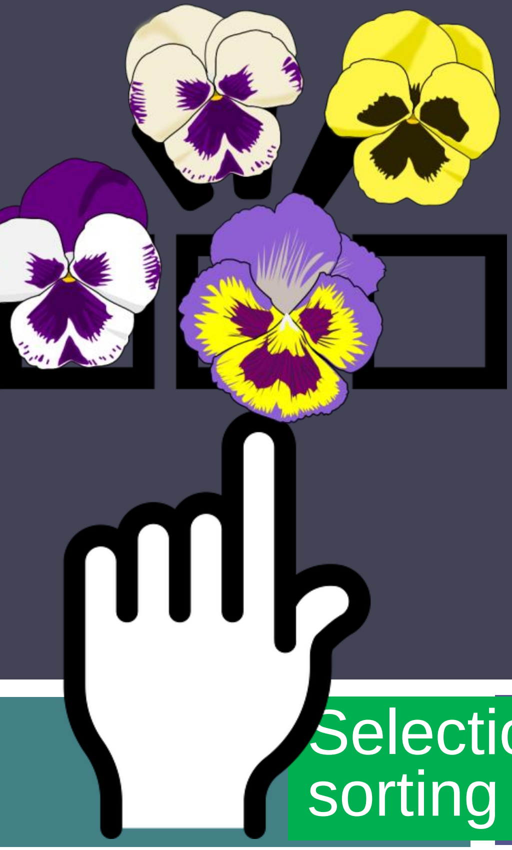


LECTURE 34

Selection Sorting; Insertion Sorting; Shell sort; Java sorting methods

Quiz (and Final) Study Guide

- Be able to execute the following algorithms
  - Counting sort
  - selection sort
  - bubble sort
  - insertion sort
  - Shell sort
  - merge sort
  - heapsort
  - quicksort
- Understand the differences in performance of these algorithms
- Read chapter 8 of the book (but don't take everything it says for granted (see slide 120)!
- **For even more fun**, watch

1. https://www.youtube.com/watch?v=OyQ6c-XHN8Y (Merge sort dance) and other dances by the same group
2. https://www.youtube.com/watch?v=ZZuD6iUe3Pc (Comparison of different sorting algorithms)

Selection Sort

Selection Sort

- Find the minimum (or maximum
- Put it in its place at the lowest (or the highest) index value
- Increase (or decrease) the index

Trace of Selection Sort

n = number of elements in the array

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

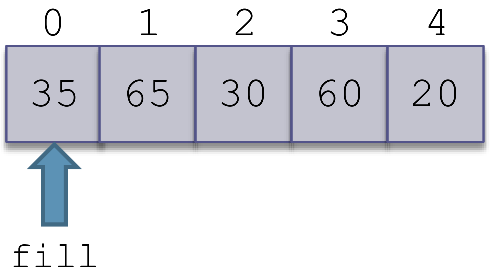
0	1	2	3	4
35	65	30	60	20

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 the smallest
        item in the subarray starting at subscript fill
3.      Exchange the item at posMin with the one at
        fill
```

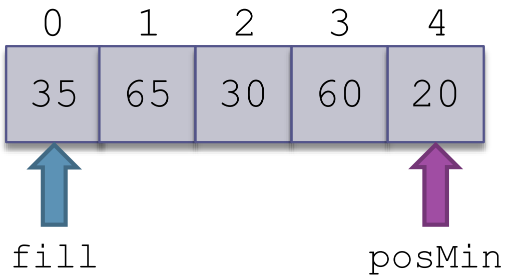


n	5
fill	0
posMin	

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 the smallest
         item in the subarray starting at subscript fill
3.       Exchange the item at posMin with the one at
         fill
```

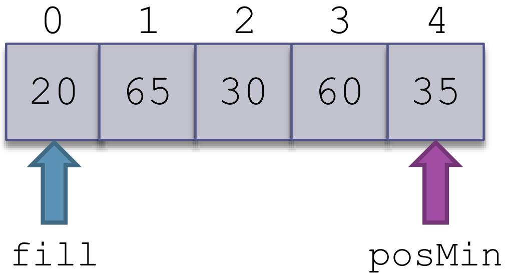


n	5
fill	0
posMin	4

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 the smallest
        item in the subarray starting at subscript fill
3.      Exchange the item at posMin with the one at
        fill
```

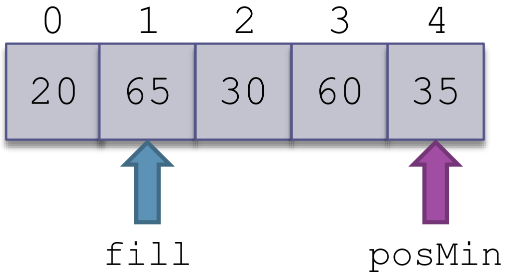


n	5
fill	0
posMin	4

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 the smallest
          item in the subarray starting at subscript fill
 3.       Exchange the item at posMin with the one at
          fill
```



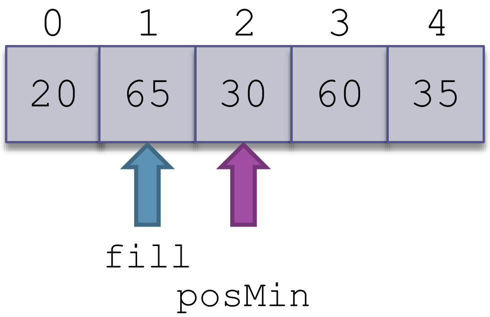
0	1	2	3	4
20	65	30	60	35

n	5
fill	1
posMin	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 the 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

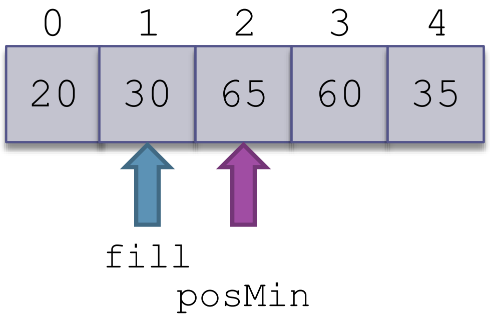
posMin

n	5
fill	1
posMin	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 the 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

fill

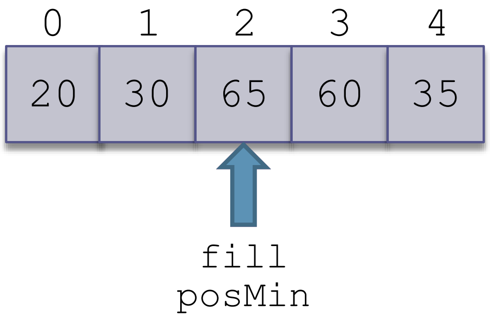
posMin

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 the smallest
         item in the subarray starting at subscript fill
3.       Exchange the item at posMin with the one at
         fill
```



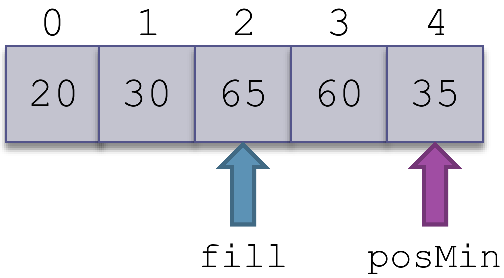
0	1	2	3	4
20	30	65	60	35

n	5
fill	2
posMin	2

Trace of Selection Sort (cont.)

```
n = number of elements in the array

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



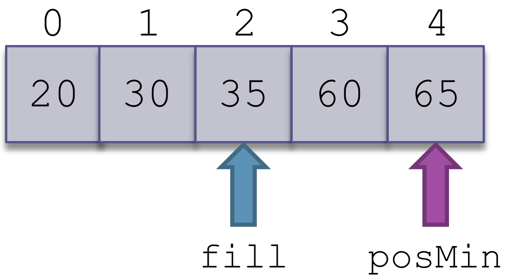
0	1	2	3	4
20	30	65	60	35

n	5
fill	2
posMin	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 the smallest
       item in the subarray starting at subscript fill
3.     Exchange the item at posMin with the one at
       fill
```



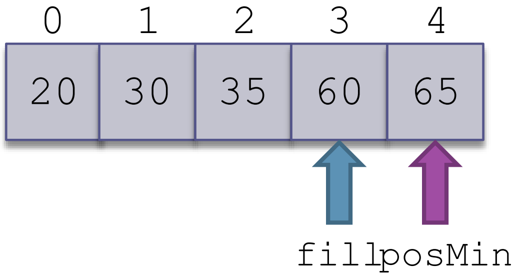
0	1	2	3	4
20	30	35	60	65

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 the smallest
         item in the subarray starting at subscript fill
3.       Exchange the item at posMin with the one at
         fill
```

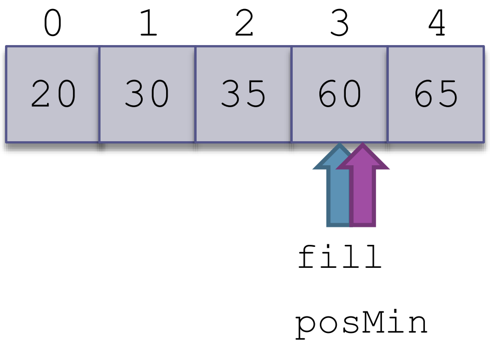


n	5
fill	3
posMin	4

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 the 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

fill

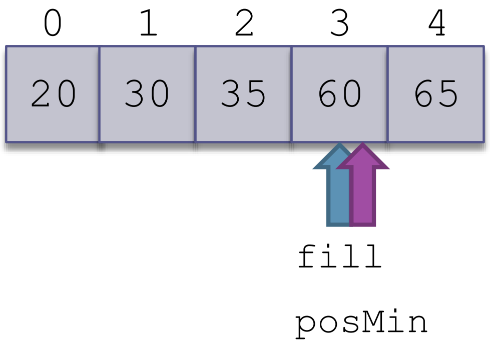
posMin

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 the 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 the 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

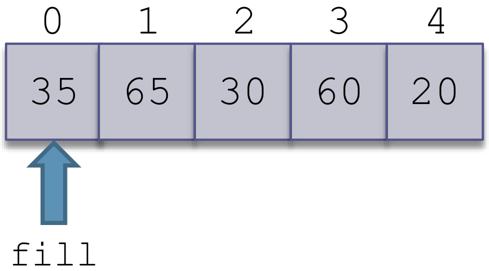
n	5
fill	
posMin	
next	

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	

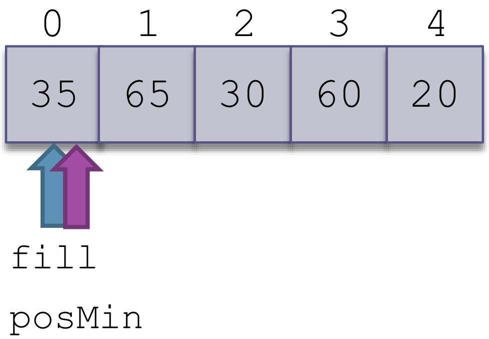


0	1	2	3	4
35	65	30	60	20

```
▶ 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	



0	1	2	3	4
35	65	30	60	20

fill

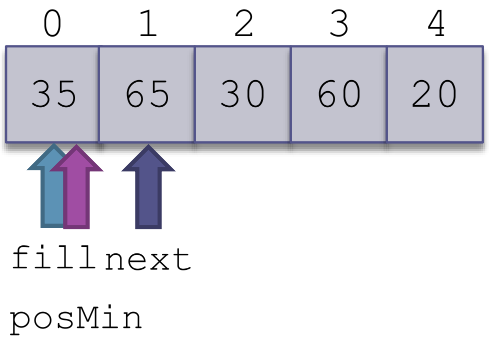
posMin

```
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

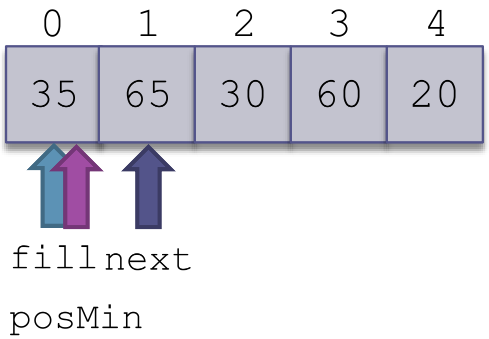
```
1. for fill = 0 to n - 2 do
2.     Initialize posMin to fill
3.     for next = fill + 1 to n - 1 do
4.         if the item at next is less than the
           item at posMin
5.             Reset posMin to next
6.     Exchange the item at posMin with the one
       at fill
```



0	1	2	3	4
35	65	30	60	20

Trace of Selection Sort Refinement (cont.)

n	5
fill	0
posMin	0
next	1

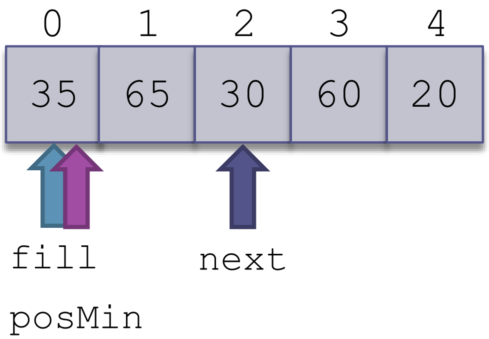


0	1	2	3	4
35	65	30	60	20

```
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

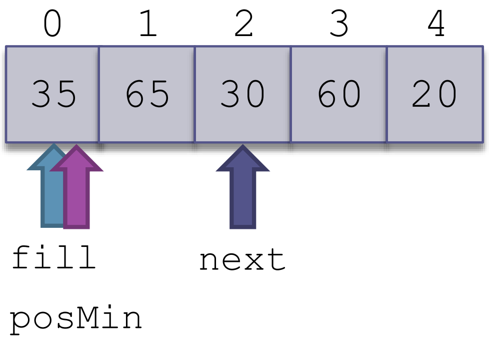


0	1	2	3	4
35	65	30	60	20

```
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

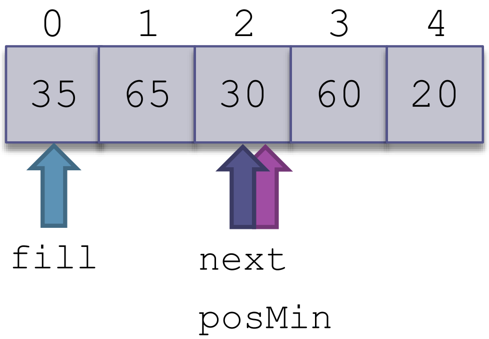


0	1	2	3	4
35	65	30	60	20

```
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	2

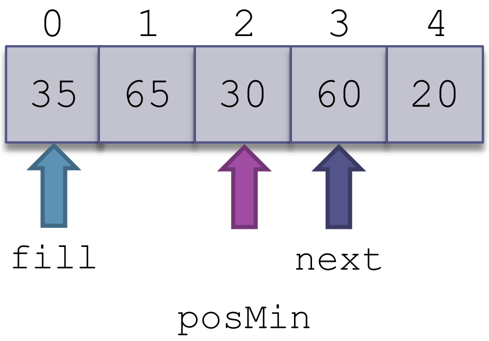


0	1	2	3	4
35	65	30	60	20

```
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

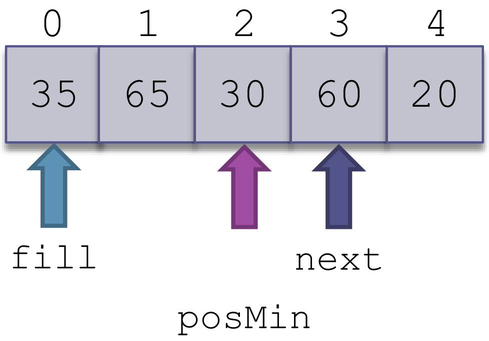


0	1	2	3	4
35	65	30	60	20

```
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.)

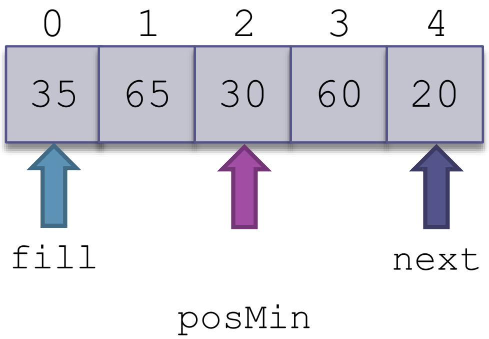
n	5
fill	0
posMin	2
next	3



```
  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.)

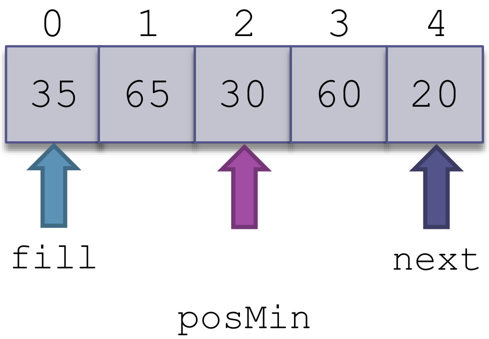
n	5
fill	0
posMin	2
next	4



```
  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	0
posMin	2
next	4

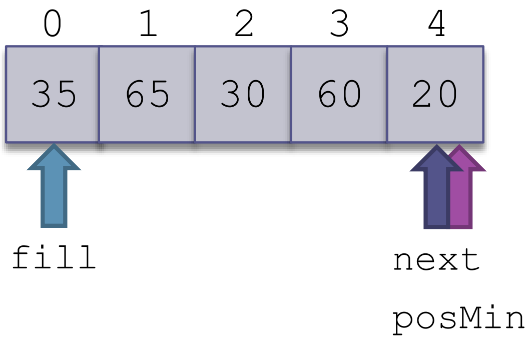


0	1	2	3	4
35	65	30	60	20

```
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.)

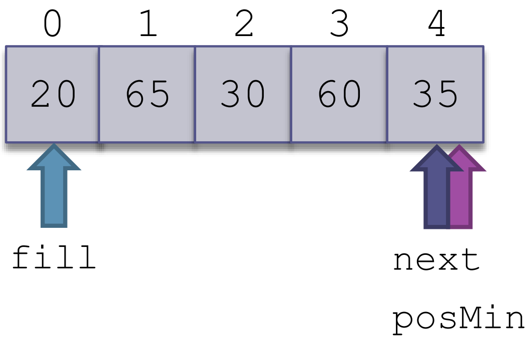
n	5
fill	0
posMin	4
next	4



```
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	0
posMin	4
next	4

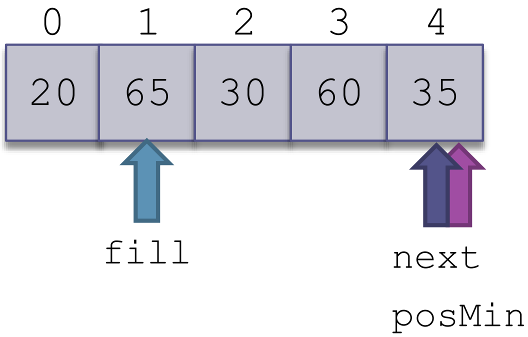


0	1	2	3	4
20	65	30	60	35

```
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

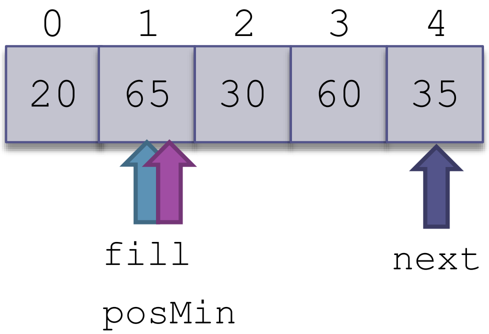


0	1	2	3	4
20	65	30	60	35

```
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.)

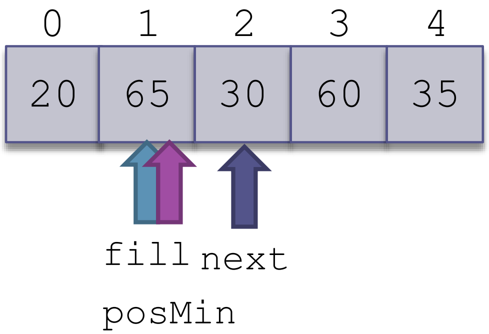
n	5
fill	1
posMin	1
next	4



```
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

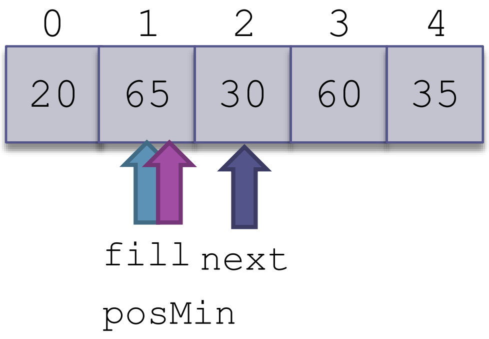


0	1	2	3	4
20	65	30	60	35

```
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	1
next	2

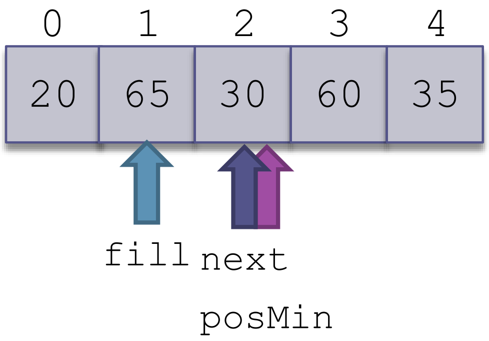


0	1	2	3	4
20	65	30	60	35

```
  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	2
next	2

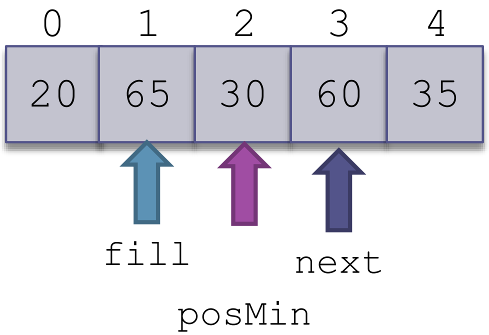


0	1	2	3	4
20	65	30	60	35

```
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.)

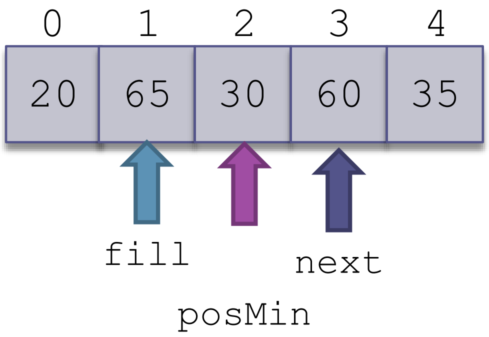
n	5
fill	1
posMin	2
next	3



```
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.)

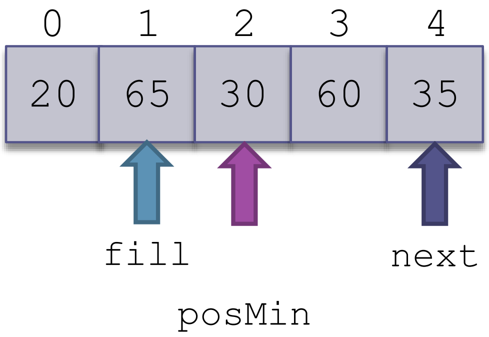
n	5
fill	1
posMin	2
next	3



```
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

fill

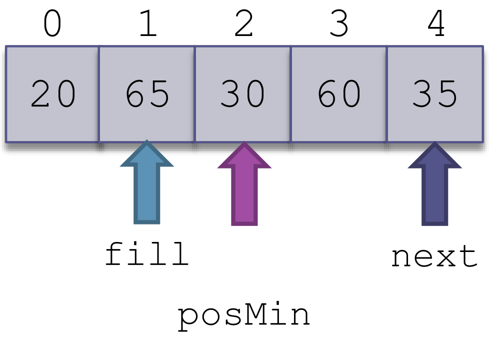
next

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.)

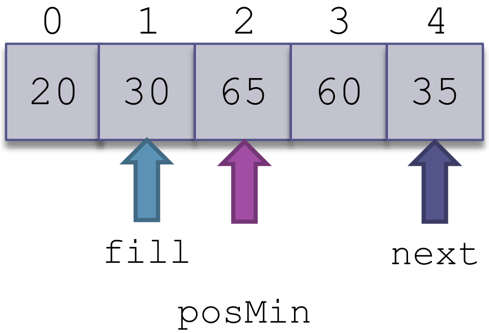
n	5
fill	1
posMin	2
next	4



```
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.)

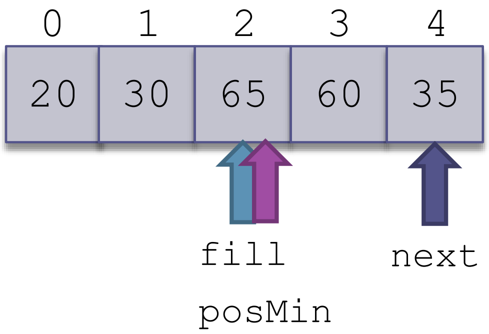
n	5
fill	1
posMin	2
next	4



```
▻ 
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.)

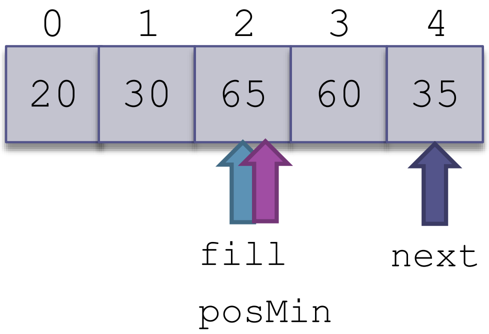
n	5
fill	2
posMin	2
next	4



```
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.)

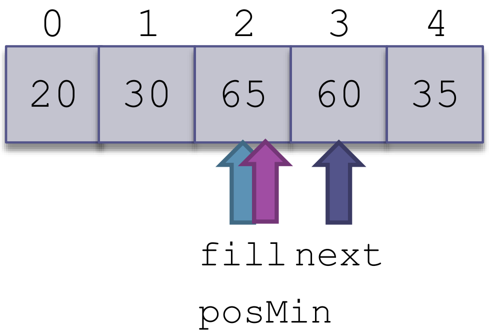
n	5
fill	2
posMin	2
next	4



```
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

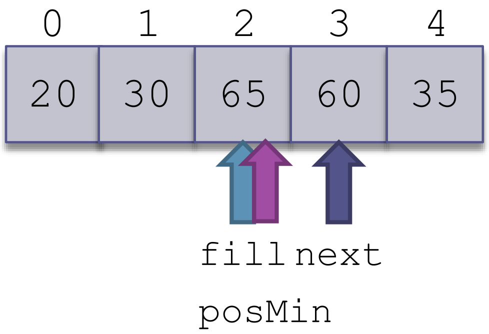


0	1	2	3	4
20	30	65	60	35

```
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

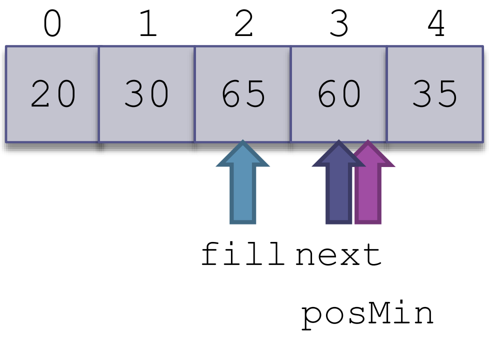


0	1	2	3	4
20	30	65	60	35

```
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	3

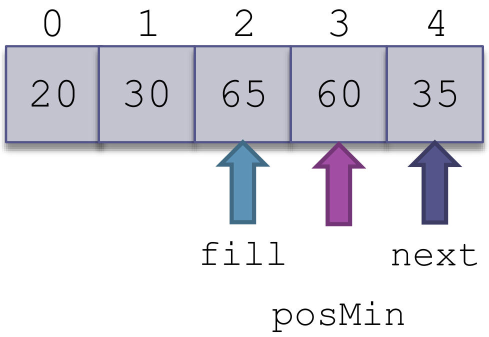


0	1	2	3	4
20	30	65	60	35

```
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

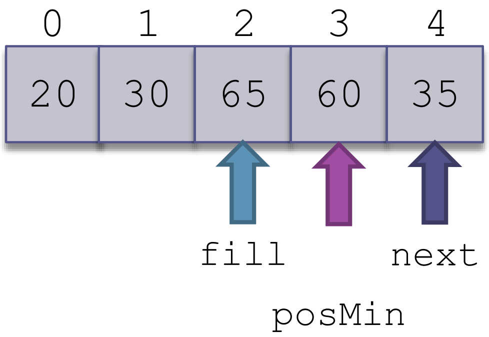


0	1	2	3	4
20	30	65	60	35

```
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	3
next	4

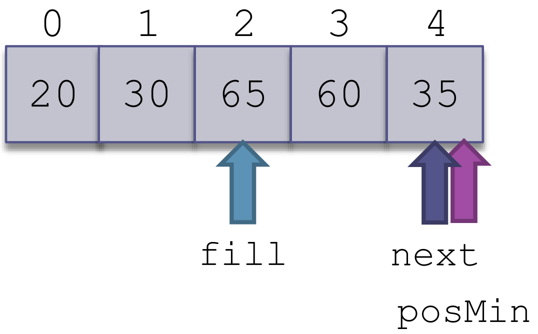


0	1	2	3	4
20	30	65	60	35

```
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	4
next	4

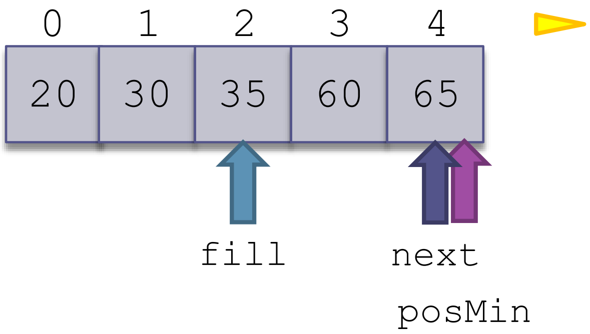


0	1	2	3	4
20	30	65	60	35

```
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.)

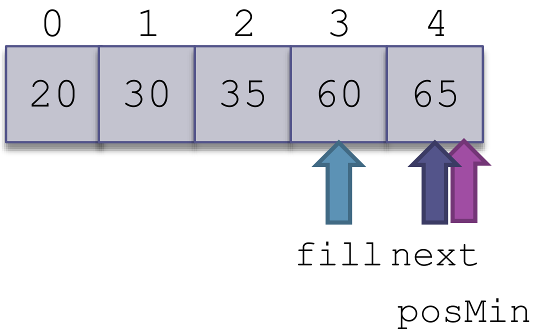
n	5
fill	2
posMin	4
next	4



```
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	3
posMin	4
next	4

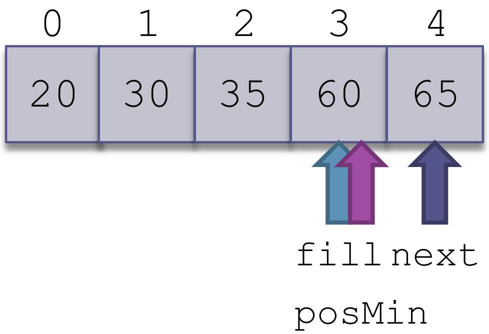


0	1	2	3	4
20	30	35	60	65

```
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

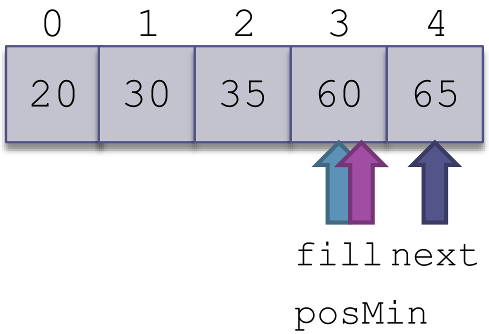


0	1	2	3	4
20	30	35	60	65

```
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

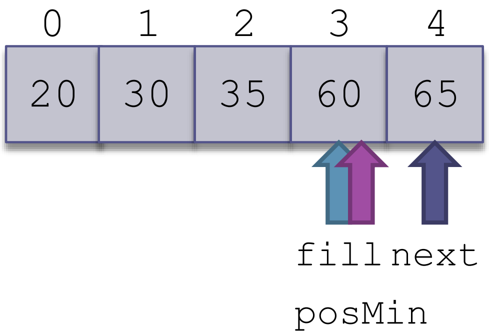


0	1	2	3	4
20	30	35	60	65

```
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.)

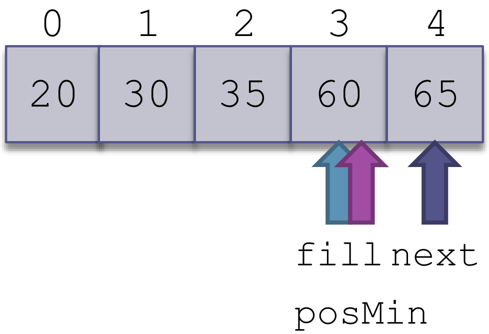
n	5
fill	3
posMin	3
next	4



```
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



```
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.)

This comparison is performed (*n* – 1 - *fill*) times for each value of *fill* = *0, 1, … (n-2)*, resulting in (n-1) + (n-2) + … 1 = = *n(n-1)/2 comparisons*

```
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 (cont.)

```
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
```

Code for Selection Sort (cont.)

- Listing 8.1(`SelectionSort.java,` pages 426 - 427)

Insertion Sort

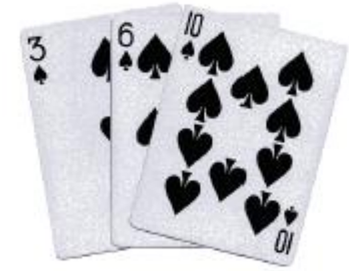
The simile

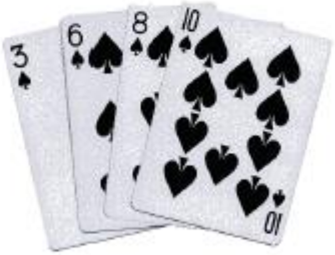
- Arranging a hand of cards

1) 8 is dealt

The simile

□ Arranging a hand of cards

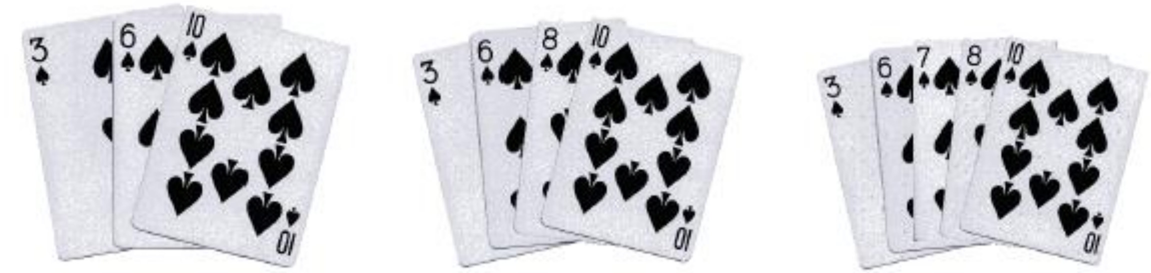




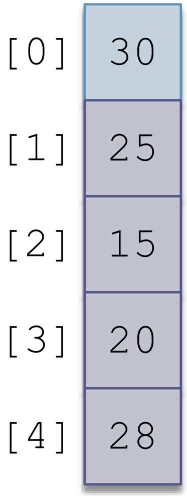
2) 7 is dealt

The simile

- Arranging a hand of cards



Trace of Insertion Sort



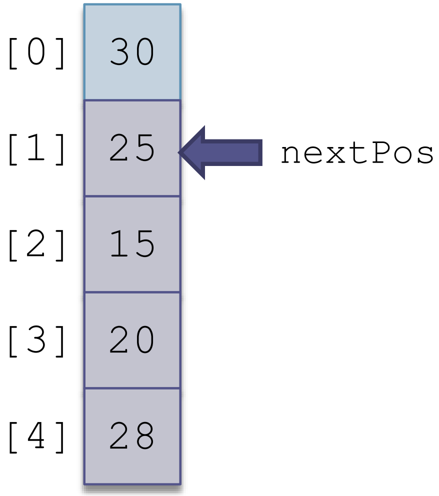
[0]	30
[1]	25
[2]	15
[3]	20
[4]	28

start with a sorted subarray consisting of only the first element

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.)

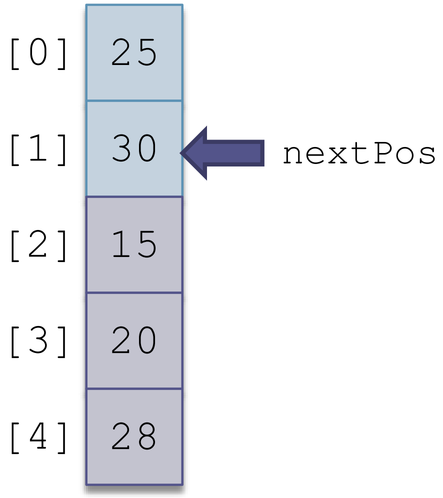
nextPos	1



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	1

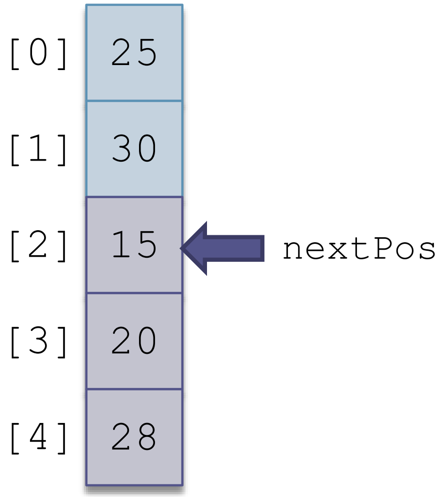


[0]	25
[1]	30
[2]	15
[3]	20
[4]	28

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.)

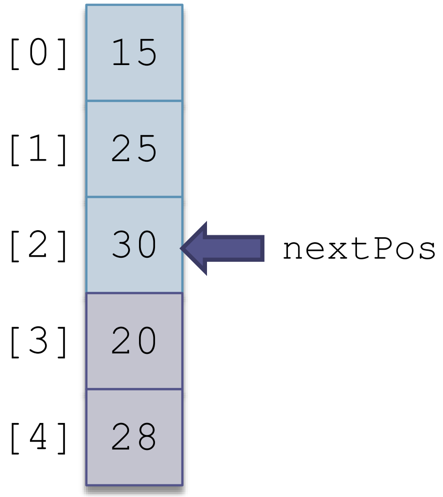
`nextPos`	2



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



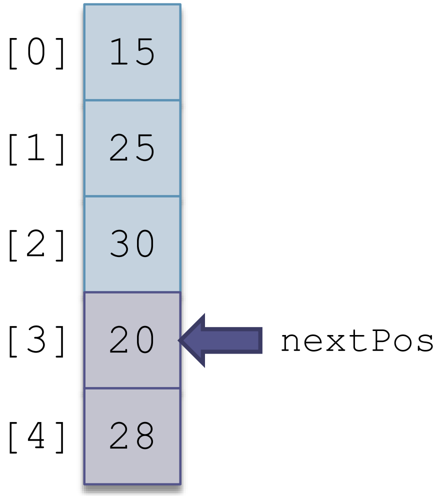
[0]	15
[1]	25
[2]	30
[3]	20
[4]	28

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	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



[0] 15

[1] 25

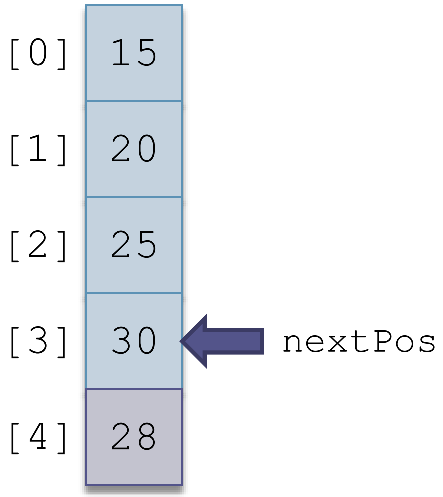
[2] 30

[3] 20

[4] 28

Trace of Insertion Sort (cont.)

nextPos	3



[0] 15

[1] 20

[2] 25

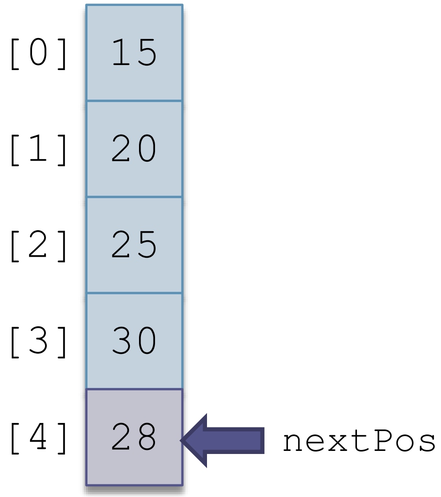
[3] 30 ← nextPos

[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

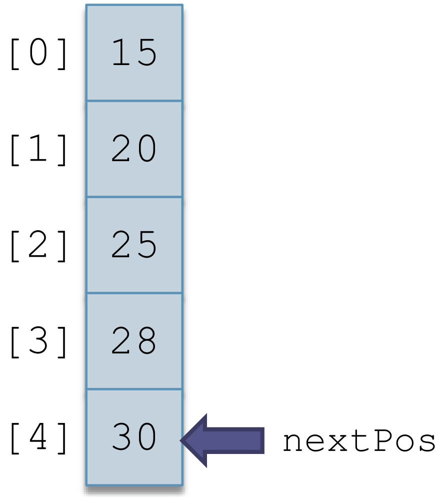


[0]	15
[1]	20
[2]	25
[3]	30
[4]	28

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



[0] 15

[1] 20

[2] 25

[3] 28

[4] 30 nextPos

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

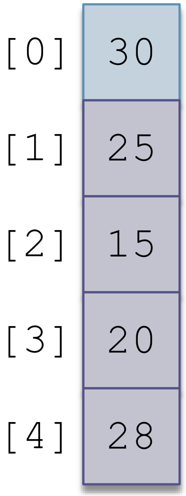
Trace of Insertion Sort (cont.)

nextPos	-

[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

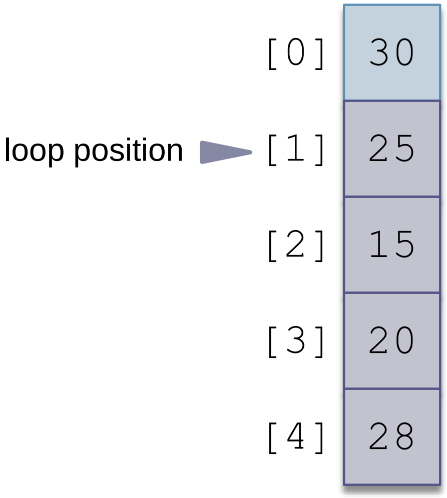


[0]	30
[1]	25
[2]	15
[3]	20
[4]	28

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

Trace of Insertion Sort Refinement (cont.)

nextPos	1
nextVal	

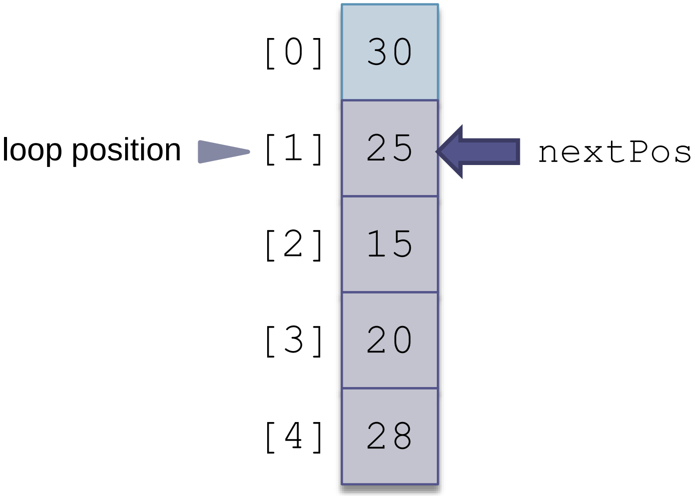


	[0]	30
loop position	[1]	25
	[2]	15
	[3]	20
	[4]	28

```
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

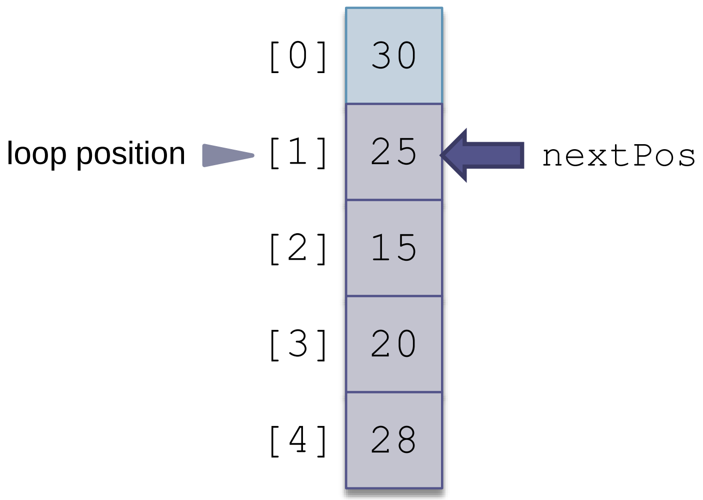
[0]	30
[1]	25
[2]	15
[3]	20
[4]	28

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.)

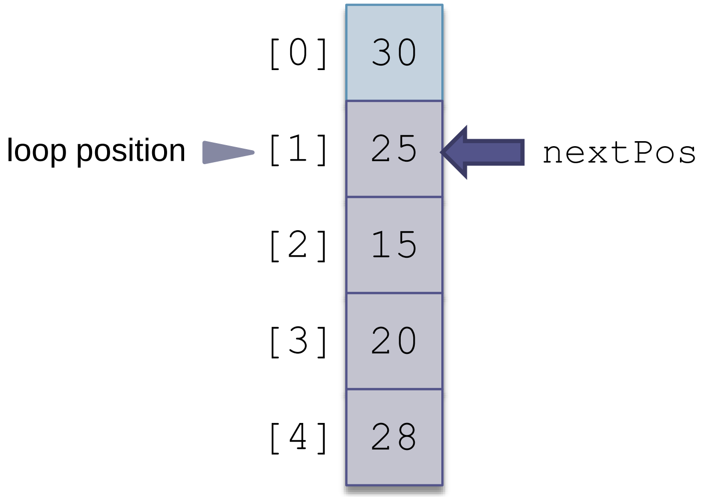
nextPos	1
nextVal	25



```
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.)

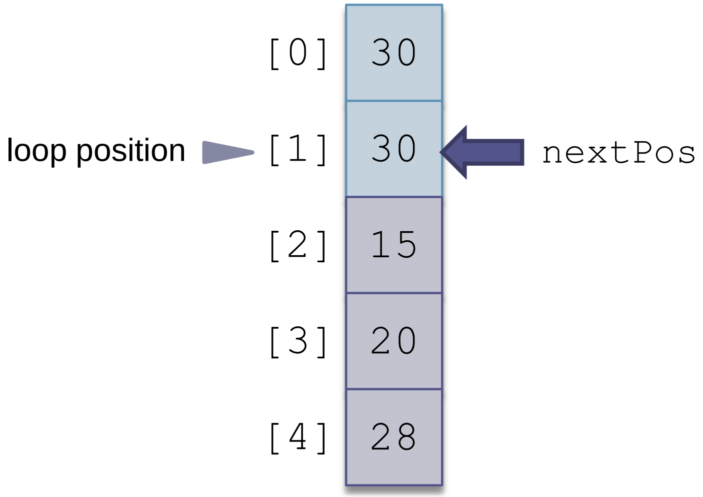
nextPos	1
nextVal	25



```
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.)

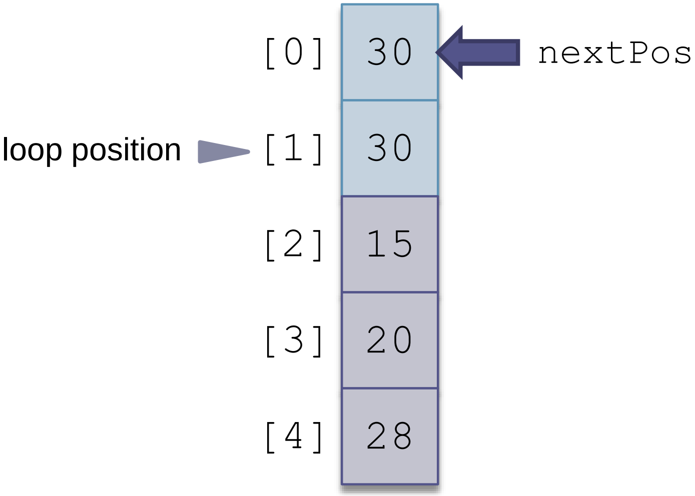
nextPos	1
nextVal	25



```
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.)

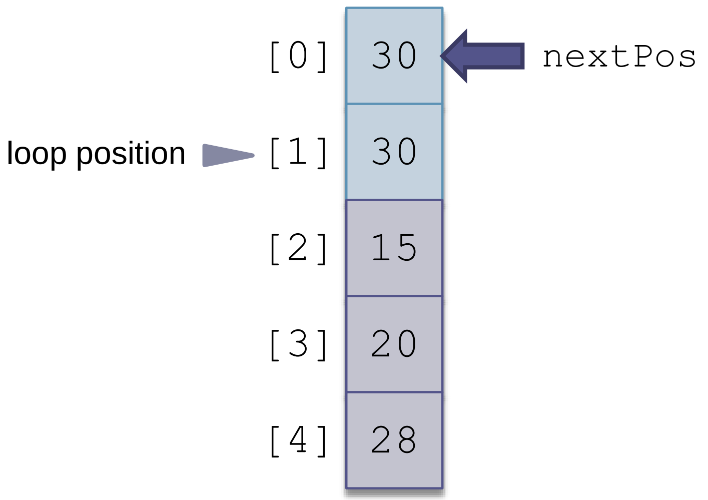
nextPos	0
nextVal	25



```
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



[0] 30 ← nextPos

loop position → [1] 30

[2] 15

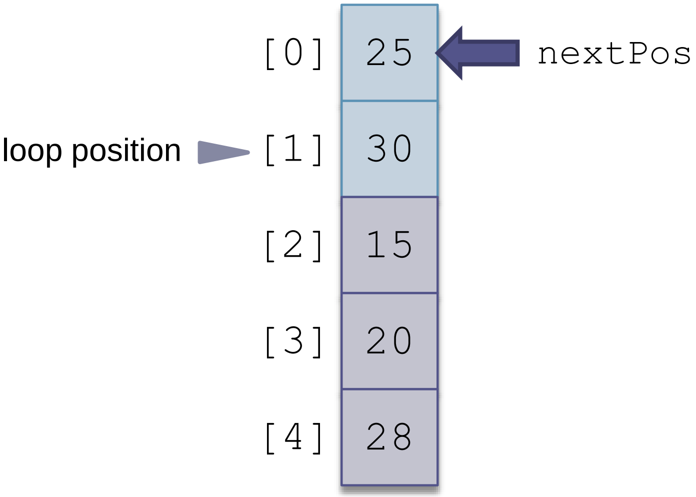
[3] 20

[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.)

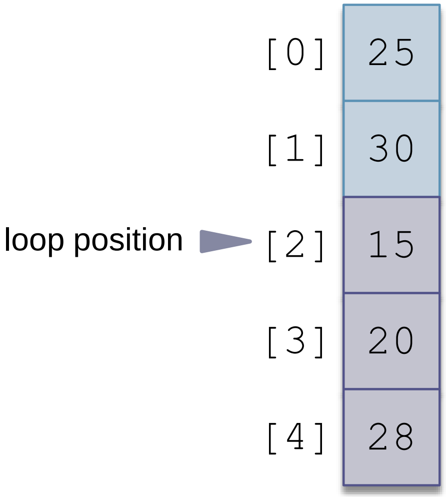
nextPos	0
nextVal	25



```
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.)

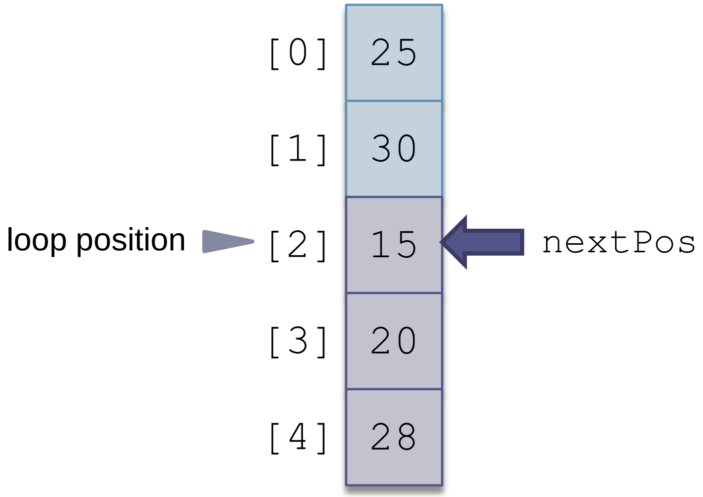
nextPos	0
nextVal	25



```
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.)

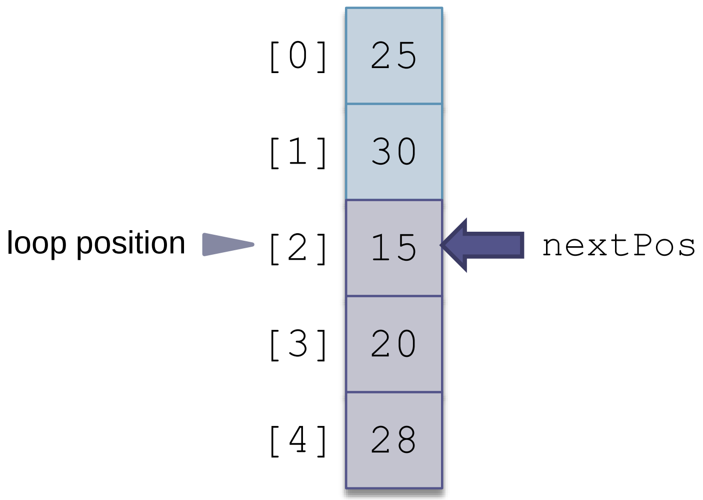
nextPos	2
nextVal	25



```
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.)

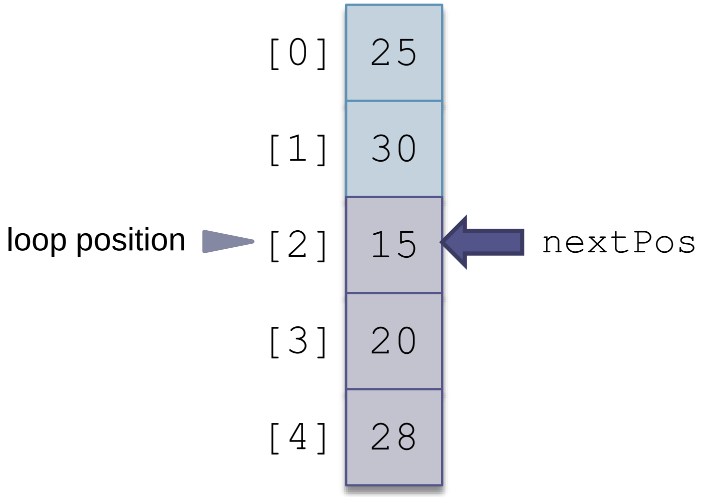
nextPos	2
nextVal	15



```
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.)

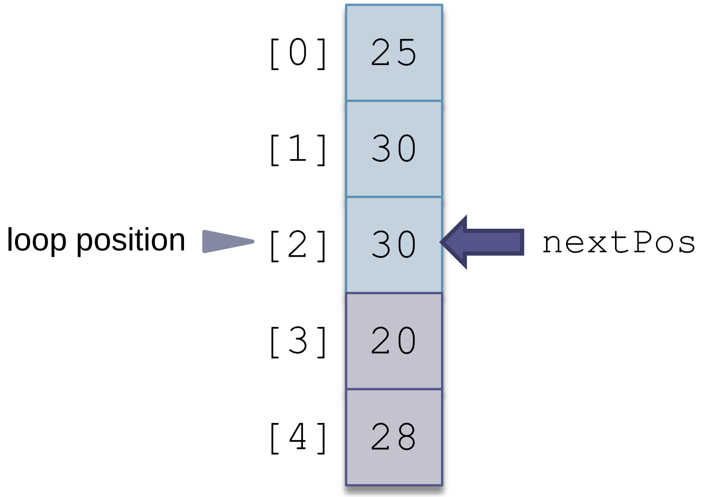
nextPos	2
nextVal	15



```
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.)

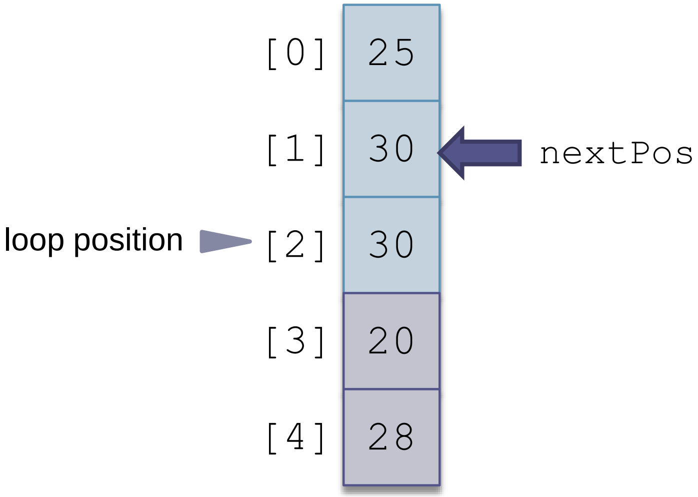
nextPos	2
nextVal	15



```
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.)

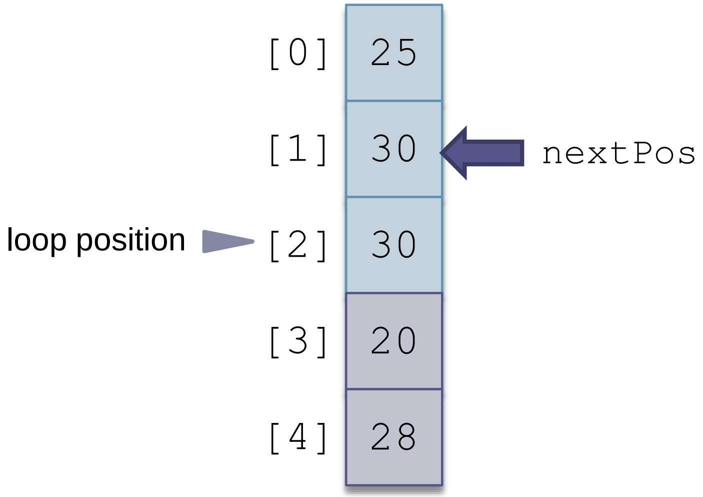
`nextPos`	1
`nextVal`	15



```
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.)

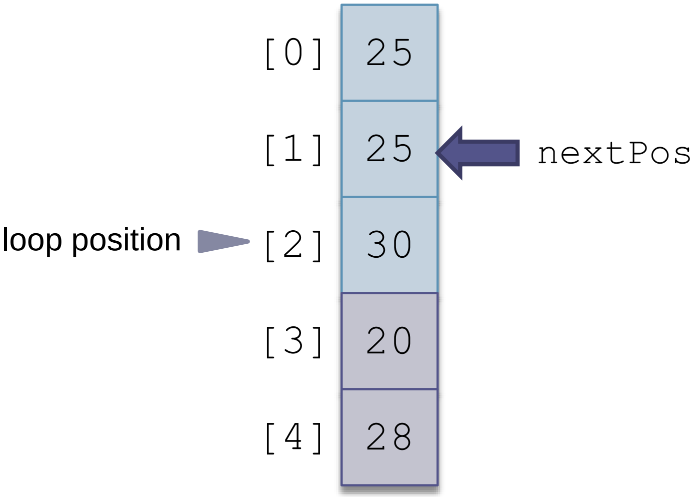
nextPos	1
nextVal	15



```
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.)

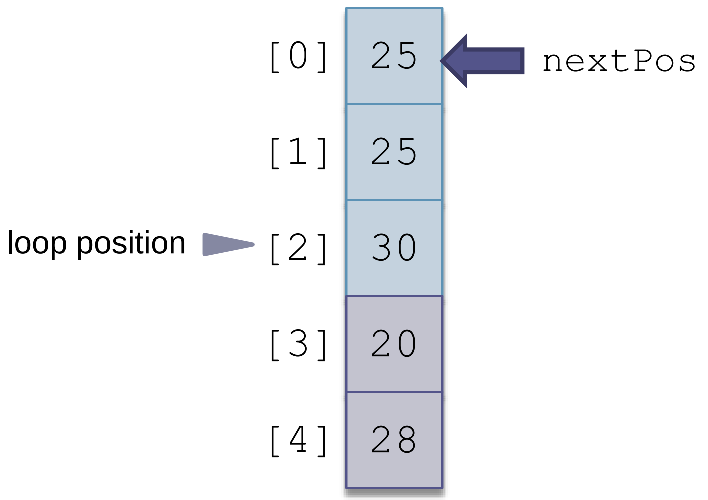
nextPos	1
nextVal	15



```
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.)

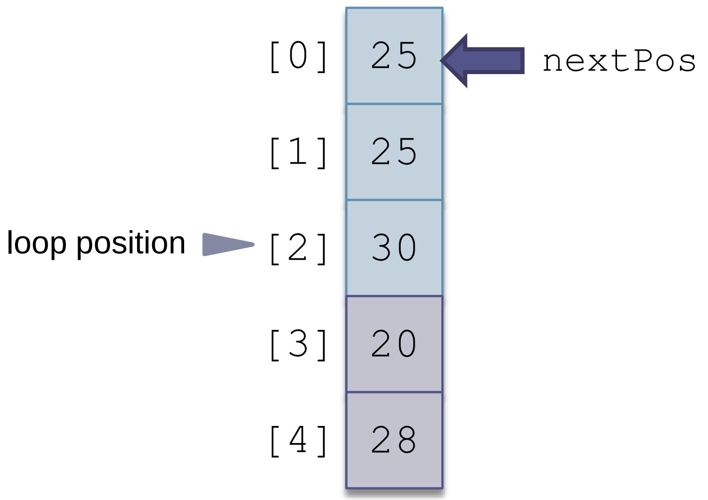
nextPos	0
nextVal	15



```
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.)

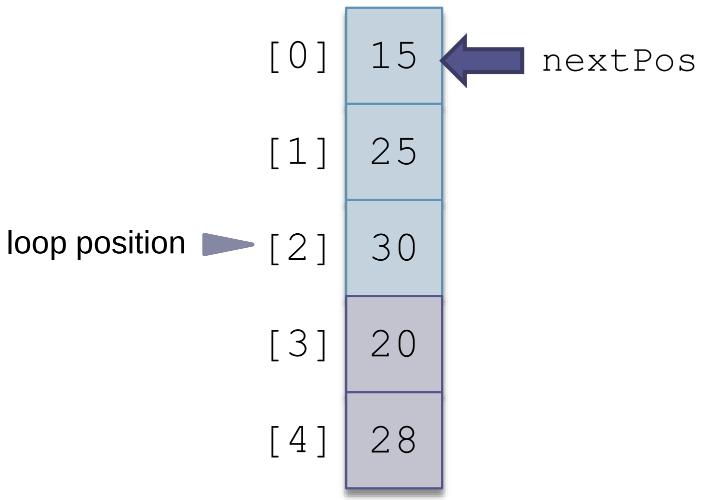
nextPos	0
nextVal	15



```
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.)

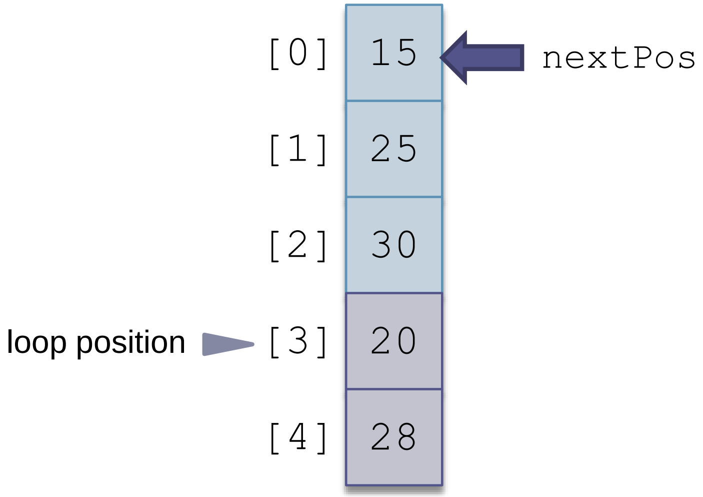
nextPos	0
nextVal	15



```
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.)

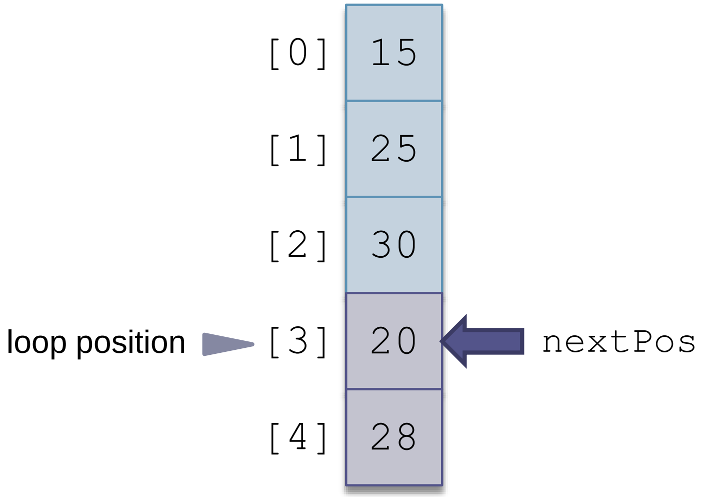
nextPos	0
nextVal	15



```
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



loop position

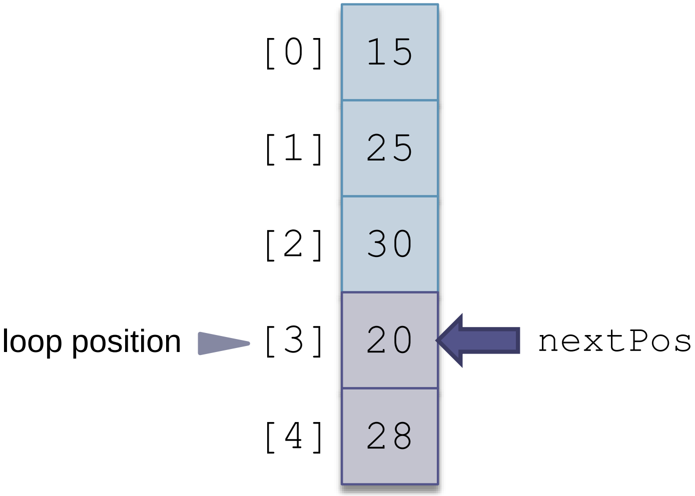
[0]	15
[1]	25
[2]	30
[3]	20
[4]	28

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.)

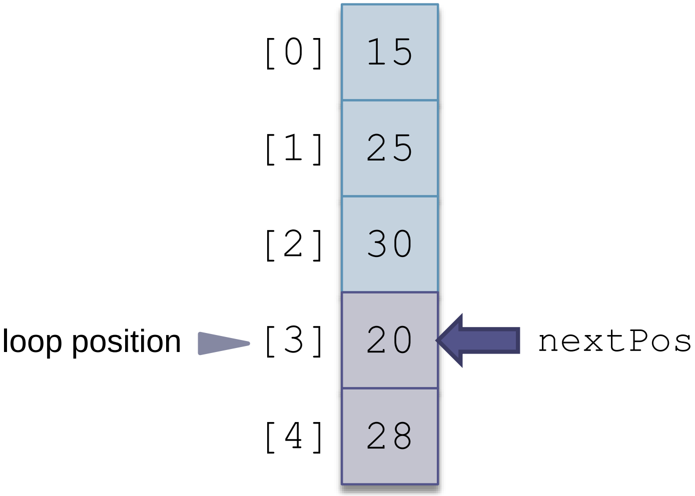
nextPos	3
nextVal	20



```
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.)

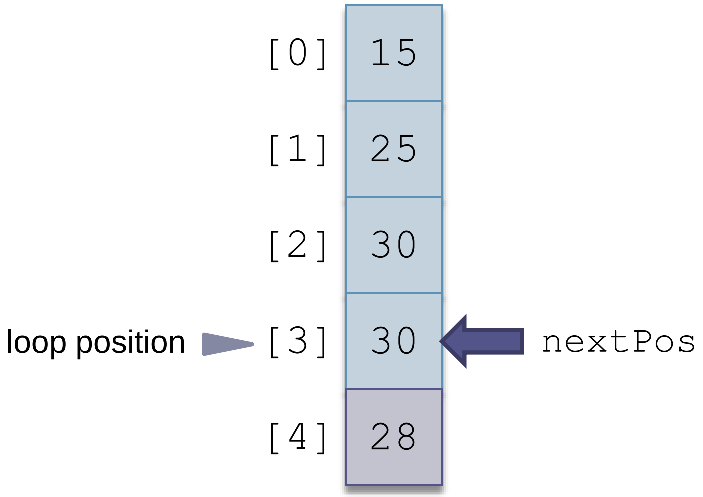
nextPos	3
nextVal	20



```
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.)

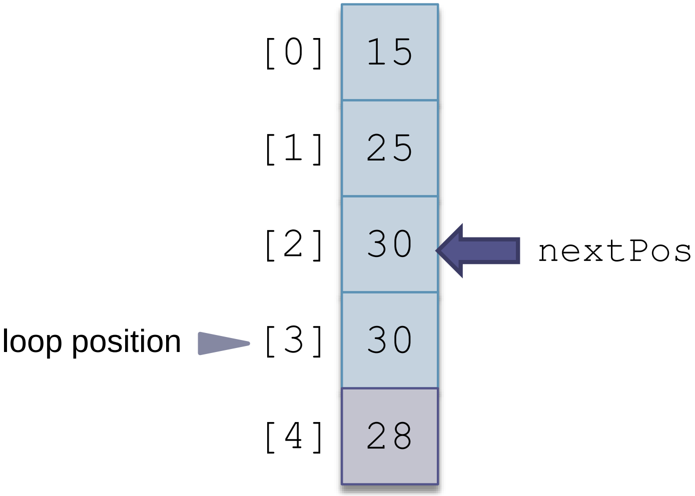
nextPos	3
nextVal	20



```
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.)

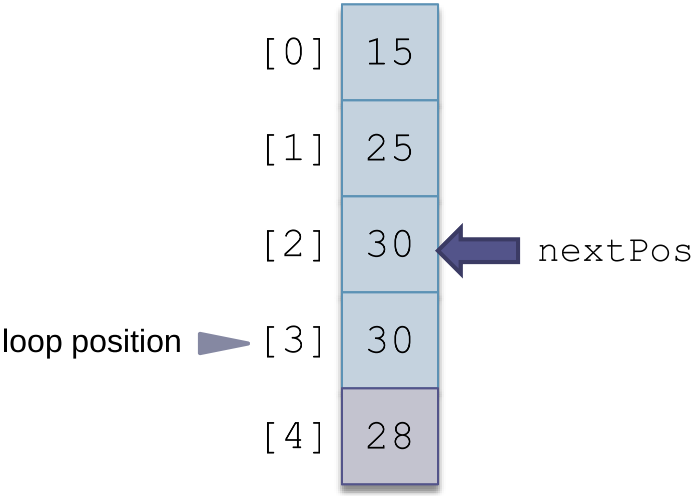
nextPos	2
nextVal	20



```
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.)

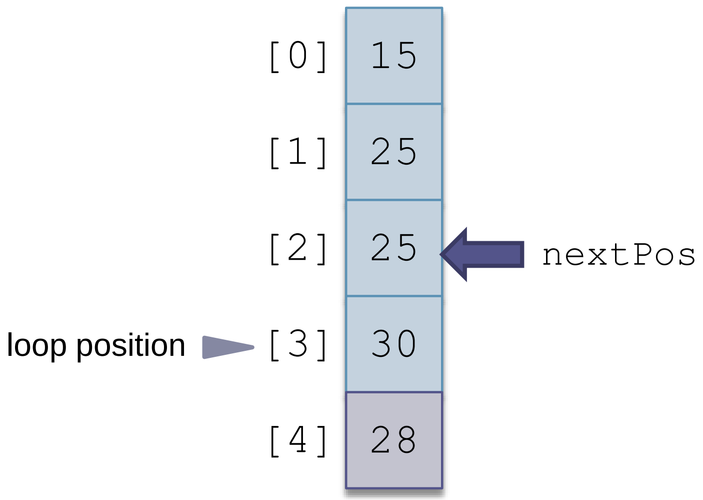
nextPos	2
nextVal	20



```
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.)

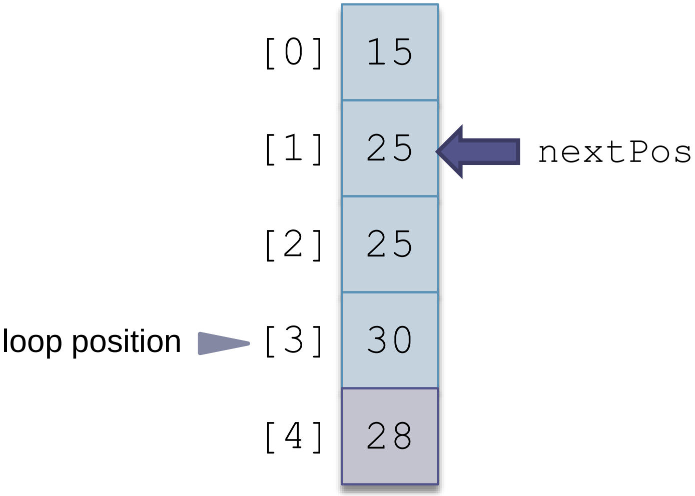
nextPos	2
nextVal	20



```
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.)

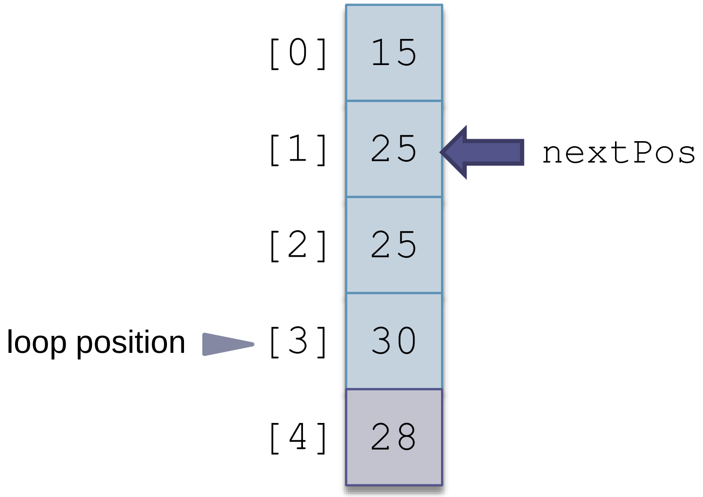
nextPos	1
nextVal	20



```
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] 25 nextPos

[2] 25

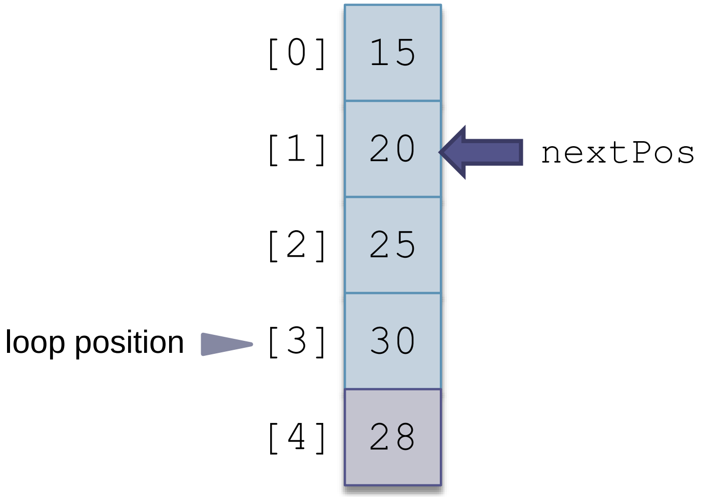
loop position [3] 30

[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.)

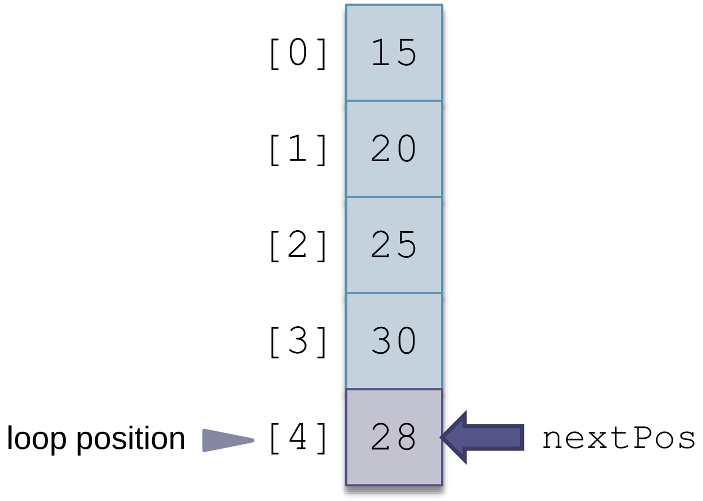
nextPos	1
nextVal	20

	[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	1
nextVal	20



loop position

[0] 15

[1] 20

[2] 25

[3] 30

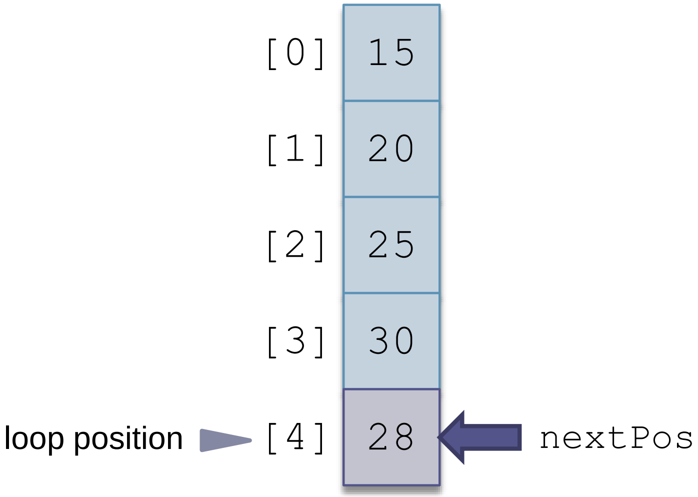
[4] 28

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



[0] 15

[1] 20

[2] 25

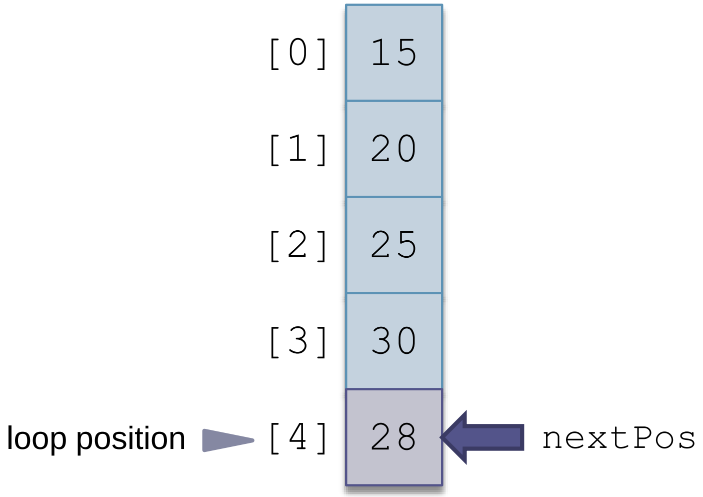
[3] 30

loop position [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.)

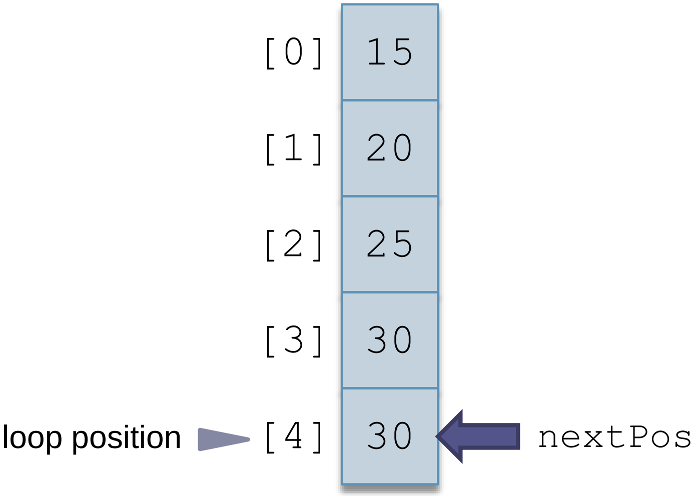
nextPos	4
nextVal	28



```
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.)

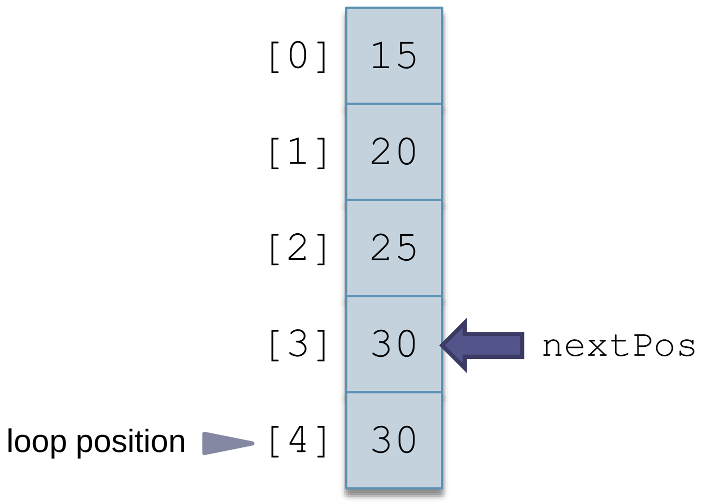
nextPos	4
nextVal	28



```
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

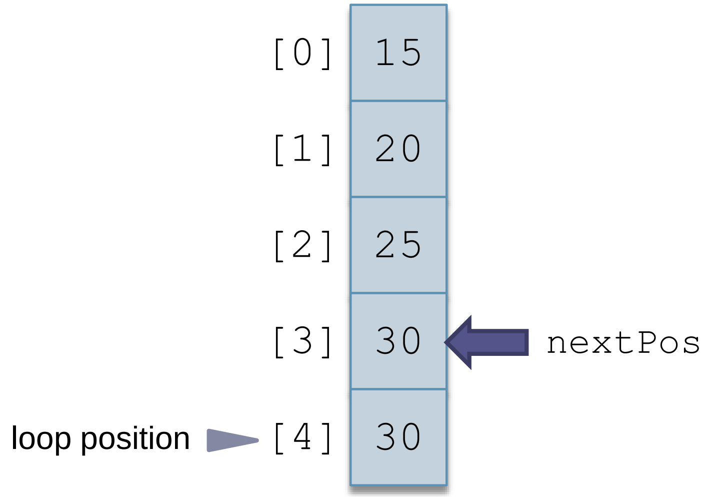
[3] 30

[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.)

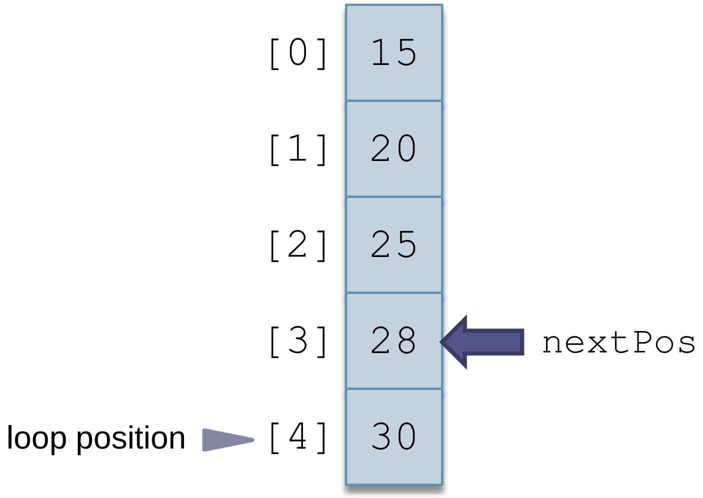
nextPos	3
nextVal	28



```
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 will then be:

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

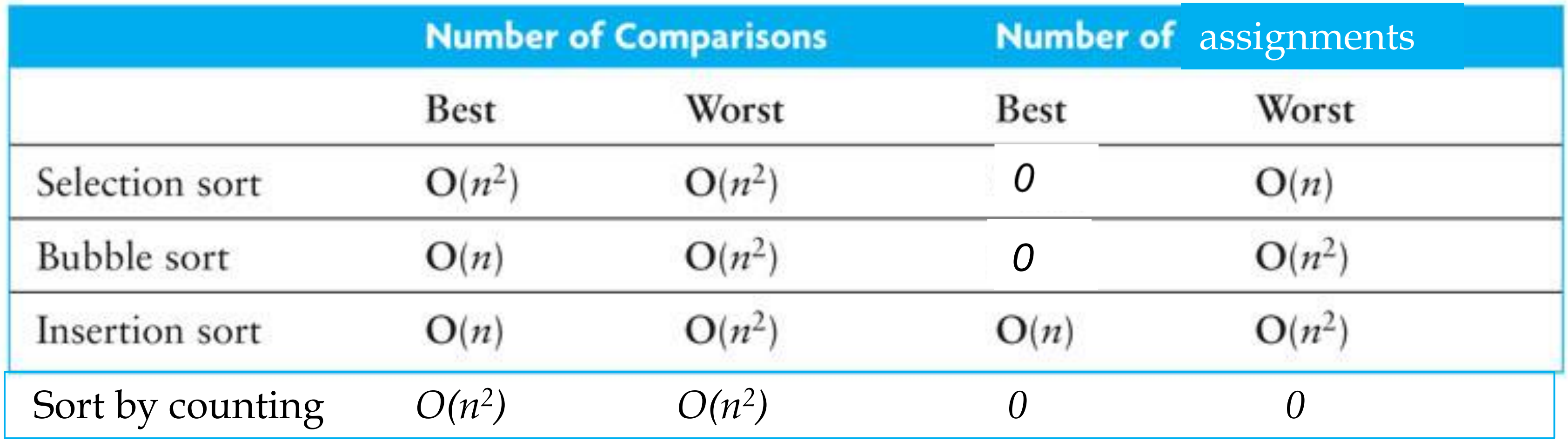
- And… we may do another $O(n^2)$ of assignments because of shifting

Code for Insertion Sort

- Listing 8.3 (`InsertionSort.java`, page 434)

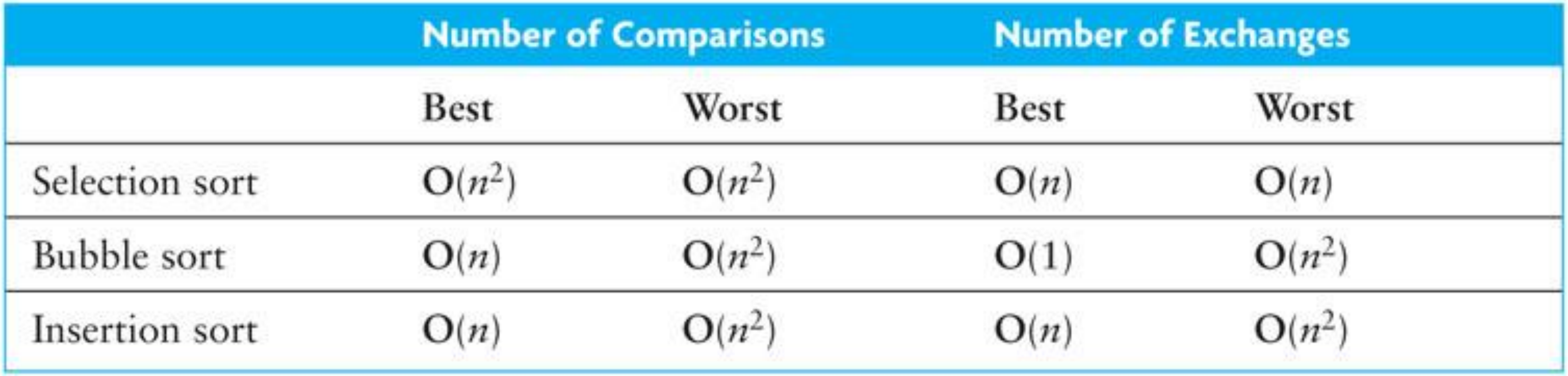
Comparison of Quadratic Sorts

Comparison of Quadratic Sorts (and a close look at the book…)



	Number of Comparisons		Number of assignments	
	Best	Worst	Best	Worst
Selection sort	$O(n^2)$	$O(n^2)$	0	$O(n)$
Bubble sort	$O(n)$	$O(n^2)$	0	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n)$	$O(n^2)$
Sort by counting	$O(n^2)$	$O(n^2)$	0	0

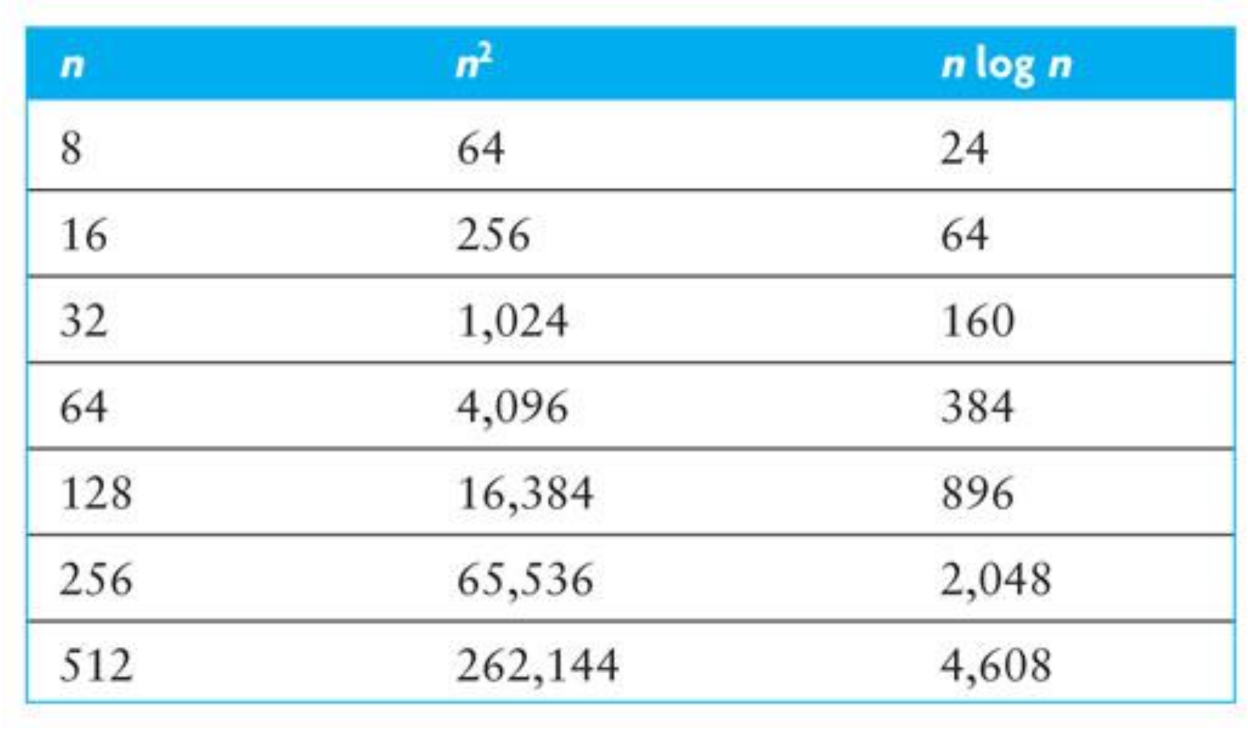
Compare with the table in the book:



	Number of Comparisons		Number of Exchanges	
	Best	Worst	Best	Worst
Selection sort	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$
Bubble sort	$O(n)$	$O(n^2)$	$O(1)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n)$	$O(n^2)$

Comparison of Quadratic Sorts (cont.)

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
- Bubble sort generally gives the worst performance—unless the array is nearly sorted
  - big-O analysis ignores constants and overhead
- None of the quadratic search algorithms are 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)

Comparisons versus Exchanges

- In Java, an exchange requires a switch of two object references using a third object reference as an intermediary
- A comparison requires an execution of a `compareTo` method
- The cost of a comparison depends on its complexity, but is generally more costly than an exchange
- For some other languages, an exchange may involve physically moving information rather than swapping object references. In these cases, an exchange may be more costly than a comparison

Shell Sort: A Better Insertion Sort

Donald L. Shell (1924-2015)

Graduated with the Ph.D. degree in Mathematics from the University of Cincinnati in 1959 and immediately published the Shell Sort algorithm "*A High-Speed Sorting Procedure"* in the *Communications of the ACM, Vol. 2, No. 7 [1959]*

Has also made major breakthroughs in mathematics.

Served as a manager in GE and, for several years, owned a robotics company.

Shell Sort: A Better Insertion Sort

- A Shell sort is a type of insertion sort, but with O($n^{3/2}$) performance
- A Shell sort can be thought of as a divide-and-conquer approach to insertion sort
- Instead of sorting the entire array, Shell sort sorts many smaller subarrays using insertion sort before sorting the entire array

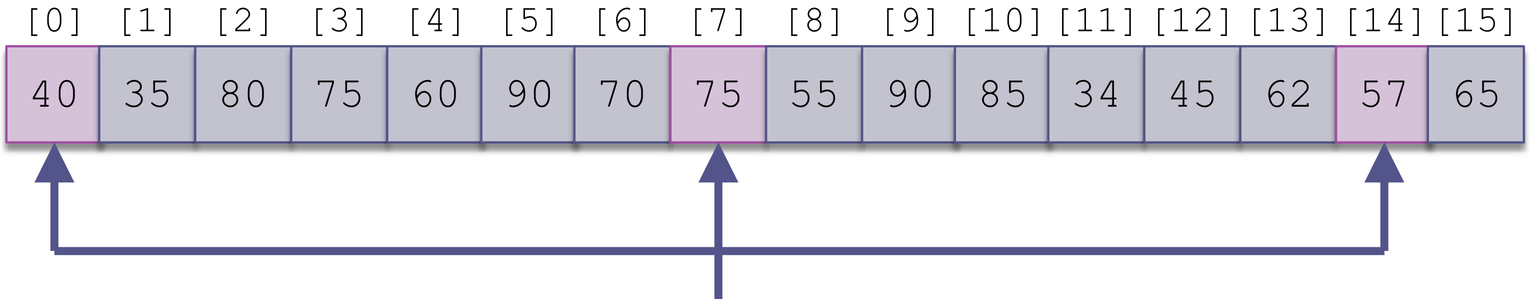
Trace of Shell Sort

gap value	7

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	75	55	90	85	34	45	62	57	65

Trace of Shell Sort (cont.)

gap value	7

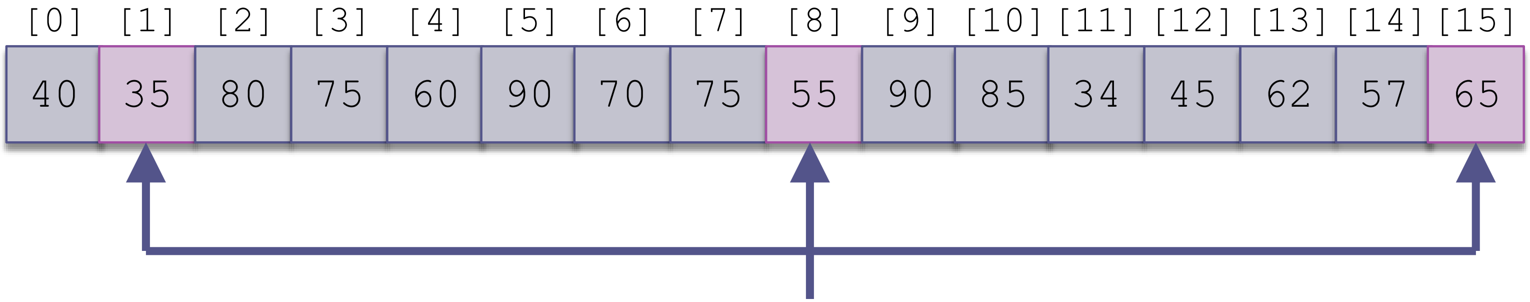


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	75	55	90	85	34	45	62	57	65

subarray 1

Trace of Shell Sort (cont.)

gap value	7

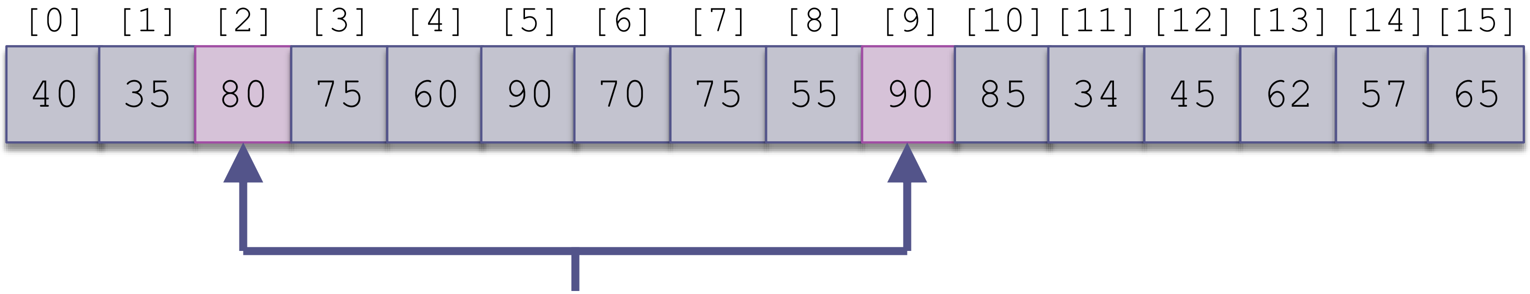


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	75	55	90	85	34	45	62	57	65

subarray 2

Trace of Shell Sort (cont.)

gap value	7

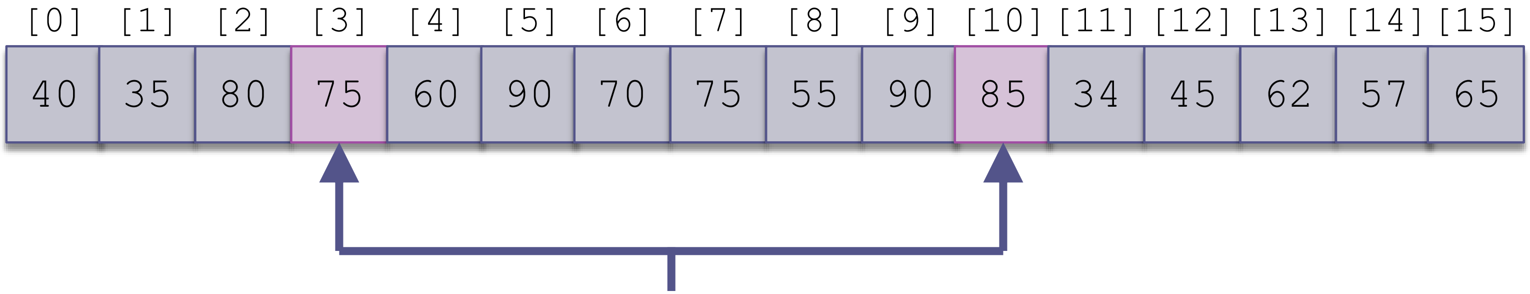


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	75	55	90	85	34	45	62	57	65

subarray 3

Trace of Shell Sort (cont.)

gap value	7

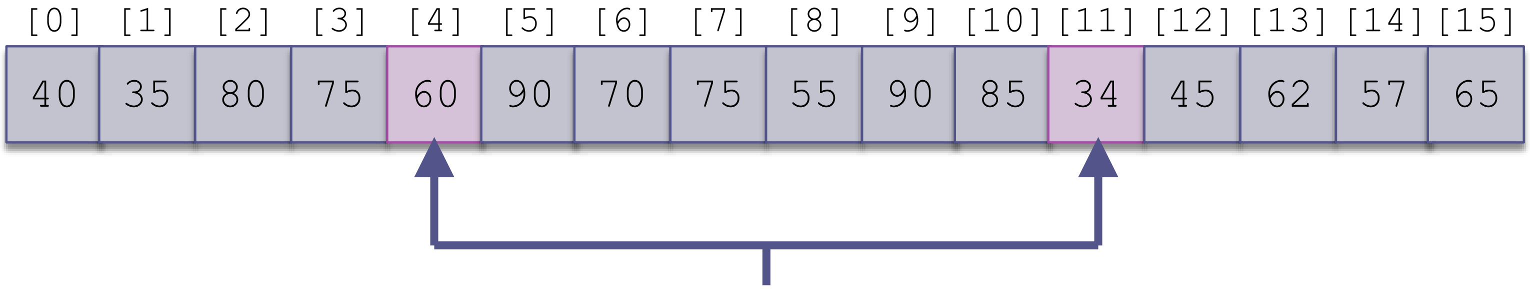


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	75	55	90	85	34	45	62	57	65

subarray 4

Trace of Shell Sort (cont.)

gap value	7

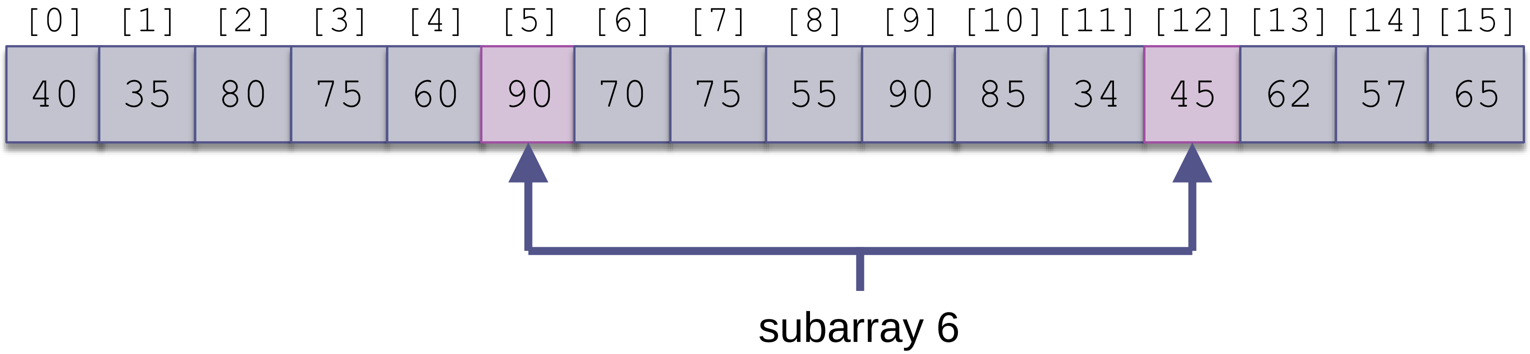


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	75	55	90	85	34	45	62	57	65

subarray 5

Trace of Shell Sort (cont.)

gap value	7

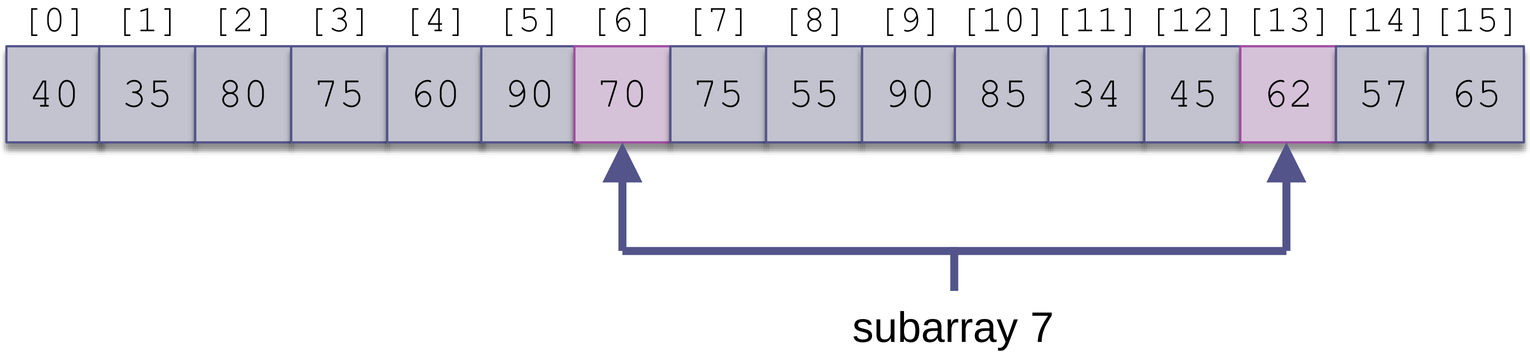


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	75	55	90	85	34	45	62	57	65

subarray 6

Trace of Shell Sort (cont.)

gap value	7



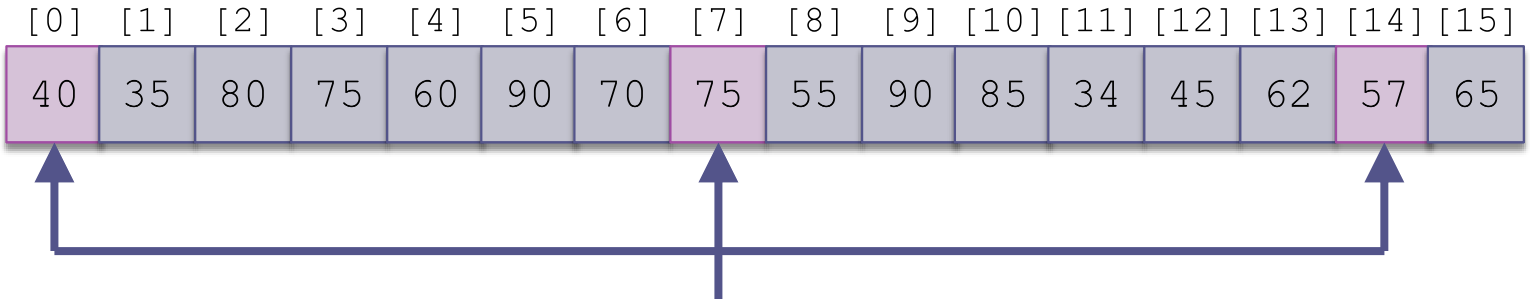
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	75	55	90	85	34	45	62	57	65

subarray 7

Trace of Shell Sort (cont.)

gap value	7

Sort subarray 1



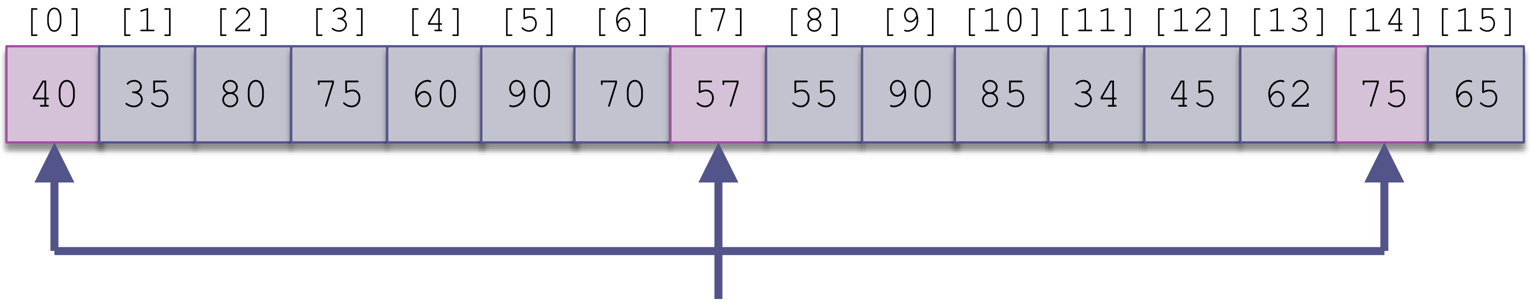
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	75	55	90	85	34	45	62	57	65

subarray 1

Trace of Shell Sort (cont.)

gap value	7

Sort subarray 1



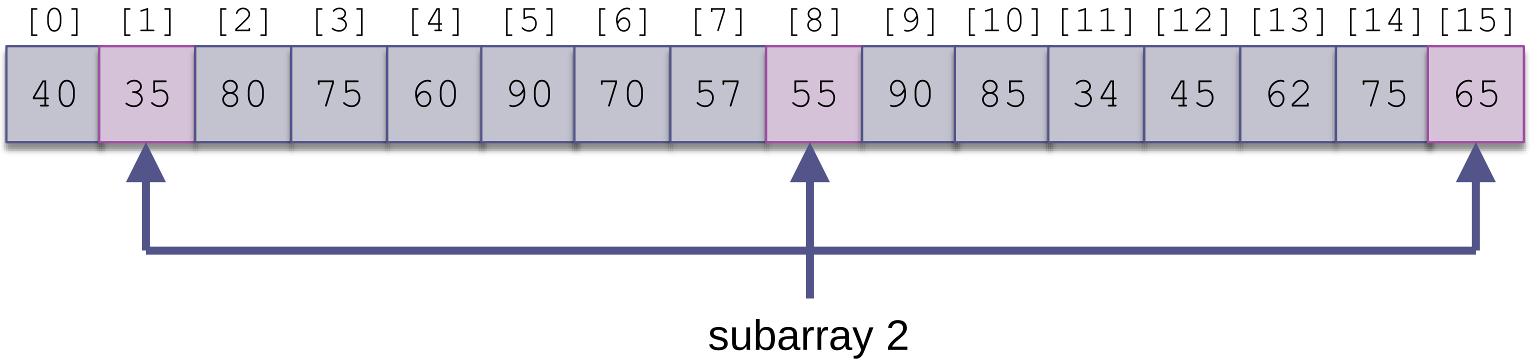
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	57	55	90	85	34	45	62	75	65

subarray 1

Trace of Shell Sort (cont.)

gap value	7

Sort subarray 2



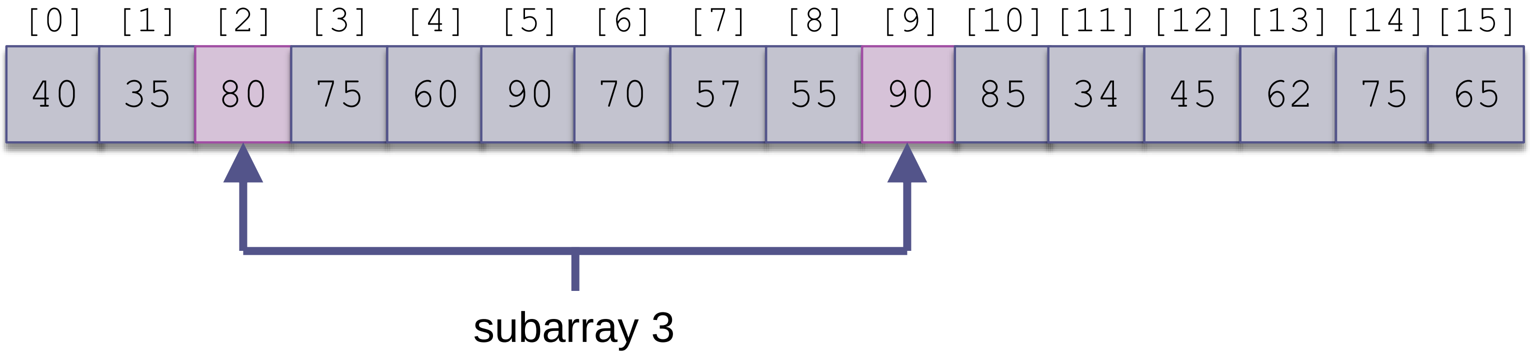
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	57	55	90	85	34	45	62	75	65

subarray 2

Trace of Shell Sort (cont.)

gap value	7

Sort subarray 3



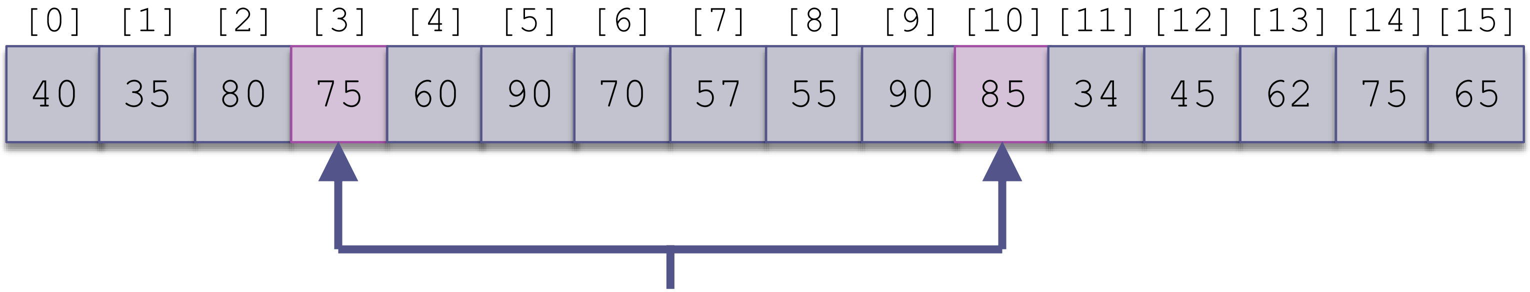
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	57	55	90	85	34	45	62	75	65

subarray 3

Trace of Shell Sort (cont.)

gap value	7

Sort subarray 4



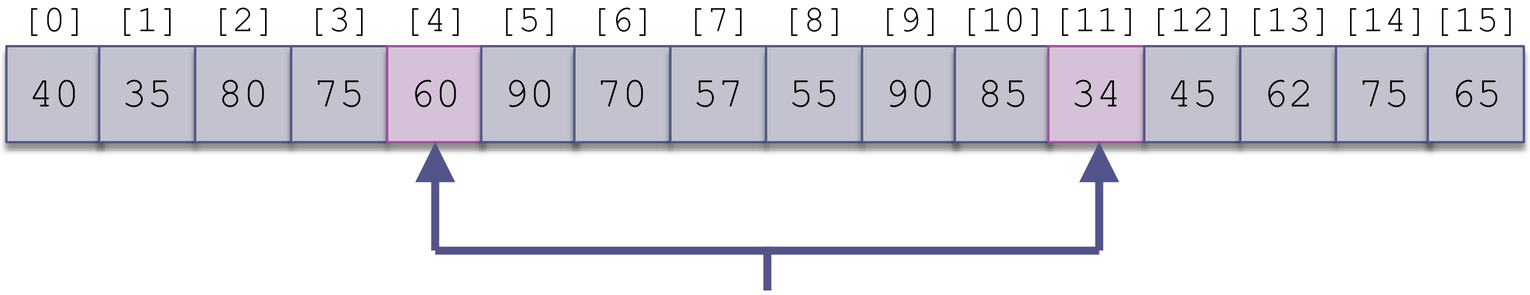
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	57	55	90	85	34	45	62	75	65

subarray 4

Trace of Shell Sort (cont.)

gap value	7

Sort subarray 5



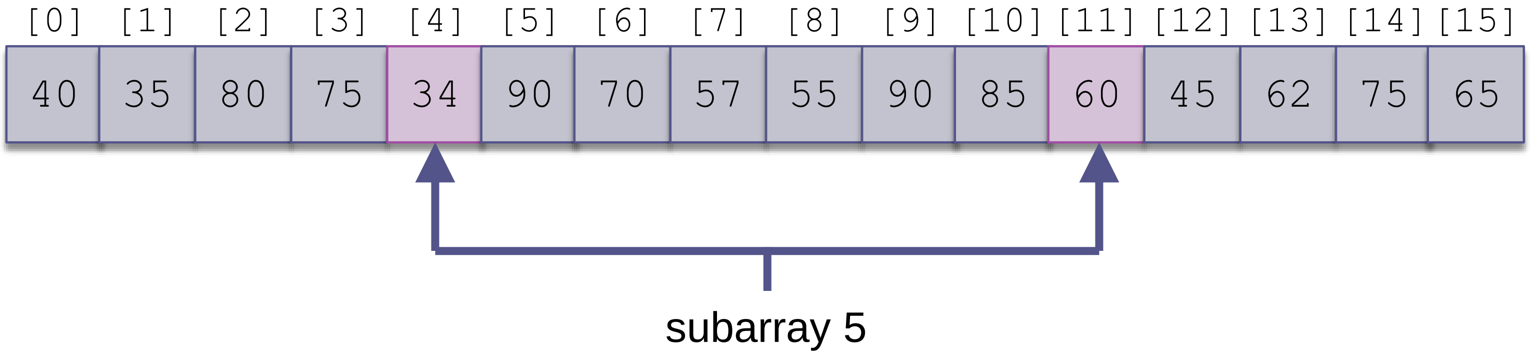
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	60	90	70	57	55	90	85	34	45	62	75	65

subarray 5

Trace of Shell Sort (cont.)

gap value	7

Sort subarray 5



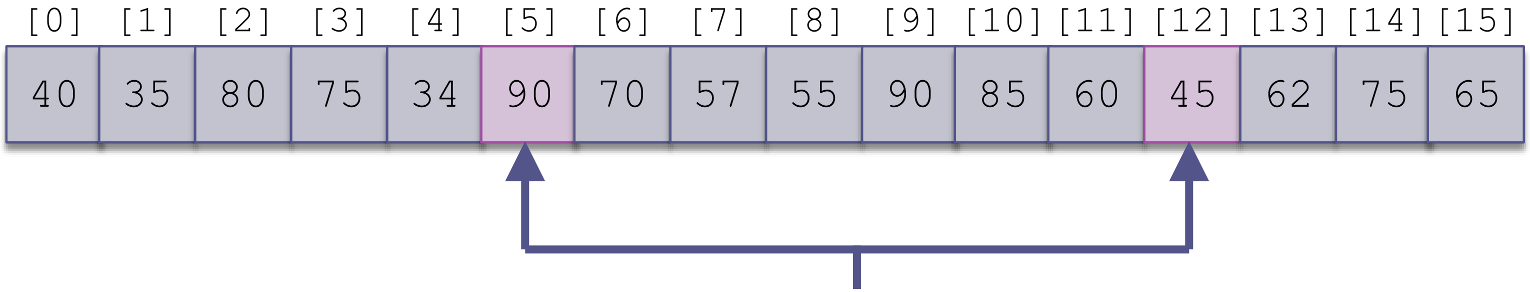
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	34	90	70	57	55	90	85	60	45	62	75	65

subarray 5

Trace of Shell Sort (cont.)

gap value	7

Sort subarray 6



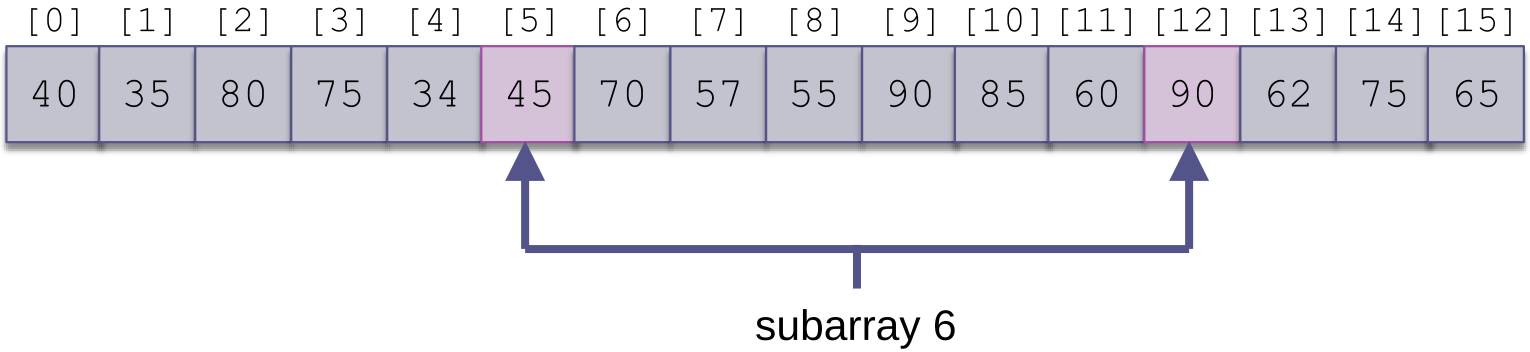
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	34	90	70	57	55	90	85	60	45	62	75	65

subarray 6

Trace of Shell Sort (cont.)

gap value	7

Sort subarray 6



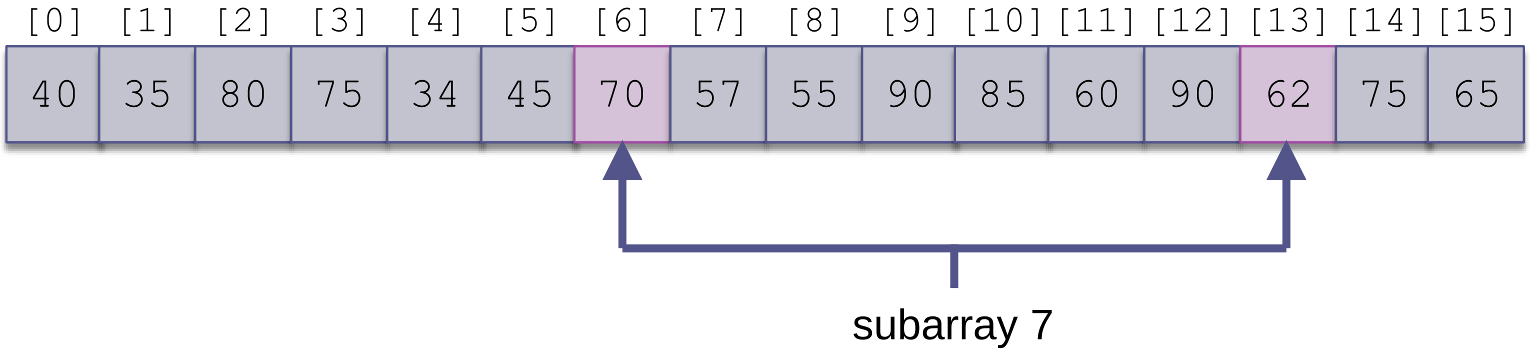
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	34	45	70	57	55	90	85	60	90	62	75	65

subarray 6

Trace of Shell Sort (cont.)

gap value	7

Sort subarray 7



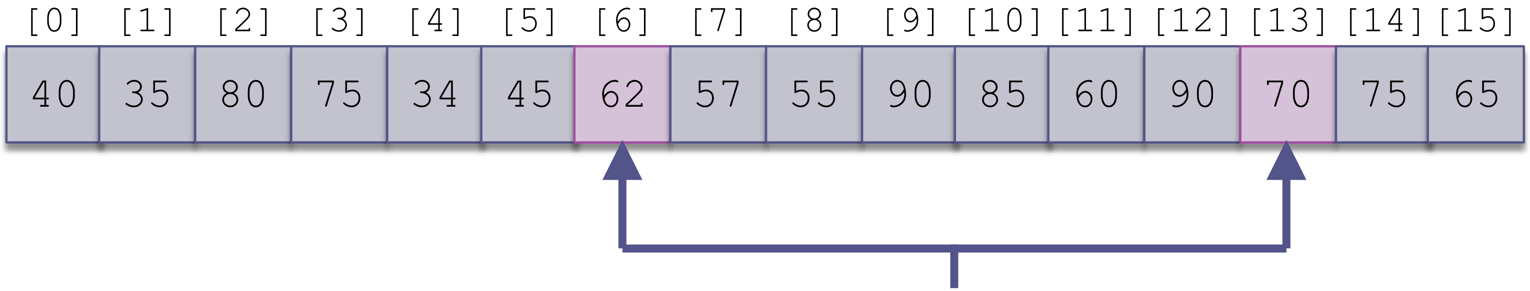
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	34	45	70	57	55	90	85	60	90	62	75	65

subarray 7

Trace of Shell Sort (cont.)

gap value	7

Sort subarray 7



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	34	45	62	57	55	90	85	60	90	70	75	65

subarray 7

Trace of Shell Sort (cont.)

gap value	7

Sort on smaller gap value next

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	34	45	62	57	55	90	85	60	90	70	75	65

Trace of Shell Sort (cont.)

gap value	3

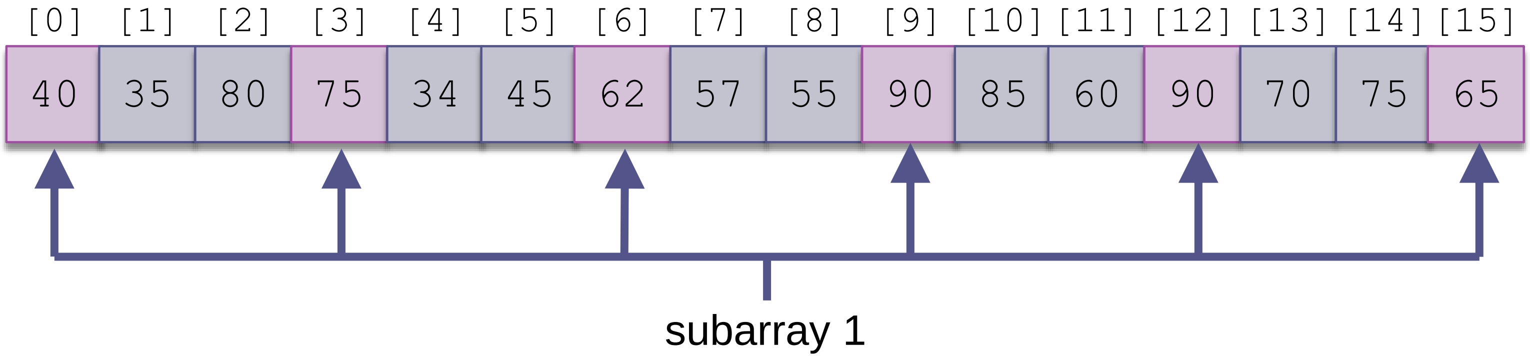
Sort on smaller gap value

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	34	45	62	57	55	90	85	60	90	70	75	65

Trace of Shell Sort (cont.)

gap value	3

Sort subarray 1

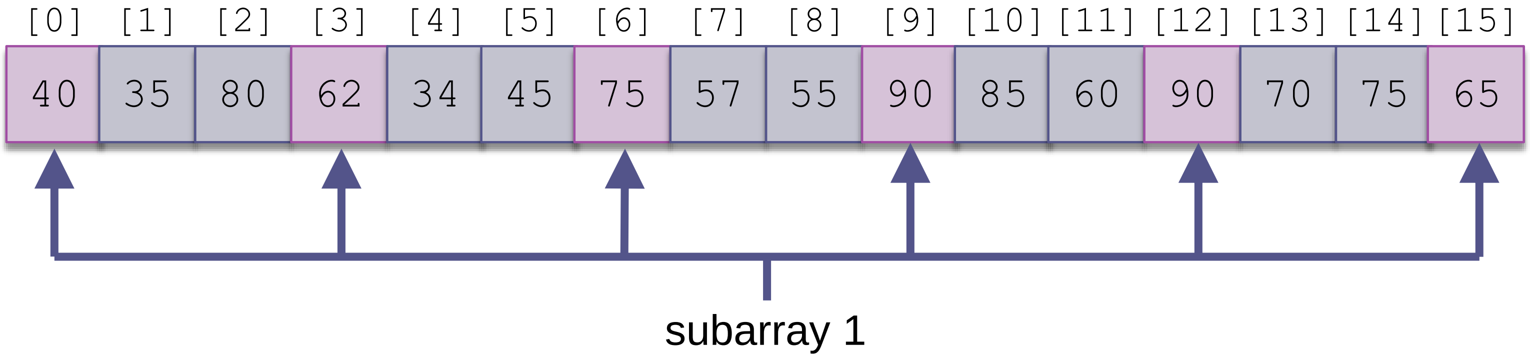


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	75	34	45	62	57	55	90	85	60	90	70	75	65

Trace of Shell Sort (cont.)

gap value	3

Sort subarray 1



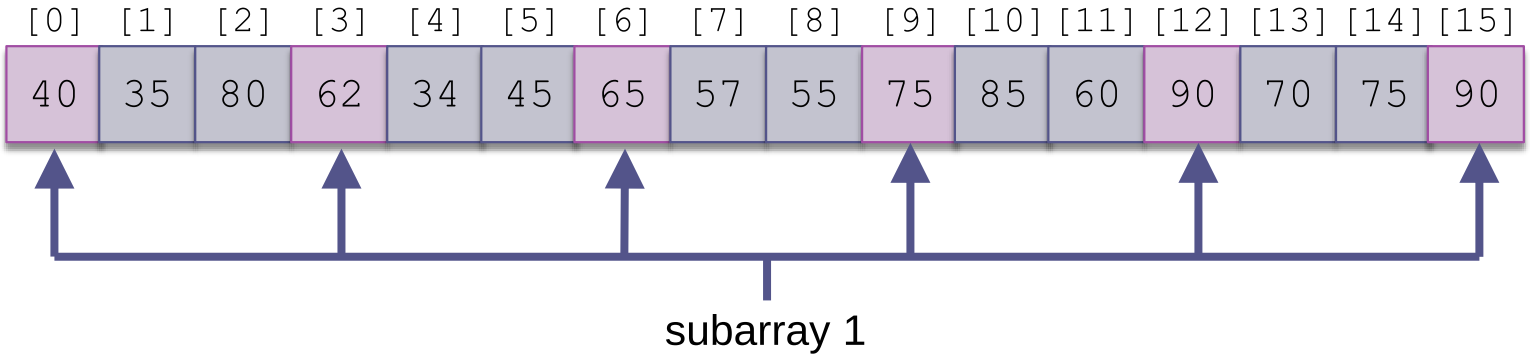
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	62	34	45	75	57	55	90	85	60	90	70	75	65

subarray 1

Trace of Shell Sort (cont.)

gap value	3

Sort subarray 1



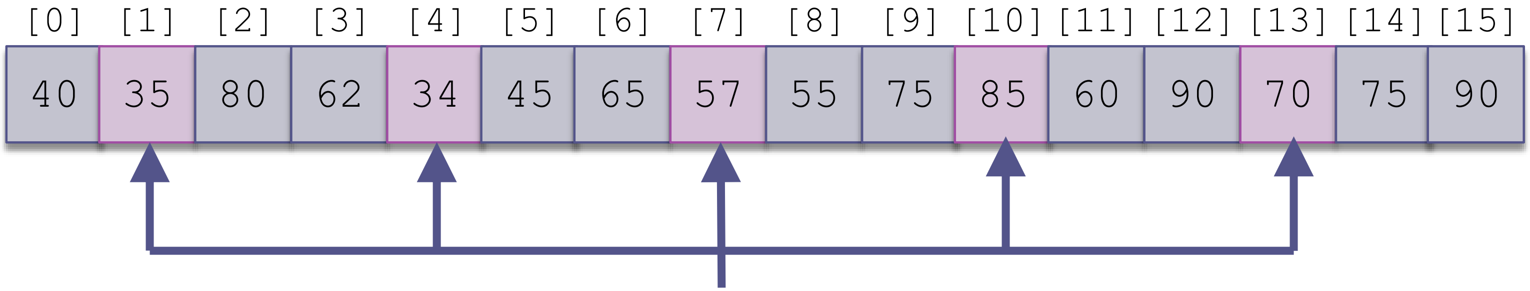
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	62	34	45	65	57	55	75	85	60	90	70	75	90

subarray 1

Trace of Shell Sort (cont.)

gap value	3

Sort subarray 2



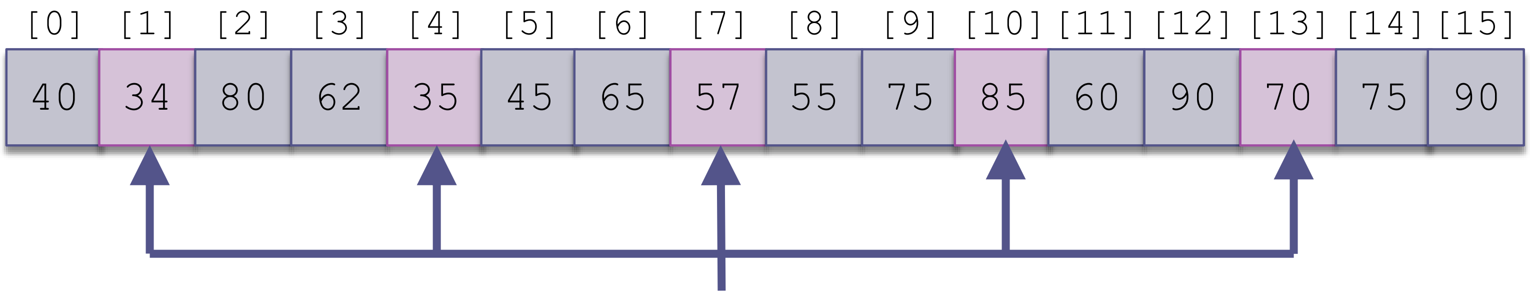
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	35	80	62	34	45	65	57	55	75	85	60	90	70	75	90

subarray 2

Trace of Shell Sort (cont.)

gap value	3

Sort subarray 2



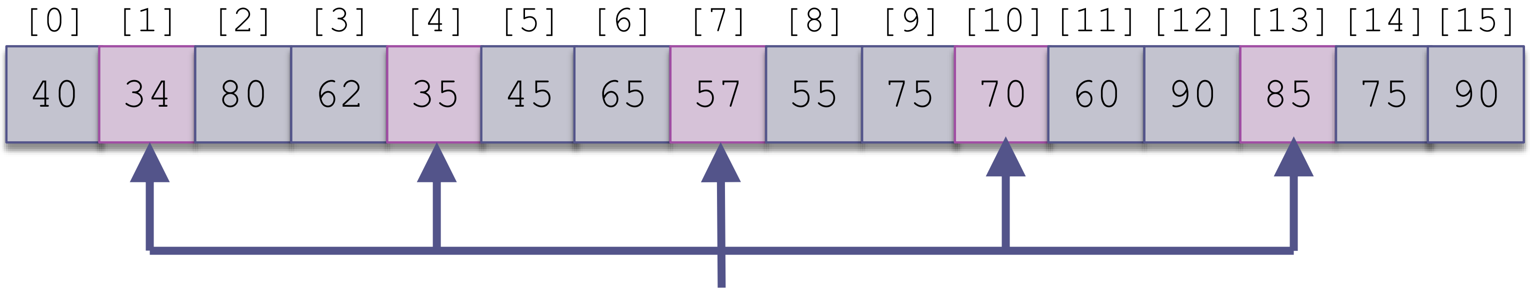
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	34	80	62	35	45	65	57	55	75	85	60	90	70	75	90

subarray 2

Trace of Shell Sort (cont.)

gap value	3

Sort subarray 2



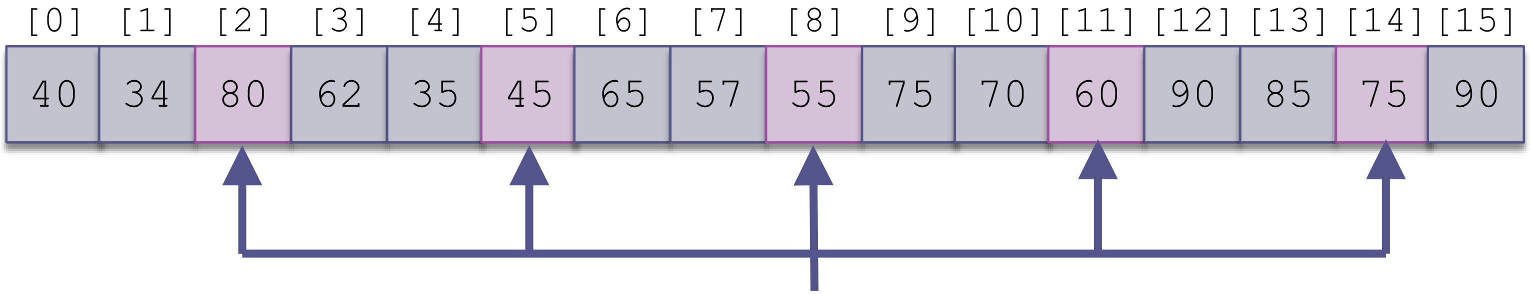
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	34	80	62	35	45	65	57	55	75	70	60	90	85	75	90

subarray 2

Trace of Shell Sort (cont.)

gap value	3

Sort subarray 3



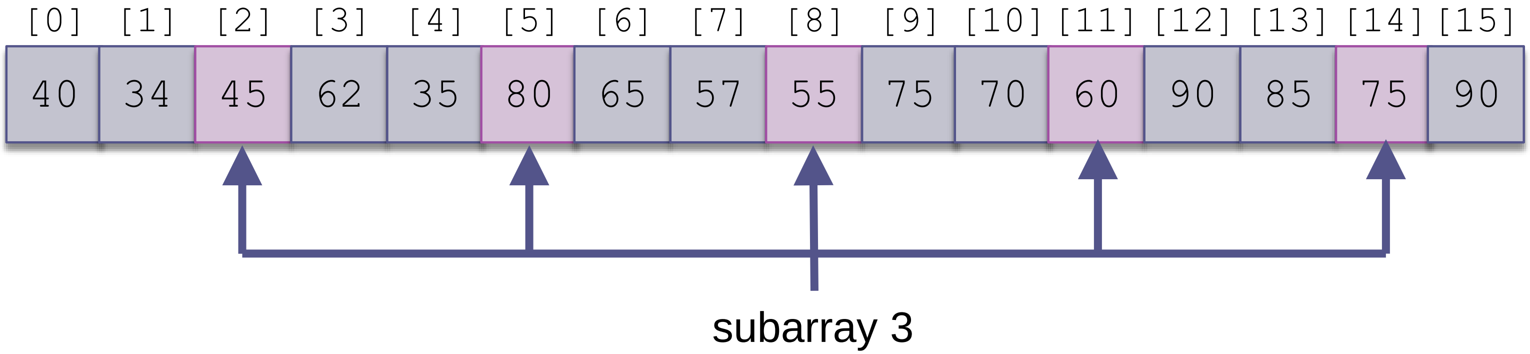
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	34	80	62	35	45	65	57	55	75	70	60	90	85	75	90

subarray 3

Trace of Shell Sort (cont.)

gap value	3

Sort subarray 3



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	34	45	62	35	80	65	57	55	75	70	60	90	85	75	90

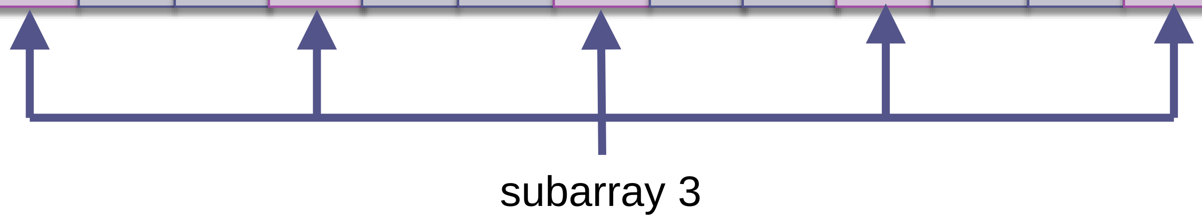
subarray 3

Trace of Shell Sort (cont.)

gap value	3

Sort subarray 3

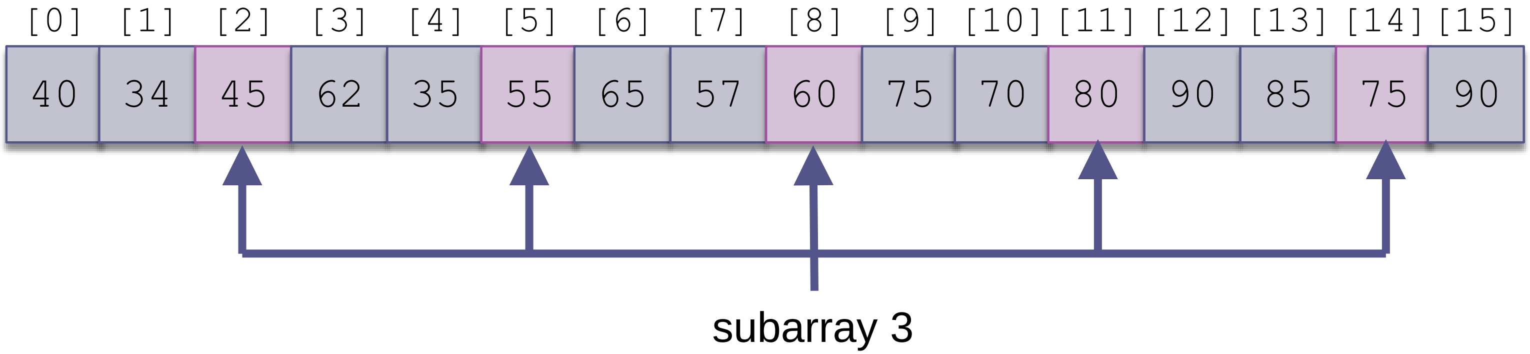
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	34	45	62	35	55	65	57	80	75	70	60	90	85	75	90



Trace of Shell Sort (cont.)

gap value	3

Sort subarray 3



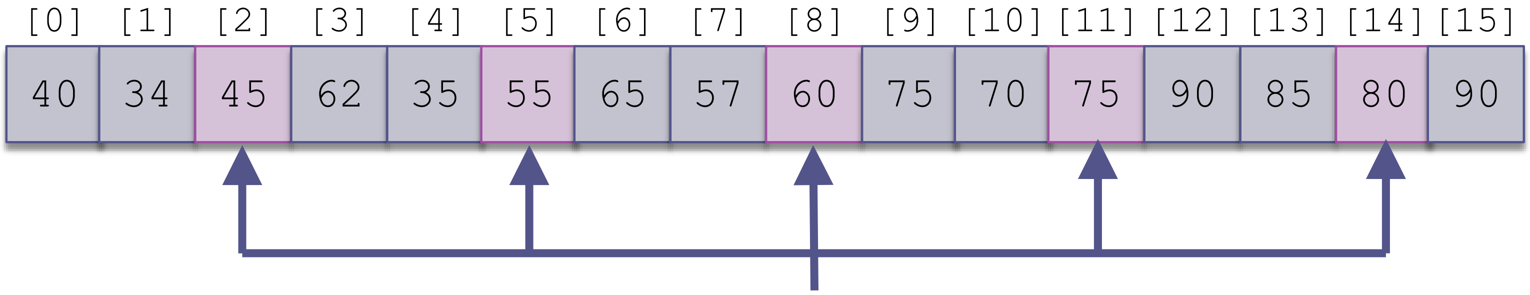
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	34	45	62	35	55	65	57	60	75	70	80	90	85	75	90

subarray 3

Trace of Shell Sort (cont.)

gap value	3

Sort subarray 3



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	34	45	62	35	55	65	57	60	75	70	75	90	85	80	90

subarray 3

Trace of Shell Sort (cont.)

gap value	3

Sort on gap value of 1 (a regular insertion sort)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	34	45	62	35	55	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

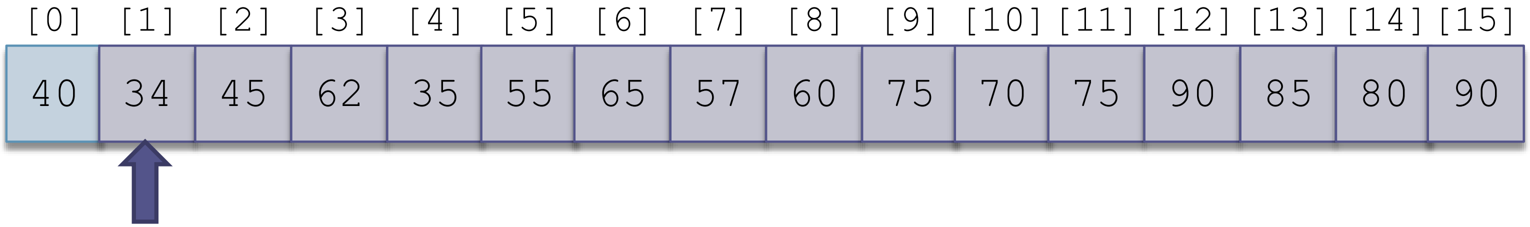
Sort on gap value of 1 (a regular insertion sort)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	34	45	62	35	55	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

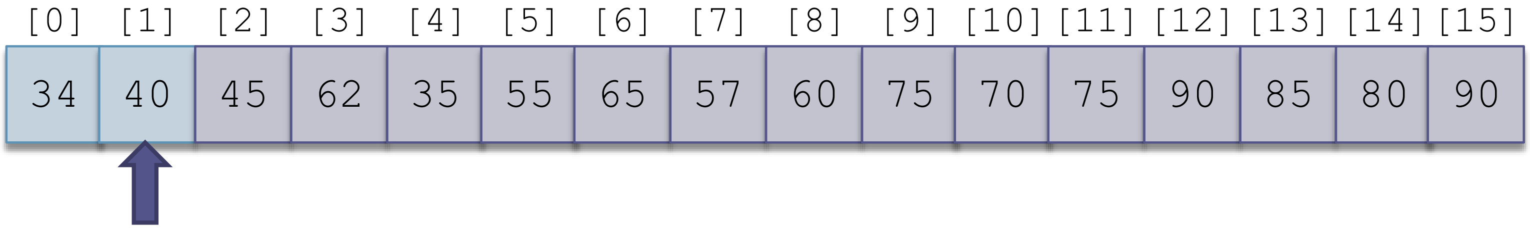


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
40	34	45	62	35	55	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

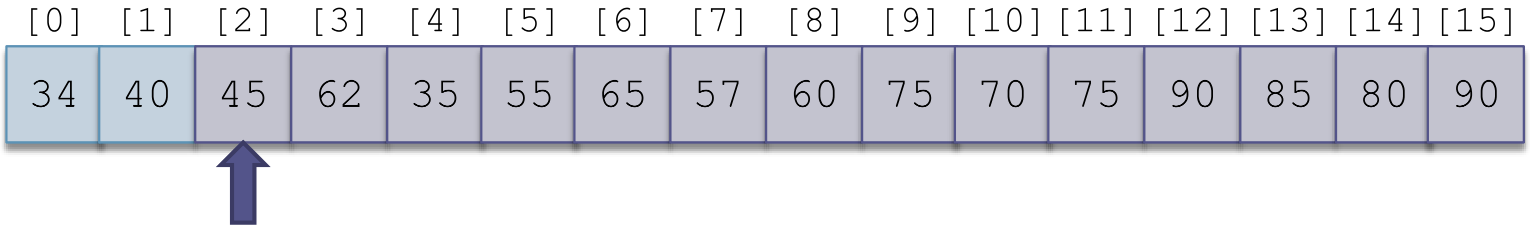


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	40	45	62	35	55	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

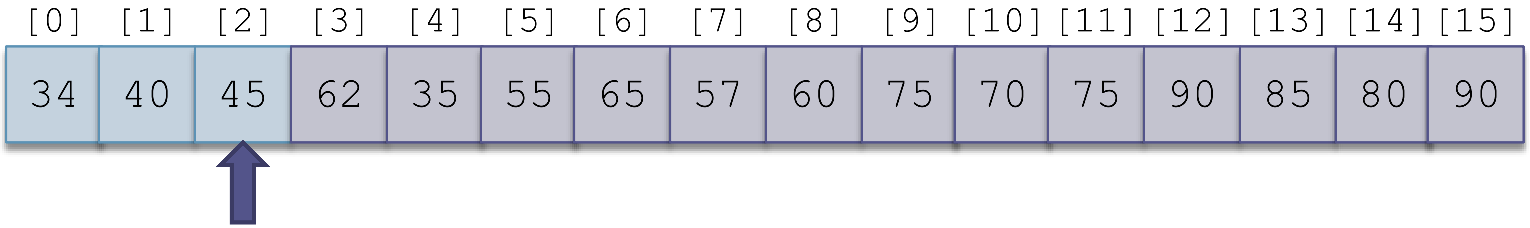


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	40	45	62	35	55	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

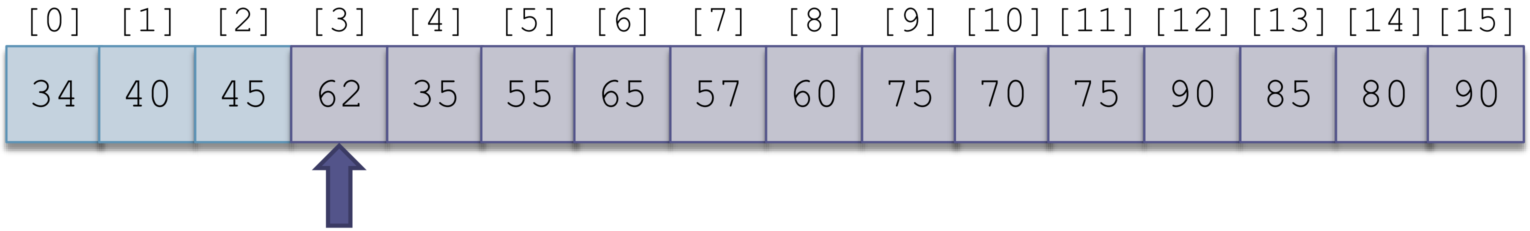


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	40	45	62	35	55	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

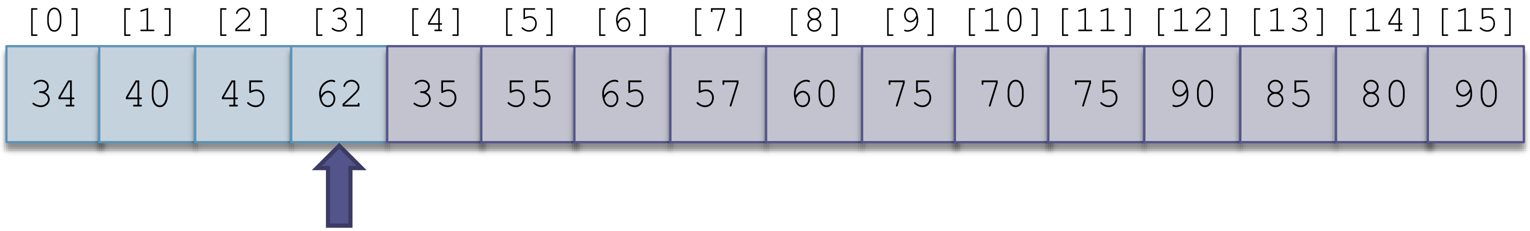


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	40	45	62	35	55	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

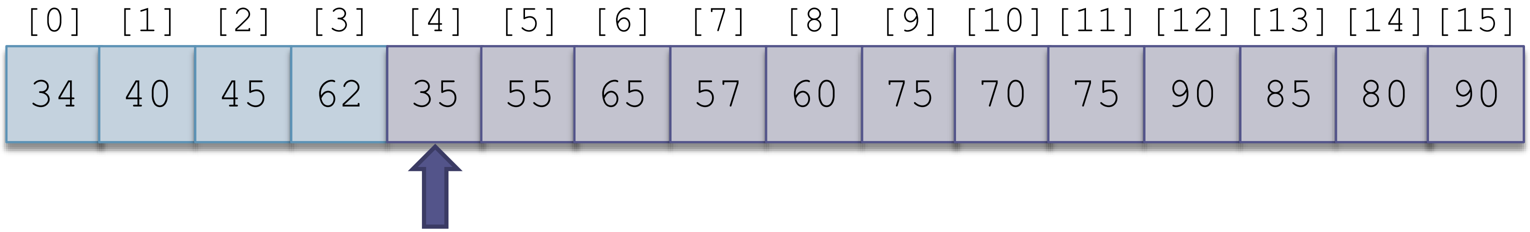


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	40	45	62	35	55	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

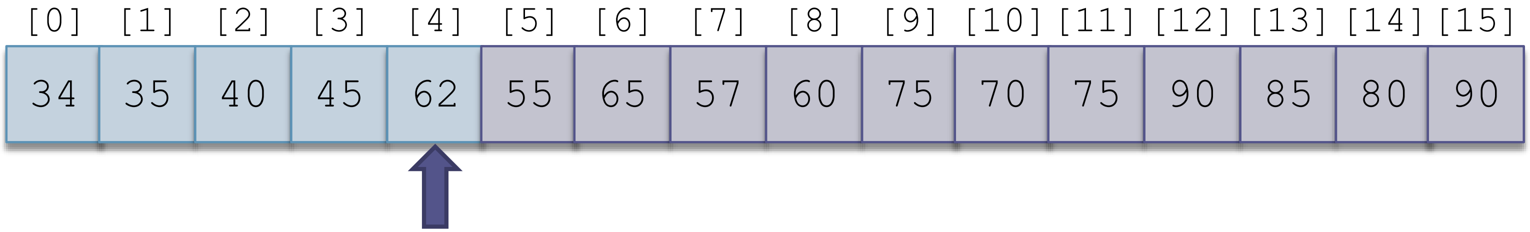


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	40	45	62	35	55	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

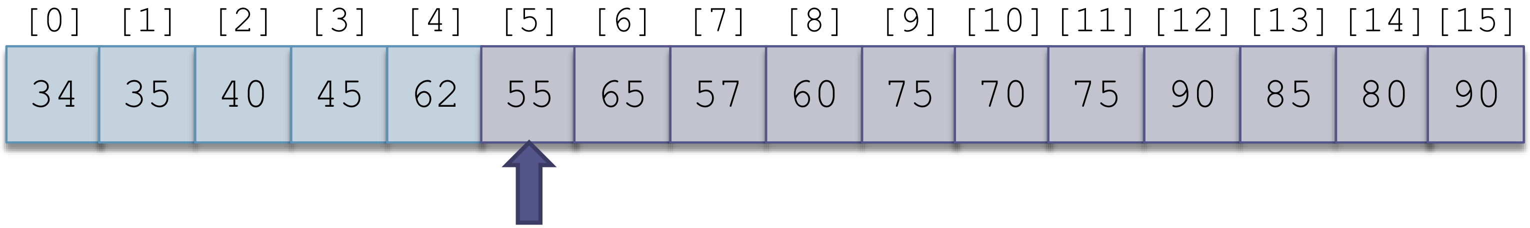


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	62	55	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

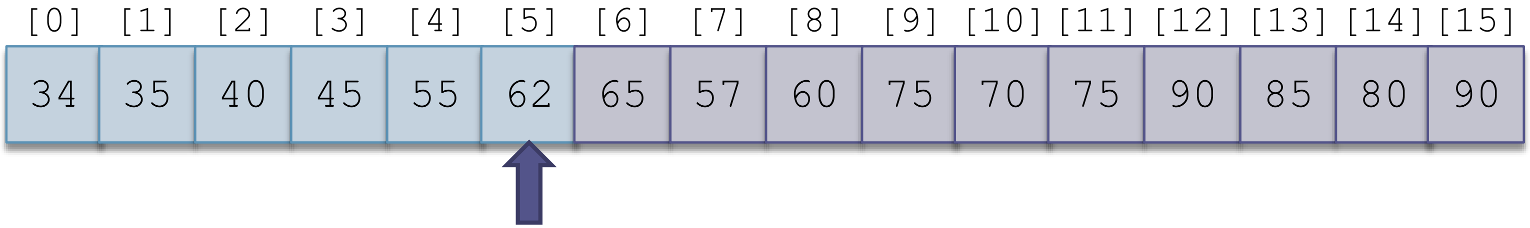


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	62	55	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

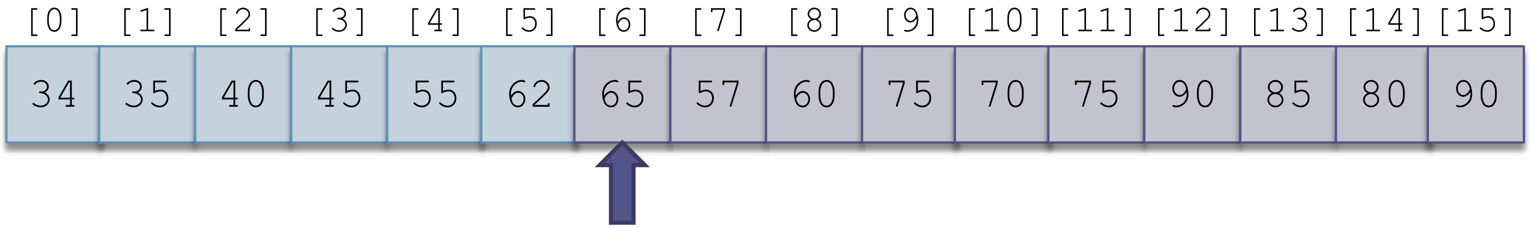


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	62	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

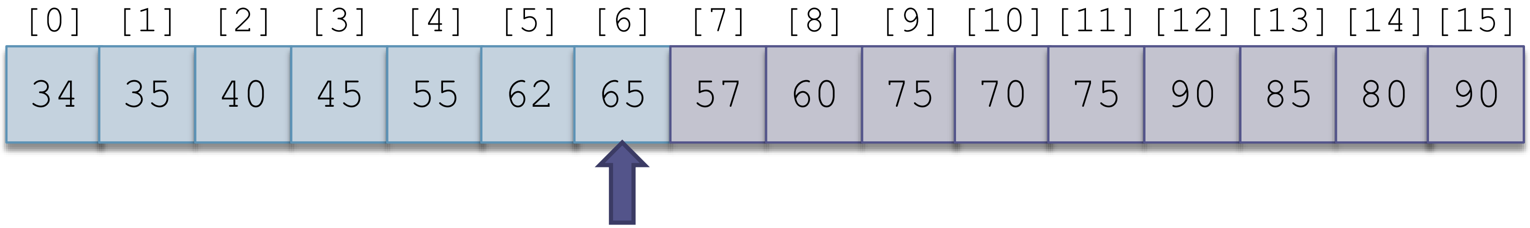


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	62	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

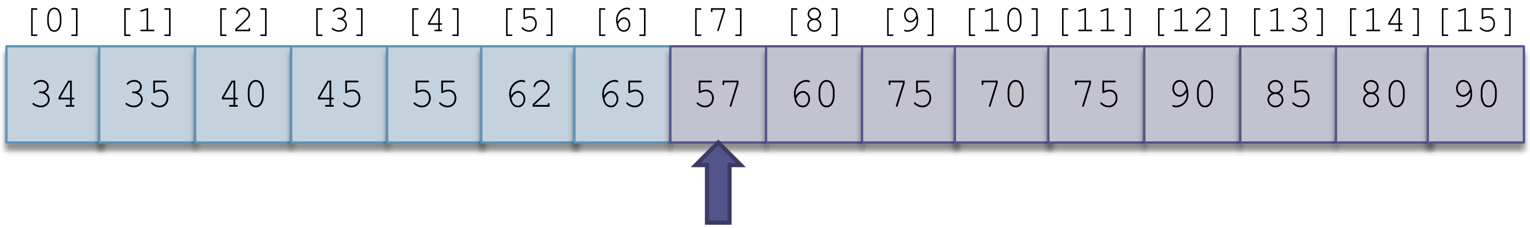


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	62	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

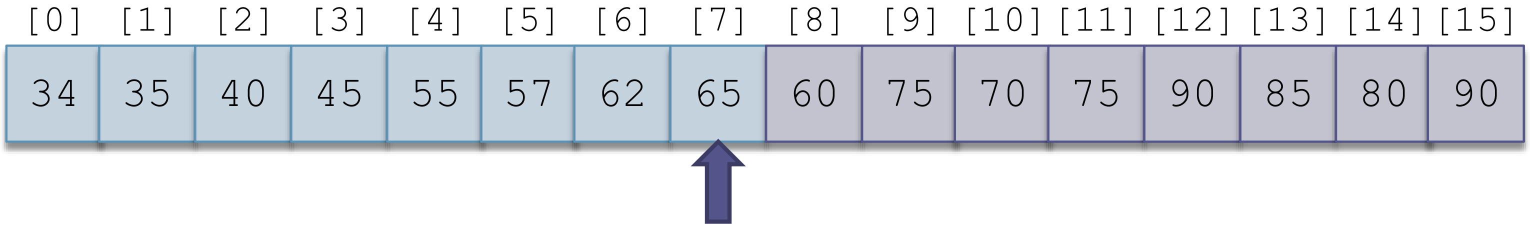


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	62	65	57	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

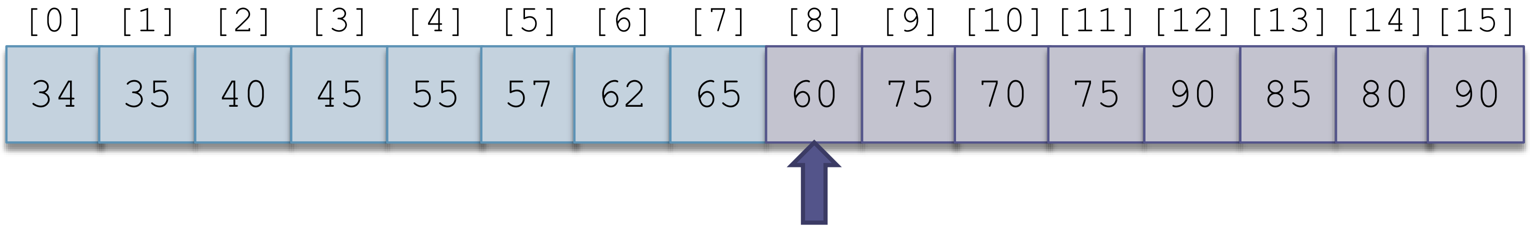


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	62	65	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

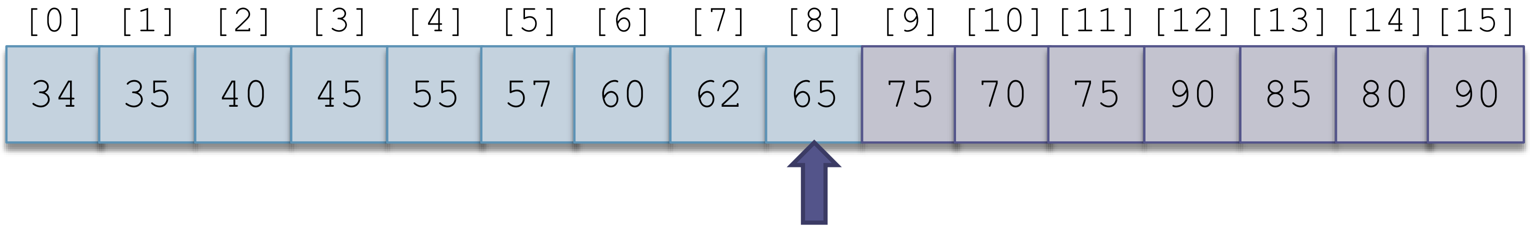


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	62	65	60	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

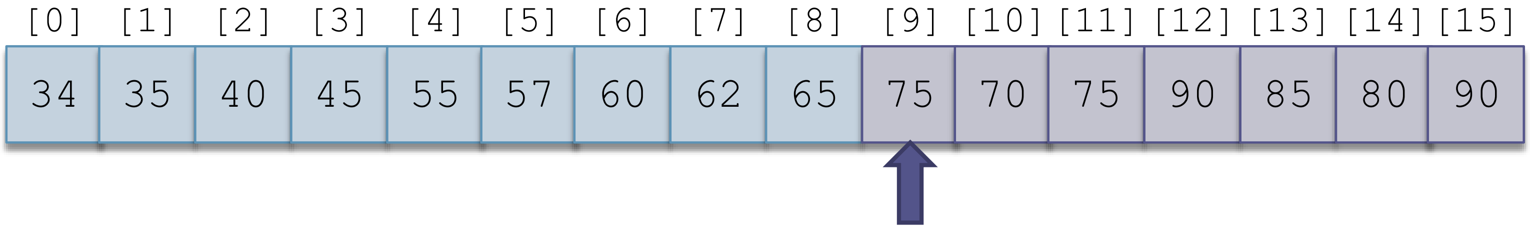


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

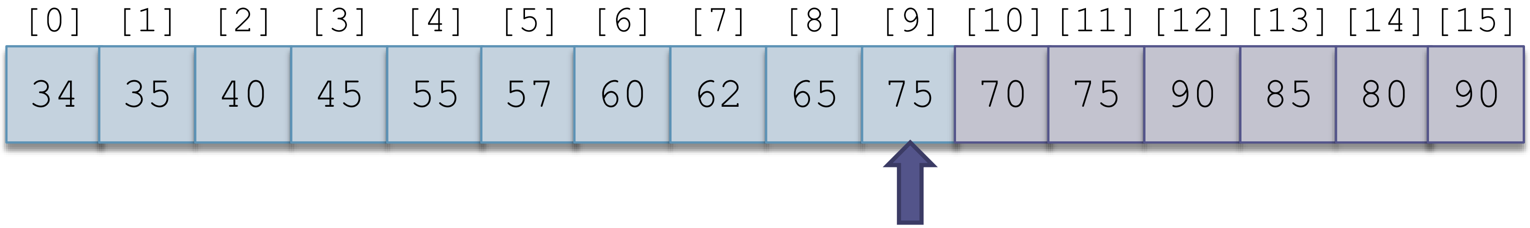


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

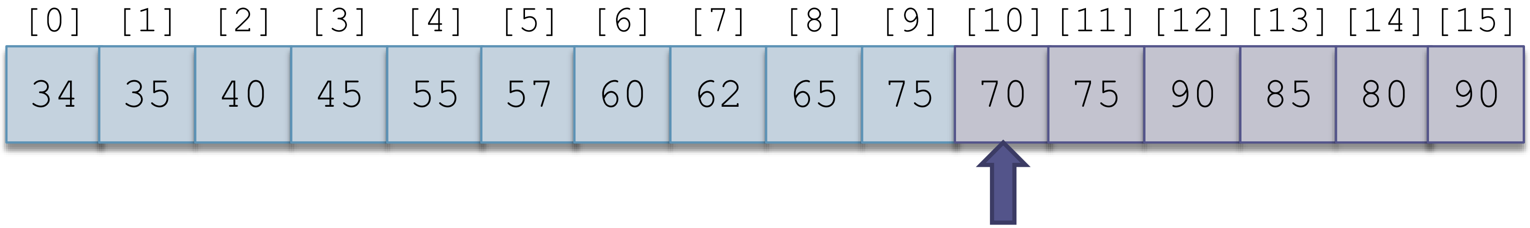


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

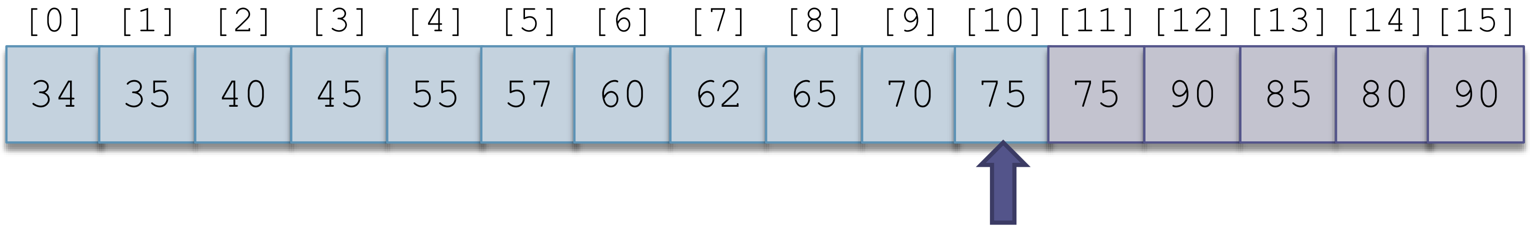


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	75	70	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

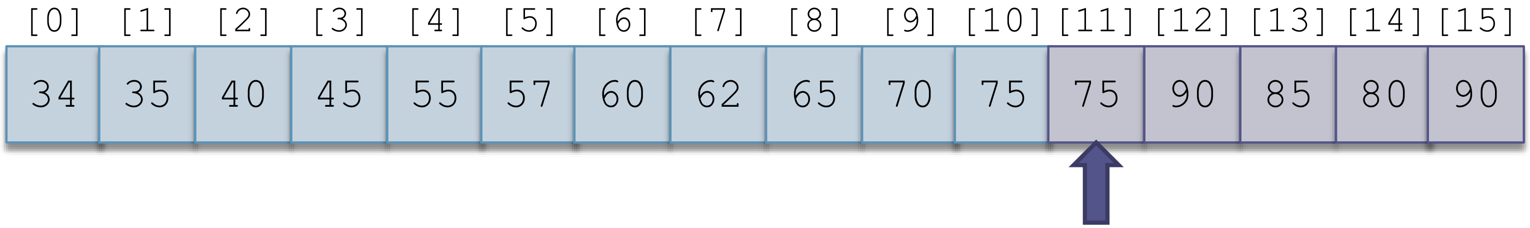


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	70	75	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

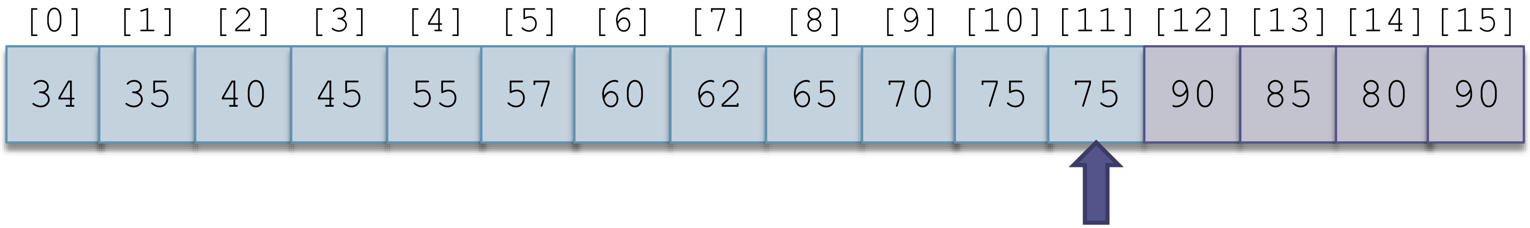


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	70	75	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

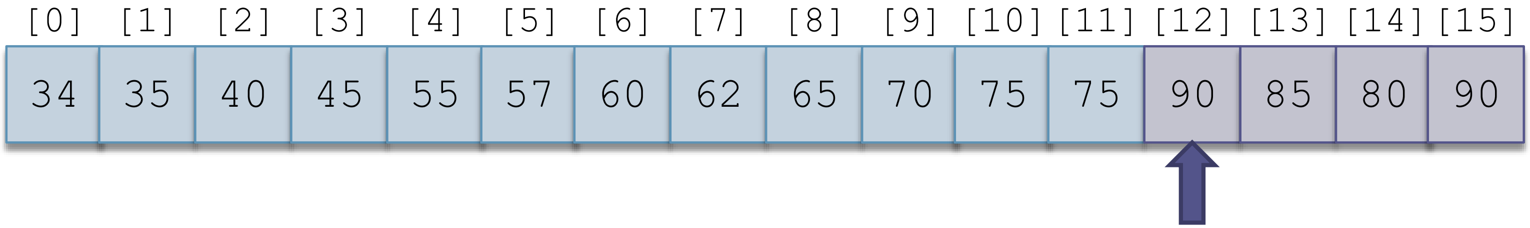


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	70	75	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

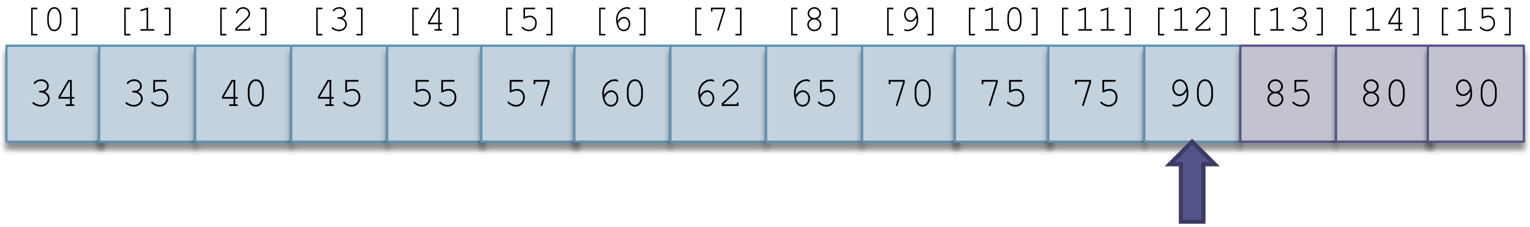


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	70	75	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

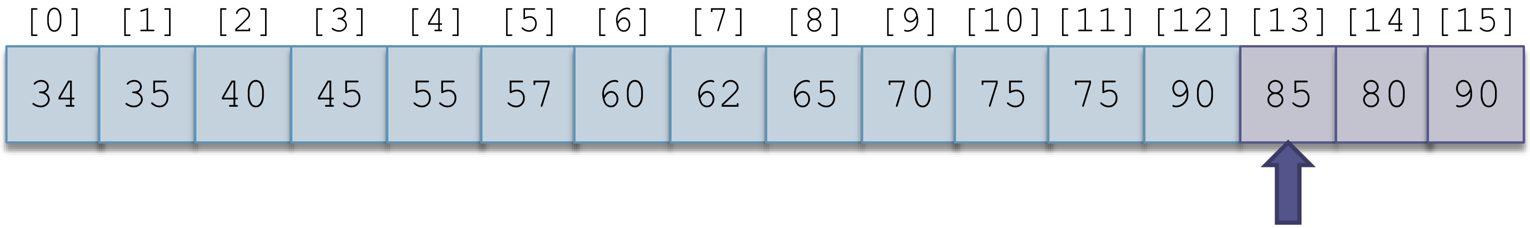


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	70	75	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

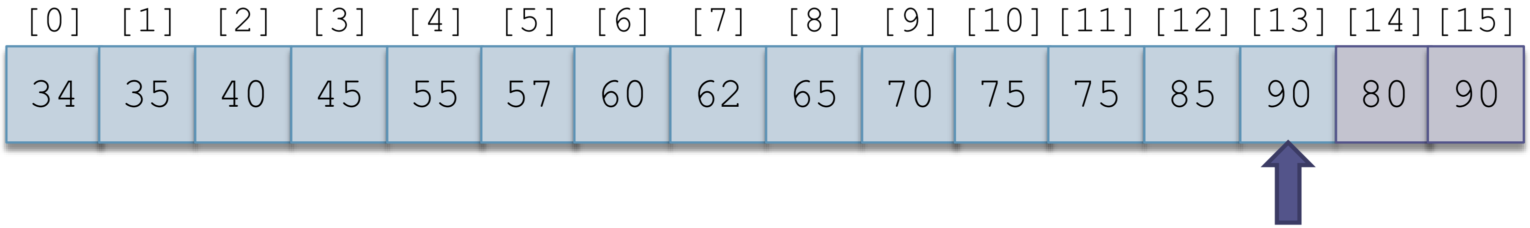


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	70	75	75	90	85	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	70	75	75	85	90	80	90

Trace of Shell Sort (cont.)

gap value	1

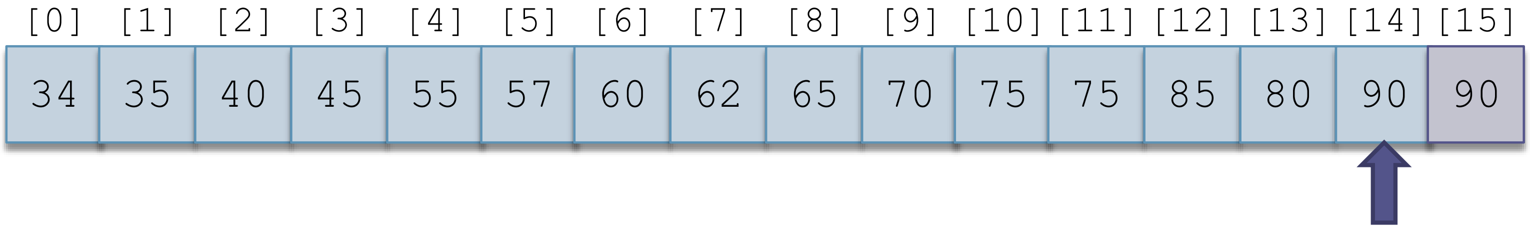
Sort on gap value of 1 (a regular insertion sort)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	70	75	75	85	90	80	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

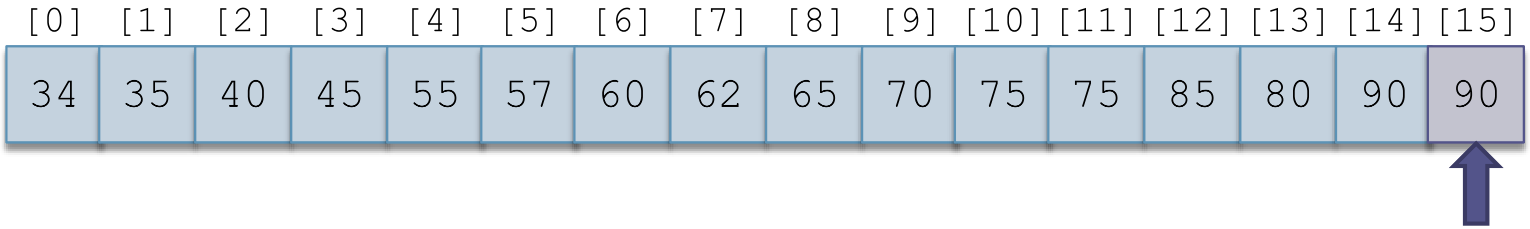


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	70	75	75	85	80	90	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	70	75	75	85	80	90	90

Trace of Shell Sort (cont.)

gap value	1

Sort on gap value of 1 (a regular insertion sort)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
34	35	40	45	55	57	60	62	65	70	75	75	85	80	90	90

Shell Sort Algorithm

Shell Sort Algorithm

```
Set the initial value of gap to n / 2
while gap > 0
    for each array element from position gap to the last element
        Insert this element where it belongs in its subarray.
    if gap is 2, set it to 1
    else gap = gap / 2.2. // chosen by experimentation
```

Shell Sort Algorithm (cont.)

Refinement of Step 4, the Insertion Step

```
4.1   nextPos is the position of the element to insert
4.2   Save the value of the element to insert in nextVal
4.3   while nextPos > gap and the element at nextPos – gap > nextVal
4.4       Shift the element at nextPos – gap to position nextPos
4.5       Decrement nextPos by gap
4.6   Insert nextVal at nextPos
```

Analysis of Shell Sort

- Because the behavior of insertion sort is closer to O(*n*) than O(n^2) when an array is nearly sorted, presorting speeds up later sorting
- This is critical when sorting large arrays

Analysis of Shell Sort (cont.)

- A general analysis of Shell sort is an open research problem in computer science
- Performance depends on how the decreasing sequence of values for `gap` is chosen
- If successive powers of 2 are used for `gap`, performance is O(n^2)
- If successive values for `gap` are based on *Hibbard's sequence*,

  $\texttt{Gap(k)} = 2^k - 1,\ \texttt{k} = \texttt{m},\ \texttt{m-1},\ \ldots\ 1$

  it can be proven that the performance is O($n^{3/2}$)
-

Analysis of Shell Sort (cont.)

- Our algorithm selects the initial value of `gap` as $\frac{n}{2}$ and then divides by 2.2 and truncates the result
- Empirical studies of this approach show that the performance is O($n^{5/4}$) or even O($n^{7/6}$) , but there is no theoretical basis for this result

Code for Shell Sort

- Listing 8.4 (`ShellSort.java`, pages 439 – 440)

Testing the Sort Algorithms

Testing the Sort Algorithms

- Use a variety of test cases
  - small and large arrays
  - arrays in random order
  - arrays that are already sorted
  - arrays with duplicate values
- Compare performance on each type of array

Driver to Test Sort Algorithms

- Listing 8.11(`TestSort.java`, page 461)

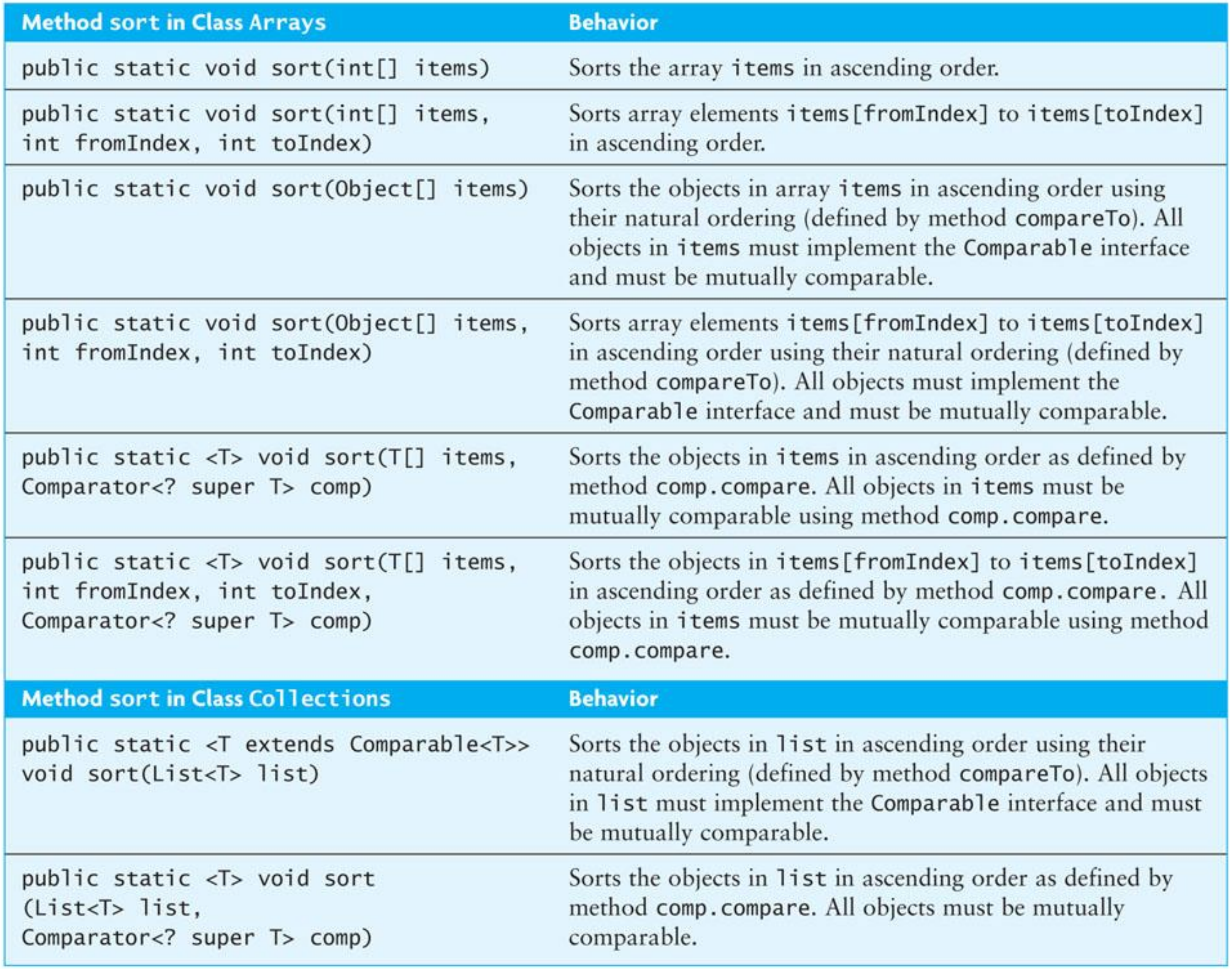
Comparison of Sort Algorithms

Summary

Using Java Sorting Methods

- The Java API provides a class `Arrays` with several overloaded sort methods for different array types
- The `Collections` class provides similar sorting methods for `Lists`
- Sorting methods for arrays of primitive types are based on the quicksort algorithm
- Sorting methods for arrays of objects and `Lists` are based on the merge sort algorithm
- Both algorithms are O(*n* log *n*)

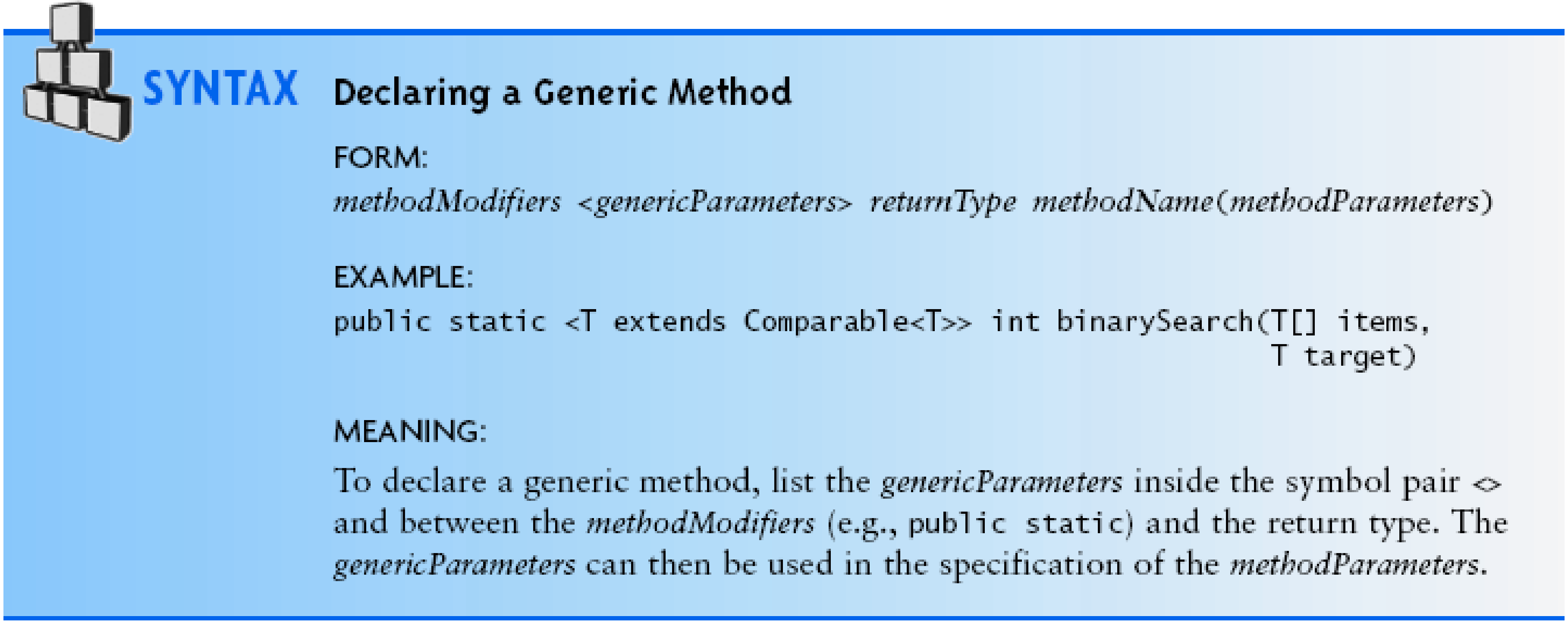
Using Java Sorting Methods (cont.)



Method sort in Class Arrays	Behavior
`public static void sort(int[] items)`	Sorts the array `items` in ascending order.
`public static void sort(int[] items, int fromIndex, int toIndex)`	Sorts array elements `items[fromIndex]` to `items[toIndex]` in ascending order.
`public static void sort(Object[] items)`	Sorts the objects in array `items` in ascending order using their natural ordering (defined by method `compareTo`). All objects in `items` must implement the `Comparable` interface and must be mutually comparable.
`public static void sort(Object[] items, int fromIndex, int toIndex)`	Sorts array elements `items[fromIndex]` to `items[toIndex]` in ascending order using their natural ordering (defined by method `compareTo`). All objects must implement the `Comparable` interface and must be mutually comparable.
`public static <T> void sort(T[] items, Comparator<? super T> comp)`	Sorts the objects in `items` in ascending order as defined by method `comp.compare`. All objects in `items` must be mutually comparable using method `comp.compare`.
`public static <T> void sort(T[] items, int fromIndex, int toIndex, Comparator<? super T> comp)`	Sorts the objects in `items[fromIndex]` to `items[toIndex]` in ascending order as defined by method `comp.compare`. All objects in `items` must be mutually comparable using method `comp.compare`.

Method sort in Class Collections	Behavior
`public static <T extends Comparable<T>> void sort(List<T> list)`	Sorts the objects in `list` in ascending order using their natural ordering (defined by method `compareTo`). All objects in `list` must implement the `Comparable` interface and must be mutually comparable.
`public static <T> void sort (List<T> list, Comparator<? super T> comp)`	Sorts the objects in `list` in ascending order as defined by method `comp.compare`. All objects must be mutually comparable.

Declaring a Generic Method



SYNTAX **Declaring a Generic Method**

FORM:

methodModifiers <genericParameters> returnType methodName(methodParameters)

EXAMPLE:

```
public static <T extends Comparable<T>> int binarySearch(T[] items,
                                                  T target)
```

MEANING:

To declare a generic method, list the *genericParameters* inside the symbol pair <> and between the *methodModifiers* (e.g., `public static`) and the return type. The *genericParameters* can then be used in the specification of the *methodParameters*.

Declaring a Generic Method (cont.)

- Sample declarations:

```
public static <T> void sort(T[] items, Comparator<? super T> comp)
```

`<T>` represents the generic parameter for the sort method

Declaring a Generic Method (cont.)

- Sample declarations:

```
public static <T> void sort(T[] items, Comparator<? super T> comp)
```

`<T>` should also appear in the method parameter list

Declaring a Generic Method (cont.)

- Sample declarations:

```
public static <T> void sort(T[] items, Comparator<? super T> comp)
```

The second method parameter means that `comp` must be an object that implements the `Comparator` interface for type `T` or for a superclass of type `T`

Declaring a Generic Method (cont.)

- Sample declarations:

```
public static <T> void sort(T[] items, Comparator<? super T> comp)
```

For example, we can define a class that implements `Comparator<Number>` and use it to sort an array of `Integer` objects or an array of `Double` objects

Declaring a Generic Method (cont.)

- Sample declarations:

```
public static <T extends Comparable<T>> void sort(List<T> list)
```

`<T extends Comparable<T>>` means that generic parameter `T` must implement the interface `Comparable<T>`

Declaring a Generic Method (cont.)

- Sample declarations:

```
public static <T extends Comparable<T>> void sort(List<T> list)
```

The method parameter `list` (the object being sorted) is of type `List<T>`

Example 8.2

```java
public class Person implements Comparable<Person> {
     // Data Fields
     /* The last name */
     private String lastName;
     /* The first name */
     private String firstName;
     /* Birthday represented by an integer from 1 to 366 */
     private int birthDay;

     // Methods
     /** Compares two Person objects based on names. The result is based on the last names if they are
         different; otherwise, it is based on the first names.
         @param obj The other Person
         @return A negative integer if this person’s name
                 precedes the other person’s name;
                 0 if the names are the same;
                 a positive integer if this person’s name follows the other person’s name.
     */
     @Override
     public int compareTo(Person obj) {
         Person other = obj;
         // Compare this Person to other using last names.
         int result = lastName.compareTo(other.lastName);
         // Compare first names if last names are the same.
         if (result == 0)
             return firstName.compareTo(other.firstName);
         else
             return result;
     }

     // Other methods
     . . .
}
```

Example 8.3

```
import java.util.Comparator;

public class ComparePerson implements Comparator<Person> {
     /** Compare two Person objects based on birth date.
         @param left The left-hand side of the comparison
         @param right The right-hand side of the comparison
         @return A negative integer if the left person’s birthday
                precedes the right person’s birthday;
                0 if the birthdays are the same;
                a positive integer if the left person’s birthday
                follows the right person’s birthday.
     */
     @Override
     public int compare(Person left, Person right) {
          return left.getBirthDay() - right.getBirthDay();
     }
}
```