Naiv sortering

Koffman & Wolfgang

- kapitel 8, avsnitt 1–2 och 4–5

Javas sorteringsmetoder

Java.Arrays innehåller flera sorteringsmetoder för olika sorters fält

- fält med primitiva typer sorteras med QuickSort
- fält med objekt sorteras med MergeSort

Java.Collections innehåller liknande sorteringsmetoder för listor

- listor sorteras med MergeSort

Både QuickSort och MergeSort och flera andra algoritmer är O(n log n)

Javas sorteringsmetoder

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.

Generiska metoder

Allmän struktur:

methodModifiers <genericParameters>
returnType methodName(methodParameters)

Två vanliga exempel:

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


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

Comparable vs Comparator

Exempel på en komparabel klass:

```
public class Person implements Comparable<Person> {
    . . .
    public int compareTo(Person other) {
        return getName().compareTo(other.getName());
    }
    . . .
}
```

Exempel på en komparator:

```
public class CompareDate implements Comparator<Person> {
    public int compare(Person left, Person right) {
        return left.getBirthDay() – right.getBirthDay();
    }
}
```

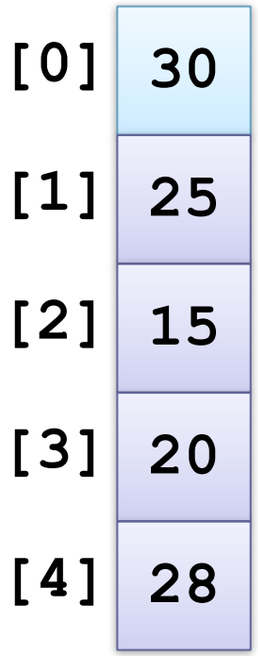
Insättningssortering

Kursboken, avsnitt 8.4

Baseras på hur man kan sortera en korthand:

- plocka upp ett kort i taget
- stoppa in kortet på sin rätta plats i handen

Trace of Insertion Sort

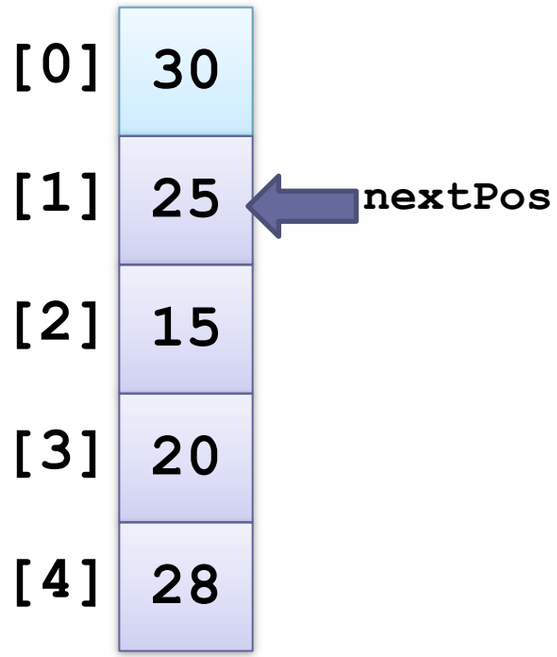


1. `for` each array element from the second (`nextPos` = 1) to the last
2. Insert the element at `nextPos` where it belongs in the array, increasing the length of the sorted subarray by 1 element

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

Trace of Insertion Sort (cont.)

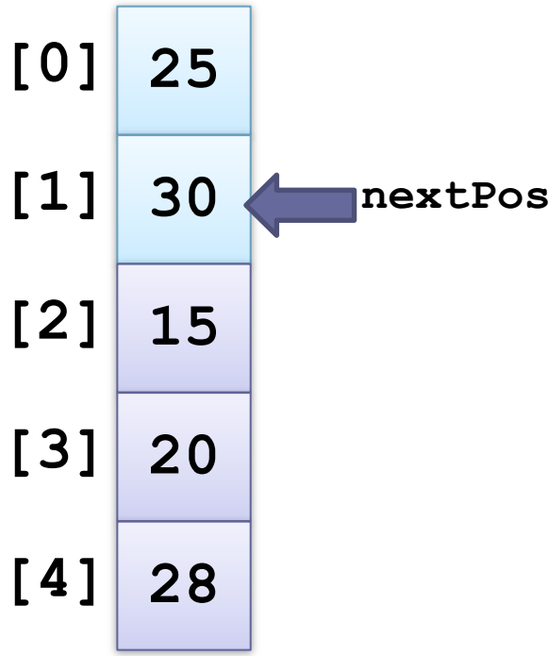
nextPos	1



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

Trace of Insertion Sort (cont.)

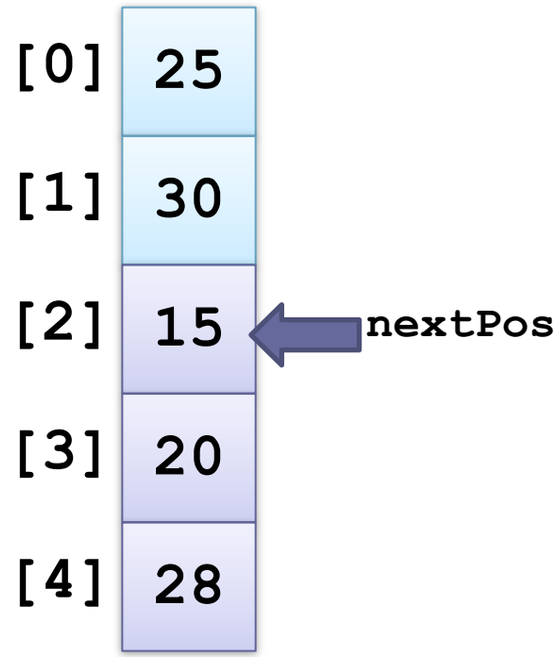
nextPos	1



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

Trace of Insertion Sort (cont.)

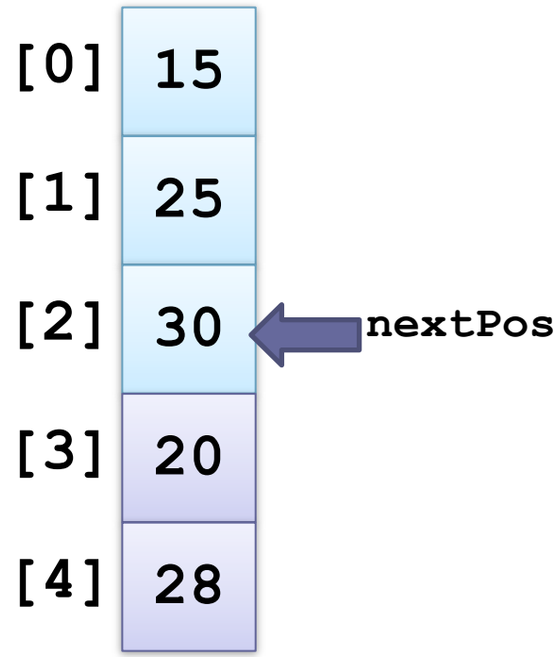
nextPos	2



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

Trace of Insertion Sort (cont.)

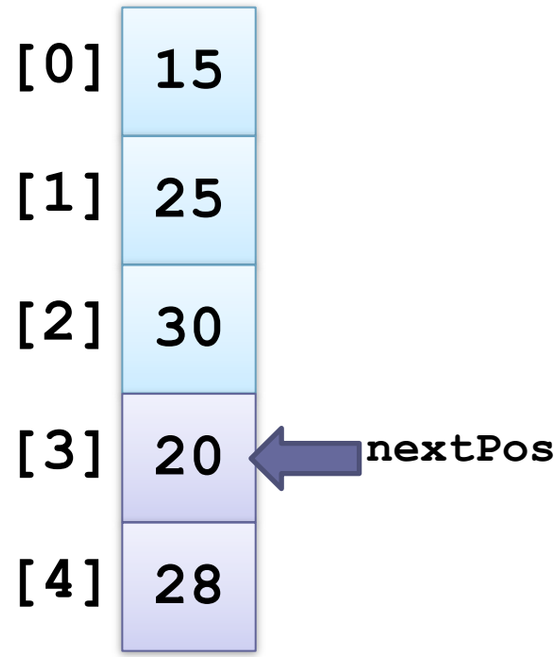
nextPos	2



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

Trace of Insertion Sort (cont.)

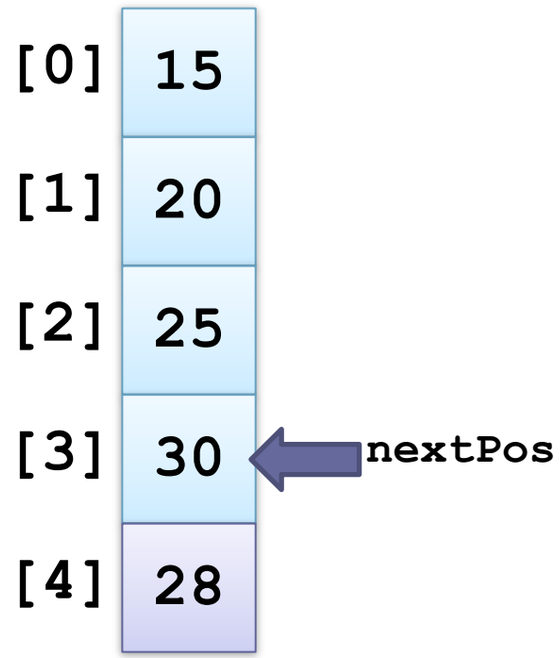
nextPos	3



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

Trace of Insertion Sort (cont.)

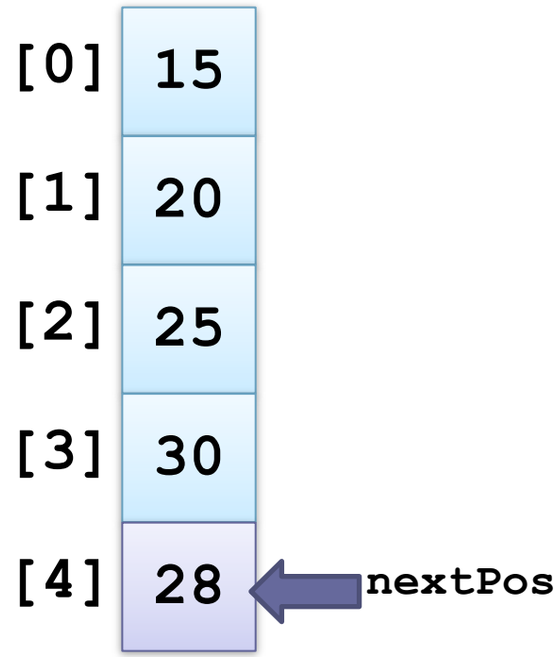
nextPos	3



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

Trace of Insertion Sort (cont.)

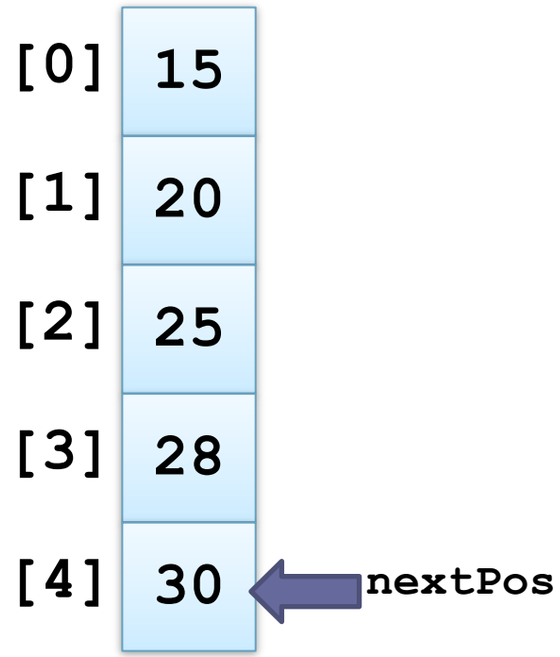
nextPos	4



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

Trace of Insertion Sort (cont.)

nextPos	4



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

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**

Samma exempel, förfinad version

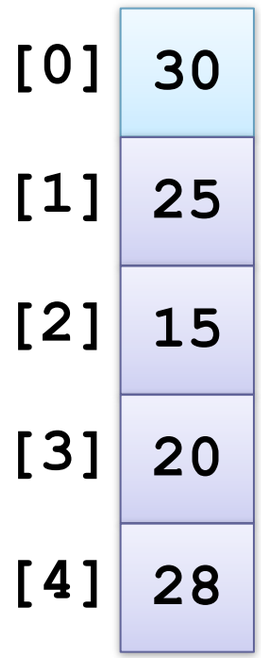
Denhär raden:

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

kan förfinas till en while-loop:

```
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
```

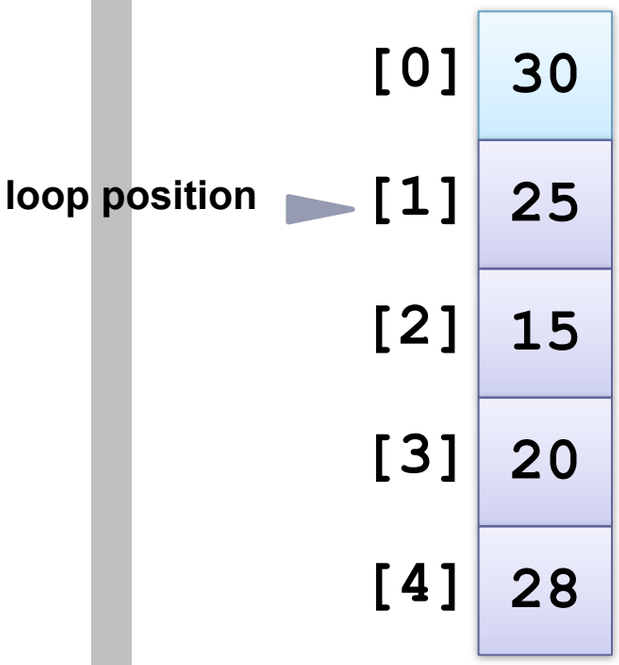
Trace of Insertion Sort Refinement (cont.)



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.)

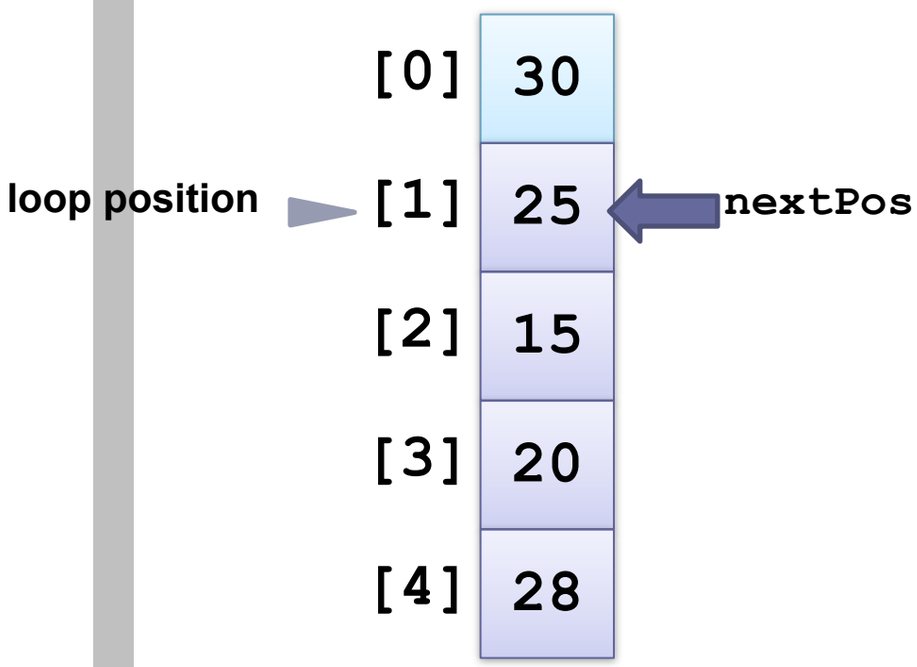
nextPos	1
nextVal	



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.)

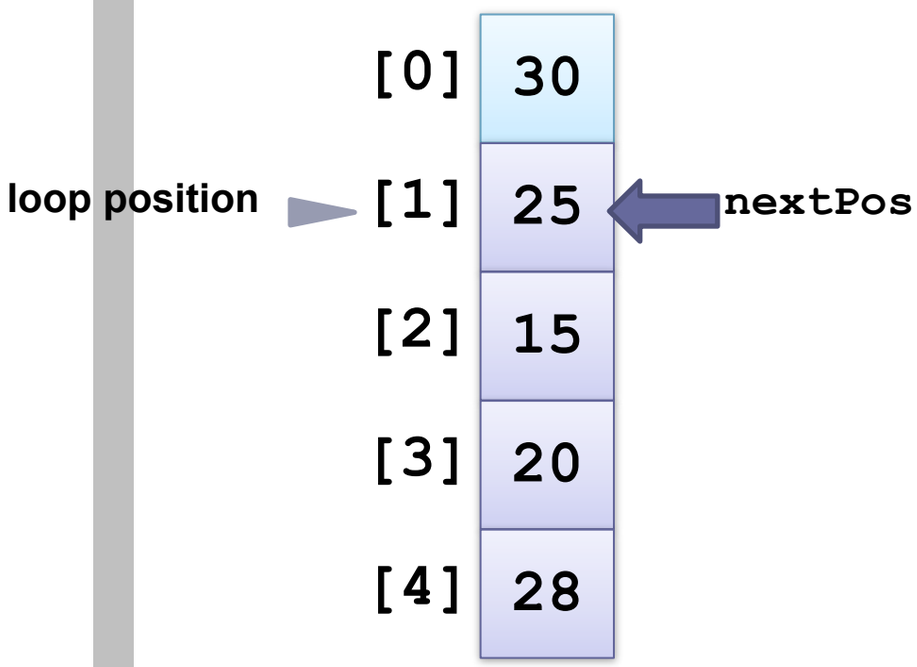
nextPos	1
nextVal	



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

Trace of Insertion Sort Refinement (cont.)

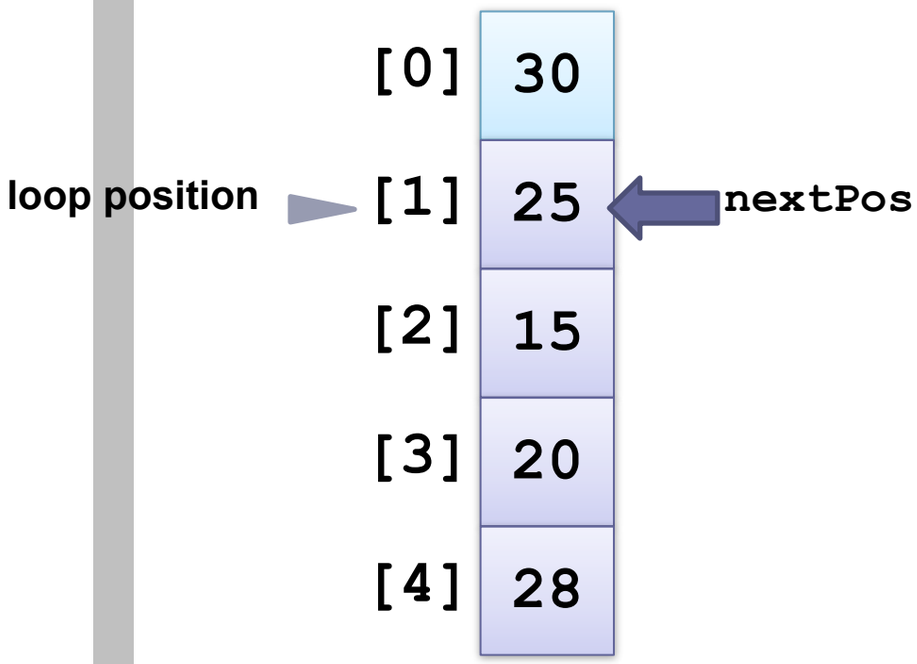
nextPos	1
nextVal	25



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

Trace of Insertion Sort Refinement (cont.)

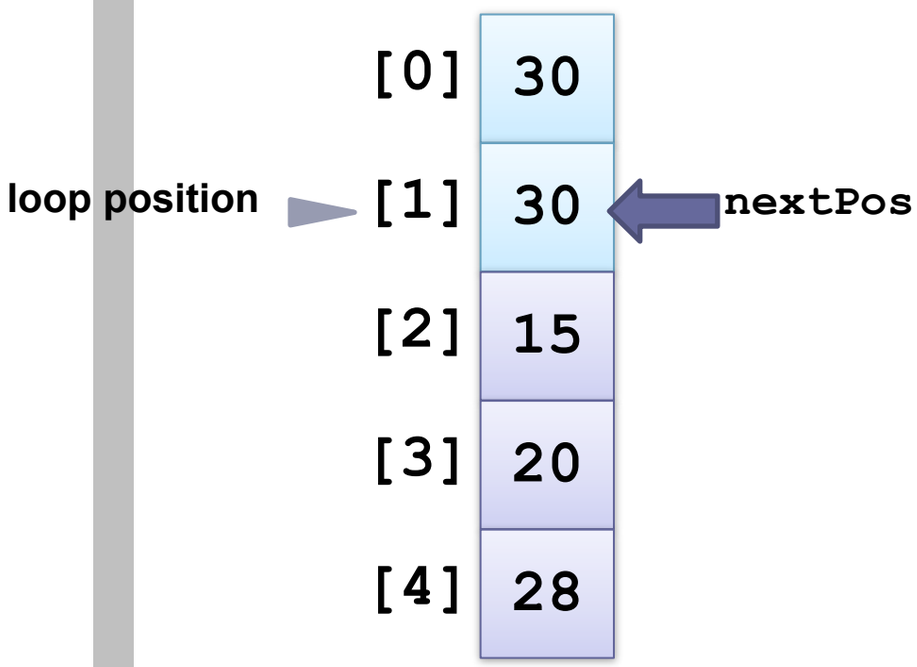
nextPos	1
nextVal	25



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

Trace of Insertion Sort Refinement (cont.)

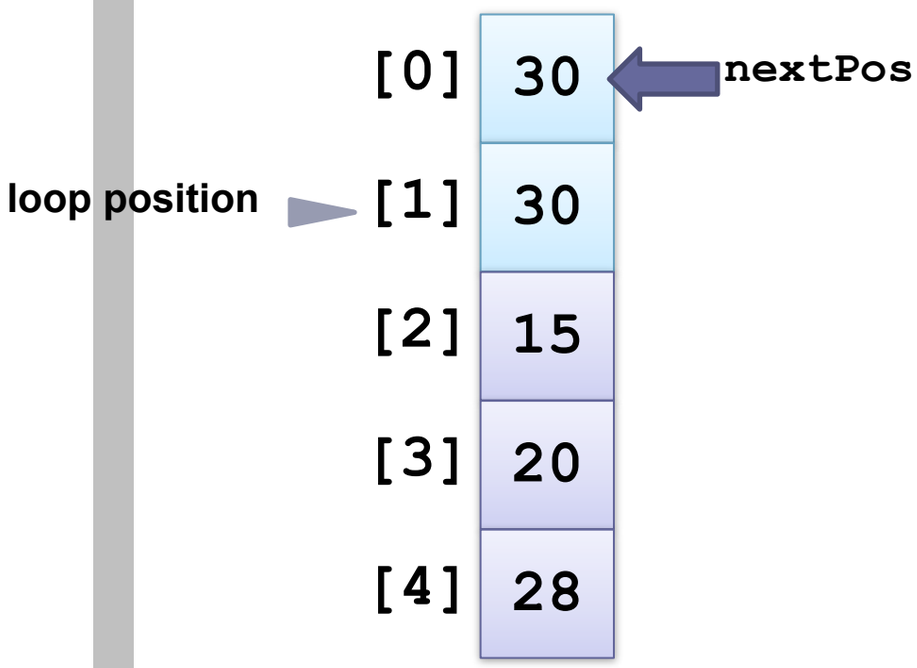
nextPos	1
nextVal	25



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

Trace of Insertion Sort Refinement (cont.)

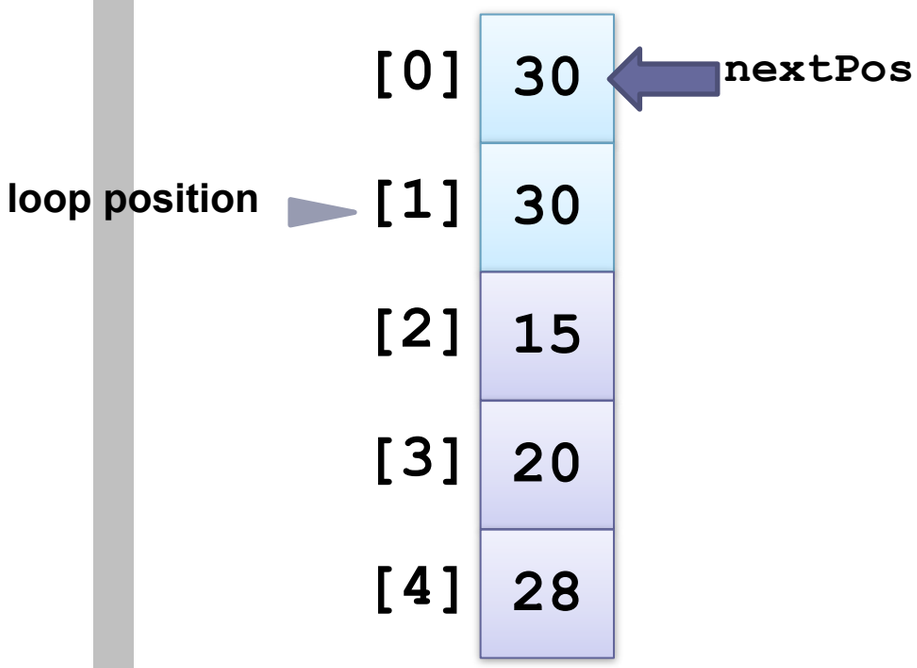
nextPos	0
nextVal	25



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

Trace of Insertion Sort Refinement (cont.)

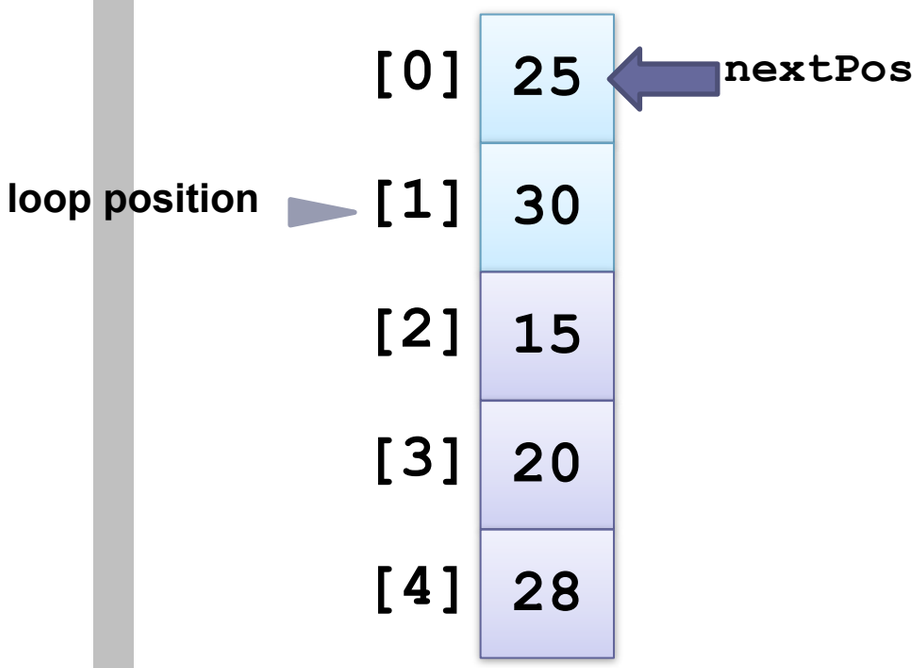
nextPos	0
nextVal	25



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

Trace of Insertion Sort Refinement (cont.)

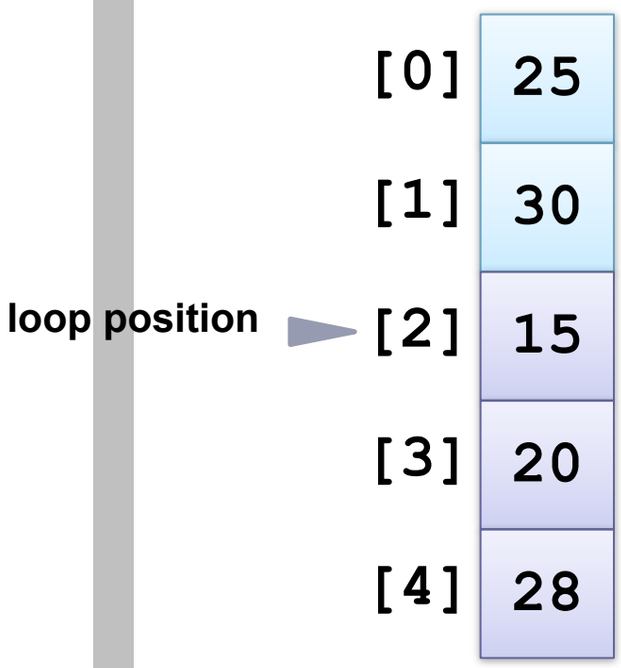
nextPos	0
nextVal	25



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

Trace of Insertion Sort Refinement (cont.)

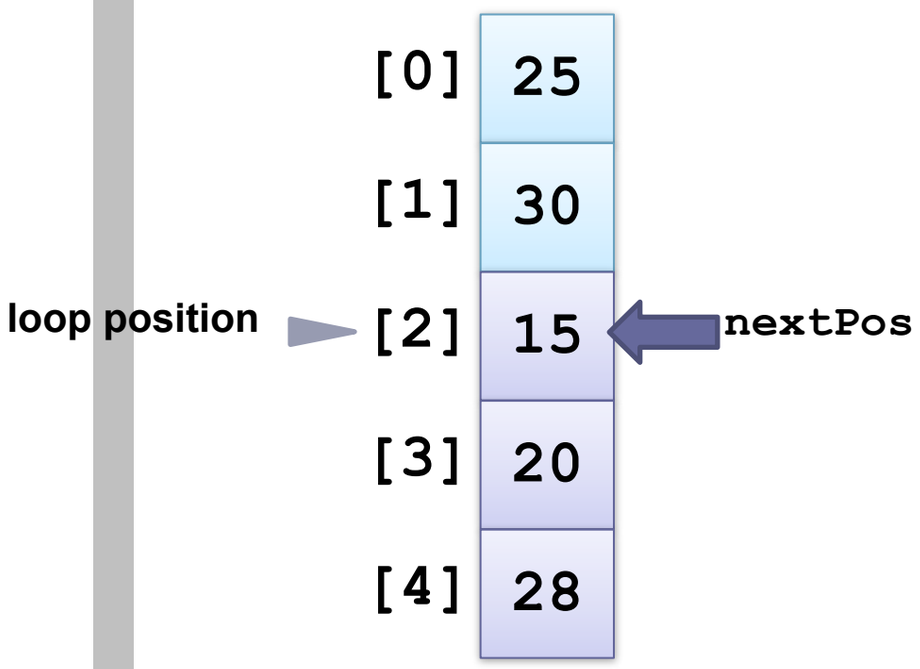
nextPos	0
nextVal	25



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.)

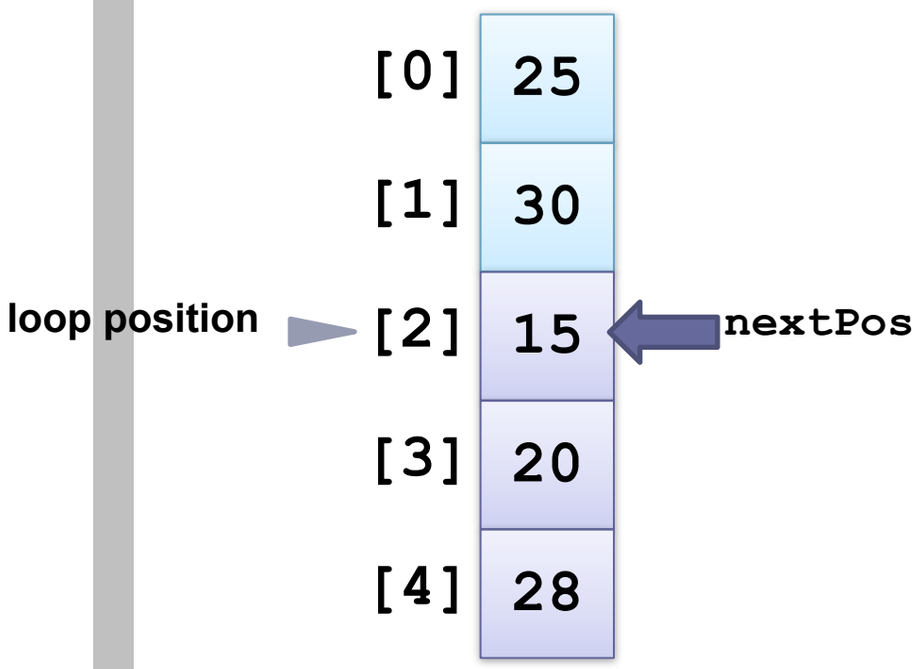
nextPos	2
nextVal	25



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.)

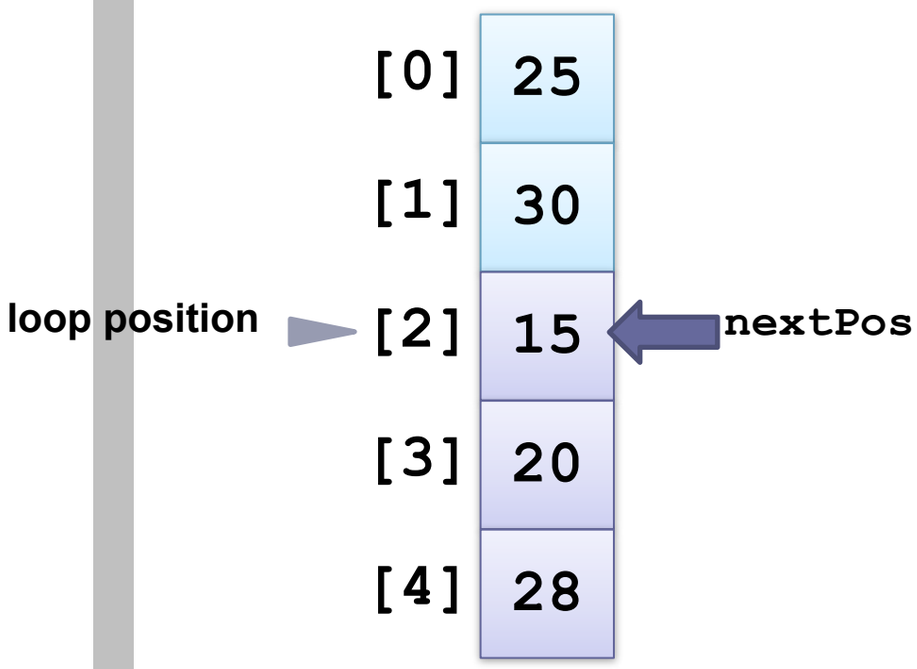
nextPos	2
nextVal	15



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.)

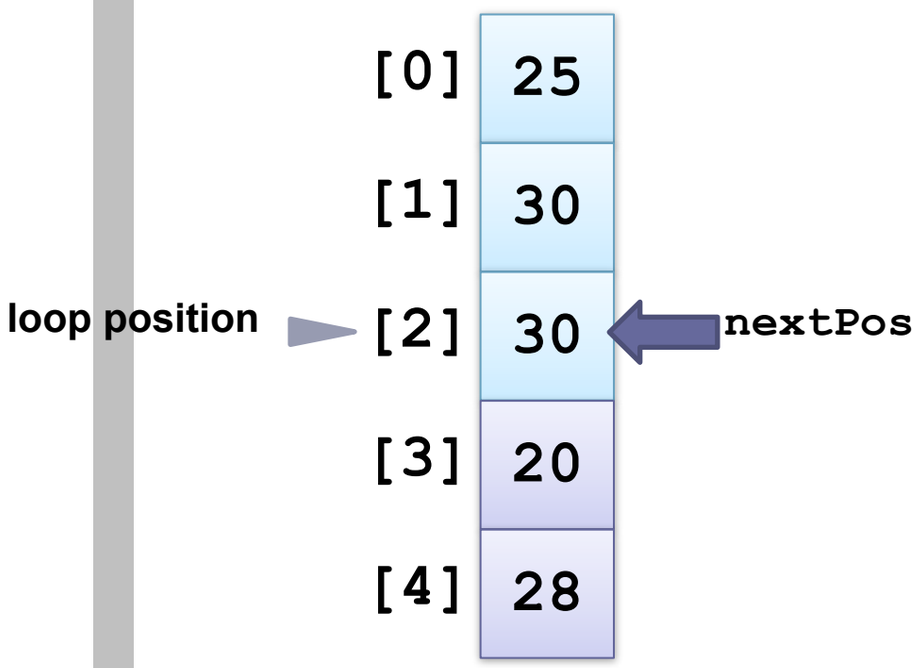
nextPos	2
nextVal	15



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.)

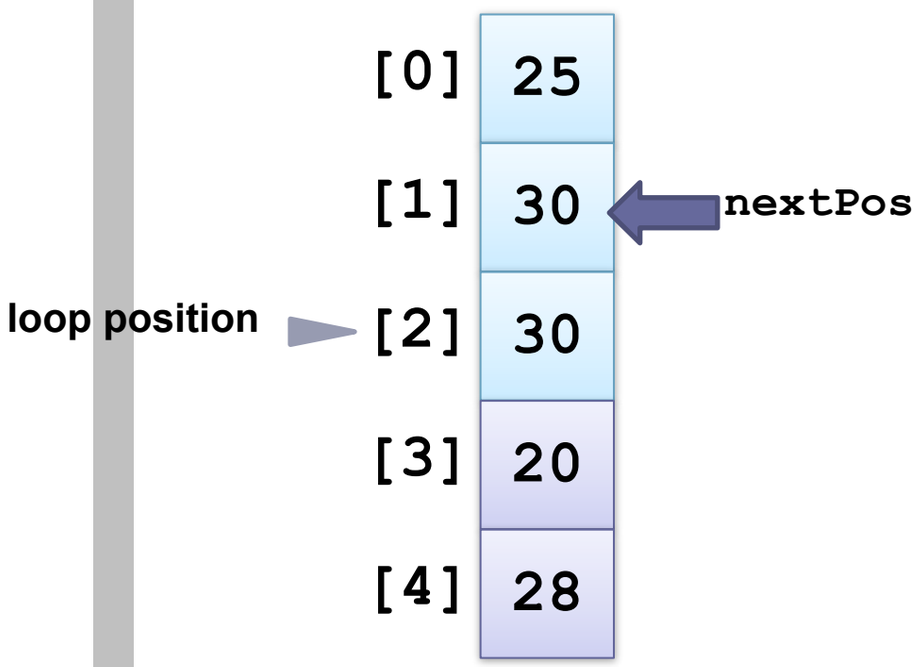
nextPos	2
nextVal	15



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.)

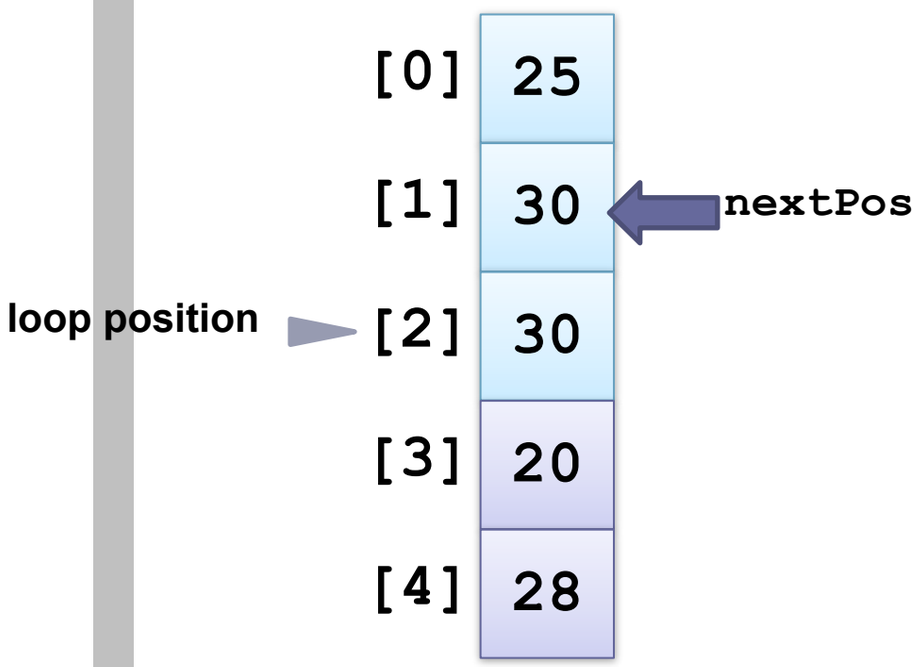
nextPos	1
nextVal	15



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.)

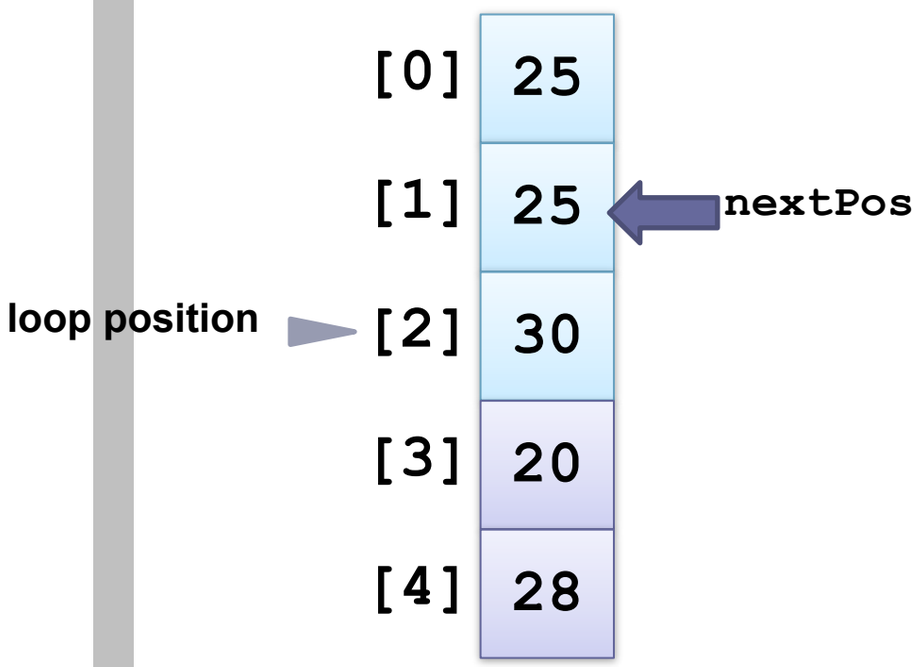
nextPos	1
nextVal	15



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.)

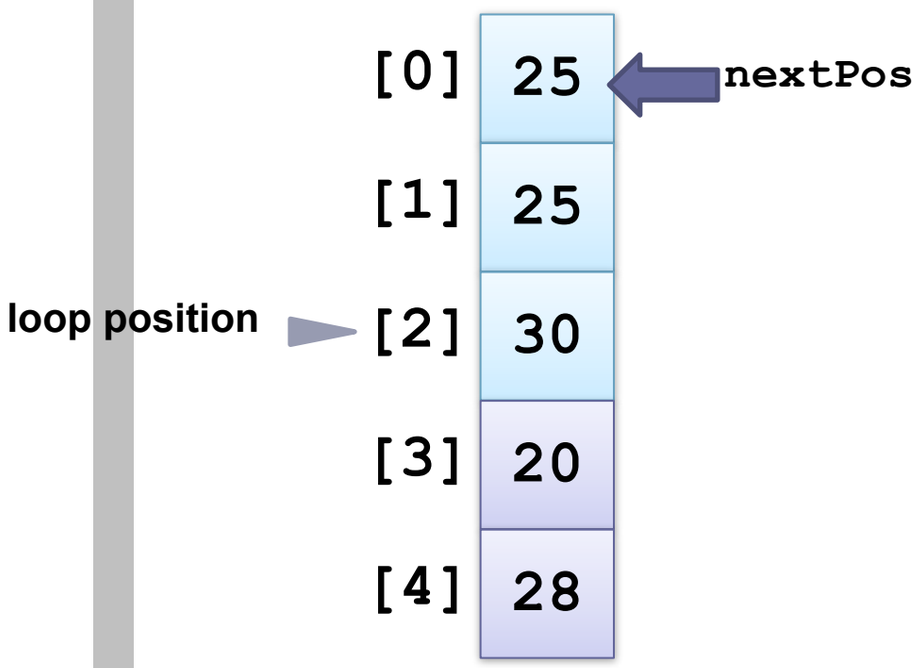
nextPos	1
nextVal	15



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.)

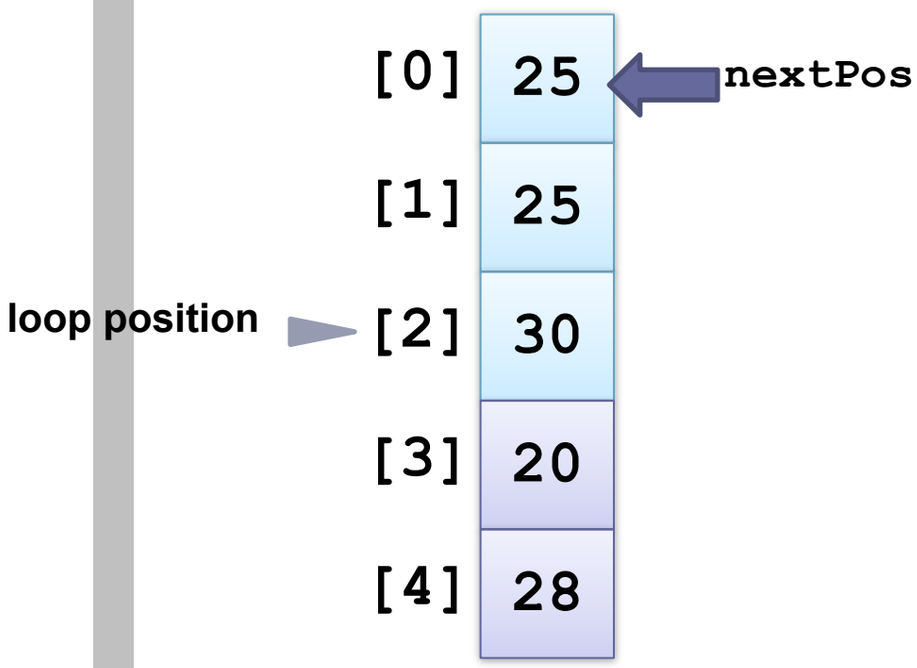
nextPos	0
nextVal	15



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.)

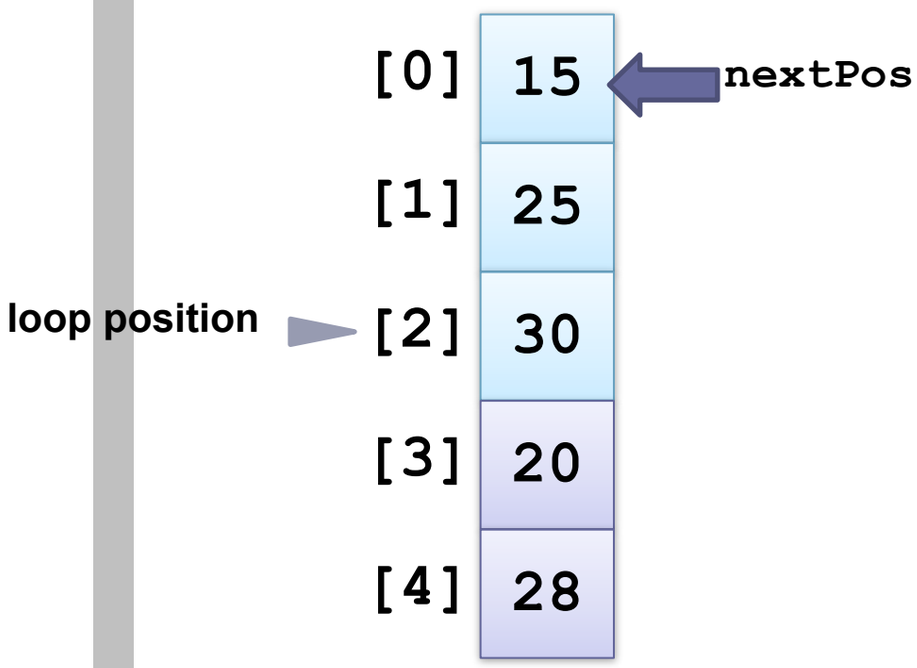
nextPos	0
nextVal	15



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.)

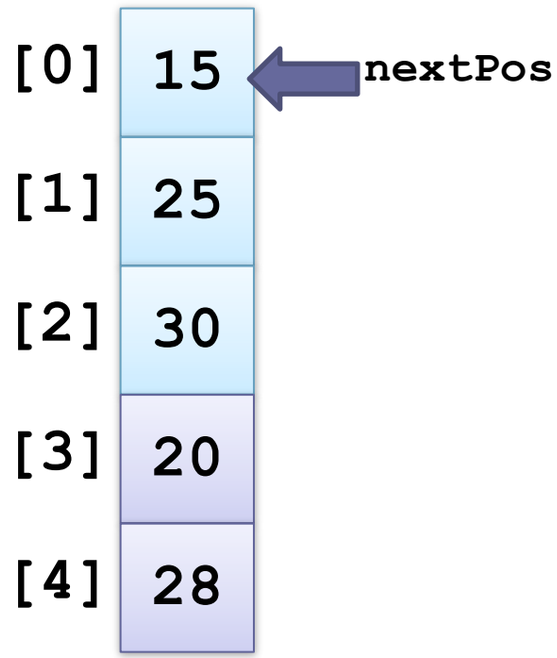
nextPos	0
nextVal	15



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.)

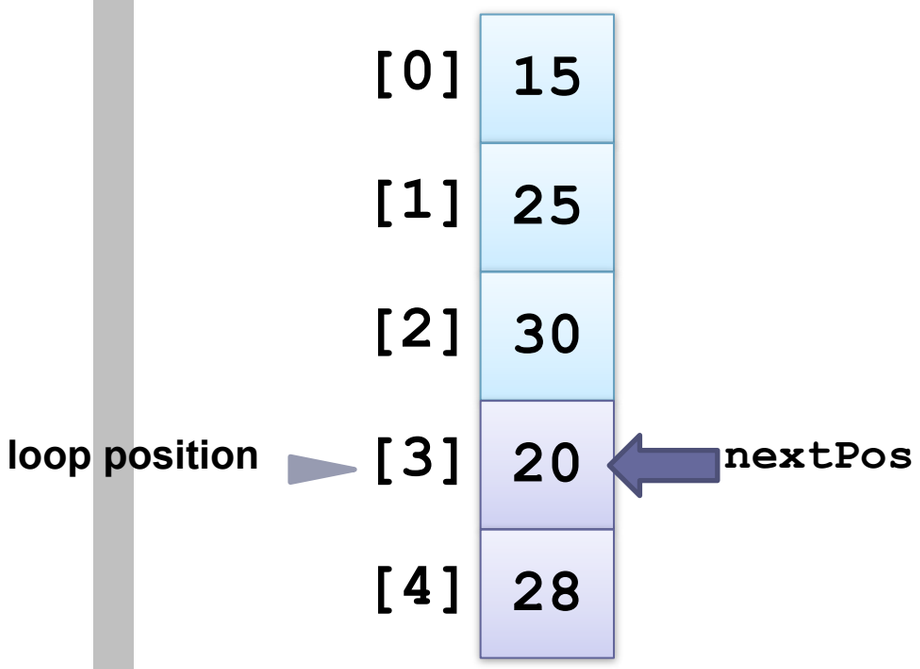
nextPos	0
nextVal	15



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.)

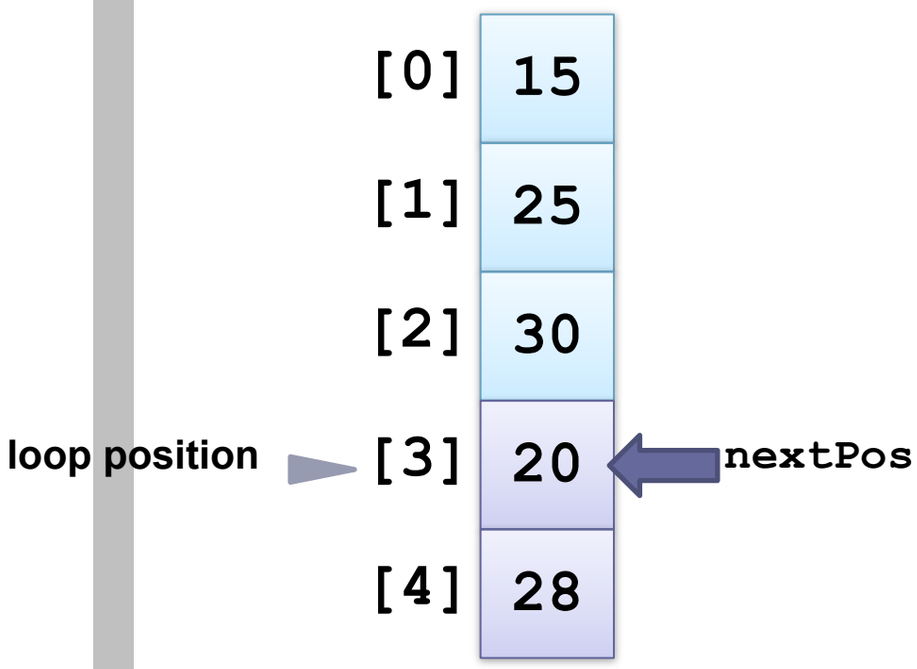
nextPos	3
nextVal	15



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.)

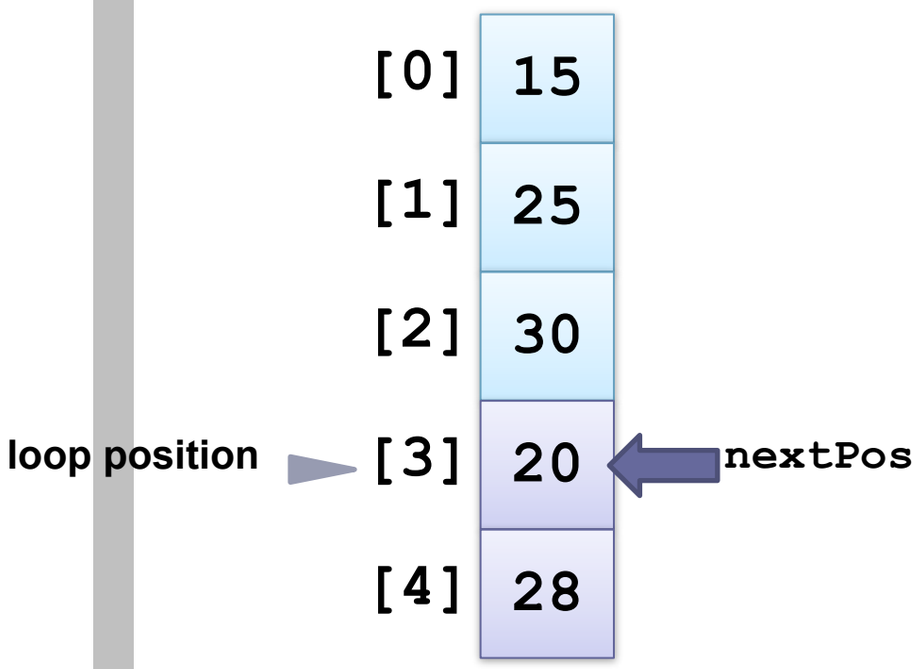
nextPos	3
nextVal	20



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.)

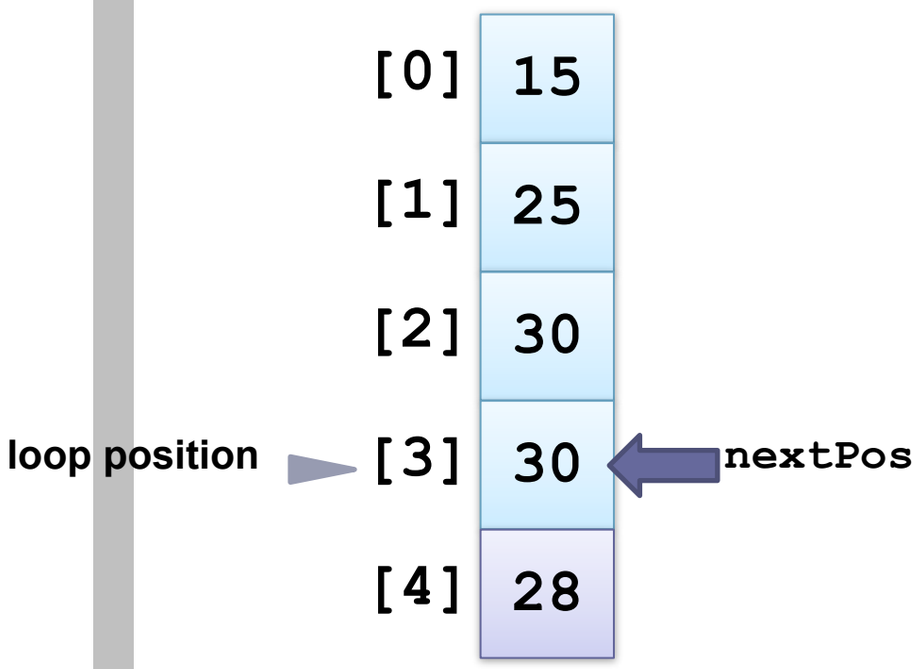
nextPos	3
nextVal	20



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.)

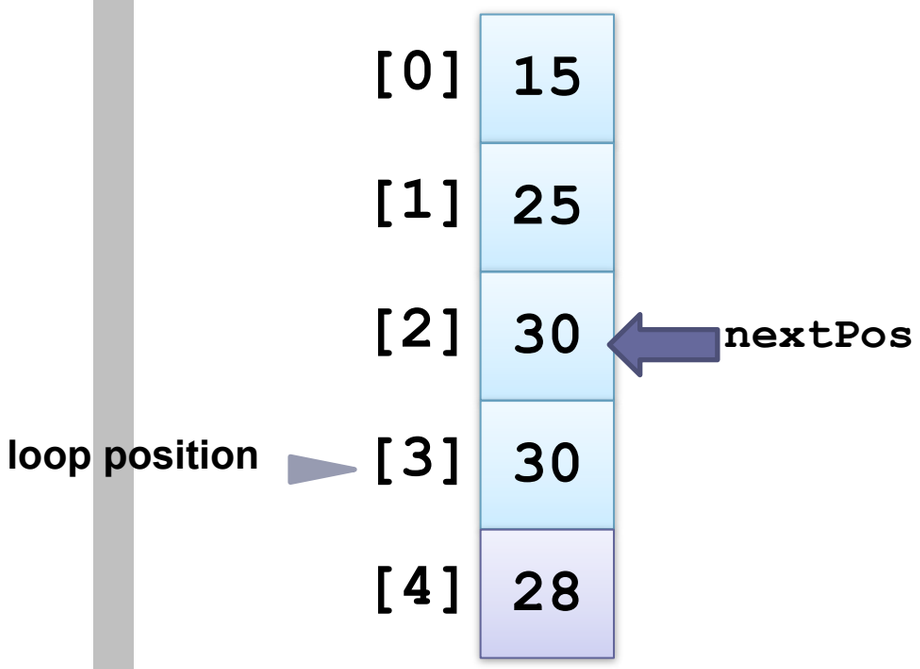
nextPos	3
nextVal	20



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.)

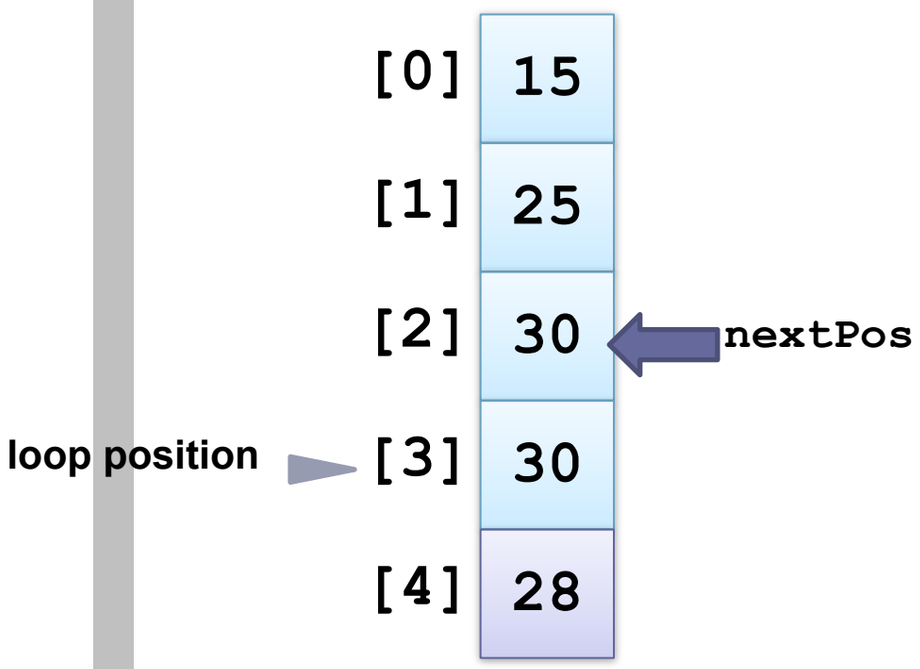
nextPos	2
nextVal	20



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.)

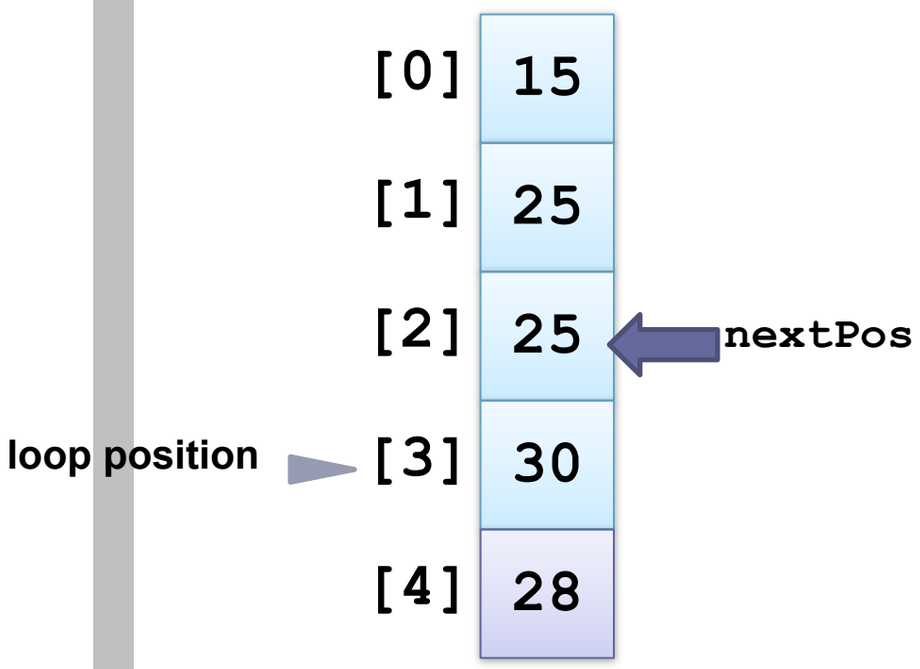
nextPos	2
nextVal	20



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.)

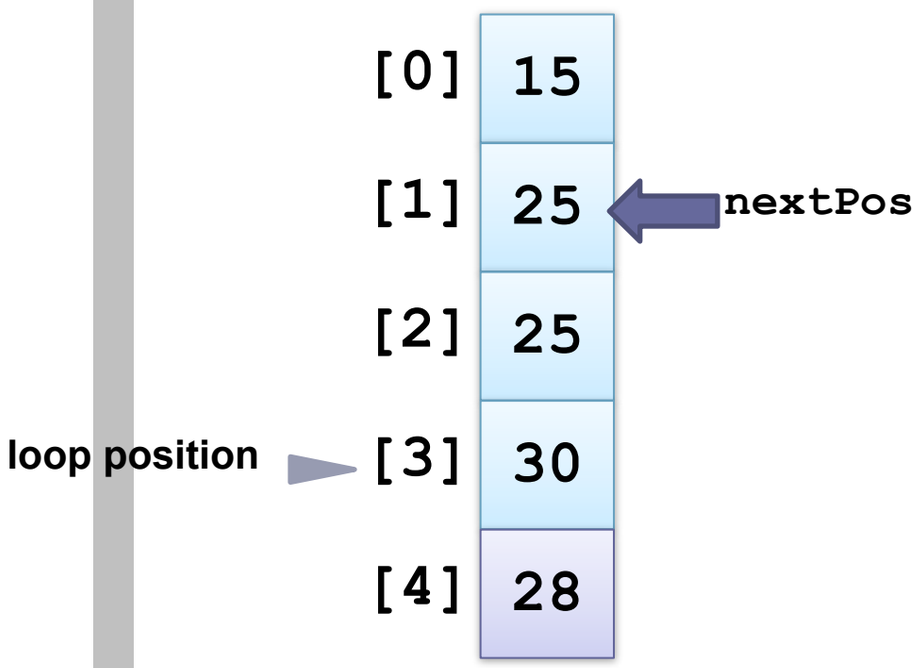
nextPos	2
nextVal	20



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.)

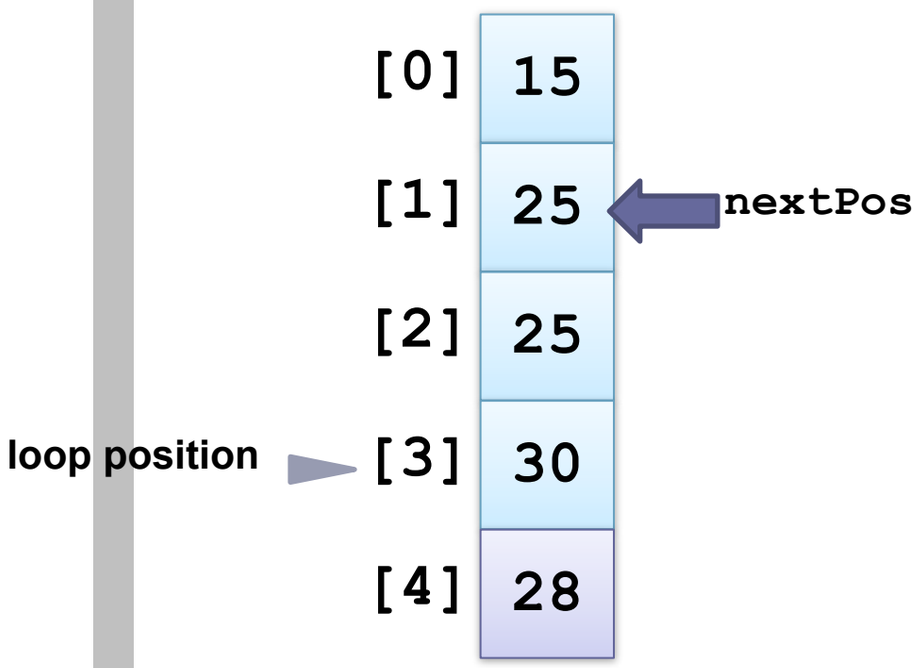
nextPos	1
nextVal	20



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.)

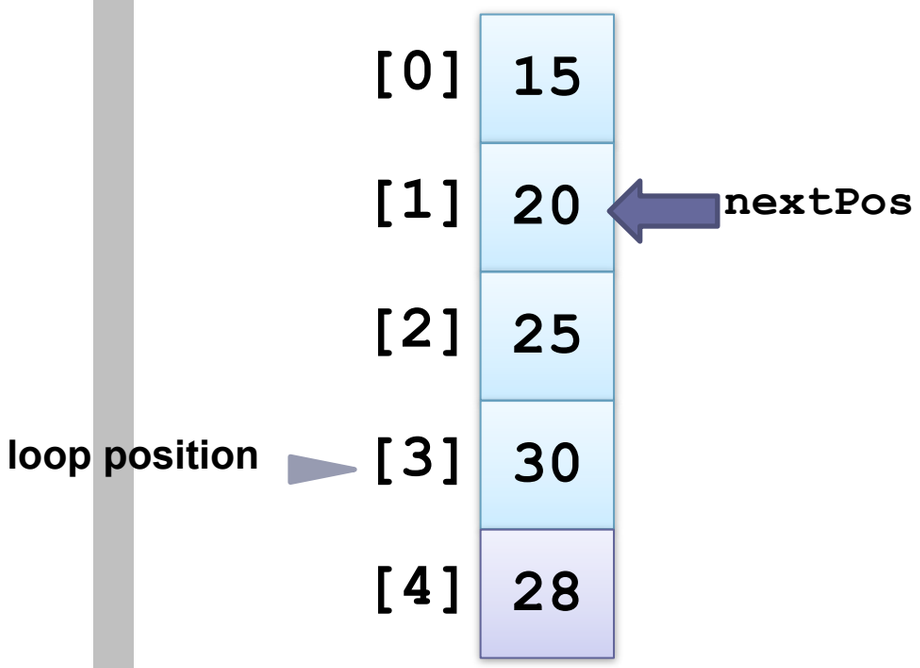
nextPos	1
nextVal	20



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

Trace of Insertion Sort Refinement (cont.)

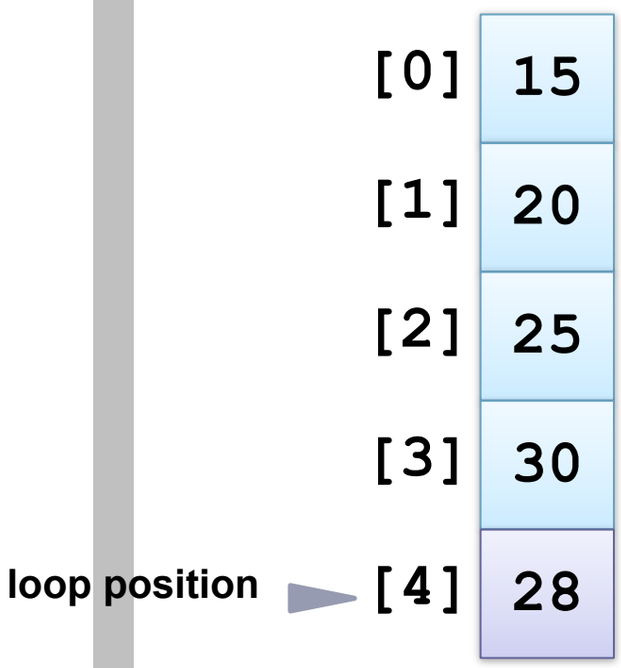
nextPos	1
nextVal	20



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

Trace of Insertion Sort Refinement (cont.)

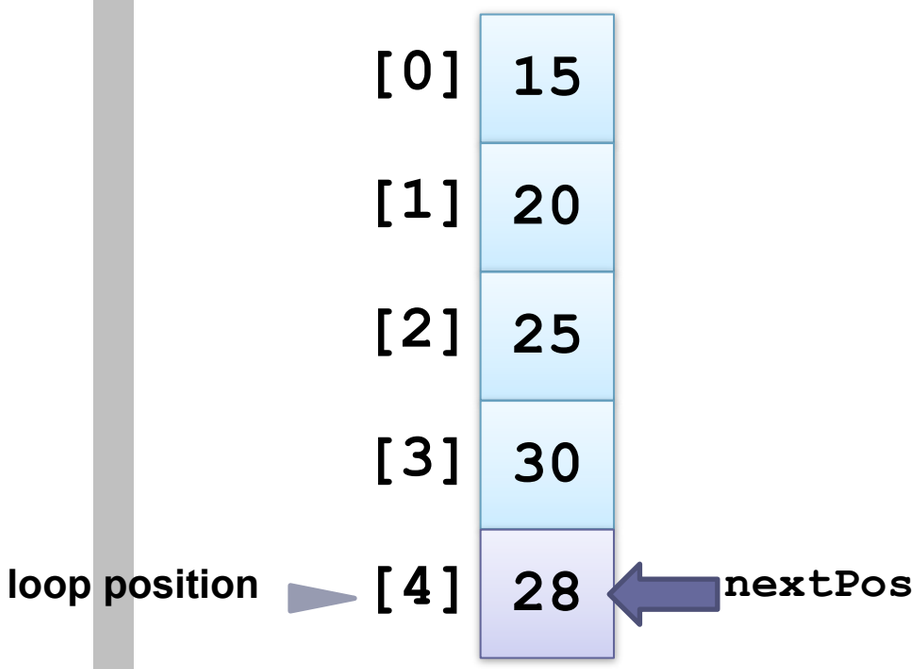
nextPos	1
nextVal	20



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

Trace of Insertion Sort Refinement (cont.)

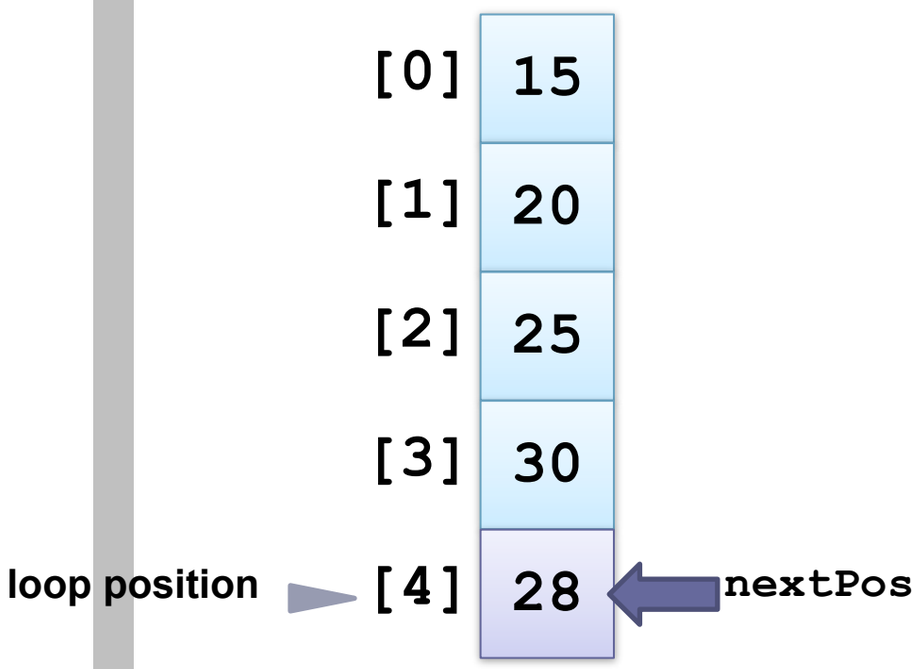
nextPos	4
nextVal	20



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

Trace of Insertion Sort Refinement (cont.)

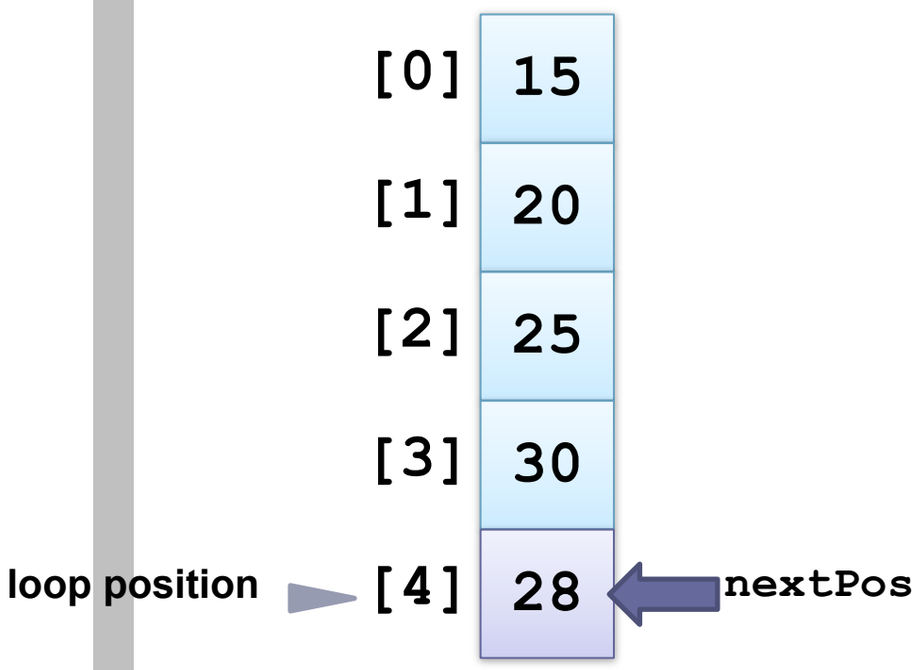
nextPos	4
nextVal	28



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.)

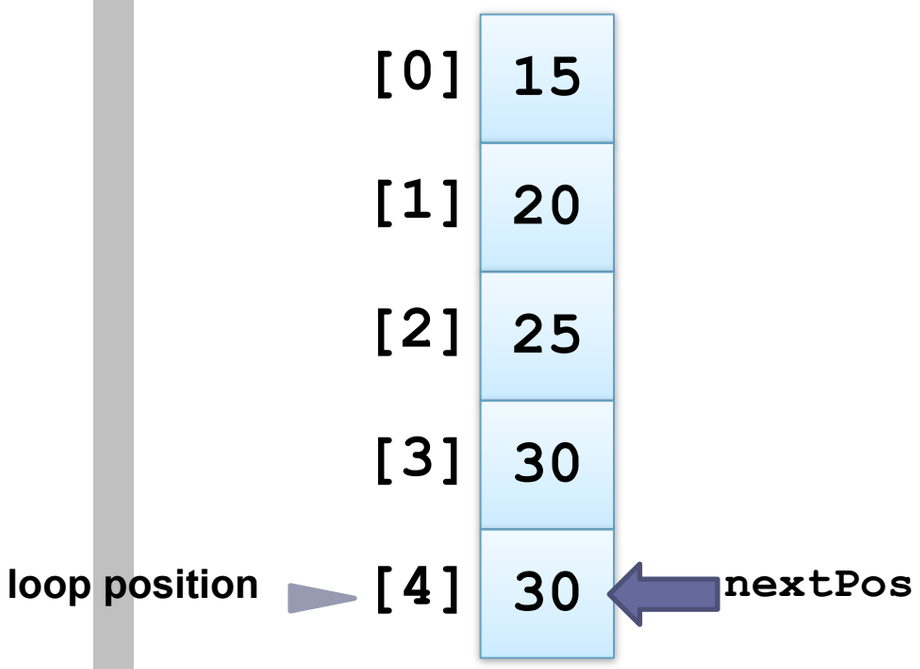
nextPos	4
nextVal	28



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.)

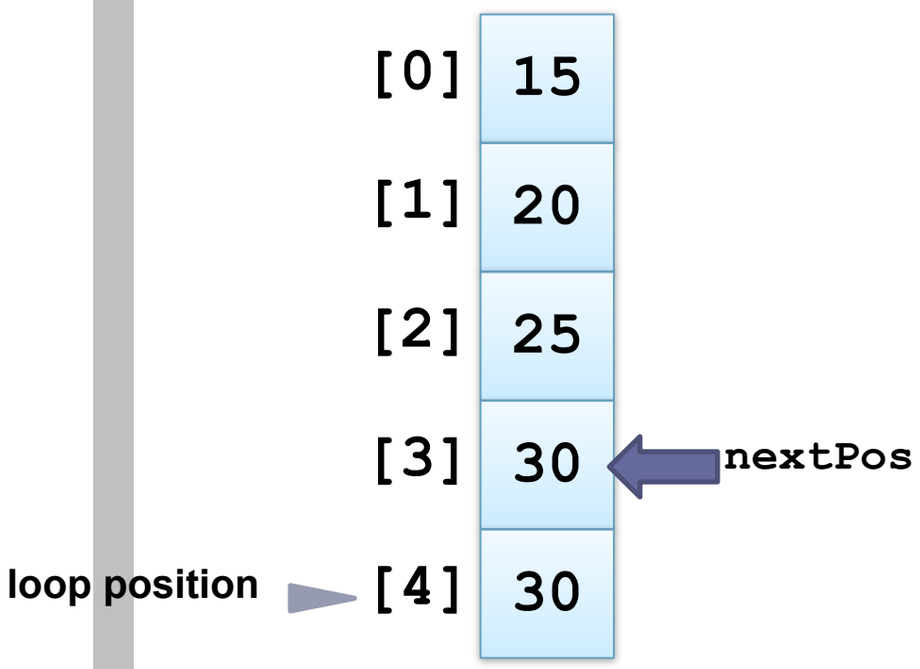
nextPos	4
nextVal	28



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.)

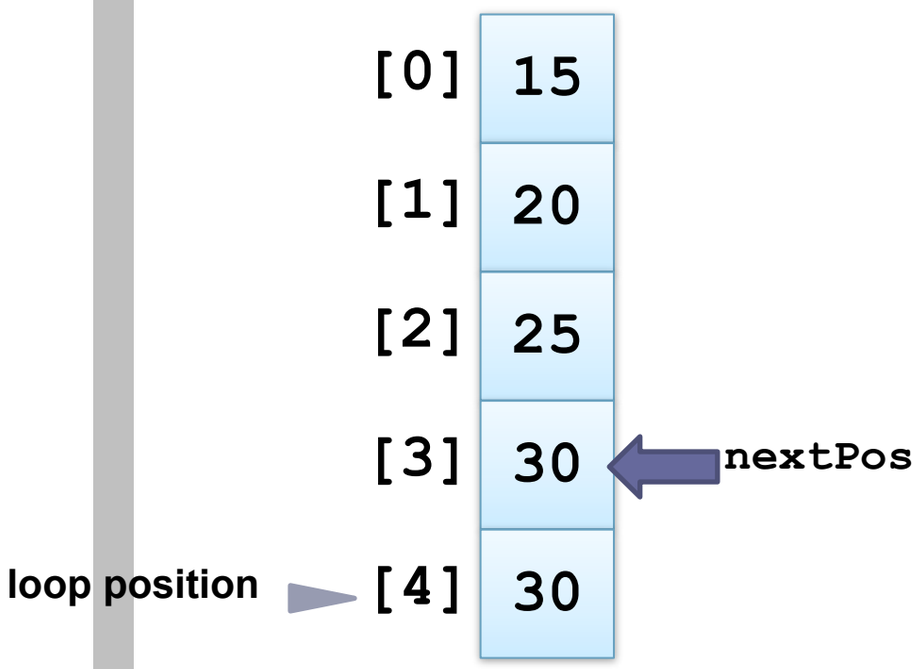
nextPos	3
nextVal	28



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.)

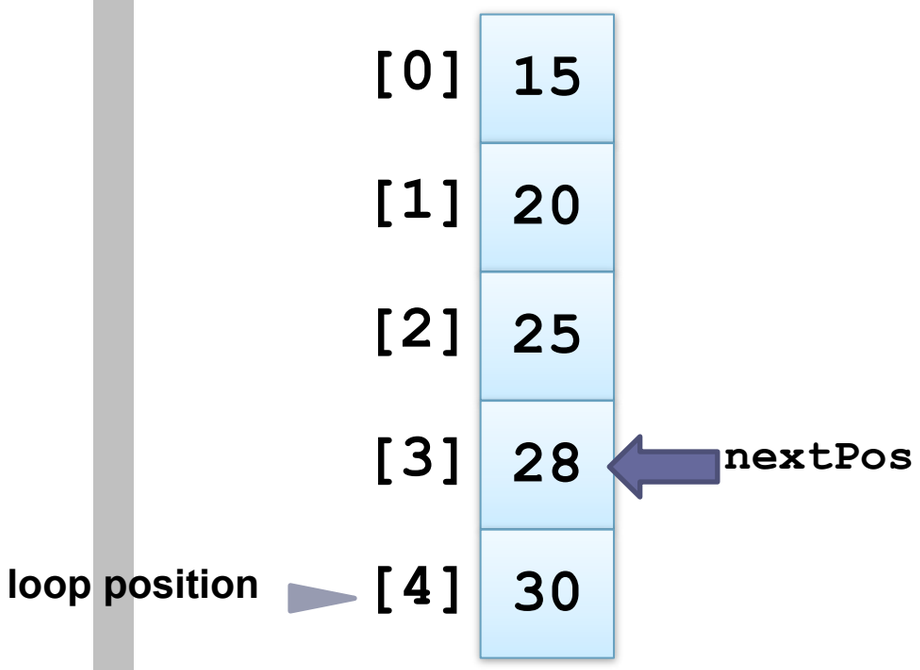
nextPos	3
nextVal	28



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`**

Komplexitetsanalys

Två nästade loopar som varje är O(n)

- komplexiteten är alltså kvadratisk, O(n^2)

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`**

Urvalssortering

Kursboken, avsnitt 8.2

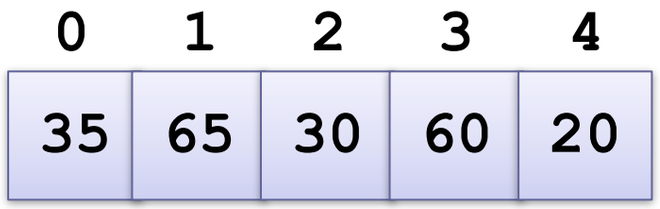
Algoritmen är väldigt lik insättningssortering, fast ”tvärtom”:

- leta upp det minsta elementet
- lägg till det i slutet av den nya listan
- fortsätt tills den gamla listan är tom

Urvalssortering, exempel

`n` = number of elements in the array

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

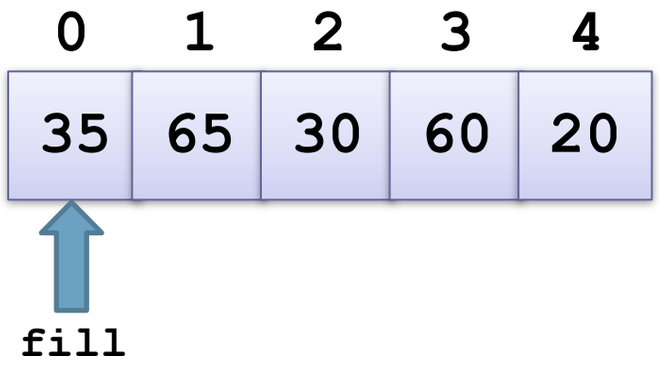


n	5
fill	
posMin	

Urvalssortering, exempel

`n` = number of elements in the array

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

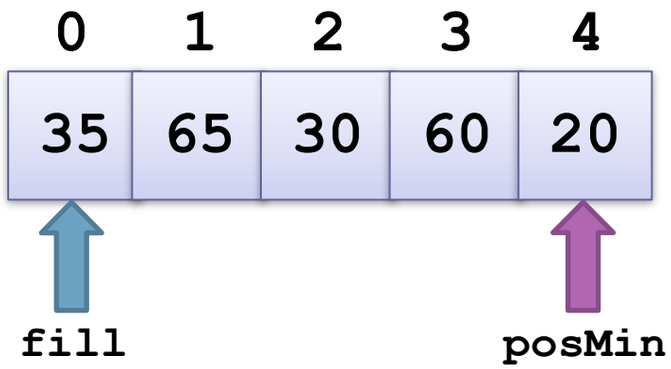


n	5
fill	0
posMin	

Urvalssortering, exempel

`n` = number of elements in the array

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

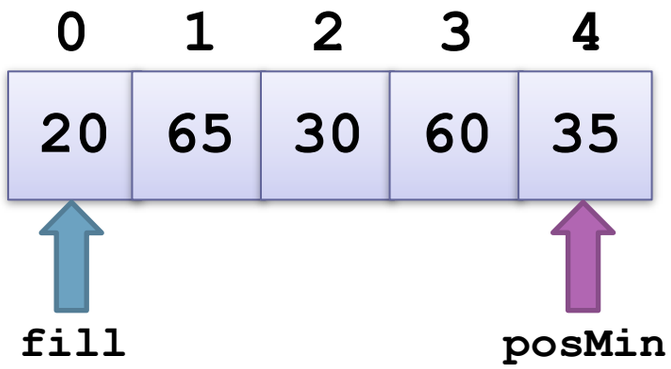


n	5
fill	0
posMin	4

Urvalssortering, exempel

`n` = number of elements in the array

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

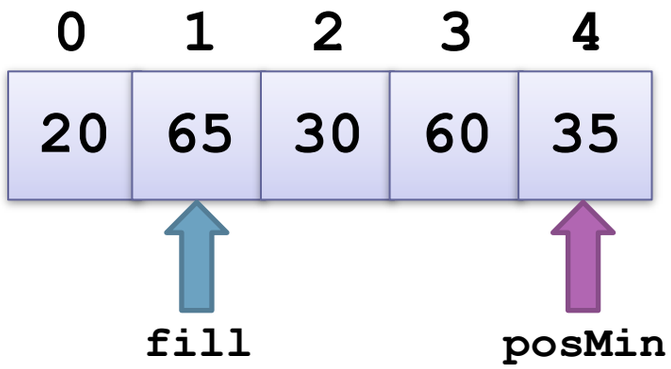


n	5
fill	0
posMin	4

Urvalssortering, exempel

`n` = number of elements in the array

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

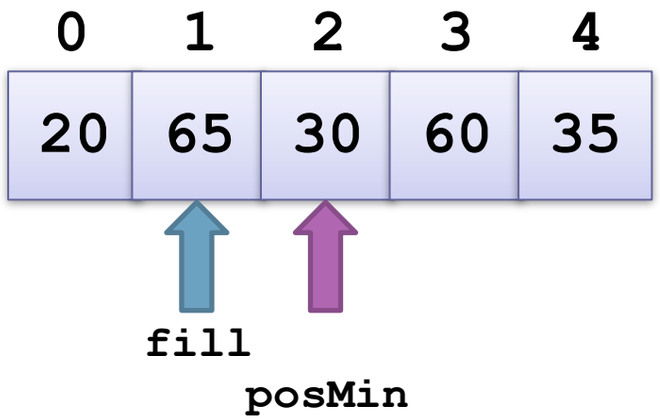


n	5
fill	1
posMin	4

Urvalssortering, exempel

`n` = number of elements in the array

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

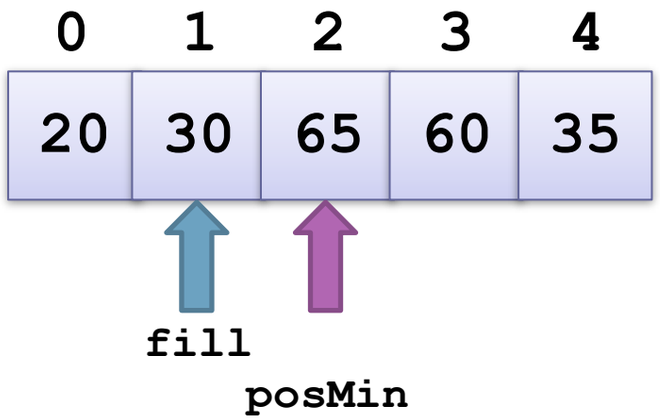


n	5
fill	1
posMin	2

Urvalssortering, exempel

`n` = number of elements in the array

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

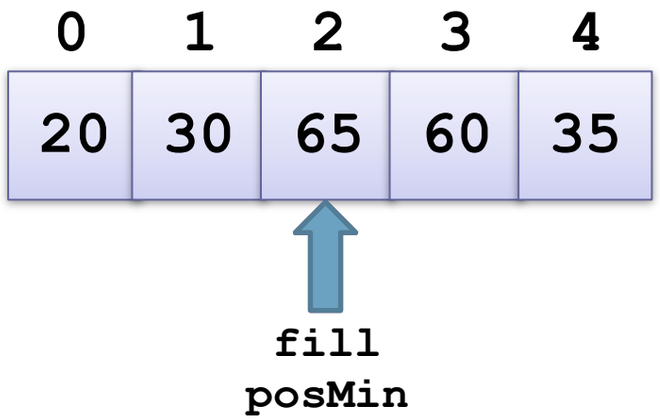


n	5
fill	1
posMin	2

Urvalssortering, exempel

`n` = number of elements in the array

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

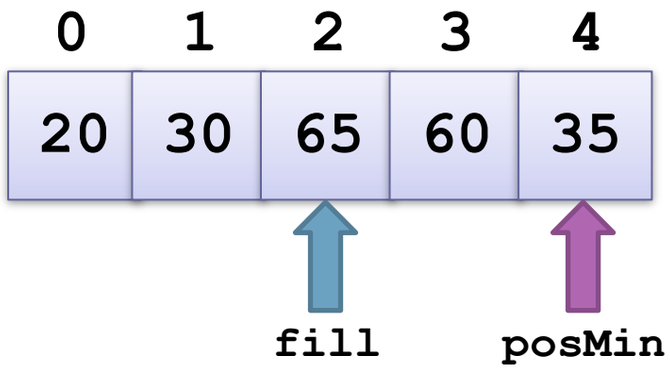


n	5
fill	2
posMin	2

Urvalssortering, exempel

`n` = number of elements in the array

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

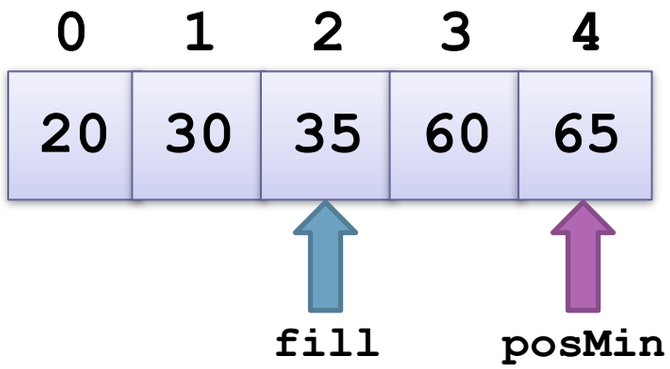


n	5
fill	2
posMin	4

Urvalssortering, exempel

`n` = number of elements in the array

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

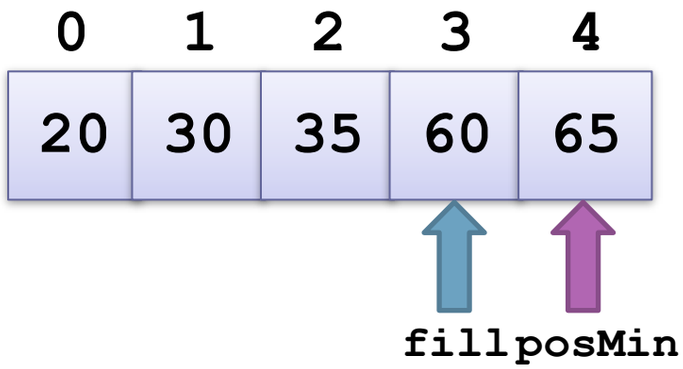


n	5
fill	2
posMin	4

Urvalssortering, exempel

`n` = number of elements in the array

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

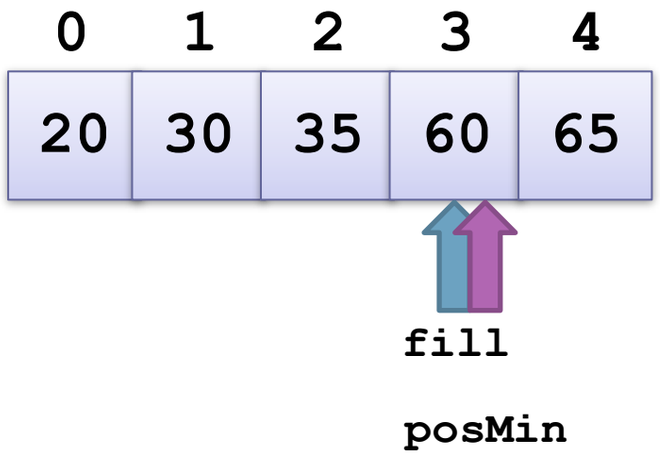


n	5
fill	3
posMin	4

Urvalssortering, exempel

`n` = number of elements in the array

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

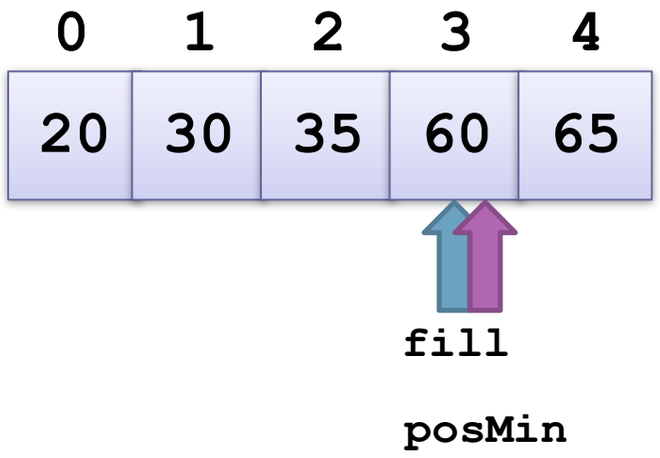


n	5
fill	3
posMin	3

Urvalssortering, exempel

`n` = number of elements in the array

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

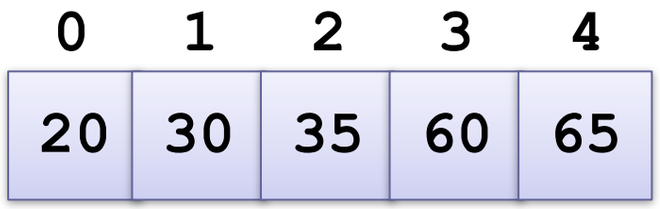


n	5
fill	3
posMin	3

Urvalssortering, exempel

`n` = number of elements in the array

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



n	5
fill	3
posMin	3

Samma exempel, förfinad version

Denhär raden:

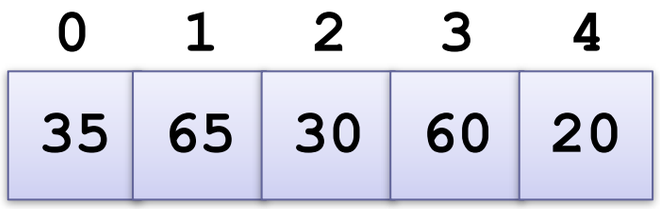
> Set posMin to the subscript of a smallest item
> in the subarray starting at subscript fill

kan förfinas till en for-loop:

```
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
```

Trace of Selection Sort Refinement (cont.)

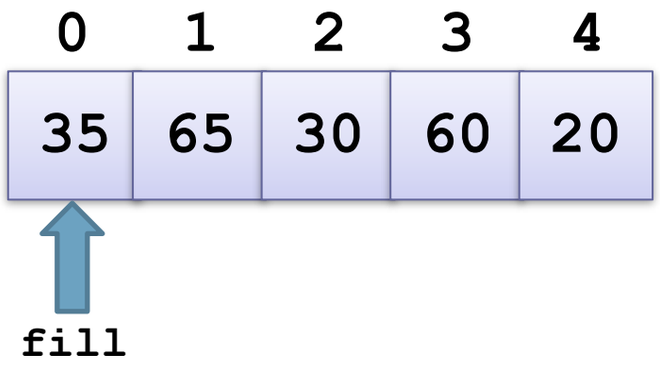
n	5
fill	
posMin	
next	



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	
next	

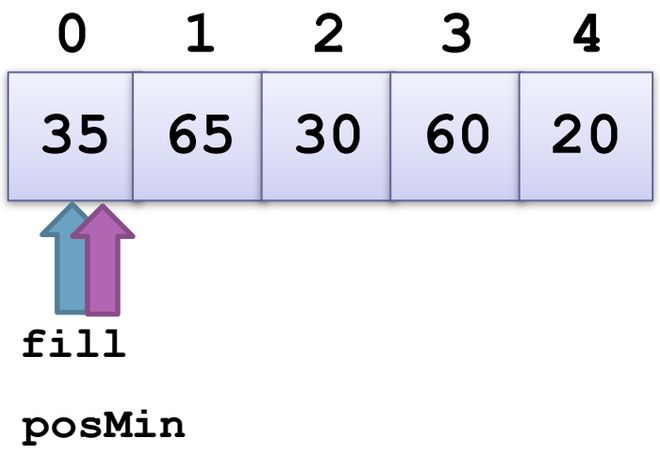


1. `for fill = 0 to n - 2 do`
2. Initialize `posMin` to `fill`
3. `for next = fill + 1 to n - 1 do`
4. `if` the item at `next` is less than the item at `posMin`
5. Reset `posMin` to `next`
6. Exchange the item at `posMin` with the one at `fill`

Trace of Selection Sort Refinement (cont.)

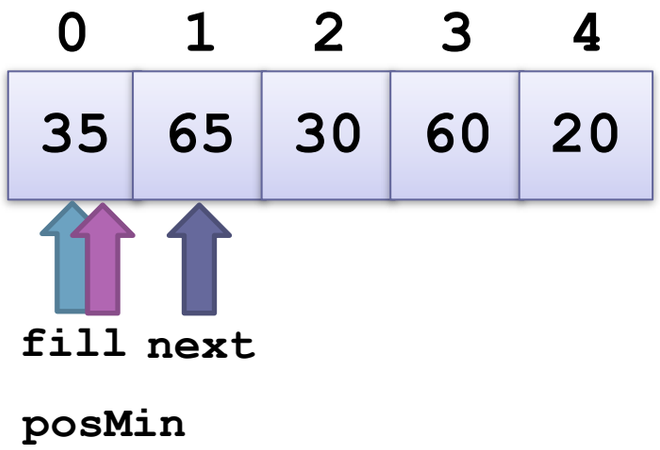
n	5
fill	0
posMin	0
next	

1. 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.)

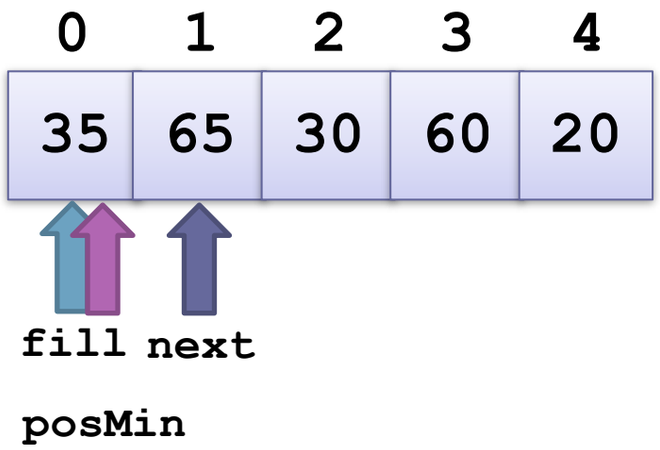
n	5
fill	0
posMin	0
next	1



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`

Trace of Selection Sort Refinement (cont.)

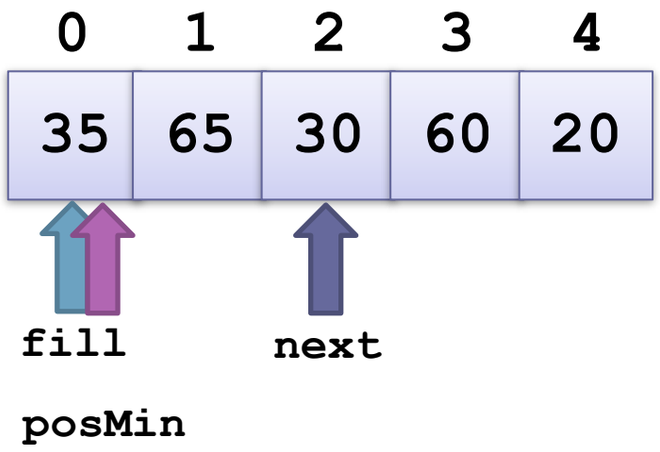
n	5
fill	0
posMin	0
next	1



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`

Trace of Selection Sort Refinement (cont.)

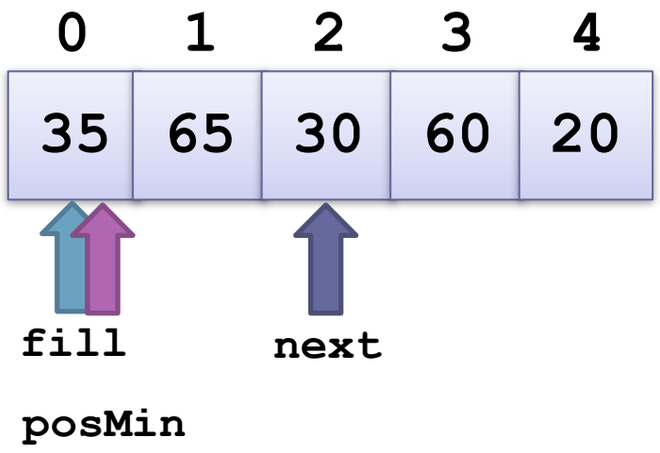
n	5
fill	0
posMin	0
next	2



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

Trace of Selection Sort Refinement (cont.)

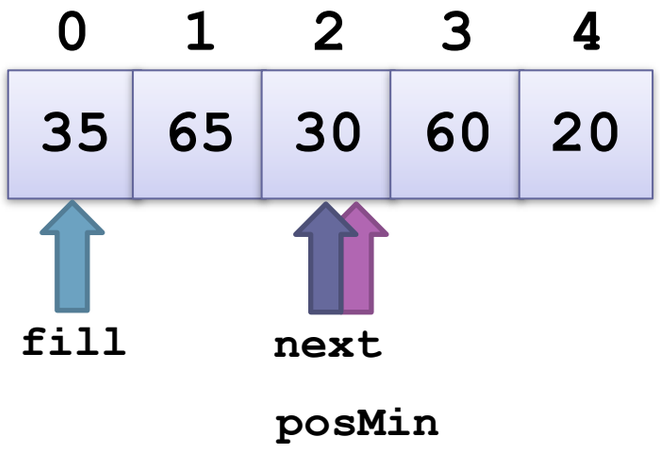
n	5
fill	0
posMin	0
next	2



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

Trace of Selection Sort Refinement (cont.)

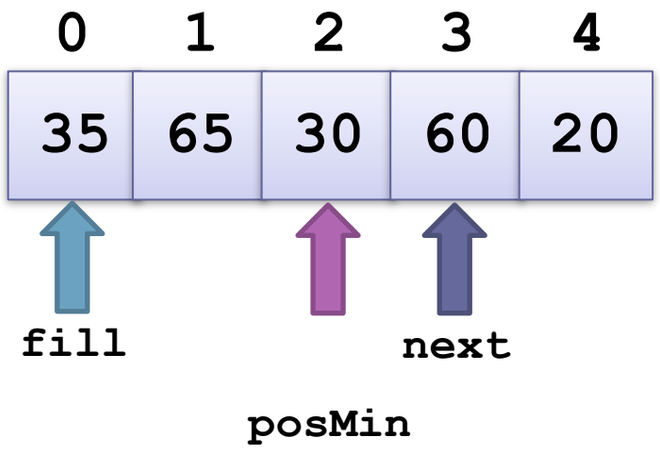
n	5
fill	0
posMin	2
next	2



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.)

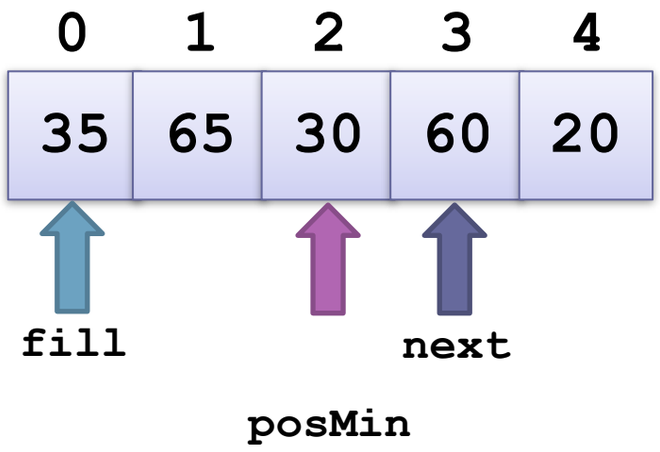
n	5
fill	0
posMin	2
next	3



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.)

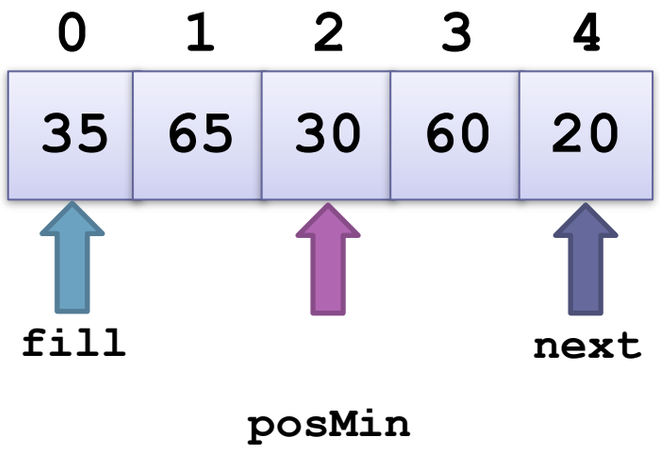
n	5
fill	0
posMin	2
next	3



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.)

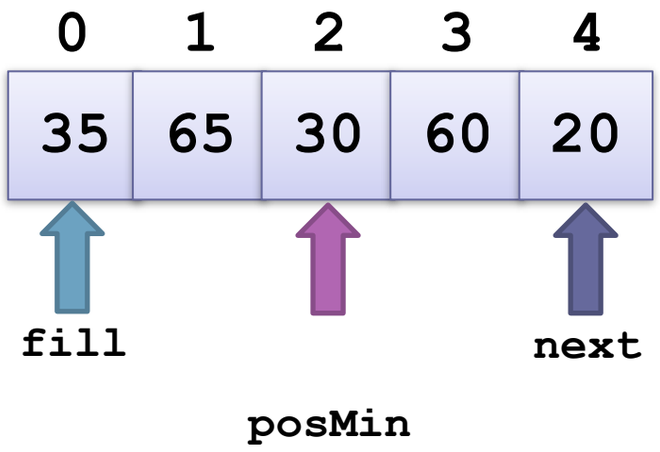
n	5
fill	0
posMin	2
next	4



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.)

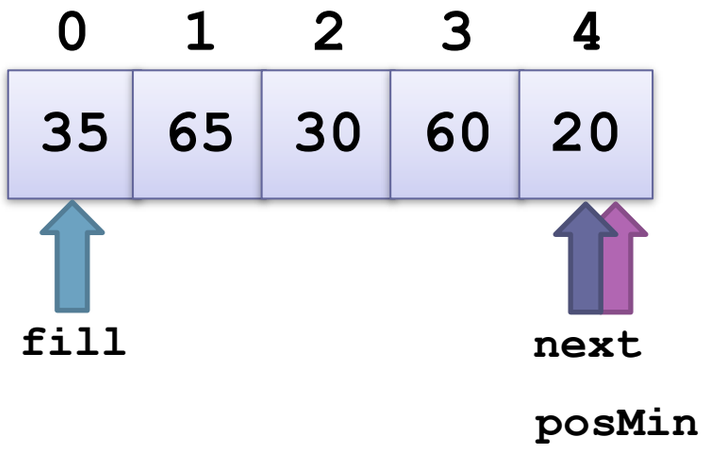
n	5
fill	0
posMin	2
next	4



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.)

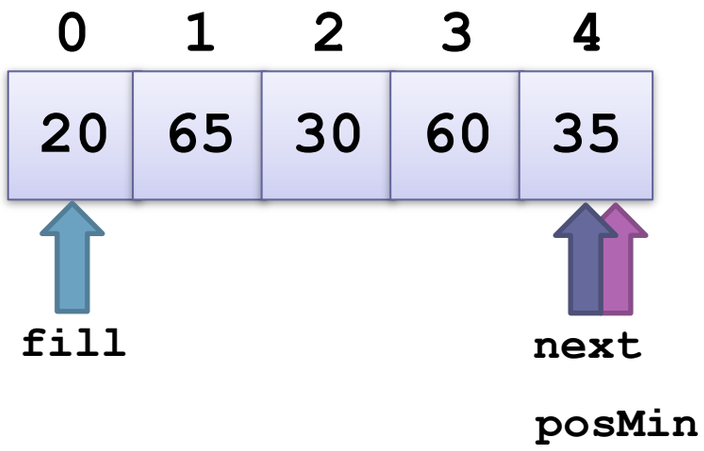
n	5
fill	0
posMin	4
next	4



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.)

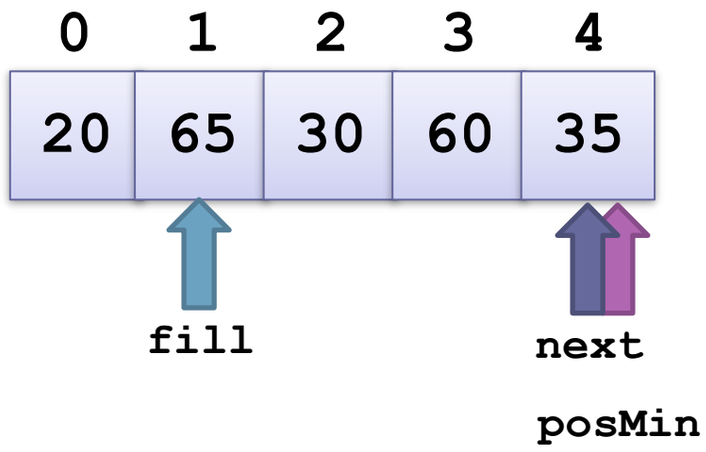
n	5
fill	0
posMin	4
next	4



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

Trace of Selection Sort Refinement (cont.)

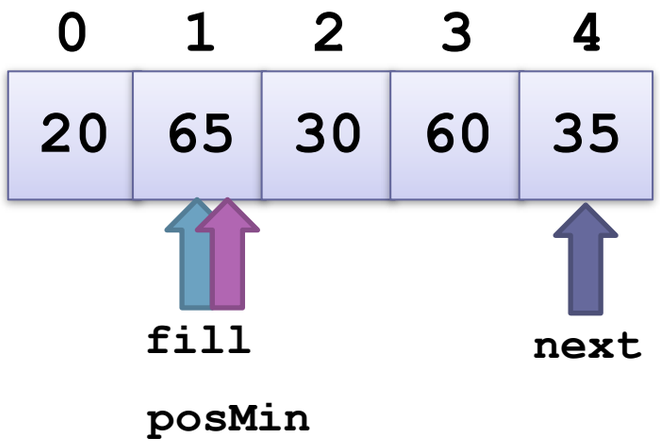
n	5
fill	1
posMin	4
next	4



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

Trace of Selection Sort Refinement (cont.)

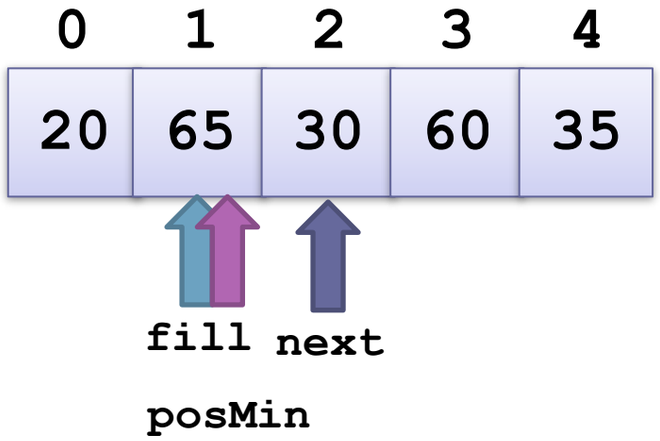
n	5
fill	1
posMin	1
next	4



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

Trace of Selection Sort Refinement (cont.)

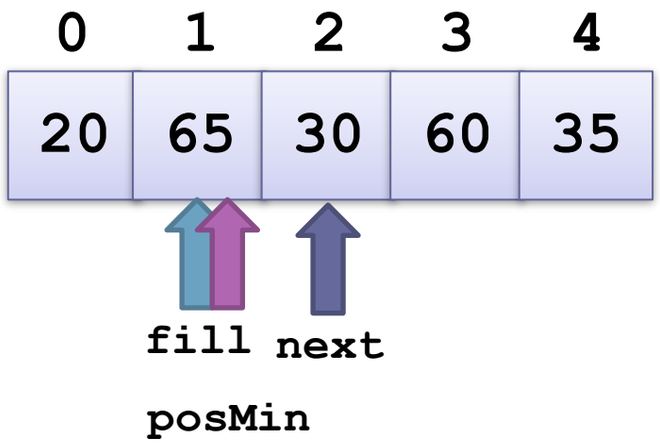
n	5
fill	1
posMin	1
next	2



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.)

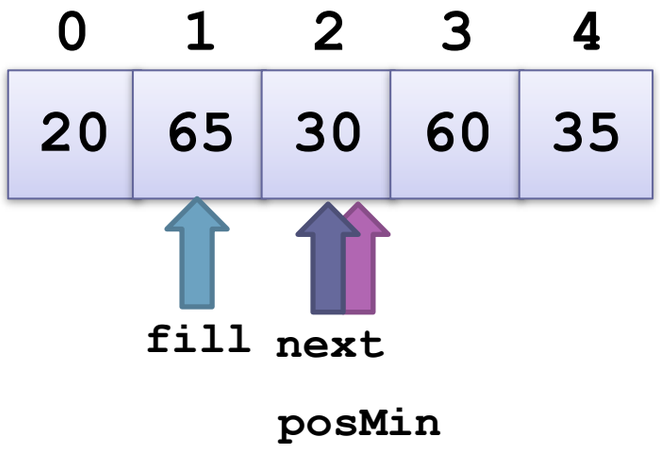
n	5
fill	1
posMin	1
next	2



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.)

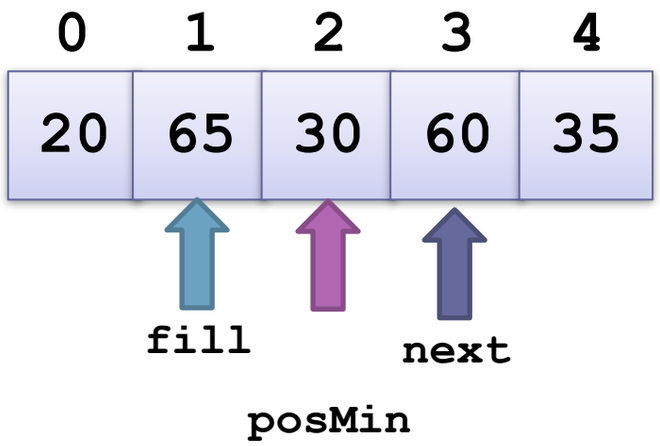
n	5
fill	1
posMin	2
next	2



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.)

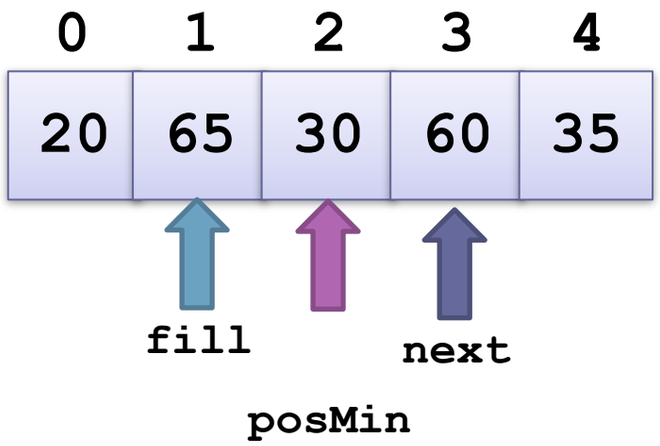
n	5
fill	1
posMin	2
next	3



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.)

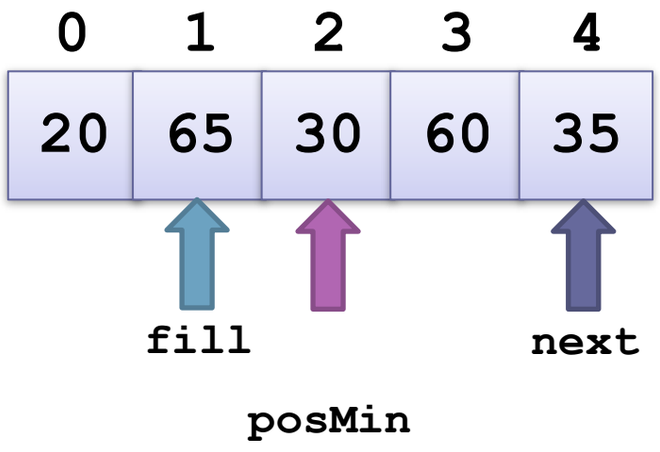
n	5
fill	1
posMin	2
next	3



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

Trace of Selection Sort Refinement (cont.)

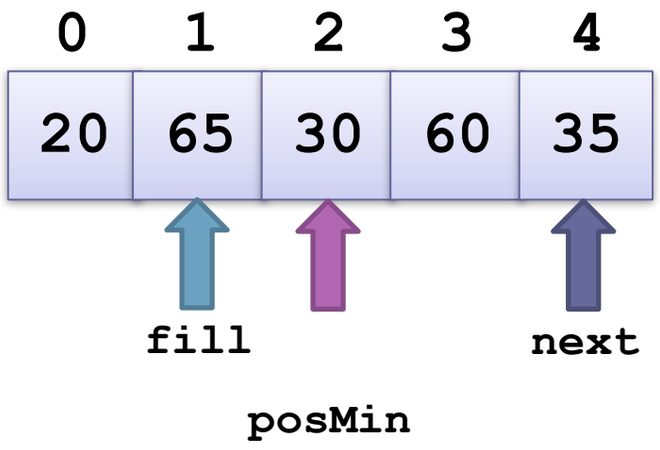
n	5
fill	1
posMin	2
next	4



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

Trace of Selection Sort Refinement (cont.)

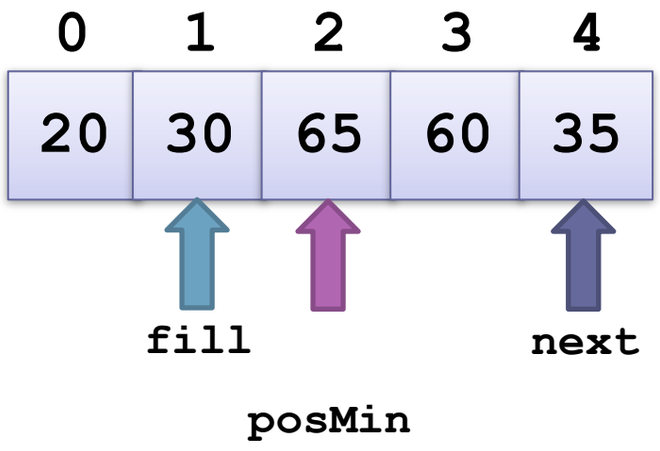
n	5
fill	1
posMin	2
next	4



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

Trace of Selection Sort Refinement (cont.)

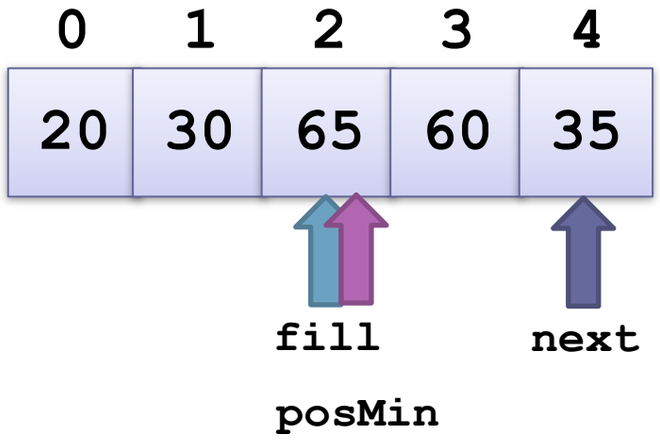
n	5
fill	1
posMin	2
next	4



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.)

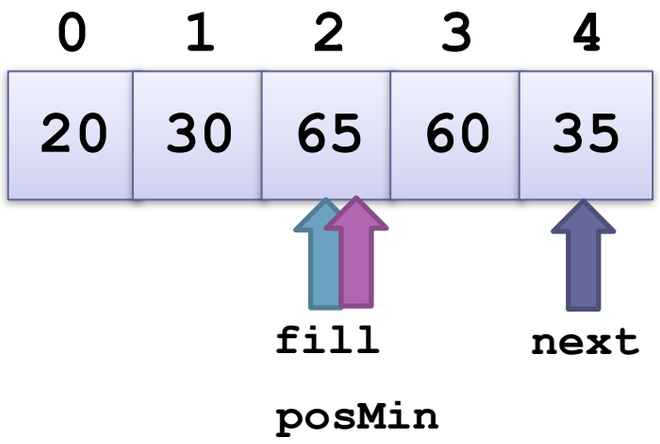
n	5
fill	2
posMin	2
next	4



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.)

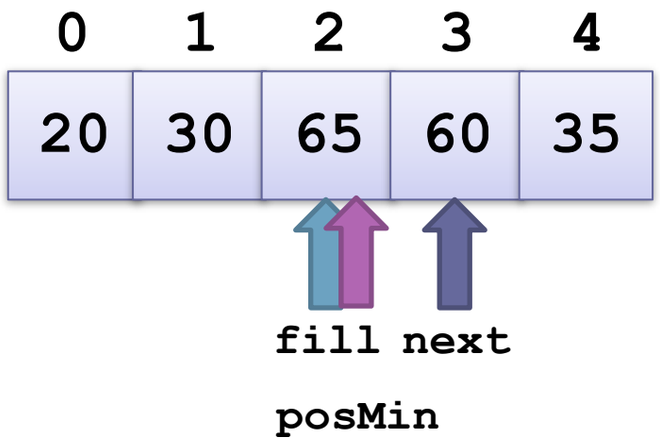
n	5
fill	2
posMin	2
next	4



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.)

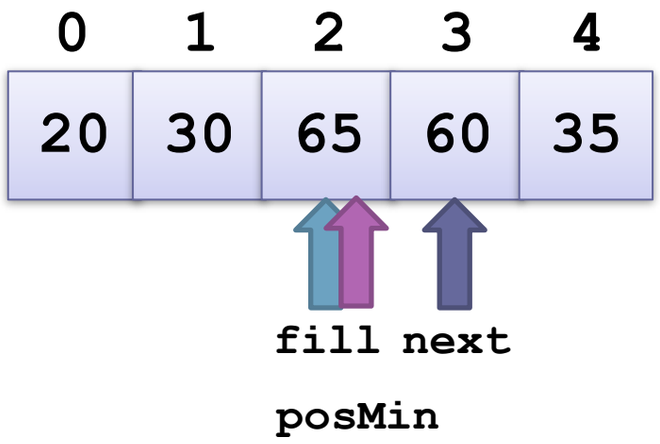
n	5
fill	2
posMin	2
next	3



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.)

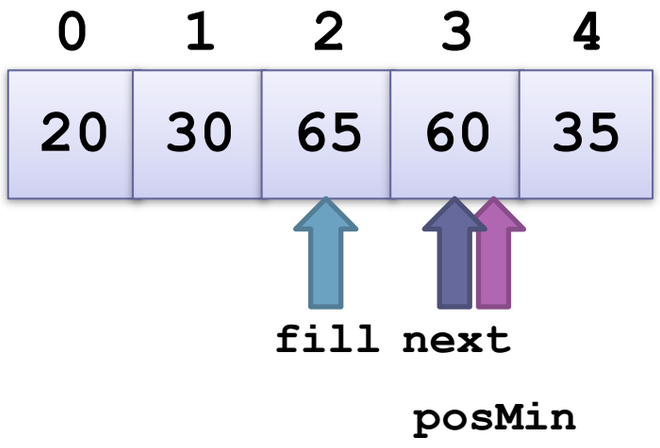
n	5
fill	2
posMin	2
next	3



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.)

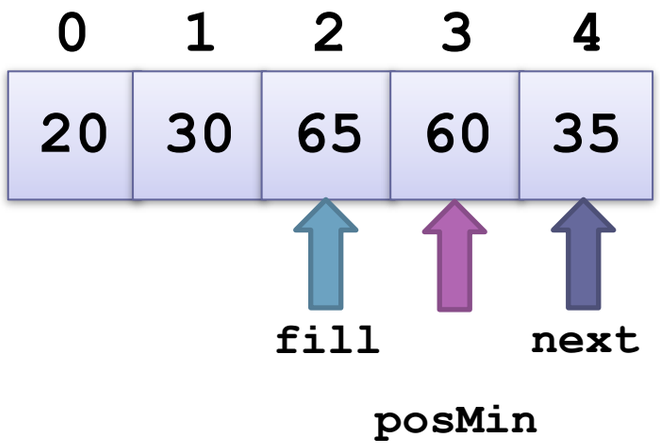
n	5
fill	2
posMin	3
next	3



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

Trace of Selection Sort Refinement (cont.)

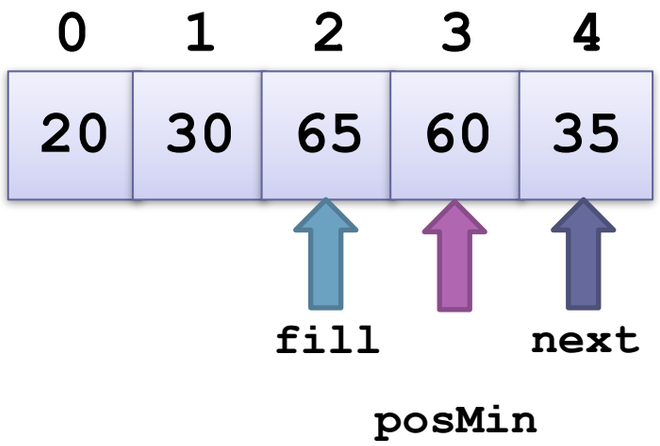
n	5
fill	2
posMin	3
next	4



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.)

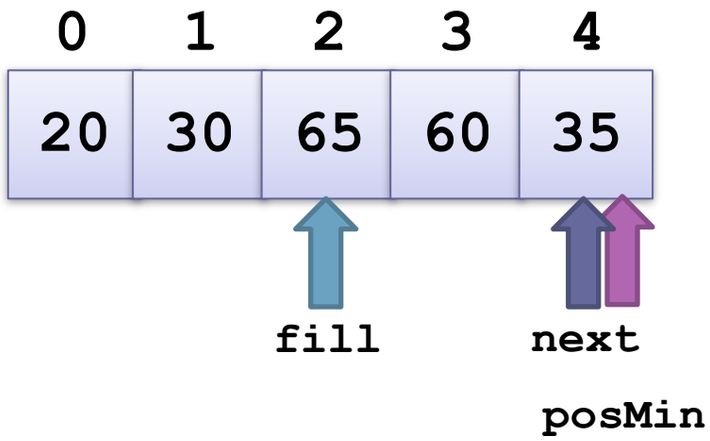
n	5
fill	2
posMin	3
next	4



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

Trace of Selection Sort Refinement (cont.)

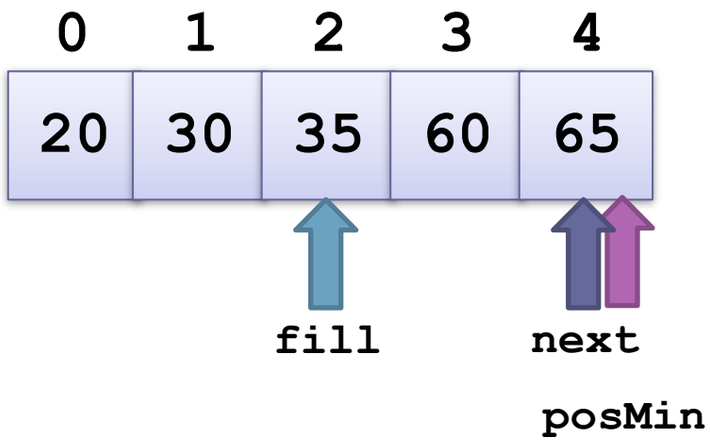
n	5
fill	2
posMin	4
next	4



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

Trace of Selection Sort Refinement (cont.)

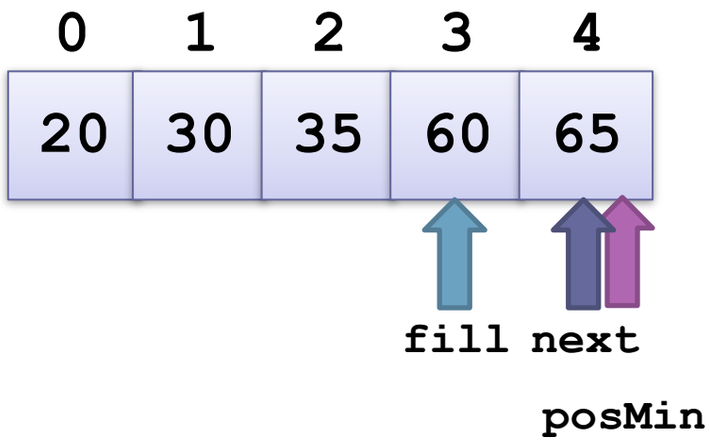
n	5
fill	2
posMin	4
next	4



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.)

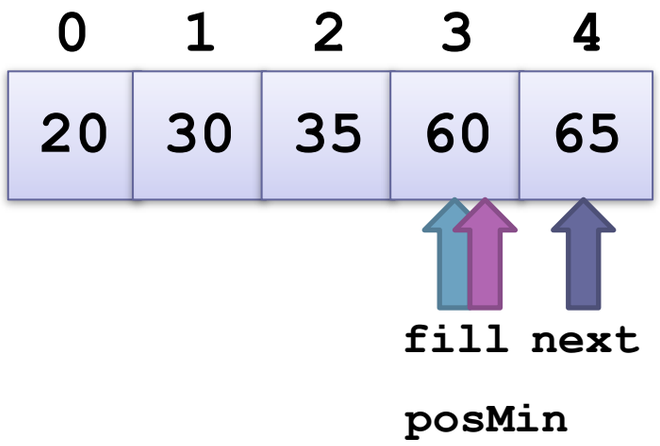
n	5
fill	3
posMin	4
next	4



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.)

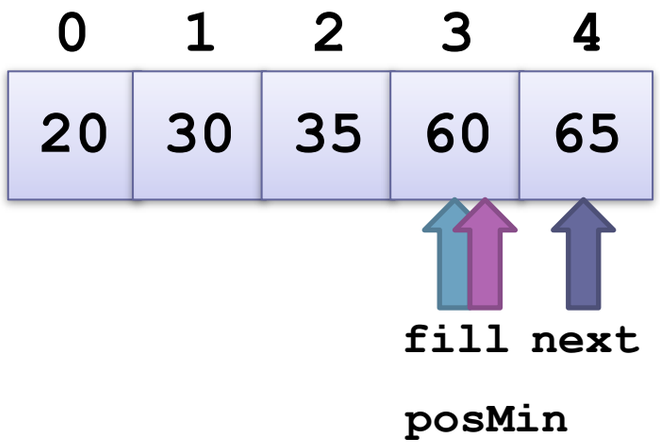
n	5
fill	3
posMin	3
next	4



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.)

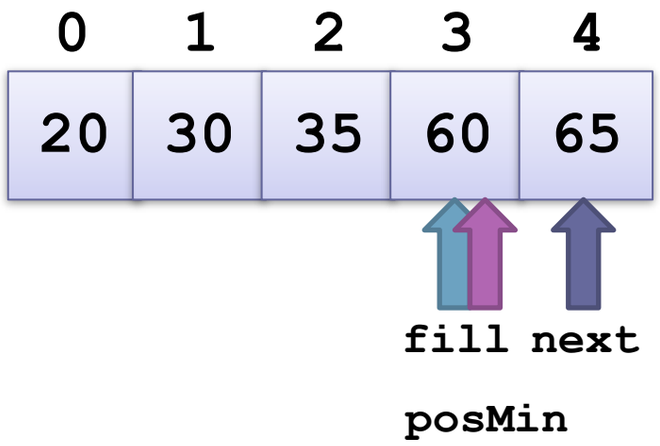
n	5
fill	3
posMin	3
next	4



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.)

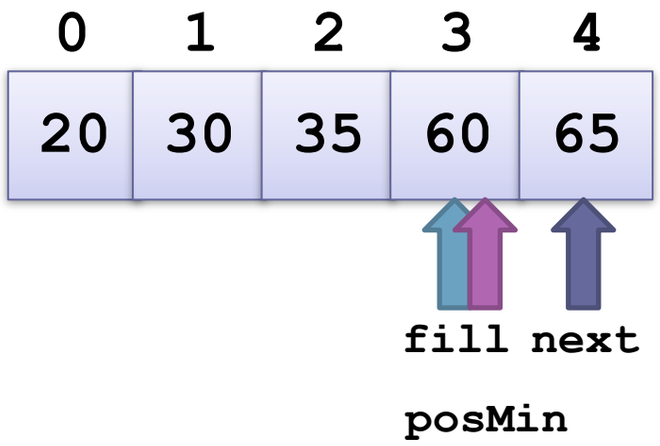
n	5
fill	3
posMin	3
next	4



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.)

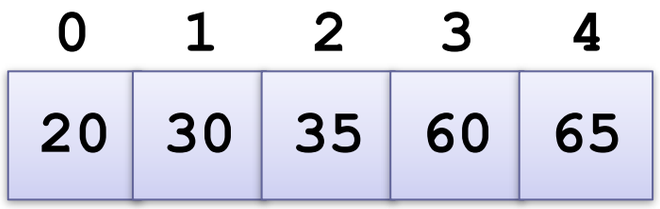
n	5
fill	3
posMin	3
next	4



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



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`

Komplexitetsanalys

Två nästade loopar som varje är O(n)

- komplexiteten är alltså kvadratisk, O(n^2)

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

Bubbelsortering

Kursboken, avsnitt 8.3

- Idén är att jämföra grannelement med varandra, och byta plats på dem ifall de är oordnade
- Mindre värden ”bubblar” upp till toppen av listan, och större värde ”sjunker” till botten
- Även denna algoritm har kvadratisk komplexitet

In-place-sortering

Alla dessa tre algoritmer är *in-place*:

- inget extra minne behövs
- det är originallistan som ändras

In-place är mycket svårare i Haskell eftersom man inte kan ändra värden:

- Haskells insättningssortering bygger alltså en ny sorterad lista
- den gamla listan försvinner i skräpsamlingen, men det sker inte direkt, så det krävs en del extraminne

Sorteringsalgoritmer

	Number of Comparisons		
	Best	Average	Worst
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Heapsort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quicksort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$

n^2 vs $n \log n$

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