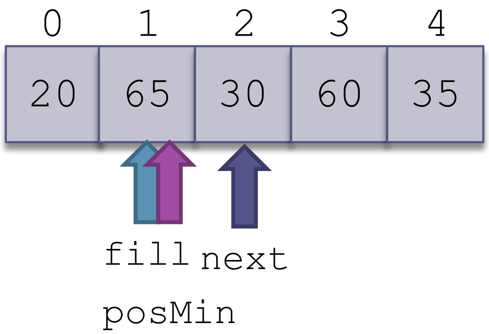


0	1	2	3	4
20	65	30	60	35

```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the
           item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```

Trace of Selection Sort Refinement (cont.)

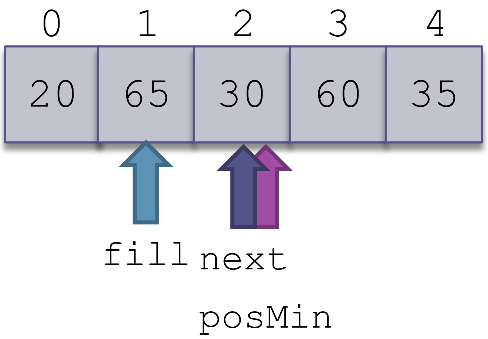
n	5
fill	1
posMin	1
next	2



```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	1
posMin	2
next	2

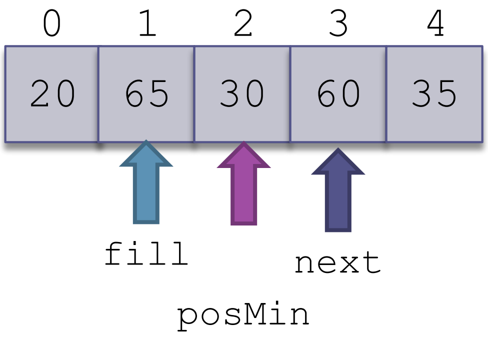


0	1	2	3	4
20	65	30	60	35

```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the
           item at posMin
➤ 5.           Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	1
posMin	2
next	3

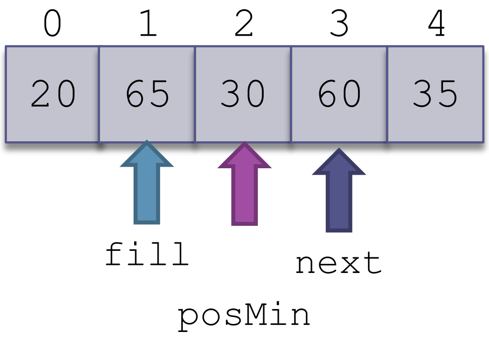


0	1	2	3	4
20	65	30	60	35

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	1
posMin	2
next	3

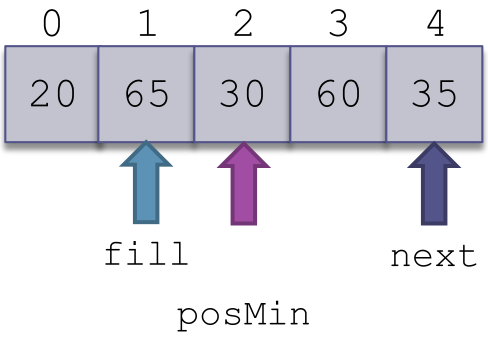


0	1	2	3	4
20	65	30	60	35

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	1
posMin	2
next	4

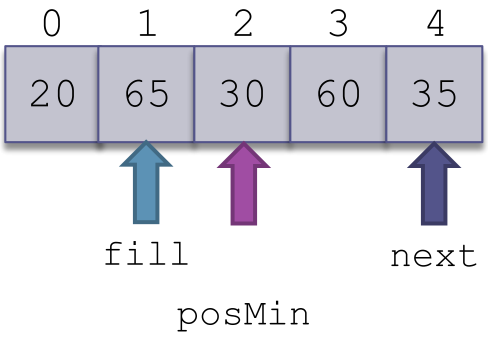


0	1	2	3	4
20	65	30	60	35

```
  1. for fill = 0 to n - 2 do
  2.     Initialize posMin to fill
► 3.     for next = fill + 1 to n - 1 do
  4.         if the item at next is less than the
             item at posMin
  5.             Reset posMin to next
  6.     Exchange the item at posMin with the one
         at fill
```

Trace of Selection Sort Refinement (cont.)

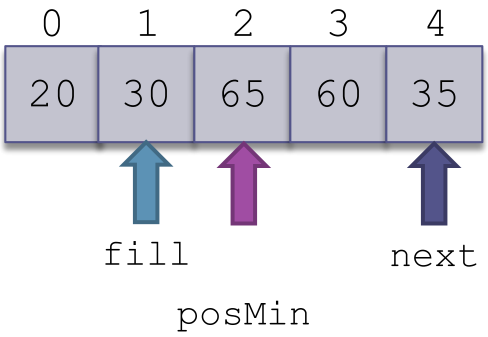
n	5
fill	1
posMin	2
next	4



```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

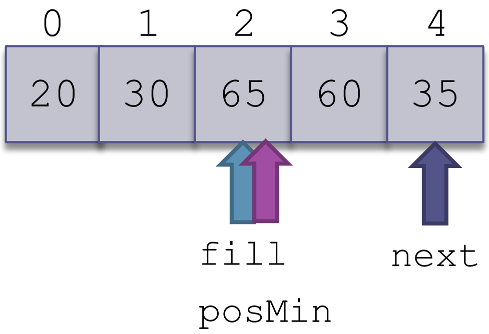
n	5
fill	1
posMin	2
next	4



```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the
           item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```

Trace of Selection Sort Refinement (cont.)

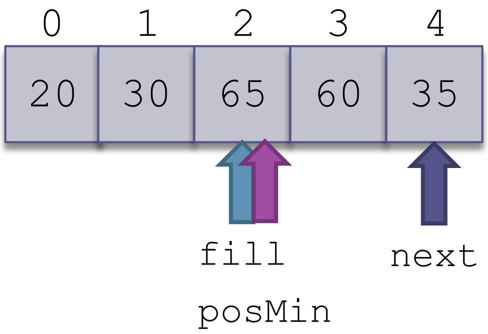
n	5
fill	2
posMin	2
next	4



```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	2
posMin	2
next	4



0	1	2	3	4
20	30	65	60	35

fill

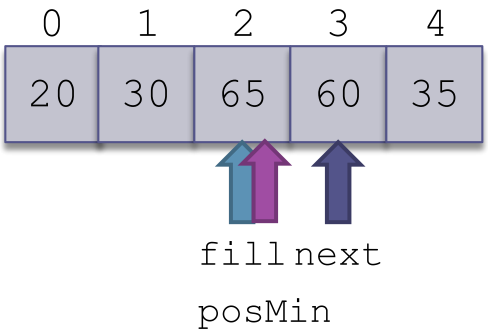
posMin

next

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	2
posMin	2
next	3

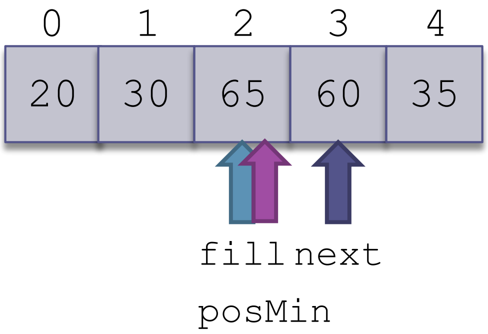


0	1	2	3	4
20	30	65	60	35

```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the
           item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```

Trace of Selection Sort Refinement (cont.)

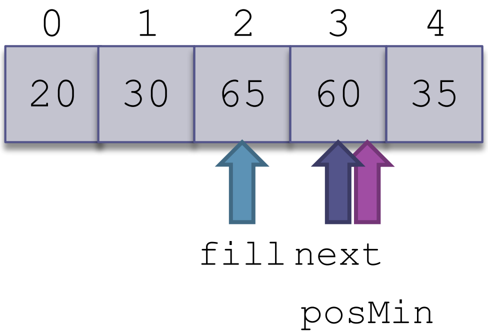
n	5
fill	2
posMin	2
next	3



```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

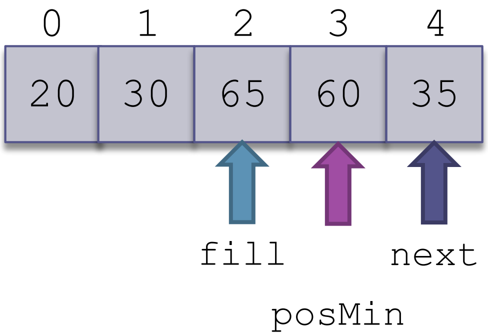
n	5
fill	2
posMin	3
next	3



```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the
           item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	2
posMin	3
next	4

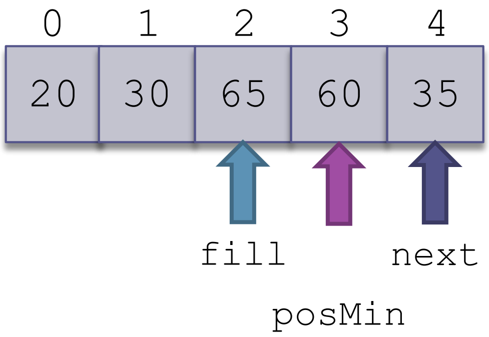


0	1	2	3	4
20	30	65	60	35

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

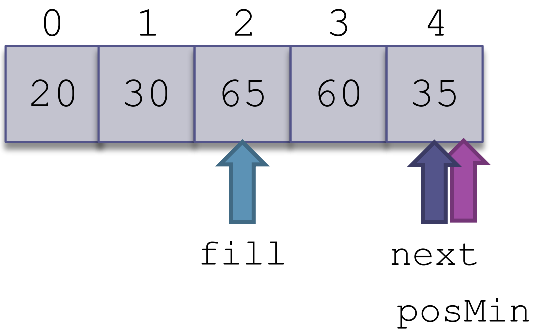
n	5
fill	2
posMin	3
next	4



```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the
           item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```

Trace of Selection Sort Refinement (cont.)

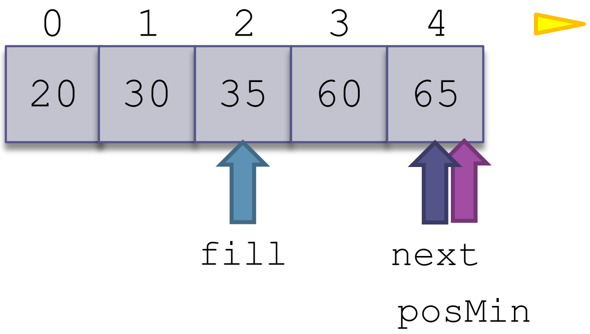
n	5
fill	2
posMin	4
next	4



```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the
           item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	2
posMin	4
next	4

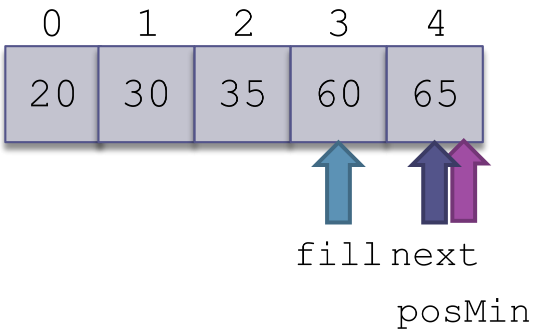


0	1	2	3	4
20	30	35	60	65

```
  1. for fill = 0 to n - 2 do
  2.     Initialize posMin to fill
  3.     for next = fill + 1 to n - 1 do
  4.         if the item at next is less than the item at posMin
  5.             Reset posMin to next
► 6.     Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	3
posMin	4
next	4

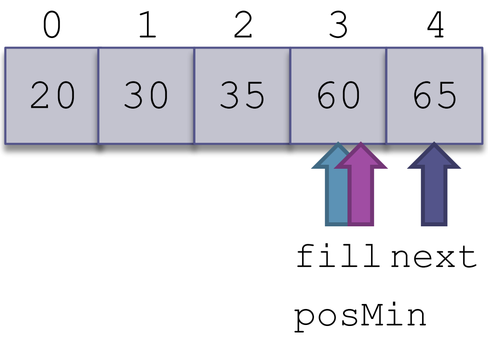


0	1	2	3	4
20	30	35	60	65

```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	3
posMin	3
next	4



0 1 2 3 4

20 30 35 60 65

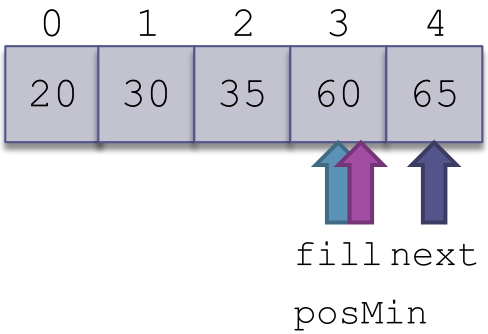
fill next

posMin

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	3
posMin	3
next	4



0	1	2	3	4
20	30	35	60	65

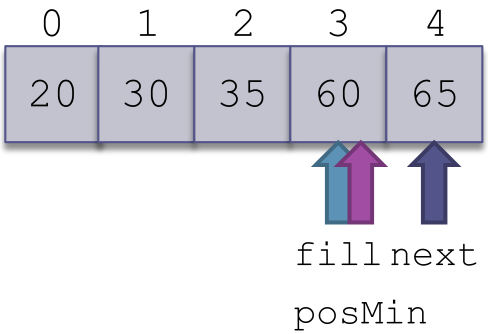
fill next

posMin

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Trace of Selection Sort Refinement (cont.)

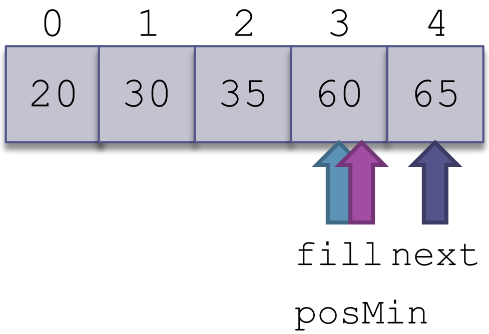
n	5
fill	3
posMin	3
next	4



```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the
           item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	3
posMin	3
next	4



```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the
        item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one
    at fill
```

Trace of Selection Sort Refinement (cont.)

n	5
fill	3
posMin	3
next	4

0	1	2	3	4
20	30	35	60	65

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

Analysis of Selection Sort

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

This loop is performed n-1 times

Analysis of Selection Sort (cont.)

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one at fill
```

There are n-1 exchanges

Analysis of Selection Sort (cont.)

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the
        item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one
    at fill
```

This comparison is performed
$(n - 1 - fill)$
times for each value of *fill* and can be represented by the following series:
$(n\text{-}1) + (n\text{-}2) + \ldots + 3 + 2 + 1$

Analysis of Selection Sort (cont.)

The series

$(n\text{-}1) + (n\text{-}2) + \ldots + 3 + 2 + 1$

is a well-known series and can be written as

$$\frac{n \times (n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

```
for fill = 0 to n - 2 do
    Initialize posMin to fill
    for next = fill + 1 to n - 1 do
        if the item at next is less than the
        item at posMin
            Reset posMin to next
    Exchange the item at posMin with the one
    at fill
```

Code for Selection Sort

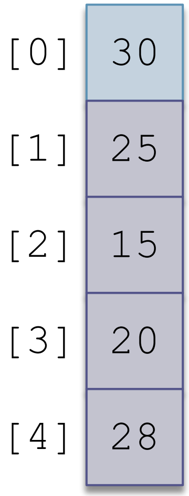
```
public static void sort(Object[] table) {
  int n = table.length;
  for (int fill = 0; fill < n – 1; fill++) {
    // Invariant: table[0 . . . fill – 1] is sorted.
    int posMin = fill;
    for (int next = fill + 1; next < n; next++) {
      // Invariant: table[posMin] is smallest item in
      // table[fill...next - 1].
      if (((Comparable) table[next]).compareTo(table[posMin]) < 0)
         posMin = next;
    }
    // assert: table[posMin] is smallest item in table[fill...n – 1]
    // Exchange table[fill] and table[posMin].
    var temp = table[fill];
    table[fill] = table[posMin];  table[posMin] = temp;
    // assert: table[fill] is smallest item in table[fill...n - 1]
  }
  // assert: table[0...n - 1] is sorted.
}
```

Insertion Sort

- Another quadratic sort, *insertion* sort, is based on the technique used by card players to arrange a hand of cards
  - The player keeps the cards that have been picked up so far in sorted order
  - When the player picks up a new card, the player makes room for the new card and then inserts it in its proper place

Trace of Insertion Sort

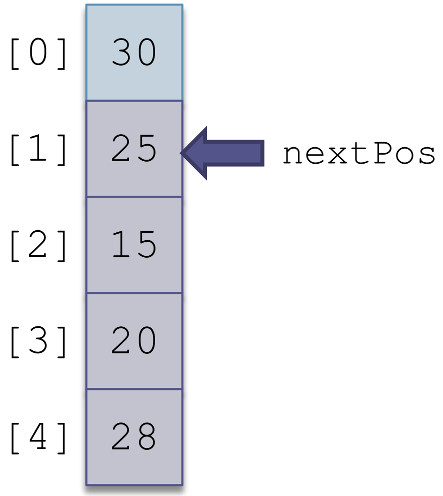
1. **`for`** each array element from the second (**`nextPos`** = 1) to the last
2. Insert the element at **`nextPos`** where it belongs in the array, increasing the length of the sorted subarray by 1 element


[0] 30
[1] 25
[2] 15
[3] 20
[4] 28

To adapt the insertion algorithm to an array that is filled with data, we start with a sorted subarray consisting of only the first element

Trace of Insertion Sort (cont.)

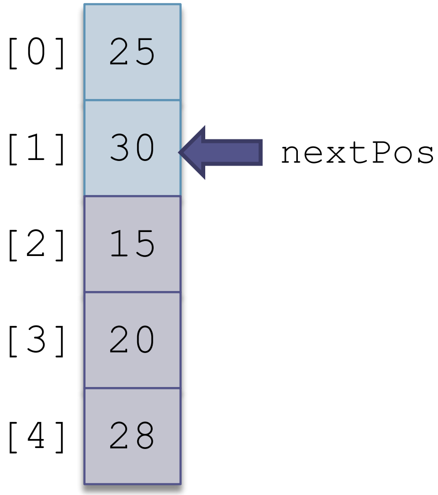
nextPos	1



1. **for** each array element from the second (**nextPos** = 1) to the last
2. Insert the element at **nextPos** where it belongs in the array, increasing the length of the sorted subarray by 1 element

Trace of Insertion Sort (cont.)

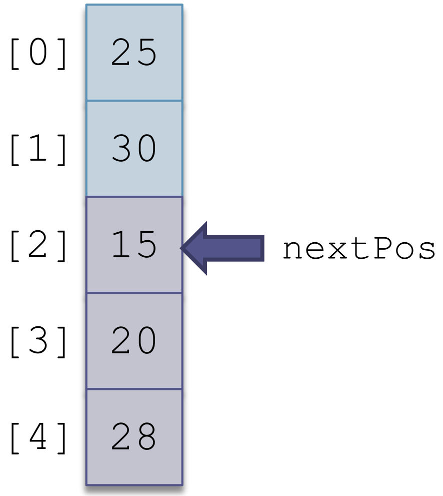
nextPos	1



1. **for** each array element from the second (**nextPos** = 1) to the last
2. Insert the element at **nextPos** where it belongs in the array, increasing the length of the sorted subarray by 1 element

Trace of Insertion Sort (cont.)

nextPos	2

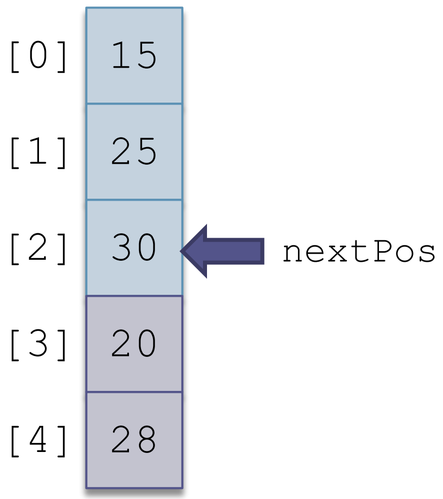


1. **for** each array element from the second (**nextPos** = 1) to the last

2. Insert the element at **nextPos** where it belongs in the array, increasing the length of the sorted subarray by 1 element

Trace of Insertion Sort (cont.)

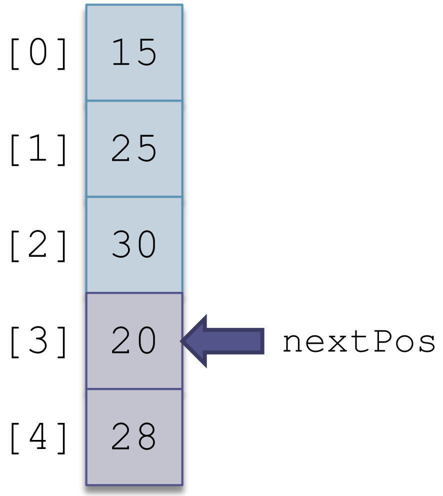
nextPos	2



```
1. for each array element from the second (nextPos = 1) to the last
2.     Insert the element at nextPos where it belongs in the array, increasing the length of the sorted subarray by 1 element
```

Trace of Insertion Sort (cont.)

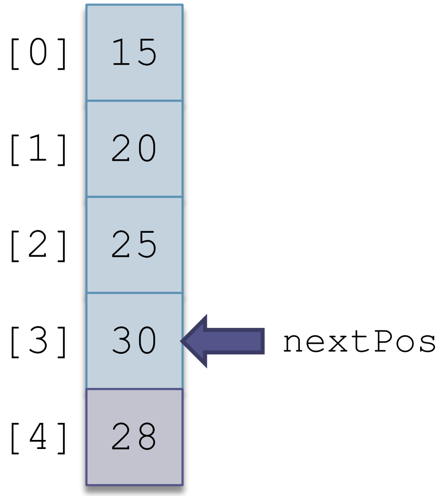
nextPos	3



1. **for** each array element from the second (**nextPos** = 1) to the last
2. Insert the element at **nextPos** where it belongs in the array, increasing the length of the sorted subarray by 1 element

Trace of Insertion Sort (cont.)

nextPos	3



[0] 15

[1] 20

[2] 25

[3] 30 nextPos

[4] 28

1. **for** each array element from the second (**nextPos** = 1) to the last

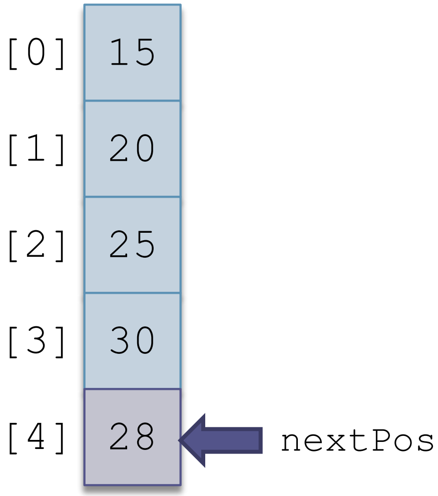
2. Insert the element at **nextPos** where it belongs in the array, increasing the length of the sorted subarray by 1 element

Trace of Insertion Sort (cont.)

nextPos	4

1. **for** each array element from the second (**nextPos** = 1) to the last

2. Insert the element at **nextPos** where it belongs in the array, increasing the length of the sorted subarray by 1 element



[0] 15

[1] 20

[2] 25

[3] 30

[4] 28

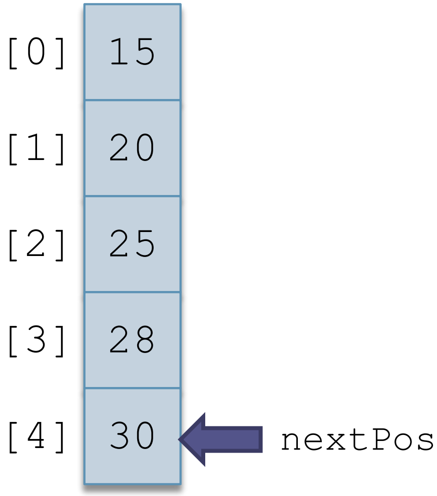
nextPos

Trace of Insertion Sort (cont.)

nextPos	4

1. **for** each array element from the second (**nextPos** = 1) to the last

2. Insert the element at **nextPos** where it belongs in the array, increasing the length of the sorted subarray by 1 element



[0]	15	
[1]	20	
[2]	25	
[3]	28	
[4]	30	nextPos

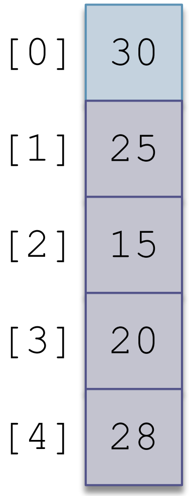
Trace of Insertion Sort (cont.)

nextPos	-

[0]	15
[1]	20
[2]	25
[3]	28
[4]	30

1. **for** each array element from the second (**nextPos** = 1) to the last
2. Insert the element at **nextPos** where it belongs in the array, increasing the length of the sorted subarray by 1 element

Trace of Insertion Sort Refinement

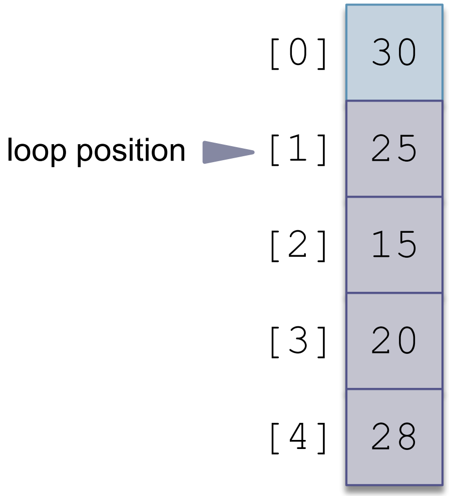


[0]	30
[1]	25
[2]	15
[3]	20
[4]	28

```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos - 1 > nextVal
5.         Shift the element at nextPos - 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	1
nextVal	



[0] 30

loop position ► [1] 25

[2] 15

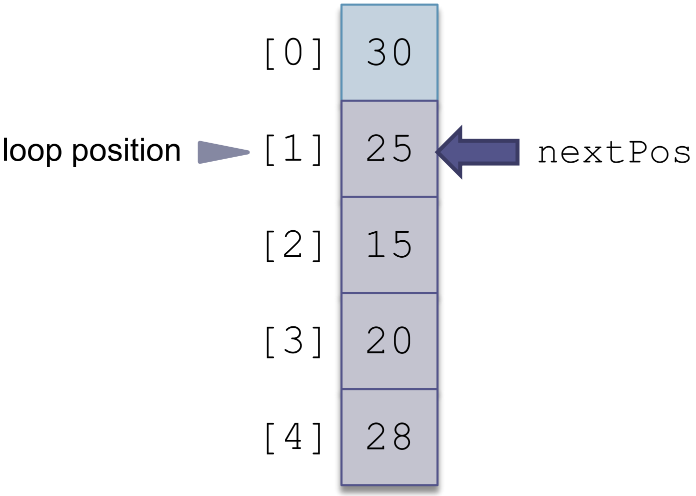
[3] 20

[4] 28

```
► 1. for each array element from the second
     (nextPos = 1) to the last
  2.    nextPos is the position of the element to
        insert
  3.    Save the value of the element to insert in
        nextVal
  4.    while nextPos > 0 and the element
        at nextPos - 1 > nextVal
  5.        Shift the element at nextPos - 1 to
            position nextPos
  6.        Decrement nextPos by 1
  7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	1
nextVal	



loop position

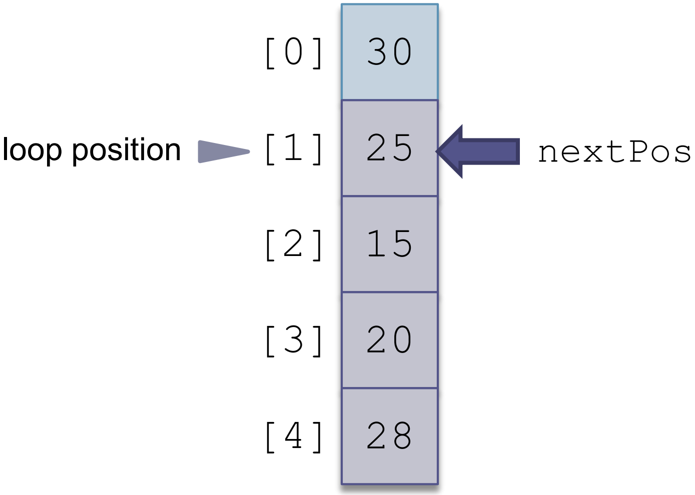
[0]	30
[1]	25
[2]	15
[3]	20
[4]	28

nextPos

```
   1. for each array element from the second
      (nextPos = 1) to the last
►  2.     nextPos is the position of the element to
          insert
   3.     Save the value of the element to insert in
          nextVal
   4.     while nextPos > 0 and the element
          at nextPos – 1 > nextVal
   5.         Shift the element at nextPos – 1 to
              position nextPos
   6.         Decrement nextPos by 1
   7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

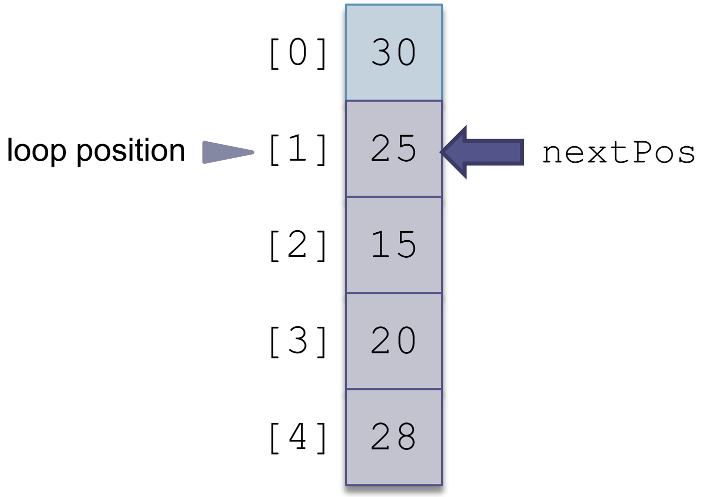
nextPos	1
nextVal	25



```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
▶ 3.   Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos – 1 > nextVal
5.         Shift the element at nextPos – 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

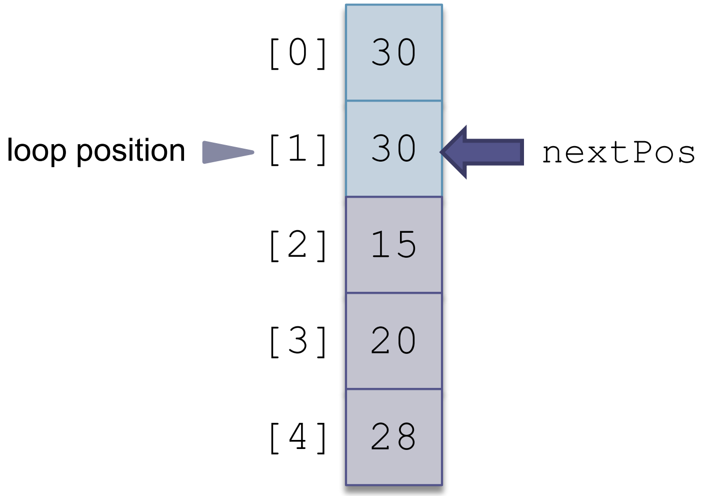
nextPos	1
nextVal	25



```
1. for each array element from the second
   (nextPos = 1) to the last
2.    nextPos is the position of the element to
      insert
3.    Save the value of the element to insert in
      nextVal
4.    while nextPos > 0 and the element
      at nextPos – 1 > nextVal
5.        Shift the element at nextPos – 1 to
          position nextPos
6.        Decrement nextPos by 1
7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

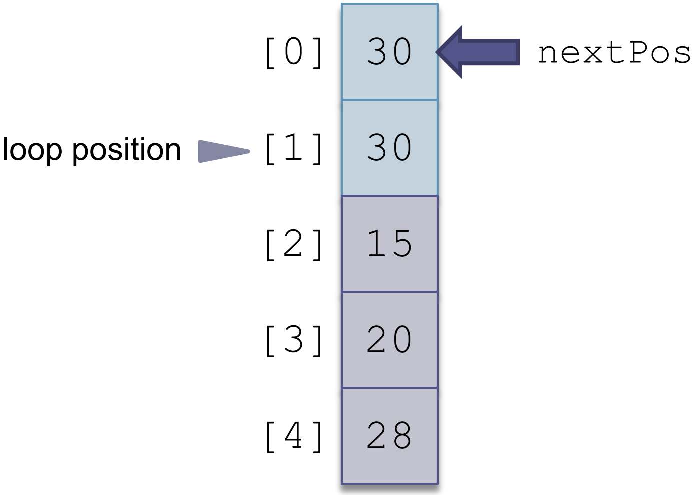
nextPos	1
nextVal	25



```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos - 1 > nextVal
5.         Shift the element at nextPos - 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

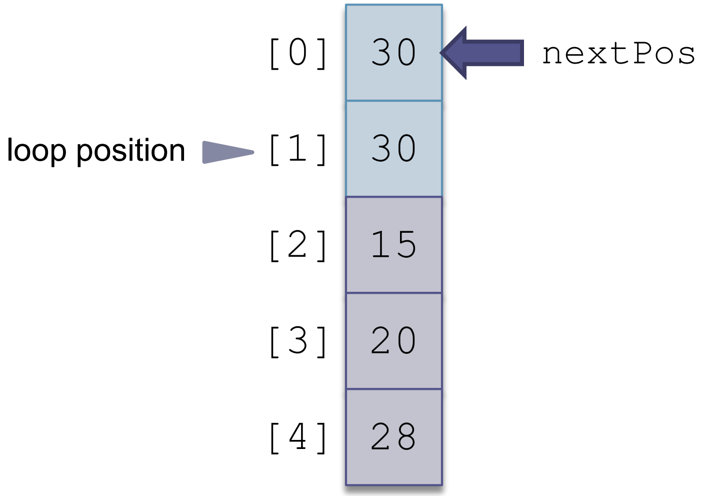
nextPos	0
nextVal	25



```
1. for each array element from the second (nextPos = 1) to the last
2.     nextPos is the position of the element to insert
3.     Save the value of the element to insert in nextVal
4.     while nextPos > 0 and the element at nextPos - 1 > nextVal
5.         Shift the element at nextPos - 1 to position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

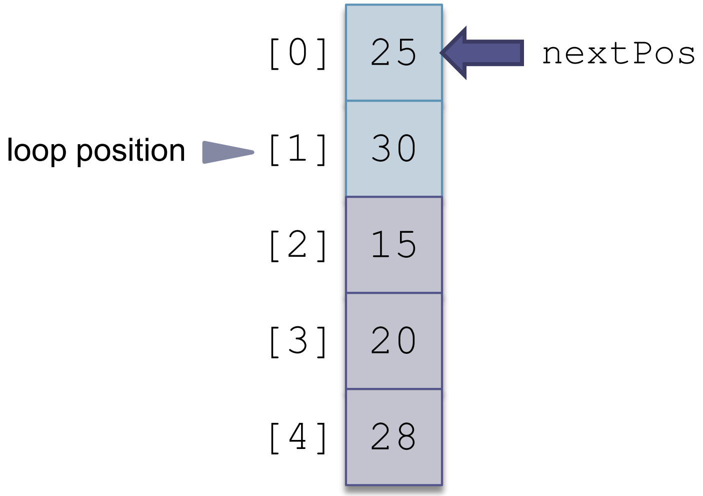
nextPos	0
nextVal	25



```
1. for each array element from the second
   (nextPos = 1) to the last
2.    nextPos is the position of the element to
      insert
3.    Save the value of the element to insert in
      nextVal
4.    while nextPos > 0 and the element
      at nextPos – 1 > nextVal
5.        Shift the element at nextPos – 1 to
          position nextPos
6.        Decrement nextPos by 1
7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

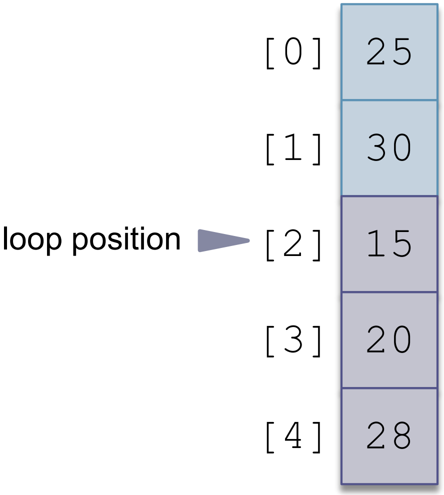
nextPos	0
nextVal	25



```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos - 1 > nextVal
5.         Shift the element at nextPos - 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	0
nextVal	25



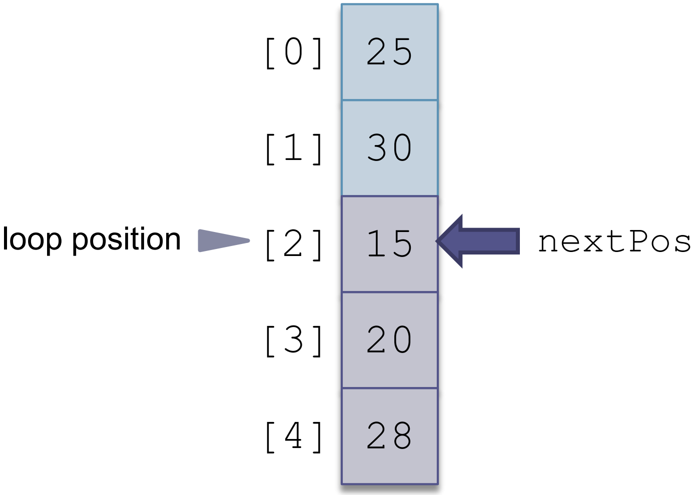
loop position

[0]	25
[1]	30
[2]	15
[3]	20
[4]	28

```
1. for each array element from the second (nextPos = 1) to the last
2.     nextPos is the position of the element to insert
3.     Save the value of the element to insert in nextVal
4.     while nextPos > 0 and the element at nextPos – 1 > nextVal
5.         Shift the element at nextPos – 1 to position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

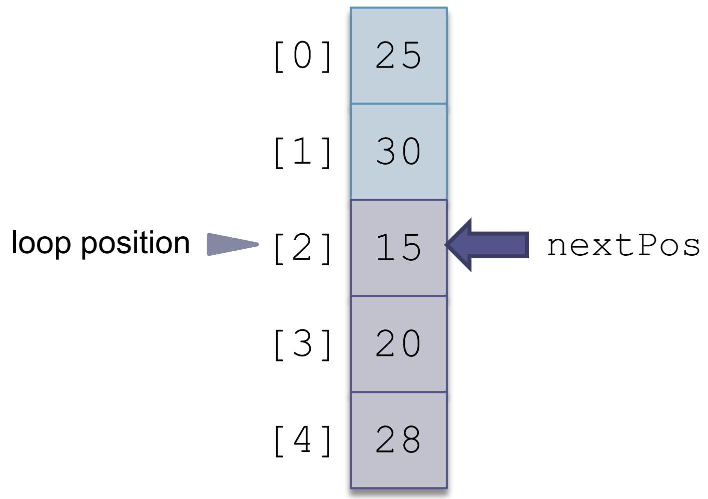
nextPos	2
nextVal	25



```
1. for each array element from the second
   (nextPos = 1) to the last
2.    nextPos is the position of the element to
      insert
3.    Save the value of the element to insert in
      nextVal
4.    while nextPos > 0 and the element
      at nextPos – 1 > nextVal
5.        Shift the element at nextPos – 1 to
          position nextPos
6.        Decrement nextPos by 1
7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

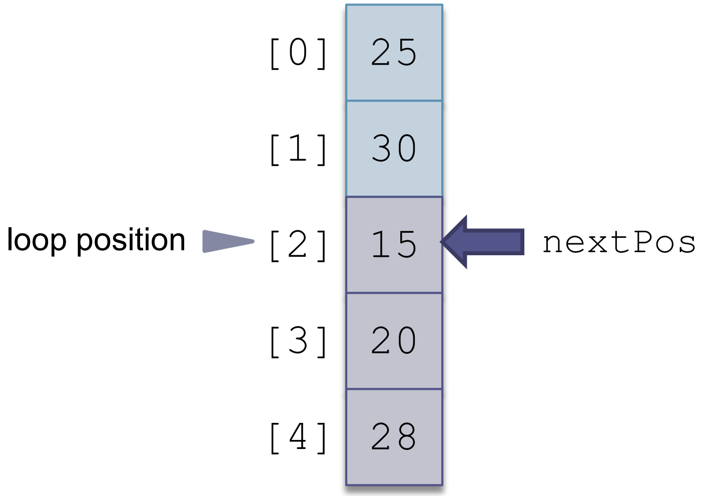
nextPos	2
nextVal	15



```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos - 1 > nextVal
5.         Shift the element at nextPos - 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

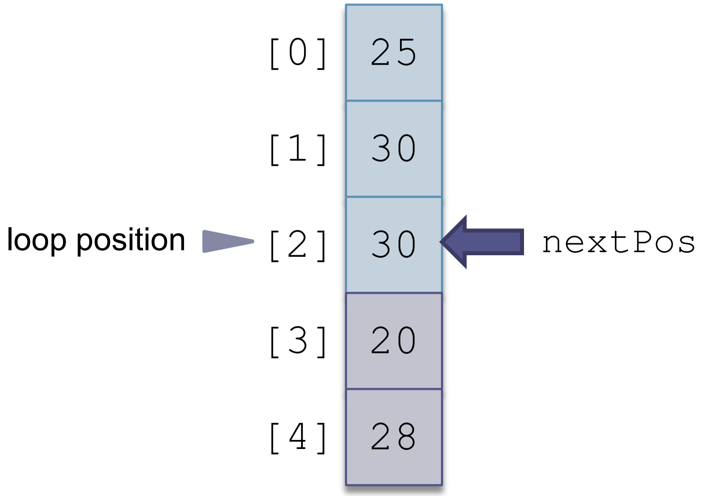
nextPos	2
nextVal	15



```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos - 1 > nextVal
5.         Shift the element at nextPos - 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

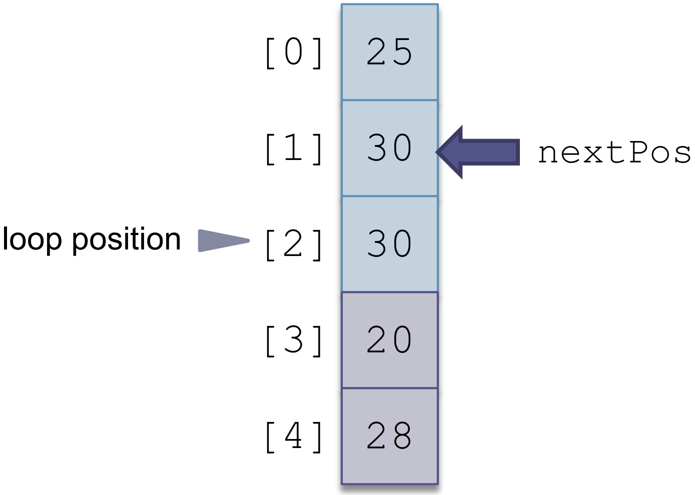
nextPos	2
nextVal	15



```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos – 1 > nextVal
5.         Shift the element at nextPos – 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

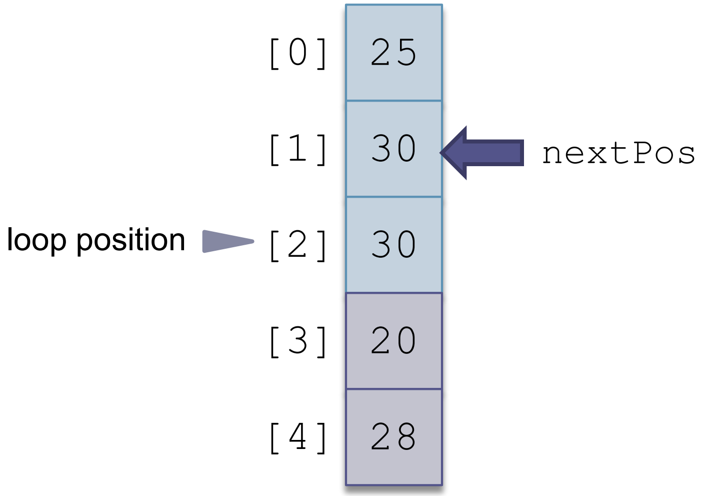
nextPos	1
nextVal	15



```
1. for each array element from the second (nextPos = 1) to the last
2.     nextPos is the position of the element to insert
3.     Save the value of the element to insert in nextVal
4.     while nextPos > 0 and the element at nextPos - 1 > nextVal
5.         Shift the element at nextPos - 1 to position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

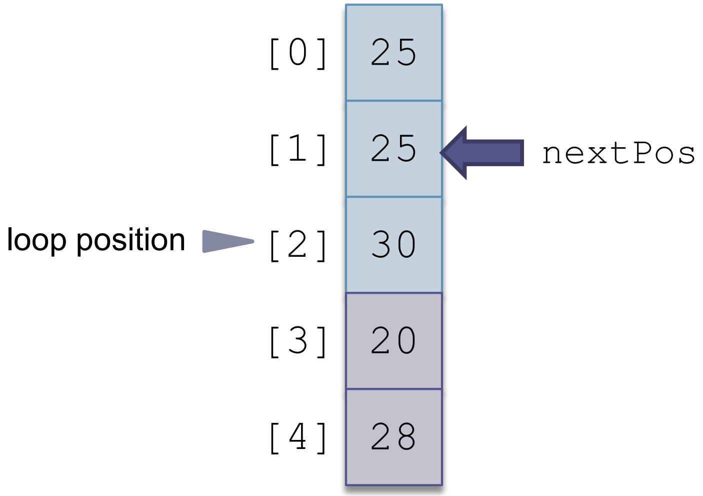
nextPos	1
nextVal	15



```
  1. for each array element from the second
     (nextPos = 1) to the last
  2.    nextPos is the position of the element to
        insert
  3.    Save the value of the element to insert in
        nextVal
► 4.    while nextPos > 0 and the element
        at nextPos – 1 > nextVal
  5.        Shift the element at nextPos – 1 to
            position nextPos
  6.        Decrement nextPos by 1
  7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

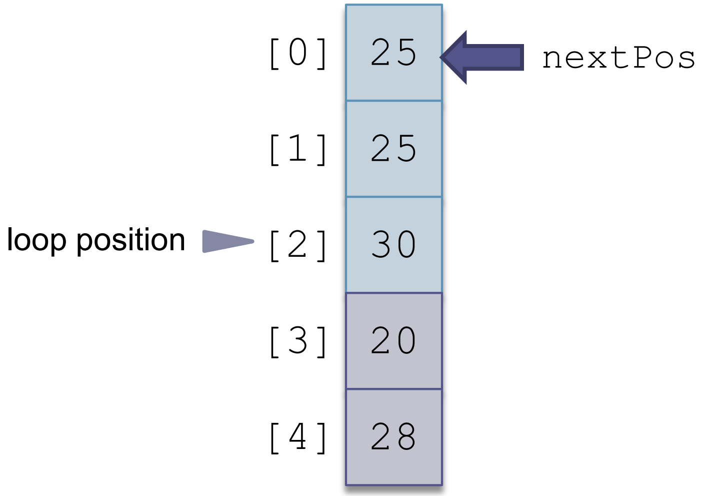
nextPos	1
nextVal	15



```
1. for each array element from the second (nextPos = 1) to the last
2.     nextPos is the position of the element to insert
3.     Save the value of the element to insert in nextVal
4.     while nextPos > 0 and the element at nextPos – 1 > nextVal
5.         Shift the element at nextPos – 1 to position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	0
nextVal	15



loop position

[0] 25 ← nextPos

[1] 25

[2] 30

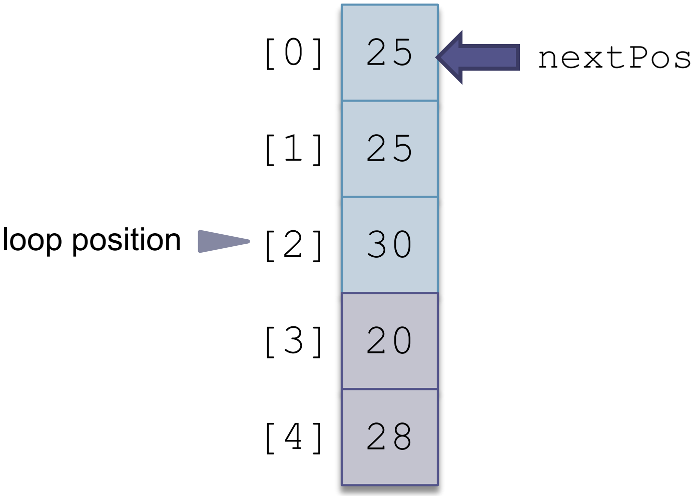
[3] 20

[4] 28

```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos − 1 > nextVal
5.         Shift the element at nextPos − 1 to
           position nextPos
▶ 6.       Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

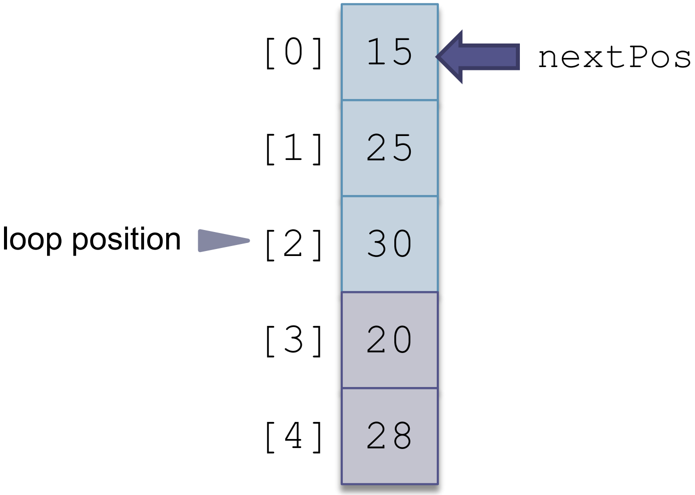
nextPos	0
nextVal	15



```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos – 1 > nextVal
5.         Shift the element at nextPos – 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

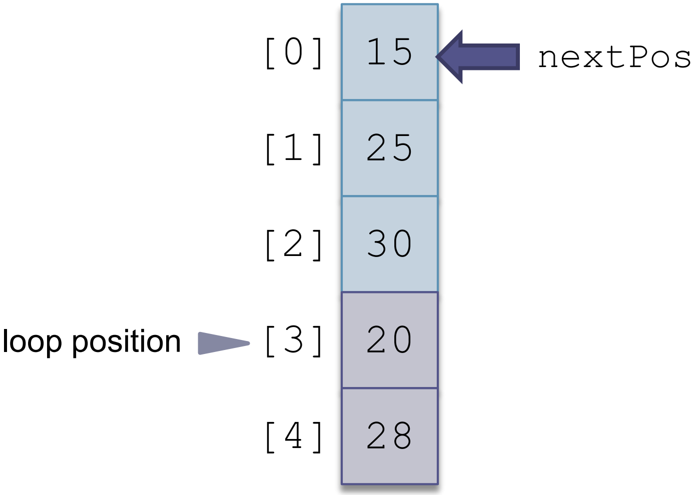
nextPos	0
nextVal	15



```
1. for each array element from the second
   (nextPos = 1) to the last
2.    nextPos is the position of the element to
      insert
3.    Save the value of the element to insert in
      nextVal
4.    while nextPos > 0 and the element
      at nextPos – 1 > nextVal
5.         Shift the element at nextPos – 1 to
           position nextPos
6.         Decrement nextPos by 1
7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

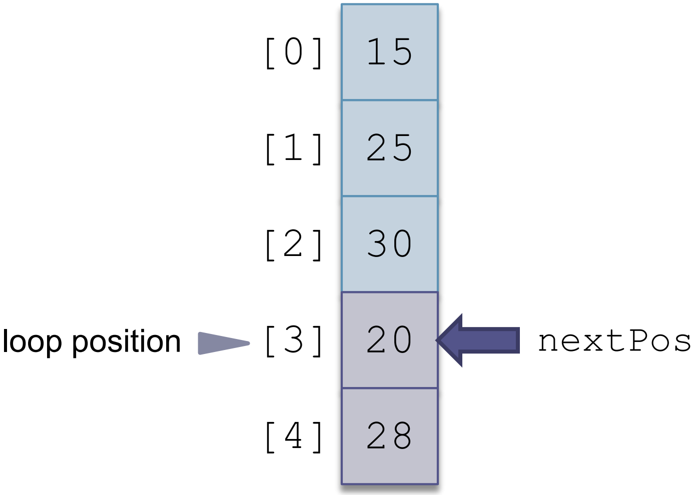
nextPos	0
nextVal	15



```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos - 1 > nextVal
5.         Shift the element at nextPos - 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	3
nextVal	15



[0]	15
[1]	25
[2]	30
[3]	20
[4]	28

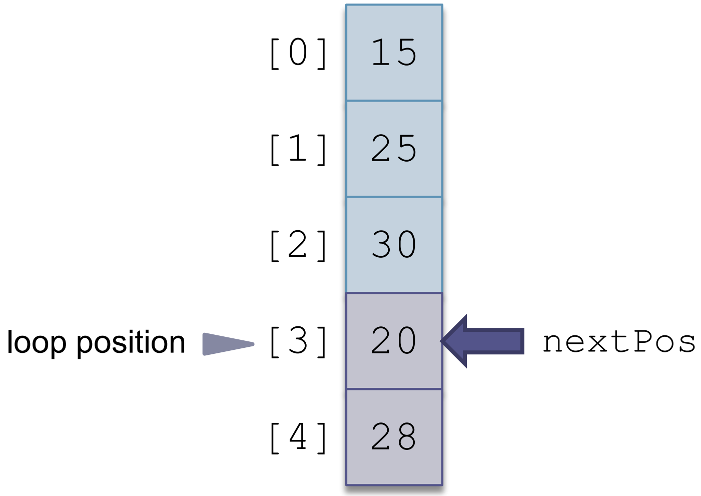
loop position

nextPos

```
  1. for each array element from the second
     (nextPos = 1) to the last
► 2.   nextPos is the position of the element to
       insert
  3.   Save the value of the element to insert in
       nextVal
  4.   while nextPos > 0 and the element
       at nextPos - 1 > nextVal
  5.       Shift the element at nextPos - 1 to
           position nextPos
  6.       Decrement nextPos by 1
  7.   Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

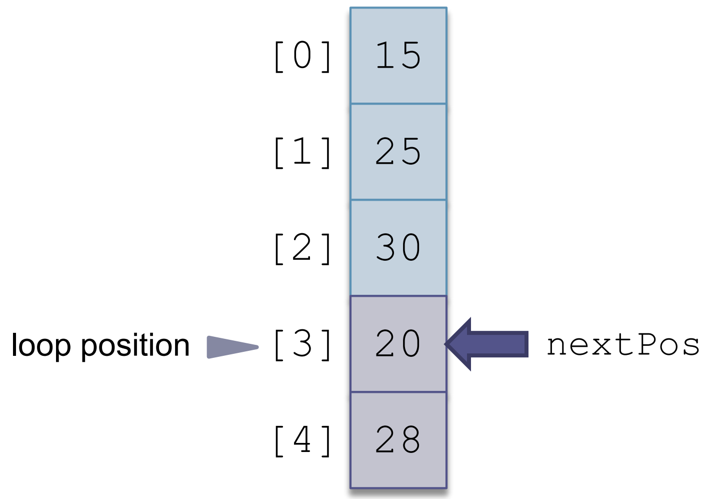
nextPos	3
nextVal	20



```
1. for each array element from the second (nextPos = 1) to the last
2.    nextPos is the position of the element to insert
3.    Save the value of the element to insert in nextVal
4.    while nextPos > 0 and the element at nextPos – 1 > nextVal
5.        Shift the element at nextPos – 1 to position nextPos
6.        Decrement nextPos by 1
7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

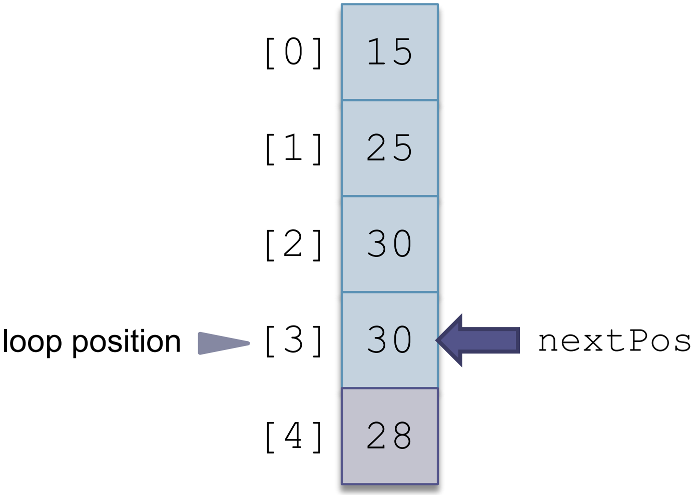
nextPos	3
nextVal	20



```
  1.  for each array element from the second
      (nextPos = 1) to the last
  2.    nextPos is the position of the element to
        insert
  3.    Save the value of the element to insert in
        nextVal
➤ 4.    while nextPos > 0 and the element
        at nextPos - 1 > nextVal
  5.        Shift the element at nextPos - 1 to
            position nextPos
  6.        Decrement nextPos by 1
  7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

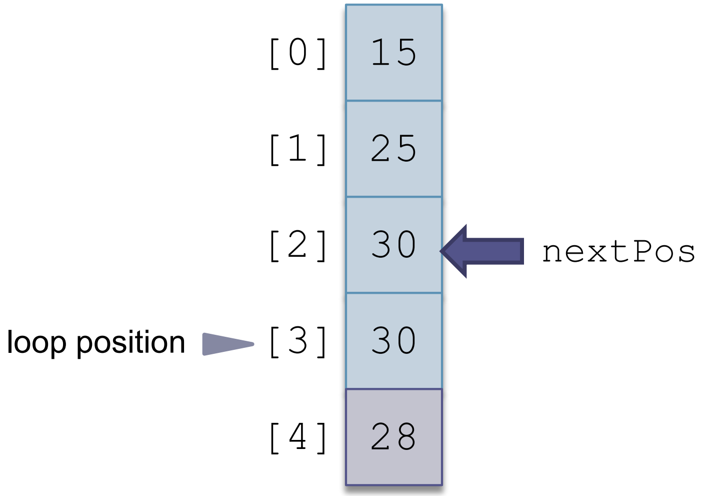
nextPos	3
nextVal	20



```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos - 1 > nextVal
▷ 5.       Shift the element at nextPos - 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

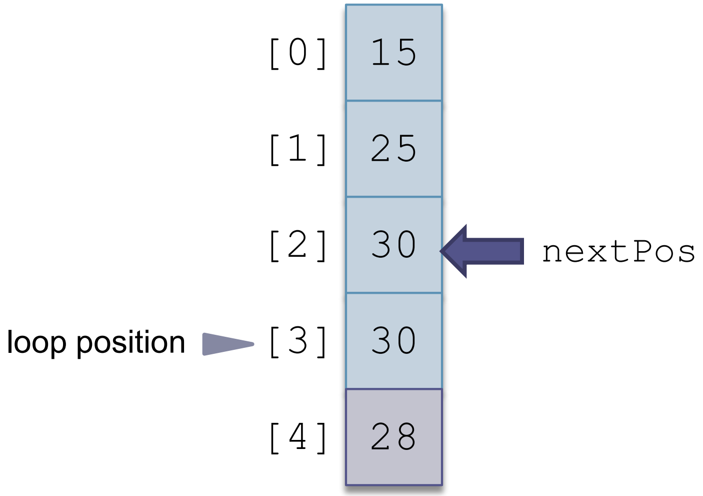
nextPos	2
nextVal	20



```
1. for each array element from the second (nextPos = 1) to the last
2.   nextPos is the position of the element to insert
3.   Save the value of the element to insert in nextVal
4.   while nextPos > 0 and the element at nextPos – 1 > nextVal
5.       Shift the element at nextPos – 1 to position nextPos
6.       Decrement nextPos by 1
7.   Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

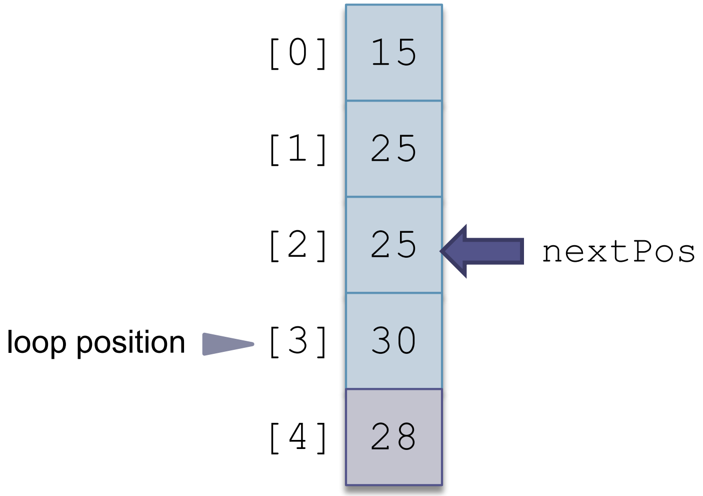
nextPos	2
nextVal	20



```
1.  for each array element from the second (nextPos = 1) to the last
2.    nextPos is the position of the element to insert
3.    Save the value of the element to insert in nextVal
4.    while nextPos > 0 and the element at nextPos - 1 > nextVal
5.        Shift the element at nextPos - 1 to position nextPos
6.        Decrement nextPos by 1
7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	2
nextVal	20



[0] 15

[1] 25

[2] 25 nextPos

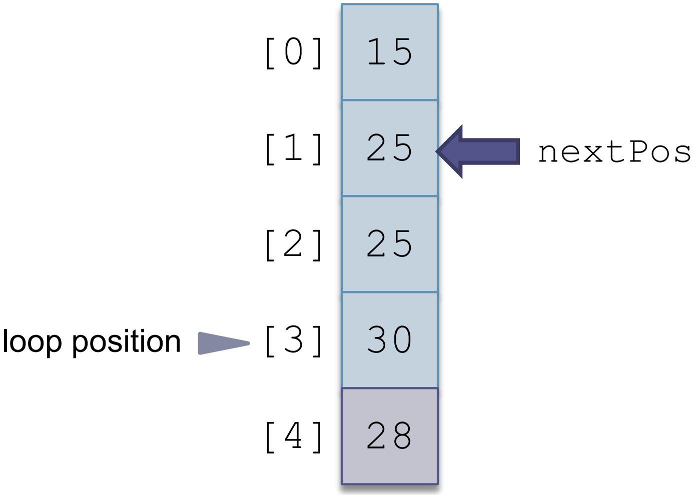
loop position [3] 30

[4] 28

```
1. for each array element from the second
   (nextPos = 1) to the last
2.    nextPos is the position of the element to
      insert
3.    Save the value of the element to insert in
      nextVal
4.    while nextPos > 0 and the element
      at nextPos - 1 > nextVal
5.        Shift the element at nextPos - 1 to
          position nextPos
6.        Decrement nextPos by 1
7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

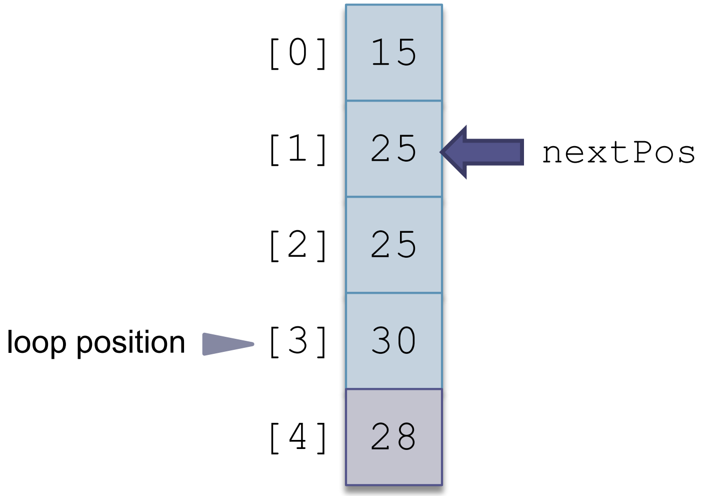
nextPos	1
nextVal	20



```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos – 1 > nextVal
5.         Shift the element at nextPos – 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

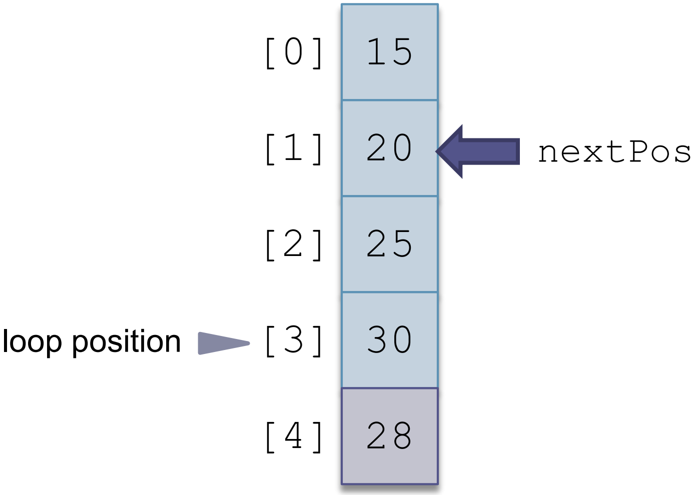
nextPos	1
nextVal	20



```
1. for each array element from the second
   (nextPos = 1) to the last
2.    nextPos is the position of the element to
      insert
3.    Save the value of the element to insert in
      nextVal
4.    while nextPos > 0 and the element
      at nextPos - 1 > nextVal
5.        Shift the element at nextPos - 1 to
          position nextPos
6.        Decrement nextPos by 1
7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	1
nextVal	20



```
1. for each array element from the second (nextPos = 1) to the last
2.    nextPos is the position of the element to insert
3.    Save the value of the element to insert in nextVal
4.    while nextPos > 0 and the element at nextPos – 1 > nextVal
5.        Shift the element at nextPos – 1 to position nextPos
6.        Decrement nextPos by 1
7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

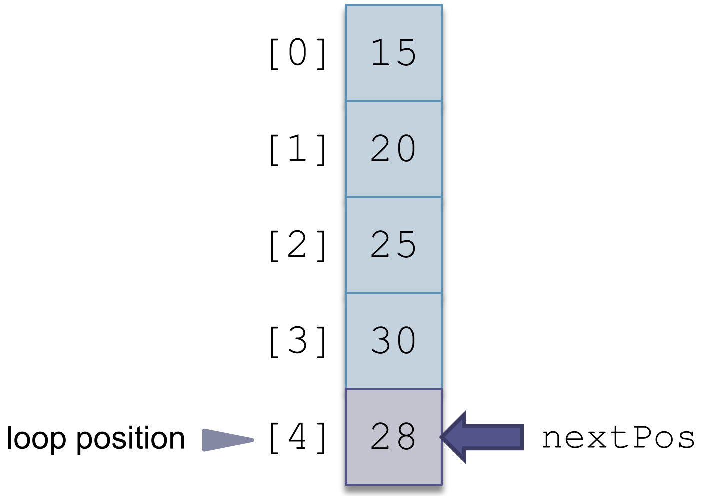
nextPos	1
nextVal	20

	[0]	15
	[1]	20
	[2]	25
	[3]	30
loop position ►	[4]	28

```
► 1. for each array element from the second
        (nextPos = 1) to the last
  2.    nextPos is the position of the element to
        insert
  3.    Save the value of the element to insert in
        nextVal
  4.    while nextPos > 0 and the element
        at nextPos – 1 > nextVal
  5.        Shift the element at nextPos – 1 to
            position nextPos
  6.        Decrement nextPos by 1
  7.    Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

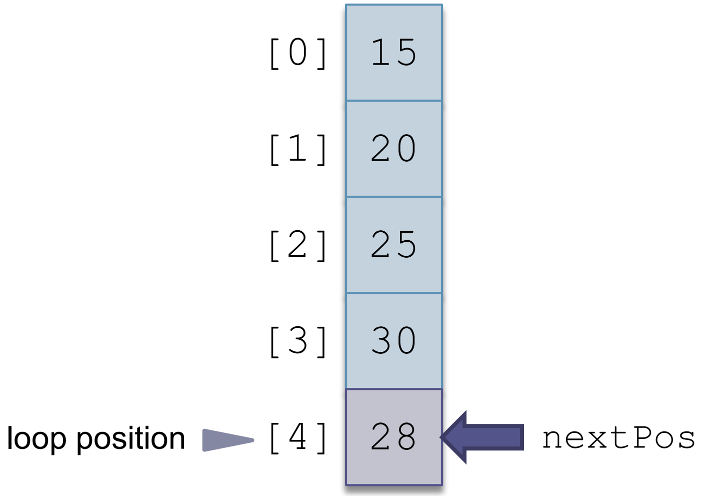
nextPos	4
nextVal	20



```
1.  for each array element from the second
    (nextPos = 1) to the last
2.      nextPos is the position of the element to
        insert
3.      Save the value of the element to insert in
        nextVal
4.      while nextPos > 0 and the element
        at nextPos – 1 > nextVal
5.          Shift the element at nextPos – 1 to
            position nextPos
6.          Decrement nextPos by 1
7.      Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	4
nextVal	28



loop position

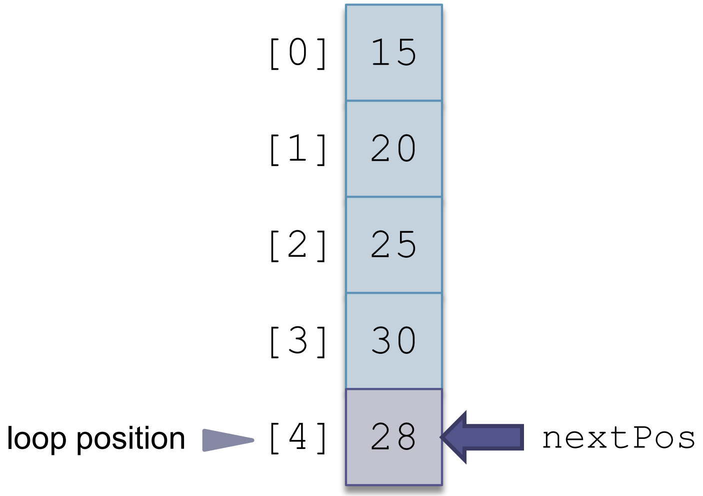
[0]	15
[1]	20
[2]	25
[3]	30
[4]	28

nextPos

```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos – 1 > nextVal
5.         Shift the element at nextPos – 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

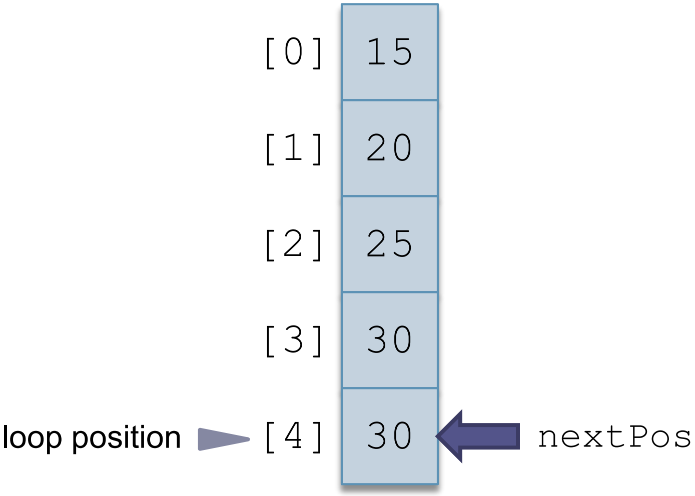
nextPos	4
nextVal	28



```
1. for each array element from the second (nextPos = 1) to the last
2.     nextPos is the position of the element to insert
3.     Save the value of the element to insert in nextVal
4.     while nextPos > 0 and the element at nextPos - 1 > nextVal
5.         Shift the element at nextPos - 1 to position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	4
nextVal	28

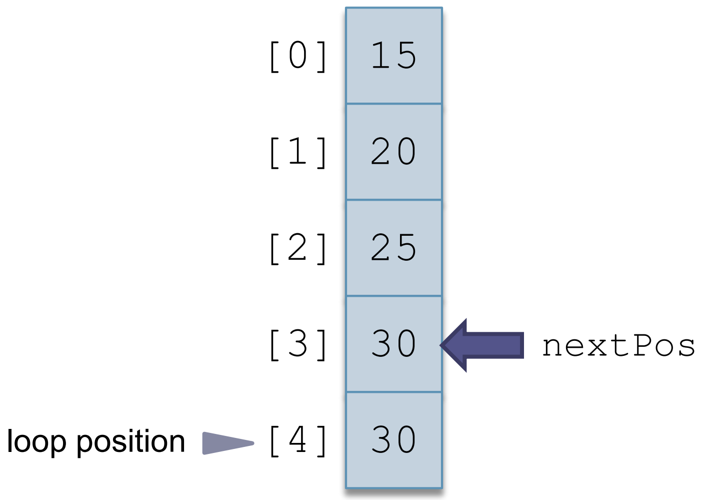


[0]	15
[1]	20
[2]	25
[3]	30
[4]	30

```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos – 1 > nextVal
5.         Shift the element at nextPos – 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	3
nextVal	28


[0] 15

[1] 20

[2] 25

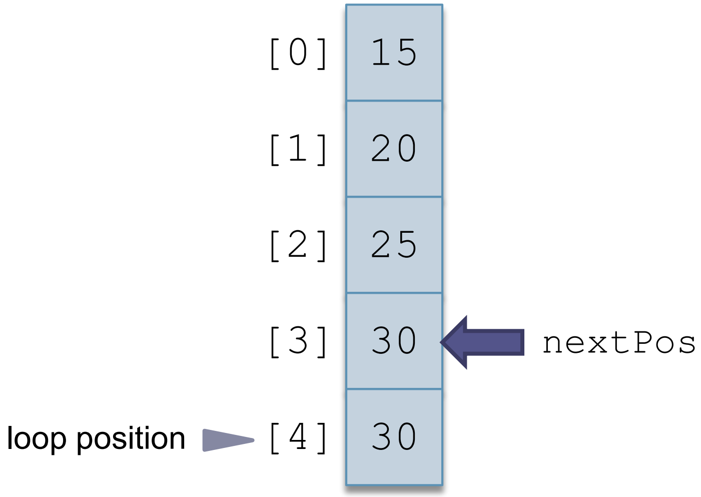
[3] 30

[4] 30

```
1. for each array element from the second (nextPos = 1) to the last
2.     nextPos is the position of the element to insert
3.     Save the value of the element to insert in nextVal
4.     while nextPos > 0 and the element at nextPos - 1 > nextVal
5.         Shift the element at nextPos - 1 to position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	3
nextVal	28



[0] 15

[1] 20

[2] 25

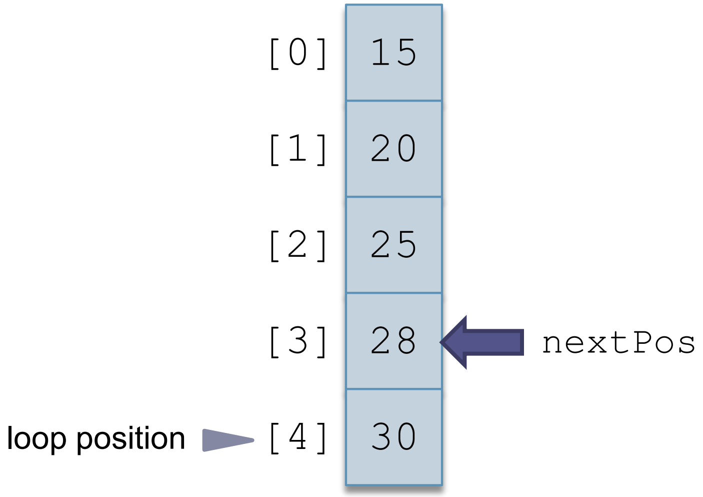
[3] 30 nextPos

loop position [4] 30

```
1. for each array element from the second
   (nextPos = 1) to the last
2.     nextPos is the position of the element to
       insert
3.     Save the value of the element to insert in
       nextVal
4.     while nextPos > 0 and the element
       at nextPos – 1 > nextVal
5.         Shift the element at nextPos – 1 to
           position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	3
nextVal	28



```
  1. for each array element from the second
     (nextPos = 1) to the last
  2.   nextPos is the position of the element to
       insert
  3.   Save the value of the element to insert in
       nextVal
  4.   while nextPos > 0 and the element
       at nextPos - 1 > nextVal
  5.       Shift the element at nextPos - 1 to
           position nextPos
  6.       Decrement nextPos by 1
▶ 7.   Insert nextVal at nextPos
```

Trace of Insertion Sort Refinement (cont.)

nextPos	3
nextVal	28

[0]	15
[1]	20
[2]	25
[3]	28
[4]	30

```
1. for each array element from the second (nextPos = 1) to the last
2.     nextPos is the position of the element to insert
3.     Save the value of the element to insert in nextVal
4.     while nextPos > 0 and the element at nextPos – 1 > nextVal
5.         Shift the element at nextPos – 1 to position nextPos
6.         Decrement nextPos by 1
7.     Insert nextVal at nextPos
```

Analysis of Insertion Sort

- The insertion step is performed $n - 1$ times
- In the worst case, all elements in the sorted subarray are compared to `nextVal` for each insertion
- The maximum number of comparisons then will be:

$$1 + 2 + 3 + \ldots + (n - 2) + (n - 1)$$

which is $O(n^2)$

Analysis of Insertion Sort (cont.)

- In the best case (when the array is sorted already), only one comparison is required for each insertion
- In the best case, the number of comparisons is O(*n*)
- The number of shifts performed during an insertion is one less than the number of comparisons
- Or, when the new value is the smallest so far, it is the same as the number of comparisons
- A shift in an insertion sort requires movement of only 1 item, while an exchange in a selection sort involves a temporary item and the movement of three items
  - The item moved may be a primitive or an object reference
  - The objects themselves do not change their locations

Code for Insertion Sort

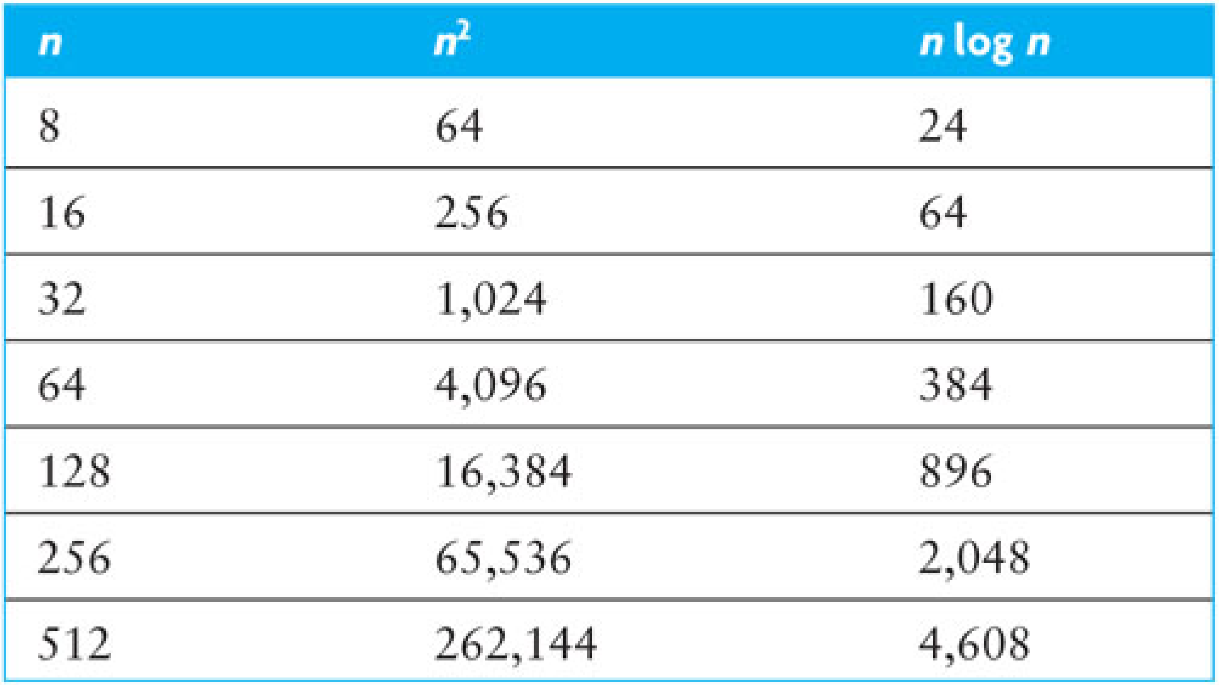
```
public class InsertionSort {
   /** Sort the table using insertion sort algorithm.
       @param table The array to be sorted
    */
    public static <T extends Comparable<T>> void sort(T[] table) {
            for (int nextPos = 1; nextPos < table.length; nextPos++) {
                  // Invariant: table[0 . . nextPos - 1] is sorted.
                  // Insert element at position nextPos
                  // in the sorted subarray.
                  insert(table, nextPos);
             } // End for.
     } // End sort.

    /** Insert the element at nextPos where it belongs in the array.
       @param table The array being sorted
       @param nextPos The position of the element to insert
     */
     private static <T extends Comparable<T>> void insert(T[] table, int nextPos) {
          T nextVal = table[nextPos];
          // Element to insert.
          while (nextPos > 0 && nextVal.compareTo(table [nextPos - 1]) < 0) {
                  table[nextPos] = table[nextPos - 1];
                  nextPos-- ;          // Shift down.
                  // repeat loop - Check next smaller element.
           }
           // Insert nextVal at nextPos.
           table[nextPos] = nextVal;
     }
}
```

Comparison of Quadratic Sorts

Sort kind	Comparisons	Comparisons	Exchanges	Exchanges
	Best	Worst	Best	Worst
Selection sort	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$
Insertion sort	$O(n)$	$O(n^2)$	$O(1)$	$O(n^2)$

Comparison of growth rates



n	n^2	$n \log n$
8	64	24
16	256	64
32	1,024	160
64	4,096	384
128	16,384	896
256	65,536	2,048
512	262,144	4,608

Comparison of Quadratic Sorts (cont.)

- Insertion sort
  - gives the best performance for most arrays
  - takes advantage of any partial sorting in the array and uses less costly shifts
- Niether quadratic search algorithm is particularly good for large arrays (*n* > 1000)
- The best sorting algorithms provide *n* log *n* average case performance

Comparison of Quadratic Sorts (cont.)

- All quadratic sorts require storage for the array being sorted
- However, the array is sorted in place
- While there are also storage requirements for variables, for large *n*, the size of the array dominates and extra space usage is O(1)

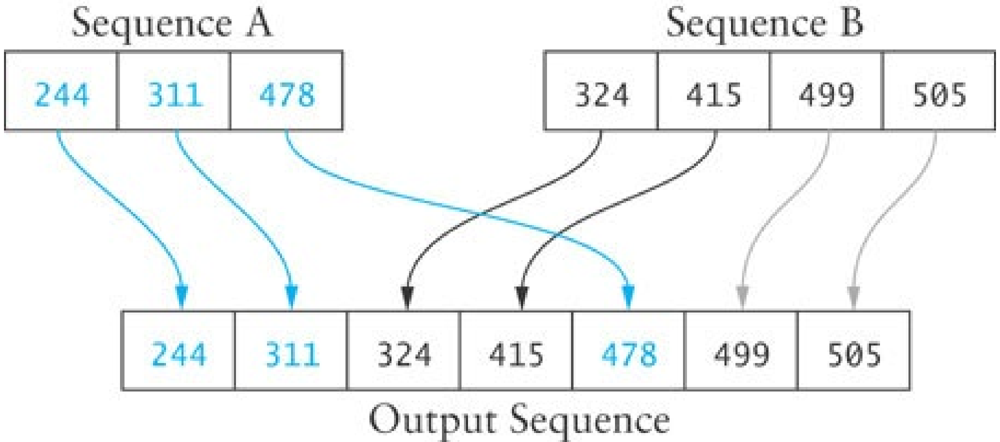
Merge

- A *merge* is a common data processing operation performed on two sequences of data with the following characteristics
  - Both sequences contain items with a common `compareTo` method
  - The objects in both sequences are ordered in accordance with this `compareTo` method
- The result is a third sequence containing all the data from the first two sequences

Merge Algorithm

Merge Algorithm

1. Access the first item from both sequences.
2. `while` not finished with either sequence
3. Compare the current items from the two sequences, copy the smaller current item to the output sequence, and access the next item from the input sequence whose item was copied.
4. Copy any remaining items from the first sequence to the output sequence.
5. Copy any remaining items from the second sequence to the output sequence.



Analysis of Merge

- For two input sequences each containing *n* elements, each element needs to move from its input sequence to the output sequence
- Merge time is O(*n*)
- Space requirements
  - The array cannot be merged in place
  - Additional space usage is O(*n*)

Code for Merge Method

```
private static <T extends Comparable<T>>  void merge(T[] outputSequence,
                                          int dest;
                                T[] leftSequence,
                                T[] rightSequence) {
    int i = 0; // Index into the left input sequence.
    int j = 0; // Index into the right input sequence.
    int k = dest; // Index into the output sequence.
    // While there is data in both input sequences
    while (i < leftSequence.length && j < rightSequence.length) {
        // Find the smaller and
        // insert it into the output sequence.
        if (leftSequence[i].compareTo(rightSequence[j]) < 0) {
            outputSequence[k++] = leftSequence[i++];
        } else {
            outputSequence[k++] = rightSequence[j++];
        }
    }
```

dest allows the merge result to start at a position > 0 in the output array.

Code for Merge Method (cont.)

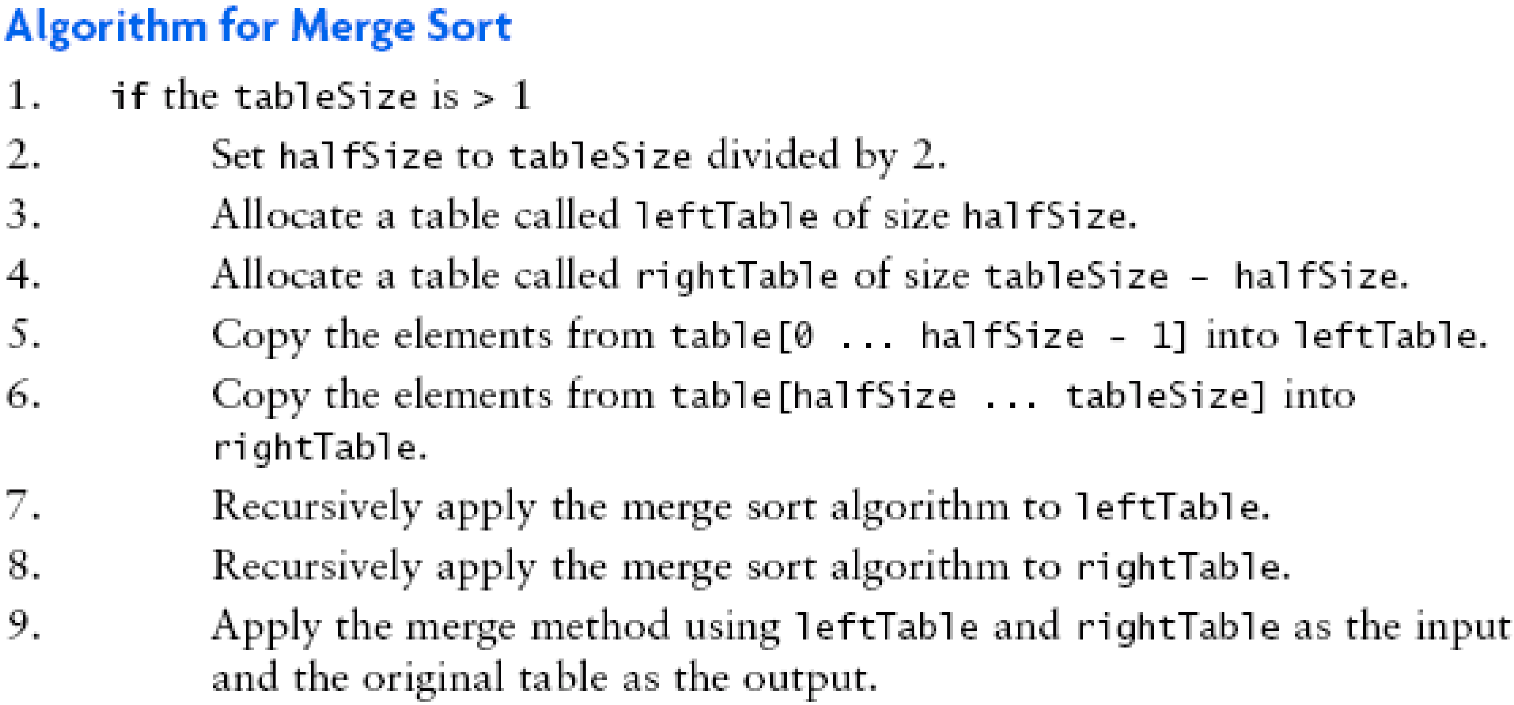
```
    // assert: one of the sequences has more items to copy.
    // Copy remaining input from left sequence into the output.
    while (i < leftSequence.length) {
        outputSequence[k++] = leftSequence[i++];
    }
    // Copy remaining input from right sequence into output.
    while (j < rightSequence.length) {
        outputSequence[k++] = rightSequence[j++];
    }
}
```

Merge Sort

- We can modify merging to sort a single, unsorted array
  1. Split the array into two halves
  2. Sort the left half
  3. Sort the right half
  4. Merge the two

- This algorithm can be written with a recursive step

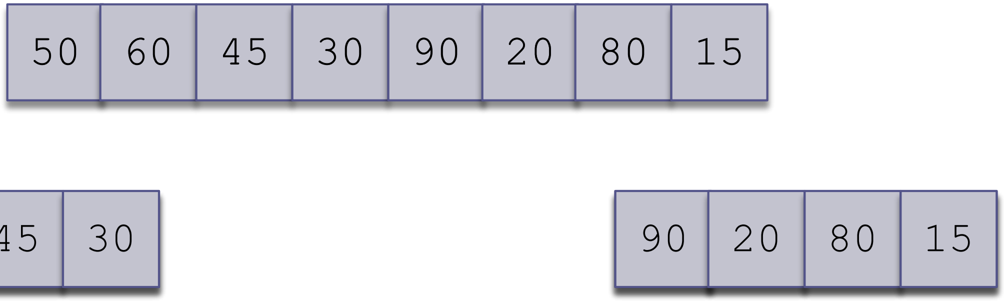
(recursive) Algorithm for Merge Sort



Algorithm for Merge Sort

```
1.  if the tableSize is > 1
2.      Set halfSize to tableSize divided by 2.
3.      Allocate a table called leftTable of size halfSize.
4.      Allocate a table called rightTable of size tableSize – halfSize.
5.      Copy the elements from table[0 ... halfSize - 1] into leftTable.
6.      Copy the elements from table[halfSize ... tableSize] into
        rightTable.
7.      Recursively apply the merge sort algorithm to leftTable.
8.      Recursively apply the merge sort algorithm to rightTable.
9.      Apply the merge method using leftTable and rightTable as the input
        and the original table as the output.
```

Trace of Merge Sort

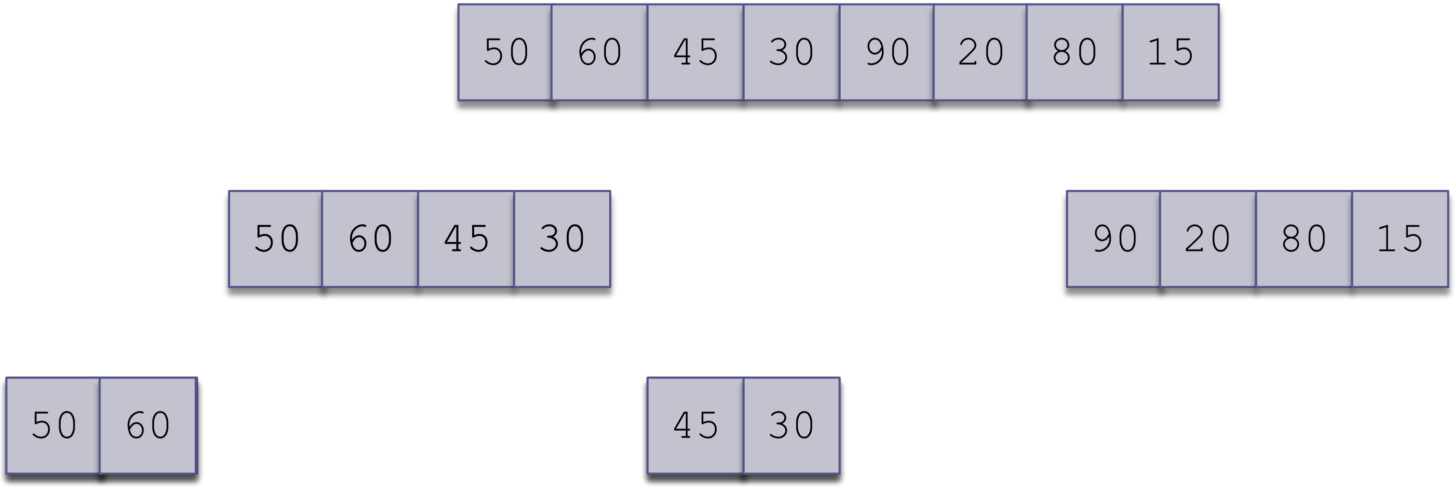


50	60	45	30	90	20	80	15

50	60	45	30

90	20	80	15

Trace of Merge Sort (cont.)



50 60 45 30 90 20 80 15

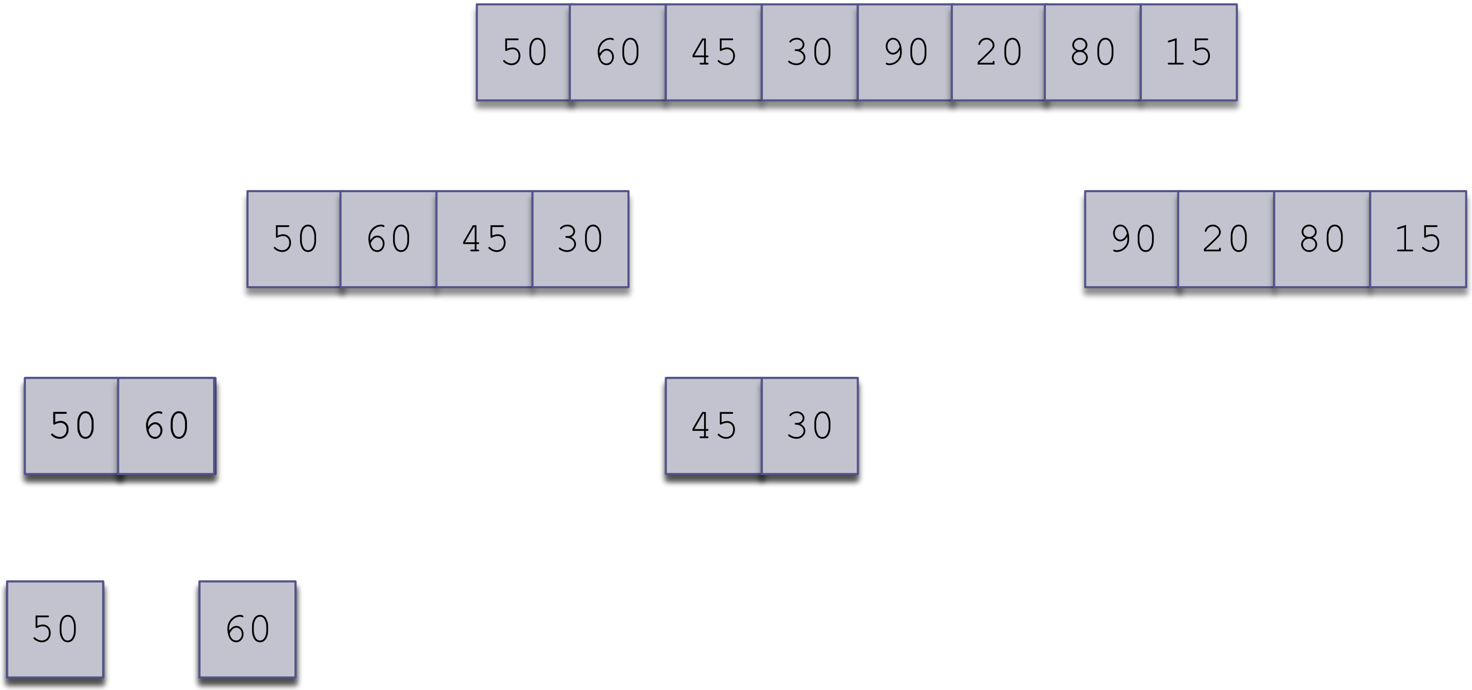
50 60 45 30

90 20 80 15

50 60

45 30

Trace of Merge Sort (cont.)



50 60 45 30 90 20 80 15

50 60 45 30

90 20 80 15

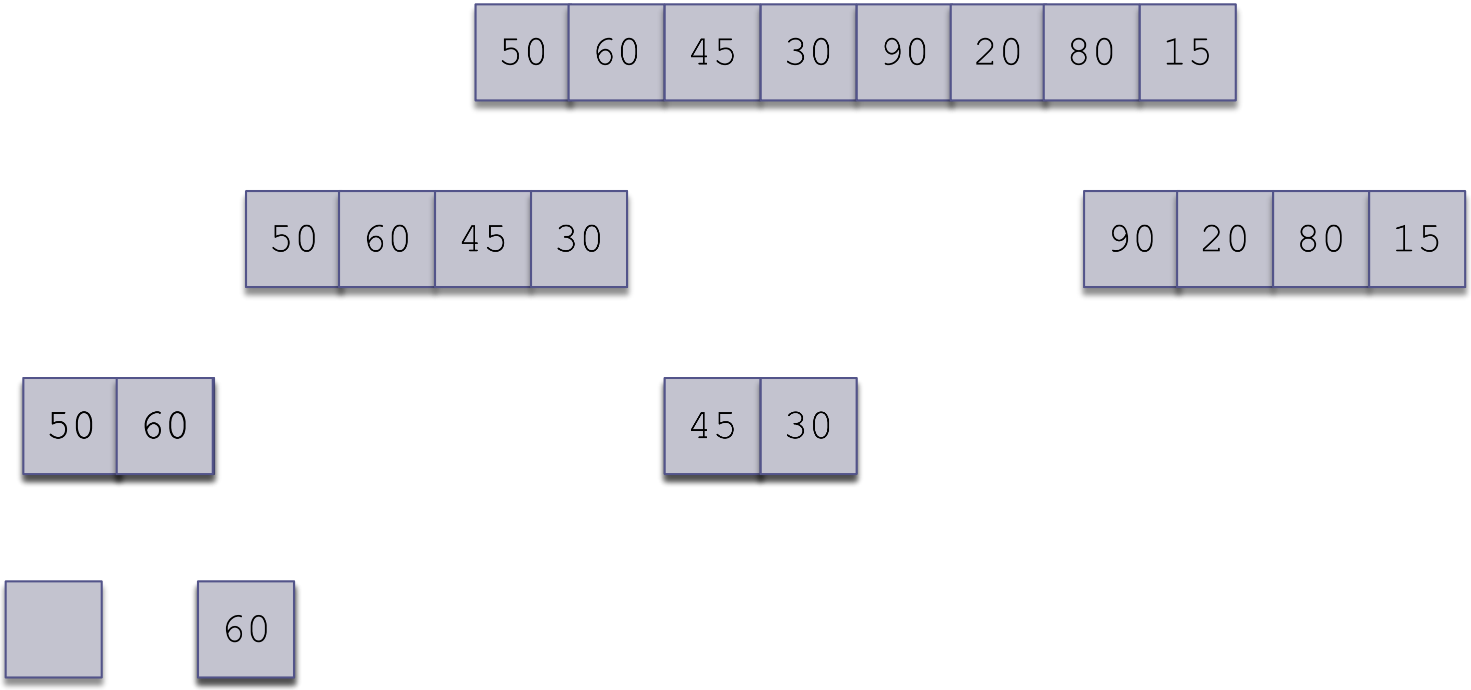
50 60

45 30

50

60

Trace of Merge Sort (cont.)



50 60 45 30 90 20 80 15

50 60 45 30

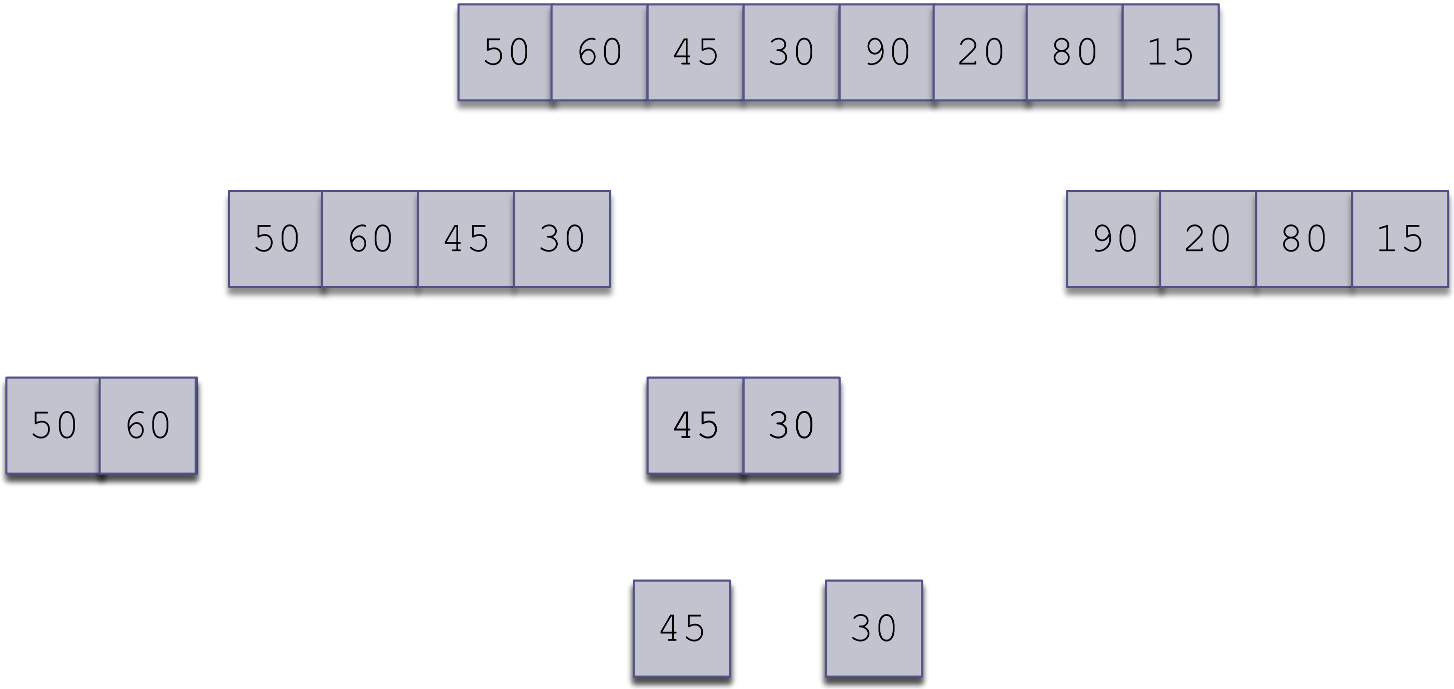
90 20 80 15

50 60

45 30

60

Trace of Merge Sort (cont.)



50 60 45 30 90 20 80 15

50 60 45 30

90 20 80 15

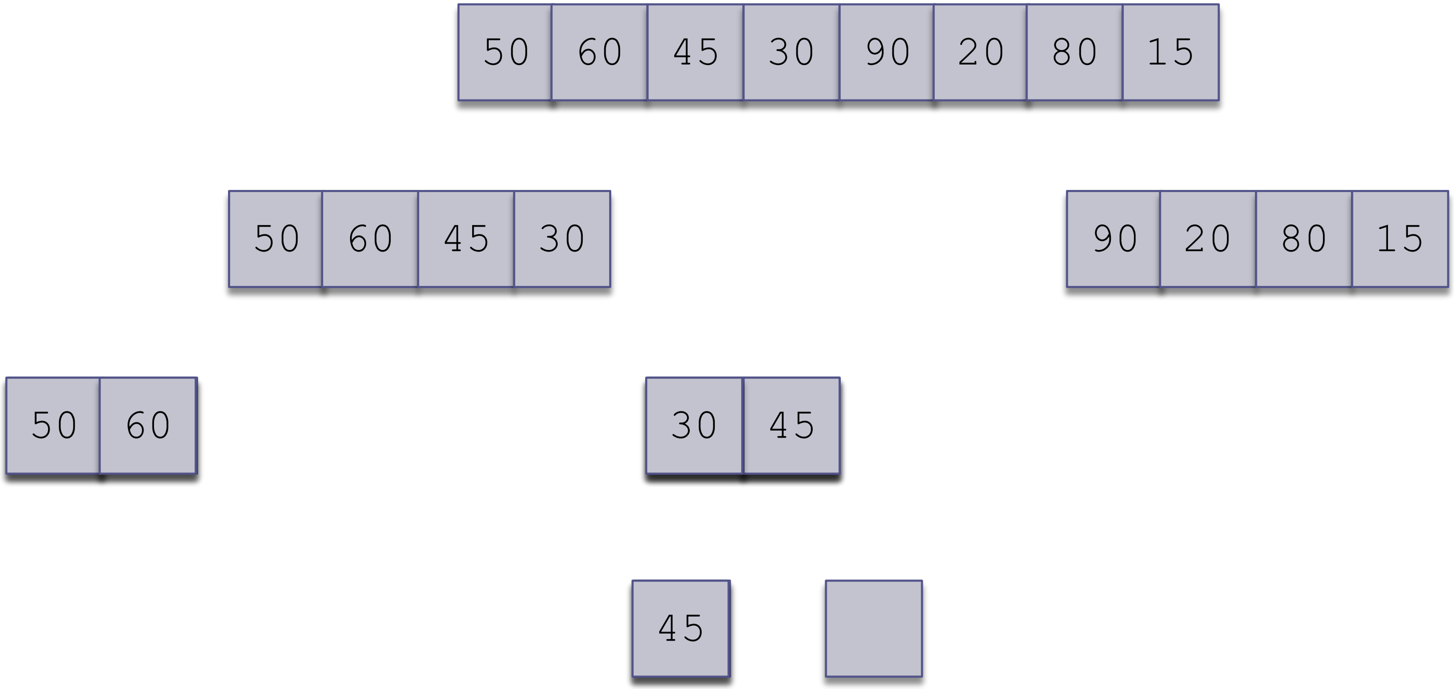
50 60

45 30

45

30

Trace of Merge Sort (cont.)



50 60 45 30 90 20 80 15

50 60 45 30

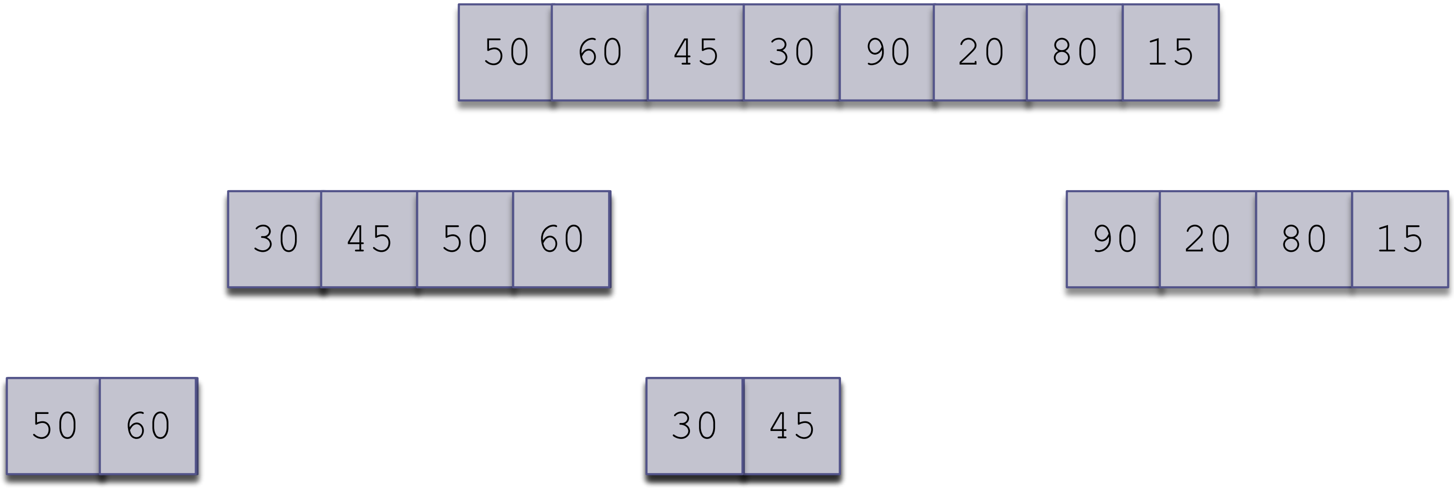
90 20 80 15

50 60

30 45

45

Trace of Merge Sort (cont.)



50 60 45 30 90 20 80 15

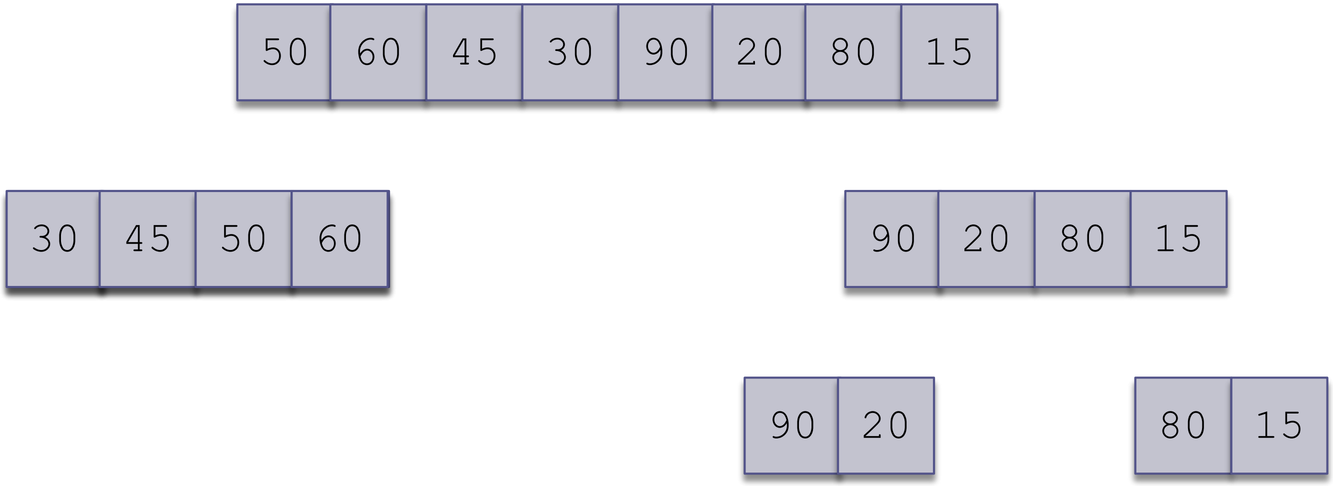
30 45 50 60

90 20 80 15

50 60

30 45

Trace of Merge Sort (cont.)



50 60 45 30 90 20 80 15

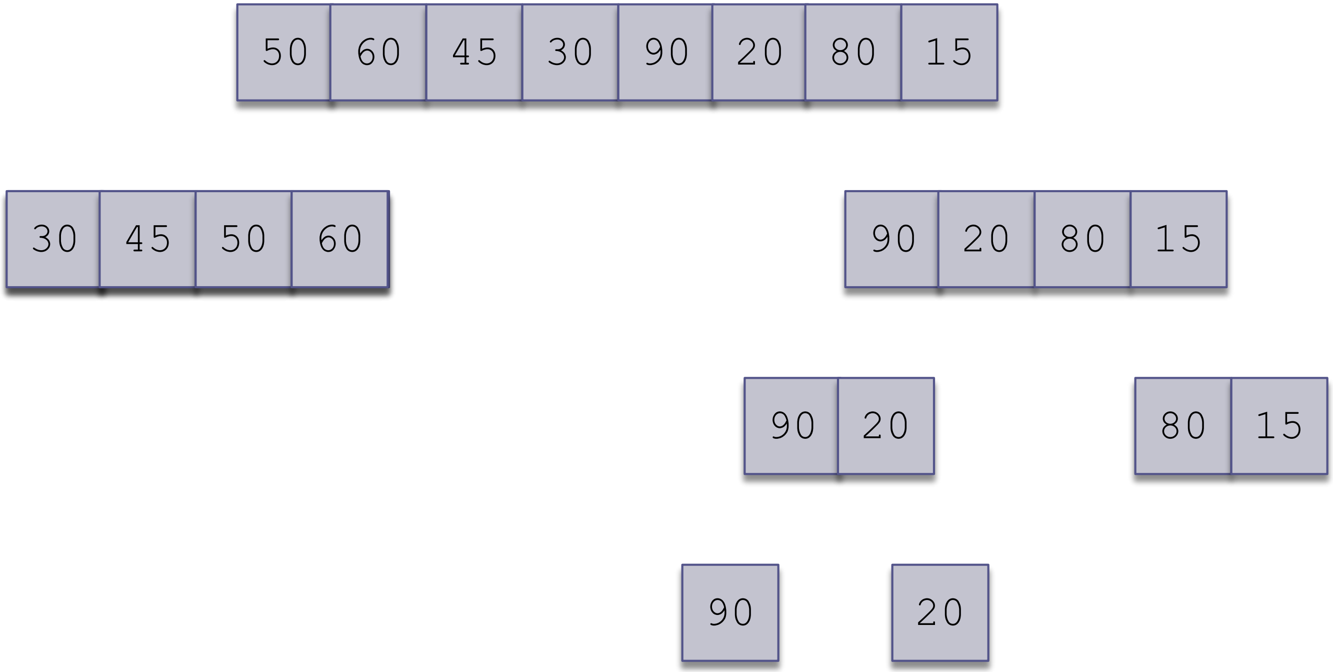
30 45 50 60

90 20 80 15

90 20

80 15

Trace of Merge Sort (cont.)



50 60 45 30 90 20 80 15

30 45 50 60

90 20 80 15

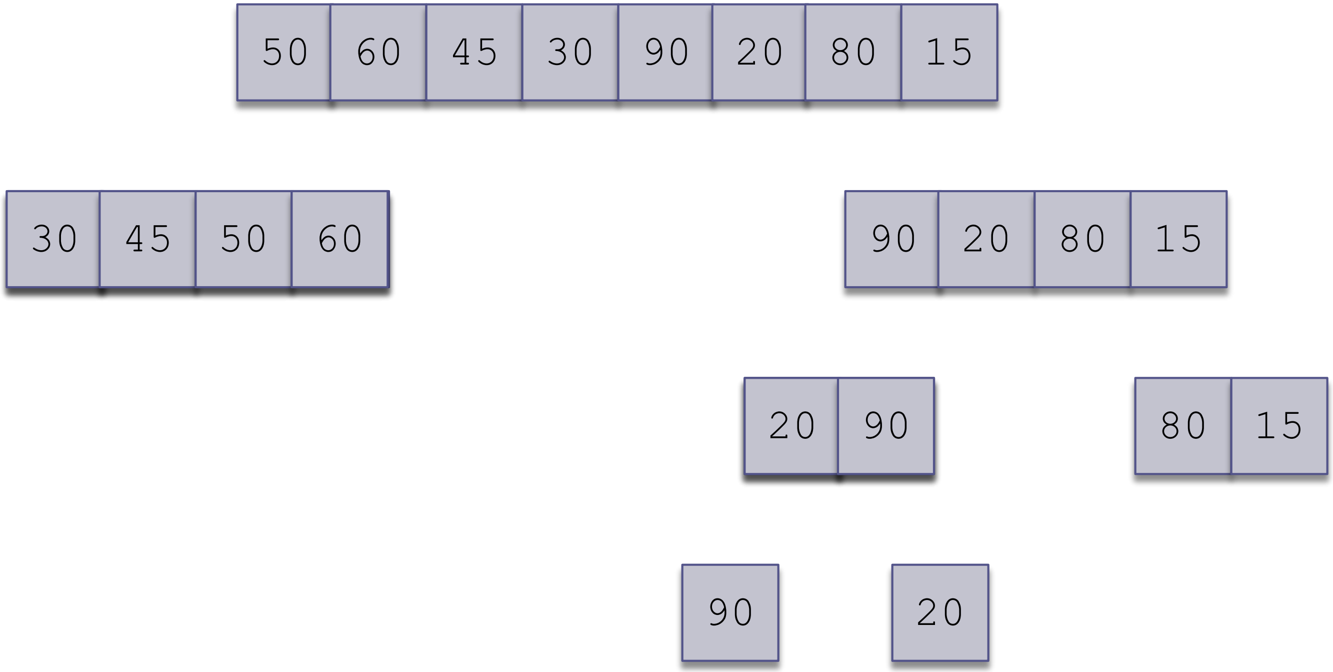
90 20

80 15

90

20

Trace of Merge Sort (cont.)



50 60 45 30 90 20 80 15

30 45 50 60

90 20 80 15

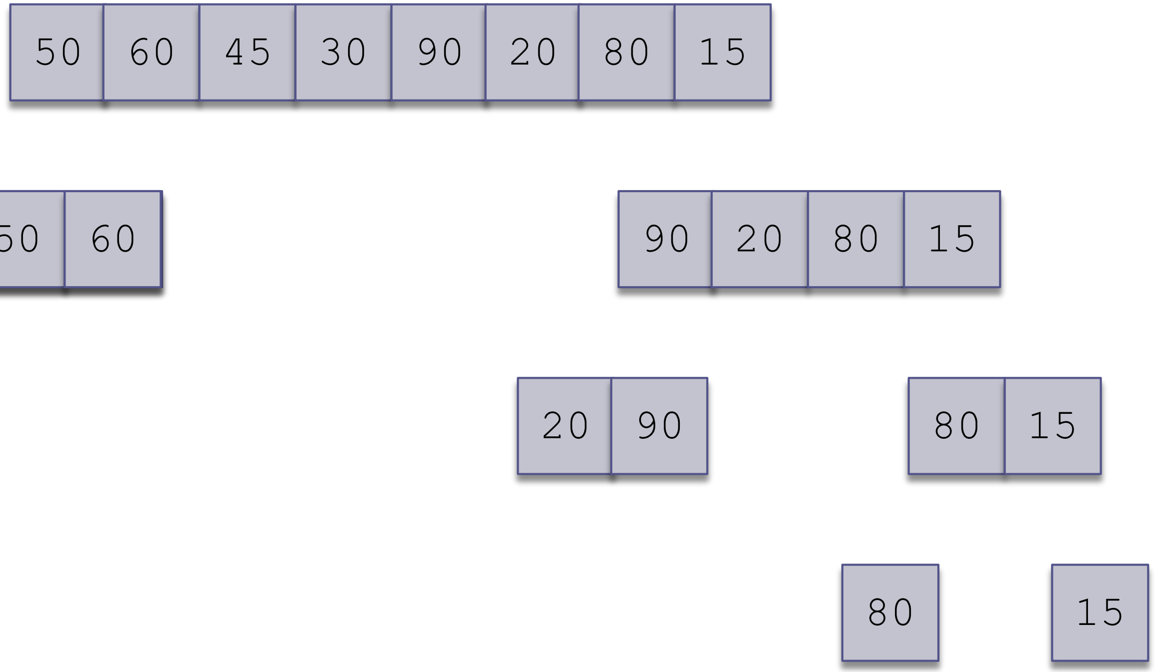
20 90

80 15

90

20

Trace of Merge Sort (cont.)



50 60 45 30 90 20 80 15

30 45 50 60

90 20 80 15

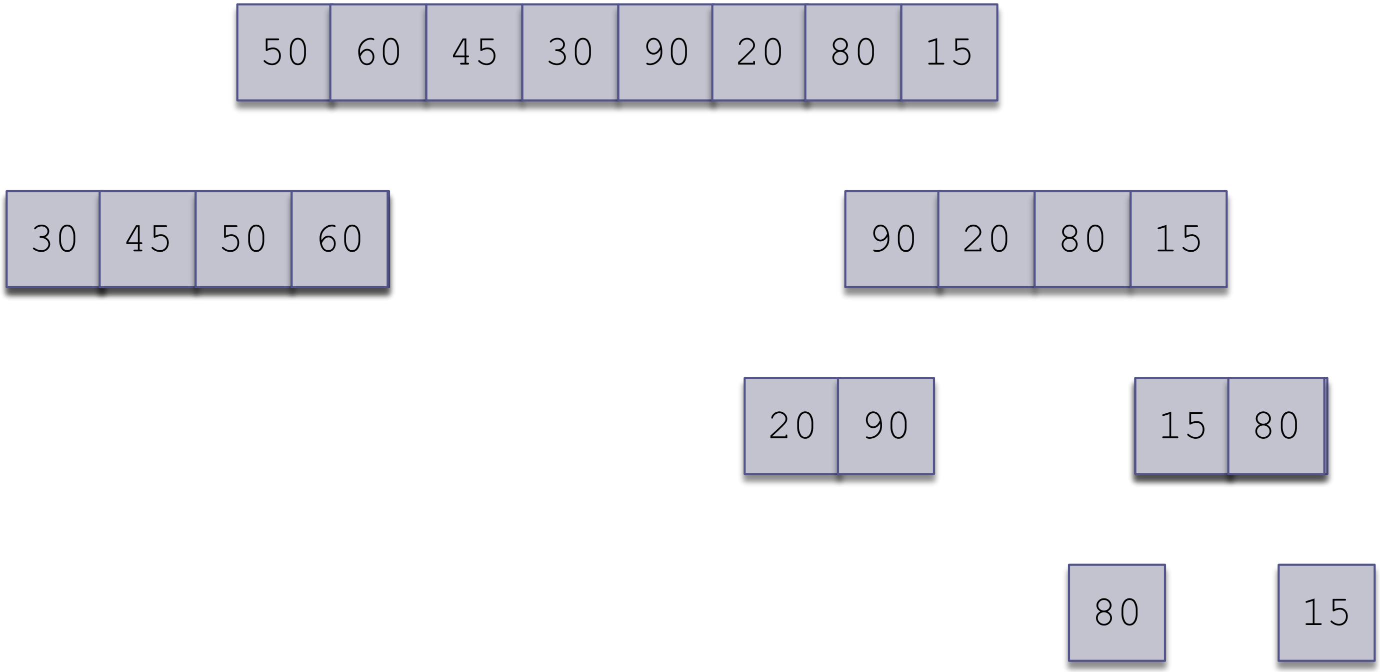
20 90

80 15

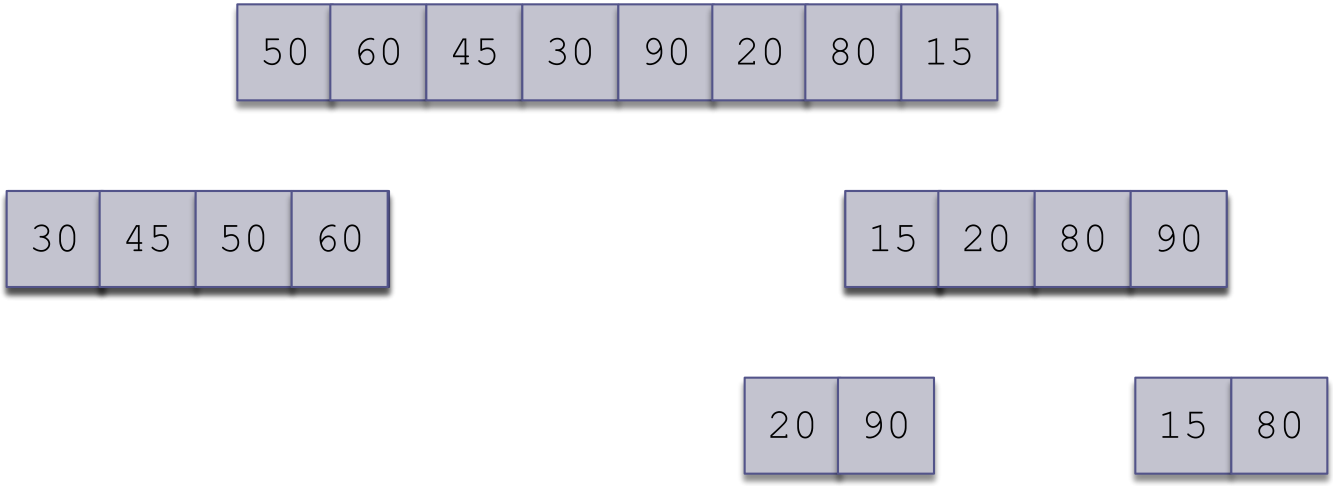
80

15

Trace of Merge Sort (cont.)



Trace of Merge Sort (cont.)



50 60 45 30 90 20 80 15

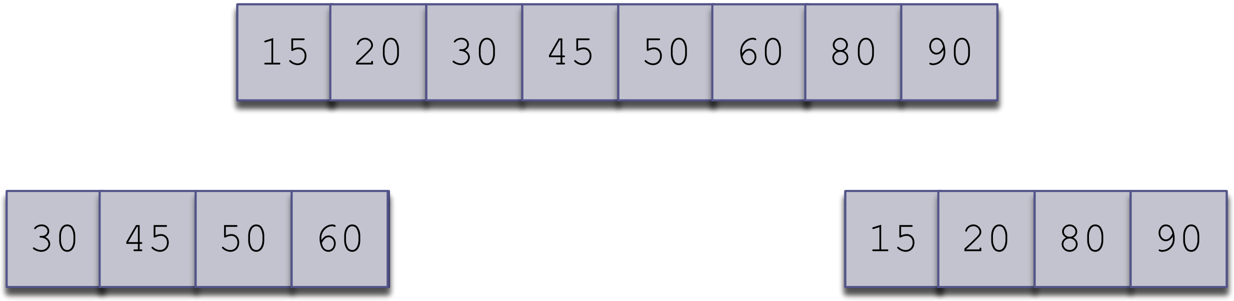
30 45 50 60

15 20 80 90

20 90

15 80

Trace of Merge Sort (cont.)



15	20	30	45	50	60	80	90

30	45	50	60

15	20	80	90

Analysis of Merge Sort

- Each backward step requires a movement of *n* elements from smaller-size arrays to larger arrays; the effort is O(*n*)
- The number of steps that require merging is log *n* because each recursive call splits the array in half
- The total effort to reconstruct the sorted array through merging is O(*n* log *n*)

Analysis of Merge Sort (cont.)

- Going down through the recursion chain, sorting the left tables, a sequence of right tables of size

  $$\frac{n}{2}, \frac{n}{4}, \ldots, \frac{n}{2^k}$$

  is allocated
- Since

  $$\frac{n}{2} + \frac{n}{4} + \ldots + 2 + 1 = n - 1$$

  a total of *n* additional storage locations are required

Code for Merge Sort

```
public class MergeSort {
    public <T extends Comparable<T>>  void sort(T[] table) {
        // A table with one element is sorted already.
        if (table.length > 1) {
            // Split table into halves.
            int halfSize = table.length / 2;
            T[] leftTable = (T[]) new Comparable[halfSize];
            T[] rightTable =
                    (T[]) new Comparable[table.length - halfSize];
            System.arraycopy(table, 0, leftTable, 0, halfSize);
            System.arraycopy(table, halfSize, rightTable, 0,
                    table.length - halfSize);

            // Sort the halves.
            sort(leftTable);
            sort(rightTable);
            // Merge the halves.
            merge(table, leftTable, rightTable);
        }
    }
    // Insert merge method here.
}
```